ABSTRACT

HART, FRANK PATRICK. Frequency-Domain Behavioral Modeling of Nonlinear Systems Using The Arithmetic Operator Method. (Under the direction of Professor Michael B. Steer.)

A behavioral modeling environment for the rapid, high-dynamic range analysis of steady-state systems is presented. The Arithmetic Operator Method (AOM) is extended and cast in a solid mathematical framework. AOM is implemented as the AOM Toolbox within the Matlab® environment. The AOM Toolbox operates entirely in the frequency domain using behavioral modeling to permit large-signal nonlinearities and handle input signals consisting of one hundred or more distinct input tones with arbitrary spacing of the tones. Frequency domain analysis techniques have been applied previously to nonlinear circuits with multitone inputs, but in environments limited to a few tones processed by transfer functions with low orders of nonlinearity. Among the contributions brought forward by the development of the enhanced AOM method are the decomposition of the frequency domain into an underlying vector space that permits easy separation of correlated and uncorrelated spectral components after nonlinear processing, an algorithm for computing the underlying vector space decomposition of the frequency domain, algorithms for sparse matrix construction of convolution matrices, and the discovery of properties of the double-sided complex matrix and vector forms that facilitate the use of eigendecomposition as an alternative path to solutions. Validation of the AOM Toolbox is achieved by comparing the time-domain expansion of AOM Toolbox output to results produced purely in the time domain by two other means. Several applications the AOM Toolbox are given, including the prediction of the response of an amplifying device to correlated and uncorrelated multitone inputs, the response of an amplifying device to a broad-band linear FM chirp, and the response of a broadband television broadcast amplifier to a multicarrier input.
Frequency-Domain Behavioral Modeling of Nonlinear Systems
Using The Arithmetic Operator Method

by
Frank Patrick Hart

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Electrical Engineering

Raleigh, North Carolina

2009

APPROVED BY:

______________________________  ________________________________
Dr. K.G. Gard                  Dr. G. Lazzi

______________________________  ________________________________
Dr. M.B. Steer                      Dr. D.W. Barlage
Chair of Advisory Committee
DEDICATION

To my mother for the brains and to the memory of my father,
who had no earthly idea how he inspired me.
Frank P. Hart is presently working toward the Ph.D. degree at North Carolina State University, where he is a research assistant in the Electronics Research Laboratory. He received the Bachelor of Electrical Engineering degree, Cum Laude, from the University of Delaware in 1984 and the MSEE degree from North Carolina State University in 1994. From 1984 to 1995 he was employed by the IBM Corporation in Research Triangle Park, NC, where he worked on telecommunications networking and notebook personal computer hardware development. During his years at IBM he was promoted several times, rising to the level of Advisory Engineer before his departure. From 1995 through 2001 he was employed by the Intel Corporation, first in Hillsboro, Oregon and later in Santa Clara, California, where he worked on the development of “mobile module” subassemblies for notebook computers and mobile chipset architecture definition. He was a Senior Staff Engineer at the time of his departure from Intel. He has previously co-authored 4 papers and a book chapter, and he is named as a co-inventor on 16 patents.
ACKNOWLEDGMENTS

First and foremost I would like to thank my dissertation advisor, Dr. Michael B. Steer, for his advice throughout this work and his encouragement during its low points. I’d also like to thank the other members of my advisory committee, Drs. Doug Barlage, Gianluca Lazzi, and particularly, Kevin Gard for the many fruitful conversations that helped me to work through issues with the newer laboratory measuring equipment that he knows so well. Thanks also to Dr. Nuno Carvalho (with the Instituto de Telecomunicações, Universidad de Aveiro, Portugal) whose discussions on correlated and uncorrelated distortion during his sabbatical visit to NC State motivated me to nail down the correlation properties of spectral vectors based upon a vector description table; and to Dr. Yannis Viniotis, without whose remedially instructive hints in the discipline of random variables I’d have never finished that task. I’d also like to thank Dr. Jeffrey Thompson for serving as the Graduate School representative on my thesis advisory committee.

I would like to thank Ramya Mohan, Alan Victor, and Drs. Sonali R. Luniya, Nikhil M. Kriplani, Jayesh Nath, and Aaron Walker, all of whom collaborated with me in laboratory work and publications during my research assistantship. Special thanks go to Houssam Kanj, whose kind and patient assistance helped me to quickly learn a lot about fREEDA™’s internals as I was beginning my research work. Dr. Carlos Christoffersen, who knows fREEDA™ best, provided timely assistance from a distance. Dr. Wael Fathelbab furnished helpful advice in the realm of microwave filter design, fabrication, and simulation. Whenever we weren’t discussing our research, Dr. Khaled Gharaiheb and I had rip-roaring conversations on topics ranging from the Middle East to Donald Trump. (And Khaled: You’re Fired!) Dr. Steven Lipa provided a valuable introduction to the laboratory equipment and practices when I began my work there and – going above and beyond the call of duty – offered an ambulance service when one of those pieces of equipment struck me with a lightning bolt! I also benefited from conversations with Dr. Mark Buff, and the coffee service instigated by Aaron and Mark played no small part in my laboratory productivity during many late evenings. Dr. Wonhoon Jang and I shared laughs over our extended stays in the program. Dr. Steer thinks that Patsy Ashe, Claire Sideri, and Michelle Joyner worked for him, but they were all instrumental in rearranging his calendar to fit me in when necessary 😊. Patsy also did the heavy lifting in the procurement process for the components of the multitone oscillators, making it painless for me and tracking the timely
delivery of purchased items. Sang-Tae Bae and Dr. Meeta Yadav deserve special mention for having the intestinal fortitude to share a cubicle with me. Finally, I want to thank Jonathan Wilkerson, Greg Mazzaro, Glen Garner, Rob Harris, Justin Lowry, Vrinda P. Haridason, Jie Hu, and Chris Saunders for continuing the tradition of a stimulating doctoral research environment. The torch is now passed to you... good luck with your marathons.

I’d like to express my appreciation to the following organizations which supported the research conducted in this dissertation:

- The Space and Naval Warfare Systems Center San Diego, under grant number N6601-01-1-8921 as part of the DARPA NeoCAD Program
- The U.S. Army Research Laboratory and the U.S. Army Research Office on Multifunctional Adaptive Radio Radar and Sensors (MARRS) under grant number DAAD19-01-1-0496
- The U.S. Army Research Laboratory and the U.S. Army Research Office as a Multidisciplinary University Research Initiative on Standoff Inverse Analysis and Manipulation of Electronic Systems under grant number W911NF-05-1-0337
- Harris Corporation, for mounting Filtronics LMA-411 MMICs on carriers
# TABLE OF CONTENTS

**LIST OF TABLES** ................................................................. xiii

**LIST OF FIGURES** ............................................................ xiv

**LIST OF ABBREVIATIONS** .................................................. xix

**LIST OF SYMBOLS** ........................................................... xxi

1. Introduction ........................................................................ 1
   1.1 Why Nonlinear Analysis in the Frequency Domain? ............... 1
   1.2 Motivations and Objectives of This Study ............................ 4
   1.3 Dissertation Overview ................................................... 6
   1.4 Publications ............................................................... 9
      1.4.1 As Primary Author .................................................. 9
      1.4.2 Not As Primary Author ............................................. 10
      1.4.3 Planned Publications .............................................. 10

2. Literature Review ............................................................. 11
   2.1 Introductory Remarks ................................................... 11
   2.2 Notation ........................................................................ 11
   2.3 Time Domain Simulation ................................................ 12
   2.4 Mixed Time and Frequency Domain Simulation .................. 12
   2.5 Envelope Transient Simulation ....................................... 15
   2.6 Nonlinear Frequency Domain Simulation .......................... 15
      2.6.1 Simulation with Volterra Functionals .......................... 15
      2.6.2 Simulation with Power Series Models ......................... 17
      2.6.3 Simulation with the Arithmetic Operator Method .......... 20
      2.6.4 Frequency Counting Simulators ................................ 25
      2.6.5 Additional Developments ........................................... 26
   2.7 Concluding Remarks ..................................................... 26

3. The AOM Toolbox Behavioral Modeling Environment ................ 27
   3.1 Introductory Remarks ................................................... 27
   3.2 Mathematical Preliminaries ............................................. 29
      3.2.1 Euler's Formula .................................................... 30
      3.2.2 The Sine Over Argument Function .............................. 31
      3.2.3 The Dirac Delta Function ........................................ 34
      3.2.4 Fourier Transform Definition .................................... 36
      3.2.5 Fourier Transform of a DC Signal .............................. 36
      3.2.6 Fourier Transform of an Impulse ............................... 38
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.7</td>
<td>Fourier Transform of a Complex Exponential</td>
<td>39</td>
</tr>
<tr>
<td>3.2.8</td>
<td>Fourier Transform of Sinusoidal Functions</td>
<td>40</td>
</tr>
<tr>
<td>3.2.9</td>
<td>Fourier Transform of a Time-Domain Product of Functions</td>
<td>41</td>
</tr>
<tr>
<td>3.2.10</td>
<td>Fourier Transform of a Time-delayed Function</td>
<td>42</td>
</tr>
<tr>
<td>3.3</td>
<td>Illustrative Overview of the Arithmetic Operator Method</td>
<td>44</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Two-Tone Signals and Polynomial Nonlinear Transfer Function</td>
<td>44</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Vector Frequency Description for a Second Order Nonlinearity</td>
<td>47</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Introduction of Matrix Methods</td>
<td>50</td>
</tr>
<tr>
<td>3.3.4</td>
<td>Vector Frequency Description for a Third Order Nonlinearity</td>
<td>59</td>
</tr>
<tr>
<td>3.4</td>
<td>The Vector Frequency Description</td>
<td>64</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Input Signal</td>
<td>64</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Input Signal using the VFD</td>
<td>64</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Convolution of the Input using the VFD Form</td>
<td>65</td>
</tr>
<tr>
<td>3.4.4</td>
<td>Modified Vector Space Properties of the VFD Table</td>
<td>70</td>
</tr>
<tr>
<td>3.4.5</td>
<td>Construction of the VFD Table</td>
<td>71</td>
</tr>
<tr>
<td>3.4.5.1</td>
<td>Storage Allocation</td>
<td>71</td>
</tr>
<tr>
<td>3.4.5.2</td>
<td>Algorithm Description</td>
<td>73</td>
</tr>
<tr>
<td>3.4.5.3</td>
<td>Duplicate Candidate VFDs</td>
<td>76</td>
</tr>
<tr>
<td>3.4.5.4</td>
<td>Final Form of VFD Table</td>
<td>76</td>
</tr>
<tr>
<td>3.4.5.5</td>
<td>VFD Table Construction Algorithm</td>
<td>77</td>
</tr>
<tr>
<td>3.4.5.6</td>
<td>Matlab Implementation Details</td>
<td>80</td>
</tr>
<tr>
<td>3.4.6</td>
<td>3-tone VFD Construction Example</td>
<td>80</td>
</tr>
<tr>
<td>3.4.6.1</td>
<td>Linear Response</td>
<td>81</td>
</tr>
<tr>
<td>3.4.6.2</td>
<td>Second Order Response</td>
<td>81</td>
</tr>
<tr>
<td>3.4.6.3</td>
<td>Third Order Response</td>
<td>83</td>
</tr>
<tr>
<td>3.4.6.4</td>
<td>3-tone VFD in Final Form</td>
<td>89</td>
</tr>
<tr>
<td>3.5</td>
<td>The Spectral Vector</td>
<td>99</td>
</tr>
<tr>
<td>3.5.1</td>
<td>Complex Spectral Vector</td>
<td>99</td>
</tr>
<tr>
<td>3.5.2</td>
<td>Double-Sided Real Spectral Vector</td>
<td>101</td>
</tr>
<tr>
<td>3.5.3</td>
<td>Single-Sided Real Spectral Vector</td>
<td>102</td>
</tr>
<tr>
<td>3.5.4</td>
<td>Spectral Vector Construction Algorithm</td>
<td>102</td>
</tr>
<tr>
<td>3.5.5</td>
<td>Correlation Properties of Spectral Vectors</td>
<td>104</td>
</tr>
<tr>
<td>3.5.5.1</td>
<td>Uncorrelated Spectral Content</td>
<td>107</td>
</tr>
<tr>
<td>3.5.5.2</td>
<td>Correlated Spectral Content</td>
<td>110</td>
</tr>
<tr>
<td>3.5.5.3</td>
<td>Observations on Correlated and Uncorrelated IM Distortion</td>
<td>111</td>
</tr>
<tr>
<td>3.5.5.4</td>
<td>Spectral Correlation Theorems and Remarks</td>
<td>113</td>
</tr>
<tr>
<td>3.6</td>
<td>Spectrum Mapping Table</td>
<td>118</td>
</tr>
<tr>
<td>3.6.1</td>
<td>Convolution Steps and the 1-Norm Sum Test</td>
<td>119</td>
</tr>
<tr>
<td>3.6.2</td>
<td>Determining The Frequency Index of the Output VFD</td>
<td>121</td>
</tr>
<tr>
<td>3.6.3</td>
<td>Spectrum Mapping Table Contents</td>
<td>123</td>
</tr>
<tr>
<td>3.6.4</td>
<td>Spectrum Mapping Table Construction Algorithm</td>
<td>123</td>
</tr>
<tr>
<td>3.6.5</td>
<td>Matlab Implementation Details</td>
<td>126</td>
</tr>
<tr>
<td>3.7</td>
<td>Spectrum Transform Matrix</td>
<td>127</td>
</tr>
</tbody>
</table>
3.7.1 Spectrum Transform Matrix Construction Algorithm ........................................ 127
3.7.2 Properties of the Spectrum Transform Matrix .............................................. 131
3.7.3 Further Information on Eigendecomposition ..................................................... 136
3.8 Dynamic Range of Two-Tone Tests ................................................................. 138
3.9 Elements of FREDA2 Revisited ........................................................................ 140
3.9.1 Background .................................................................................................. 140
3.9.2 One-Sided Spectrum Mapping Table .............................................................. 144
3.9.3 1-Sided Spectrum Mapping Table Construction Algorithm .............................. 144
3.9.4 One-Sided Spectrum Transform Matrix .......................................................... 145
3.9.5 1-Sided Spectrum Transform Matrix Construction Algorithm ....................... 147
3.9.6 The Exponential Function ............................................................................. 149
3.9.7 The Hyperbolic Tangent Function .................................................................. 153
3.9.8 Summary ........................................................................................................ 155
3.10 Concluding Remarks ......................................................................................... 155
4 Validation of The AOM Toolbox With a Logarithmic Amplifier Model ................... 157
4.1 Introductory Remarks ......................................................................................... 157
4.2 Logarithmic Amplifier ....................................................................................... 158
4.2.1 Circuit Analysis ............................................................................................. 158
4.2.2 AOM Behavioral Model and Input Signal Form ............................................. 159
4.2.3 Time-Domain Input Signal .......................................................................... 163
4.2.4 Selection of Order of Spectral Truncation .................................................... 163
4.3 Results ............................................................................................................... 166
4.4 Effect of Spectral Truncation ............................................................................ 170
4.4.1 Spectral Truncation to the Third Order ....................................................... 170
4.4.2 Spectral Truncation to the Second Order .................................................... 173
4.4.3 Further Discussion on Spectral Truncation ................................................... 175
4.5 Concluding Remarks ......................................................................................... 177
5 Modeling the Nonlinear Response to Multitones with Uncorrelated Phase ............. 179
5.1 Introductory Remarks ......................................................................................... 179
5.2 Measurement Apparatus .................................................................................... 180
5.2.1 Uncorrelated Phase Multitone Signal Generator .......................................... 180
5.2.2 Laboratory Equipment Setup ....................................................................... 181
5.2.3 Signal Chain Characterization for Simulation ............................................. 183
5.3 Results and Discussion ..................................................................................... 185
5.3.1 Adjacent-Band IM Analysis Example ......................................................... 188
5.3.2 Narrow-Band IM Analysis Example .............................................................. 200
5.3.3 Phase Invariance of Simulated Average Power Computations ....................... 202
5.3.4 Separating Correlated and Uncorrelated Distortion ...................................... 207
5.3.5 Computational Considerations ..................................................................... 211
5.4 Concluding Remarks ......................................................................................... 212
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6. Model the Response of a Nonlinear Amplifier to a Linear FM Chirp</td>
<td>214</td>
</tr>
<tr>
<td>6.1 Introductory Remarks</td>
<td>214</td>
</tr>
<tr>
<td>6.2 Mathematical Analysis</td>
<td>215</td>
</tr>
<tr>
<td>6.2.1 Fourier Analysis of a Linear FM Chirp Signal</td>
<td>215</td>
</tr>
<tr>
<td>6.2.2 Discrete Fourier Transform Scale Factors</td>
<td>220</td>
</tr>
<tr>
<td>6.3 AOM and FREEEDA Modeling Environment Setup</td>
<td>220</td>
</tr>
<tr>
<td>6.3.1 FREEEDA Circuit Models</td>
<td>221</td>
</tr>
<tr>
<td>6.3.2 AOM Behavioral Models</td>
<td>221</td>
</tr>
<tr>
<td>6.3.3 Butterworth Bandpass Filter Model</td>
<td>222</td>
</tr>
<tr>
<td>6.4 Results and Discussion</td>
<td>224</td>
</tr>
<tr>
<td>6.4.1 500 MHz Chirp Signal and Response</td>
<td>224</td>
</tr>
<tr>
<td>6.4.1.1 MMIC Amplifier Response to a 500 MHz Chirp</td>
<td>226</td>
</tr>
<tr>
<td>6.4.1.2 Bandpass Filtering the 500 MHz Chirp</td>
<td>231</td>
</tr>
<tr>
<td>6.4.2 1 GHz Chirp Signal and Response</td>
<td>233</td>
</tr>
<tr>
<td>6.4.2.1 MMIC Amplifier Response to a 1 GHz Chirp</td>
<td>233</td>
</tr>
<tr>
<td>6.4.2.2 Bandpass Filtering the 1 GHz Chirp</td>
<td>240</td>
</tr>
<tr>
<td>6.4.3 Computational Considerations</td>
<td>243</td>
</tr>
<tr>
<td>6.5 Concluding Remarks</td>
<td>243</td>
</tr>
<tr>
<td>7. Model the Multicarrier Response of a CATV Trunk Amplifier</td>
<td>245</td>
</tr>
<tr>
<td>7.1 Introductory Remarks</td>
<td>245</td>
</tr>
<tr>
<td>7.2 The CATV Frequency Plan</td>
<td>246</td>
</tr>
<tr>
<td>7.3 Forms of Distortion Considered</td>
<td>247</td>
</tr>
<tr>
<td>7.4 Forms of Distortion Not Considered</td>
<td>247</td>
</tr>
<tr>
<td>7.5 Nonlinear Amplifier Model</td>
<td>248</td>
</tr>
<tr>
<td>7.5.1 Input Signal</td>
<td>249</td>
</tr>
<tr>
<td>7.5.2 Gain Parameters</td>
<td>249</td>
</tr>
<tr>
<td>7.5.3 Filter Parameters</td>
<td>252</td>
</tr>
<tr>
<td>7.6 Results</td>
<td>253</td>
</tr>
<tr>
<td>7.6.1 79 Channel Plan CSO/CTB Results</td>
<td>253</td>
</tr>
<tr>
<td>7.6.1.1 79 Channel CSO Results</td>
<td>254</td>
</tr>
<tr>
<td>7.6.1.2 79 Channel CTB Results</td>
<td>255</td>
</tr>
<tr>
<td>7.6.2 158 Channel Plan CSO/CTB Results</td>
<td>258</td>
</tr>
<tr>
<td>7.6.2.1 158 Channel CSO Results</td>
<td>258</td>
</tr>
<tr>
<td>7.6.2.2 158 Channel CTB Results</td>
<td>260</td>
</tr>
<tr>
<td>7.6.3 158 Channel Plan Filter Results</td>
<td>260</td>
</tr>
<tr>
<td>7.6.4 Computational Considerations</td>
<td>261</td>
</tr>
<tr>
<td>7.7 Concluding Remarks</td>
<td>264</td>
</tr>
<tr>
<td>8. Conclusion</td>
<td>266</td>
</tr>
<tr>
<td>8.1 Summary</td>
<td>266</td>
</tr>
<tr>
<td>8.2 Recommendations for Further Study</td>
<td>270</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>274</td>
</tr>
</tbody>
</table>
## LIST OF TABLES

<p>| Table 2.1 | Mixer BIPD Table built using the method of Rhyno | 19 |
| Table 2.2 | Output Index Vector generated by method of Carvalho &amp; Pedro | 25 |
| Table 3.1 | 1-Norm Sort VFD Table for 2 Tones in a 2nd Order Nonlinearity | 48 |
| Table 3.2 | 1-Norm Sort VFD Table for 2 Tones in a 3rd Order Nonlinearity | 59 |
| Table 3.3 | 1-Norm Sort 1-Side VFD Table for 3 Tones in a 3rd Order Nonlinearity | 90 |
| Table 3.4 | 1-Norm Sort 2-Side VFD Table for 3 Tones in a 3rd Order Nonlinearity | 91 |
| Table 3.5 | Frequency Sort 1-Side VFD Table for 3 Tones in a 3rd Order Nonlinearity | 94 |
| Table 3.6 | Frequency Sort 2-Side VFD Table for 3 Tones in a 3rd Order Nonlinearity | 95 |
| Table 3.7 | 1-Sided Version of Table 3.1 on page 48 | 142 |
| Table 4.1 | Log Amplifier VFD Table for 2 Tones in a 5th Order Nonlinearity | 164 |
| Table 4.2 | Log Amplifier VFD Table for 2 Tones in a 3rd Order Nonlinearity | 171 |
| Table 4.3 | Log Amplifier VFD Table for 2 Tones in a 2nd Order Nonlinearity | 173 |
| Table 5.1 | Numbers of phasors averaged at adjacent-band IM frequencies | 189 |
| Table 5.2 | Measured frequencies of carrier oscillation for narrow-band IM analysis | 200 |
| Table 5.3 | Numbers of phasors averaged at in-band IM frequencies | 202 |
| Table 5.4 | Costs of computation for uncorrelated multitone | 211 |
| Table 6.1 | Amplifier Coefficients | 222 |
| Table 7.1 | Gain Parameters | 250 |
| Table 7.2 | Filter Block Parameters | 252 |
| Table 7.3 | Costs of computation for CATV amplification | 264 |
| Table 8.1 | Comparison of AOM to other methods | 268 |
| Table A.1 | z-domain Butterworth lowpass filter block coefficient definitions | 305 |
| Table A.2 | Transfer function denominator coefficients | 313 |
| Table A.3 | Bandpass filter block coefficient definitions | 315 |
| Table B.1 | Bill of Materials for 3-oscillator Assembly in Figure B.2 | 326 |
| Table C.1 | BIPD Table for spectrum of Figure C.6 | 389 |
| Table C.2 | Spectrum Mapping Table entry | 390 |
| Table C.3 | Portions of the Spectrum Mapping Table | 391 |
| Table C.4 | BIPD Table for 3rd Order Non-Linear Transfer Function | 408 |
| Table D.1 | Mixer A VFD Table for 3 Tones in a 2nd Order Nonlinearity | 418 |
| Table D.2 | Mixer B VFD Table for 3 Tones in a 2nd Order Nonlinearity | 424 |</p>
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Circuit partitioning for Harmonic Balance</td>
<td>13</td>
</tr>
<tr>
<td>2.2</td>
<td>Frequencies of interest in a mixer</td>
<td>18</td>
</tr>
<tr>
<td>3.1</td>
<td>Complex contour integral of $e^{j\alpha z}/z$</td>
<td>32</td>
</tr>
<tr>
<td>3.2</td>
<td>$I = 1$, the 1st step of spectral vector discrete convolution</td>
<td>52</td>
</tr>
<tr>
<td>3.3</td>
<td>$I = 2$, the 2nd step of spectral vector discrete convolution</td>
<td>52</td>
</tr>
<tr>
<td>3.4</td>
<td>$I = 6$, the 6th step of spectral vector discrete convolution</td>
<td>53</td>
</tr>
<tr>
<td>3.5</td>
<td>$I = 7$, the 7th step of spectral vector discrete convolution</td>
<td>53</td>
</tr>
<tr>
<td>3.6</td>
<td>$I = 13$, the 13th step of spectral vector discrete convolution</td>
<td>54</td>
</tr>
<tr>
<td>3.7</td>
<td>$I = 14$, the 14th step of spectral vector discrete convolution</td>
<td>55</td>
</tr>
<tr>
<td>3.8</td>
<td>$I = 17$, the 17th step of spectral vector discrete convolution</td>
<td>56</td>
</tr>
<tr>
<td>3.9</td>
<td>$I = 25$, the 25th step of spectral vector discrete convolution</td>
<td>56</td>
</tr>
<tr>
<td>3.10</td>
<td>Spectrum Transform Matrix constructed from the 25 convolution steps</td>
<td>57</td>
</tr>
<tr>
<td>3.11</td>
<td>Spectrum Transform Matrix created from (3.135)</td>
<td>63</td>
</tr>
<tr>
<td>3.12</td>
<td>Number of 1-sided VFD Table entries for given $Q, N$</td>
<td>72</td>
</tr>
<tr>
<td>3.13</td>
<td>Storage required for minimal data structures for a given problem</td>
<td>72</td>
</tr>
<tr>
<td>3.14</td>
<td>Upper circulant shift permutation matrix $P$</td>
<td>74</td>
</tr>
<tr>
<td>3.15</td>
<td>Upper circulant shift permutation matrix $P^2$</td>
<td>75</td>
</tr>
<tr>
<td>3.16</td>
<td>Row-flipping permutation matrix $J$</td>
<td>77</td>
</tr>
<tr>
<td>3.17</td>
<td>Quadrature modulation of two real input signals</td>
<td>100</td>
</tr>
<tr>
<td>3.18</td>
<td>$I \approx K - O$, 1st atomic multiplication</td>
<td>119</td>
</tr>
<tr>
<td>3.19</td>
<td>$I \approx 2K - L - O$, 1st valid atomic product</td>
<td>120</td>
</tr>
<tr>
<td>3.20</td>
<td>$I \approx 2K - O$, step with valid atomic multiplication products</td>
<td>120</td>
</tr>
<tr>
<td>3.21</td>
<td>$I \approx 2K + O$, step with valid atomic multiplication products</td>
<td>121</td>
</tr>
<tr>
<td>3.22</td>
<td>$I \approx 4K - O$, final convolution step with valid products</td>
<td>121</td>
</tr>
<tr>
<td>3.23</td>
<td>$I \approx 4K - 1$, the final convolution step – with invalid products</td>
<td>122</td>
</tr>
<tr>
<td>3.24</td>
<td>2-Sided Spectrum Transform Matrix Stamps</td>
<td>128</td>
</tr>
<tr>
<td>3.25</td>
<td>Spectrum Transform Matrix from the example in Section 3.3.3</td>
<td>131</td>
</tr>
<tr>
<td>3.26</td>
<td>Spectrum Transform Matrix for a conjugate symmetric two-tone signal</td>
<td>132</td>
</tr>
<tr>
<td>3.27</td>
<td>$\tanh(x)$ and Cann models for $s = 2, 3, 4$</td>
<td>135</td>
</tr>
<tr>
<td>3.28</td>
<td>Two tone dynamic range of the AOM Toolbox vs. the FFT</td>
<td>138</td>
</tr>
<tr>
<td>3.29</td>
<td>Relative Error in the approximation of $\exp(x)$ for $n = 6, 7, 8$</td>
<td>149</td>
</tr>
<tr>
<td>3.30</td>
<td>Relative errors in function approximations</td>
<td>154</td>
</tr>
<tr>
<td>4.1</td>
<td>Circuit diagram of an ideal logarithmic amplifier</td>
<td>158</td>
</tr>
<tr>
<td>4.2</td>
<td>Results of simulating the logarithmic amplifier</td>
<td>167</td>
</tr>
<tr>
<td>4.3</td>
<td>Relative error between AOM Toolbox and Matlab® time-domain results</td>
<td>168</td>
</tr>
<tr>
<td>4.4</td>
<td>Relative error between AOM Toolbox and PSpice simulation results</td>
<td>169</td>
</tr>
<tr>
<td>4.5</td>
<td>Relative error between Matlab® reference and PSpice simulation results</td>
<td>169</td>
</tr>
</tbody>
</table>
Figure 4.6 Results of spectral truncation at the 3rd order
Figure 4.7 Relative error for 3rd order spectral truncation
Figure 4.8 Results of spectral truncation at the 2nd order
Figure 4.9 Relative error for 2nd order spectral truncation
Figure 4.10 Iteration tolerance for at various orders of spectral truncation
Figure 4.11 Actual error at various orders of spectral truncation

Figure 5.1 Simplified block diagram of a three oscillator assembly
Figure 5.2 Block diagram of the multitone system of 5 three-tone assemblies
Figure 5.3 Block diagram of the laboratory setup
Figure 5.4 Laboratory photo of measurements in progress
Figure 5.5 Block diagram of the uncorrelated phase signal measurement chain
Figure 5.6 Characterized average output power from 15 oscillators before combining
Figure 5.7 Characterized average output power from 15 oscillators after combining
Figure 5.8 Simulated and measured results for adjacent-band IM
Figure 5.9 Detail view of the linear response
Figure 5.10 Detail view of the lower adjacent-band response
Figure 5.11 Results of enforced correlation simulations 1-10
Figure 5.12 Results of enforced correlation simulations 11-20
Figure 5.13 Results of enforced correlation simulations 21-30
Figure 5.14 Results of averaging 8 enforced correlation simulations
Figure 5.15 Average of Enforced Correlation Phase Regimes 1–16
Figure 5.16 Average of Enforced Correlation Phase Regimes 1–30
Figure 5.17 Linear response detail of the average
Figure 5.18 Left adjacent IM band response detail of the average
Figure 5.19 Relative error between ensemble and a single simulation
Figure 5.20 Simulated and measured results for narrowband IM
Figure 5.21 Average PSD results from 30 simulations with 1-sided vectors
Figure 5.22 Absolute difference between min and max powers for 1-sided simulations
Figure 5.23 Average PSD results from 30 simulations with 2-sided vectors
Figure 5.24 Absolute difference between min and max powers for 2-sided simulations
Figure 5.25 Adjacent and co-channel band view for measured source amplitude
Figure 5.26 Magnification of the co-channel band view for measured source amplitude
Figure 5.27 Adjacent and co-channel band view for average source amplitude
Figure 5.28 Magnification of the co-channel band view for average source amplitude

Figure 6.1 Block diagram of the simulated radar transmission chain
Figure 6.2 Attenuation (in dB) of the bandpass filters
Figure 6.3 Phase change (in degrees) introduced by the bandpass filters
Figure 6.4 Magnitude of the 500 MHz linear FM chirp input
Figure 6.5 Magnitude detail of the peak of the 500 MHz LFM chirp input
Figure 6.6 Phase of the 500 MHz linear FM chirp input
Figure 6.7 Magnitude of the MMIC output W(f) in the chirp range
Figure 6.8 Magnitude detail of the MMIC chirp peak output of W(f)
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.9</td>
<td>Magnitude of the whole-spectrum MMIC output $W(f)$</td>
</tr>
<tr>
<td>6.10</td>
<td>Detail view of the 7th harmonic of the fREEDA\textsuperscript{-}MMIC output $W(f)$</td>
</tr>
<tr>
<td>6.11</td>
<td>Detail view of the 3rd harmonic output $W(f)$</td>
</tr>
<tr>
<td>6.12</td>
<td>Magnitude of the adjacent band outputs</td>
</tr>
<tr>
<td>6.13</td>
<td>Magnitude of the 3rd harmonic outputs</td>
</tr>
<tr>
<td>6.14</td>
<td>Magnitude of the whole-spectrum fREEDA\textsuperscript{-} output $Y(f)$</td>
</tr>
<tr>
<td>6.15</td>
<td>Magnitude of the 1 GHz linear FM chirp input</td>
</tr>
<tr>
<td>6.16</td>
<td>Magnitude detail of the peak of the 1 GHz LFM chirp input</td>
</tr>
<tr>
<td>6.17</td>
<td>Phase of the 1 GHz linear FM chirp input</td>
</tr>
<tr>
<td>6.18</td>
<td>Magnitude of the MMIC output $W(f)$ in the chirp range</td>
</tr>
<tr>
<td>6.19</td>
<td>Magnitude detail of the MMIC chirp peak output of $W(f)$</td>
</tr>
<tr>
<td>6.20</td>
<td>Magnitude of the whole-spectrum MMIC output $W(f)$</td>
</tr>
<tr>
<td>6.21</td>
<td>Detail view of the 7th harmonic of the fREEDA\textsuperscript{-}MMIC output $W(f)$</td>
</tr>
<tr>
<td>6.22</td>
<td>Detail view of the 3rd harmonic output $W(f)$</td>
</tr>
<tr>
<td>6.23</td>
<td>Magnitude of the adjacent band outputs</td>
</tr>
<tr>
<td>6.24</td>
<td>Magnitude of the 3rd harmonic outputs</td>
</tr>
<tr>
<td>6.25</td>
<td>Magnitude of the whole-spectrum fREEDA\textsuperscript{-} output $Y(f)$</td>
</tr>
<tr>
<td>7.1</td>
<td>Behavioral model for the CATV amplifier</td>
</tr>
<tr>
<td>7.2</td>
<td>Nonlinearity Distortion Parameters used to derive $G_2$ and $G_3$</td>
</tr>
<tr>
<td>7.3</td>
<td>Nonlinearity Distortion Parameter $D_2(f)$</td>
</tr>
<tr>
<td>7.4</td>
<td>Nonlinearity Distortion Parameter $D_3(f)$</td>
</tr>
<tr>
<td>7.5</td>
<td>Magnitude Response of $K_2(f), K_3(f)$, and $K_3(f)$</td>
</tr>
<tr>
<td>7.6</td>
<td>Phase response of filters $K_2(f), K_3(f)$, and $K_3(f)$</td>
</tr>
<tr>
<td>7.7</td>
<td>CSO power spectral density for a 79 channel plan</td>
</tr>
<tr>
<td>7.8</td>
<td>$\pm1.25$ MHz CSO beats for a 79 channel plan</td>
</tr>
<tr>
<td>7.9</td>
<td>$\pm0.75$ MHz CSO beats for a 79 channel plan</td>
</tr>
<tr>
<td>7.10</td>
<td>CTB power spectral density for a 79 channel plan</td>
</tr>
<tr>
<td>7.11</td>
<td>CTB beats for a 79 channel plan</td>
</tr>
<tr>
<td>7.12</td>
<td>CSO power spectral density for a 158 channel plan</td>
</tr>
<tr>
<td>7.13</td>
<td>$\pm1.25$ MHz CSO beats for a 158 channel plan</td>
</tr>
<tr>
<td>7.14</td>
<td>$\pm0.75$ MHz CSO beats for a 158 channel plan</td>
</tr>
<tr>
<td>7.15</td>
<td>CTB power spectral density for a 158 channel plan</td>
</tr>
<tr>
<td>7.16</td>
<td>CTB beats for a 158 channel plan</td>
</tr>
<tr>
<td>7.17</td>
<td>Effect of filtering the linear output for a 158 channel plan</td>
</tr>
<tr>
<td>7.18</td>
<td>Effect of filtering the 2nd order harmonic for a 158 channel plan</td>
</tr>
<tr>
<td>7.19</td>
<td>Effect of filtering the 3rd order harmonic for a 158 channel plan</td>
</tr>
<tr>
<td>A.1</td>
<td>Filter synthesis limitations of a leading commercial simulator</td>
</tr>
<tr>
<td>A.2</td>
<td>Two-stage ladder circuit with ill-conditioned MNAM</td>
</tr>
<tr>
<td>A.3</td>
<td>Condition number of MNAM for the two-stage ladder circuit</td>
</tr>
<tr>
<td>A.4</td>
<td>Typical parameters for a Butterworth lowpass filter</td>
</tr>
<tr>
<td>A.5</td>
<td>Discrete-time lowpass Butterworth filter in cascade form</td>
</tr>
<tr>
<td>A.6</td>
<td>Typical discrete-time filter block</td>
</tr>
</tbody>
</table>
Figure A.7  Discrete-time filter block permitting Newton iteration ................................................. 307
Figure A.8  Discrete-time filter block at initial iterate of time step .................................................. 308
Figure A.9  Typical parameters for a Chebychev Type I lowpass filter .......................................... 309
Figure A.10  Typical parameters for a bandpass filter ................................................................. 312
Figure A.11  Discrete-time bandpass filter in cascade form .......................................................... 314
Figure A.12  Typical discrete-time filter block in canonical form .................................................. 317
Figure A.13  Frequency response results for a 5-section coaxial filter ........................................... 318
Figure A.14  Response of simulated filter to a Linear FM chirp signal ......................................... 319
Figure A.15  Block diagram of the laboratory measurements setup ................................................. 320
Figure A.16  Layout of the microstrip patterns in the fabricated filter ........................................... 321
Figure A.17  Measured and simulated results near the lower edge of the passband .......................... 322

Figure B.1  Schematic of circuitry supporting one VCO ................................................................. 329
Figure B.2  Block Diagram of an assembly of three oscillators ..................................................... 330
Figure B.3  Photo of a 3 oscillator assembly .................................................................................. 331
Figure B.4  Results of two injection pulling experiments ............................................................... 334
Figure B.5  Histogram of oscillator frequency locations for Assembly 1 ........................................ 336
Figure B.6  Amplitude and Frequency Running Standard Deviation for Assembly 1 ...................... 337
Figure B.7  Instantaneous and Running Average Amplitude for Assembly 1 ............................... 338
Figure B.8  Instantaneous and Running Average Frequency for Assembly 1 ............................... 339
Figure B.9  Histogram of oscillator frequency locations for Assembly 2 ........................................ 341
Figure B.10  Amplitude and Frequency Running Standard Deviation for Assembly 2 .................... 342
Figure B.11  Instantaneous and Running Average Amplitude for Assembly 2 ............................. 343
Figure B.12  Instantaneous and Running Average Frequency for Assembly 2 ............................. 344
Figure B.13  Histogram of oscillator frequency locations for Assembly 3 ........................................ 346
Figure B.14  Amplitude and Frequency Running Standard Deviation for Assembly 3 .................... 347
Figure B.15  Instantaneous and Running Average Amplitude for Assembly 3 ............................. 348
Figure B.16  Instantaneous and Running Average Frequency for Assembly 3 ............................. 349
Figure B.17  Histogram of oscillator frequency locations for Assembly 4 ........................................ 351
Figure B.18  Amplitude and Frequency Running Standard Deviation for Assembly 4 .................... 352
Figure B.19  Instantaneous and Running Average Amplitude for Assembly 4 ............................. 353
Figure B.20  Instantaneous and Running Average Frequency for Assembly 4 ............................. 354
Figure B.21  Histogram of oscillator frequency locations for Assembly 5 ........................................ 356
Figure B.22  Amplitude and Frequency Running Standard Deviation for Assembly 5 .................... 357
Figure B.23  Instantaneous and Running Average Amplitude for Assembly 5 ............................. 358
Figure B.24  Instantaneous and Running Average Frequency for Assembly 5 ............................. 359
Figure B.25  Histogram of oscillator frequency locations for Assembly 6 ........................................ 361
Figure B.26  Amplitude and Frequency Running Standard Deviation for Assembly 6 .................... 362
Figure B.27  Instantaneous and Running Average Amplitude for Assembly 6 ............................. 363
Figure B.28  Instantaneous and Running Average Frequency for Assembly 6 ............................. 364
Figure B.29  Histogram of oscillator frequency locations for Assembly 7 ........................................ 366
Figure B.30  Amplitude and Frequency Running Standard Deviation for Assembly 7 .................... 367
Figure B.31  Instantaneous and Running Average Amplitude for Assembly 7 ............................. 368
Figure C.1  An ideal multiplier with \( y(t) = x(t)z(t) \). ........................................ 373
Figure C.2  Spectral representation of a three-tone signal ........................................ 375
Figure C.3  The approximate spectrum resulting from the excitation by \( f_1 \) and \( f_2 \) ... 388
Figure C.4  Generic Double-Sided spectrum with frequencies identified. ..................... 405

Figure D.1  Typical analysis flow for solving a problem with the AOM Toolbox .......... 413
Figure D.2  Microwave mixer schematic model ......................................................... 415
Figure D.3  Microwave mixer analysis model .............................................................. 415
Figure D.4  IF output from Mixer A ......................................................................... 420
Figure D.5  RF (linear) output from Mixer A ............................................................... 420
Figure D.6  2\textsuperscript{nd} order harmonic and IM output from Mixer A .................. 421
Figure D.7  IF output from Mixer A ............................................................................ 422
Figure D.8  IF output from Mixer B ............................................................................ 425
Figure D.9  RF (linear) output from Mixer B ............................................................... 425
Figure D.10  2\textsuperscript{nd} order harmonic and IM output from Mixer B ...................... 426

Figure E.1  Exponential response to 2 tones with 2\textsuperscript{nd} order truncation ............ 430
Figure E.2  Detail view of 2\textsuperscript{nd} order truncated exp() response ..................... 431
Figure E.3  Exponential response to 2 tones with 3\textsuperscript{rd} order truncation ............... 431
Figure E.4  Detail view of 3\textsuperscript{rd} order truncated exp() response ..................... 432
Figure E.5  Hyperbolic tangent response to 2 tones with 2\textsuperscript{nd} order truncation .... 438
Figure E.6  Detail view of 2\textsuperscript{nd} order truncated exp() response ..................... 439
Figure E.7  Hyperbolic tangent response to 2 tones with 3\textsuperscript{rd} order truncation .... 439
Figure E.8  Detail view of 3\textsuperscript{rd} order truncated exp() response ..................... 440
**LIST OF ABBREVIATIONS**

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC</td>
<td>Alternating Current: Within the context of spectral analysis, refers to spectral content not at DC</td>
</tr>
<tr>
<td>ADS</td>
<td>Advanced Design System: A microwave computer-aided design software environment available from Agilent</td>
</tr>
<tr>
<td>AOM</td>
<td>Arithmetic Operator Method: Frequency-domain convolution simulation method</td>
</tr>
<tr>
<td>BIPD</td>
<td>Basic Intermodulation Product Description: User-select tabular weightings of incommensurate input frequencies</td>
</tr>
<tr>
<td>CATV</td>
<td>Cable TV: System of multi-channel video content delivered over coaxial cable or fiber</td>
</tr>
<tr>
<td>CDMA</td>
<td>Code Division Multiple Access: A signalling scheme for wireless communications used primarily in cellular devices</td>
</tr>
<tr>
<td>CSO</td>
<td>Composite Second Order: Collection of 2\textsuperscript{nd} order intermodulation of 2 distinct frequencies mapping to a distinct frequency</td>
</tr>
<tr>
<td>CTB</td>
<td>Composite Triple Beat: Collection of 3\textsuperscript{rd} order intermodulation involving 3 distinct frequencies</td>
</tr>
<tr>
<td>DC</td>
<td>Direct Current: Within the context of spectral analysis, refers to spectral content at 0 Hz</td>
</tr>
<tr>
<td>FFT</td>
<td>Fast Fourier Transform: A fast method of computing the Discrete Fourier Transform</td>
</tr>
<tr>
<td>FM</td>
<td>Frequency Modulation: Constant-amplitude, variable-phase modulation method</td>
</tr>
<tr>
<td>GaAs</td>
<td>Gallium Arsenide: A material used to create high-speed semiconductor devices</td>
</tr>
<tr>
<td>HB</td>
<td>Harmonic Balance: A mixed time/frequency domain simulation method</td>
</tr>
<tr>
<td>I-FFT</td>
<td>Inverse Fast Fourier Transform: A fast method of computing the Inverse Discrete Fourier Transform</td>
</tr>
<tr>
<td>IM</td>
<td>Intermodulation: Distortion due to nonlinear mixing of signals at two or more input frequencies</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>MESFET</td>
<td>Metal Epitaxial Semiconductor Field Effect Transistor: A transistor created with a metal semiconductor gate instead of a bipolar junction</td>
</tr>
<tr>
<td>MNAM</td>
<td>Modified Nodal Admittance Matrix: An admittance matrix for nodal circuit analysis which includes voltage sources</td>
</tr>
<tr>
<td>PHEMT</td>
<td>Pseudomorphic High Electron Mobility Transistor: A high-speed transistor created by joining two materials without a doped bandgap</td>
</tr>
<tr>
<td>Spice</td>
<td>Simulation Program with Integrated Circuit Emphasis: Simulation Program with Integrated Circuit Emphasis</td>
</tr>
<tr>
<td>VFD</td>
<td>Vector Frequency Description: Tabular weighting of input frequencies spanning a modified vector space</td>
</tr>
</tbody>
</table>
# LIST OF SYMBOLS

- $A^H$: Hermitian of matrix $A$ .......................................................... 132
- $A_q$: Amplitude (Volts) of the $q^{th}$ sinusoid when using AOM ............... 29
- $B$: 3 dB passband of a bandpass filter ........................................... 297
- $\beta$: Multiplicative complex constant in correlation computations .......... 108
- $b_n$: Coefficients of the polynomial transfer function $y(t)$ .................... 22
- $\ast$: Operator denoting the convolution of two quantities ........................ 21
- $C_s^{(c)}$: Spectral vector resulting from repeated convolution · times ........... 58
- $\text{dB}$: Decibel, a logarithmic power measurement scale ........................ 14
- $\delta(\cdot)$: Dirac delta function ......................................................... 33
- $D_3(f)$: Third-order CATV nonlinearity parameter .................................. 249
- $D_2(f)$: Second-order CATV nonlinearity parameter ................................. 249
- $e_0$: Unit spectral vector corresponding to DC .................................... 22
- $E[g(\phi)]$: Expected value of the function $g(\phi)$ .................................. 109
- $\mathcal{H}$: Set of VFDs forming a VFD Table ...................................... 70
- $\eta_k^T$: $k^{th}$ VFD appearing in a VFD table, a row vector of integers ...... 47
- $\mathcal{H}^-$: Negative frequency portion of a VFD Table ............................. 77
- $\mathcal{H}^+$: Estimated storage required to hold VFD table under construction ... 71
- $e_{\tilde{m}}$: Iteration error in time domain .............................................. 162
- $J$: Exchange matrix ................................................................. 77
- $f_c$: Center frequency using the indexing scheme of Carvalho and Pedro ...... 23
- $F_{\mathcal{H}}$: VFD Table in frequency-sort order, symmetric about DC when 2-sided ...... 77
- $\tilde{\mathcal{F}}^{-1}[-]$: Inverse Fourier transform operation ............................. 86
- $f_{\text{in}}$: Vector of input frequencies (Hz) ........................................ 47
- $f_k$: $k^{th}$ frequency of interest, where $k = 1..K$ using BIPD method of Rhyne ... 18
∀ Means for any or for all as in ∀k.................. 19
\xleftrightarrow{\text{K}} Denotes functions forming a Fourier transform pair............. 36
f_\Phi(\phi) Probability density function of \phi ................. 109
f_q Frequency (Hz) of the q\textsuperscript{th} sinusoid when using AOM............. 20
\mathcal{F}[-] Fourier transform operation .................. 21
\Gamma_{kk} k\textsuperscript{th} element of phasor delay matrix \Gamma_n .................. 22
\Gamma_n Diagonal matrix of delay phasors corresponding to delay \tau_n .......... 22
A_n Associate memory (hash table) that returns VFD location in \mathcal{H}^+ ........ 76
H_{nc} Temporary matrix of candidate VFDs during VFD Table construction..... 74
I Identity matrix........................................ 25
I Number of convolution index steps in discrete convolution ............. 119
I(f) In-phase baseband data stream in frequency-domain .................. 100
\tilde{I}(f) In-phase modulated data in frequency-domain .................. 100
\mathfrak{Im}[-] Imaginary part of a complex argument .................. 32
\in Means is a member of the set ................................ 12
\|\|_\infty Infinity norm of a vector................................ 12
I Field of integers; \mathbb{I}^{1\times Q} denotes a row vector of integers........ 69
i(t) In-phase baseband data stream in time-domain .................. 100
j The imaginary number .................................. 12
K Number of entries in a VFD table ................................ 7
\hat{K}_n Estimate of the number of VFD table entries at nonlinear order n .... 71
\hat{K}^+ Estimate of the number of VFD table entries over all nonlinear orders.... 71
k_x Frequency index for vector X .................................. 48
k_y Frequency index for vector Y .................................. 48
k_z Frequency index for vector Z .................................. 48
\Lambda Diagonal matrix appearing in Eigendecomposition.................... 134
M Terminating iteration number .................................. 162
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu$</td>
<td>Chirp rate in a radar transmission system</td>
</tr>
<tr>
<td>$N$</td>
<td>Maximum nonlinear order of a transfer function</td>
</tr>
<tr>
<td>$N_{ht}$</td>
<td>VFD Table in one-norm sort order, symmetric about DC when 2-sided</td>
</tr>
<tr>
<td>$</td>
<td>\mathcal{H}</td>
</tr>
<tr>
<td>$</td>
<td>\mathcal{H}_n</td>
</tr>
<tr>
<td>$\omega_a$</td>
<td>Continuous-time or analog frequency (rad/sec)</td>
</tr>
<tr>
<td>$\omega_0$</td>
<td>Center frequency of a bandpass filter (rad/sec)</td>
</tr>
<tr>
<td>$\omega_p$</td>
<td>Pre-warped discrete-time equivalent of $\omega_a$</td>
</tr>
<tr>
<td>$|\cdot|_1$</td>
<td>1-norm of a vector</td>
</tr>
<tr>
<td>$P$</td>
<td>Permutation matrix</td>
</tr>
<tr>
<td>$\phi_m$</td>
<td>Vector of input phases (radians)</td>
</tr>
<tr>
<td>$\phi_q$</td>
<td>Phase (Radians) of the $q$th sinusoid when using AOM</td>
</tr>
<tr>
<td>$\psi$</td>
<td>Scaling factor for ladder filter circuits</td>
</tr>
<tr>
<td>$Q$</td>
<td>Number of independent input tones</td>
</tr>
<tr>
<td>$Q(f)$</td>
<td>Quadrature baseband data stream in frequency-domain</td>
</tr>
<tr>
<td>$\tilde{Q}(f)$</td>
<td>Quadrature modulated data in frequency-domain</td>
</tr>
<tr>
<td>$q_n$</td>
<td>Dummy index variable in multitone summation expressions</td>
</tr>
<tr>
<td>$q(t)$</td>
<td>Quadrature baseband data stream in time-domain</td>
</tr>
<tr>
<td>$C_{2k}^{(n)}$</td>
<td>Phasor component $k$ of $C_2^{(n)}$</td>
</tr>
<tr>
<td>$\Re {\cdot}$</td>
<td>Real part of a complex argument</td>
</tr>
<tr>
<td>$\Pi(\cdot)$</td>
<td>Rectangular function</td>
</tr>
<tr>
<td>$R_{XY}$</td>
<td>Complex correlation of quantities $X$ and $Y$</td>
</tr>
<tr>
<td>$\sigma_F$</td>
<td>Running frequency standard deviation</td>
</tr>
<tr>
<td>$\tau_n$</td>
<td>Order-dependent time delays in the polynomial transfer function $y(t)$</td>
</tr>
<tr>
<td>$\theta$</td>
<td>Phase angle</td>
</tr>
<tr>
<td>$U$</td>
<td>Unitary matrix, where $U^H U = I$</td>
</tr>
<tr>
<td>$\int_C$</td>
<td>Open contour integral along path $C$</td>
</tr>
<tr>
<td>Symbol</td>
<td>Definition</td>
</tr>
<tr>
<td>--------</td>
<td>------------</td>
</tr>
<tr>
<td>$f_C$</td>
<td>Closed contour integral along path C</td>
</tr>
<tr>
<td>$V_{be}$</td>
<td>Base-to-emitter junction voltage</td>
</tr>
<tr>
<td>$V_T$</td>
<td>Thermal voltage of a BJT</td>
</tr>
<tr>
<td>$W$</td>
<td>Numerator vector in rational polynomial</td>
</tr>
<tr>
<td>$Q(f)$</td>
<td>Sum of in-phase and quadrature modulated data in frequency-domain</td>
</tr>
<tr>
<td>$X(f)$</td>
<td>Fourier transform of $x(t)$</td>
</tr>
<tr>
<td>$X_q$</td>
<td>Phasor form of the $q^{th}$ sinusoid, i.e. $X_q = A_q e^{j\theta_q}$</td>
</tr>
<tr>
<td>$X_q^*$</td>
<td>Complex conjugate of phasor $X_q$</td>
</tr>
<tr>
<td>$X$</td>
<td>Spectral vector corresponding to $X(f)$</td>
</tr>
<tr>
<td>$x(t)$</td>
<td>Time-domain function designating a collection of sinusoids</td>
</tr>
<tr>
<td>$T_x$</td>
<td>Spectrum transform matrix created from $X$</td>
</tr>
<tr>
<td>$Y$</td>
<td>Output spectral vector</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Why Nonlinear Analysis in the Frequency Domain?

Computer simulation is used in many fields of theoretical and applied science, by both researchers and practitioners. Simulation is used to model things where it is impossible to make analogous scale models — such as planetary atmospheric behavior or the movements of celestial bodies — as well as situations where computer modeling (and subsequent revision as an integral part of the design flow) prior to fabrication of the real thing is far more economical than the alternative approach of trial and error prototype model construction with repeated design revisions. Fabrication of most semiconductor device designs is an example of the latter case where computer simulation is used. In most cases, behavioral models of real nonlinear systems are simulated with time as the independent variable. That is, most models of real systems are simulated with a view to predicting the response of the system if it were started at some point in time and then subsequently observed. Within the computer simulation environment, the starting moment is usually set to zero, the system’s behavior is initialized to some known state, and the system’s behavior going forward is observed. This is known as transient or time-domain simulation, and simulation environments operating in the time-domain usually operate by numerically solving a set of discretized nonlinear differential equations subject to whatever initial values are given.

Human beings have a natural preference for simulation in the time-domain, since
in addition to the three-dimensional spatial coordinates it is a physical dimension in which humans live and it is a dimension which they can visually observe. With the assistance of measuring devices they can observe and measure time differences beyond human observational abilities. There are, however, other domains upon which scientific inquiry and experimental observation may be based, but since these domains are alien to human existence, they are not as popular. The frequency-domain is one such domain. The frequency-domain is one of inverse observational characteristics to the time-domain: Impulsive time-domain behavior such as noise spikes appears as white noise that spans the entire frequency domain as a constant value, while pairs of positive and negative impulsive spikes (at the same absolute frequency value) in the frequency-domain correspond to an infinitely-oscillating sinusoidal signal (unless the frequency at which the frequency-domain impulse occurs is zero, which is then just a constant DC signal). Where it has proven useful, sophisticated measuring devices — such as spectrum analyzers — which produce visual output in the frequency-domain have also been produced, although these devices necessarily operate in the time-domain and post-process measurements into the frequency-domain.

Despite the human preference for the time-domain, there are classes of scientific and engineering problems which are far more amenable to analysis in the frequency domain. In the realm of optics, for example, physicists are far more accustomed to analyzing portions of frequency-domain spectra when considering wave propagation; use of the time-domain for such problems would be inappropriate, if not inconceivable. In the field of electrical engineering, there exist similar classes of problems which are far more amenable to nonlinear frequency-domain analysis than they are to nonlinear time-domain analysis. The preference for the frequency-domain is in some cases driven by computational economics and in other cases by the relative tractability of the analysis. An example involving computational economics is the analysis of up- or down-conversion modulation circuits. Such circuits nonlinearly operate on small portions of frequency-domain spectra and translate them from low-frequency spectra near DC (a.k.a. the baseband) to high-frequency spectra centered about the modulation frequency (in the case of up-conversion), or vice-versa for down-conversion. In both cases, however, accurate time-domain simulation will require a time-domain simulator to operate over several wavelengths of the lowest-frequency baseband spectral content, and this will necessarily correspond to a long simulation time even at the baseband, but in terms of the amount of computation at the high-frequency modulated signal, it will be highly inefficient, since a few periods at the modulation
frequency happens in a small fraction of the time required to simulate the baseband. A typical modulation or demodulation circuit simulated over just a few baseband frequency periods will result in millions or even billions of periods of oscillation in the modulated signal. Note that this inefficiency is an inherent consequence of using time-domain simulation techniques to simulate the \textit{intended} nonlinear behavior of a particular circuit – and the existence of this inefficiency has spurred interest in hybrid simulator environments that combine time-domain and frequency-domain simulation to increase computational efficiency while still obtaining useful output for evaluating the simulated model’s conformance to performance metrics.

In addition to problems where the use of time-domain simulation is simply inherently inefficient, there are a few other classes of problems which are simply intractable for simulation in the time-domain. One example is the problem of frequency planning or frequency spectrum allocation, which is ultimately the assignment of carrier frequencies in multi-carrier systems with the objective of minimizing the in-band distortion products produced by nonlinear interactions (usually caused by nonlinear amplification) of two or more carriers for other bands. This problem is well-known in the field of satellite communications, where very broad amplification bandwidth spanning several frequency octaves combined with narrowband frequency assignments causes concern with the harmonic distortion of carriers in the lower adjacent frequency octave. A similar problem exists in other multicarrier systems, such as television broadcasting and cellular communications. In these realms, a related problem is the performance of nonlinear amplifiers for a given frequency plan. In some cases — Code Division Multiple Access (CDMA) cellular communications, for example — performance metrics are defined by signal inputs in the time-domain with output measurements in the frequency-domain, and in these cases, one of the hybrid time- and frequency-domain simulator environments that perform harmonic balance analysis is appropriate. However, there are cases where the signal input for performance metrics is defined in the frequency-domain and the outputs are measured in the frequency-domain. Cable and satellite television broadcasting is an example. Here, independent carriers — with no firmly defined time-domain behavioral definition for their modulated baseband signals — are combined and amplified by a nonlinear amplifier, either for rebroadcast by a satellite down-link or for redriving down a cable television network trunk cable. Transient simulation is thus useless and simulation — when it is used at all — must be done in the frequency-domain.

Still other cases lend themselves to the possibility of simulation in either domain, with one domain’s advantages being the other’s disadvantage. However, when the output perfor-
mance metric is defined in the frequency-domain, there is a possibility that numerical limitations in the simulator environment may tend to preclude using an input in the time-domain. For example, if very low-level signals defined in the time-domain (for example, simulations of radio signals received from space and processed through low-noise amplifiers in a temperature-controlled environment — usually with the amplifier immersed in liquid nitrogen) are processed through a nonlinearity and then the output is transformed to the frequency-domain using the Fast Fourier Transform (FFT), the resulting signal may be very close to the numerical noise floor such that invalid results are obtained. In these cases, numerical computation must be minimized and thus the use of the FFT must be avoided, which leads to defining the input in the frequency-domain.

In general, circuit simulation is performed predominantly in the time-domain. However, there are cases where frequency-domain simulation is preferable, and others where it is the only available simulation option. The growth of the commercial offerings in the hybrid simulation environments is a testament to the pragmatism of engineers for choosing the environment best suited to the needs of their problems, but also indicates an increasing acceptance for using the frequency-domain as an alternative. There is, however, a dearth of nonlinear simulation environments that operate solely in the frequency-domain. The body of work presented here is a contribution to remedying this situation.

1.2 Motivations and Objectives of This Study

Simulation is an essential tool for both researchers and practitioners in the field of electrical engineering. Circuit simulation in the time domain using software based on the Simulation Program with Integrated Circuit Emphasis (Spice) engine has been routine since the 1970s. However, Spice’s ability to perform simulation in the frequency domain is limited to a linearized analysis of small perturbations about a quiescent operating point. Furthermore, since it is not efficient to simulate circuits containing sinusoidal sources with frequencies of differing orders of magnitude in the time domain, researchers developed the Harmonic Balance (HE) method – a combination of time and frequency domain simulation – to permit the simulation of nonlinear systems where it is expected that the nonlinear system will have a steady-state response. While HE simulation is an improvement over the limited capability of Spice, it can only successfully simulate nonlinear systems when the number of independent input tones is limited. Popular
contemporary commercial simulators are limited to twelve or fewer input tones.

In the realm of simulation entirely in the frequency domain, circuit simulator engines based on power series analysis have been used to simulate memoryless nonlinearities and systems with order-dependent time delays. In these simulators, it was thus possible to simulate more strongly nonlinear systems with larger-signal inputs. However, these simulators used ad hoc techniques to tabulate the frequencies at which output signals would occur, and reported results were typically oriented toward characterizing two-tone, 3rd order intermodulation amplifier responses. No attempt was made in these simulator engines to handle large numbers of input tones. In an exception to this generality, one group of researchers implemented a rigorous technique to identify the set of output frequencies based upon a set of uniformly-spaced input tones. Researchers working on implementing Volterra series analysis in frequency domain simulators have been able to simulate complex weakly nonlinear systems containing memory. However, these efforts have also been limited to low orders of nonlinearity and low numbers of input tones because of the complexity of extracting the Volterra kernels for incorporation into the simulator engine.

Today, there is an increasing need for frequency-domain simulator environments capable of handling large numbers of independent input tones and high orders of nonlinearity. Both requirements are outside the scope of simulators based on HB methods. Furthermore, there is a desire for simulator environments with very high dynamic range. Dynamic range tends to be limited in HB simulators because of the accumulation of numerical error due to repetitive use of Newton iteration. Within such Newton iteration loops a contributing factor to numerical error is repeated use of the Fast Fourier Transform (FFT) and Inverse Fast Fourier Transform (IFFT). Finally, there is also a need for an environment that can rapidly compute the average output power when the input is a set of multitonies with uncorrelated random phases. With one exception that will be noted, existing environments enforce correlation of multitone input signals, thus requiring simulation users to perform multiple simulation runs and perform averages of the ensembles of enforced-correlation responses in order to arrive at the correct result. Furthermore, there is a desire for an environment that can facilitate the easy separation of correlated and uncorrelated phase components of intermodulation distortion. No existing frequency domain simulation environment has been able to do this, although it has been done with some success through extensive post-processing of time-domain results. The objective of this study is to develop such a simulator environment and demonstrate its utility.
for investigating problems beyond the capability of existing HB simulator environments and other existing frequency domain simulator environments.

1.3 Dissertation Overview

The literature on computer-aided simulation of electrical and electronic circuits is vast, dating back to the 1950s. No single review could do it justice. Thus the objective of Chapter 2 is emphasizing literature discussing simulator environments, with special emphasis on the literature about simulators operating in the frequency domain. The chapter begins with brief coverage of noteworthy literature about time domain and HB simulators, discusses some efforts in the realm of simulators based on Volterra series analysis, and then delves more deeply into the literature on frequency domain simulators. The origins of the Basic Intermodulation Product Description (BIPD) and Spectrum Mapping tables and Arithmetic Operator Method (AOM) as implemented in the FREDA2 circuit simulator are described. The novel technique that identified output frequencies based on a set of uniformly-spaced input tones is also described. Other efforts to identify output frequency sets are also noted.

Chapter 3 begins with the mathematical foundations for the Arithmetic Operator Method in the form of a select set of Fourier transforms of generalized functions. The foundation begins with Euler’s formula, followed by a discussion of properties of the \( \sin \frac{x}{x} \) function, and then proceeds to describing Dirac delta function along with a brief discussion of generalized functions. This is followed by Fourier Transforms of select generalized functions and the convolution equations that undergird the Arithmetic Operator Method. An illustrative overview section using a 2-tone input signal then covers applications of all of the concepts underlying AOM and implemented in the AOM Toolbox. Subsequent sections then delve formally into each of AOM’s data objects and the algorithms in the AOM Toolbox that create them. The Vector Frequency Description (VFD) and its vector nature are defined and its vector properties enumerated, and the algorithm for creating a VFD table is described and illustrated with an example of three commensurate input tones defining a VFD Table. Exploitation of the vector nature of the VFD table to extend the VFD table for two-sided input is also noted. This leads into a discussion of the Spectral Vector, which uses the established 3-tone VFD table as a basis to describe the correlation properties of spectral vectors when they are created from multitone inputs with uncorrelated phase. Two theorems and several remarks about the
correlation properties of spectral vectors created from uncorrelated phase multitone inputs are given. The peculiar phenomenon of correlated intermodulation distortion occurring at higher power levels than uncorrelated distortion due to interaction with a previously rectified Direct Current (DC) signal is illustrated in the 3-tone analytical example. The Spectrum Mapping table (which drives the creation of the spectrum transform matrix) is then described, with a description of an improved algorithm (versus previous work) which reduces the computational complexity of creating the spectrum mapping table from $O(K^3)$ to $O(K^2)$, where $K$ is the number of entries in the VFD table. The Spectrum Transform matrix and its construction algorithm are then described, with an emphasis on the exploitation of the sparsity of the matrix in its construction. An important discovery that two-sided complex-conjugate spectral vectors yield Hermitian spectrum transform matrices is then described and proved. This discovery provides a path for the use of eigendecomposition as an alternative to the usual linear algebra methods for solving systems of equations, and may well be the only way to solve problems with rational polynomial transfer functions when the input is a multitone signal with uncorrelated phase. A final section derives the one-sided version of the Arithmetic Operator Method, first introduced by Chang, as a special case of the two-sided complex version when the input signal is purely real. The final section also expands on the discussion of implementing exponential and hyperbolic functions using the Arithmetic Operator Method, filling in some details that Chang did not include in his work.

Chapter 4 validates the functions of the AOM Toolbox by simulating the response of a logarithmic amplifier to a two-tone sinusoidal input signal. The nonlinearly-created output phasors of the AOM Toolbox simulation are expanded in the time domain and compared to the results of a pure time-domain simulation of the input signals using Matlab® natural logarithm function to perform the computations. Results from a PSpice simulation of a circuit netlist are also shown and compared in order to have a reference independent from the Matlab® computing environment. The effects of spectral truncation — a necessity whenever when transcendental transfer functions are simulated — are illustrated.

Chapter 5 describes the application of the AOM Toolbox to the prediction of the response of an amplifier to multitone input signals with both correlated and uncorrelated phases. More precisely, the inputs are uncorrelated, but the AOM Toolbox treats the inputs with an enforced correlation scheme akin to that of any time-domain simulator and of other frequency-domain simulators; the AOM Toolbox also treats the inputs properly as uncorrelated phase
inputs thanks to the ability of the VFD table to permit more than one spectral vector element at each numeric frequency. When correlation is enforced by the AOM Toolbox, it is possible to observe the effect of input phase on intermodulation power in the adjacent bands and note that it is similar to the results produced by environments that enforce correlation. When the AOM Toolbox treats the uncorrelated phase inputs (i.e. inputs where the phases are independent, identically distributed random variables \[1, 2\]) properly however, it is possible to predict the average power of the output spectrum from a single simulation, something which is has been demonstrated in only one other instance \[3\] — but an instance, unfortunately, where measurements used phase-locked sources as contrasted to the unlocked sources used in this work. Recovery of the correlated and uncorrelated co-channel intermodulation power is also possible in a single simulation, and results of this investigation are given.

Chapter 6 explores an application of AOM Toolbox to simulating a broadband linear Frequency Modulation (FM) chirp \[4\] through a radar transmission chain. Beginning with the analytical description of the linear FM chirp in the frequency domain (a Fresnel integral form), a discretized version of the input signal will be created in the frequency domain and processed through the chain in the AOM Toolbox. Results are compared to those produced from the FFT of the \texttt{fREEDA}™ output of the same chain in the time domain for inputs with 500 MHz and 1 GHz chirp bandwidths.

Chapter 7 addresses the realm of multicarrier broadband transmissions modeling, for which there are presently no simulation tools whatsoever. Thus a frequency domain behavioral modeling environment capable of handling a large number of tones would prove quite useful in these fields. One commercial application is to the prediction of Intermodulation (IM) in Cable TV (CATV) distribution networks. An illustrative use of the AOM Toolbox to predicting the Composite Second Order (CSO) and Composite Triple Beat (CTB) \[5\] distortion produced by a hypothetical amplifier is given. (For lack of broadband amplifier models, comparison to measured results will not be possible.) A 79-channel case furnishes results that can compared to previous reports in the literature, while a 158-channel case produces results not previously reported.

Chapter 8 summarizes this body of work and discusses areas for further research.

Appendix A discusses Z-Domain Filtering in a transient circuit simulator. The issue of ill-conditioned modified nodal admittance matrices arising from stamping in a high-order behavioral model of an S-domain filter is described to motivate the need for the Z-Domain
models, followed by descriptions of Butterworth and Chebychev filter blocks that may be synthesized into an arbitrary number of cascaded blocks so as to facilitate the creation of filters of arbitrary order for use in transient simulation. Results of modeling two filters are given.

Appendix B describes the design and characterization of the uncorrelated phase multitone signal generator which was measured in Chapter 5.

Appendix C encapsulates the content of the first publication by the author (as lead author) on an early prototype implementation of the AOM Toolbox in Matlab®. The work disclosed in this publication has been so thoroughly eclipsed by the subsequent developments in the AOM Toolbox codes that it could not be effectively included in the main body of the dissertation.

Appendix D provides a User’s Guide to the AOM Toolbox with a specific application to the analysis of two cascaded mixer stages that perform a heterodyne demodulator function.

Appendix E furnishes code listings with embedded documentation.

1.4 Publications

1.4.1 As Primary Author


1.4.2 Not As Primary Author


1.4.3 Planned Publications

1. AOM Improvements This paper will disclose results of Chapters 3 and 4 in detail and may disclose selected results of other chapters (followup to the 2003 foundations paper).

2. Correlated and Uncorrelated Distortion in IS-95 Reverse Channels, targeted as a letter to IEEE MWCL or a 6-page T-MTT paper. This paper will include correlation properties of spectra using the VFD from Chapter 5 and results of Chapter 5. It will show correlated and uncorrelated output spectra for a QPSK input signal also attempt to show that EVM can be estimated from the magnitude of the distortion phasors.

3. CATV Trunk Amplifier Behavioral Modeling, targeted as an IEEE T-MTT or T-Broadcasting journal paper. This paper will disclose the results of Chapter 7.
Chapter 2

Literature Review

2.1 Introductory Remarks

This chapter provides an introduction to the relevant literature pertaining to simulation of circuits, with the emphasis on behavior environments and frequency-domain methods. Due to the vast breadth of the literature, the most vital literature is cited, and these publications contain references which furnish greater depth.

2.2 Notation

The notation used in this work will generally follow that given in [6]:

Matrices: Upper case bold Roman letters, e.g. $A$, but may include a subscripted letter, e.g. $T_x$.

Column Vectors: Lower or upper case bold Roman letters, e.g. $v$, or lower or upper case bold Greek letters, e.g. $\xi$, but may include a subscripted letter, e.g. $Y_1$.

Row Vectors: Lower case bold Roman letters with the transpose indicator, e.g. $v^T$, or lower case bold Greek letters with the transpose indicator, e.g. $\xi^T$.

Scalars: Lower case Roman or Greek letters, possibly with subscripts, e.g. $\alpha, \beta, v_i, a_{ij}$. 
Index variables: $i, j, k, l, m, n, q$. 

Set Membership: $\in$ means is a member of the set. 

Norms: $\|v\|_1$ will denote the 1-norm of vector $v$. Note that $\|v\|_1 = \sum_{i=1}^{I} |v_i|$ for some $I > 1$. $\|v\|_\infty$ will denote the $\infty$-norm of vector $v$. Note that $\|v\|_\infty = \max(|v_i|)$ for $i \in 1..I$, with $I > 1$.

The elements of the matrix $A$ are denoted $a_{ij}$ or $A_{ij}$, and the elements of vector $v$ are denoted $v_i$. The imaginary number will be denoted as $j$.

### 2.3 Time Domain Simulation

Simulation of electronic systems in the time domain has been routine since the introduction of the Spice circuit simulator by Nagel at the University of California at Berkeley in the mid-1970s \cite{7} (under the direction of Pederson \cite{8} and Rohrer \cite{9}). However, the Spice engine has limited applicability in the frequency domain — roughly equal to that disclosed in the CANCER simulator first discussed by Rohrer \cite{9}. Specifically, it is limited to simulating small, linear perturbations about a quiescent operating point — what is now well-known as Alternating Current (AC) analysis \cite{10, 11}. At least one historical authority on Spice \cite{8} implied that simulation in the frequency domain was a secondary concern at the time Spice was developed. Nevertheless, Spice became wildly popular as students who had used the software in courses at Berkeley found themselves writing back to Berkeley for program listings so they could use it professionally \cite{12}. A number of books on the theory of circuit simulation \cite{13, 15} were authored by those who participated in the research and development of Spice. These books remain relevant references today. Other relevant references not authored by researchers from Berkeley include books by Vlach and Singhal \cite{16} and Ogrodzki \cite{17}. Researchers at IBM also developed complete time-domain simulation environments independently and concurrently with the work at Berkeley; a good historical reference is \cite{18}.

### 2.4 Mixed Time and Frequency Domain Simulation

At around the same time in the mid-1970s, researchers became aware that simulations rooted entirely in the time domain were computationally inefficient for some classes of problems, and they proceeded to develop alternative methods for finding the response of a nonlinear system
to periodic sinusoidal inputs. Two alternatives for achieving this end were “Shooting Methods” and the simultaneous development of the HB by Baily and Lindenlaub — both implementations of Galerkin’s methods. The “Shooting Methods” essentially added boundary values at beginning and end time points, but were otherwise time-domain simulation engines. While more efficient than time-domain simulation, since transient decay conditions are eliminated through the enforcement of boundary conditions, shooting methods could still be inefficient since the entire network — both linear and nonlinear portions — was simulated to a time dictated by the period of the lowest frequency input. The same was true of the early HB implementations, which optimized systems of equations of truncated Fourier series for every node — both linear and nonlinear — in the system.

In 1976 Nakhla and Vlach disclosed further gains in the computing efficiency of HB by partitioning the network into linear and nonlinear portions, as shown in Fig. 2.1. Their simulation engine simulated the linear portion in the frequency domain while continuing to simulate the nonlinear portion in the time domain. A Newton iteration loop containing the FFT and I-FFT is used to iteratively determine currents $i_n$ and $\bar{i}_n$ at the nodal interfaces between the linear and nonlinear portions of the system. Kirchhoff’s Current Law demands that these currents sum vectorially to zero, and this forms the basis for the error function that must be satisfied. Nakhla and Vlach termed their implementation “piecewise” HB. The term “piecewise” has since been dropped, but contemporary HB simulators remain based on this same partitioning of circuits into linear and nonlinear portions. Despite being credited with the idea of using the FFT to solve the sinusoidal steady-state problem, however, it appears that one researcher proposed this idea — without the partitioning — about seven years earlier. Since Nakhla and Vlach’s developments, HB methods have garnered much more research attention, particularly from the University of California at Berkeley and the University of Bologna, Italy. Kundert and Sangiovanni-Vincentelli marked Berkeley’s work on HB simulation with the release
of Harmonica in 1985 [24, 28], and disclosed improvements to handle almost-periodic signals in HB in [29]. At about the same time, Hewlett-Packard released a commercial HB simulator product based on the work at Berkeley, and the EESoF company (which later merged with Hewlett-Packard) also released a commercial product [30]. (Agilent is today a separate company from Hewlett-Packard, having been formed of HP’s former instrumentation division.) This was followed by authoritative books on HB simulation [31] and the usage of HB in a commercial product, Cadence Design System’s Spectre [32]. In the mid-1990’s, Bell Labs [33] disclosed the first HB environment to use Krylov subspace methods [34] to simulate a large-scale circuit [35].

At the University of Bologna, Italy, a similar path of scaled development progress on an HB simulator was disclosed by Rizzoli, beginning in 1983 with an environment tailored to microwave circuits and demonstrating HB on single tones [36]. Further enhancements led to the disclosure of two-tone results in 1988 [37] followed by improvements in the formulation of the Jacobian matrix [38]. A commercial HB offering based on the Bologna work was introduced by Compact Software in 1987 [39]. More recently the simulator was scaled to handle larger systems through the use of Krylov subspace solver methods [40, 41]. Other recent results include implementation of spline-based behavioral models [42], but the use of splines with Galerkin methods has previously been reported in other fields [43]. However, despite the ability to simulate larger systems with Krylov methods, when the input stimulus was sinusoidal the published results still showed inputs consisting of only two independent sinusoidal tones. Results for other spectrally-rich signals used inputs defined in the time-domain [44].

Even today, the leading commercial simulator product for microwave circuit design permits a maximum of only twelve input tones [45] for a harmonic balance simulation. This presents a particular problem for simulating broadband multicarrier systems as they are naturally described in the frequency domain; converting the stimulus for such an input to the time domain in order to suit HB methods makes no sense. Another issue in the use of HB concerns aliasing in the FFT when two or more input tones are spaced closely together. In this case, it becomes necessary for the FFT to use a large number of points [46]. Furthermore, while the stability of the FFT in single forward and inverse operations is well-established [47], it is unclear whether the accumulation of numerical error from repeated use of the FFT in the iterative solver loops causes enough error to prevent convergence to a solution. Finally, recent results of two-tone simulations indicate that the dynamic range of leading commercial simulators is limited to 125 dB [48].
2.5 Envelope Transient Simulation

Although it is a relatively recent development [49–51], envelope transient simulation is now implemented pervasively in commercial simulator environments. Envelope simulation uses at least two different time scales proportionate to underlying time constants in order to increase computational efficiency versus simple transient simulation. A typical simulation might have one time scale for a high-frequency carrier signal with a small period (and thus, a rapidly decaying time constant) and another, much slower, time scale oriented toward simulating the slowly-varying envelope of the modulated signal (with a much longer time constant akin to a corresponding baseband signal). Envelope transient simulation is often used for modeling up-conversion and down-conversion mixers, and these are inherently stiff physical systems since they intentionally change the frequencies of sinusoidal signals by several orders of magnitude through modulation or demodulation.

Since the physical systems being simulated are inherently stiff, it perhaps isn’t surprising that one of the issues with envelope transient modeling is its inherent stiffness [52]. Nevertheless, this issue and its subtleties were adequately understood only after several years of research [53, 54]. In practice as well as research [55, 57], envelope transient techniques have been combined with HB methods.

2.6 Nonlinear Frequency Domain Simulation

The realm of nonlinear simulation entirely in the frequency-domain is divided broadly into two categories — simulation using Volterra functional models and simulation using models other than Volterra models, with the latter category further divided into power series methods and convolutional methods. It should also be noted that, although their work is understood to be the forerunners of modern-day HB methods and is usually discussed in an HB context, the results of Baily [20] and Lindenlaub [21] were also achieved entirely in the frequency-domain.

2.6.1 Simulation with Volterra Functionals

The earliest noteworthy frequency domain simulator that could model nonlinear behavior from a netlist was SIGNCAP [58], a simulator engine rooted in the works of Norbert Weiner [59] and developed by SIGNATRON under contract to the U.S. Air Force. SIGNCAP could find
frequency-domain transfer functions of circuits based on weakly non-linear Volterra functional models. The study that led to the development of SIGNCAP was spurred by the observation that memoryless weakly nonlinear power series models were often insufficient for modeling RF receivers, and the need to include memory effects was identified. The developers of SIGNCAP benefited significantly from earlier research on Volterra functional modeling of bipolar transistor amplifiers. In particular, Narayanan disclosed a program that could model cascades of bipolar transistors. SIGNCAP was never made available to the public, but much of the development done under that contract is described in a book published by (contract engineering firm) SIGNATRON that is still a useful reference for understanding the relationship between Volterra systems with memory and memoryless nonlinear systems. The mathematics describing the effects of multitone mixing are also used quite widely. Some of the material from this book is also covered in.

Another general-purpose simulator based on weakly nonlinear Volterra series models was disclosed by Maas in 1988. The simulator, apparently called VERMIN, was written in Turbo Pascal and designed to run on personal computers and incorporated the MESFET Volterra series model developed earlier by Minasian. However, the paper is equally noteworthy for an admission on the part of the author that simulators of that era were capable of accurately simulating much stronger nonlinearities. Ushida and Chua observed the same result several years earlier in a paper disclosing improved algorithms for Maas later used an apparent successor of VERMIN, called C/NL2, to simulate spectral regrowth using Volterra series analysis and followed that with work on a commercial product from Applied Wave Research.

Going in the opposite direction from Maas’ modest PC-based simulator, researchers at Aalborg University in Denmark created a Volterra-series simulator, called hovsa, that was designed to execute on parallel computers. Of particular note was their ability to take on harder nonlinearities: Results were reported for nonlinear MESFET circuits driven by one tone up to the 9th order, with a speedup of about 6.6 for 8 parallel processors. With the parallel speedup, the reduction in computing time was from 91+ hours to 13+ hours for a 9th order nonlinearity. However, for 6th order nonlinearities — a degree not attempted by others — hovsa produced results in about 1.5 minutes on one processor.

Among recent developments in Volterra series simulators is the work done by researchers led by Brazil, at the University College of Dublin. In 2003, they disclosed the
development of a behavioral modeling simulator based on discrete (Z-domain) Volterra series models \cite{76,77} and published results comparing a 5th order discrete Volterra series behavioral model to measurements for an IS-95 Code Division Multiple Access (CDMA) input signal. The simulator environment was developed using a combination of Visual C++ and Matlab\textsuperscript{®} codes; curiously, no execution times for the simulations were furnished. In 2004, a related publication discussed reductions in the model complexity \cite{78}, but again no times of execution for the simulator were furnished. Another recent development in Volterra analysis is the 5\textsuperscript{th} order Advanced Design System (ADS) behavioral model developed by Boulejfen \cite{79}.

### 2.6.2 Simulation with Power Series Models

The earliest general-purpose frequency domain nonlinear circuit simulator environment (not based on Volterra series) is \textit{FREDA}, disclosed by Rhyne et al \cite{80,81}. The simulator benefited from the same partitioning of a network into linear and nonlinear portions for error function creation — just as in HB simulation — but made no attempt to simulate anything in the time domain. Instead, the simulation was performed entirely in the frequency domain. Central to \textit{FREDA} was the implementation of memoryless power series analysis, as described by Price \cite{82}, for nonlinear device modeling. \textit{FREDA} was later enhanced \cite{83,84} to permit power series models including memory effects as described by Steer \cite{85}. Computation of the response of nonlinear devices in \textit{FREDA} was done via direct evaluation of multinomial response formulas to determine the amplitude and phase at each output frequency. Once the mixtures defining the set of output frequencies was known, determination of the nonlinear response was straightforward. The nonlinear response was then used by the error function in an iterative optimization loop.

\textit{FREDA} tabulated frequencies of interest in a table called the \textit{Basic Intermodulation Product Description} (BIPD) Table \cite{86}. A single entry in the table is referred to as a BIPD. Each BIPD is a set of integers that describes the weightings, \( n_k \), of a set of \( K \) user-chosen frequencies, with each of the user-chosen frequencies having an integer, or tuple, assigned to describe the weighting, so that an output frequency \( f_o \) is given by

\[
 f_o = n_1 f_1 + n_2 f_2 + \ldots n_k f_k + \ldots + f_K n_K .
\]  

(2.1)

By the nature of nonlinear mixing under investigation, intermodulation distortion products will occur at the same set of user-chosen frequencies, and their effects will be summed together to create the simulator output at that frequency.
For example, consider a mixer circuit that is assumed to have nonlinear behavior up to the 8th order. Denoting the maximum nonlinear order as $N = \max(n) = 8$ in this case, so that $n$ can range from 1 to 8. The frequencies of interest are the RF carrier, $f_3$, the Local Oscillator, $f_2$, the Intermediate Frequency created by down-conversion, $f_1$, and DC, $f_4$, as shown in Fig. 2.2 (By definition, then, $f_1 = f_2 - f_3$.). As shown in Table 2.1 on the following page, each of the user-chosen frequencies of interest is assigned a unit positive weighting. An algorithm then determines every possible weighted combination of the user-chosen frequencies whose sum is equal to each user-chosen frequency, up to the maximum nonlinear order. Here DC is excluded from inclusion as a weighting frequency because of an assumption that it is not being driven as an input — usually a safe assumption in microwave circuits. However, since there are weighted combinations that map to DC, it is included as one of the user-chosen frequencies in the table. Thus, the weightings occur for $f_1$, $f_2$, and $f_3$, so that $K = \max(k) = 3$ in this case. Now, for each $f_o$ (output frequency) in the leftmost column of Table 2.1 weighted combinations of the $n_k$ are sought such that

$$f_o = \sum_{k=1}^{K} n_k f_k ,$$

subject to the constraint that the sum of the absolute values of the $n_k$ cannot be greater than
the maximum nonlinear order,

\[ n = \sum_{k=1}^{K} |n_k|, \ \forall n \in (1..N). \tag{2.3} \]

Table 2.1 is filled out with the remaining weighted combinations that meet these constraints. Note that the weightings, the \( n_k \), must be integers here, since the assumed underlying model is a power series of increasing integer nonlinear order. Note also that it is not necessary to have numeric values to verify the table; it can be done symbolically by simply keeping in mind that \( f_1 = f_2 - f_3 \) in this case.

Table 2.1: Mixer BIPD Table built using the method of Rhyne

<table>
<thead>
<tr>
<th>Output Frequency</th>
<th>( n )</th>
<th>( n_1 )</th>
<th>( n_2 )</th>
<th>( n_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_1, IF )</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>-1</td>
<td>2</td>
<td>-2</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>3</td>
<td>-2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>-2</td>
<td>3</td>
<td>-3</td>
</tr>
<tr>
<td>( f_2, LO )</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>-1</td>
<td>2</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>2</td>
<td>-1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>-2</td>
<td>3</td>
<td>-2</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>3</td>
<td>-2</td>
<td>3</td>
</tr>
<tr>
<td>( f_3, RF )</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>-1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1</td>
<td>-1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>-2</td>
<td>2</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>2</td>
<td>-2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>-3</td>
<td>3</td>
<td>-2</td>
</tr>
</tbody>
</table>

*Continued on next page*
Table 2.1 (continued)

<table>
<thead>
<tr>
<th>Output Frequency</th>
<th>$n$</th>
<th>$n_1$</th>
<th>$n_2$</th>
<th>$n_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_4$, DC</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>2</td>
<td>-2</td>
<td>2</td>
</tr>
</tbody>
</table>

One weakness in the algorithm that creates BIPD Tables is that the computational effort grows geometrically for each additional frequency of interest since it works through nesting for loops to a depth of $K$, the number of non-zero frequencies of interest. Such nesting is impractical when the number of non-zero input frequencies is very large.

2.6.3 Simulation with the Arithmetic Operator Method

In 1988, Haywood and Chow [87] published a paper disclosing the creation of a frequency domain simulator environment rooted in the simple observation that, when an input signal to a nonlinear power series model is composed of a discrete sum of sinusoids, the frequency domain response at each order of nonlinearity could be computed through repeated discrete convolution.

Specifically, they presumed an input signal $x(t)$ that is composed of $Q$ incommensurate tones as described similarly in [62],

$$x(t) = \sum_{q=1}^{Q} A_q \cos(2\pi f_q t + \phi_q)$$

$$= \frac{1}{2} \sum_{q=1}^{Q} A_q e^{j(2\pi f_q t + \phi_q)} + \frac{1}{2} \sum_{q=1}^{Q} A_q e^{-j(2\pi f_q t + \phi_q)}$$

$$= \frac{1}{2} \sum_{q=1}^{Q} X_q e^{j2\pi f_q t} + \frac{1}{2} \sum_{q=1}^{Q} X_q^* e^{-j2\pi f_q t},$$

where $A_q$, $f_q$, and $\phi_q$ are the amplitude (assumed non-negative), frequency, and phase offsets, respectively, of each signal component, and where $X_q$ is the phasor representation of the amplitude and phase, so $X_q = A_q e^{j\phi_q}.$ Also, $X_q^* = A_q e^{-j\phi_q}$ is the complex conjugate of $X_q$. In the
frequency domain, the Fourier transform of this signal is given by (see Section 3.2.8 on page 40):

\[ X(f) = \mathcal{F} [x(t)] = \frac{1}{2} \sum_{q=1}^{Q} [A_q e^{j\phi_q} \delta(f - f_q) + A_q e^{-j\phi_q} \delta(f + f_q)] . \]

\[ = \frac{1}{2} \sum_{q=1}^{Q} [X_q \delta(f - f_q) + X_q^* \delta(f + f_q)] . \]  

(2.7)

Next, noting that repeated multiplication of time domain functions corresponds to repeated convolution in the frequency domain \cite{HaywoodChow}, Haywood and Chow observed that

\[ \mathcal{F} [(x(t))^n] = \underbrace{X(f) \ast X(f) \ast \cdots \ast X(f)}_{n \text{ times}} , \]  

(2.8)

where \( \ast \) denotes the convolution operation. The notion of achieving time domain multiplication via discrete convolution in the frequency domain was extended by Chang, Steer, and Rhyne \cite{ChangSteerRhyne}, who termed a collection of possible frequency domain operations using discrete convolution the *Arithmetic Operator Method* (AOM) and combined AOM operations with many of the notions of generalized power series analysis. Chang later extended the use of AOM \cite{Chang} to several transcendental transfer functions \cite{Chang} in a new version of *FREDA* called *FREDA2* \cite{FREDA2}, which also included improvements in Jacobian construction to speed the iterative solver loops \cite{FREDA2}.

Central to the implementation of AOM in *FREDA2* was the construction of a matrix, called the *Spectrum Transform Matrix*, that permitted discrete convolution to be performed by a simple matrix-vector multiplication, i.e. given an input signal \( X(f) = \mathcal{F} [x(t)] \) formed as a *Spectral Vector* \( \mathbf{X} \) in the frequency domain, the spectrum transform matrix \( \mathbf{T}_x \) is constructed such that

\[ \mathcal{F} [x(t) \cdot x(t)] = \mathcal{F} [x^2(t)] = X(f) \ast X(f) \]

\[ = \mathbf{X} \ast \mathbf{X} = \mathbf{T}_x \mathbf{X} , \]  

(2.9)

so that squaring a time-domain function corresponds to convolution in the frequency-domain, but is implemented with a simple matrix-vector multiplication operation. An explanation of how to construct spectral vectors and the spectrum transform matrix is given in \cite{Chang, FREDA2}. Note that repeated multiplication of \( \mathcal{F} [x(t)] \) by \( \mathbf{T}_x \) generalizes \( \text{2.8} \) such that

\[ \mathcal{F} [(x(t))^n] = \underbrace{X(f) \ast X(f) \ast \cdots \ast X(f)}_{n \text{ times}} = \mathbf{T}_x^{n-1} \mathbf{X} . \]  

(2.10)
Now, consider a memoryless polynomial time-domain transfer function $y(t)$,

$$y(t) = b_0 + b_1 x(t) + b_2 [x(t)]^2 + \cdots + b_N [x(t)]^N \tag{2.11}$$

$$= \sum_{n=0}^{N} b_n [x(t)]^n \tag{2.12}$$

where the $b_n$ are generally real coefficients, but may be complex if the even order $b_n$ are zero.

Using the spectral vector representation so that $Y = \mathcal{F}[y(t)]$ and substituting (2.8–2.10) into the transform of (2.12), $y(t)$ can be expressed in the frequency domain as

$$Y = b_0 e_0 + b_1 X + b_2 T_x X + \cdots + b_N T_x^{N-1} X \tag{2.13}$$

$$= b_0 e_0 + \sum_{n=1}^{N} b_n T_x^{n-1} X, \tag{2.14}$$

where $b_0 e_0$ captures the constant (DC) part of the transfer function. Here, $e_0$ is the unit vector of the same length as the spectral vectors $X$ and $Y$ with 1 in the DC location and 0 elsewhere, i.e. if the spectrum is one-sided and comprised of only positive frequencies – as is the case with FREDA2– then

$$e_0 = [1 \ 0 \ 0 \ \cdots \ 0 \ 0]^T. \tag{2.15}$$

Chang [92] also described an extension to AOM that permitted the use of simplified Volterra series transfer functions having only order-dependent time delays, i.e. if $y(t)$ is given as

$$y(t) = b_0 + \sum_{n=1}^{N} b_n [x(t - \tau_n)]^n , \tag{2.16}$$

then the order-dependent time delays $\tau_n$ at each nonlinear order can be included by solving the memoryless case using AOM in the frequency domain and then post-multiplying by a diagonal matrix, $\Gamma_n$, comprised of entries $\Gamma_{kk} = e^{-j2\pi f_k \tau_n}$ equal to the delay in phasor form, where $f_k$ is the frequency of the $k^{th}$ spectral vector element, i.e.

$$Y = b_0 e_0 + \sum_{n=1}^{N} b_n T_x^{n-1} X \Gamma_n. \tag{2.17}$$

Chang’s implementation of AOM in FREDA2, based as it was on FREDA, was still rooted in solving systems of circuit equations governed by Kirchoff’s laws. FREDA2 also assumed that all input signals were real sinusoids so that conjugate symmetry applied, i.e. $X(-f) = X^*(f)$. 
*FREDA2* exploited this symmetry by limiting the spectral vector to DC and positive frequencies and then substituting the conjugate phasor form during the construction of spectrum transform matrices whenever instances of subtractive intermodulation occurred.

Chang’s work drew interest from researchers at the Universities of Warsaw and Rome, who cited his works in a 1994 conference presentation [94] of a frequency domain circuit simulator called Rupz. In 2000, researchers in Belgium [95] reported success in extracting parameters and implementing a Gallium Arsenide (GaAs) Pseudomorphic High Electon Mobility Transistor (PHEMT) model in rational polynomial form in *Rupz*. This rational polynomial method was investigated earlier [96] during the development of a similar model for a Metal Epitaxial Semiconductor Field Effect Transistor (MESFET) by some of the researchers from Rome involved in the creation of *Rupz*. While the rational polynomial implementation goes beyond Chang’s work in *FREDA2*, it does not appear that models based on transcendental functions have been implemented in *Rupz*.

At the 1997 European Microwave Conference [97] two researchers — Carvalho and Pedro, from the University of Aviero, Portugal — disclosed a frequency domain circuit simulator implementing a novel scheme for identifying the output frequency locations for sinusoidal input signals applied to a nonlinear device model. The novelty of the scheme was its computational simplicity compared to existing schemes; the authors claimed their scheme required only 90 computing loops to find all of the frequency locations for a 5th order system response to a 3-tone input while asserting that the comparable scheme of Chang [90] required 1331 loops. The researchers made a compelling case for their simulator by demonstrating the spectral regrowth of a 5th order nonlinear device to a 10-tone input signal. At a 1998 conference [98], results were furnished for the response of a CATV amplifier to a 32-tone signal using an innovative amplifier model based on Hermite polynomial approximation. At another conference [99] in 1998, they demonstrated the results of using two 11-tone frequency intervals with an ensemble average of 15 different phase regimes to predict the noise power ratio of a broadband amplifier. Despite touting their *index vector generating scheme* at these conferences, though, the mechanics of how it worked were comprehensively explained for the first time in [100]. The environment was later enhanced [3] so that each frequency index vector entry had corresponding entries in spectral vectors and separate power vectors. The spectral vector summed the incremental convolution products using phasor sums, while the power vector summed the squared magnitude of each incremental convolution product. It was thus possible to predict the total average power of
uncorrelated phase inputs using the enhanced environment, but the frequency indexing scheme precluded the separation of correlated and uncorrelated distortion.

Carvalho and Pedro’s index vector generating scheme may be described as follows: Let $N$ be the maximum order of nonlinearity in a system of interest, and let $Q$ be a number of equally-spaced, narrow-band input tones. The response of a nonlinear system to the input will be the creation of $N + 1$ distinct bands of tones roughly centered about DC and a center frequency at each order of nonlinearity. Designating the $n$ as one of the orders of nonlinearity, i.e. $n \in (1..N)$, then each band will have a center frequency $f_c$ defined by

$$f_c = \begin{cases} \left( \frac{f_m + f_{m+1}}{2} \right) \cdot n, & \text{for } Q \text{ even} \\ f_m \cdot n, & \text{for } Q \text{ odd}, \end{cases}$$

(2.18)

where $f_m$ is the center input frequency for $Q$ odd and $f_m$ and $f_{m+1}$ are the two center-most input frequencies when $Q$ is even. Define the following:

$$N_o = \max_{n \text{ odd}} [n \in (1..N)]$$

(2.19)

$$N_e = \max_{n \text{ even}} [n \in (1..N)].$$

(2.20)

Now, excluding the baseband area around DC for the moment, within each of the distinct bands, the number of frequencies that will be present is given by

$$Q_n = \begin{cases} N_e \cdot Q - (N_e - 1), & \text{for } n \text{ even} \\ N_o \cdot Q - (N_o - 1), & \text{for } n \text{ odd}. \end{cases}$$

(2.21)

The task of identifying the numerical value of each of the unique frequencies is thus reduced to knowing the center frequency, the number of frequencies in the band, and the frequency spacing. Carvalho and Pedro did not adequately address the what occurs in the baseband area, but it appears that $N$ frequencies will appear there, DC included. As an illustrative example, the case of $Q = 3$ input tones at frequencies of 1, 1.001, and 1.002 GHz (so $\Delta f = 0.001$ GHz) and a 5th order nonlinearity was given. $N_o$ is then the maximum odd $n$ or 5, and $N_e$ is the maximum even $n$, or 4. Table 2.2 on the following page shows the result.

While the scheme of Carvalho and Pedro was certainly a step toward right-sizing this computing task for the workstations of the mid-1990s (they noted doing their computing on Intel 486 platforms), it does have the unfortunate effect of enforcing a uniform frequency spacing on any problem to be investigated. Thus their simulator engine is unable to simulate problems having three truly incommensurate tones.
Table 2.2: Output Index Vector generated by method of Carvalho & Pedro

<table>
<thead>
<tr>
<th>Non-Linear Order</th>
<th>$n$</th>
<th>$\Delta f$</th>
<th>$\Delta f$</th>
<th>$\Delta f$</th>
<th>$\Delta f$</th>
<th>$\Delta f$</th>
<th>$\Delta f$</th>
<th>$\Delta f$</th>
<th>$\Delta f$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0.001</td>
<td>0.002</td>
<td>0.003</td>
<td>0.004</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.996</td>
<td>0.997</td>
<td>0.998</td>
<td>0.999</td>
<td>1.001</td>
<td>1.003</td>
<td>1.004</td>
<td>1.005</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1.998</td>
<td>1.999</td>
<td>2</td>
<td>2.001</td>
<td>2.002</td>
<td>2.003</td>
<td>2.004</td>
<td>2.005</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>2.998</td>
<td>2.999</td>
<td>3</td>
<td>3.001</td>
<td>3.002</td>
<td>3.003</td>
<td>3.004</td>
<td>3.005</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4</td>
<td>4.001</td>
<td>4.002</td>
<td>4.003</td>
<td>4.004</td>
<td>4.005</td>
<td>4.006</td>
<td>4.007</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>5</td>
<td>5.001</td>
<td>5.002</td>
<td>5.003</td>
<td>5.004</td>
<td>5.005</td>
<td>5.006</td>
<td>5.007</td>
</tr>
</tbody>
</table>

Most recently, researchers in Iran [101] modified a primitive prototype of the AOM Toolbox (developed in the present work) to successfully analyze the behavior of an antenna [102] with a nonlinear load. The solution to the problem used Newton iteration, which Chang [92] noted is streamlined when AOM is used because of the simplicity of Jacobian matrix construction for the solution of the system $f(X) = 0$. Specifically, if $Y$ is given by (2.14), then

$$\frac{\partial Y}{\partial X} = \sum_{n=1}^{N} b_n T_x^{n-1} I = \sum_{n=1}^{N} b_n T_x^{n-1},$$

(2.22)

where $I$ is the identity matrix.

2.6.4 Frequency Counting Simulators

The importance of frequency “accounting” schemes has only recently emerged in the realm of microwave circuit analysis, but they have been of interest for decades to engineering professionals in the field of satellite communications who are concerned with optimizing frequency assignments to minimize both adjacent channel and harmonic distortion, since satellites typically amplify hundreds of carriers over very broad bandwidths. Early work on automating this analysis problem at COMSAT produced two programming efforts [103], with the latter effort [104] able to operate on up to 110 carrier inputs and sort 3,000 intermodulation products. In the mid-1980s, researchers in Brazil made algorithmic simplifications to the combinatorics, limiting the intermodulation frequencies of interest to those “triple beat” combinations of three
or more distinct carriers with their Fast Counting Algorithm (FCA) \[105\]. For problems with similar goals, their method produced results in a 4 minutes whereas 7 hours were required for previous exhaustive “direct counting” methods. The FCA algorithm has been improved by other researchers, first by Nam and Powers \[106\], who achieved a 90% reduction in program execution time for larger problem sizes, and then later by Buruffa \[107\] to count higher orders of nonlinearity with a view to using the algorithm for wireless communications frequency planning, but with a few simplifying assumptions, such as equally-spaced and equally-modulated carriers.

### 2.6.5 Additional Developments

Two researchers – Närhi \[108\] and Palusinski \[109\] independently implemented frequency domain circuit simulators that were unusual in that they implemented models using only Chebychev polynomials.

Other publications not geared toward specific simulator environments deserve mention. A good review of the history of simulation is given in the journal articles by Gilmore and Steer \[110, 111\]. Two books that delve deeply into the methods and applications used in microwave circuit simulators are from Maas \[112\] and Rodrigues \[113\]. The theory of Volterra series analysis is treated authoritatively by Schetzen \[59\] and Rugh \[114\], and applications of Volterra series in microwave simulation can be found in the book by Pedro and Carvalho \[115\]. Behavioral modeling of nonlinear systems is covered broadly by Jeruchim, Balaban, and Shanmugan \[116\], and the mathematics of modeling nonlinear systems subjected to stochastics are described by Bendat \[117, 119\].

### 2.7 Concluding Remarks

This chapter has introduced the literature pertaining to nonlinear frequency-domain simulation. Due to a focus in this work on behavioral simulation, some depth of details on circuit simulation environments has been omitted. However, the literature review chapters of the dissertations of Rhyne \[86\] and Chang \[92\] contain far more detailed material and references on the theory, methods, and models of nonlinear frequency-domain circuit simulation.
Chapter 3

The AOM Toolbox Behavioral Modeling Environment

3.1 Introductory Remarks

This is a lengthy chapter that can be roughly divided into four blocks of content, with some blocks containing multiple sections. The Description in Depth block contains a level of detail and depth that is more appropriate for reference purposes after reviewing the entire work. Therefore, those reading this work for the first time may find it beneficial to skim this material or even skip over it, since the Illustrative Overview block furnishes most of the necessary introductory depth for proceeding to subsequent material. The blocks of content in this chapter are organized as follows:

Mathematical Foundation Section 3.2 and its subsections lay a firm mathematical foundation for the Arithmetic Operator Method (AOM). Subsections 3.2.1–3.2.3 review Euler’s formula, the sine over argument function, and the Dirac delta function. The concepts in these subsections are central to the development of the generalized functions used in the Fourier analysis that leads to the frequency-domain functional forms used by AOM. Subsections 3.2.4–3.2.10 show the Fourier transforms of the functions of interest in app-
Illustrative Overview Section 3.3 and its subsections provide an introductory overview of the elements and mechanics of AOM through an illustrative example using a two-tone input signal and a 2nd order non-linearity. The input signal form is given in Section 3.3.1 and the mathematics of convolving the signal with itself — with the resulting expansion of spectral content — are shown. The Vector Frequency Description (VFD) table is introduced in Section 3.3.2 and — by comparison with the results from Section 3.3.1 — it is shown how the construction of the table predicts the locations of output spectral content. The subsection ends by casting all of the output spectral vectors – DC, linear, and 2nd order response – as spectral vectors based upon the VFD table. Section 3.3.3 steps through the concept of spectrum mapping — which implements the mathematical equivalent of convolution — with the end result being a spectrum transform matrix which can be multiplied by the input spectral vector to produce the 2nd order output spectral vector. Section 3.3.4 extends the VFD table, spectral vectors, and spectrum transform matrix from a 2nd order nonlinearity to a 3rd order nonlinearity, illustrating how all three data structures grow in size.

Description in Depth Sections 3.4–3.7 describe the VFD, the spectral vector, the spectrum mapping table, and the spectrum transform matrix in great detail and include pseudocodes for the construction algorithms that are implemented in the AOM Toolbox. The subsection on the VFD table begins with the input signal, then casts it in spectral vector form using the VFD table unit vector descriptions. Convolution of the spectral vectors shows how the VFD table is created as a by-product, with extension to the identification of the VFD table as a set having properties akin to a vector space. The subsection on spectral vectors describes the three forms of spectral vectors that are possible and
contains important theorems on the correlation properties of spectral vector elements. The spectrum mapping subsection describes in detail how spectrum mapping determines all of the valid convolution operations necessary to cast a spectral vector in a matrix form that will implement convolution when multiplied by another spectral vector. Finally, the spectrum transform matrix subsection shows the construction of the spectrum transform matrix and reveals some important properties of the complex form of the matrix that were discovered in this work that facilitates Eigendecomposition solution methods.

**Other Considerations** Section 3.8 shows that the dynamic range available in the AOM Toolbox is unlimited and also illustrates the numerical floor inherent in the FFT, which is a critical ingredient in harmonic balance methods. Section 3.9 takes a view back at the results produced by Chang [92], describes the 1-sided spectrum mapping table and the 1-sided spectrum transform matrix and also furnishes additional insight into the mechanics of implementing transfer functions using exponential and hyperbolic tangent functions.

### 3.2 Mathematical Preliminaries

Although AOM has been used for almost twenty years by researchers from around the globe [90, 94, 95, 97, 102], to date the fundamental mathematics underpinning it has not been particularly well-described. One earlier publication by the author (the contents of which can be found in Appendix C) made a modest attempt to do this, but subsequent work in this area has revealed some shortcomings. First, the earlier publication was heavily focused on describing the algorithmic mechanics of AOM as implemented at the time and demonstrating how to apply those algorithms. Second, the previous work lacked precision in describing how the frequency-domain form of the input signals is arrived at from its corresponding time-domain form. The same observation can be made concerning the thesis of Chang [92], which took an axiomatic approach by beginning with the frequency-domain vector form. This chapter attempts to furnish
that unpublished mathematical rigor. This section reviews several mathematical properties and puts them in the precise form that will be used in the rigorous AOM development later in this chapter.

3.2.1 Euler’s Formula

This work will make use of Euler’s formula, \(e^{\jmath \theta} = \cos \theta + \jmath \sin \theta\), which is shown to be true through a simple Taylor series expansion [120],

\[
e^{\jmath \theta} = 1 + \frac{(\jmath \theta)^1}{1!} + \frac{(\jmath \theta)^2}{2!} + \frac{(\jmath \theta)^3}{3!} + \cdots
\]  

(3.1)

\[
= \left(1 - \frac{\theta^2}{2!} + \frac{\theta^4}{4!} - \cdots\right) + \jmath \left(\frac{\theta}{1!} - \frac{\theta^3}{3!} + \cdots\right)
\]  

(3.2)

\[
= \cos \theta + \jmath \sin \theta,
\]  

(3.3)

where, upon separation into the real and imaginary parts, the quantities inside the parentheses are recognized as the Taylor series expansions for \(\cos \theta\) and \(\sin \theta\).

An alternative form for deriving Euler’s equation [121] is rooted in the solution to the ordinary differential equation of the Stürm-Liouville [122] problem \(g''(\theta) + g(\theta) = 0\), for which the solution is known to be a sum of cosine and sine functions, i.e. \(g(\theta) = \alpha \cos \theta + \beta \sin \theta\), with the unknown coefficients determined by the initial values. Assuming a solution of the form \(g(\theta) = \exp(\jmath \theta)\), the coefficients are thus determined from the initial values of \(g\) and its first derivative:

\[
g(\theta) = e^{\jmath \theta} \Rightarrow g'(\theta) = \jmath e^{\jmath \theta} \Rightarrow g''(\theta) = -e^{\jmath \theta}
\]  

(3.4)

\[
g(\theta) = \alpha \cos \theta + \beta \sin \theta \Rightarrow g'(\theta) = -\alpha \sin \theta + \beta \cos \theta
\]  

(3.5)

Using initial conditions \(g(0) = \exp(\jmath 0) = 1\) and \(g'(0) = \jmath \exp(\jmath 0) = \jmath\),

\[
g(0) = \alpha \cos 0 + \beta \sin 0 = 1 \Rightarrow \alpha = 1
\]  

(3.6)

\[
g'(0) = -\alpha \sin 0 + \beta \cos 0 = \jmath \Rightarrow \beta = \jmath
\]  

(3.7)

\[
\therefore g(\theta) = e^{\jmath \theta} = \cos \theta + \jmath \sin \theta.
\]  

(3.8)

Using (3.1)-(3.3) and some algebraic manipulation leads to the following:

\[
\cos \theta = \frac{e^{\jmath \theta} + e^{-\jmath \theta}}{2}
\]  

(3.9)

\[
\sin \theta = \frac{e^{\jmath \theta} - e^{-\jmath \theta}}{2\jmath}
\]  

(3.10)
3.2.2 The Sine Over Argument Function

One function that frequently occurs in signal analysis is \( g(\xi) = \sin \frac{\xi}{\xi} \). This function has the following properties:

\[
\lim_{\xi \to 0} g(\xi) = 1 \quad (3.11)
\]

\[
\int_{-\infty}^{\infty} \frac{\sin(\alpha \xi)}{\xi} d\xi = \pi \quad \forall \alpha > 0 , \quad (3.12)
\]

where \( \alpha \) is a positive real parameter. Proof of (3.11) is via direct application of L'Hôpital's Rule \[123\],

\[
\lim_{\xi \to 0} g(\xi) = \lim_{\xi \to 0} \frac{\sin \xi}{\xi} = \lim_{\xi \to 0} \frac{d\sin \xi}{d\xi} \quad (3.13)
\]

\[
= \lim_{\xi \to 0} \frac{d\xi}{d\xi} \quad (3.14)
\]

\[
= \lim_{\xi \to 0} \cos \xi = 1 . \quad (3.15)
\]

The proof of (3.12) (which is also stated in \[124\]) is done through the use of theorems from the discipline of complex variables \[125\]. The proof sketched here follows that in \[126\] and begins with the observation that (3.12) is equal to the imaginary part of the result of integrating (3.12) along the real line. Letting \( z = x + jy \),

\[
\int_{-\infty}^{\infty} \frac{\sin(\alpha \xi)}{\xi} d\xi = \Im \left[ \int_{-\infty}^{\infty} \frac{\cos(\alpha x) + j \sin(\alpha x)}{x} dx \right] . \quad (3.16)
\]

\[
\int_{-\infty}^{\infty} \frac{\sin(\alpha \xi)}{\xi} d\xi = \Im \left[ \int_{-\infty}^{\infty} \frac{e^{j\alpha(x+jy)}}{x+jy} d(x+jy) \right] \quad (3.17)
\]

\[
= \Im \left[ \int_{-\infty}^{\infty} \frac{e^{j\alpha(x+j0)}}{x+j0} d(x+j0) \right] \quad (3.18)
\]

\[
= \Im \left[ \int_{-\infty}^{\infty} \frac{e^{j\alpha x}}{x} dx \right] \quad (3.19)
\]

\[
= \Im \left[ \int_{-\infty}^{\infty} \cos(\alpha x) + j \sin(\alpha x) \right] dx . \quad (3.20)
\]
Note that the real part of the integrand in (3.20) is an odd function, so the real part of that integral is zero. Now, consider the closed-path complex contour integral

$$\int_C \frac{e^{i\alpha z}}{z} \, dz = \int_{C_1} \frac{e^{i\alpha z}}{z} \, dz + \int_{C_2} \frac{e^{i\alpha z}}{z} \, dz + \int_{C_3} \frac{e^{i\alpha z}}{z} \, dz + \int_{C_4} \frac{e^{i\alpha z}}{z} \, dz = 0\,,$$  \hspace{1cm} (3.21)

that consists of the real-axis line integrals $C_1$ and $C_3$ as shown in Figure 3.1 and note that the sum of the integrals along $C_1$ and $C_3$ is equal to the integral in (3.20), assuming that the radius, $\rho$ along $C_2$ tends to zero. The closed-path contour integral in (3.21) is equal to 0, by virtue of Cauchy’s Integral Theorem [125] which states that every closed contour integral on an analytic function integrand must vanish. However, since the closed contour on the left is the sum of the line integrals along the open contours shown in Figure 3.1 this leads to

$$\lim_{R \to \infty} \int_{C_1} \frac{e^{i\alpha z}}{z} \, dz + \lim_{R \to \infty} \int_{C_3} \frac{e^{i\alpha z}}{z} \, dz = -\lim_{\rho \to 0} \int_{C_2} \frac{e^{i\alpha z}}{z} \, dz - \lim_{R \to \infty} \int_{C_4} \frac{e^{i\alpha z}}{z} \, dz\,,$$  \hspace{1cm} (3.22)

where the left side of (3.22) is now equal to the integral in (3.20). Thus the solution is reduced to evaluating the open contour line integrals along $C_2$ and $C_4$. These integrals are best evaluated using polar coordinates.

Consider the integral along the contour $C_4$ first. Let $z = Re^{i\theta}$, where $R$ is the magnitude of $z$ and $\theta$ is its polar angle. Then by virtue of the geometry of $C_4$, $dz = jRe^{i\theta} \, d\theta$,
Thus, as squeeze-play (3.24), the mathematician’s Collecting the results of (3.26)-(3.27) and substituting them into (3.25) (and considering (3.23).

The result in (3.26) is known as Jordan’s Lemma. At this point, (3.25) is considered along the two intervals 0 ≤ θ ≤ π/2 and π/2 ≤ θ ≤ π. In the first interval, note that sin θ ≥ 2θ/π, so that \( e^{-R \sin \theta} \leq e^{-2R/\pi} \). Thus,

\[
\lim_{R \to \infty} \left| \int_0^{\pi/2} e^{\alpha R \cos \theta} e^{-\alpha R \sin \theta} d\theta \right| \leq \lim_{R \to \infty} \left| \int_0^{\pi/2} e^{\alpha R \cos \theta} e^{-\alpha R \sin \theta} d\theta \right| \leq \lim_{R \to \infty} \left| \int_0^{\pi/2} e^{-2\alpha R/\pi} d\theta \right| = \lim_{R \to \infty} \frac{\pi}{2\alpha R} \left(1 - e^{-\alpha R} \right) .
\]

The result in (3.26) is known as Jordan’s Lemma. Now, noting that \( \sin \theta = \sin(\pi - \theta) \) and then making the substitutions \( \psi = \pi - \theta \) and \( d\psi = -d\theta \), in the second interval,

\[
\lim_{R \to \infty} \int_{\pi/2}^{\pi} e^{-\alpha R \sin \theta} d\theta = \lim_{R \to \infty} \int_{\pi/2}^{\pi/2} e^{-\alpha R \sin(\pi - \theta)} d\theta = \lim_{R \to \infty} \int_0^{\pi/2} e^{-\alpha R \sin \psi} (-d\psi)
\]

\[
= \lim_{R \to \infty} \int_0^{\pi/2} e^{-\alpha R \sin \psi} d\psi \leq \lim_{R \to \infty} \int_0^{\pi/2} e^{-2\alpha R/\pi} d\psi = \lim_{R \to \infty} \frac{\pi}{2\alpha R} \left(1 - e^{-\alpha R} \right) .
\]

Collecting the results of (3.26)-(3.27) and substituting them into (3.25) (and considering (3.23) - (3.24), the mathematician’s squeeze-play results:

\[
\lim_{R \to \infty} \int_{C_4} \frac{e^{jz}}{z} dz \leq \lim_{R \to \infty} \frac{\pi}{\alpha R} \left(1 - e^{-\alpha R} \right) \to 0 .
\]

Thus, as \( R \to \infty \), the integral over the line contour \( C_4 \) tends to zero.

Now consider the integral along the contour \( C_2 \). Let \( z = \rho e^{j\theta} \), where \( \rho \) is the magnitude of \( z \) and \( \theta \) is its polar angle. Then by virtue of the geometry of \( C_2 \), \( dz = j\rho e^{j\theta} d\theta \), so

\[
\lim_{\rho \to 0} \int_{C_2} \frac{e^{jz}}{z} dz = \lim_{\rho \to 0} \int_0^\pi \frac{e^{j\rho e^{j\theta}}}{\rho e^{j\theta}} j\rho e^{j\theta} d\theta = \lim_{\rho \to 0} \int_0^\pi e^{j\rho e^{j\theta}} d\theta
\]

\[
= \lim_{\rho \to 0} \int_0^\pi e^{j\rho(\cos \theta + j\sin \theta)} d\theta = \int_0^\pi e^{j\rho(\cos \theta + j\sin \theta)} d\theta = \int_0^\pi 1 d\theta
\]

\[
= j\pi .
\]

(3.31)
Note that the result in (3.31) is valid because \( z = 0 \) is an isolated singularity, so the integrand is analytic in the limit.

Collecting the results of (3.28) and (3.31) and substituting them into (3.22),

\[
\lim_{R \to \infty} \int_{C_1} \frac{e^{j\alpha z}}{z} dz + \lim_{\rho \to 0} \int_{C_3} \frac{e^{j\alpha z}}{z} dz = j\pi .
\]

Finally, since the result of (3.32) is equal to the integral in (3.20), this leads to the final result:

\[
\int_{-\infty}^{\infty} \sin \alpha \xi d\xi = \Im \left[ \int_{-\infty}^{\infty} \frac{\cos \alpha x + j \sin \alpha x}{x} dx \right] = \Im [j\pi] = \pi \quad \forall \alpha > 0 .
\]

### 3.2.3 The Dirac Delta Function

The Dirac delta function originated in the field of quantum mechanics in 1927 \([127] \), but the underlying mathematics of the delta function and the broader category of generalized functions was found to be useful in other branches of mathematics \([128, 129] \) and the physical sciences \([130, 131] \). In the discipline of Fourier Analysis, the concept of generalized functions was used to analytically develop the Fourier Transform for a DC signal and other time-domain functions which had not previously had Fourier Transforms \([132] \). A more modern reference that covers generalized functions and Fourier analysis is \([133] \). The essential notion of a generalized function is that its exact form may be specified in many ways – often with a parameter leading to a limiting form \([134] \) – but that certain properties are always fixed. In the case of the Dirac delta function \( \delta(\xi) \), the following properties hold:

\[
\delta(\xi) = 0 \quad \forall \xi \neq 0
\]

\[
\delta(\xi) \to \infty \quad \text{for } \xi = 0
\]

\[
\int_{-\infty}^{\infty} \delta(\xi) d\xi = 1
\]

The intuitive explanation for (3.34)–(3.36) is that \( \delta(\xi) \) is a function with most of its non-zero content concentrated around the origin, but which under all circumstances has unit area. The delta function also allows for movement of its non-zero content through the use of offsets to its argument. For example, the function \( \delta(\xi - \chi) \) is zero everywhere except at \( \xi = \chi \).

One other property of the delta function is the sifting property. This property isolates the value of any well-behaved function \( g(\xi) \) at the zero-argument point of the delta function,
There are many mathematical functions meeting the requirements of (3.34)–(3.36). A number of them are noted in [135, 136] and include cases of functions having continuous derivatives. For the purposes of the work here, two generalized functions describing the delta function are of interest. The first is in the form of a simple rectangular function [137]. Begin with the following definition of the rectangular function:

\[ \Pi \left( \frac{\xi}{\beta} \right) = \begin{cases} 1 & \text{for } |\xi| < \beta/2, \\ 0 & \text{else}. \end{cases} \]  

(3.39)

Now, with the \( \Pi \) function defined as in (3.39), the delta function can be described as follows:

\[ \delta(\xi) = \lim_{\beta \to 0} \beta^{-1} \Pi \left( \frac{\xi}{\beta} \right). \]  

(3.40)

Note that this definition meets the requirements of (3.34)–(3.36). The second generalized function of interest for use as \( \delta(\xi) \) is in the form of a limit on the sine over argument function given by [138]

\[ \delta(\xi) = \lim_{\kappa \to \infty} \frac{\sin \kappa \pi \xi}{\kappa \pi \xi}, \]  

(3.41)

or, equivalently,

\[ \delta(\xi) = \lim_{\kappa \to \infty} \frac{\sin \kappa \pi \xi}{\kappa \pi}. \]  

(3.42)

Although these forms are algebraically equivalent, in the second form (3.42) it is quite easily seen that this form has a sine over argument part that is 1 when \( \xi \) is 0 and a parametric magnitude, \( \kappa \), that tends to infinity. At the same time, in the first form of (3.41), the argument to the sine function is equivalent to that in (3.33) by simply noting that \( \alpha = \kappa \pi \). Note that the integral over the real line of either of (3.41) or (3.42) will be equal to 1 due to the result of (3.33) in Section 3.2.2 and the presence of \( \pi \) in the denominator of (3.41).
3.2.4 Fourier Transform Definition

When dealing with frequency-domain quantities, microwave and communications engineers have a preference for the use of Hertz (i.e., cycles per second) rather than radians per second (rad/sec) for the frequency-domain variable. Specialists in the discipline of Signal Processing and Analysis tend to use radians per second, and since most of the training materials and reference literature is written by them [88, 139], usually engineers are introduced to Fourier Transforms in the form using radians per second. Throughout this work, Fourier Transforms and Fourier Transform pairs will be given using the Hertzian [140] form rather than the Radial form. One reference that is notable for its use of Hertzian Fourier Transforms is [141]. However, none of the transforms involving $\delta$ functions are developed using generalized functions in that work, so it is done here for completeness. Some of the transforms developed in Hertzian form here are given in Radial form in [134].

The Hertzian Fourier Transform of a time-domain function $x(t)$ is defined as

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi ft} dt,$$  \hspace{1cm} (3.43)

where $f$ is the frequency-domain variable with the units of Hertz (or cycles per second) and $j$ is the imaginary number. The corresponding inverse Fourier Transform to (3.43) is

$$x(t) = \int_{-\infty}^{\infty} X(f)e^{j2\pi ft} dt.$$  \hspace{1cm} (3.44)

The time and frequency-domain functions that form a Fourier Transform pair will be denoted $x(t) \leftrightarrow_X X(f)$. In subsequent equations in this work, a short-hand notation for (3.43) will be given by the expression $X(f) = \mathfrak{F}[x(t)]$, which may be followed by additional expressions. Similarly, a short-hand notation for (3.44) will be given by the expression $x(t) = \mathfrak{F}^{-1}[X(f)]$.

3.2.5 Fourier Transform of a DC Signal

The Fourier transform of the function $g(t) = 1$ does not exist. However, if the constant is redefined as a generalized function with the assistance of the $\Pi$ function, the limiting value of the transform does exist. Thus let

$$g(t) = \lim_{\tau \to \infty} \Pi \left( \frac{t}{\tau} \right).$$  \hspace{1cm} (3.45)
Substituting $g(t)$ into (3.43),

$$G(f) = \lim_{\tau \to \infty} \int_{-\infty}^{\infty} \Pi \left( \frac{t}{\tau} \right) e^{-j2\pi ft} dt$$

(3.46)

$$= \lim_{\tau \to \infty} \int_{-\tau/2}^{\tau/2} e^{-j2\pi ft} dt = \lim_{\tau \to \infty} e^{-j2\pi f \tau} \left|_{-\tau/2}^{\tau/2} \right.$$ (3.47)

$$= \lim_{\tau \to \infty} \frac{-2j \sin \pi f \tau}{-j2\pi f} = \lim_{\tau \to \infty} \frac{\sin \pi f \tau}{\pi f} $$

(3.48)

$$= \lim_{\tau \to \infty} \tau \frac{\sin \pi f \tau}{\pi f \tau} $$

(3.49)

$$= \delta(f). $$

(3.50)

Thus $\delta(f) = \mathcal{F}[1]$. Note that if $G(f) = \delta(f)$ is substituted into (3.44), then the sifting property of the delta function results in the integral trivially evaluating to 1. Being a bit more formal, substitute the generalized functional form of $G(f)$,

$$G(f) = \delta(f) = \lim_{\nu \to 0} \nu^{-1} \Pi \left( \frac{f}{\nu} \right), $$

(3.51)

into (3.44):

$$g(t) = \lim_{\nu \to 0} \nu^{-1} \int_{-\infty}^{\infty} \Pi \left( \frac{f}{\nu} \right) e^{i2\pi ft} df $$

(3.52)

$$= \lim_{\nu \to 0} \nu^{-1} \int_{-\nu/2}^{\nu/2} e^{i2\pi ft} df = \lim_{\nu \to 0} \nu^{-1} e^{i2\pi f \nu} \left|_{-\nu/2}^{\nu/2} \right.$$ (3.53)

$$= \lim_{\nu \to 0} \nu^{-1} \frac{-2j \sin \pi t \nu}{j2\pi t} = \lim_{\nu \to 0} \nu^{-1} \frac{\sin \pi t \nu}{\pi t} $$

(3.54)

$$= \lim_{\nu \to 0} \frac{\sin \pi t \nu}{\pi t \nu} = \lim_{\nu \to 0} \frac{\nu}{\nu} \frac{\sin \pi t \nu}{\pi t} $$

(3.55)

$$= 1, $$

(3.56)

where in (3.55) L'Hôpital’s Rule has been used. Thus $\mathcal{F}^{-1}[\delta(f)] = 1$ and so $1 \iff \delta(f)$.

Physically, this Fourier transform pair states that a constant DC signal in the time-domain maps to an impulse at 0 Hz in the frequency-domain, and vice versa.
3.2.6 Fourier Transform of an Impulse

Impulse signals in the time domain are usually cast in the form of a delta function, \( g(t) = \delta(t) \). Using the generalized function form for \( \delta(t) \) in the time domain,

\[
g(t) = \delta(t) = \lim_{\tau \to 0} \tau^{-1} \Pi \left( \frac{t}{\tau} \right),
\]

the Fourier transform of \( \delta(t) \) is given by

\[
G(f) = \lim_{\tau \to 0} \tau^{-1} \int_{-\infty}^{\infty} \Pi \left( \frac{t}{\tau} \right) e^{-j2\pi ft} dt
\]

\[
= \lim_{\tau \to 0} \tau^{-1} \int_{-\tau/2}^{\tau/2} e^{-j2\pi ft} dt = \lim_{\tau \to 0} \tau^{-1} \left[ \frac{e^{-j2\pi ft}}{-j2\pi f} \right]_{-\tau/2}^{\tau/2}
\]

\[
= \lim_{\tau \to 0} \tau^{-1} \frac{j2 \sin \pi f \tau}{-j2\pi f} = \lim_{\tau \to 0} \tau^{-1} \frac{\sin \pi f \tau}{\pi f}
\]

\[
= \lim_{\tau \to 0} \frac{\sin \pi f \tau}{\pi f \tau} = \lim_{\tau \to 0} \frac{\frac{d}{d \tau} \sin \pi f \tau}{\pi f} = \lim_{\tau \to 0} \frac{\pi f \cos \pi f \tau}{\pi f}
\]

\[
= 1.
\]

where in (3.61) L’Hôpital’s Rule has again been used. Thus \( \mathfrak{F}[\delta(t)] = 1 \). Showing the reverse, i.e. that \( \mathfrak{F}^{-1}[1] = \delta(t) \), requires casting the frequency-domain constant as a generalized function,

\[
G(f) = 1 = \lim_{\nu \to \infty} \Pi \left( \frac{f}{\nu} \right).
\]

Substituting (3.63) into (3.44),

\[
g(t) = \lim_{\nu \to \infty} \int_{-\nu/2}^{\nu/2} \Pi \left( \frac{f}{\nu} \right) e^{j2\pi ft} df
\]

\[
= \lim_{\nu \to \infty} \int_{-\nu/2}^{\nu/2} e^{j2\pi ft} df = \lim_{\nu \to \infty} e^{j2\pi ft} \left[ \frac{1}{j2\pi t} \right]_{-\nu/2}^{\nu/2}
\]

\[
= \lim_{\nu \to \infty} \frac{j2 \sin \pi t \nu}{j2\pi t} = \lim_{\nu \to \infty} \frac{\sin \pi t \nu}{\pi t}
\]

\[
= \lim_{\nu \to \infty} \frac{\sin \pi t \nu}{\pi t \nu} = \delta(t),
\]

Thus \( \mathfrak{F}^{-1}[1] = \delta(t) \), and so \( \delta(t) \stackrel{\mathfrak{F}}{\leftrightarrow} 1 \). Physically, this Fourier transform pair states that an impulsive signal in the time-domain has a constant or white noise Fourier transform.
3.2.7 Fourier Transform of a Complex Exponential  

A complex exponential function at a given frequency \( f_0 \), \( g(t) = e^{2\pi f_0 t} \), does not have a Fourier Transform unless generalized functions are used. Thus let

\[
g(t) = \lim_{\tau \to \infty} e^{2\pi f_0 t} \Pi \left( \frac{t}{\tau} \right). \tag{3.69}
\]

Then,

\[
G(f) = \lim_{\tau \to \infty} \int_{-\infty}^{\infty} e^{2\pi f_0 t} \Pi \left( \frac{t}{\tau} \right) e^{-j2\pi ft} dt
\]

\[
= \lim_{\tau \to \infty} \int_{-\tau/2}^{\tau/2} e^{-j2\pi (f-f_0)t} dt = \lim_{\tau \to \infty} \frac{e^{-j2\pi (f-f_0)t}}{-j2\pi (f-f_0)} \bigg|_{-\tau/2}^{\tau/2} \tag{3.70}
\]

\[
= \lim_{\tau \to \infty} \frac{2\sin \pi (f-f_0)\tau}{\pi (f-f_0)\tau} = \lim_{\tau \to \infty} \frac{\sin \pi (f-f_0)\tau}{\pi (f-f_0)\tau} \tag{3.71}
\]

\[
= \lim_{\tau \to \infty} \frac{\sin \pi (f-f_0)\tau}{\pi (f-f_0)\tau} \tag{3.72}
\]

\[
= \delta(f - f_0). \tag{3.73}
\]

Thus \( \hat{g}[e^{2\pi f_0 t}] = \delta(f - f_0) \). Showing the reverse also requires use of generalized functions, so substituting

\[
G(f) = \delta(f - f_0) = \lim_{\nu \to 0} \nu^{-1} \Pi \left( \frac{f - f_0}{\nu} \right), \tag{3.75}
\]

into (3.41):

\[
g(t) = \lim_{\nu \to 0} \nu^{-1} \int_{-\infty}^{\infty} \Pi \left( \frac{f - f_0}{\nu} \right) e^{2\pi f t} df \tag{3.76}
\]

\[
= \lim_{\nu \to 0} \nu^{-1} \int_{f_0-\nu/2}^{f_0+\nu/2} e^{2\pi f t} df = \lim_{\nu \to 0} \nu^{-1} \frac{e^{2\pi f_0 t} e^{2\pi (f_0/2)^2}}{2\pi t} \bigg|_{f_0-\nu/2}^{f_0+\nu/2} \tag{3.77}
\]

\[
= \lim_{\nu \to 0} \nu^{-1} \frac{e^{2\pi f_0 t} e^{2\pi (f_0+\nu/2) t} - e^{2\pi f_0 t} e^{2\pi (f_0-\nu/2) t}}{j2\pi t} \tag{3.78}
\]

\[
= e^{j2\pi f_0 t} \lim_{\nu \to 0} \nu^{-1} \frac{2j \sin \pi t \nu}{j2\pi t} = e^{j2\pi f_0 t} \lim_{\nu \to 0} \nu^{-1} \frac{\sin \pi t \nu}{\pi t} \tag{3.79}
\]

\[
= e^{j2\pi f_0 t} \lim_{\nu \to 0} \frac{\sin \pi t \nu}{\tau t} = e^{j2\pi f_0 t} \lim_{\nu \to 0} \frac{\frac{df}{d\nu} \sin \pi t \nu}{d\nu \pi t \nu} \tag{3.80}
\]

\[
= e^{j2\pi f_0 t} \lim_{\nu \to 0} \frac{\pi t \cos \pi t \nu}{\pi t} \tag{3.81}
\]

\[
= e^{j2\pi f_0 t}. \tag{3.82}
\]
Thus \( \mathcal{F}^{-1} [\delta(f - f_0)] = e^{j2\pi f_0 t} \), and so \( e^{j2\pi f_0 t} \xrightarrow{\mathcal{F}} \delta(f - f_0) \). Complex exponentials do not occur in nature, but conjugate pairs of complex exponentials form real cosine and sine signals, so the Fourier transform pair here states that a single complex exponential oscillating at a fixed frequency \( f_0 \) has a Fourier transform that appears as an impulse in the frequency-domain.

### 3.2.8 Fourier Transform of Sinusoidal Functions

The Fourier transforms of cosine and sine functions follow from \((3.9)\)–\((3.10)\) and the results of Section 3.2.7. For cosines,

\[
g(t) = \cos 2\pi f_0 t = \frac{e^{j2\pi f_0 t} + e^{-j2\pi f_0 t}}{2},
\]

so,

\[
\mathcal{F}[g(t)] = \frac{\delta(f - f_0)}{2} + \frac{\delta(f + f_0)}{2}.
\]

For sines,

\[
h(t) = \sin 2\pi f_0 t = \frac{e^{j2\pi f_0 t} - e^{-j2\pi f_0 t}}{2j},
\]

so,

\[
\mathcal{F}[h(t)] = \frac{\delta(f - f_0)}{2j} - \frac{\delta(f + f_0)}{2j}.
\]

The inverse transforms follow by using the results of Section 3.2.7. Thus,

\[
\cos 2\pi f_0 t \xrightarrow{\mathcal{F}} \frac{\delta(f - f_0) + \delta(f + f_0)}{2},
\]

and

\[
\sin 2\pi f_0 t \xrightarrow{\mathcal{F}} \frac{\delta(f - f_0) - \delta(f + f_0)}{2j}.
\]

Note that if a fixed phase angle appears as an argument to either a cosine or sine function, then a fixed phase rotation is factored out of the Fourier transform. For cosines,

\[
g(t) = \cos(2\pi f_0 t + \phi) = \frac{e^{j2\pi f_0 t + j\phi} + e^{-j2\pi f_0 t - j\phi}}{2} = \frac{e^{j\phi} e^{j2\pi f_0 t} + e^{-j\phi} e^{-j2\pi f_0 t}}{2},
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]
so,
\[
\mathcal{F} [g(t)] = \frac{e^{j\phi} \mathcal{F} [e^{j2\pi f_0 t}] + e^{-j\phi} \mathcal{F} [e^{-j2\pi f_0 t}]}{2} \quad (3.92)
\]
\[
= \frac{e^{j\phi} \delta(f - f_0) + e^{-j\phi} \delta(f + f_0)}{2} . \quad (3.93)
\]

For sines,
\[
h(t) = \sin(2\pi f_0 t + \phi) \quad (3.94)
\]
\[
= \frac{e^{j2\pi f_0 t + j\phi} - e^{-j2\pi f_0 t - j\phi}}{2j} \quad (3.95)
\]
\[
= \frac{e^{j\phi} e^{j2\pi f_0 t} - e^{-j\phi} e^{-j2\pi f_0 t}}{2j} , \quad (3.96)
\]
so,
\[
\mathcal{F} [h(t)] = \frac{e^{j\phi} \mathcal{F} [e^{j2\pi f_0 t}] - e^{-j\phi} \mathcal{F} [e^{-j2\pi f_0 t}]}{2j} \quad (3.97)
\]
\[
= \frac{e^{j\phi} \delta(f - f_0) - e^{-j\phi} \delta(f + f_0)}{2j} . \quad (3.98)
\]

Physically, these Fourier transform pairs relate purely real cosine and sine functions at a particular frequency \(f_0\) to pairs of delta impulses in the frequency-domain. Time-domain cosines transform to two identical delta impulses at equal and opposite frequencies \(\pm f_0\), and sines transform to two conjugate-symmetric impulses at equal and opposite frequencies \(\pm f_0\).

### 3.2.9 Fourier Transform of a Time-Domain Product of Functions

The Fourier transform of time-domain product of two functions, say \(g(t)h(t)\), is the convolution of their transforms in the frequency domain. Noting again that \(*\) denotes the convolution operation, this is illustrated by simply finding the inverse Fourier transform of \(G(f) \ast H(f)\):
\[
\mathcal{F}^{-1} [G(f) \ast H(f)] = \int_{-\infty}^{\infty} G(f) \ast H(f) e^{j2\pi ft} df
\]
\[
= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} G(\nu)H(f - \nu) d\nu e^{j2\pi ft} df . \quad (3.99)
\]

Interchanging the order of integration,
\[
\mathcal{F}^{-1} [G(f) \ast H(f)] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} G(\nu)H(f - \nu) e^{j2\pi ft} df d\nu
\]
\[
= \int_{-\infty}^{\infty} G(\nu) \int_{-\infty}^{\infty} H(f - \nu) e^{j2\pi (f-\nu)t} df e^{j2\pi \nu t} d\nu . \quad (3.100)
\]
Substituting $v = f - \nu$, so $dv = df$,

$$
\mathcal{F}^{-1} [G(f) \ast H(f)] = \int_{-\infty}^{\infty} G(\nu) \int_{-\infty}^{\infty} H(v)e^{j2\pi vt} dv e^{j2\pi \nu t} d\nu
$$

(3.103)

$$
= \left[ \int_{-\infty}^{\infty} G(\nu)e^{j2\pi \nu t} d\nu \right] \left[ \int_{-\infty}^{\infty} H(v)e^{j2\pi \nu t} d\nu \right]
$$

(3.104)

$$
= g(t)h(t)
$$

(3.105)

and thus it can be seen that the frequency-domain convolution of two functions corresponds to the product of their time-domain forms. Note that if $g(t) = h(t)$, then (3.105) shows that $\mathcal{F}^{-1} [G(f) \ast G(f)] = g(t)g(t) = [g(t)]^2$. Through induction, it can be shown that if $G(f)$ is convolved with itself $n$ times, then

$$
\mathcal{F}^{-1} \left[ \underbrace{G(f) \ast G(f) \ast \cdots \ast G(f)}_{n \text{ times}} \right] = [g(t)]^n,
$$

(3.106)

and so repeated frequency domain convolution of a function with itself corresponds to raising its time-domain representation to the same power.

### 3.2.10 Fourier Transform of a Time-delayed Function

Suppose $h(t)$ is a function of $g(t)$ with a fixed time delay $\tau$, i.e. $h(t) = g(t - \tau)$. Then the Fourier transform of $h(t)$ is given by

$$
H(f) = \int_{-\infty}^{\infty} h(t)e^{-j2\pi ft} dt = \int_{-\infty}^{\infty} g(t - \tau)e^{-j2\pi ft} dt
$$

(3.107)

Substituting $\gamma = t - \tau$ or $t = \gamma + \tau$,

$$
H(f) = \int_{-\infty}^{\infty} g(\gamma)e^{-j2\pi f\gamma} e^{-j2\pi f\tau} d\gamma = e^{-j2\pi f\tau} \int_{-\infty}^{\infty} g(\gamma)e^{-j2\pi f\gamma} d\gamma
$$

(3.108)

$$
= e^{-j2\pi f\tau} G(f)
$$

(3.109)

Substituting the result into (3.104) gives the original time-delayed function. Thus, $g(t - \tau) \leftrightarrow e^{-j2\pi f\tau} G(f)$.

Note that if $h(t)$ is in the form of a time-delayed function raised to a power, i.e. $h(t) = [g(t - \tau)]^n$, then (3.108) dictates that the time delay phasor that is factored out of the
transform is not raised to the power of $n$. Thus,

$$H(f) = \tilde{\delta}[g(t - \tau)^n]$$  \hspace{1cm} (3.110)

$$= e^{-j2\pi f\tau} \tilde{\delta}[g(t)^n]$$ \hspace{1cm} (3.111)

$$= e^{-j2\pi f\tau} \left(G(f) \ast G(f) \ast \cdots \ast G(f) \right) \text{ n times} \hspace{1cm} (3.112)$$

This Fourier transform pair shows that when a given function is delayed by a fixed time delay, then its Fourier transform is the same as the transform absent a time delay which is then multiplied by a frequency-dependent fixed phasor offset.
3.3 Illustrative Overview of the Arithmetic Operator Method

In Section 2.6.3 it was noted that AOM performs convolution in the frequency domain. One area where implementations of AOM have differed has been in the functional relationship of the spectral vector elements to the independent frequency domain variable. Chang [92] used a user-selected set of objective frequencies, i.e. a set of user-chosen numerical frequencies, while Carvalho and Pedro chose uniformly incremented input frequencies leading to sets of uniformly incremented linear and nonlinear spectra. For this work, a new frequency domain construct rooted in a vector space has been developed which has some desirable statistical properties that will be discussed in Section 3.5.5. Two illustrative examples in this section will demonstrate how this frequency domain construct, termed the Vector Frequency Description (VFD), rises naturally from the applied mathematics used to formulate the nonlinear analysis problem in the frequency domain. Some of the properties of AOM will be informally observed here when the VFD is used as the underlying functional basis for describing the independent frequency variables. More formality will appear in subsequent sections. The mechanics of implementing AOM using the VFD will also be illustrated. These mechanics will be precisely cast into algorithms in subsequent sections.

3.3.1 Two-Tone Signals and Polynomial Nonlinear Transfer Function

Consider a two-tone sinusoidal input signal,

\[ x(t) = \sum_{q=1}^{2} A_q \cos(2\pi f_q t + \phi_q) \]

\[ = \sum_{q=1}^{2} A_q \frac{e^{j2\pi f_q t + j\phi_q}}{2} + \sum_{q=1}^{2} A_q \frac{e^{-j2\pi f_q t - j\phi_q}}{2} , \quad \text{(3.113)} \]

where \( A_q \) is the amplitude, \( f_q \) the frequency (and \( f_2 > f_1 \)), and \( \phi_q \) the phase of tone \( q \). Defining \( f_{-q} = -f_q \) and \( \phi_{-q} = -\phi_q \) (and with \( A_q = A_{-q} \)), the expression in (3.113) can be
written compactly as

\[
x(t) = \sum_{q=1}^{2} A_q \frac{e^{j2\pi f_q t + j\phi_q}}{2} + \sum_{q=1}^{2} A_{-q} \frac{e^{j2\pi f_{-q} t + j\phi_{-q}}}{2}
\]

\[
= \sum_{q=1}^{2} A_q \frac{e^{j2\pi f_q t + j\phi_q}}{2} + \sum_{q=-2}^{-1} A_q \frac{e^{j2\pi f_q t + j\phi_q}}{2}
\]

\[
= \sum_{q=-2}^{2} A_q \frac{e^{j2\pi f_q t + j\phi_q}}{2} .
\]

(3.114)

From (3.74) on page 39 in Section 3.2.7, the Fourier transform of (3.114) is given by

\[
X(f) = \frac{1}{2} \sum_{q=-2}^{2} A_q e^{j\phi_q} \delta(f - f_q)
\]

(3.115)

\[
= \frac{1}{2} \sum_{q=-2}^{2} X_q \delta(f - f_q) ,
\]

(3.116)

where \(X_q = A_q e^{j\phi_q}\). Now let \(y(t)\) be a memoryless transfer function of the form of (2.12),

\[
y(t) = b_0 + b_1 x(t) + b_2 [x(t)]^2 + \cdots + b_N [x(t)]^N
\]

\[
= y_0(t) + y_1(t) + y_2(t) + \cdots + y_N(t)
\]

\[
= \sum_{n=0}^{N} b_n [x(t)]^n .
\]

(3.117)

From (3.106) on page 42 in Section 3.2.9, the Fourier transform of (3.117) is given by

\[
Y(f) = b_0 \delta(f) + b_1 X(f) + b_2 X(f) \ast X(f) +
\]

\[
= b_3 X(f) \ast X(f) \ast X(f) + \cdots +
\]

\[
b_N \left( X(f) \ast X(f) \ast \cdots \ast X(f) \right)_{N \text{ times}}
\]

\[
= Y_0(f) + Y_1(f) + Y_2(f) + \cdots + Y_N(f) ,
\]

(3.118)

(3.119)

where \(Y_0(f) = b_0 \delta(f)\) and \(Y_1(f) = b_1 X(f)\). Consider now \(Y_2(f)\). To compute \(Y_2(f)\), \(X(f)\) must be convolved with itself. Define \(G(f)\) and \(H(f)\) as equivalent to \(X(f)\) but with differing
dummy index variables:

\[ H(f) = \frac{1}{2} \sum_{q_1=-2}^{2} X_{q_1} \delta(f - f_{q_1}) \]  
\[ G(f) = \frac{1}{2} \sum_{q_2=-2}^{2} X_{q_2} \delta(f - f_{q_2}) . \]

Now, using (3.100) on page 41, which defines the convolution operation,

\[ X(f) \ast X(f) = G(f) \ast H(f) \]
\[ = \int_{-\infty}^{\infty} G(\nu)H(f - \nu)d\nu \]
\[ = \int_{-\infty}^{\infty} \left[ \frac{1}{2} \sum_{q_2=-2}^{2} A_{q_2} e^{\phi_{q_2} \nu} \delta(\nu - f_{q_2}) \right] \cdot \left[ \frac{1}{2} \sum_{q_1=-2}^{2} A_{q_1} e^{\phi_{q_1} \nu} \delta(f - f_{q_1} - \nu) \right] d\nu \]
\[ = \frac{1}{2^2} \sum_{q_2=-2}^{2} \sum_{q_1=-2}^{2} A_{q_2} A_{q_1} e^{\bar{X}(\phi_{q_2} + \phi_{q_1})} \int_{-\infty}^{\infty} \delta(\nu - f_{q_2}) \delta(f - f_{q_1} - \nu) d\nu \]
\[ = \frac{1}{2^2} \sum_{q_2=-2}^{2} \sum_{q_1=-2}^{2} A_{q_2} A_{q_1} e^{\bar{X}(\phi_{q_2} + \phi_{q_1})} \delta(f - f_{q_2} - f_{q_1}) , \]

where the summations and the integral have been reversed in order and the sifting property of the \( \delta \) function described in Section 3.2.3 has been used to obtain the result form of the \( \delta \) function in (3.126). Notice that it makes no difference which \( \delta \) function in (3.125) is used for sifting; the form of (3.126) is the result in both cases. Expanding the double summations to eliminate the dummy index variables, substituting \(-f_q\) for \( f_{-q}\) (and \(-\phi_q\) for \( \phi_{-q}\)), respectively,
and simplifying the resulting expression leads to

\[
X(f) \ast X(f) = \begin{cases}
\frac{1}{4} A_2^2 e^{-j2\phi_2} \delta(f + 2f_2) & \text{(at } -2f_2) \\
\frac{1}{2} A_1 A_2 e^{-j(\phi_1 + \phi_2)} \delta(f + f_1 + f_2) & \text{(at } -f_1 - f_2) \\
\frac{1}{4} A_1^2 e^{-j2\phi_1} \delta(f + 2f_1) & \text{(at } -2f_1) \\
\frac{1}{2} A_1 A_2 e^{-j(\phi_1 - \phi_2)} \delta(f - f_1 + f_2) & \text{(at } f_1 - f_2) \\
\frac{1}{2} A_1 A_2 e^{j(\phi_1 + \phi_2)} \delta(f + f_1 - f_2) & \text{(at } -f_1 + f_2) \\
\frac{1}{4} A_1^2 e^{j2\phi_1} \delta(f - 2f_1) & \text{(at } 2f_1) \\
\frac{1}{2} A_1 A_2 e^{j(\phi_1 - \phi_2)} \delta(f - f_1 - f_2) & \text{(at } f_1 + f_2) \\
\frac{1}{4} A_2^2 e^{j2\phi_2} \delta(f - 2f_2) & \text{(at } 2f_2) .
\end{cases}
\]

### 3.3.2 Vector Frequency Description for a Second Order Nonlinearity

The result of convolving \(X(f)\) with itself in (3.127) is a set of discrete spectra, as is expected when delta functions are convolved. Examining (3.127), it can be seen that for all of the spectral content other than DC, the phase of the output spectral content maintains the same relationship to the input phase as its frequency maintains to the input frequency. For example, the first line in (3.127) occurs at \(-2\) times input frequency \(f_2\) and has a phase that is \(-2\) times input phase \(\phi_2\). This suggests an alternate way of describing the phase and spectral content using a common underlying vector space which is termed the **Vector Frequency Description** (VFD) space. In this space, the element vectors will have a number of tuples equal to the number of input tones; 2 in this illustrative case. The tones corresponding to the input frequencies will be assigned to the unit vector elements \(e_1^T = [1 \ 0]\) and \(e_2^T = [0 \ 1]\) so that when the input frequencies are assigned to a vector \(f_{in} = [f_1 \ f_2]^T\), then the input frequencies are recovered simply from the inner products \(f_1 = e_1^T \cdot f_{in}\) and \(f_2 = e_2^T \cdot f_{in}\).

Since the set of vector frequency descriptions necessary for solving a given problem will always include more than just the unit VFDs, the following notation will be adopted: A particular VFD – say the \(k^{th}\) VFD – will be denoted as \(\eta_k^T\) (i.e. always a row vector), while an element tuple corresponding to the \(q^{th}\) input frequency will be denoted as \(\eta_{k,q}\) (an ordinary
scalar). Table 3.1 shows all of the VFDs necessary to describe the second order output in (3.127) and the DC and linear outputs, $Y_0(f)$ and $Y_1(f)$, respectively, from (3.119) also.

Table 3.1: 1-Norm Sort VFD Table for 2 Tones in a 2nd Order Nonlinearity. The indices $k_x, k_y,$ and $k_z$ correspond to entries in the VFD Table for spectral vector phasors at the same indices in $X, Y,$ and $Z$, respectively.

<table>
<thead>
<tr>
<th>Frequency Index $k_x$ $(k_y, k_z)$</th>
<th>VFD $\eta^T_k$</th>
<th>1-Norm Weight $|\eta^T_k|_1$</th>
<th>$f_1$ Weight $\eta_{k,1}$</th>
<th>$f_2$ Weight $\eta_{k,2}$</th>
<th>Output Frequency (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-6$ $\eta^T_{-6}$</td>
<td>2</td>
<td>0</td>
<td>$-2$</td>
<td>$-2f_2$</td>
<td></td>
</tr>
<tr>
<td>$-5$ $\eta^T_{-5}$</td>
<td>2</td>
<td>$-1$</td>
<td>$-1$</td>
<td>$-f_1 - f_2$</td>
<td></td>
</tr>
<tr>
<td>$-4$ $\eta^T_{-4}$</td>
<td>2</td>
<td>$-2$</td>
<td>0</td>
<td>$-2f_1$</td>
<td></td>
</tr>
<tr>
<td>$-3$ $\eta^T_{-3}$</td>
<td>2</td>
<td>1</td>
<td>$-1$</td>
<td>$f_1 - f_2$</td>
<td></td>
</tr>
<tr>
<td>$-2$ $\eta^T_{-2}$</td>
<td>1</td>
<td>0</td>
<td>$-1$</td>
<td>$-f_2$</td>
<td></td>
</tr>
<tr>
<td>$-1$ $\eta^T_{-1}$</td>
<td>1</td>
<td>$-1$</td>
<td>0</td>
<td>$-f_1$</td>
<td></td>
</tr>
<tr>
<td>0 $\eta^T_0$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1 $\eta^T_1$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$f_1$</td>
<td></td>
</tr>
<tr>
<td>2 $\eta^T_2$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$f_2$</td>
<td></td>
</tr>
<tr>
<td>3 $\eta^T_3$</td>
<td>2</td>
<td>$-1$</td>
<td>1</td>
<td>$-f_1 + f_2$</td>
<td></td>
</tr>
<tr>
<td>4 $\eta^T_4$</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>$2f_1$</td>
<td></td>
</tr>
<tr>
<td>5 $\eta^T_5$</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>$f_1 + f_2$</td>
<td></td>
</tr>
<tr>
<td>6 $\eta^T_6$</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>$2f_2$</td>
<td></td>
</tr>
</tbody>
</table>

Note that each entry in the VFD table is assigned a unique frequency index (with negative frequencies having the negated indices of their positive frequency counterparts), but since the VFD table defines the form of the three vectors $Y, X,$ and $Z$, the frequency index pertaining to specific vectors will be denoted $k_y, k_x,$ and $k_z$, respectively. Also, the table is sorted in 1-norm priority fashion such that the 1-norms of the VFDs increase ($\|\eta^T_k\|_1 = |\eta_{k,1}| + |\eta_{k,2}|$) in either direction from $\eta^T_0$, the DC vector. (Another form to be introduced later sorts the
VFD table by increasing numerical frequency.) With the VFD defined in this fashion, if the phase of the two-tone input signal is described as \( \phi_{\text{in}} = [\phi_1 \ \phi_2]^T \), then (3.127) may be written somewhat more succinctly as

\[
X(f) \ast X(f) = \begin{cases}
\frac{1}{4} A_2^2 e^{j\eta_{10}^T \phi_{\text{in}}} \delta(f - \eta_{-6}^T \cdot f_{\text{in}}) & \text{at } -2f_2 \\
+ \frac{1}{2} A_1 A_2 e^{j\eta_{25}^T \phi_{\text{in}}} \delta(f - \eta_{-5}^T \cdot f_{\text{in}}) & \text{at } -f_1 - f_2 \\
+ \frac{1}{4} A_1^2 e^{j\eta_{44}^T \phi_{\text{in}}} \delta(f - \eta_{-4}^T \cdot f_{\text{in}}) & \text{at } -2f_1 \\
+ \frac{1}{2} A_1 A_2 e^{j\eta_{33}^T \phi_{\text{in}}} \delta(f - \eta_{-3}^T \cdot f_{\text{in}}) & \text{at } f_1 - f_2 \\
+ 0 \delta(f - \eta_{-2}^T \cdot f_{\text{in}}) & \text{at } -f_2 \\
+ 0 \delta(f - \eta_{-1}^T \cdot f_{\text{in}}) & \text{at } -f_1 \\
+ \frac{1}{4} A_1^2 + \frac{1}{2} A_2^2 \right] \delta(f - \eta_{0}^T \cdot f_{\text{in}}) & \text{at } \text{DC} \\
+ 0 \delta(f - \eta_{1}^T \cdot f_{\text{in}}) & \text{at } f_1 \\
+ 0 \delta(f - \eta_{2}^T \cdot f_{\text{in}}) & \text{at } f_2 \\
+ \frac{1}{2} A_1 A_2 e^{j\eta_{12}^T \phi_{\text{in}}} \delta(f - \eta_{3}^T \cdot f_{\text{in}}) & \text{at } -f_1 + f_2 \\
+ \frac{1}{4} A_1^2 e^{j\eta_{04}^T \phi_{\text{in}}} \delta(f - \eta_{1}^T \cdot f_{\text{in}}) & \text{at } 2f_1 \\
+ \frac{1}{2} A_1 A_2 e^{j\eta_{05}^T \phi_{\text{in}}} \delta(f - \eta_{2}^T \cdot f_{\text{in}}) & \text{at } f_1 + f_2 \\
+ \frac{1}{4} A_2^2 e^{j\eta_{06}^T \phi_{\text{in}}} \delta(f - \eta_{0}^T \cdot f_{\text{in}}) & \text{at } 2f_2 .
\end{cases}
\]

If the delta functions are now dropped, \( Y_0(f) = b_0 \delta(f) \), \( Y_1(f) = b_1 X(f) \) and \( Y_2(f) = b_2 X(f) \ast X(f) \) from (3.119) can be expressed in corresponding vector forms – forms that will be defined as *spectral vectors* \( \mathbf{Y}_0, \mathbf{Y}_1, \) and \( \mathbf{Y}_2 \) (noting that the subscripted \( \mathbf{Y} \) still denotes a vector, whereas an unbolded form \( Y_1 \) would refer to an element) – using a number of complex element entries
(i.e. phasors) equal to the number of entries in the VFD table, as

\[
\begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0
\end{bmatrix}
= b_0
, \quad
\begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0
\end{bmatrix}
= b_1
, \quad
\begin{bmatrix}
\frac{1}{2} A_2 e^{j \eta_1 \cdot \phi_n} \\
\frac{1}{2} A_1 e^{j \eta_2 \cdot \phi_n} \\
0 \\
0 \\
0 \\
0 \\
0 \\
0
\end{bmatrix}
= b_2
, \quad
\begin{bmatrix}
\frac{1}{4} A_2^2 e^{j \eta_1 \cdot \phi_n} \\
\frac{1}{4} A_1 A_2 e^{j \eta_2 \cdot \phi_n} \\
\frac{1}{4} A_1^2 e^{j \eta_2 \cdot \phi_n} \\
0 \\
0 \\
0 \\
0 \\
0
\end{bmatrix}, \quad (3.129)
\]

or

\[
\begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0
\end{bmatrix}
= b_0 e_0
, \quad
\begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0
\end{bmatrix}
= b_1 X
, \quad
\begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0
\end{bmatrix}
= b_2 X \ast X
. \quad (3.130)
\]

In (3.129), notice that the linear output \( Y_1 \) is just a scaled version of the input signal cast in the appropriate spectral form, and that \( Y_2 = b_2 X \ast X \) as shown in (3.130), and this can be shown by discretely convolving \( X \) with itself. Notice that the frequencies of the output spectra are found by simply adding the VFDs together, since the hidden delta functions in the spectral vectors call for addition of the frequencies as dictated by (3.126). Also note that the phases of the output spectra are found by adding VFDs together in the convolution operation.

### 3.3.3 Introduction of Matrix Methods

While convolving spectral vectors is mathematically valid and straightforward to do on ad-hoc basis, matrix-vector methods are more amenable to computer-aided implementations, and it is possible to cast the convolution of the spectral vector \( X \) with another vector using a matrix-vector form as long as both vectors have a basis rooted in the same VFD table. To see this, consider two spectral vectors, \( X \) and \( Z \), which have the same vector basis (i.e. same VFD table). Since the spectral vectors are discrete mathematical entities, a matrix which facilitates
the convolution of the two vectors can be constructed by executing the steps necessary to perform a discrete convolution of the two vectors.

As described in [88], discrete convolution of two sequences can be achieved by reversing the order of one sequence and then stepping the sequence by unit increments of the independent time or frequency variable, forming products among the vector elements which intersect at that particular step, and summing those discrete products to give the output at that step. Additional examples of discrete convolution in time are given in [142]. However, since spectral vectors in this work are functions of a vector frequency description, instead of incrementing frequencies by uniform steps, elements which intersect during the convolution of spectral vectors will have output frequencies determined by the addition of their VFD’s as called for by (3.126). Notice that the vectors in (3.129)–(3.130) have \( K = 13 \) elements. Discrete convolution of the vectors will thus require \( I = 2K - 1 = 25 \) steps of a convolution index counter. For illustrative purposes here, it will be convenient to define the elements of \( X \) from (3.129)–(3.130) using their frequency indices \( k_x \) in Table 3.1 as subscripts (and define \( Z \) using frequency indices \( k_z \)), so that

\[
X = \begin{bmatrix}
X_{-6} \\
X_{-5} \\
X_{-4} \\
X_{-3} \\
X_{-2} \\
X_{-1} \\
X_0 \\
X_1 \\
X_2 \\
X_3 \\
X_4 \\
X_5 \\
X_6 \\
\end{bmatrix}, \\
Z = \begin{bmatrix}
Z_{-6} \\
Z_{-5} \\
Z_{-4} \\
Z_{-3} \\
Z_{-2} \\
Z_{-1} \\
Z_0 \\
Z_1 \\
Z_2 \\
Z_3 \\
Z_4 \\
Z_5 \\
Z_6 \\
\end{bmatrix}.
\]

(3.131)

The discrete convolution of \( X \) and \( Z \) begins in Figure 3.2 on the following page, where the first two intersecting elements for \( I = 1 \) (indicated by a dash connecting element boxes) do not correspond to a valid convolution product since the sums of the VFDs do not map back to the VFD table.
Figure 3.2: $I = 1$, the 1st step of spectral vector discrete convolution. The crossbar connecting elements of the spectral vectors denotes elements which are multiplied together, forming atomic convolution products. At this step, there is only 1 atomic convolution product.

Figure 3.3: $I = 2$, the 2nd step of spectral vector discrete convolution. Here, there are two atomic convolution products — both invalid.

Put another way, the sum of the two VFDs in Figure 3.2 maps to what would be an output caused by a 4th order nonlinearity, but the system under consideration is limited to the 2nd order, so this convolution product does not correspond to a behavior present in the physical system being modeled. The same result occurs at the second step ($I = 2$) of the discrete convolution in Figure 3.3. These invalid convolution products are a consequence of the fact that the spectral vector must be defined such that all output spectral content can be defined, and since matrix-vector techniques with square matrices are being used, the spectral vector for the input must have the same number of element locations as the output. As a result, input vectors are usually quite sparse.

Moving along to convolution step $I = 6$, Figure 3.4 on the next page shows what appears to be a valid convolution product between $X_{-3}$ and $Z_{-4}$, where the sum of the VFDs
Figure 3.4: Step 6 of spectral vector discrete convolution shows what appears to be a mapping to a valid output VFD location, but is actually an invalid convolution product because the 1-norm of the VFD sum exceeds the system’s maximum nonlinearity.

\[
\begin{align*}
X_6 & \quad Z_6 \\
\vdots & \quad \vdots \\
X_{-1} & \quad Z_{-6} \\
X_{-2} & \quad Z_{-5} \\
X_{-3} & \quad Z_{-4} \\
X_{-4} & \quad Z_{-3} \\
X_{-5} & \quad Z_{-2} \\
X_{-6} & \quad Z_{-1} \\
\vdots & \quad \vdots \\
Z_6 & 
\end{align*}
\]

\[\eta_{-3}^T + \eta_{-4}^T = [1 \ -1] + [-2 \ 0] = [-1 \ -1] = \eta_{-5}^T\]

\[\text{BUT!}, \ ||\eta_{-3}^T||_1 + ||\eta_{-4}^T||_1 = 4 > \text{Max 1-norm of 2}\]

Therefore, this is not a valid mixing product.

Figure 3.5: Step 7 of spectral vector discrete convolution yields the first valid atomic convolution products.

\[
\begin{align*}
X_6 & \quad Z_6 \\
\vdots & \quad \vdots \\
X_0 & \quad Z_{-6} \\
X_{-1} & \quad Z_{-5} \\
X_{-2} & \quad Z_{-4} \\
X_{-3} & \quad Z_{-3} \\
X_{-4} & \quad Z_{-2} \\
X_{-5} & \quad Z_{-1} \\
X_{-6} & \quad Z_0 \\
\vdots & \quad \vdots \\
Z_6 & 
\end{align*}
\]

\[\eta_0^T + \eta_{-6}^T = [0 \ 0] + [0 \ -2] = [0 \ -2] = \eta_{-6}^T\]

\[\eta_{-6}^T + \eta_0^T = [0 \ -2] + [0 \ 0] = [0 \ -2] = \eta_{-6}^T\]

Note: Mixing of \(Z_0\) puts \(X\) in matrix column \(k_x = 0\).
maps to output frequency index $k_y = -5$. However, it is not valid because the order of nonlinearity required for this mixture would again exceed the maximum order of nonlinearity for the physical system being modeled. Figure 3.5 on the preceding page shows the first two valid convolution products that must be considered for inclusion in a convolution matrix. Here, $Z_{-6}$ forms a product with $X_0$, and the sum of the two VFDs maps to $k_y = -6$. As a result, $X_0$ will be entered into the convolution matrix in the element $(k_y, k_z) = (-6, -6)$. Also, $Z_0$ forms a product with $X_{-6}$ and the sum of the two VFDs maps to $k_y = -6$. Therefore, $X_{-6}$ will be entered into the convolution matrix in the element $(k_y, k_z) = (-6, 0)$. Notice here that knowledge of the amplitude and phase of $Z$ is not necessary to complete the task of constructing the convolution matrix. For the purpose of matrix construction, $Z$ is a dummy vector which is used only to assure the correct VFDs are summed at each step of the discrete convolution.

Moving along to convolution step $I = 13$, Figure 3.6 shows what happens at the point where the two vectors have a completely overlapping intersection: Rectification. Entries are made in the convolution matrix in row $k_y = 0$ only for those products whose VFD sums do not exceed the maximum order of nonlinearity of the system. These correspond to the

$$
\begin{array}{c|c}
X_6 & Z_{-6} \\
X_5 & Z_{-5} \\
X_4 & Z_{-4} \\
X_3 & Z_{-3} \\
X_2 & Z_{-2} \\
X_1 & Z_{-1} \\
X_0 & Z_0 \\
X_{-1} & Z_1 \\
X_{-2} & Z_2 \\
X_{-3} & Z_3 \\
X_{-4} & Z_4 \\
X_{-5} & Z_5 \\
X_{-6} & Z_6 \\
\end{array}
$$

Figure 3.6: Step 13 of spectral vector discrete convolution has atomic convolution products among every element in both spectral vectors Note that the output VFD is at DC here, so the valid products from this are due to rectification.
convolution products with frequency indices $k_x$ and $k_z$ ranging from $-2$ to $2$. In Figure 3.7, corresponding to convolution step $I = 14$, it can be seen that the discrete convolution products at a given convolution index do not necessarily map to the same output frequency indices. The same observation can be made about convolution step $I = 17$ in Figure 3.8 on the next page. Figure 3.9 on the following page shows step $I = 25$, the final convolution step, which yields no valid convolution product entries for the convolution matrix.
Figure 3.8: Step 17 of spectral vector discrete convolution, which illustrates that the output VFDs at each convolution step are not generally the same.

\[
\begin{align*}
X_6 & \quad Z_0 & \quad \eta_4^T + \eta_0^T = [2\ 0] + [0\ 0] = [2\ 0] = \eta_4^T \\
X_5 & \quad Z_1 & \quad \eta_2^T + \eta_2^T = [0\ 1] + [0\ 1] = [0\ 2] = \eta_2^T \\
X_4 & \quad Z_2 & \quad \eta_0^T + \eta_4^T = [0\ 0] + [2\ 0] = [2\ 0] = \eta_4^T \\
X_3 & \quad Z_3 \\
X_2 & \quad Z_4 \\
X_1 & \quad Z_5 \\
X_0 & \quad Z_6 \\
X_{-1} & \quad Z_{-5} \\
X_{-2} & \quad Z_{-6} \\
X_{-3} & \quad Z_{-7} \\
\vdots & \quad \vdots \\
X_{-6} & \quad Z_{-11} \\
\end{align*}
\]

Figure 3.9: Step 25, the final of spectral vector discrete convolution, is an invalid atomic convolution product.

\[
\begin{align*}
X_6 & \quad Z_6 & \quad \eta_6^T + \eta_6^T = [0\ 2] + [0\ 2] = [0\ 4] \notin \text{VFD Table} \\
\end{align*}
\]

Also, \[||\eta_6^T||_1 + ||\eta_6^T||_1 = 4 > \text{Max 1-norm of 2}\]
The complete convolution matrix, constructed by entering the valid convolution products from the convolution steps 1 to 25, is shown in Figure 3.10 where the ‘−’ entries correspond to zeros under the (forward-looking) assumption that the matrix will be constructed using a sparse matrix specification. The convolution matrix in Figure 3.10 is termed a spectrum transform matrix. Note that both the row space and the column space of the matrix correspond to the VFD table in Table 3.1 on page 48 and that the matrix contains elements of \( X \). Since a spectrum transform matrix is constructed from a specific spectral vector (based on a specific VFD table), the following terminology will be adopted: The spectrum transform matrix for a spectral vector \( X \) will be denoted \( T_x \) (same notation as \( \mathbf{T} \)), and the operator that transforms the spectral vector to a spectrum transform matrix will be denoted \( \mathbf{T} = \mathbf{T}[\cdot] \), so for example \( T_x = \mathbf{T}[X] \). Note that

\[
X * X = T_x X.
\]  

(3.132)

Notice that for the real-valued signal \( x(t) \) defined in (3.113) on page 44, the resulting elements in its spectral vector \( X \) will have conjugate symmetry, and as a result, \( T_x \) will have conjugate symmetry about its center row and column. If \( x(t) \) had been a conjugate-asymmetric...
collection of complex exponentials, then \( T_x \) would not have this symmetry. Note also that, due to the non-uniform frequency increments underlying the use of the VFD, the spectrum transform matrix does not have the Toeplitz form (where elements along each of the diagonals are constant \([143]\) that arises from simple linear convolution of a discrete signal with a discrete linear impulse function \([144]\). Furthermore, if the spectrum transform matrix is based solely upon sinusoidal input signals, the \( X_0 \) elements appearing in the main diagonal of \( T_x \) will be zero, and such matrices will be useful for direct convolution, but they are obviously not invertible since the presence of only zeros in the outer columns \( k_z = \pm 3 \) to \( k_z = \pm 6 \) indicates a lack of full column rank \([145]\). (Fortunately, when inversion is necessary, the computational situation is usually such that the main diagonal elements are non-zero.) Also, notice that, aside from the center column and the main diagonal, all of the elements appearing in the matrix happen to be the non-zero inputs in \( X \) – this will also hold for all spectrum transform matrices created from input spectral vectors. Finally, note that the center column (i.e. the \( k_z = 0 \) column) of \( T_x \) in Figure 3.10 on the previous page is equal to \( X \). This is a direct result of the \( Z_0 \) element multiplication with every element in \( X \) over the course of the 25 discrete convolution steps. Every spectrum transform matrix will contain the spectral vector \( X \) used to construct it.

One additional bit of new notation that will prove useful is to denote the spectral vector that results from repeated convolution of \( X \) with itself by \( C_x^{(1)} \), where the dot is replaced with the number of times the spectral vector has been convolved with itself. Thus, for example,

\[
C_x^{(2)} = X \ast X = T_x X,
\]

and, if \( C_x^{(1)} \) is defined as \( C_x^{(1)} = X \) for \( n = 1 \), then in general for \( n > 1 \),

\[
C_x^{(n)} = X(f) \ast X(f) \ast \cdots \ast X(f) = T_x^{n-1} X = T_x C_x^{(n-1)},
\]

where it can be seen in the last equality in \([3.133]\) that the spectrum transform matrix must be constructed such that it can raise a spectral vector from one order of nonlinearity to the next up to the maximum nonlinear order, \( N \). In \([92]\), this was not noted as the spectrum transform matrices were small and dense, but in the larger scale multitone implementation being presented in this work, it is vital that the matrices remain sparse, so instead of repeated matrix-matrix multiplication and resulting fill-in of a sparse matrix, the implementation in this work maintains the vector \( C_x^{(n)} \) internally for use in computations at the next order of
nonlinearity. This not only prevents the possibility of having Matlab® convert a sparse matrix implementation into a dense one, but it is also the most computationally efficient matrix-vector implementation, since a sparse matrix-vector multiplication is guaranteed at each order of nonlinearity with an operation count strictly proportional to the number of non-zero entries in the sparse matrix (usually fewer than 5% of matrix elements are nonzero, so the operation count is proportional to $\frac{1}{20}K^2$), whereas the dense matrix form of matrix-vector multiplication incurs an operation count proportional to $K^3$, where $K \times K$ is the size of the matrix. Note that this computational preference is counter to the use of Horner’s rule [146], which is the preferred method for computation when matrices are dense.

### 3.3.4 Vector Frequency Description for A Third Order Nonlinearity

The results obtained in Section 3.3.2 are fine for two tones through a second order nonlinearity. Suppose the analysis is continued through the third order nonlinearity. Now it is necessary to find the spectral vector for $X * X * X$. From Section 3.3.2, the solution that suggests itself is to convolve $X$ with the vector for $X * X$ from (3.129)–(3.130). Now, however, the maximum nonlinear order to which VFD 1-norms must be allowed is elevated to 3, so the 2nd order limited VFD table in Table 3.1 on page 48 must be augmented to account for 3rd order output mixtures. Table 3.2 shows such a VFD table.

**Table 3.2: 1-Norm Sort VFD Table for 2 Tones in a 3rd Order Nonlinearity**

<table>
<thead>
<tr>
<th>Frequency Index $k_x$ $(k_y, k_z)$</th>
<th>VFD $\eta^T_{k_x}$</th>
<th>1-Norm Weight $|\eta^T_k|_1$</th>
<th>$f_1$ Weight $\eta_{k,1}$</th>
<th>$f_2$ Weight $\eta_{k,2}$</th>
<th>Output Frequency (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-12$ $\eta^T_{-12}$</td>
<td>3</td>
<td>0</td>
<td>$-3$</td>
<td>$-3f_2$</td>
<td></td>
</tr>
<tr>
<td>$-11$ $\eta^T_{-11}$</td>
<td>3</td>
<td>$-1$</td>
<td>$-2$</td>
<td>$-f_1 - 2f_2$</td>
<td></td>
</tr>
<tr>
<td>$-10$ $\eta^T_{-10}$</td>
<td>3</td>
<td>$-2$</td>
<td>$-1$</td>
<td>$-2f_1 - f_2$</td>
<td></td>
</tr>
<tr>
<td>$-9$ $\eta^T_{-9}$</td>
<td>3</td>
<td>$-3$</td>
<td>0</td>
<td>$-3f_1$</td>
<td></td>
</tr>
<tr>
<td>$-8$ $\eta^T_{-8}$</td>
<td>3</td>
<td>1</td>
<td>$-2$</td>
<td>$f_1 - 2f_2$</td>
<td></td>
</tr>
<tr>
<td>$-7$ $\eta^T_{-7}$</td>
<td>3</td>
<td>$-2$</td>
<td>1</td>
<td>$-2f_1 + f_2$</td>
<td></td>
</tr>
<tr>
<td>$-6$ $\eta^T_{-6}$</td>
<td>2</td>
<td>0</td>
<td>$-2$</td>
<td>$-2f_2$</td>
<td></td>
</tr>
</tbody>
</table>

*Continued on next page*
### Table 3.2 (continued)

<table>
<thead>
<tr>
<th>Frequency Index $k_x$ $(k_y, k_z)$</th>
<th>VFD</th>
<th>1-Norm $|\eta_k^T|_1$</th>
<th>$f_1$ Weight $\eta_{k,1}$</th>
<th>$f_2$ Weight $\eta_{k,2}$</th>
<th>Output Frequency (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-5$ $\eta_{-5}^T$</td>
<td>2</td>
<td>-1</td>
<td>-1</td>
<td></td>
<td>$-f_1 - f_2$</td>
</tr>
<tr>
<td>$-4$ $\eta_{-4}^T$</td>
<td>2</td>
<td>-2</td>
<td>0</td>
<td></td>
<td>$-2f_1$</td>
</tr>
<tr>
<td>$-3$ $\eta_{-3}^T$</td>
<td>2</td>
<td>1</td>
<td>-1</td>
<td></td>
<td>$f_1 - f_2$</td>
</tr>
<tr>
<td>$-2$ $\eta_{-2}^T$</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td></td>
<td>$-f_2$</td>
</tr>
<tr>
<td>$-1$ $\eta_{-1}^T$</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td></td>
<td>$-f_1$</td>
</tr>
<tr>
<td>0 $\eta_0^T$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>1 $\eta_1^T$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td>$f_1$</td>
</tr>
<tr>
<td>2 $\eta_2^T$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td>$f_2$</td>
</tr>
<tr>
<td>3 $\eta_3^T$</td>
<td>2</td>
<td>-1</td>
<td>1</td>
<td></td>
<td>$-f_1 + f_2$</td>
</tr>
<tr>
<td>4 $\eta_4^T$</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td></td>
<td>$2f_1$</td>
</tr>
<tr>
<td>5 $\eta_5^T$</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
<td>$f_1 + f_2$</td>
</tr>
<tr>
<td>6 $\eta_6^T$</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td></td>
<td>$2f_2$</td>
</tr>
<tr>
<td>7 $\eta_7^T$</td>
<td>3</td>
<td>2</td>
<td>-1</td>
<td></td>
<td>$2f_1 - f_2$</td>
</tr>
<tr>
<td>8 $\eta_8^T$</td>
<td>3</td>
<td>-1</td>
<td>2</td>
<td></td>
<td>$-f_1 + 2f_2$</td>
</tr>
<tr>
<td>9 $\eta_9^T$</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td></td>
<td>$3f_1$</td>
</tr>
<tr>
<td>10 $\eta_{10}^T$</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
<td>$2f_1 + f_2$</td>
</tr>
<tr>
<td>11 $\eta_{11}^T$</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td></td>
<td>$f_1 + 2f_2$</td>
</tr>
<tr>
<td>12 $\eta_{12}^T$</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td></td>
<td>$3f_2$</td>
</tr>
</tbody>
</table>

The input spectral vector $X$ as a function of Table 3.2 on the preceding page is shown...
in (3.135), along with the result of convolving this vector with itself, i.e. \( \mathbf{C}_x^{(2)} = \mathbf{X} \ast \mathbf{X} \).

\[
\mathbf{X} = \begin{bmatrix}
X_{-12} \\
X_{-11} \\
X_{-10} \\
X_{-9} \\
X_{-8} \\
X_{-7} \\
X_{-6} \\
X_{-5} \\
X_{-4} \\
X_{-3} \\
X_{-2} \\
X_{-1} \\
X_0 \\
X_1 \\
X_2 \\
X_3 \\
X_4 \\
X_5 \\
X_6 \\
X_7 \\
X_8 \\
X_9 \\
X_{10} \\
X_{11} \\
X_{12}
\end{bmatrix}, \quad \mathbf{C}_x^{(2)} = \begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
\frac{1}{2}A_2^2 \varepsilon \mathbf{m}_{6e}^T \phi_m \\
\frac{1}{2}A_1 A_2 \varepsilon \mathbf{m}_{5e}^T \phi_m \\
\frac{1}{4}A_1^2 \varepsilon \mathbf{m}_{6d}^T \phi_m \\
\frac{1}{2}A_1 A_2 \varepsilon \mathbf{m}_{4e}^T \phi_m \\
\frac{1}{4}A_1^2 \varepsilon \mathbf{m}_{5d}^T \phi_m \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0
\end{bmatrix} \tag{3.135}
\]

Construction of a spectrum transform matrix suitable for 3rd order nonlinearities follows the same process illustrated for the 2nd order nonlinearity. Its details will be omitted. The spectrum transform matrix corresponding to Table 3.2 on page 59 is shown in Figure 3.11 on page 63. Note the shaded entries in the spectrum transform matrix in Figure 3.11. These appear in matrix locations (1, -3), (1, 0), (1, 4), and (1, 5). The result of matrix-vector multiplication of
\( T_x \) and \( C_x^{(3)} \) in output row \( k_y = 1 \) will be

\[
\begin{align*}
C_{x_1}^{(3)} &= X_2 \frac{1}{2} A_1 A_2 e^{m_{x_1} T} \phi_{in} + X_1 \left[ \frac{1}{2} A_1^2 + \frac{1}{2} A_2^2 \right] + X_{-1} \frac{1}{4} A_1^2 e^{m_{x_1} T} \phi_{in} \\
&+ X_{-2} \frac{1}{2} A_1 A_2 e^{m_{x_1} T} \phi_{in} \\
&= \frac{1}{2} A_2 e^{m_{x_1} T} \phi_{in} + \frac{1}{2} A_1 A_2 e^{m_{x_3} T} \phi_{in} + \frac{1}{2} A_1 e^{m_{x_1} T} \phi_{in} \left[ \frac{1}{2} A_1^2 + \frac{1}{2} A_2^2 \right] \\
&+ \frac{1}{2} A_1 e^{m_{x_1} T} \phi_{in} + \frac{1}{2} A_2 e^{m_{x_3} T} \phi_{in} + \frac{1}{2} A_1 A_2 e^{m_{x_1} T} \phi_{in} \\
&= \frac{1}{4} A_1 A_2^2 e^{(n_{x_1} T + n_{x_3} T)} \phi_{in} + \left[ \frac{1}{4} A_1^3 + \frac{1}{4} A_1 A_2^2 \right] e^{m_{x_1} T} \phi_{in} \\
&+ \frac{1}{8} A_1^3 e^{m_{x_1} T} \phi_{in} + \frac{1}{4} A_1 A_2^2 e^{(n_{x_3} T + n_{x_1} T)} \phi_{in} \\
&= \frac{1}{4} A_1 A_2^2 e^{m_{x_1} T} \phi_{in} + \left[ \frac{1}{4} A_1^3 + \frac{1}{4} A_1 A_2^2 \right] e^{m_{x_1} T} \phi_{in} \\
&+ \frac{1}{8} A_1^3 e^{m_{x_1} T} \phi_{in} + \frac{1}{4} A_1 A_2^2 e^{m_{x_1} T} \phi_{in} \\
&= \frac{3}{8} A_1 A_2^2 e^{m_{x_1} T} \phi_{in} + \frac{3}{8} A_1^3 e^{m_{x_1} T} \phi_{in} \\
&= \frac{3}{8} [A_1 A_2^2 + A_1^3] e^{m_{x_1} T} \phi_{in},
\end{align*}
\]

where it can be seen that \( C_{x_1}^{(3)} \) is a 3rd order nonlinear output that appears at \( k_y = 1 \), or at exactly the same frequency and — after accounting for possible phase rotations introduced by the scale factors \( b_1 \) and \( b_3 \) [in (3.118) on page 45] — with a phase that is coherently aligned to the linear output. These matrix entries are responsible for creating correlated intermodulation distortion. The phenomenon will be explored in detail in Section 3.5.5, where it will be seen that uncorrelated intermodulation distortion products will occur at different frequency indices (but at the same numeric frequency!) when spectral vectors are based upon a VFD table and three or more equally-spaced input tones are used.

The fractional scale factor \( \frac{3}{8} \) appearing in (3.141) may appear to be too small by a factor of \( \frac{1}{2} \) to those familiar with traditional sinusoidal distortion analysis [147], but note that here both positive and negative frequencies are being used, while the traditional distortion analysis uses only positive frequencies. The results here are equivalent to the traditional analysis if the outputs for \( k_y = 1 \) and \( k_y = -1 \) (which are complex conjugates in this case) are added.
Figure 3.11: Spectrum Transform Matrix created from (3.135) on page 61.
3.4 The Vector Frequency Description

In Section 3.3.2 the Vector Frequency Description (VFD) was informally introduced for the specific case of two input tones through a 2\textsuperscript{nd} order nonlinearity. In this section, the VFD and the space it spans will be described formally and generally. The development will begin with substituting the unit VFDs into the equation describing the input signal, followed by the substitution of this input signal into an arbitrary polynomial nonlinearity, where it will be seen that the number of VFDs grows with the order of nonlinearity up to the maximum nonlinear order.

3.4.1 Input Signal

Consider an input signal composed of $Q$ incommensurate tones as described in [62],

$$x(t) = \sum_{q=1}^{Q} A_q [\cos(2\pi f_q t + \phi_q)] = \frac{1}{2} \sum_{q=-Q}^{Q} A_q e^{j2\pi f_q t + j\phi_q},$$

(3.142)

where $A_q$, $f_q$, and $\phi_q$ are the amplitude, frequency, and phase offsets, respectively, of each signal component, and with $A_{-q} = A_q$, $f_{-q} = -f_q$, and $\phi_{-q} = -\phi_q$. From (3.74) on page 39, the Fourier transform of (3.142) is given by

$$X(f) = \frac{1}{2} \sum_{q=-Q}^{Q} A_q e^{j\phi_q} \delta(f - f_q) = \frac{1}{2} \sum_{q=-Q}^{Q} X_q \delta(f - f_q),$$

(3.143)

where $X_q = A_q e^{j\phi_q}$.

3.4.2 Input Signal using the VFD

The input signal $X(f)$ in (3.143) can be defined using a vector form as follows: Let the frequencies of the $Q$ sinusoidal inputs frequencies be given by a vector called $f_{in} = [f_1 \ f_2 \ f_3 \ \ldots \ f_Q]^T$ (where $f_{in} \in \mathbb{R}^{Q \times 1}$) and let the phases of the $Q$ sinusoidal inputs be given by $\phi_{in} = [\phi_1 \ \phi_2 \ \phi_3 \ \ldots \ \phi_Q]^T$ (where $\phi_{in} \in \mathbb{R}^{Q \times 1}$). Let each of the sinusoidal inputs be described by a unit vector VFD $\eta_q^T = e_q^T$. For example,

$$\eta_1^T = [1 \ 0 \ 0 \ \ldots \ 0]$$

$$\eta_2^T = [0 \ 1 \ 0 \ \ldots \ 0],$$
and, finally,

$$\eta^T_Q = [0 \ 0 \ 0 \ \ldots \ 1].$$ (3.144)

For negative indices of $q$, the following holds:

$$\eta^T_{-1} = [-1 \ 0 \ 0 \ \ldots \ 0]$$
$$\eta^T_{-2} = [0 \ -1 \ 0 \ \ldots \ 0],$$

and, finally,

$$\eta^T_{-Q} = [0 \ 0 \ 0 \ \ldots \ -1].$$ (3.145)

With these definitions in hand, the following substitutions can be made in (3.143)

$$f_q = \eta^T_q f_{in}$$ (3.146)
$$\phi_q = \eta^T_q \phi_{in},$$ (3.147)

with the resulting form

$$X(f) = \frac{1}{2} \sum_{q=-Q}^Q A_q e^{j\eta_q \phi_{in}} \delta(f - \eta^T_q \cdot f_{in}),$$ (3.148)

### 3.4.3 Convolution of the Input using the VFD Form

In (3.117) on page 45 the polynomial transfer function

$$y(t) = \sum_{n=0}^N b_n [x(t)]^n$$ (3.149)

was first encountered. The Fourier transform of $y(t)$ is given by (3.119) on page 45 and is repeated here for convenience:

$$Y(f) = b_0 \delta(f) + b_1 X(f) + b_2 X(f) \ast X(f) +$$
$$b_3 X(f) \ast X(f) \ast X(f) + \cdots +$$
$$b_N \underbrace{X(f) \ast X(f) \ast \cdots \ast X(f)}_{N \text{ times}}$$ (3.150)

$$= Y_0(f) + Y_1(f) + Y_2(f) + \cdots + Y_N(f).$$ (3.151)
If \( X(f) \) as given by (3.148) is now substituted into the various expressions for the \( Y_n(f) \) in (3.151), the following results are obtained: For \( Y_1(f) \),

\[
Y_1(f) = b_1 X(f) = b_1 \frac{1}{2} \sum_{q=-Q}^{Q} A_q e^{j\eta^T_q \cdot \phi_{in}} \delta (f - \eta^T_q \cdot \mathbf{f}_{in}) ;
\]

(3.152)

for \( Y_2(f) \),

\[
Y_2(f) = b_2 X(f) \ast X(f) = b_2 \int_{-\infty}^{\infty} X(\nu) X(f - \nu) d\nu
\]

\[
= b_2 \int_{-\infty}^{\infty} \left[ \frac{1}{2} \sum_{q_2=-Q}^{Q} A_q e^{j\eta^T_{q2} \cdot \phi_{in}} \delta (\nu - \eta^T_{q2} \cdot \mathbf{f}_{in}) \right] \cdot \left[ \frac{1}{2} \sum_{q_1=-Q}^{Q} A_q e^{j\eta^T_{q1} \cdot \phi_{in}} \delta (f - \eta^T_{q1} \cdot \mathbf{f}_{in} - \nu) \right] d\nu
\]

\[
= b_2 \frac{1}{2^2} \sum_{q_2=-Q}^{Q} \sum_{q_1=-Q}^{Q} A_q A_q e^{j(\eta^T_{q2} + \eta^T_{q1}) \cdot \phi_{in}} \int_{-\infty}^{\infty} \delta \left( \nu - \eta^T_{q2} \cdot \mathbf{f}_{in} \right) \delta \left( f - \eta^T_{q1} \cdot \mathbf{f}_{in} - \nu \right) d\nu
\]

\[
= b_2 \frac{1}{2^2} \sum_{q_2=-Q}^{Q} \sum_{q_1=-Q}^{Q} A_q A_q e^{j(\eta^T_{q2} + \eta^T_{q1}) \cdot \phi_{in}} \delta \left( f - (\eta^T_{q2} + \eta^T_{q1}) \cdot \mathbf{f}_{in} \right) .
\]

(3.153)

From (3.153) it can be observed that the result of convolving the input signal with itself in the frequency domain is a set of new vector tuples of the form \( \eta^T_{k2} = \eta^T_{q2} + \eta^T_{q1} \), where the 1-norm of \( \eta^T_{k2} \) will be

\[
\| \eta^T_{k2} \|_1 = \begin{cases} 0 & \text{for } q2 = -q1 \\ 2 & \text{otherwise} \end{cases}
\]

(3.154)

When \( \| \eta^T_{k2} \|_1 = 0 \), then rectification has taken place and the result must appear at the DC location in the function output. The need to account for rectification means that any valid VFD table must contain a zero-vector entry \( \eta^T_0 = [0 \ 0 \ 0 \ldots 0] \). (This function mapping will be handled by code for the spectrum mapping table, which will be discussed in Section 3.6.) Otherwise, a new second-order VFD is created with one-norm \( \| \eta^T_{k2} \|_1 = 2 \). This second-order VFD will have two non-zero tuples for cases of intermodulation distortion, which will occur when \( q2 \neq q1 \) (and \( q2 \neq -q1 \)). When \( q2 = q1 \), then the second-order VFD will have only one non-zero tuple and this will denote the case of harmonic distortion. Observe also that the resulting VFD in (3.153) determines the phase of the output.
From (3.153) it can be noted that

\[
X(f) \ast X(f) = \frac{1}{2^2} \sum_{q_2=-Q}^{Q} \sum_{q_1=-Q}^{Q} A_{q_2} A_{q_1} e^{j(q_{2} + q_{1}) \phi_m \delta (f - (\eta_{q_2}^T + \eta_{q_1}^T) \cdot f_{in})},
\]

(3.155)

so that the convolution of the spectral vectors has created new spectral content at the sums of unit VFD frequencies. Consider now \(Y_3(f) = b_3 X(f) \ast X(f) \ast X(f)\). Using (3.155) to advantage,

\[
Y_3(f) = b_3 X(f) \ast X(f) \ast X(f) = b_3 \int_{-\infty}^{\infty} X(\nu) [X \ast X(f - \nu)] d\nu
\]

\[
= b_3 \int_{-\infty}^{\infty} \left[ \frac{1}{2^2} \sum_{q_3=-Q}^{Q} A_{q_3} e^{j(q_{3} \phi_m \eta_{q_3}^T) \cdot f_{in}} \right] \cdot
\]

\[
\left[ \frac{1}{2^2} \sum_{q_2=-Q}^{Q} \sum_{q_1=-Q}^{Q} A_{q_2} A_{q_1} e^{j(q_{2} + q_{1}) \phi_m \delta (f - (\eta_{q_2}^T + \eta_{q_1}^T) \cdot f_{in} - \nu)} \right] d\nu
\]

\[
= b_3 \frac{1}{2^3} \sum_{q_3=-Q}^{Q} \sum_{q_2=-Q}^{Q} \sum_{q_1=-Q}^{Q} \left\{ A_{q_3} A_{q_2} A_{q_1} e^{j(q_{3} \phi_m + q_{2} \phi_m + q_{1} \phi_m) \cdot f_{in}} \right\} \cdot
\]

\[
\left[ \int_{-\infty}^{\infty} \delta (\nu - \eta_{q_3}^T \cdot f_{in}) \delta (f - (\eta_{q_2}^T + \eta_{q_1}^T) \cdot f_{in} - \nu) d\nu \right]
\]

\[
= b_3 \frac{1}{2^3} \sum_{q_3=-Q}^{Q} \sum_{q_2=-Q}^{Q} \sum_{q_1=-Q}^{Q} A_{q_3} A_{q_2} A_{q_1} e^{j(q_{3} + q_{2} + q_{1}) \phi_m \delta (f - (\eta_{q_3}^T + \eta_{q_2}^T + \eta_{q_1}^T) \cdot f_{in})},
\]

(3.156)

From (3.156) it can be observed that the result of convolving the input signal with itself for a third time creates new VFDs of the form \(\eta_{k3}^T = \eta_{q_3}^T + \eta_{k2}^T\), where the 1-norm of \(\eta_{k3}^T\) will be

\[
\|\eta_{k3}^T\|_1 = \begin{cases} 1 & \text{for } \|\eta_{k2}^T\|_1 = 2 \text{ and } q_3 = -q_1 \text{ or } q_3 = -q_2 \\
3 & \text{otherwise} \end{cases}
\]

(3.157)

When \(\|\eta_{k3}^T\|_1 = 1\), then assuming \(\eta_{k2}^T\) had a one-norm of 2, then a phase cancellation process has taken place where \(q_3 = -q_1\) or \(q_3 = -q_2\) and the resulting 3rd order output signal becomes coherently aligned with the linear output at one of existing VFDs defined for the linear response.
(It will be seen later that tracking this 3rd order output separately from the linear output is not a problem since output vectors are created for each order of nonlinearity.) Otherwise, a new 3rd order VFD is created with one-norm \( \| \eta_{k3}^T \|_1 = 3 \). This 3rd order VFD will have two or three non-zero tuples for cases of intermodulation distortion or else it will have a single tuple with an absolute value of 3 which will denote an instance of 3rd order harmonic distortion. Again observe that the resulting VFD in (3.156) determines the phase of the output.

In order to generalize the above development, the following shorthand expressions for the product of the amplitudes \( A_q \) and sum of the unit VFDs will be adopted:

\[
\prod_{m=1}^{N} A_{qm} = A_{qN} \cdots A_{q1} \tag{3.158}
\]

\[
\sum_{m=1}^{N} \eta_{qm}^T = \eta_{qN}^T + \cdots + \eta_{q1}^T. \tag{3.159}
\]

Note that in creating these expressions that \( m \), the index variable in both cases, does not denote a new independent variable. The underlying independent variables remain \( q1, \ldots, qN \).

Substituting (3.158) and (3.159) into (3.156) and generalizing it to the highest order of nonlin-
the system under consideration. If those vectors having a 1-norm less than or equal to \( N \) in the VFD Table will not span the vector space \( \mathbb{N} \). From (3.160) it can be seen that the result of convolving the input signal with itself for the \( \eta \):

\[
Y_N(f) = b_N X(f) * X * \cdots * X(f) = b_N \int_{-\infty}^{\infty} X(\nu)[X * \cdots * X(f-\nu)]d\nu
\]

\[
= b_N \int_{-\infty}^{\infty} \left[ \frac{1}{2} \sum_{qN=-Q}^{Q} A_q \eta_{qN} \phi_{in} \delta \left( \nu - \eta_{qN}^T \cdot f_{in} \right) \right] \frac{1}{2(N-1)}
\]

\[
= b_N \int_{-\infty}^{\infty} \left[ \sum_{q(N-1)=0}^{Q} \sum_{q_1=0}^{Q} \left( \prod_{m=1}^{N-1} A_{qm} \right) \left( \delta \left( \nu - \sum_{m=1}^{N-1} \eta_{qm}^T \cdot f_{in} \right) \right) \delta \left( f - \sum_{m=1}^{N-1} \eta_{qm}^T \cdot f_{in} - \nu \right) \right] d\nu
\]

\[
= b_N \left[ \int_{-\infty}^{\infty} \delta \left( \nu - \eta_{qN}^T \cdot f_{in} \right) \delta \left( f - (\eta_{q(N-1)}^T + \cdots + \eta_{q1}^T) \cdot f_{in} - \nu \right) d\nu \right]
\]

From (3.160) it can be seen that the result of convolving the input signal with itself for the \( N \)-th time creates new VFDs with 1-norm of \( \eta_{kN}^T = N \) in some cases, while in other cases phase cancellation will cause a result with a 1-norm of \( N - 2 \):

\[
\| \eta_{kN}^T \|_1 = \begin{cases} 
N - 2 & \text{for } qN = -qX \text{ for } X \in (N-1), \ldots, 1 \\
N & \text{otherwise}
\end{cases}
\]

Again observe that the resulting VFD in (3.160) also determines the phase of the output.

In general, then, each time \( X(f) \) is convolved with itself, a new set of VFDs with 1-norms equal to the order of nonlinearity is created. Thus, for any problem, the set of VFDs in the VFD Table will not span the vector space \( \mathbb{N}^{1\times Q} \) of all \( Q \)-tuples, but will be constrained to those vectors having a 1-norm less than or equal to \( N \), the maximum order of nonlinearity of the system under consideration. If \( H \) is defined as the set of VFDs in a particular VFD Table,
then,

\[ \mathcal{H} = \left\{ \eta_k^T \mid \|\eta_k^T\|_1 \leq N \right\} \] (3.162)

### 3.4.4 Modified Vector Space Properties of the VFD Table

It was seen in Section 3.4.3 how the set of VFDs is formed from the unit vector elements defining the input signal, and that, in a strict mathematical sense, the VFD table for a particular problem defines a set, but does not span a vector space. Nevertheless, the VFD has many vector qualities such that a formal definition of those qualities is now in order. Given an input signal composed of \( Q \) incommensurate tones as in (3.142) and a nonlinear transfer function with maximum order of nonlinearity \( N \) as in (3.149), the frequencies at which spectral content occurs in the nonlinear system may be described by the complete set, \( \mathcal{H} \), of VFD \( Q \)-tuples subject to the limitation that the 1-norm of every vector in the set must not be greater than \( N \).

**Definition 1** (Modified Vector Space Properties). Assume that \( \eta_k^T, \eta_l^T, \) and \( \eta_m^T \in \mathcal{H} \). Then the following properties hold:

- **Closure** \( \eta_k^T + \eta_l^T \in \mathcal{H} \) subject to the constraint that \( \|\eta_k^T\|_1 + \|\eta_l^T\|_1 \leq N \).
- **Associativity** \( (\eta_k^T + \eta_l^T) + \eta_m^T = \eta_k^T + (\eta_l^T + \eta_m^T) \) subject to the constraint that \( \|\eta_k^T\|_1 + \|\eta_l^T\|_1 + \|\eta_m^T\|_1 \leq N \).
- **Commutativity** \( \eta_k^T + \eta_l^T = \eta_l^T + \eta_k^T \) subject to the constraint that \( \|\eta_k^T\|_1 + \|\eta_l^T\|_1 \leq N \).
- **Additive Identity** There is a vector \( \eta_0^T \in \mathcal{H} \) with \( \|\eta_0^T\|_1 = 0 \) such that \( \eta_k^T + \eta_0^T = \eta_k^T \).
- **Additive Inverse** For each \( \eta_k^T \in \mathcal{H} \) there exists \( (\eta_{-k}^T = -\eta_k^T) \in \mathcal{H} \) such that \( \eta_k^T + \eta_{-k}^T = 0 \).

All of these properties are those of vector spaces [143], qualified by the fact that the 1-norms of every vector in the set must not be greater than the order of nonlinearity, \( N \).

There are conditions under which it is advantageous to dispense with the Additive Inverse property. This property is a statement that spectra considered are always two-sided. However, when the input signals of interest are real (i.e. not complex baseband), it is possible to exploit the conjugate symmetry of the input spectrum to obtain results with vectors that require only half the storage of their two-sided counterparts and matrices requiring only one quarter
of the storage of their two-sided counterparts. In prior work, Chang \cite{92} made exclusive use of single-sided spectra with smaller dense spectrum transform matrices. However, the present work exploits the sparse matrix capabilities of Matlab\textsuperscript{®}, so the differences in storage requirements between one-sided and two-sided spectrum transform matrices is modest.

3.4.5 Construction of the VFD Table

3.4.5.1 Storage Allocation

The cardinality or number of distinct vector elements $|\mathcal{H}|$ in $\mathcal{H}$ for a given $Q$ and $N$ is a combinatorial problem for which a predictive closed-form expression is still being investigated. The algorithm to be described here is able to compute the VFD table exactly, where the maximum index for VFDs with positive frequencies is denoted by $K$, the corresponding index for negative frequencies is $-K$, and 0 is included to account for DC. Thus there are exactly $2K+1$ VFDs when negative frequencies are included, while there are only $K+1$ VFDs if only non-negative frequencies are considered. Finding the exact cardinality $|\mathcal{H}|$ is no mere academic exercise, however, as some guess must be made of the number of entries in the VFD table so that the algorithm which creates the table can allocate and manage its storage. For the case of $Q > N$, the number of VFDs with positive output frequencies created at each order of nonlinearity, $n = 1, \ldots, N$, is bound by the following relationship:

$$|\mathcal{H}_n| \leq \hat{K}_n = 2^{(n-1)} \left[ \binom{Q+n-1}{n} - \sum_{n_z=1}^{n} \binom{Q+n_z-1}{n_z} \right] + \sum_{n_z=1}^{n} 2^{(n-1)} \binom{Q+n_z-1}{n_z} ,$$

(3.163)

where $n_z$ denotes the number of non-zero tuples in VFDs. The total number of VFDs with positive output frequencies over all $n$ is given by

$$\hat{K}^+ = \sum_{n=1}^{N} \hat{K}_n .$$

(3.164)

At the beginning of computations, $\hat{K}^+ \times Q$ bytes of storage is reserved to hold $\mathcal{H}^+$, the positive-frequency portion of the VFD table, during construction. Any over-allocated storage is freed at the completion of construction.

To understand how the combinatoric relationships in (3.163) can create very large computing problems, Figure 3.12 on the following page shows the number of VFDs in a 1-sided
Figure 3.12: Number of 1-sided VFD Table entries for given number of input tones $Q$ and order of nonlinearity, $N$, assuming no spectral truncation.

Figure 3.13: Estimate of storage required for the minimal set of data structures to solve a particular problem.
VFD Table created as a function of the number of input frequencies and the order of nonlinearity, and Figure 3.13 on the previous page provides an estimate of the storage required to hold a minimal set of the data structures (VFD Table, input and output spectral vectors, sparse spectrum transform matrix) required to solve a particular problem with the AOM Toolbox.

3.4.5.2 Algorithm Description

Despite the difficulty of predicting the number of vectors in the VFD table in advance, the computation of the elements in the VFD table is a straightforward matter. Since repeated discrete convolution corresponds to repeated addition of the unit vector frequency descriptions corresponding to the input tones as shown in Section 3.4.3, the VFD table is built using this same notion, beginning with the linear response, where \( n = 1 \). The first \( Q \) VFD entries in \( \mathcal{H}^+ \) are those corresponding to a \( Q \times Q \) identity matrix, \( I \). Let \( \mathcal{H}_n^{+} \) denote the portion of \( \mathcal{H} \) having 1-norm \( n \), so that \( \mathcal{H}_1^{+} = I \). These vectors denote the input frequencies in VFD form. Note that each vector \( \eta_q^T \) has 1-norm \( \| \eta_q^T \|_1 = 1 \) with a 1 in the \( q \)th (column) position.

Next, all of the vectors for the second order nonlinearity, \( \mathcal{H}_2^+ \) are computed. To facilitate this, a \( Q \times Q \) upper circulant shift matrix – a permutation matrix since it contains a single 1 element in each row and column and 0 otherwise – is used. Let \( \mathbf{P} \) denote this matrix and \( p_{ij} \) denote its elements, where \( i \) and \( j \) range from 1 to \( Q \). The elements of \( \mathbf{P} \) are defined as follows:

\[
p_{ij} = \begin{cases} 
1 & \text{for } i = Q, j = 1 \\
1 & \text{for } i = 1, \ldots, Q - 1 \text{ and } j = i + 1 \\
0 & \text{otherwise}
\end{cases}
\]  

(3.165)

Visually, \( \mathbf{P} \) is an identity matrix with all of its columns shifted up one row, and with the first
column wrapped to the $Q^{th}$ row:

$$P = \begin{bmatrix}
0 & 1 & 0 & 1 & 0 & \ldots & 0 \\
0 & 0 & 1 & 0 & 1 & \ldots & 0 \\
0 & \ldots & \ldots & \ldots & \ldots & \ldots & 0 \\
1 & 0 & 0 & 1 & 0 & \ldots & 0 \\
\end{bmatrix}.$$  

(3.166)

In figure form, with row and column indices, $P$ appears in Figure 3.14. The matrix $P$ has the property that when it left multiplies any other conformable matrix, it moves all of the rows up and flips the top row down to the bottom. Notice that when $P$ is multiplied by itself, the effect is to cause a further shift of its columns up, as in Figure 3.15. Second order VFDs are computed by adding and subtracting upper circulant [143] shifted versions of the identity matrix to $H_1^+$ (which is also the identity matrix). This creates new matrices that will be termed candidate matrices, denoted $H_{nc}$, where the rows of the candidate matrices contain potential VFD entries. For the second order, $H_{2c}$ matrices are creating from the following matrix equation:

$$H_{2c} = H_1^+ \pm P^q$$

$$= I \pm P^q \quad \forall q \in 1, \ldots, Q.$$  

(3.167)

Each row of $H_{2c}$ is then tested in two ways:
1. If a row entry, i.e. a candidate VFD entry, does not have a 1-norm equal to $n$ (2 in this case), then it is discarded.

2. For those candidate VFD entries with a 1-norm of 2, the dot product with the input frequency vector $f_{in}$ is performed to see if the resulting frequency is non-negative. Note that VFD table entries during this construction phase are limited to non-negative frequencies. Candidate VFDs that pass both tests are appended as rows to $\mathcal{H}$. Also note that the upper circulant shift must be applied $Q$ times in order to visit every possible tuple location with a 1-norm of 2, and thus (3.167) must be computed $Q$ times and each of $Q$ candidate matrices must be inspected.

The process continues for the third order, $n = 3$, with the added complication that $\mathcal{H}^+_2$ is not now simply an identity matrix, but is a much larger set, so that it must be blocked into $Q \times Q$ matrices $\mathcal{H}^+_2$, and to each unique $\mathcal{H}^+_2$ matrix the series of permutation matrices $P^q$ must be added and subtracted:

$$H_{3c} = \mathcal{H}^+_2 \pm P^q \quad \forall b \quad \text{and, for each } b \quad \forall q \in 1, \ldots, Q.$$  \hfill (3.168)

In general, for all orders of nonlinearity from 3 up to $N$, candidate VFDs are created by forming the candidate matrices $H_{nc}$:

$$H_{nc} = \mathcal{H}^+_{(n-1)b} \pm P^q \quad \forall b \quad \text{and, for each } b \quad \forall q \in 1, \ldots, Q.$$  \hfill (3.169)

Valid VFD entries will be all those rows in candidate matrices with the correct 1-norm and with a non-negative frequency upon dotting with $f_{in}$.

---

Figure 3.15: Upper circulant shift permutation matrix $P^2$. 

<table>
<thead>
<tr>
<th>$i$</th>
<th>$1$</th>
<th>$2$</th>
<th>$3$</th>
<th>...</th>
<th>$Q-2$</th>
<th>$Q-1$</th>
<th>$Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td></td>
<td>..</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td></td>
<td>..</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td></td>
<td>..</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Q-2$</td>
<td></td>
<td></td>
<td>..</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Q-1$</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Q$</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3.4.5.3 Duplicate Candidate VFDs

The basic algorithm described here overlooks the possibility of duplicate candidate VFDs – the situation where $\mathcal{H}^+$ already contains the candidate VFD – but this occurs quite often in practice. The necessity to avoid putting duplicate vector entries in $\mathcal{H}^+$ during construction is a crucial limiting factor to computational efficiency in constructing $\mathcal{H}^+$. For a limited number of tones and low orders of nonlinearity, $|\mathcal{H}^+|$ is small enough that linear searching techniques are sufficient for checking $\mathcal{H}^+$ for duplicate entries. However, for multitone input signals (greater numbers of columns in $\mathcal{H}^+$) and higher orders of nonlinearity (greater numbers of rows in $\mathcal{H}^+$), linear searching is untenable. The solution to this problem is to create an associative memory data structure at each order of nonlinearity, denoted $\mathcal{A}_n$, in the form of a hash table which is accessed using the Golden-rule hashed value $[148, 149]$ of the candidate VFD entry. The associative memory returns a set of one or more indexes into $\mathcal{H}^+$ which are then checked for duplicate entries. Candidate entries which are not already present in the hash table are added to $\mathcal{H}^+$ and their indices into $\mathcal{H}^+$ are captured into $\mathcal{A}_n$. During the construction phase of the VFD table, the hash table’s sole purpose is to prevent duplicate VFD entries at a particular nonlinear order $n$, but it has further use during spectrum mapping, so it is stored and reinitialized when the construction method concludes at a given $n$.

3.4.5.4 Final Form of VFD Table

When construction of $\mathcal{H}^+$ for non-negative frequencies has been completed for all $n$, $\mathcal{H}^+$ is then used to create the final form of the VFD table in two sort orders. Depending upon the user’s selection, single-sided VFD table or double-sided VFD tables are returned.

The first sort order is that of increasing 1-norm, which is the form that was shown in Sec 3.3.4. This is also the form in which $\mathcal{H}^+$ was built, so all that is necessary is to deal with the $\mathbf{0}^\top$ vector and whether to return a single- or double-sided table. The single-sided table with the 1-norm sort order is given by

$$N_{\mathcal{H}} = \begin{bmatrix} \mathbf{0}^\top \\ \mathcal{H}^+ \end{bmatrix}.$$  \hspace{1cm} (3.170)

For the double-sided table with 1-norm sort order, it is necessary to create the negative-frequency portion of the table. This is done by using the *Additive Inverse* property of Def-
\[ H^- = -JH^+, \]  

where \( J \) denotes the exchange matrix shown in Figure 3.16, which effects a flip of the row contents of \( H^+ \) when it left-multiplies \( H^+ \). The double-sided table with 1-norm sort order is then given by

\[
N_H = \begin{bmatrix}
H^- \\
0^T \\
H^+
\end{bmatrix}.
\]  

The second sort order is denoted by \( \mathcal{F}_H \) and sorts the VFDs in \( H^+ \) by increasing frequency. For the single-sided table the \( 0^T \) vector is prepended so that it has the same form as (3.170). For the double-sided version, the frequency-sorted \( H^+ \) is negated as in (3.171) and the double-sided version in final form is similar to (3.172).

### 3.4.5.5 VFD Table Construction Algorithm

Procedure `CONSTRUCT-VFD` on page 79 describes the algorithm documented in the previous sections. The procedure `INITIALIZE-HASH` on line 6 is not described in depth as it is not central to creating the VFD table, but it essentially creates a table of sets of VFD addresses, where each set is addressed by the hash value of the VFD. Each set is initialized to a given size and
each address is initialized to nil (an invalid VFD address by construction), which eases the identification of insertion points. Hashing collisions are allowed by design, and sets may grow larger than the given size, if necessary.
CONSTRUCT-VFD($Q, N, f_{in}, \text{numsides}$)

1. if $Q \neq \text{length}[f_{in}]$ error
2. $\mathcal{H}^+ = \text{ALLOCATE}(Q, N) \triangleright$ See Section 3.4.5.1
3. $\mathcal{H}^+[\text{Entries 1 to } Q] \leftarrow \text{I} \triangleright$ First $Q$ entries are identity
4. for $n \leftarrow 2..N$
5. do
6. $\mathcal{A}_n = \text{INITIALIZE-HASH}(Q, n) \triangleright$ See text
7. $\mathcal{H}_{(n-1)}^+ \leftarrow \mathcal{H}^+[\text{all entries with 1-norm } n - 1]$
8. $B \leftarrow \text{length}[^{\mathcal{H}_{(n-1)}^+}] \mod Q + 1 \triangleright$ Just $Q$ if exact
9. for $b \leftarrow 1..B$
10. do ▶ Create block matrix $b$
11. $\mathcal{H}_{(n-1)b}^+ \leftarrow \mathcal{H}^+[Q \times Q \text{ block } b]$
12. for $q \leftarrow 1..Q$
13. do ▶ $+P$ through $Q$ shifts
14. $\mathcal{H}_{nc} \leftarrow \mathcal{H}_{(n-1)b}^+ \pm P^q$
15. for $m \leftarrow 1..Q$
16. do ▶ Check each row entry of $\mathcal{H}_{nc}$
17. $\eta_c^T \leftarrow \mathcal{H}_{nc}[\text{Row } m]$
18. if $\|\eta_c^T\|_1 \neq n$ break ▶ 1-norm ok?
19. if $\eta_c^T \cdot f_{in} < 0$ break ▶ freq > 0?
20. $V_a \leftarrow \mathcal{A}_n[\text{hash of } \eta_c^T]$
21. if $\eta_c^T \in \mathcal{H}^+[V_a]$ break ▶ Dupe or not?
22. ▶ Passed, add to VFD Table; update hash table
23. $\mathcal{H}^+[\text{next-open-addr}] \leftarrow \eta_c^T$
24. $\mathcal{A}_n[\text{hash of } \eta_c^T] \leftarrow \text{next-open-addr}$
25. end
26. end
27. $\mathcal{A} \leftarrow \mathcal{A} \cup \mathcal{A}_n$ ▶ Save $\mathcal{A}_n$ into a set of hash tables
28. end
29. nfreq $\leftarrow \mathcal{H}^+ \cdot f_{in}$, $\text{nsort} \leftarrow [1: \text{length}[^{\mathcal{H}^+}]]$
30. $[\mathcal{F}_\mathcal{H}, \text{ffreq}, \text{fsort}] \leftarrow \text{SORTROWS}[\mathcal{H}^+, \text{nfreq}, \text{nsort}] \triangleright$ Matlab sortrows cmd, sort on nfreq
31. if $\text{numsides} = 1$
32. then ▶ One-sided spectrum
33. $\mathcal{N}_\mathcal{H} \leftarrow \mathbf{0}^T \cup \mathcal{H}^+$, $\text{nfreq} \leftarrow 0 \cup \text{nfreq}$, $\mathcal{F}_\mathcal{H} \leftarrow \mathbf{0}^T \cup \mathcal{F}_\mathcal{H}$, $\text{ffreq} \leftarrow 0 \cup \text{ffreq}$
34. $\text{nfidx} \leftarrow 0 \cup \text{nsort}$, $\text{ffidx} \leftarrow 0 \cup \text{fsort}$
35. else ▶ Two-sided spectrum
36. $\mathcal{N}_\mathcal{H} \leftarrow \text{FLIPUD}[\mathcal{H}^+] \cup \mathbf{0}^T \cup \mathcal{H}^+$, $\text{nfreq} \leftarrow \text{FLIPUD}[\text{nfreq}] \cup 0 \cup \text{nfreq}$
37. $\mathcal{F}_\mathcal{H} \leftarrow \text{FLIPUD}[\mathcal{F}_\mathcal{H}] \cup \mathbf{0}^T \cup \mathcal{F}_\mathcal{H}$, $\text{ffreq} \leftarrow \text{FLIPUD}[\text{ffreq}] \cup 0 \cup \text{ffreq}$
38. $\text{nfidx} \leftarrow \text{FLIPUD}[\text{nsort}] \cup 0 \cup \text{nsort}$, $\text{ffidx} \leftarrow \text{FLIPUD}[\text{fsort}] \cup 0 \cup \text{fsort}$
39. end
40. return $[\mathcal{F}_\mathcal{H}, \text{ffreq}, \text{ffidx}, \mathcal{N}_\mathcal{H}, \text{nfreq}, \text{nfidx}]$
3.4.5.6 Matlab Implementation Details

The 1-norm sorted and frequency sorted VFD tables returned by Procedure \textsc{Construct-VFD} has VFD entries as row elements as indicated. However, during the construction phase of $\mathcal{H}^+$ (from lines 3–29 on page 79), $\mathcal{H}^+$ is actually built in a transpose form, with valid candidate VFD entries being added as columns rather than rows. This is done because Matlab\textsuperscript{®} stores matrices internally in column major order\cite{151, 152}, so that adding a column to $\mathcal{H}^+$ during construction merely appends another column to the end of a linear data structure. The alternative of maintaining $\mathcal{H}^+$ in row form during construction (which was the case during the early phases of development of the code) results in memory thrashing, since adding an extra row to $\mathcal{H}^+$ forces a re-allocation of the entire VFD Table, with insertion of individual column tuples at each of $Q$ non-consecutive locations. Such memory thrashing was transparent for small problem sizes, but became apparent for larger problems. The Matlab\textsuperscript{®} profiler\cite{151, 153} proved to be a crucial tool for detecting and remedying the problem.

Finally, it is noted here that the tuples of a VFD entry are stored as 8-bit integer quantities in the Matlab\textsuperscript{®} implementation. This choice facilitates the use of inputs with a large number of tones, but since 8-bit arithmetic in Matlab is limited to a range of $-128 \ldots 127$\cite{154}, then as a practical matter the magnitude of any tuple in a VFD entry is limited to 63 in order to prevent arithmetic overflow. In turn, this implies a maximum nonlinear order $N$ of 63. Presently, most researchers in the discipline of microwave circuits are concerned with orders of nonlinearity much lower than this. For example, Gard\cite{155} showed that extracted polynomial transfer functions with a maximum nonlinear order of 23 were sufficient for modeling the response of MESFETs to large-signal inputs.

3.4.6 3-tone VFD Construction Example

Consider the case of an input signal composed of three sinusoids at frequencies $f_{in} = [f_1 \ f_2 \ f_3]^T$, where $f_2 = f_1 + \Delta f$, and $f_3 = f_1 + 2\Delta f$ so that $f_1 < f_2 < f_3$ and also $f_1 >> 2\Delta f$. Let the input signal be passed through a nonlinear system having linear, second order, and third order response characteristics, so the maximal nonlinear order is $N = 3$. This example is somewhat more involved than the introductory example given in Sections 3.3.2 and 3.3.4 but it will have further applications. The intricate details of the algorithm execution may be skipped without loss of continuity by moving ahead to Section 3.4.6.4 on page 89.
3.4.6.1 Linear Response

Construction of the positive-frequency portion of the VFD Table, $\mathcal{H}^+$, begins by entering a $3 \times 3$ identity matrix into the table:

\[
\mathcal{H}^+ = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix} \quad \triangleright \text{These are the linear response VFD entries.} \quad (3.173)
\]

3.4.6.2 Second Order Response

Next, all of the second order VFD entries are identified by adding and subtracting the upper circulant shifted identity matrix through to the linear response. Candidate VFDs are referred to in the running commentary hereafter as $\eta_T^c$, and valid candidate VFDs are noted by $\checkmark$ symbols in the comments. First, $P$ is added and subtracted:

\[
\begin{align*}
\mathbf{I} + P &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \quad \triangleright \| \eta_T^c \|_1 \checkmark \\
\mathbf{I} - P &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ -1 & 0 & 1 \end{bmatrix} \quad \triangleright \eta_T^c \cdot f_{\text{in}} = -\Delta f < 0, \text{invalid} \quad (3.174) \\
& \quad \triangleright \eta_T^c \cdot f_{\text{in}} = -\Delta f < 0, \text{invalid} \quad \triangleright \eta_T^c \cdot f_{\text{in}} = 2\Delta f > 0 \checkmark
\end{align*}
\]

After adding the valid candidate VFDs from (3.174), the VFD Table so far is:

\[
\mathcal{H}^+ = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
1 & 1 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
-1 & 0 & 1
\end{bmatrix} \quad \triangleright \text{New VFD entry} \quad (3.175)
\]
Second, $P^2$ is added and subtracted to the linear response and any valid candidate VFDs are identified:

$$I + P^2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \triangleright \text{Already in table}$$

$$I - P^2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \\ -1 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \triangleright \eta_c^T \cdot f_{\text{in}} = -2\Delta f < 0, \text{invalid}$$

(3.176)

After adding the valid candidate VFDs from (3.176), the VFD Table so far is:

$$\mathcal{H}^+ = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ -1 & 0 & 0 \end{bmatrix} \triangleright \text{New VFD entry}$$

(3.177)

Third, $P^3 = I$ is added and subtracted to the linear response and any valid candidate VFDs are identified:

$$I + P^3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix} \triangleright \|\eta_c^T\|_1 = 2 \checkmark$$

(3.178)
After adding the valid candidate VFDs from (3.178), the VFD Table including the linear and 2\textsuperscript{nd} order responses is:

\[
\begin{array}{cccc}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
1 & 1 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
-1 & 0 & 1 \\
-1 & 1 & 0 \\
0 & -1 & 1 \\
2 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 2 \\
\end{array}
\]

\(\mathcal{H}^+ = \) (3.179)

3.4.6.3 Third Order Response

Next, all of the third order VFD entries are identified by adding and subtracting the upper circulant shifted identity matrix to the second order response entries in the VFD Table. By inspection of (3.179), it can be seen that \(\mathcal{H}_2^+\) is given by

\[
\begin{array}{cccc}
1 & 1 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
-1 & 0 & 1 \\
-1 & 1 & 0 \\
0 & -1 & 1 \\
2 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 2 \\
\end{array}
\]

\(\mathcal{H}_2^+ = \) (3.180)
$\mathcal{H}_2^+$ has 9 entries, so blocking it into $3 \times 3$ matrices yields:

$$
\mathcal{H}_{2,1b}^+ = \begin{bmatrix} 1 & 1 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \end{bmatrix}, \quad \mathcal{H}_{2,2b}^+ = \begin{bmatrix} -1 & 0 & 1 \\
-1 & 1 & 0 \\
0 & -1 & 1 \end{bmatrix}, \quad \mathcal{H}_{2,3b}^+ = \begin{bmatrix} 2 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 2 \end{bmatrix}.
$$  \hspace{1cm} (3.181)

Beginning with $\mathcal{H}_{2,1b}^+$, the algorithm executes by adding and subtracting the upper-circulant shifted identity matrix through three shifts. For $P$:

- $\mathcal{H}_{2,1b}^+ + P =$
  $$
  \begin{bmatrix} 1 & 1 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 \\
0 & 0 & 1 \\
1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 0 \\
0 & 1 & 2 \\
2 & 0 & 1 \end{bmatrix} \triangleright ||\eta_c^T||_1 = 3 \checkmark
  $$

- $\mathcal{H}_{2,1b}^+ - P =$
  $$
  \begin{bmatrix} 1 & 1 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 \\
0 & 0 & 1 \\
1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \end{bmatrix} \triangleright ||\eta_c^T||_1 = 1 \neq 3, \text{ invalid}
  $$

For $P^2$:

- $\mathcal{H}_{2,1b}^+ + P^2 =$
  $$
  \begin{bmatrix} 1 & 1 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \end{bmatrix} \triangleright \text{ Already in table}
  $$

- $\mathcal{H}_{2,1b}^+ - P^2 =$
  $$
  \begin{bmatrix} 1 & 1 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & -1 & -1 \\
-1 & 1 & 1 \\
1 & -1 & 1 \end{bmatrix} \triangleright \eta_c^T \cdot f_{in} = f_1 - \Delta f > 0 \checkmark
  $$

$$
\begin{align}
\mathcal{H}_{2,1b}^+ + P^2 = & \begin{bmatrix} 1 & 1 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \end{bmatrix} \triangleright ||\eta_c^T||_1 = 3 \checkmark \\
\mathcal{H}_{2,1b}^+ - P^2 = & \begin{bmatrix} 1 & 1 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & -1 & -1 \\
-1 & 1 & 1 \\
1 & -1 & 1 \end{bmatrix} \triangleright \eta_c^T \cdot f_{in} = f_1 - \Delta f > 0 \checkmark
\end{align}
$$
For $P^3$:

$$\mathcal{H}^+_{2,1b} + P^3 =$$

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 0 \\ 0 & 2 & 1 \\ 1 & 0 & 2 \end{bmatrix} \triangleright \|
\eta^T_c\|_1 = 3 \checkmark$$

$$\mathcal{H}^+_{2,1b} - P^3 =$$

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \triangleright \|
\eta^T_c\|_1 = 1 \neq 3, \text{ invalid} \quad (3.184)$$

After adding the valid candidate VFDs from $(3.182)$–$(3.184)$, the VFD Table is:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 1 & 0 \\ 0 & -1 & 1 \\ 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \\ 1 & 2 & 0 \\ 0 & 1 & 2 \\ 2 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & -1 \\ -1 & 1 & 1 \\ -1 & -1 & 1 \\ 1 & -1 & 1 \\ 2 & 1 & 0 \\ 0 & 2 & 1 \\ 1 & 0 & 2 \end{bmatrix} \triangleright \text{New VFD entry}$$
Moving to the second block matrix, $\mathcal{H}_{2,2b}^+$, the algorithm repeats. For $\mathbf{P}$:

$$\mathcal{H}_{2,2b}^+ + \mathbf{P} =$$

$$\begin{bmatrix}
-1 & 0 & 1 \\ -1 & 1 & 0 \\ 0 & -1 & 1
\end{bmatrix}
+ \begin{bmatrix}
0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0
\end{bmatrix}
= \begin{bmatrix}
-1 & 1 & 1 \\ -1 & 1 & 1 \\ 1 & -1 & 1
\end{bmatrix} \triangleright \text{Already in table}
$$

$$\mathcal{H}_{2,2b}^+ - \mathbf{P} =$$

$$\begin{bmatrix}
-1 & 0 & 1 \\ -1 & 1 & 0 \\ 0 & -1 & 1
\end{bmatrix}
- \begin{bmatrix}
0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0
\end{bmatrix}
= \begin{bmatrix}
-1 & -1 & 1 \\ -1 & 1 & -1 \\ -1 & -1 & 1
\end{bmatrix} \triangleright \|\eta_c^T\|_1 = 3 \checkmark
\begin{bmatrix}
-1 & 0 & 1 \\ -1 & 1 & 0 \\ 0 & -1 & 1
\end{bmatrix}
- \begin{bmatrix}
0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0
\end{bmatrix}
= \begin{bmatrix}
0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0
\end{bmatrix} \triangleright \|\eta_c^T\|_1 = 1 \neq 3, \text{invalid}
$$

For $\mathbf{P}^2$:

$$\mathcal{H}_{2,2b}^+ + \mathbf{P}^2 =$$

$$\begin{bmatrix}
-1 & 0 & 1 \\ -1 & 1 & 0 \\ 0 & -1 & 1
\end{bmatrix}
+ \begin{bmatrix}
0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0
\end{bmatrix}
= \begin{bmatrix}
-1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1
\end{bmatrix} \triangleright \|\eta_c^T\|_1 \neq 3, \text{invalid}
$$

$$\mathcal{H}_{2,2b}^+ - \mathbf{P}^2 =$$

$$\begin{bmatrix}
-1 & 0 & 1 \\ -1 & 1 & 0 \\ 0 & -1 & 1
\end{bmatrix}
- \begin{bmatrix}
0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0
\end{bmatrix}
= \begin{bmatrix}
-1 & 0 & -2 \\ -2 & 1 & 0 \\ 0 & -2 & 1
\end{bmatrix} \triangleright \eta_c^T \cdot \mathbf{f}_{\text{in}} = -f_1 - 2\Delta f < 0, \text{invalid}
$$

$$(3.186)$$
For $P^3$:

$$\mathcal{H}_{2,2b}^+ + P^3 = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 2 & 0 \\ 0 & -1 & 2 \end{bmatrix} \triangleright \| \eta_c^T \|_1 = 1 \neq 3, \text{ invalid} \quad \triangleright \checkmark$$

$$\mathcal{H}_{2,2b}^- - P^3 = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} - \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -2 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \triangleright \| \eta_c^T \|_1 = 1 \neq 3, \text{ invalid} \quad \triangleright \checkmark$$

\begin{align*}
(3.188)
\end{align*}

After adding the valid candidate VFDs from (3.186)–(3.188), the VFD Table is:

$$\mathcal{H}^+ = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 1 & 0 \\ 0 & -1 & 1 \\ 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \\ 1 & 2 & 0 \\ 0 & 1 & 2 \\ 2 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & -1 \\ -1 & 1 & 1 \\ 1 & -1 & 1 \\ 2 & 1 & 0 \\ 0 & 2 & 1 \\ 1 & 0 & 2 \\ -1 & 0 & 2 \end{bmatrix} \triangleright \text{New VFD entry}$$

$$\begin{bmatrix} -1 & 2 & 0 \end{bmatrix} \triangleright \text{New VFD entry}$$

$$\begin{bmatrix} 0 & -1 & 2 \end{bmatrix} \triangleright \text{New VFD entry}$$

(3.189)
Moving to the third block matrix, $\mathcal{H}_{2,3b}^+$, the algorithm repeats. For $\mathbf{P}$:

$$\mathcal{H}_{2,3b}^+ + \mathbf{P} =$$

$$\begin{bmatrix}
2 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 2
\end{bmatrix} +
\begin{bmatrix}
0 & 1 & 0 \\
0 & 0 & 1 \\
1 & 0 & 0
\end{bmatrix} =
\begin{bmatrix}
2 & 1 & 0 \\
0 & 2 & 1 \\
1 & 0 & 2
\end{bmatrix} \triangleright \text{Already in table}
$$

$$\mathcal{H}_{2,3b}^+ - \mathbf{P} =$$

$$\begin{bmatrix}
2 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 2
\end{bmatrix} -
\begin{bmatrix}
0 & 1 & 0 \\
0 & 0 & 1 \\
1 & 0 & 0
\end{bmatrix} =
\begin{bmatrix}
2 & -1 & 0 \\
0 & 2 & -1 \\
-1 & 0 & 2
\end{bmatrix} \triangleright \text{Already in table}
$$

For $\mathbf{P}^2$:

$$\mathcal{H}_{2,3b}^+ + \mathbf{P}^2 =$$

$$\begin{bmatrix}
2 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 2
\end{bmatrix} +
\begin{bmatrix}
0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0
\end{bmatrix} =
\begin{bmatrix}
2 & 0 & -1 \\
1 & 2 & 0 \\
0 & 1 & 2
\end{bmatrix} \triangleright \text{Already in table}
$$

$$\mathcal{H}_{2,3b}^+ - \mathbf{P}^2 =$$

$$\begin{bmatrix}
2 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 2
\end{bmatrix} -
\begin{bmatrix}
0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0
\end{bmatrix} =
\begin{bmatrix}
2 & 0 & -1 \\
-1 & 2 & 0 \\
0 & -1 & 2
\end{bmatrix} \triangleright \text{Already in table}
$$

(3.190)

For $\mathbf{P}^3$:

$$\mathcal{H}_{2,3b}^+ + \mathbf{P}^3 =$$

$$\begin{bmatrix}
2 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 2
\end{bmatrix} +
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix} =
\begin{bmatrix}
3 & 0 & 0 \\
0 & 3 & 0 \\
0 & 0 & 3
\end{bmatrix} \triangleright \|\mathbf{\eta}_c^T\|_1 = 3 \checkmark$$

$$\mathcal{H}_{2,3b}^+ - \mathbf{P}^3 =$$

$$\begin{bmatrix}
2 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 2
\end{bmatrix} -
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix} =
\begin{bmatrix}
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0
\end{bmatrix} \triangleright \|\mathbf{\eta}_c^T\|_1 = 0 \neq 3, \text{invalid}
$$

(3.191)

(3.192)
After adding the valid candidate VFDs from (3.190)–(3.192), the VFD Table is:

\[
H^+ = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
1 & 1 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
-1 & 0 & 1 \\
-1 & 1 & 0 \\
0 & -1 & 1 \\
2 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 2 \\
1 & 2 & 0 \\
0 & 1 & 2 \\
2 & 0 & 1 \\
1 & 1 & 1 \\
1 & 1 & -1 \\
-1 & 1 & 1 \\
1 & -1 & 1 \\
2 & 1 & 0 \\
0 & 2 & 1 \\
1 & 0 & 2 \\
-1 & 0 & 2 \\
-1 & 2 & 0 \\
0 & -1 & 2 \\
2 & -1 & 0 & \triangleright \text{New VFD entry} \\
0 & 2 & -1 & \triangleright \text{New VFD entry} \\
2 & 0 & -1 & \triangleright \text{New VFD entry} \\
3 & 0 & 0 & \triangleright \text{New VFD entry} \\
0 & 3 & 0 & \triangleright \text{New VFD entry} \\
0 & 0 & 3 & \triangleright \text{New VFD entry}
\end{bmatrix}
\]

(3.193)

At this point, \(H^+\) is close to \(H\) in its final single-sided form. All that remains is to prepend the VFD for DC, \(\eta_0^T = [0 \ 0 \ 0]\), to \(H^+\).

### 3.4.6.4 3-tone VFD in Final Form

The complete one-sided VFD Table, in 1-norm sort order, for 3 tones through a 3rd order nonlinearity developed in the previous sections is shown in Table 3.3.
Table 3.3: 1-Norm Sorted 1-Side VFD Table for 3 Tones in a 3rd Order Non-linearity

<table>
<thead>
<tr>
<th>Frequency Index $k_x$</th>
<th>VFD $(k_y, k_z)$</th>
<th>1-Norm $|\eta^T_k|_1$</th>
<th>$f_1$ Weight $\eta_{k,1}$</th>
<th>$f_2$ Weight $\eta_{k,2}$</th>
<th>$f_3$ Weight $\eta_{k,3}$</th>
<th>Output Frequency (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\eta^T_0$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>$\eta^T_1$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$f_1$</td>
</tr>
<tr>
<td>2</td>
<td>$\eta^T_2$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$f_1 + \Delta f$</td>
</tr>
<tr>
<td>3</td>
<td>$\eta^T_3$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$f_1 + 2\Delta f$</td>
</tr>
<tr>
<td>4</td>
<td>$\eta^T_4$</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$2f_1 + \Delta f$</td>
</tr>
<tr>
<td>5</td>
<td>$\eta^T_5$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$2f_1 + 2\Delta f$</td>
</tr>
<tr>
<td>6</td>
<td>$\eta^T_6$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$2f_1 + 2\Delta f$</td>
</tr>
<tr>
<td>7</td>
<td>$\eta^T_7$</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$2\Delta f$</td>
</tr>
<tr>
<td>8</td>
<td>$\eta^T_8$</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$\Delta f$</td>
</tr>
<tr>
<td>9</td>
<td>$\eta^T_9$</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$\Delta f$</td>
</tr>
<tr>
<td>10</td>
<td>$\eta^T_{10}$</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$2f_1$</td>
</tr>
<tr>
<td>11</td>
<td>$\eta^T_{11}$</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>$2f_1 + 2\Delta f$</td>
</tr>
<tr>
<td>12</td>
<td>$\eta^T_{12}$</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>$2f_1 + 4\Delta f$</td>
</tr>
<tr>
<td>13</td>
<td>$\eta^T_{13}$</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>$3f_1 + 2\Delta f$</td>
</tr>
<tr>
<td>14</td>
<td>$\eta^T_{14}$</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>$3f_1 + 5\Delta f$</td>
</tr>
<tr>
<td>15</td>
<td>$\eta^T_{15}$</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$3f_1 + 2\Delta f$</td>
</tr>
<tr>
<td>16</td>
<td>$\eta^T_{16}$</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$3f_1 + 3\Delta f$</td>
</tr>
<tr>
<td>17</td>
<td>$\eta^T_{17}$</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$f_1 - \Delta f$</td>
</tr>
<tr>
<td>18</td>
<td>$\eta^T_{18}$</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$f_1 + \Delta f$</td>
</tr>
<tr>
<td>19</td>
<td>$\eta^T_{19}$</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$f_1 + 2\Delta f$</td>
</tr>
<tr>
<td>20</td>
<td>$\eta^T_{20}$</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>$3f_1 + \Delta f$</td>
</tr>
<tr>
<td>21</td>
<td>$\eta^T_{21}$</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>$3f_1 + 2\Delta f$</td>
</tr>
<tr>
<td>22</td>
<td>$\eta^T_{22}$</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$3f_1 + 3\Delta f$</td>
</tr>
<tr>
<td>23</td>
<td>$\eta^T_{23}$</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$f_1 + 4\Delta f$</td>
</tr>
<tr>
<td>24</td>
<td>$\eta^T_{24}$</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$f_1 + 2\Delta f$</td>
</tr>
<tr>
<td>25</td>
<td>$\eta^T_{25}$</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$f_1 + 3\Delta f$</td>
</tr>
</tbody>
</table>

Continued on next page
The double-sided form of Table 3.3 is shown in Table 3.4.

Table 3.4: 1-Norm Sort 2-Side VFD Table for 3 Tones in a 3rd Order Nonlinearity

<table>
<thead>
<tr>
<th>Frequency Index $k_x$ $(k_y, k_z)$</th>
<th>VFD $\eta^T_{k_x}$</th>
<th>1-Norm $|\eta^T_{k_x}|_1$</th>
<th>$f_1$ Weight $\eta_{k,1}$</th>
<th>$f_2$ Weight $\eta_{k,2}$</th>
<th>$f_3$ Weight $\eta_{k,3}$</th>
<th>Output Frequency (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>$\eta^T_{26}$</td>
<td>3</td>
<td>2</td>
<td>-1</td>
<td>0</td>
<td>$f_1 - \Delta f$</td>
</tr>
<tr>
<td>27</td>
<td>$\eta^T_{27}$</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>-1</td>
<td>$f_1$</td>
</tr>
<tr>
<td>28</td>
<td>$\eta^T_{28}$</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>-1</td>
<td>$f_1 - 2\Delta f$</td>
</tr>
<tr>
<td>29</td>
<td>$\eta^T_{29}$</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>$3f_1$</td>
</tr>
<tr>
<td>30</td>
<td>$\eta^T_{30}$</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>$3f_1 + 3\Delta f$</td>
</tr>
<tr>
<td>31</td>
<td>$\eta^T_{31}$</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>$3f_1 + 6\Delta f$</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Frequency Index $k_x$ $(k_y, k_z)$</th>
<th>VFD $\eta_{T}^k$</th>
<th>1- \norm{\eta_{k}^T\eta_{k}}_1</th>
<th>$f_1$ Weight $\eta_{k,1}$</th>
<th>$f_2$ Weight $\eta_{k,2}$</th>
<th>$f_3$ Weight $\eta_{k,3}$</th>
<th>Output Frequency (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-20$ $\eta_{T-20}$</td>
<td>3</td>
<td>$-2$</td>
<td>$-1$</td>
<td>$0$</td>
<td>$-3f_1 - \Delta f$</td>
<td></td>
</tr>
<tr>
<td>$-19$ $\eta_{T-19}$</td>
<td>3</td>
<td>$-1$</td>
<td>$1$</td>
<td>$-1$</td>
<td>$-f_1 - \Delta f$</td>
<td></td>
</tr>
<tr>
<td>$-18$ $\eta_{T-18}$</td>
<td>3</td>
<td>$1$</td>
<td>$-1$</td>
<td>$-1$</td>
<td>$-f_1 - 3\Delta f$</td>
<td></td>
</tr>
<tr>
<td>$-17$ $\eta_{T-17}$</td>
<td>3</td>
<td>$-1$</td>
<td>$-1$</td>
<td>$1$</td>
<td>$-f_1 + \Delta f$</td>
<td></td>
</tr>
<tr>
<td>$-16$ $\eta_{T-16}$</td>
<td>3</td>
<td>$-1$</td>
<td>$-1$</td>
<td>$-1$</td>
<td>$-3f_1 - 3\Delta f$</td>
<td></td>
</tr>
<tr>
<td>$-15$ $\eta_{T-15}$</td>
<td>3</td>
<td>$-2$</td>
<td>$0$</td>
<td>$-1$</td>
<td>$-3f_1 - 2\Delta f$</td>
<td></td>
</tr>
<tr>
<td>$-14$ $\eta_{T-14}$</td>
<td>3</td>
<td>$0$</td>
<td>$-1$</td>
<td>$-2$</td>
<td>$-3f_1 - 5\Delta f$</td>
<td></td>
</tr>
<tr>
<td>$-13$ $\eta_{T-13}$</td>
<td>3</td>
<td>$-1$</td>
<td>$-2$</td>
<td>$0$</td>
<td>$-3f_1 - 2\Delta f$</td>
<td></td>
</tr>
<tr>
<td>$-12$ $\eta_{T-12}$</td>
<td>2</td>
<td>$0$</td>
<td>$0$</td>
<td>$-2$</td>
<td>$-2f_1 - 4\Delta f$</td>
<td></td>
</tr>
<tr>
<td>$-11$ $\eta_{T-11}$</td>
<td>2</td>
<td>$0$</td>
<td>$-2$</td>
<td>$0$</td>
<td>$-2f_1 - 2\Delta f$</td>
<td></td>
</tr>
<tr>
<td>$-10$ $\eta_{T-10}$</td>
<td>2</td>
<td>$-2$</td>
<td>$0$</td>
<td>$0$</td>
<td>$-2f_1$</td>
<td></td>
</tr>
<tr>
<td>$-9$ $\eta_{T-9}$</td>
<td>2</td>
<td>$0$</td>
<td>$1$</td>
<td>$-1$</td>
<td>$-\Delta f$</td>
<td></td>
</tr>
<tr>
<td>$-8$ $\eta_{T-8}$</td>
<td>2</td>
<td>$1$</td>
<td>$-1$</td>
<td>$0$</td>
<td>$-\Delta f$</td>
<td></td>
</tr>
<tr>
<td>$-7$ $\eta_{T-7}$</td>
<td>2</td>
<td>$1$</td>
<td>$0$</td>
<td>$-1$</td>
<td>$-2\Delta f$</td>
<td></td>
</tr>
<tr>
<td>$-6$ $\eta_{T-6}$</td>
<td>2</td>
<td>$-1$</td>
<td>$0$</td>
<td>$-1$</td>
<td>$-2f_1 - 2\Delta f$</td>
<td></td>
</tr>
<tr>
<td>$-5$ $\eta_{T-5}$</td>
<td>2</td>
<td>$0$</td>
<td>$-1$</td>
<td>$-1$</td>
<td>$-2f_1 - 3\Delta f$</td>
<td></td>
</tr>
<tr>
<td>$-4$ $\eta_{T-4}$</td>
<td>2</td>
<td>$-1$</td>
<td>$-1$</td>
<td>$0$</td>
<td>$-2f_1 - \Delta f$</td>
<td></td>
</tr>
<tr>
<td>$-3$ $\eta_{T-3}$</td>
<td>1</td>
<td>$0$</td>
<td>$0$</td>
<td>$1$</td>
<td>$-f_1 - 2\Delta f$</td>
<td></td>
</tr>
<tr>
<td>$-2$ $\eta_{T-2}$</td>
<td>1</td>
<td>$0$</td>
<td>$-1$</td>
<td>$0$</td>
<td>$-f_1 - \Delta f$</td>
<td></td>
</tr>
<tr>
<td>$-1$ $\eta_{T-1}$</td>
<td>1</td>
<td>$-1$</td>
<td>$0$</td>
<td>$0$</td>
<td>$-f_1$</td>
<td></td>
</tr>
<tr>
<td>0 $\eta_{T0}$</td>
<td>0</td>
<td>$0$</td>
<td>$0$</td>
<td>$0$</td>
<td>$0$</td>
<td></td>
</tr>
<tr>
<td>1 $\eta_{T1}$</td>
<td>1</td>
<td>$1$</td>
<td>$0$</td>
<td>$0$</td>
<td>$f_1$</td>
<td></td>
</tr>
<tr>
<td>2 $\eta_{T2}$</td>
<td>1</td>
<td>$0$</td>
<td>$1$</td>
<td>$0$</td>
<td>$f_1 + \Delta f$</td>
<td></td>
</tr>
<tr>
<td>3 $\eta_{T3}$</td>
<td>1</td>
<td>$0$</td>
<td>$0$</td>
<td>$1$</td>
<td>$f_1 + 2\Delta f$</td>
<td></td>
</tr>
<tr>
<td>4 $\eta_{T4}$</td>
<td>2</td>
<td>$1$</td>
<td>$1$</td>
<td>$0$</td>
<td>$2f_1 + \Delta f$</td>
<td></td>
</tr>
<tr>
<td>5 $\eta_{T5}$</td>
<td>2</td>
<td>$0$</td>
<td>$1$</td>
<td>$1$</td>
<td>$2f_1 + 3\Delta f$</td>
<td></td>
</tr>
<tr>
<td>6 $\eta_{T6}$</td>
<td>2</td>
<td>$1$</td>
<td>$0$</td>
<td>$1$</td>
<td>$2f_1 + 2\Delta f$</td>
<td></td>
</tr>
</tbody>
</table>
Table 3.4 (continued)

<table>
<thead>
<tr>
<th>Frequency Index $k_x$</th>
<th>VFD $\eta^T_k$</th>
<th>1-Norm $|\eta^T_k|_1$</th>
<th>$f_1$ Weight $\eta_{k,1}$</th>
<th>$f_2$ Weight $\eta_{k,2}$</th>
<th>$f_3$ Weight $\eta_{k,3}$</th>
<th>Output Frequency $\Delta f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>$\eta^T_7$</td>
<td>2</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>$2\Delta f$</td>
</tr>
<tr>
<td>8</td>
<td>$\eta^T_8$</td>
<td>2</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>$\Delta f$</td>
</tr>
<tr>
<td>9</td>
<td>$\eta^T_9$</td>
<td>2</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>$\Delta f$</td>
</tr>
<tr>
<td>10</td>
<td>$\eta^T_{10}$</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>$2f_1$</td>
</tr>
<tr>
<td>11</td>
<td>$\eta^T_{11}$</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>$2f_1 + 2\Delta f$</td>
</tr>
<tr>
<td>12</td>
<td>$\eta^T_{12}$</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>$2f_1 + 4\Delta f$</td>
</tr>
<tr>
<td>13</td>
<td>$\eta^T_{13}$</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>$3f_1 + 2\Delta f$</td>
</tr>
<tr>
<td>14</td>
<td>$\eta^T_{14}$</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>$3f_1 + 5\Delta f$</td>
</tr>
<tr>
<td>15</td>
<td>$\eta^T_{15}$</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>$3f_1 + 2\Delta f$</td>
</tr>
<tr>
<td>16</td>
<td>$\eta^T_{16}$</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$3f_1 + 3\Delta f$</td>
</tr>
<tr>
<td>17</td>
<td>$\eta^T_{17}$</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>$f_1 - \Delta f$</td>
</tr>
<tr>
<td>18</td>
<td>$\eta^T_{18}$</td>
<td>3</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>$f_1 + 3\Delta f$</td>
</tr>
<tr>
<td>19</td>
<td>$\eta^T_{19}$</td>
<td>3</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>$f_1 + \Delta f$</td>
</tr>
<tr>
<td>20</td>
<td>$\eta^T_{20}$</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>$3f_1 + \Delta f$</td>
</tr>
<tr>
<td>21</td>
<td>$\eta^T_{21}$</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>$3f_1 + 4\Delta f$</td>
</tr>
<tr>
<td>22</td>
<td>$\eta^T_{22}$</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>$3f_1 + 4\Delta f$</td>
</tr>
<tr>
<td>23</td>
<td>$\eta^T_{23}$</td>
<td>3</td>
<td>-1</td>
<td>0</td>
<td>2</td>
<td>$f_1 + 4\Delta f$</td>
</tr>
<tr>
<td>24</td>
<td>$\eta^T_{24}$</td>
<td>3</td>
<td>-1</td>
<td>2</td>
<td>0</td>
<td>$f_1 + 2\Delta f$</td>
</tr>
<tr>
<td>25</td>
<td>$\eta^T_{25}$</td>
<td>3</td>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>$f_1 + 3\Delta f$</td>
</tr>
<tr>
<td>26</td>
<td>$\eta^T_{26}$</td>
<td>3</td>
<td>2</td>
<td>-1</td>
<td>0</td>
<td>$f_1 - \Delta f$</td>
</tr>
<tr>
<td>27</td>
<td>$\eta^T_{27}$</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>-1</td>
<td>$f_1$</td>
</tr>
<tr>
<td>28</td>
<td>$\eta^T_{28}$</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>-1</td>
<td>$f_1 - 2\Delta f$</td>
</tr>
<tr>
<td>29</td>
<td>$\eta^T_{29}$</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>$3f_1$</td>
</tr>
<tr>
<td>30</td>
<td>$\eta^T_{30}$</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>$3f_1 + 3\Delta f$</td>
</tr>
<tr>
<td>31</td>
<td>$\eta^T_{31}$</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>$3f_1 + 6\Delta f$</td>
</tr>
</tbody>
</table>

When Table 3.3 is sorted in order of increasing frequency, Table 3.5 is the result. Notice
that there are many instances of differing VFDs having the same numeric output frequency. This is to be expected when commensurate input tones are passed through a nonlinear transfer function. The VFD Table is able to distinguish these unique mixtures of inputs at the output.

Table 3.5: Frequency Sort 1-Side VFD Table for 3 Tones in a 3rd Order Non-linearity

<table>
<thead>
<tr>
<th>Frequency Index $k_x$ ($k_y, k_z$)</th>
<th>VFD</th>
<th>1-Norm $|\eta_k^T|_1$</th>
<th>$f_1$ Weight $\eta_{k,1}$</th>
<th>$f_2$ Weight $\eta_{k,2}$</th>
<th>$f_3$ Weight $\eta_{k,3}$</th>
<th>Output Frequency (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\eta_0^T$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>$\eta_8^T$</td>
<td>2</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>$\Delta f$</td>
</tr>
<tr>
<td>9</td>
<td>$\eta_9^T$</td>
<td>2</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>$\Delta f$</td>
</tr>
<tr>
<td>7</td>
<td>$\eta_7^T$</td>
<td>2</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>$2\Delta f$</td>
</tr>
<tr>
<td>28</td>
<td>$\eta_{28}^T$</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>-1</td>
<td>$f_1 - 2\Delta f$</td>
</tr>
<tr>
<td>17</td>
<td>$\eta_{17}^T$</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>$f_1 - \Delta f$</td>
</tr>
<tr>
<td>26</td>
<td>$\eta_{26}^T$</td>
<td>3</td>
<td>2</td>
<td>-1</td>
<td>0</td>
<td>$f_1 - \Delta f$</td>
</tr>
<tr>
<td>1</td>
<td>$\eta_1^T$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$f_1$</td>
</tr>
<tr>
<td>27</td>
<td>$\eta_{27}^T$</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>-1</td>
<td>$f_1$</td>
</tr>
<tr>
<td>2</td>
<td>$\eta_2^T$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$f_1 + \Delta f$</td>
</tr>
<tr>
<td>19</td>
<td>$\eta_{19}^T$</td>
<td>3</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>$f_1 + \Delta f$</td>
</tr>
<tr>
<td>3</td>
<td>$\eta_3^T$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$f_1 + 2\Delta f$</td>
</tr>
<tr>
<td>24</td>
<td>$\eta_{24}^T$</td>
<td>3</td>
<td>-1</td>
<td>2</td>
<td>0</td>
<td>$f_1 + 2\Delta f$</td>
</tr>
<tr>
<td>18</td>
<td>$\eta_{18}^T$</td>
<td>3</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>$f_1 + 3\Delta f$</td>
</tr>
<tr>
<td>25</td>
<td>$\eta_{25}^T$</td>
<td>3</td>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>$f_1 + 3\Delta f$</td>
</tr>
<tr>
<td>23</td>
<td>$\eta_{23}^T$</td>
<td>3</td>
<td>-1</td>
<td>0</td>
<td>2</td>
<td>$f_1 + 4\Delta f$</td>
</tr>
<tr>
<td>10</td>
<td>$\eta_{10}^T$</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>$2f_1$</td>
</tr>
<tr>
<td>4</td>
<td>$\eta_4^T$</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$2f_1 + \Delta f$</td>
</tr>
<tr>
<td>6</td>
<td>$\eta_6^T$</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$2f_1 + 2\Delta f$</td>
</tr>
<tr>
<td>11</td>
<td>$\eta_{11}^T$</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>$2f_1 + 2\Delta f$</td>
</tr>
<tr>
<td>5</td>
<td>$\eta_5^T$</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>$2f_1 + 3\Delta f$</td>
</tr>
</tbody>
</table>

Continued on next page
The double-sided form of Table 3.3 is shown in Table 3.4.

### Table 3.5 (continued)

<table>
<thead>
<tr>
<th>Frequency Index $k_x$ $(k_y, k_z)$</th>
<th>VFD $\eta^T_k$</th>
<th>1-Norm $|\eta^T_k|_1$</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>Output Frequency (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12 $\eta^T_{12}$</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>$2f_1 + 4\Delta f$</td>
</tr>
<tr>
<td>29 $\eta^T_{29}$</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$3f_1$</td>
</tr>
<tr>
<td>20 $\eta^T_{20}$</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$3f_1 + \Delta f$</td>
</tr>
<tr>
<td>13 $\eta^T_{13}$</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>$3f_1 + 2\Delta f$</td>
</tr>
<tr>
<td>15 $\eta^T_{15}$</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$3f_1 + 2\Delta f$</td>
</tr>
<tr>
<td>16 $\eta^T_{16}$</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$3f_1 + 3\Delta f$</td>
</tr>
<tr>
<td>30 $\eta^T_{30}$</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>$3f_1 + 3\Delta f$</td>
</tr>
<tr>
<td>21 $\eta^T_{21}$</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>$3f_1 + 4\Delta f$</td>
</tr>
<tr>
<td>22 $\eta^T_{22}$</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>$3f_1 + 4\Delta f$</td>
</tr>
<tr>
<td>14 $\eta^T_{14}$</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>$3f_1 + 5\Delta f$</td>
</tr>
<tr>
<td>31 $\eta^T_{31}$</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>$3f_1 + 6\Delta f$</td>
</tr>
</tbody>
</table>

Table 3.6: Frequency Sorted 2-Side VFD Table for 3 Tones in a 3rd Order Nonlinearity

<table>
<thead>
<tr>
<th>Frequency Index $k_x$ $(k_y, k_z)$</th>
<th>VFD $\eta^T_k$</th>
<th>1-Norm $|\eta^T_k|_1$</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>Output Frequency (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-31$ $\eta^T_{-31}$</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>$-3f_1 - 6\Delta f$</td>
</tr>
<tr>
<td>$-14$ $\eta^T_{-14}$</td>
<td>3</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>0</td>
<td>$-3f_1 - 5\Delta f$</td>
</tr>
<tr>
<td>$-22$ $\eta^T_{-22}$</td>
<td>3</td>
<td>-1</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>$-3f_1 - 4\Delta f$</td>
</tr>
<tr>
<td>$-21$ $\eta^T_{-21}$</td>
<td>3</td>
<td>0</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>$-3f_1 - 4\Delta f$</td>
</tr>
<tr>
<td>$-30$ $\eta^T_{-30}$</td>
<td>3</td>
<td>0</td>
<td>-3</td>
<td>0</td>
<td>0</td>
<td>$-3f_1 - 3\Delta f$</td>
</tr>
<tr>
<td>$-16$ $\eta^T_{-16}$</td>
<td>3</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>$-3f_1 - 3\Delta f$</td>
</tr>
</tbody>
</table>

Continued on next page
| Frequency Index $k_x$ (k_y, k_z) | VFD $\eta_{k_x}$ | $1-\text{Norm}$ $||\eta_k^T||_1$ | $f_1$ Weight $\eta_{k,1}$ | $f_2$ Weight $\eta_{k,2}$ | $f_3$ Weight $\eta_{k,3}$ | Output Frequency (Hz) |
|-----------------|------------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| $-15$ | $\eta_{-15}^T$ | 3 | $-2$ | 0 | $-1$ | $-3f_1 - 2\Delta f$ |
| $-13$ | $\eta_{-13}^T$ | 3 | $-1$ | $-2$ | 0 | $-3f_1 - 2\Delta f$ |
| $-20$ | $\eta_{-20}^T$ | 3 | $-2$ | $-1$ | 0 | $-3f_1 - \Delta f$ |
| $-29$ | $\eta_{-29}^T$ | 3 | $-3$ | 0 | 0 | $-3f_1$ |
| $-12$ | $\eta_{-12}^T$ | 2 | 0 | 0 | $-2$ | $-2f_1 - 4\Delta f$ |
| $-5$ | $\eta_{-5}^T$ | 2 | 0 | $-1$ | $-1$ | $-2f_1 - 3\Delta f$ |
| $-11$ | $\eta_{-11}^T$ | 2 | 0 | $-2$ | 0 | $-2f_1 - 2\Delta f$ |
| $-6$ | $\eta_{-6}^T$ | 2 | $-1$ | 0 | $-1$ | $-2f_1 - 2\Delta f$ |
| $-4$ | $\eta_{-4}^T$ | 2 | $-1$ | $-1$ | 0 | $-2f_1 - \Delta f$ |
| $-10$ | $\eta_{-10}^T$ | 2 | $-2$ | 0 | 0 | $-2f_1$ |
| $-23$ | $\eta_{-23}^T$ | 3 | 1 | 0 | $-2$ | $-f_1 - 4\Delta f$ |
| $-25$ | $\eta_{-25}^T$ | 3 | 0 | 1 | $-2$ | $-f_1 - 3\Delta f$ |
| $-18$ | $\eta_{-18}^T$ | 3 | 1 | $-1$ | $-1$ | $-f_1 - 3\Delta f$ |
| $-24$ | $\eta_{-24}^T$ | 3 | 1 | $-2$ | 0 | $-f_1 - 2\Delta f$ |
| $-3$ | $\eta_{-3}^T$ | 1 | 0 | 0 | $-1$ | $-f_1 - 2\Delta f$ |
| $-19$ | $\eta_{-19}^T$ | 3 | $-1$ | 1 | $-1$ | $-f_1 - \Delta f$ |
| $-2$ | $\eta_{-2}^T$ | 1 | 0 | $-1$ | 0 | $-f_1 - \Delta f$ |
| $-27$ | $\eta_{-27}^T$ | 3 | 0 | $-2$ | 1 | $-f_1$ |
| $-1$ | $\eta_{-1}^T$ | 1 | $-1$ | 0 | 0 | $-f_1$ |
| $-26$ | $\eta_{-26}^T$ | 3 | $-2$ | 1 | 0 | $-f_1 + \Delta f$ |
| $-17$ | $\eta_{-17}^T$ | 3 | $-1$ | $-1$ | 1 | $-f_1 + \Delta f$ |
| $-28$ | $\eta_{-28}^T$ | 3 | $-2$ | 0 | 1 | $-f_1 + 2\Delta f$ |
| $-7$ | $\eta_{-7}^T$ | 2 | 1 | 0 | $-1$ | $-2\Delta f$ |
| $-9$ | $\eta_{-9}^T$ | 2 | 0 | 1 | $-1$ | $-\Delta f$ |
| $-8$ | $\eta_{-8}^T$ | 2 | 1 | $-1$ | 0 | $-\Delta f$ |
| 0 | $\eta_{0}^T$ | 0 | 0 | 0 | 0 | 0 |
| 8 | $\eta_{8}^T$ | 2 | $-1$ | 1 | 0 | $\Delta f$ |

Continued on next page
| Frequency Index $k_x$ | VFD $\eta^T$ | 1-Norm $||\eta^T_k||_1$ | $f_1$ Weight $\eta_{k,1}$ | $f_2$ Weight $\eta_{k,2}$ | $f_3$ Weight $\eta_{k,3}$ | Output Frequency $\Delta f$ |
|----------------------|-------------|------------------|-----------------|-----------------|-----------------|------------------|
| 9 $\eta^T_9$         | 2           | 0                | −1              | 1               | $\Delta f$      |
| 7 $\eta^T_7$         | 2           | −1               | 0               | 1               | $2\Delta f$     |
| 28 $\eta^T_{28}$     | 3           | 2                | 0               | −1              | $f_1 - 2\Delta f$ |
| 17 $\eta^T_{17}$     | 3           | 1                | 1               | −1              | $f_1 - \Delta f$ |
| 26 $\eta^T_{26}$     | 3           | 2                | −1              | 0               | $f_1 - \Delta f$ |
| 1 $\eta^T_1$         | 1           | 1                | 0               | 0               | $f_1$           |
| 27 $\eta^T_{27}$     | 3           | 0                | 2               | −1              | $f_1$           |
| 2 $\eta^T_2$         | 1           | 0                | 1               | 0               | $f_1 + \Delta f$ |
| 19 $\eta^T_{19}$     | 3           | 1                | −1              | 1               | $f_1 + \Delta f$ |
| 3 $\eta^T_3$         | 1           | 0                | 0               | 1               | $f_1 + 2\Delta f$ |
| 24 $\eta^T_{24}$     | 3           | −1               | 2               | 0               | $f_1 + 2\Delta f$ |
| 18 $\eta^T_{18}$     | 3           | −1               | 1               | 1               | $f_1 + 3\Delta f$ |
| 25 $\eta^T_{25}$     | 3           | 0                | −1              | 2               | $f_1 + 3\Delta f$ |
| 23 $\eta^T_{23}$     | 3           | −1               | 0               | 2               | $f_1 + 4\Delta f$ |
| 10 $\eta^T_{10}$     | 2           | 2                | 0               | 0               | $2f_1$          |
| 4 $\eta^T_4$         | 2           | 1                | 1               | 0               | $2f_1 + \Delta f$ |
| 6 $\eta^T_6$         | 2           | 1                | 0               | 1               | $2f_1 + 2\Delta f$ |
| 11 $\eta^T_{11}$     | 2           | 0                | 2               | 0               | $2f_1 + 2\Delta f$ |
| 5 $\eta^T_5$         | 2           | 0                | 1               | 1               | $2f_1 + 3\Delta f$ |
| 12 $\eta^T_{12}$     | 2           | 0                | 0               | 2               | $2f_1 + 4\Delta f$ |
| 29 $\eta^T_{29}$     | 3           | 3                | 0               | 0               | $3f_1$          |
| 20 $\eta^T_{20}$     | 3           | 2                | 1               | 0               | $3f_1 + \Delta f$ |
| 13 $\eta^T_{13}$     | 3           | 1                | 2               | 0               | $3f_1 + 2\Delta f$ |
| 15 $\eta^T_{15}$     | 3           | 2                | 0               | 1               | $3f_1 + 2\Delta f$ |
| 16 $\eta^T_{16}$     | 3           | 1                | 1               | 1               | $3f_1 + 3\Delta f$ |
| 30 $\eta^T_{30}$     | 3           | 0                | 3               | 0               | $3f_1 + 3\Delta f$ |
| 21 $\eta^T_{21}$     | 3           | 0                | 2               | 1               | $3f_1 + 4\Delta f$ |

Continued on next page
<table>
<thead>
<tr>
<th>Frequency Index $k_x$ $(k_y, k_z)$</th>
<th>VFD $\eta^T$</th>
<th>1-Norm $|\eta^T_k|_1$</th>
<th>$f_1$ Weight $\eta_{k,1}$</th>
<th>$f_2$ Weight $\eta_{k,2}$</th>
<th>$f_3$ Weight $\eta_{k,3}$</th>
<th>Output Frequency $(Hz)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>22 $\eta^T_{22}$</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>$3f_1 + 4\Delta f$</td>
<td></td>
</tr>
<tr>
<td>14 $\eta^T_{14}$</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>$3f_1 + 5\Delta f$</td>
<td></td>
</tr>
<tr>
<td>31 $\eta^T_{31}$</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>$3f_1 + 6\Delta f$</td>
<td></td>
</tr>
</tbody>
</table>
3.5 The Spectral Vector

In this work, the *Spectral Vector* can take one of three user-selected forms:

Complex: This is the form discussed in Sections 3.3.2 and 3.3.4 where the elements are phasors (i.e., complex numbers), although in those sections the underlying signal being described was real-valued so that the vectors had conjugate symmetry. When the complex form is chosen, the element holding the DC value of the signal is permitted to have an imaginary part, and the spectrum is doubled-sided.

Double-Sided Real: In this form, the vector contains real elements ordered in pairs so that each pair corresponds to a phasor in the complex form. An exception is DC, where there is only one real element. The spectrum is two-sided.

Single-Sided Real: In this form, the vector contains real elements ordered in pairs so that each pair corresponds to a phasor in the complex form. An exception is DC, where there is only one real element. The spectrum is single-sided.

Despite their different forms, each of the spectral vectors maintains a 1 to 1 phasor relationship with the VFD Table. That is, every VFD entry has a corresponding phasor in the spectral vector upon which it is based (and the case of real-valued spectral vectors, two elements are required to store the phasor value except for the DC value). The number of elements in the spectral vector may change as a result of the choice of spectral vector form, but the 1 to 1 phasor relationship to the VFD does not. Frequency indices for the spectral vectors in this work have been based upon the frequency-sorted form of the VFD Table demonstrated in Table 3.5 on page 94, although it is possible to use the 1-norm sorted form by passing the appropriate frequency index vector (nfdx) to the construction routine described in Section 3.5.4.

3.5.1 Complex Spectral Vector

The complex spectral vector is the most general form of spectral vector, and it is the form used to describe the spectral vectors in Sections 3.3.2 and 3.3.4. The complex spectral vector has a 1:1 relationship with the VFD Table. Thus for a two-sided VFD Table containing \(2K + 1\) elements (\(K\) non-DC positive and negative frequencies plus DC), the complex spectral vector
will also contain $2K + 1$ complex-valued elements.

Elements of the complex spectral vector are arranged such that the element at frequency index $-K$ appears first, the DC element appears as the center or middle element of the vector, and the element at frequency index $+K$ appears last, so that

$$X = [X_{-K} X_{-K+1} \ldots X_{-1} X_0 X_1 \ldots X_{K-1} X_K]^T.$$

Because the complex spectral vector form allows for conjugate asymmetry in the signal, the DC element of the spectral vector must have an imaginary part. Complex spectral vectors are suited to modeling situations like the quadrature modulator shown in Figure 3.17. In a quadrature modulator, the independent baseband input data streams $i(t)$ and $q(t)$ are both real-valued so that their Fourier transforms have even-symmetry real parts and odd-symmetry imaginary parts (i.e. conjugate symmetry), but $q(t)$ is rotated through a 90 degree angle as it is modulated, and this corresponds to multiplication by $j$. Once $q(t)$ is modulated to the form $j\tilde{Q}(f)$, it is no longer conjugate symmetric, but instead possesses an odd real part and an even
imaginary part. Thus the sum signal $W(f) = I(f) + j\bar{Q}(f)$ no longer possesses a symmetry relationship that can be exploited to reduce its storage requirements; hence the necessity for double-sided complex elements.

Note that the complex spectral vector is not the best choice for modeling problems that might require iterative solutions, since both popular iterative methods — Newton’s method [156] and Krylov subspace methods [34] — are defined only over the field of real numbers. However, when direct methods are used, the complex spectral vector form is usually the best choice.

### 3.5.2 Double-Sided Real Spectral Vector

The double-sided real spectral vector is a two-sided alternative to the complex spectral vector that may be used in modeling problems requiring iterative solution methods. Relative to the complex spectral vector, the double-sided real spectral vector dispenses with the imaginary part of the DC element, and then defines ordered pairs of real elements for each of the non-DC phasors at both positive and negative frequencies. Since the imaginary part of DC is eliminated, it is imperative that only real signals be stored in this vector form. Thus, for a VFD Table containing $2K + 1$ entries, the double-sided real spectral vector will contain $4K + 1$ real-valued elements.

When the spectral vector is based on a 1-norm sorted VFD Table (such as Table 3.4), elements of the double-sided real spectral vector are arranged such that the imaginary part of the phasor for frequency index $-K$ appears first, followed by its real counterpart. The lone real DC element appears as the center or middle element of the vector. On the other side of the DC element, the positive-frequency elements are ordered beginning with the real part of the phasor and followed by the imaginary part, i.e. a mirror-image of the element ordering scheme for the negative frequency phasors. The vector ends with the imaginary part of the phasor at frequency index $+K$, so that,

$$X = [X_{-K,i}, X_{-K,r}, \ldots, X_{-1,i}, X_{-1,r}, X_{0,r}, X_{1,r}, X_{1,i}, \ldots, X_{K,r}, X_{K,i}]^T.$$  (3.195)

When the spectral vector is based upon a frequency-sorted VFD Table (such as Table 3.6), the phasor appearing first in the vector corresponds to the VFD with the most negative numerical frequency, the DC element is again the middle element of the vector, and the last phasor appearing in the vector corresponds to the VFD of the most positive frequency.
3.5.3 Single-Sided Real Spectral Vector

The single-sided real spectral vector is a one-sided version of the double-sided spectral vector which also assumes real input signals with conjugate symmetry. For a one-sided VFD Table containing \( K + 1 \) entries (such as Tables 3.3 and 3.5), the single-sided real spectral vector will contain \( 2K + 1 \) real-valued elements.

Phasors of the single-sided real spectral vector are ordered in exactly the same fashion used by Chang [92] in the FREDA2 circuit simulator:

\[
X = [X_{0,r}, X_{1,r}, X_{1,i}, \ldots, X_{K,r}, X_{K,i}]^T.
\]  

(3.196)

Note that \( X \) here is based on a 1 to 1 phasor relationship with a VFD Table, while Chang’s spectral vector was based upon a 1 to 1 phasor relationship with a user-selected set of numerical frequencies. When the VFD Table is 1-norm sorted, the \( k^{th} \) phasor in the spectral vector corresponds to the VFD at frequency index \( k \) exactly, with the real part appearing at vector location \( 2k \) and the imaginary part at vector location \( 2k + 1 \) for \( k > 1 \), while for \( k = 0 \), the DC element appears in the first element).

3.5.4 Spectral Vector Construction Algorithm

The spectral vector construction algorithm, which follows its Matlab® code implementation fairly closely, is shown in Procedure 3.5.4 on page 103. The algorithm is designed to allocate storage and seed the amplitude and phase values for a set of non-zero real sinusoids and a DC value for an input signal into the appropriate locations for the user’s selected form of spectral vector. One possible point of confusion concerns the frequency-sorted frequency index vector, \( ffidx \), which is used to guide the allocation of storage for spectral vectors. Note that it includes an index for DC so that it has length \( K + 1 \) for a single-sided real spectral vector and \( 2K + 1 \) for the double-sided real and the complex spectral vectors.

Notice that the double-sided form, on lines 12–15, splits the input amplitudes into halves and allocates them to the positive and negative frequencies, while the single-sided form on lines 26–29 does not scale the input amplitude. This scale factor of 2 on the input spectral vector (versus the double-sided case) is nothing more than the adoption of a convention for where to place the factor of 2 that would arise from summing complex conjugate outputs in the double-sided case.
CONSTRUCT-SPV(A, dc, fin, \phi_d, ffidx, sptype)

1 numsides ← 2
2 if ffidx[1] = 0 numsides ← 1 ▷ One-sided if first frequency index is DC.
3 if (sptype = COMPLEX) and (numsides = 1) error ▷ Invalid - complex must be 2s
4 \phi_in = (\pi/180) \cdot \phi_d ▷ Convert to radians
5 if (sptype = COMPLEX) and (numsides = 2)
6       then ▷ Two-sided, so length is 2K + 1
7       Xre ← ALLOC[length[ffidx]], Xim ← ALLOC[length[ffidx]]
8       ffidx ← (length[ffidx] - 1) + 1, Xdc ← ffidx, Xre[Xdc] ← dc
9       for m ← 1..length[f_in]
10          do
11             k ← FIND[ffidx - m] - ffidx
12             Xre[Xdc + k] ← \frac{1}{2}A[m] \cdot \cos(\phi_in[m])
13             Xre[Xdc - k] ← \frac{1}{2}A[m] \cdot \cos(\phi_in[m])
14             Xim[Xdc + k] ← \frac{1}{2}A[m] \cdot \sin(\phi_in[m])
15             Xim[Xdc - k] ← -\frac{1}{2}A[m] \cdot \sin(\phi_in[m])
16       end
17       X ← COMPLEX[Xre, Xim]
18 end
19 if (sptype = REAL) and (numsides = 2)
20       then ▷ Two-sided, so length is 4K + 1
21       X ← ALLOC[2 \cdot length[ffidx] - 1], ffidx ← \frac{1}{2}(length[ffidx] - 1) + 1
22       Xdc ← \frac{1}{2}(length[X] - 1) + 1, X[Xdc] ← dc
23       for m ← 1..length[f_in]
24          do
25             k ← FIND[ffidx - m] - ffidx
26             X[Xdc + 2k - 1] ← \frac{1}{2}A[m] \cdot \cos(\phi_in[m])
27             X[Xdc - 2k + 1] ← \frac{1}{2}A[m] \cdot \cos(\phi_in[m])
28             X[Xdc + 2k] ← \frac{1}{2}A[m] \cdot \sin(\phi_in[m])
29             X[Xdc - 2k] ← -\frac{1}{2}A[m] \cdot \sin(\phi_in[m])
30          end
31 end
32 if (sptype = REAL) and (numsides = 1)
33       then ▷ One-sided, so length is 2K + 1
34       X ← ALLOC[2 \cdot length[ffidx] - 1], ffidx ← 1, Xdc ← 1, X[Xdc] ← dc
35       for m ← 1..length[f_in]
36          do
37             k ← FIND[ffidx - m] - ffidx
38             X[Xdc + 2k - 1] ← A[m] \cdot \cos(\phi_in[m])
39             X[Xdc + 2k] ← A[m] \cdot \sin(\phi_in[m])
40          end
41 end
42 return X
3.5.5 Correlation Properties of Spectral Vectors

One of the essential properties motivating the use of the VFD table (rather than the numeric frequency domain) may now be explored. In Section 3.3.4 the spectrum transform matrix for two tones through a 3rd order nonlinearity was given in Figure 3.11 on page 63, and it was shown that non-zero spectral content occurs in the 3rd order output spectral vector at the same VFD as one of linear outputs in 3.141 on page 62. Here, the case of three commensurate tones through a 3rd order nonlinearity as described in Table 3.6 will be considered.

Let the 3 input tones have amplitudes $A_1$, $A_2$, and $A_3$, and phases $\phi_1$, $\phi_2$, and $\phi_3$, respectively, and let the input phases be independent, zero-mean, uniformly distributed random variables. Let $f_{\text{in}}$ and $\phi_{\text{in}}$ again be vectors of the input frequencies and phases, respectively. Rather than specifying a phasor for all 63 elements in spectral vectors based upon Table 3.6, a more compact form showing the spectral vector elements of interest (and their corresponding VFD Table entries) will be adopted. Thus, $X$ is given by:

\[
X = \begin{bmatrix}
\eta_{-31}^T & \Delta & 0 \\
\vdots & \Delta & \vdots \\
\eta_{-3}^T & \Delta & \frac{1}{2} A_3 e^{j \eta_{-3}^T \cdot \phi_{\text{in}}} \\
\eta_{-19}^T & \Delta & 0 \\
\eta_{-2}^T & \Delta & \frac{1}{2} A_2 e^{j \eta_{-2}^T \cdot \phi_{\text{in}}} \\
\eta_{-27}^T & \Delta & 0 \\
\eta_{-1}^T & \Delta & \frac{1}{2} A_1 e^{j \eta_{-1}^T \cdot \phi_{\text{in}}} \\
\vdots & \Delta & \vdots \\
\eta_0^T & \Delta & 0 \\
\vdots & \Delta & \vdots \\
\eta_1^T & \Delta & \frac{1}{2} A_1 e^{j \eta_1^T \cdot \phi_{\text{in}}} \\
\eta_{27}^T & \Delta & 0 \\
\eta_2^T & \Delta & \frac{1}{2} A_2 e^{j \eta_2^T \cdot \phi_{\text{in}}} \\
\eta_{19}^T & \Delta & 0 \\
\eta_3^T & \Delta & \frac{1}{2} A_3 e^{j \eta_3^T \cdot \phi_{\text{in}}} \\
\vdots & \Delta & \vdots \\
\eta_{31}^T & \Delta & 0
\end{bmatrix},
\] (3.197)
where the vertical dots denote 1 or more spectral vector elements containing 0. $C_x^{(2)} = X \ast X$ is given by

$$
C_x^{(2)} = \begin{bmatrix}
0 \\
\vdots \\
\eta_3^T \cdot \frac{1}{2} A_3 z \eta_7^T \cdot \phi_{1:} \\
\eta_5^T \cdot \frac{1}{2} A_2 A_3 e^{\eta_8^T \cdot \phi_{1:}} \\
\eta_6^T \cdot \frac{1}{2} A_2 A_3 e^{\eta_9^T \cdot \phi_{1:}} \\
\eta_7^T \cdot \frac{1}{2} A_1 A_3 e^{\eta_8^T \cdot \phi_{1:}} \\
\eta_9^T \cdot \frac{1}{2} A_2 A_3 e^{\eta_9^T \cdot \phi_{1:}} \\
\eta_8^T \cdot \frac{1}{2} (A_1^2 + A_2^2 + A_3^2) \\
\eta_1^T \\
\eta_2^T \\
\eta_3^T \\
\eta_4^T \\
\eta_5^T \\
\eta_6^T \\
\eta_7^T \\
\eta_8^T \\
\eta_9^T \\
\eta_{10}^T \\
\vdots \\
\eta_{11}^T \\
\eta_{12}^T \\
\vdots \\
\eta_{13}^T \\
\vdots \\
\vdots \\
\vdots
\end{bmatrix}.
$$

(3.198)
Let the transfer function of the nonlinear system be given by

\[ Y = Y_0 + Y_1 + Y_2 + Y_3 \]

\[ = b_0 e_0 + b_1 X + b_2 C^{(2)}_x + b_3 C^{(3)}_x \]  

(3.199)

Rather than compute \( C^{(3)}_x \) entirely without the assistance of a spectrum transform matrix, the purpose of this section will be served by considering just two elements of \( C^{(3)}_x \) — those appearing at frequency indices 1 and 27, which both share the same numerical frequency, \( f_1 \).

Note that \( \eta^T_1 = [1 \ 0 \ 0] \) and \( \eta^T_{27} = [0 \ 2 \ -1] \). The atomic convolution products that form \( C^{(3)}_{x_1} \) can be determined by performing the additions of the VFDs (for non-zero elements) of \( C^{(3)}_x \) and \( \phi \), determining those that give \( \eta^T_1 \) as the result, and then determining the phasor products of the corresponding elements. Summing the phasor products will determine the output. In the equations that follow, the VFD addition will be shown first, followed by the corresponding phasor products. For \( C^{(3)}_{x_1} \),

\[ \eta^T_1 = \eta^T_{10} + \eta^T_{-1} = [2 \ 0 \ 0] + [-1 \ 0 \ 0] \Rightarrow \left[ \frac{1}{4} A_1 e^{\eta^T_{10} \cdot \phi_{in}} \right] \left[ \frac{1}{2} A_1 e^{\eta^T_{-1} \cdot \phi_{in}} \right] = \frac{1}{8} A_1^3 e^{\eta^T \cdot \phi_{in}} \]  

(3.200)

\[ \eta^T_1 = \eta^T_6 + \eta^T_{-3} = [1 \ 0 \ 1] + [0 \ 0 \ -1] \Rightarrow \left[ \frac{1}{2} A_1 A_3 e^{\eta^T_{6} \cdot \phi_{in}} \right] \left[ \frac{1}{2} A_3 e^{\eta^T_{-3} \cdot \phi_{in}} \right] = \frac{1}{4} A_1 A_3^2 e^{\eta^T \cdot \phi_{in}} \]  

(3.201)

\[ \eta^T_1 = \eta^T_4 + \eta^T_{-2} = [1 \ 1 \ 0] + [0 \ -1 \ 0] \Rightarrow \left[ \frac{1}{2} A_1 A_2 e^{\eta^T_{4} \cdot \phi_{in}} \right] \left[ \frac{1}{2} A_2 e^{\eta^T_{-2} \cdot \phi_{in}} \right] = \frac{1}{4} A_1 A_2^2 e^{\eta^T \cdot \phi_{in}} \]  

(3.202)

\[ \eta^T_1 = \eta^T_{-8} + \eta^T_2 = [1 \ -1 \ 0] + [0 \ 1 \ 0] \Rightarrow \left[ \frac{1}{2} A_1 A_2 e^{\eta^T_{-8} \cdot \phi_{in}} \right] \left[ \frac{1}{2} A_2 e^{\eta^T_{2} \cdot \phi_{in}} \right] = \frac{1}{4} A_1 A_2^2 e^{\eta^T \cdot \phi_{in}} \]  

(3.203)

\[ \eta^T_1 = \eta^T_{-7} + \eta^T_3 = [1 \ 0 \ -1] + [0 \ 0 \ 1] \Rightarrow \left[ \frac{1}{2} A_1 A_3 e^{\eta^T_{-7} \cdot \phi_{in}} \right] \left[ \frac{1}{2} A_3 e^{\eta^T_{3} \cdot \phi_{in}} \right] = \frac{1}{4} A_1 A_3^2 e^{\eta^T \cdot \phi_{in}} \]  

(3.204)

\[ \eta^T_1 = \eta^T_0 + \eta^T_1 = [0 \ 0 \ 0] + [1 \ 0 \ 0] \Rightarrow \left[ \frac{1}{2} A_1^2 + \frac{1}{2} A_2^2 + \frac{1}{2} A_3^2 \right] \left[ \frac{1}{2} A_1 e^{\eta^T \cdot \phi_{in}} \right] = \left[ A_1^3 + A_1 A_2^2 + A_1 A_3^2 \right] e^{\eta^T \cdot \phi_{in}} . \]  

(3.205)

Summing the results of (3.200)–(3.205),

\[ C^{(3)}_{x_1} = C^{(3)}_{x_1} e^{\eta^T \cdot \phi_{in}} = \left[ \frac{3}{8} A_1^3 + \frac{3}{4} A_1 A_2^2 + \frac{3}{4} A_1 A_3^2 \right] e^{\eta^T \cdot \phi_{in}} , \]  

(3.206)
and thus,

\[ Y_{3,1} = b_3 C_{x_1}^{(3)}. \]  

(3.207)

Now, for \( C_{x_2x_7}^{(3)} \),

\[ \eta_{27}^T = \eta_{11}^T + \eta_{-3}^T = [0 \ 2 \ 0] + [0 \ 0 \ -1] \Rightarrow \left[ \frac{1}{4} A_2^2 e^{m_{11}^T \cdot \Phi_m} \right] \left[ \frac{1}{2} A_3 e^{m_{-3}^T \cdot \Phi_m} \right] = \frac{1}{8} A_2^2 A_3 e^{m_{27}^T \cdot \Phi_m} \]  

(3.208)

\[ \eta_{27}^T = \eta_{-27}^T = [0 \ 1 \ -1] + [0 \ 1 \ 0] \Rightarrow \left[ \frac{1}{2} A_2 A_3 e^{m_{-9}^T \cdot \Phi_m} \right] \left[ \frac{1}{2} A_2 e^{m_{27}^T \cdot \Phi_m} \right] = \frac{1}{4} A_2^2 A_3 e^{m_{27}^T \cdot \Phi_m} \]  

(3.209)

Summing the results of (3.208)–(3.209),

\[ C_{x_27}^{(3)} = C_{x_27}^{(3)} \left| e^{m_{27}^T \cdot \Phi_m} = \frac{3}{8} A_2^2 A_3 e^{m_{27}^T \cdot \Phi_m}, \right. \]  

(3.210)

and thus,

\[ Y_{3,27} = b_3 C_{x_27}^{(3)}. \]  

(3.211)

### 3.5.5.1 Uncorrelated Spectral Content

Consider now the correlation of the two 3\(^{rd}\) order spectral outputs of \( Y_3 \) at \( \eta_{11}^T \) and \( \eta_{27}^T \). From [15], the correlation of two-complex valued entities \( U \) and \( V \) is given by 

\[ R_{UV} = E[UV^*], \]

where the expectation operation may result in one or more integral evaluations over the independent random phase variables. The two complex-valued entities are considered uncorrelated if their correlation equals the product of the expected values of two entities, i.e. if

\[ R_{UV} = E[UV^*] = E[U]E[V^*]. \]

The correlation of the two spectral outputs \( Y_{3,1} \) and \( Y_{3,27} \) is given by

\[
R_{Y_{3,1}, Y_{3,27}} = E \left[ Y_{3,1} Y_{3,27}^* \right]
\]

(3.212)

\[
= E \left[ b_3 C_{x_1}^{(3)} \left| b_3 C_{x_27}^{(3)} \cdot e^{m_{11}^T \cdot \Phi_m} \right| \left| b_3 C_{x_27}^{(3)} \cdot e^{m_{27}^T \cdot \Phi_m} \right|^* \right]
\]

(3.213)

\[
= E \left[ b_3 C_{x_1}^{(3)} \left| C_{x_27}^{(3)} \cdot e^{m_{11}^T \cdot \Phi_m} \right| \left| C_{x_27}^{(3)} \cdot e^{m_{27}^T \cdot \Phi_m} \right|^* \right]
\]

(3.214)

\[
= |b_3|^2 C_{x_1}^{(3)} \left| C_{x_27}^{(3)} \cdot E \left[ e^{m_{11}^T \cdot \Phi_m} e^{m_{27}^T \cdot \Phi_m} \right]^* \right|
\]

(3.215)

After moving multiplicative constants out of the expectation operation in (3.215), it can be seen that the spectral correlation is entirely a function of the phases — the complex exponentials of
weighted sums of the independent random variables. Let $\beta$ be the multiplicative constant and let $g(\phi_1, \phi_2, \phi_3)$ be the function of complex exponentials in (3.215) so that

$$
\beta = |b_3|^2 \left| C_{x_1}^{(3)} \right| \left| C_{x_{27}}^{(3)} \right|,
$$

and

$$
g(\phi_1, \phi_2, \phi_3) = e^{m_1^T \phi_0} e^{m_{27}^T \phi_n^*}.
$$

Now expand $g$ as follows:

$$
g(\phi_1, \phi_2, \phi_3) = e^{m_1^T \phi_0} e^{m_{27}^T \phi_n^*}
$$

$$
= e^{i(\eta_1^T - \eta_{27}^T) \cdot \phi_n}
$$

$$
= e^{i([1 0 0] - [0 2 -1]) \cdot [\phi_1 \phi_2 \phi_3]}
$$

$$
= e^{i[1 -2 1] \cdot [\phi_1 \phi_2 \phi_3]}
$$

$$
= e^{i\phi_1} e^{-j2\phi_2} e^{j\phi_3}.
$$

At this point, it can be seen that $g(\phi_1, \phi_2, \phi_3)$ has been separated into the product of three complex exponentials

$$
g(\phi_1, \phi_2, \phi_3) = g_1(\phi_1) g_2(\phi_2) g_3(\phi_3),
$$

where

$$
g_1(\phi_1) = e^{i\phi_1}
$$

$$
g_2(\phi_2) = e^{-j2\phi_2}
$$

$$
g_3(\phi_3) = e^{j\phi_3}.
$$

From probability, it is also known [2, page 257] that the expected value of the product of functions of independent random variables is equal to the product of their expected values, i.e.,

$$
E[g_1(\phi_1) g_2(\phi_2) g_3(\phi_3)] = E[g_1(\phi_1)] E[g_2(\phi_2)] E[g_3(\phi_3)],
$$

and substituting this into (3.215),

$$
R_{Y_{3,1} Y_{3,27}} = |b_3|^2 \left| C_{x_1}^{(3)} \right| \left| C_{x_{27}}^{(3)} \right| E[g_1(\phi_1)] E[g_2(\phi_2)] E[g_3(\phi_3)].
$$

Consider the expectation on one complex exponential function, $g(\phi) = e^{j\alpha \phi}$, with $\alpha$ an integer but $\alpha \neq 0$, over a uniformly distributed random variable, where the probability density function
to 0. Therefore

\[ E[Y_{3,1}] = 0. \]  

\[ (3.241) \]
Now consider $E[Y_{3,27}]$:

$$E[Y_{3,27}] = E\left[ b_3 C_{x_{27}}^{(3)} e^{j \eta T_{27} \phi_{in}} \right]$$  \hspace{1cm} (3.242)

$$= b_3 C_{x_{27}}^{(3)} E\left[ e^{j \eta T_{27} \phi_{in}} \right]$$  \hspace{1cm} (3.243)

$$= b_3 C_{x_{27}}^{(3)} E\left[ e^{j \phi_1} e^{j 2 \phi_2} e^{-j 1 \cdot 3} \right]$$  \hspace{1cm} (3.244)

$$= b_3 C_{x_{27}}^{(3)} E\left[ e^{j \phi_1} \right] E\left[ e^{j 2 \phi_2} \right] E\left[ e^{-j 1 \cdot 3} \right].$$  \hspace{1cm} (3.245)

In (3.245), it can be seen that both $E\left[ e^{j 2 \phi_2} \right]$ and $E\left[ e^{-j 1 \cdot 3} \right]$ will evaluate to zero since they are both of the form of (3.232) with non-zero $\alpha$. Thus

$$E[Y_{3,27}] = 0.$$  \hspace{1cm} (3.246)

Collecting the results of (3.236), (3.241), and (3.246), it can be seen that

$$R_{Y_{3,1}Y_{3,27}} = 0$$  \hspace{1cm} (3.247)

$$E[Y_{3,1}] = 0$$  \hspace{1cm} (3.248)

$$E[Y_{3,27}] = 0,$$  \hspace{1cm} (3.249)

and therefore,

$$R_{Y_{3,1}Y_{3,27}} = E[Y_{3,1}]E[Y_{3,27}].$$  \hspace{1cm} (3.250)

Thus the two spectral components $Y_{3,1}$ and $Y_{3,27}$ are uncorrelated. They are also orthogonal since the correlation in (3.236) is zero. Note that in general, all of the spectral content occurring at different VFDs is uncorrelated.

### 3.5.5.2 Correlated Spectral Content

In Section 3.5.5.1 it was seen that spectral components occurring at different VFDs are uncorrelated. Given the results of (5.282) and (5.285), it might be guessed that the only way for spectral components to be correlated would be for them to have the same VFD. This is indeed
true. Consider, for example, the correlation among $Y_{1,1}$ and $Y_{3,1}$.

$$R_{Y_{1,1}Y_{3,1}} = E[Y_{1,1}Y_{3,1}^*] \quad (3.251)$$

$$= E\left[b_1 \frac{1}{2} A_1 e^{j\eta_1}\cdot\phi_{in} \left| b_3 \left| C_{x_1}^{(3)} e^{j\eta_1}\cdot\phi_{in} \right| \right]^*\right] \quad (3.252)$$

$$= \frac{1}{2} b_1 b_3 A_1 C_{x_1}^{(3)} \left| E \left[ e^{j(1 \ 0 \ 0) - [1 \ 0 \ 0]} [\phi_1 \ \phi_2 \ \phi_3] \right] \right| \quad (3.253)$$

$$= \frac{1}{2} b_1 b_3 A_1 C_{x_1}^{(3)} \left| E \left[ e^{A[0 \ 0 \ 0]} [\phi_1 \ \phi_2 \ \phi_3] \right] \right| \quad (3.254)$$

$$= \frac{1}{2} b_1 b_3 A_1 C_{x_1}^{(3)} \left| E \left[ e^{j0\phi_1} \ E \left[ e^{j0\phi_2} \ E \left[ e^{j0\phi_3} \right] \right] \right] \right| \quad (3.255)$$

Each of the expectation integrals in (3.256) will evaluate to 1 by virtue of (3.235), and therefore,

$$R_{Y_{1,1}Y_{3,1}} = \frac{1}{2} b_1 b_3 A_1 C_{x_1}^{(3)} \quad (3.257)$$

a non-zero result is obtained. In general, spectral content with the same VFD will be correlated, regardless of the order of non-linearity at which it occurs.

### 3.5.5.3 Observations on Correlated and Uncorrelated IM Distortion

Referring back to (3.200) through (3.205), it can be seen that 3rd order correlated IM distortion is created by a mixing process that results in phase cancellation and frequency subtraction, since the mixing process mixes a VFD with 1-norm of 2 with another VFD with 1-norm of 1 and produces a result with a 1-norm of 1. Correlated IM distortion always manifests itself in this manner. Notice that the numeric frequencies are immaterial here — if $f_1$, $f_2$, and $f_3$ were completely incommensurate, the 3rd order correlated IM distortion would still be created and would still fall at the same VFD, hence the same numeric frequency, as the linear output.

Note that the same observation cannot be made about uncorrelated IM distortion. Uncorrelated IM distortion that falls upon the same numeric frequency as the linear output occurs there precisely because the input tones are equally-spaced and the mixing process commensurately produces an output at the same numeric frequency as a linear output. Note that if the three input tones were made incommensurate, then the VFDs corresponding to (3.208) and (3.209) would fall upon a numeric frequency not equal to that of the linear output at $f_1$. 
For example, suppose \( f_2 = f_1 + \Delta f \) and \( f_3 = f_1 + 10\Delta f \). Then

\[
\eta_{27}^T \cdot f_{in} = [0 \ 2 \ -1] \cdot [f_1 \ f_2 \ f_3]^T \\
= 2 \cdot f_2 - f_3 \\
= 2[f_1 + \Delta f] - f_1 - 10\Delta f \\
= f_1 - 8\Delta f .
\]

(3.258) (3.259) (3.260) (3.261)

Since correlated IM distortion is created through phase cancellation and frequency subtraction, it can be expected to occur at higher orders of nonlinearity as well. For example, all of the VFDs in Table 3.4 on page 91 with a 1-norm of 3 could be created through a 5th order mixing process that mixes a VFD with 1-norm of 4 with another VFD with 1-norm of 1 to produce a result with a 1-norm of 3. Also, there will be 4th order rectification to DC, and subsequently some 5th order spectral content that will fall at VFDs with a 1-norm of 1.

One other phenomenon of interest concerns the power levels of correlated IM distortion compared to the uncorrelated IM distortion. It was first observed in [158] that the power levels of correlated IM are several dB higher than uncorrelated IM at the same frequency, ranging from as few as 4 dB near the middle of the in-band channel to as many as 8 dB near the edge of the in-band channel. A similar phenomenon was reported in publication number 5 in the list of Section 1.4.1 on page 9. Referring back to (3.200) through (3.211), it is possible to glean some insight into this. To simplify the analysis, let all of the amplitudes be equal so that \( A_1 = A_2 = A_3 = A \). Then the uncorrelated IM spectral output in (3.211) becomes

\[
Y_{3,27} = \frac{3}{8}b_3A^3e^{jn_{17}\phi_{in}}
\]

(3.262)

so that

\[
|Y_{3,27}| = \frac{3}{8}|b_3|A^3.
\]

(3.263)

The result in (3.263) will be used to form the reference power level. Now consider the correlated IM from (3.200) through (3.205) in two portions. Let the first portion be the sum of those mixtures that do not involve mixing a DC term, i.e. the sum of (3.200) through (3.204), or

\[
Y_{3,11} = \frac{9}{8}b_3A^3e^{jn_{17}\phi_{in}}
\]

(3.264)

so that

\[
|Y_{3,11}| = \frac{9}{8}|b_3|A^3.
\]

(3.265)
The difference in power, in dB, between (3.263) and (3.265) is given by

\[
\Delta P_{dB} = 20 \log \left( \frac{|Y_{3,1I}|}{|Y_{3,27}|} \right) 
\]

(3.266)

\[
= 9.54 \text{ dB}.
\]

(3.267)

Now, the portion of \(Y_{3,1}\) produced solely by (3.205), which is caused by mixing DC with the linear input signal at \(\eta_1\), is given by

\[
Y_{3,1II} = \frac{3}{4} b_3 A^3 e^{j \eta_1 T} \phi_{in} 
\]

(3.268)

\[
= \frac{6}{8} b_3 A^3 e^{j \eta_1 T} \phi_{in}
\]

(3.269)

so that

\[
|Y_{3,1II}| = \frac{6}{8} |b_3| A^3.
\]

(3.270)

Now, when both portions of the correlated IM are included, the difference in power, in dB, between (3.263) and the sums of (3.265) and (3.270) is given by

\[
\Delta P_{dB} = 20 \log \left( \frac{|Y_{3,1I} + Y_{3,1II}|}{|Y_{3,27}|} \right) 
\]

(3.271)

\[
= 13.97 \text{ dB}.
\]

(3.272)

Thus it can be seen that inclusion of just the single mixture involving DC causes the correlated IM power to increase by \((13.97 - 9.54 =) 4.43 \text{ dB}\). The importance of this DC mixing term was not noted in publication number 5 in the list of Section 1.4.1 on page 9. From a physical point of view, then, rectified power at DC due to 2\textsuperscript{nd} order mixing will be remodulated in the third-order mixing process and significantly increase correlated IM output power. This phenomenon should be expected from all broadband multistage amplifiers which are direct-coupled from input to output and have minimum memory effects. It should also be noted that combinatorics plays a role here, too, since there are only two mixtures which create 3\textsuperscript{rd} order spectral content at \(\eta_1\), while there are six that create content at \(\eta_{27}\).

### 3.5.5.4 Spectral Correlation Theorems and Remarks

The developments of Sections 3.5.5.1 and 3.5.5.2 can be distilled into the following theorems.

**Theorem 1** (Uncorrelated Spectral Vector Elements). Let \(Q\) independent sinusoidal tones have amplitudes \(A_q\) and independent and uniformly distributed random phases \(\phi_q\). Let their sum form
the input to a polynomial nonlinearity where the spectral content is described with a VFD table. Then the spectral content at each unique VFD is uncorrelated from the spectral content at all other VFDs.

Proof. Proof is by contradiction. It will be assumed that spectral content at two different VFDs may be correlated. Let the first spectral vector element have the VFD $\eta^T_k$ and let the second have the VFD $\eta^T_l$. Since the nonlinear order of the spectral vector from which the spectral components are being compared is unknown (and immaterial), the scalar complex constants comprising the product of the output polynomial scalar $b_n$ and any repeated convolutions of the input signal’s amplitudes will be denoted by $\beta_k$ and $\beta_l$, respectively. Thus,

$$Y_k = \beta_k e^{\eta^T_k \phi_{in}}$$  \hfill (3.273)
$$Y_l = \beta_l e^{\eta^T_l \phi_{in}}$$  \hfill (3.274)

Since it is assumed that $Y_k$ and $Y_l$ are correlated, their correlation must be non-zero. Now, the VFDs for the two spectral elements may be described as follows:

$$\eta^T_k = [\eta_{k,1} \eta_{k,2} \ldots \eta_{k,q} \ldots \eta_{k,Q}]$$  \hfill (3.275)
$$\eta^T_l = [\eta_{l,1} \eta_{l,2} \ldots \eta_{l,q} \ldots \eta_{l,Q}]$$  \hfill (3.276)

Let the difference of $\eta^T_k$ and $\eta^T_l$ be given by a vector $\alpha$,

$$\alpha = \eta^T_k - \eta^T_l$$  \hfill (3.277)

$$= [\eta_{k,1} \eta_{k,2} \ldots \eta_{k,q} \ldots \eta_{k,Q}] - [\eta_{l,1} \eta_{l,2} \ldots \eta_{l,q} \ldots \eta_{l,Q}]$$  \hfill (3.278)

$$= [\eta_{k,1} - \eta_{l,1} \ldots \eta_{k,q} - \eta_{l,q} \ldots \eta_{k,Q} - \eta_{l,Q}]$$  \hfill (3.279)

$$= [\alpha_1 \alpha_2 \ldots \alpha_q \ldots \alpha_Q] ,$$  \hfill (3.280)

so that $\alpha_q = \eta_{k,q} - \eta_{l,q}$ is the difference between the $q$th tuples of $\eta^T_k$ and $\eta^T_l$. Carrying out the correlation integral,

$$R_{Y_kY_l} = E[Y_k Y_l^*]$$  \hfill (3.281)

$$= E \left[ \beta_k e^{\eta^T_k \phi_{in}} \beta_l e^{\eta^T_l \phi_{in}} \right]^*$$  \hfill (3.282)

$$= \beta_k \beta_l^* E \left[ e^{j(\eta^T_k - \eta^T_l) \phi_{in}} \right]$$  \hfill (3.283)

$$= \beta_k \beta_l^* E \left[ e^{j\alpha \phi_{in}} \right]$$  \hfill (3.284)

$$= \beta_k \beta_l^* E \left[ e^{j\alpha_1 \phi_1} \right] E \left[ e^{j\alpha_2 \phi_2} \right] \ldots E \left[ e^{j\alpha_q \phi_q} \right] \ldots E \left[ e^{j\alpha_Q \phi_Q} \right].$$  \hfill (3.285)
Since it is assumed that $\eta_k^T$ and $\eta_l^T$ are distinct, then they must differ in at least one tuple value. Assume they differ at the $q^{th}$ tuple so that $\alpha_q \neq 0$. Then by virtue of (3.232), that expectation integral in (3.285) will evaluate to zero, and the correlation will thus be zero. This contradicts the assumption that the spectral content was correlated. Furthermore, since $E[Y_k]$ and $E[Y_l^T]$ will both evaluate to zero, the spectral content occurring at different VFDs is uncorrelated. ■

Remark (Conjugate-symmetric Input Signals have Conjugate Symmetric Correlation). When input signals are conjugate-symmetric, it should be understood that the conjugate symmetry will lead to conjugate-symmetric output signals, and thus spectral content at corresponding positive and negative VFDs are correlated, regardless of what correlation computations might indicate.

Remark (Spectral Content at Different Numerical Frequencies is Uncorrelated). In Theorem 1, the proof was carried out without regard to the numerical frequency at which the spectral content occurs. From Section 3.3, the mathematical description of spectral content includes an implied delta function which was subsequently dropped in the notation. In a strict sense (3.274) should be modified as follows:

$$Y_k = \beta_k e^{j\eta_k^T \cdot \phi_n} \delta(f - \eta_k^T \cdot f_{in}) \quad (3.286)$$

$$Y_l = \beta_l e^{j\eta_l^T \cdot \phi_m} \delta(f - \eta_l^T \cdot f_{in}) \quad (3.287)$$

When this modification is done, the two $\delta$ functions will move outside the expectation integrals since they are constants with respect to the input phases. However, unless the two VFDs occur at the same numerical frequency, the product of the $\delta$ functions will be zero and the correlation will thus be zero. Therefore spectral content occurring at numerically different frequencies is uncorrelated regardless of the input phases.

Theorem 2 (Correlated Spectral Vector Elements). Let $Q$ independent sinusoidal tones have amplitudes $A_q$ and independent and uniformly distributed random phases $\phi_q$. Let their sum form the input to a polynomial nonlinearity where the spectral content is described with a VFD table. Then the spectral content occurring at each VFD is correlated, regardless of the order of non-linearity at which it occurs.

Proof. Let the two spectral elements have the same VFD, $\eta_k^T$, and let the spectral content occur in spectral vectors for differing orders of nonlinearity $m$ and $n$, with $\beta_m$ and $\beta_n$ being
the constants rising from the scaling of the polynomial order and repeated convolution of input amplitudes. Then,

\[ Y_{m,k} = \beta_m e^{i\eta_m^T \phi_m} \]
\[ Y_{n,k} = \beta_n e^{i\eta_n^T \phi_n} \]  

(3.288)

(3.289)

Carrying out the correlation integral,

\[ R_{Y_{m,k}Y_{n,k}} = E[Y_{m,k}Y_{n,k}^*] \]
\[ = E\left[ \beta_m e^{i\eta_m^T \phi_m} \beta_n e^{i\eta_n^T \phi_n} \right]^* \]  
\[ = \beta_m \beta_n^* E\left[ e^{i(\eta_m^T - \eta_n^T) \phi_m} \right] \]  
\[ = \beta_m \beta_n^* E\left[ e^{i0\phi_1} \right] E\left[ e^{i0\phi_2} \right] \ldots E\left[ e^{i0\phi_q} \right] \ldots E\left[ e^{i0\phi_Q} \right] \]  
\[ = \beta_m \beta_n^* \]  

(3.290)

(3.291)

(3.292)

(3.293)

(3.294)

Now by virtue of \ref{eq:3.285}, each expectation integral in \ref{eq:3.293} will evaluate to one, and so the two spectral elements are correlated.

\[ \text{Remark (Spectral Elements Are Not Partially-Correlated).} \] As a result of Theorems 1 and 2, which make use of (3.232) and (3.235), it can be seen that the expectation integrals that govern whether two particular spectral elements are correlated or not always evaluate to either zero or one. This calls strongly into question the notion posited in some quarters that spectral content can be intrinsically partially-correlated, which may lead casual observers to conclude that such spectral content cannot be separated into correlated and uncorrelated portions. The results developed in this work show that when the random phase variables are independent and uniformly distributed, then spectral content is either correlated or uncorrelated — with no partial correlation. Results have not been presented for other probability distributions, but the results here are likely to hold true for random variables that are independent and identically distributed. Note that the remark here does not pertain to discussions of spectral content which are consciously created in an environment of enforced correlation. In those cases, the term partially-correlated may be perfectly apt.

\[ \text{Remark (Spectral Vector Elements Form an Appropriate Set for Nonlinear Phasor Analysis).} \] In \cite{159}, the authors posit that the frequency-domain part of Harmonic Balance (HB) methods constitute a form of nonlinear phasor analysis. This view could be questioned on several
grounds, with the main one being that a Fourier transform of an arbitrary time domain waveform will often create frequency-domain locations with no spectral content, and these locations can be viewed as artifacts of the analysis method which have no analog to the physical problem being modeled. On the other hand AOM — as implemented in the AOM Toolbox — may appropriately lay claim to being an appropriate environment for nonlinear phasor analysis, for two reasons:

1. The VFD Table exactly defines a set of locations, in vector form, where spectral content will occur
2. For uncorrelated phase multitone inputs, every spectral vector phasor location is uncorrelated from the others, even when numeric frequencies are equal
3.6 Spectrum Mapping Table

The Spectrum Mapping Table identifies the valid atomic multiplication products arising from the convolution of a given spectral vector, usually the input spectral vector $X$, with another arbitrary spectral vector $Z$. For double-sided spectral vectors (whether real or complex), it records the frequency indices of the spectral vector elements involved in the atomic multiplication in a table called the Spectrum Mapping Table. The phasor contents of $Z$ are not even specified; all that is required is for $X$ and $Z$ to be defined with the same form, i.e. with the same VFD Table and the same spectral vector (two-sided complex, two-sided real, or one-sided real) form. The spectrum mapping table for double-sided spectral vectors will be discussed here; the spectrum mapping table for the single-sided real spectral vector, which is more complex due to the need to store more information, will be discussed in Section 3.9.

One innovation in the AOM Toolbox is a provision to create the spectrum mapping table even in the absence of spectral content for the input spectral vector $X$. In this case, the software that creates the spectrum mapping table assumes that the VFD Table entries with 1-norms of 1 – which always correspond to the locations of non-zero input spectral content – are non-zero in the discrete convolution process. Thus for problems requiring multiple simulation runs with different input spectral vector values (e.g. power sweeps that require sweeps of input amplitude, or random phase inputs), it is possible to create a single spectrum mapping table to serve for the entire ensemble of simulations since amplitude and phase information is not required for the creation of the spectrum mapping table – that information is necessary only for the creation of the spectrum transform matrix.

Another innovation in the spectrum mapping algorithm is a drastic reduction in the computing complexity relative to the method used by Chang [92], whose method is described in publication 4 in Section 1.4.1 on page 9 and also in Appendix C. For a VFD Table with $K$ entries, the complexity of the previous algorithm was $\mathcal{O}(K^3)$ while the complexity of the new algorithm has been reduced to $\mathcal{O}(K^2)$. The author wishes to acknowledge that, although he developed the algorithm to be described here independently, another graduate student, Jim Hall, developed essentially the same algorithm simultaneously and independently during a period when some of their work overlapped 160.

The essential mechanics of spectrum mapping were outlined in Section 3.3.3 for the case of a spectral vector with 2 input tones that was defined for a second order nonlinearity.
Here the length of the spectral vector will be assumed to be arbitrary, and the algorithmic process will be defined for the general case of an input signal of \( Q \) tones (i.e. VFD entries of length \( Q \)) passed through a nonlinearity of maximal order \( N \). In the illustration here it will be assumed that the spectral vector which will be spectrum-mapped has non-zero content only at the input frequencies, so the VFDs of interest will be only those \( Q \) VFDs with 1-norms of 1.

3.6.1 Convolution Steps and the 1-Norm Sum Test

The number of convolution steps will be \( I = 2(2K + 1) - 1 = 4K + 1 \), where \( 2K + 1 \) is the length of the VFD Table (for \( K \) positive and negative frequencies and DC). Reflecting the exploitation of sparsity in the algorithm, note that the vector representing \( X \) in Figure 3.18 shows only those elements (positive and negative) reflecting the input tones, which are the only non-zero elements in this case. As a result of exploiting sparsity, the maximum number of atomic multiplications that takes place at each convolution step will be no more than \( 2Q \), the number of non-zero spectral vector elements in \( X \), and many steps will complete with no atomic multiplications at all because the indices in \( X \) that align for atomic multiplication will correspond to spectral vector elements containing zeros.

The first convolution step at which non-zero content in \( X \) will occur is, approximately, step \( I = K - Q \), as shown Figure 3.18. Here, the \(-Q^{th}\) element in \( X \) aligns for multiplication with the \( K^{th}\) element in \( Z \). The first action in determining whether an atomic product is valid is
to add the 1-norms of the VFDs corresponding to the two spectral elements. If the sum exceeds the maximum order of non-linearity in the system, as it does in Figure 3.18, then the atomic product is not a valid one since summing the 1-norms of its corresponding VFDs leads to an order of non-linearity that is greater than allowed by the physical system being modeled. Atomic multiplications which do not pass this 1-norm sum test are dropped from further consideration, and the algorithm moves on to consider the next atomic multiplication. In Figure 3.19 it will be assumed that \( \| \eta_{T}^{T} \|_1 = N \) but that \( \| \eta_{T}^{T} \|_1 + \| \eta_{T}^{T} \|_1 = 1 + N > \text{Max 1-norm of N} \). Thus the atomic multiplication of

\[
\begin{array}{c}
X_{Q} \\
\vdots \\
X_{Q+1} \\
X_{Q} \\
\vdots \\
Z_{K}
\end{array}
\]

\( \| \eta_{T}^{T} \|_1 = N \) and \( \| \eta_{T}^{T} \|_1 = 1 \) with \( N \) the 1-norm of the sum of their VFDs. Computation of the exact VFD of the output will be discussed in Section 3.6.2.

Figures 3.20 and 3.21 draw attention to convolution steps involving element \( Z_0 \). Figure

\[
\begin{array}{c}
X_{Q} \\
\vdots \\
X_{Q+1} \\
X_{Q} \\
\vdots \\
Z_{K}
\end{array}
\]

\( \| \eta_{T}^{T} \|_1 + \| \eta_{T}^{T} \|_1 = 1 + 0 = 1 <= \text{Max 1-norm of N} \). Thus, \( \eta_{T}^{T} + \eta_{0}^{T} = \eta_{T}^{T} \in \text{VFD Table} \). Figure

\[
\begin{array}{c}
X_{Q} \\
\vdots \\
X_{Q+1} \\
X_{Q} \\
\vdots \\
Z_{K}
\end{array}
\]

3.20 shows \( Z_0 \) intersecting with \( X_{Q} \) — the first intersection of \( Z_0 \) with a non-zero element of
has a VFD of $\eta^T_0 = 0$, the 1-norm of the sum will pass the 1-norm sum test, and the VFD of the output will be the same as the input.

Figure 3.22 shows the convolution step with the final valid atomic product and Figure 3.23 shows the final convolution step.

### 3.6.2 Determining The Frequency Index of the Output VFD

During the course of the convolution steps, when an atomic product passes the 1-norm sum test, it corresponds to a valid convolution product. However, the frequency index corresponding to the output VFD is still unknown, and it must be determined before an entry can be made in the spectrum mapping table. In the introductory discussion of Section 3.3.3, the set of VFDs

\[
\begin{array}{c|c|c}
X_Q & Z_0 & \|\eta^T_Q\|_1 + \|\eta^T_0\|_1 = 1 + 0 = 1 <= \text{Max 1-norm of } N \\
X_{Q-1} & Z_1 & \eta^T_Q + \eta^T_0 = \eta^T_Q \in \text{VFD Table} \\
\vdots & \vdots & \\
X_{-Q} & Z_K & \\
\end{array}
\]

Figure 3.21: $I \approx 2K + Q$, step with valid atomic multiplication products.

\[
\begin{array}{c|c|c}
X_Q & Z_L & \|\eta^T_Q\|_1 + \|\eta^T_L\|_1 = 1 + (N - 1) = N \\
X_{Q-1} & Z_{L+1} & \|\eta^T_{Q-1}\|_1 + \|\eta^T_{L+1}\|_1 = 1 + N > \text{Max 1-norm of } N \\
\vdots & \vdots & \\
X_{-Q} & \vdots & \\
\end{array}
\]

Figure 3.22: $I \approx 4K - Q$, final convolution step with valid products.

3.23 shows the final convolution step.
Figure 3.23: $I \approx 4K - 1$, the final convolution step – with invalid products.

was small enough that simple inspection — algorithmically corresponding to a linear search — was sufficient. However, when the VFD Table contains thousands of entries, linear search techniques no longer suffice due to their computational inefficiency.

The solution to rapidly determining the frequency index of the output VFD is to re-use the hash tables created for nonlinear orders 2 through $N$ during the construction of the VFD Table. This is possible because the hash tables contain the values of the frequency indices for a 1-sided VFD Table with an offset of $-1$ owing to the fact that the DC VFD is not part of the VFD Table during its construction. Two examples will illustrate the use of the hash tables.

Consider the atomic multiplication product of $X_Q$ and $Z_L$ in Figure 3.22, which has a 1-norm sum of $N$. The dot product of the output VFD with the input frequency vector $f_{in}$ yields a positive number, so the output frequency is positive. Since $N$ is the 1-norm, the hash table $\mathcal{A}_N$ is referenced using the hash of the output VFD. The hash table returns a vector of one or more indices into the one-sided VFD Table $\mathcal{H}^+$. The value 1 is added to each of these indices, which then furnishes the correct set of indices to check a 1-norm sorted 1-sided VFD Table, at which each index is then checked. If the table is two-sided, an additional offset to the DC index value (which becomes the middle element in the VFD Table) is added to the indices, and then the VFD Table is checked at each of the resulting indices until a match is found. When the matching VFD is found, the corresponding output frequency index is then available.
for entry into the spectrum mapping table.

A more complicated example involves negative frequencies. Consider the atomic multiplication product of $X/Q$ with $Z/L$ in Figure 3.19 This atomic multiplication is assumed to pass the 1-norm sum test with value $N$, but corresponds to a negative frequency since the dot product of the output VFD with the input frequency vector $f_{in}$ is negative. The hash table $A_N$ is still referenced and 1 is added to each of the indices as before. Now, however, these indices are subtracted from the DC index value in order to establish the correct VFDs to compare in the complete VFD Table. When the matching VFD is found, the corresponding output frequency index can be entered into the spectrum mapping table.

### 3.6.3 Spectrum Mapping Table Contents

Once the frequency index of the output VFD is determined, an entry may be made in the spectrum mapping table reflecting a valid convolution product. The table entry is a 3-tuple of the form $[k_y k_x k_z]$, where $k_y$ is the frequency index of the output, $k_x$ is the frequency index of spectral vector element in $X$ involved in the atomic multiplication, and $k_z$ is the frequency index of the dummy spectral vector $Z$ involved in the atomic multiplication. The table is empty at the start of the convolution process, and entries are appended to the table for every valid atomic convolution product.

### 3.6.4 Spectrum Mapping Table Construction Algorithm

Procedure CONSTRUCT-SMT on page 124 describes the algorithm documented in the previous sections. Procedure CONSTRUCT-SMT maintains the sparse index positioning of the convolution iterations and determines output VFDs resulting from atomic convolution products. It then calls Procedure GETFREQIDX on page 125 which handles the low-level details of referencing the hash tables and the VFD Tables to locate the correct output frequency index. Within both procedures, calls to Matlab® procedures are indicated by small caps. Details of the procedure GETXMAGIND have been omitted.
CONSTRUCT-SMT($\mathcal{N}_H$, nfirst, nfidx, $f_{in}$, $F_H$, ffreq, fperm, $X$, $N$, numsides, $A$)

$[K, Q] = \text{size}(F_H) \triangleright$ Get number of VFD entries and input tones

$Xmag = \text{GETXMagInd}(X) \triangleright$ Indicator (0/1) of non-zero elements in $X$

$nzXmag = \text{FIND}(Xmag) \triangleright$ Indices of non-zero elements (same as VFD Table)

for $I_{kn} \leftarrow 1..(2 \times K - 1)$

do

if $I_{kn} < K$

then

$I_{kx,\text{start}} \leftarrow 1$, $I_{kx,\text{end}} \leftarrow I_{kn}$

if $I_{kx,\text{end}} >= nzXmag(1)$

then

$nzI_{kx,\text{start}} = \text{FIND}(nzXmag >= I_{kx,\text{start}}, 1, '{first}')$

$nzI_{kx,\text{end}} = \text{FIND}(nzXmag <= I_{kx,\text{end}}, 1, '{last}')$

end

else

$I_{kx,\text{start}} \leftarrow I_{kn} - K + 1$, $I_{kx,\text{end}} \leftarrow K$

if $I_{kx,\text{start}} <= nzXmag(\text{end})$

then

$nzI_{kx,\text{start}} = \text{FIND}(nzXmag >= I_{kx,\text{start}}, 1, '{first}')$

$nzI_{kx,\text{end}} = \text{FIND}(nzXmag <= I_{kx,\text{end}}, 1, '{last}')$

end

end

$nzI_{kx} \leftarrow nzXmag(nzI_{kx,\text{start}} : nzI_{kx,\text{end}})$

if not(ISEMPYT(nzI_{kx}))

then

$nzI_{kz} \leftarrow I_{kn} - nzI_{kx} + 1$

$H_{kx} \leftarrow F_H(nzI_{kx}, 1 : Q)$, $H_{kz} \leftarrow F_H(nzI_{kz}, 1 : Q)$

$\text{norm}_{cp} \leftarrow \|H_{kx}\|_1 + \|H_{kz}\|_1$, $\text{valid}_{cp} = \text{FIND}(\text{norm}_{cp} <= N)$

if not(ISEMPYT(\text{valid}_{cp}))

then

$sum_{cp} \leftarrow F_H(nzI_{kx}, 1 : Q) + F_H(nzI_{kz}, 1 : Q)$

for $I_{nx} \leftarrow 1..\text{length}(\text{valid}_{cp})$

do

$I_{dx} \leftarrow \text{valid}_{cp}[I_{nx}]$

$\eta_{ck} \leftarrow sum_{cp}(I_{dx}, 1 : Q) \triangleright$ Isolate a c.p. VFD to check

$[I_{ky}] = \text{GETFREQIDX}$

($\eta_{ck}, \text{numsides}, \text{nfirst}, f_{in}, N_H, I_{dx}, nzI_{kx}, nzI_{kz}, A$)

\triangleright Append an entry to the spectrum mapping table.

end

\triangleright Insert 1-sided checking (see Section 3.9.3 on page 144)

end

end \triangleright$ ... If $nzI_{kx}$ not empty ...

end \triangleright$ ... For $I_{kn}$ Do ...

return [smt]
GETFREQIDX($\eta_{ck}^T$, numsides, nfirst, $f_{in}$, $N_H$, $I_{dx}$, $nzI_{kx}$, $nzI_{kz}$, $A$)

1. $I_{ky} \leftarrow -1$ \texttt{▷ Initialize to error value}
2. $nrm_{ck} = \text{NORM}(\eta_{ck}^T)$
3. \textbf{if} $nrm_{ck} < 2$ \texttt{▷ No hash tables for 1-norms of 0 and 1, use nfirst table}
4. \textbf{then}
5. \quad $nfirst_{bi} \leftarrow \text{norm}_{cp}(I_{dx}) + 1$, $nfirst_{ei} \leftarrow nfirst_{bi} + 1$
6. \quad $ck_{bi} \leftarrow nfirst(nfirst_{bi})$, $ck_{ei} \leftarrow nfirst(nfirst_{ei} - 1)$
7. \quad \textbf{for} $K_{y\omega_{ck}} \leftarrow ck_{bi} \ldots ck_{ei}$
8. \quad \quad \textbf{do}
9. \quad \quad \quad \textbf{if} $N_H(K_{y\omega_{ck}}) = \eta_{ck}^T$
10. \quad \quad \quad \quad \quad \textbf{then}
11. \quad \quad \quad \quad \quad \quad $I_{ky} \leftarrow K_{y\omega_{ck}}$ \texttt{▷ Found match, set result}
12. \quad \quad \quad \quad \quad \quad \textbf{break}
13. \quad \quad \quad \textbf{end}
14. \quad \quad \textbf{end}
15. \quad \textbf{end}
16. \textbf{else} \texttt{▷ 1-norm $\geq 2$, use hash tables}
17. \quad $A_{nrm} \leftarrow A(nrm_{ck})$
18. \quad $vfdsav \leftarrow \eta_{0}^T$, $vfdneg \leftarrow \text{FALSE}$ \texttt{▷ assume positive frequency}
19. \quad $K_{y_{DC}} \leftarrow 1$ \texttt{▷ DC location for 1-s VFD Table}
20. \quad \textbf{if} (numsides $= 2$)
21. \quad \quad \textbf{then}
22. \quad \quad \quad $K_{y_{DC}} \leftarrow \frac{1}{2}(\text{length}(N_H) - 1) + 1$
23. \quad \quad \quad \textbf{if} ($\eta_{ck} \cdot f_{in} < 0$)
24. \quad \quad \quad \quad \quad \textbf{then} \texttt{▷ Save and negate VFD for hash table search}
25. \quad \quad \quad \quad \quad \quad $vfdsav \leftarrow \eta_{ck}^T$, $vfdneg \leftarrow \text{TRUE}$, $\eta_{ck}^T \leftarrow -\eta_{ck}^T$
26. \quad \quad \textbf{end}
27. \quad \textbf{end}
28. \quad \textbf{for} $i \leftarrow 1 \ldots \text{length}(V_a)$
29. \quad \quad \textbf{do}
30. \quad \quad \quad \textbf{if} (vfdneg)
31. \quad \quad \quad \quad \quad \texttt{then}
32. \quad \quad \quad \quad \quad \quad $K_{y\omega_{ck}} \leftarrow K_{y_{DC}} - V_a(i)$
33. \quad \quad \quad \quad \quad \textbf{else}
34. \quad \quad \quad \quad \quad \quad $K_{y\omega_{ck}} \leftarrow K_{y_{DC}} + V_a(i)$
35. \quad \quad \quad \quad \quad \textbf{end}
36. \quad \quad \quad \textbf{if} $N_H(K_{y\omega_{ck}}) = \eta_{ck}^T$
37. \quad \quad \quad \quad \quad \textbf{then}
38. \quad \quad \quad \quad \quad \quad $I_{ky} \leftarrow K_{y\omega_{ck}}$ \texttt{▷ Found match, set result}
39. \quad \quad \quad \quad \quad \quad \textbf{break}
40. \quad \quad \quad \textbf{end}
41. \quad \quad \textbf{end}
42. \quad \textbf{end}
43. \textbf{if} $I_{ky} = -1$ \texttt{error} \texttt{▷ This should not happen...}
44. \textbf{return} $[I_{ky}]$
3.6.5 Matlab Implementation Details

The Spectrum Mapping Tables returned by Procedure CONSTRUCT-SMT has entries as row elements as indicated. However, during the construction phase of the spectrum mapping table, it is actually built in a transpose form, with each new entry adding a column (rather than a row) to the table in order to perform best given the column-major storage order of Matlab\textsuperscript{®} \textsuperscript{[151, 152]}. The table is initialized to store a number of entries equal to 25\% of the product of the number of frequencies in the VFD Table and the number of input tones, but can be re-sized during construction if necessary. At the completion of the construction process, the table is transposed and returned in row-major form.
3.7 Spectrum Transform Matrix

The construction of the Spectrum Transform Matrix is driven wholly by the Spectrum Mapping Table, requiring only the same spectral vector, $X$, that was used by the Spectrum Mapping Table algorithm. The Spectrum Transform Matrix construction algorithm constructs the spectrum transform matrix as a sparse matrix since $X$ is typically sparse. This section will furnish details of spectrum transform matrix construction when $X$ is 2-sided. The 1-sided case will be handled in Section 3.9.

Since the spectrum transform matrix is square and has the same number of rows and columns as $X$, the location of elements is determined as an offset from the center element of $X$, which defines the center element of $T_x$, the spectrum transform matrix. The offset value is given by the frequency index values in each spectrum mapping table entry.

3.7.1 Spectrum Transform Matrix Construction Algorithm

Procedure 3.7.1 on page 130 shows spectrum transform matrix construction pseudocode for matrices constructed from 2-sided spectral vectors. The complex-valued spectrum matrix construction is given on lines 1–11 (on page 130) and is straightforward: Each spectrum mapping table entry gives furnishes three frequency indices $K_y, K_x$, and $K_z$, and the algorithm superposes the complex spectral vector element $X_{Kx}$ with frequency index $K_x$ into the spectrum transform matrix at the matrix location given by the frequency indices $(K_y, K_z)$, which furnish the offset to the Matlab® index value of the center element of the matrix, $X_{dci}$.

Construction of the 2-sided real-valued spectrum transform matrix, shown on lines 12–41 on page 130, is somewhat complicated, but uses the exact same underlying numerical data as the complex-valued spectrum transform matrix with the added assumption that the DC element of $X$ has no imaginary part. Letting $Y_{Ky}, X_{Kx},$ and $Z_{Kz}$ be complex-valued phasor elements of spectral vectors $Y, X,$ and $Z$, respectively, the crux of creating the two-sided real-valued spectrum transform matrix is to convert the complex-valued product $Y_{Ky} = X_{Kx} Z_{Kz}$ into two real-valued equations by separating the real and imaginary parts given by

\[
Y_{Ky} = X_{Kx} Z_{Kz} \tag{3.295}
\]

\[
Y_{Ky,r} + jY_{Ky,i} = (X_{Kx,r} + jX_{Kx,i}) (Z_{Kz,r} + jZ_{Kz,i}) \tag{3.296}
\]

\[
Y_{Ky,r} + jY_{Ky,i} = X_{Kx,r}Z_{Kz,r} + jX_{Kx,r}Z_{Kz,i} + jX_{Kx,i}Z_{Kz,r} - X_{Kx,i}Z_{Kz,i} \tag{3.297}
\]
into the two separate equations

\[ Y_{Ky,r} = X_{Kx,r}Z_{Kz,r} - X_{Kx,i}Z_{Kz,i} \]  \hspace{1cm} (3.298)

\[ Y_{Ky,i} = X_{Kx,i}Z_{Kz,r} + X_{Kx,r}Z_{Kz,i} \]  \hspace{1cm} (3.299)

The results of (3.298) and (3.299) may be cast into matrix form as follows:

\[
\begin{bmatrix}
  Y_{Ky,r} \\
  Y_{Ky,i}
\end{bmatrix} =
\begin{bmatrix}
  X_{Kx,r} & -X_{Kx,i} \\
  X_{Kx,i} & X_{Kx,r}
\end{bmatrix}
\begin{bmatrix}
  Z_{Kz,r} \\
  Z_{Kz,i}
\end{bmatrix}.
\]  \hspace{1cm} (3.300)

The 2 \times 2 matrix of the real and imaginary parts of \( X_{Kx} \) in (3.300) may be considered a “stamp” which is appropriately constructed for superposition into the lower right quadrant of the spectrum transform matrix, where the frequency indices of \( Y \) and \( Z \) are positive. This is indicated in Figure 3.24 by the ① symbol.

<table>
<thead>
<tr>
<th>( k_y )</th>
<th>( \cdots )</th>
<th>(-K_{z,i})</th>
<th>(-K_{z,r})</th>
<th>( 0 )</th>
<th>( K_{z,r})</th>
<th>( K_{z,i})</th>
<th>( \cdots )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>(-K_{y,i})</td>
<td>( X_{Kx,r} )</td>
<td>( X_{Kx,i} )</td>
<td>( X_{Kx,i} )</td>
<td>( X_{Kx,r} )</td>
<td>( X_{Kx,r} )</td>
<td>( X_{Kx,i} )</td>
<td>( X_{Kx,i} )</td>
</tr>
<tr>
<td>(-K_{y,r})</td>
<td>( -X_{Kx,i} )</td>
<td>( X_{Kx,r} )</td>
<td>( X_{Kx,i} )</td>
<td>( X_{Kx,r} )</td>
<td>( X_{Kx,r} )</td>
<td>( X_{Kx,i} )</td>
<td>( X_{Kx,i} )</td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( 0 )</td>
<td>( -X_{Kx,i} )</td>
<td>( X_{Kx,r} )</td>
<td>( X_{Kx,i} )</td>
<td>( X_{Kx,r} )</td>
<td>( X_{Kx,r} )</td>
<td>( X_{Kx,i} )</td>
<td>( X_{Kx,i} )</td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( K_{y,i})</td>
<td>( X_{Kx,i} )</td>
<td>( X_{Kx,r} )</td>
<td>( X_{Kx,i} )</td>
<td>( X_{Kx,r} )</td>
<td>( X_{Kx,r} )</td>
<td>( X_{Kx,i} )</td>
<td>( X_{Kx,i} )</td>
</tr>
<tr>
<td>( K_{y,r})</td>
<td>( X_{Kx,r} )</td>
<td>( X_{Kx,i} )</td>
<td>( X_{Kx,i} )</td>
<td>( X_{Kx,r} )</td>
<td>( X_{Kx,r} )</td>
<td>( X_{Kx,i} )</td>
<td>( X_{Kx,i} )</td>
</tr>
</tbody>
</table>

Figure 3.24: 2-Sided Spectrum Transform Matrix Stamps.

Variations on (3.300) are required for other quadrants:

\[
\begin{bmatrix}
  Y_{Ky,i} \\
  Y_{Ky,r}
\end{bmatrix} =
\begin{bmatrix}
  X_{Kx,i} & X_{Kx,r} \\
  X_{Kx,r} & -X_{Kx,i}
\end{bmatrix}
\begin{bmatrix}
  Z_{Kz,r} \\
  Z_{Kz,i}
\end{bmatrix} \text{ for quadrant } ②, \hspace{1cm} (3.301)

\[
\begin{bmatrix}
  Y_{Ky,i} \\
  Y_{Ky,r}
\end{bmatrix} =
\begin{bmatrix}
  X_{Kx,r} & X_{Kx,i} \\
  -X_{Kx,i} & X_{Kx,r}
\end{bmatrix}
\begin{bmatrix}
  Z_{Kz,i} \\
  Z_{Kz,r}
\end{bmatrix} \text{ for quadrant } ③. \hspace{1cm} (3.302)
and
\[
\begin{bmatrix}
Y_{K_y,r} \\
Y_{K_y,i}
\end{bmatrix} = \begin{bmatrix}
-X_{K_x,i} & X_{K_x,r} \\
X_{K_x,r} & X_{K_x,i}
\end{bmatrix} \begin{bmatrix}
Z_{K_z,i} \\
Z_{K_z,r}
\end{bmatrix}
\quad \text{for quadrant 4}. \tag{3.303}
\]

Special “stamps” are required for spectrum transform matrix row \(k_y = 0\) and column \(k_z = 0\). Consider column \(k_z = 0\). The 2-element “stamp” for the matrix entries between quadrants 1 and 4 corresponds to the case of multiplication of \(X_{K_x}\) by \(Z_0\), where \(Z_0\) is assumed to be purely real. The result can be obtained by substituting \(Z_{K_z,r} = Z_0\) and \(Z_{K_z,i} = 0\) into either (3.300) or (3.303). In both cases, the result is:
\[
\begin{bmatrix}
Y_{K_y,r} \\
Y_{K_y,i}
\end{bmatrix} = \begin{bmatrix}
X_{K_x,r} \\
X_{K_x,i}
\end{bmatrix} \cdot Z_0. \tag{3.304}
\]

A similar observation holds for the “stamp” between quadrants 2 and 3:
\[
\begin{bmatrix}
Y_{K_y,i} \\
Y_{K_y,r}
\end{bmatrix} = \begin{bmatrix}
X_{K_x,i} \\
X_{K_x,r}
\end{bmatrix} \cdot Z_0. \tag{3.305}
\]

Now consider row \(k_y = 0\). Here the output \(Y_{K_y} = Y_0\) is assumed to have an imaginary part equal to zero. For the 2-element “stamp” between quadrants 1 and 2, this corresponds to decimating the second row in (3.300) or the first row in (3.301). The result is
\[
Y_0 = \begin{bmatrix}
X_{K_x,r} \\
-X_{K_x,i}
\end{bmatrix} \begin{bmatrix}
Z_{K_z,r} \\
Z_{K_z,i}
\end{bmatrix}. \tag{3.306}
\]

Similarly, the 2-element “stamp” between quadrants 3 and 4 is given by decimating the first row of (3.302) or the second row of (3.303). The result is
\[
Y_0 = \begin{bmatrix}
-X_{K_x,i} \\
X_{K_x,r}
\end{bmatrix} \begin{bmatrix}
Z_{K_z,i} \\
Z_{K_z,r}
\end{bmatrix}. \tag{3.307}
\]

Finally, consider the intersection of row \(k_y = 0\) with column \(k_z = 0\). This corresponds to the atomic multiplication of purely real spectral vector elements, with the result \(Y_0 = X_0 Z_0\). Thus \(X_{0,r}\) is entered into the center element in Figure 3.24. Note that except for the \(k_y = 0\) and \(k_z = 0\) cases, the numbers of rows and columns are twice that of the complex case, since each complex phasor value is represented as two distinct real-valued elements. This causes offsets from the center element in lines 12–41 on page 130 to use 2 times the frequency index. Note also a clever use of the Matlab® sign function which reduces the need to handle each of the 9 “stamps” as distinct cases, reducing them to 4 cases instead.
\textbf{CONSTRUCT-STM}(X, smt, ffidx)
\begin{verbatim}
1 numsides ← 2
2 if (ffidx(1) = 0) numsides ← 1
3 X_{dci} ← \frac{1}{2}(\text{length}(X) - 1) + 1
4 nrows = \text{SIZE}(smt, 1)
5 T_x = \text{SPALLOC}(\text{length}(X), \text{length}(X), \text{nrows})
6 if (numsides = 2) and \text{not}(\text{ISREAL}(X)) \implies X is complex-valued
7    for row ← 1..nrows
8       do
9          K_y ← smt(row, 1), K_z ← smt(row, 3)
10         T_x(X_{dci} + K_y, X_{dci} + K_z) ← T_x(X_{dci} + K_y, X_{dci} + K_z) + X(K_z)
11     end
12 if (numsides = 2) and \text{ISREAL}(X) \implies X is real-valued
13    for row ← 1..nrows
14       do
15          K_y ← smt(row, 1), K_z ← smt(row, 3)
16         X_{Kx,r} ← X(X_{dci} + 2K_x - \text{SIGN}(K_x))
17         X_{Kx,i} ← 0
18         if (K_x ≠ 0) X_{Kx,i} ← X(X_{dci} + 2K_x)
19         if (K_y ≠ 0) and (K_z ≠ 0)
20            then \triangleright \text{Update 4 entries}
21             T_x(X_{dci} + 2K_y - \text{SIGN}(K_y), X_{dci} + 2K_z - \text{SIGN}(K_z)) ←
22             T_x(X_{dci} + 2K_y - \text{SIGN}(K_y), X_{dci} + 2K_z) ←
23             T_x(X_{dci} + 2K_y - \text{SIGN}(K_y), X_{dci} + 2K_z - \text{SIGN}(K_z)) ← X_{Kx,r}
24             T_x(X_{dci} + 2K_y - \text{SIGN}(K_y), X_{dci} + 2K_z) ← X_{Kx,i}
25             T_x(X_{dci} + 2K_y, X_{dci} + 2K_z - \text{SIGN}(K_z)) ←
26             T_x(X_{dci} + 2K_y, X_{dci} + 2K_z) ← X_{Kx,i}
27             T_x(X_{dci} + 2K_y, X_{dci} + 2K_z - \text{SIGN}(K_z)) ←
28             T_x(X_{dci} + 2K_y, X_{dci} + 2K_z) ← X_{Kx,r}
29        elseif (K_y = 0) and (K_z ≠ 0)
30            then \triangleright \text{Update 2 entries in row } K_y = 0
31             T_x(X_{dci}, X_{dci} + 2K_z - \text{SIGN}(K_z)) ←
32             T_x(X_{dci}, X_{dci} + 2K_z - \text{SIGN}(K_z)) ← X_{Kx,r}
33             T_x(X_{dci}, X_{dci} + 2K_z) ← T_x(X_{dci}, X_{dci} + 2K_z) ← X_{Kx,i}
34        elseif (K_y ≠ 0) and (K_z = 0)
35            then \triangleright \text{Update 2 entries in column } K_z = 0
36             T_x(X_{dci} + 2K_y - \text{SIGN}(K_y), X_{dci}) ←
37             T_x(X_{dci} + 2K_y - \text{SIGN}(K_y), X_{dci}) ← X_{Kx,r}
38             T_x(X_{dci} + 2K_y, X_{dci}) ← T_x(X_{dci} + 2K_y, X_{dci}) ← X_{Kx,i}
39        else T_x(X_{dci}, X_{dci}) ← T_x(X_{dci}, X_{dci}) + X_{Kx,r} \triangleright \text{Update the center entry}
40     end
41 \triangleright \text{Insert 1-sided real construction (see Section 3.9.5 on page 147)}
42 return [T_x]
\end{verbatim}
3.7.2 Properties of the Spectrum Transform Matrix

The 2-sided real spectrum transform matrix has no discernable structure which can be mathematically exploited for computational purposes. An examination of Figure 3.24 on page 128 reveals why: If an attempt is made to transpose the matrix, the presence of minus signs ahead of imaginary elements in columns is not reflected in transposed rows. For example, the top-most row in the element stamps listed shows real and imaginary parts of complex numbers, while the left-most column shows each of the imaginary parts negated. Thus the matrix has no transpose structural property which might be useful. Still, since it is real-valued, the double-sided spectrum transform matrix is suitable for use in systems of linear equations using Newton iteration or Krylov subspace iterative methods.

The issue of matrix structure is different where 2-sided complex spectrum transform matrices are concerned. In Section 3.3.3 an overview of the construction of the spectrum transform matrix was given for a 2-sided conjugate-symmetric 2-tone input signal for which the spectral vector \( X \) was given in (3.131) on page 51. The resulting spectrum transform matrix was given in Figure 3.10 on page 57 and is repeated here in Figure 3.25 for convenience. In the introductory discussion of how a spectrum transform matrix is constructed, it was noted

\[
\begin{array}{cccccccc}
  k_y & -6 & -5 & -4 & -3 & -2 & -1 & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
-k_6 & X_0 & - & - & - & X_{-2} & - & X_{-6} & - & - & - & - & - & - \\
-k_5 & - & X_0 & - & - & X_{-1} & X_{-2} & X_{-5} & - & - & - & - & - & - \\
-k_4 & - & - & X_0 & - & - & X_{-1} & X_{-4} & - & - & - & - & - & - \\
-k_3 & - & - & - & X_0 & X_1 & - & X_{-3} & X_{-2} & - & - & - & - & - \\
-k_1 & - & - & - & - & - & X_0 & X_{-1} & - & - & - & - & - & - \\
 0 & - & - & - & - & X_2 & X_1 & X_0 & X_{-1} & X_{-2} & - & - & - & - \\
 3 & - & - & - & - & - & - & - & X_2 & X_3 & - & X_{-1} & X_0 & - & - \\
 5 & - & - & - & - & - & - & - & X_5 & X_2 & X_1 & - & - & X_0 & - \\
\end{array}
\]

Figure 3.25: Spectrum Transform Matrix from the example in Section 3.3.3.
that a complete copy of the spectral vector $\mathbf{X}$ appears in the center (i.e. $k_z = 0$) column, and the construction algorithm assumed that each of the entries in $\mathbf{X}$ were non-zero. A closer examination of (3.131) on page 51 however, reveals that only elements $X_{-2}, X_{-1}, X_1,$ and $X_2$ were actually non-zero. The importance of this is that the spectrum mapping table algorithm will make entries only for non-zero phasor elements, and thus the spectrum transform matrix will be constructed only from the non-zero elements of $\mathbf{X}$. When the matrix in Figure 3.25 is constructed strictly according to the algorithms, the matrix in Figure 3.26 is the result. When

$$
\begin{array}{cccccccc}
  k_y & -6 & -5 & -4 & -3 & -2 & -1 & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
  k_z & & & & & & & & & & & & & \\
  0 & - & - & - & - & - & X_2 & X_1 & X_{-1} & X_{-2} & - & - & - & - \\
\end{array}
$$

Figure 3.26: Spectrum Transform Matrix for a conjugate symmetric two-tone signal.

the matrix in Figure 3.26 is constructed from the conjugate-symmetric spectral vector (3.131) on page 51 it has the interesting property that element $\mathbf{T}_x[-k_y,-k_z] = \mathbf{T}_x^*[k_y,k_z]$, where $k_y$ and $k_z$ are specific instances of the frequency indices $k_y$ and $k_z$, respectively, used to denote rows and columns in Figure 3.26. This property of conjugate-transpose symmetry about the main diagonal is specific to the class of matrices known as Hermitian matrices, and in the mathematical literature [161] the property is usually described such that $\mathbf{A}$ is said to be equal to its Hermitian $\mathbf{A}^H$, where

$$
\mathbf{A}^H = [\mathbf{A}^*]^T = [\mathbf{A}^T]^* .
$$

(3.308)

The identification of the Hermitian property in spectrum transform matrices constructed from
Theorem 3 (Hermitian Spectrum Transform Matrices). Let the spectral vector \( \mathbf{X} \) from which a spectrum transform matrix is to be constructed be conjugate-symmetric. Then the spectrum transform matrix \( \mathbf{T}_x = T[\mathbf{X}] \) is Hermitian.

Proof. Since \( \mathbf{X} \) is conjugate-symmetric, \( X_{-K_x} = X_{K_x}^* \) and \( X_0 \) must be real. \( \mathbf{T}_x \) is constructed such that it uses the frequency indexes for \( \mathbf{X} \) as indices into its row and column spaces. Since \( \mathbf{X} \) is conjugate-symmetric, then when \( \mathbf{T}_x \) is constructed, for every element of \( \mathbf{X} \) that is entered into one matrix location, its conjugate must necessarily be entered at the transpose location corresponding to the conjugate convolution product. Specifically, suppose that an atomic convolution product is given by \( Y_{-K_y} = X_{-K_x}Z_{-K_z} \). Then \( X_{-K_x} \) is entered into the spectrum transform matrix at location \( [-K_y, -K_z] \). At a later point in the spectrum mapping process, the atomic convolution product \( Y_{K_y} = X_{K_x}Z_{K_z} \) will be identified, and thus \( X_{K_x} \) will be entered into the spectrum transform matrix at location \( [K_y, K_z] \). By the nature of the frequency indexing, the locations \( [-K_y, -K_z] \) and \( [K_y, K_z] \) are transposes, and since \( \mathbf{X} \) is conjugate-symmetric, then \( X_{-K_x} = X_{K_x}^* \), so that the transposed locations contain complex conjugate elements. Therefore, \( \mathbf{T}_x \) is Hermitian.

Remark (Computational Importance of Hermitian Spectrum Transform Matrices). The Hermitian Spectrum Transform Matrix property opens up another avenue for solution of nonlinear frequency-domain behavioral analysis problems – that of eigendecomposition. A Hermitian matrix \( \mathbf{A} \) which is positive-definite may be decomposed into the product of three matrices \( [162] \),

\[
\mathbf{A} = \mathbf{U}\Lambda\mathbf{U}^H ,
\]

(3.309)

called the eigendecomposition, where \( \mathbf{U} \) is unitary so that

\[
\mathbf{U}\mathbf{U}^H = \mathbf{U}^H\mathbf{U} = \mathbf{I} ,
\]

(3.310)

and \( \Lambda \) is a diagonal matrix containing the eigenvalues of \( \mathbf{A} \). One of the properties of Hermitian matrices is that all of their eigenvalues are real \( [163] \), but if any eigenvalue is zero, the matrix is generally singular and thus not very useful. The positive-definite property is a statement that all of the eigenvalues in \( \Lambda \) are real and greater than zero.
The exploitation of (3.309) can be seen clearly in applications involving rational polynomial transfer functions. Let \( y(x) \) be the rational polynomial transfer function of a time-domain variable \( x \) where

\[
y(x) = \frac{w(x)}{z(x)} = \frac{w_0 + w_1 x + w_2 x^2 + \ldots}{z_0 + z_1 x + z_2 x^2 + \ldots} = \frac{\sum_{m} m \, w_m x^m}{\sum_{n} n \, z_n x^n}.
\]  

(3.311)

(3.312)

In order to solve a system of this sort using AOM, the numerator polynomial must be manipulated into the form of a spectral vector, while the denominator polynomial must be manipulated into an invertible spectrum transform matrix. Using AOM, the numerator result will be

\[
W = w_0 e_0 + \sum_{m} w_m T_x^{m-1} X,
\]

(3.313)

and the denominator matrix will be

\[
T_z = T[Z] = T \left[ z_0 e_0 + \sum_{n} n \, T_x^{n-1} X \right].
\]

(3.314)

The matrix in (3.314) must be inverted and then multiplied by \( W \) to obtain the final result:

\[
Y = T_z^{-1} W
\]

(3.315)

Now, if \( z_0 \gg 0 \) and \( \|X\|_1 \ll 1 \), which often happens in practice, then \( T_z \) will likely be Hermitian positive definite, and thus expressible in the form \( T_z = U \Lambda U^H \), so that

\[
T_z^{-1} = [U \Lambda U^H]^{-1} = U \Lambda^{-1} U^H,
\]

(3.316)

and, finally,

\[
Y = U \Lambda^{-1} U^H W,
\]

(3.317)

where \( \Lambda^{-1} \) is easily inverted by simply inverting the eigenvalues on the diagonal.

Remark (Cann Model Example Using Eigendecomposition). One popular behavioral modeling transfer function is the one originally proposed by Cann [164] and given by Gard [165], as

\[
y = \frac{g \, x}{\left[ 1 + \frac{g}{T} x^s \right]^s},
\]

(3.318)
where $g$ is the small-signal gain, $L$ the limiting output value, and $s$ a real parameter governing the sharpness of the knee of the output function. One of the attractions of the Cann model is its behavioral similarity to the hyperbolic tangent function, which describes the transfer function behavior of a class of fundamental communications circuits [166, 167]. Figure 3.27 shows the similarity of the Cann model to the hyperbolic tangent for varying values of the knee sharpness parameter $s$ (and with $g = 1$ and $L = 1$). Note that the Cann model with $s = 3$ is most similar to the hyperbolic tangent. Unfortunately, knee sharpness $s = 3$ has issues limiting its usefulness in time-domain modeling [168], so $s = 2$ or $s = 4$ are the nearest reliable alternatives. However, for use in AOM, the form of the Cann model function is

$$Y = g \frac{X}{T \left[ e_0 + \frac{\theta}{2} T_x^{-1} X \right]^\frac{1}{2}}. \quad (3.319)$$

If $T_x = T \left[ e_0 + \frac{\theta}{2} T_x^{-1} X \right]$ is Hermitian positive definite, then the difficulties of using the Cann model in the time domain [168] do not apply, so it is not necessary to use the absolute value form noted there. Now, for a conjugate-symmetric $X$ with $\|X\|_1 << 1$, $T_x$ will be Hermitian and likely positive definite with form $T_x = U \Lambda U^H$ so that

$$T_x^\frac{1}{2} = U \Lambda^\frac{1}{2} U^H. \quad (3.320)$$

Figure 3.27: $\tanh(x)$ and Cann models for $s = 2, 3, 4$. 
and
\[
\begin{pmatrix}
T_s^T
\end{pmatrix}^{-1} = T_s^s = U\Lambda^s U^H.
\]  
(3.321)

Thus the final form of \(Y\) is
\[
Y = g \, U\Lambda^s U^H \, X.
\]  
(3.322)

Since the diagonal matrix \(\Lambda\) will consist of only positive real numbers, odd integer values of \(s\) will pose no difficulties.

**Remark (Assuring Positive Definiteness).** One of the difficulties in practically applying eigendecomposition is assuring that a given matrix \(A\) is positive definite before proceeding to compute the complete eigendecomposition. One way of doing this is to compute the *Gershgorin’s circles* \([143, 169] \), which are circles in the complex plane within which the eigenvalues of a given matrix must fall. For a \(K \times K\) matrix, let \(R_k\) denote the radius, in the complex plane, of the circle defined by
\[
R_k = \left\{ z \in \mathbb{C} : |z - a_{kk}| \leq \sum_{j=1}^{K} |a_{kj}| \right\},
\]  
(3.323)

where \(z\) denotes any point within a circle centered at diagonal element \(a_{kk}\) of \(A\) and where \(R_k\) denotes a radius not larger than the sum of the absolute values of the matrix elements along row \(k\), excluding the diagonal element. As long as none of the points within the radius of each circle falls in the left half plane, then \(A\) is positive definite. Furthermore, since \(A\) is Hermitian for the matrices of interest in this work, it is necessary only to consider the radius of each circle along the real line since the \(a_{kk}\) and the eigenvalues are real.

This application of Gershgorin’s theorem amounts to assuring that \(A\) is diagonally-dominant.

### 3.7.3 Further Information on Eigendecomposition

The method for eigendecomposition of Hermitian matrices was first proposed by Lanczos \([170]\). Two recent publications that survey the state of the art in Hermitian matrix eigendecomposition are \([171, 172]\). Matlab® has integrated the ARPACK package \([173]\) and is capable of performing eigendecompositions on small to medium-sized matrices. See \([174]\) for examples of Matlab®

usage. Good overviews of how eigendecomposition occurs in physical systems are found in
[145, 175].

Eigenvalues of Hermitian matrices are known to be well-conditioned [174, 176] but their
corresponding eigenvectors may be ill-conditioned [177], and thus maintaining orthogonality in
the $U$ matrices within the limits of the number system may pose a challenge. The difficulty of
efficiently computing eigendecompositions has led to continued intense research in the area [178].
Note that eigendecomposition is not cost-free. In [179], it is noted that an LU decomposition
to solve a system of linear equations requires $\frac{1}{3}K^3$ operations for a $K \times K$ matrix, while the
eigendecomposition requires about $9K^3$ operations. However, the eigendecomposition must be
performed only one time, whereas solving the same system by linear iterative methods usually
requires many Newton iterations – occasionally accompanied by multiple LU decompositions as
in the case of the Spice engine [7]. So the total cost of computation may be more comparable —
when comparing one simulation environment to another — than the first glance at a particular
problem might suggest.
3.8 Dynamic Range of Two-Tone Tests

It was noted in Chapter 2 that the use of the FFT in HB methods may impose dynamic range limits upon those methods. Without implementing a complete HB environment in Matlab®, this question was investigated by creating time-domain records of two-tone sinusoidal signals, passing them through a 3rd order nonlinearity, and taking the FFT of the result. The same sinusoidal signals were also created within the AOM Toolbox and passed through the same nonlinearity.

The two sinusoidal signals were a carrier signal at 450 MHz and a swept power input signal offset by 10 kHz at 450.01 MHz. The time-domain records were 16 Megabytes in length in order to absolutely minimize spectral leakage. The carrier power was fixed at −10 dBm and the swept tone power was varied from −300 dBm to 0 dBm at 5 dBm increments. Figure 3.28 shows the results. It can be seen that the dynamic range of the AOM Toolbox is appears to be unlimited whereas the dynamic range of the FFT has a floor in the vicinity of -200 dBm (approximately -210 dBc). As a practical matter, the dynamic range of the AOM Toolbox should be considered to be limited by the machine epsilon of the underlying number system.
which for the IEEE 754-1985 standard is $2^{-52}$. Setting the amplitude of a signal of interest to this value, and assuming a 50 ohm reference impedance, the practical floor for simulated power output is $-303$ dBm.
3.9 Elements of FREDA2 Revisited

3.9.1 Background

In the course of developing the AOM Toolbox, a key reference was the work of Chang [92], which described the first large-scale implementation of AOM in the context of a circuit simulator environment. Although the 2-sided spectrum mapping table and spectrum transform matrix construction algorithms were discussed first in this work because of their relative simplicity, the 1-sided spectrum mapping table and spectrum transform matrix construction algorithms were actually implemented prior to the 2-sided versions by carefully following the algorithms as described in [90]. In this section, it will be seen that the 1-sided versions of the spectrum mapping tables and spectrum transform matrices follow from actions that can be taken when the input signal is assumed to be purely real.

In [92], a discussion was given for how to use AOM to evaluate exponential and hyperbolic tangent transfer functions. Unfortunately, the discussion left a number of questions unanswered, so some clarifications will be attempted here.

3.9.2 One-Sided Spectrum Mapping Table

If input signals (and systems) are assumed to be real, then their complex spectral vector representations will be conjugate-symmetric. As a result, the 2-sided complex spectrum transform matrix will also be Hermitian, and the output spectral vector will be conjugate-symmetric as well. These facts can be systematically exploited to reduce the 2-sided spectrum transform matrix to a single-sided form which can be useful in cases where the spectrum transform matrix will be dense. The steps required to obtain this final form will be illustrated by example.

Consider an input signal of two tones processed by a 2nd order nonlinearity. Then the spectrum transform matrix appearing in Figure 3.26 on page 132 applies. The outputs $Y_{-3}$ and $Y_3$ can be determined by inspection of the figure: For $Y_{-3}$, $X_1$ appears in column $k_z = -2$ and $X_{-2}$ appears in column $k_z = 1$. Therefore,

$$Y_{-3} = X_1 Z_{-2} + X_{-2} Z_1,$$  \hspace{1cm} (3.324)

and similarly, the output at $Y_3$ is given by

$$Y_3 = X_2 Z_{-1} + X_{-1} Z_2.$$  \hspace{1cm} (3.325)
Now, since \( Y \) is conjugate-symmetric, \( Y_{-3} = Y_3^* \), and therefore \((3.324)\) may be expressed as
\[
Y_{-3} = Y_3^* = [X_2 Z_{-1} + X_{-1} Z_2]^* \\
= X_2^* Z_{-1} + X_{-1}^* Z_2 \\
= X_2^* Z_1 + X_1 Z_2^* ,  
\]
where in \((3.328)\) the conjugate symmetry property has been used to exchange spectral content at negative frequencies in \((3.327)\) with its conjugate at the positive frequency. The result in \((3.328)\) is startling in that an output at a negative frequency has been expressed in a form using only positive-frequency spectral content. When appropriate conjugate-symmetric substitutions are made for elements of \( X \) and \( Z \) in \((3.325)\), a similar form emerges:
\[
Y_3 = X_2 Z_{-1} + X_{-1} Z_2 = X_2 Z_1^* + X_1^* Z_2 .  
\]
This suggests that it is possible to eliminate the negative frequency indices from the row and column spaces of the spectrum transform matrix and express the entire matrix-vector product using only positive frequency indices. It is possible, but with a few complications that must be pursued further.

Toward that end, first express the output of the prospective single-sided system as the sums of \((3.328)\) and \((3.329)\):
\[
Y_{|_{k_y=3}} = Y_3 + Y_{-3} = Y_3 + Y_3^* \\
= X_2 Z_1^* + X_1^* Z_2 + X_2^* Z_1 + X_1 Z_2^* \\
= [X_2 Z_1^* + X_1^* Z_1] + [X_1^* Z_2 + X_1 Z_2^*]  \\
= 2\Re[X_2^* Z_1] + 2\Re[X_1 Z_2^*]  \\
= 2\Re[X_2 Z_1^*] + 2\Re[X_1^* Z_2] .  
\]
In \((3.330)\), the pairs of product terms within each set of brackets are complex conjugates, and therefore are equal to twice the real part of either term. The question is which of \((3.331)\) or \((3.332)\) to use? The answer is furnished by considering modifications to the spectrum mapping table algorithm that must be made when the VFD Table is 1-sided. The VFD Table corresponding to the spectral vectors under consideration here can be found on page 148. A 1-sided version of that table is shown in Table 3.7 on the following page. As before, \( f_1 < f_2 \).
The two-sided spectrum mapping table algorithm worked solely by adding the VFD Table entries for each atomic convolution product and finding the output VFD. Since the input frequencies now are considered strictly positive, the subtractive intermodulation output at \( \eta^T \) must be determined explicitly by also performing subtraction operations on the pair of VFDs involved in the atomic convolution product, but with the restriction that output frequencies must be positive. Since negative frequencies are now proscribed, the effect of a frequency subtraction within the spectrum mapping algorithm will be to specify conjugate spectral content in atomic convolution products. Therefore, in (3.331) and (3.332), the presence of a conjugate form denotes that its VFD is negated.

Comparing (3.331) and (3.332), then, it can be seen that the preferable form for use in a 1-sided table is that given by (3.332) because the atomic convolution products involved in that equation will yield positive frequencies, whereas those in (3.331) will give negative frequencies. Consider the terms in (3.331) first, with the atomic convolution product followed by the corresponding VFD,

\[
X_2^* Z_1 \rightarrow - [0 \ 1] + [1 \ 0] = [1 \ -1] \quad \rightarrow \quad f_1 - f_2 < 0 \quad \text{(3.333)}
\]

\[
X_1 Z_2^* \rightarrow [1 \ 0] - [0 \ 1] = [1 \ -1] \quad \rightarrow \quad f_1 - f_2 < 0 \quad , \quad \text{(3.334)}
\]
and then the preferred form in (3.332):

\[ X_2 Z_1^* \rightarrow [0 \ 1] - [1 \ 0] = [1 -1] \rightarrow -f_1 + f_2 > 0 \]  

(3.335)

\[ X_1^* Z_2 \rightarrow - [1 \ 0] + [0 \ 1] = [1 -1] \rightarrow f_1 + f_2 > 0 \]  

(3.336)

Thus in general for the 1-sided spectrum mapping table, cases of subtractive intermodulation yielding positive frequencies will be retained, and the table must indicate that the conjugate form of the spectral vector element must be used.

Examining the atomic convolution products in (3.335), the next hurdle surfaces: The algorithm must specify that the conjugate form of \( Z_1 \) is to be used. However, \( Z \) is the input spectral vector, and for complex forms of spectrum transform matrices, only elements of \( X \) and its conjugates may be entered into the spectrum transform matrix. Furthermore, spectral vectors should be specified in a fixed input form, without the need to explicitly furnish the conjugate form on the input. The solution to this quandary is to forego the notion of a 1-sided complex form, and make the 1-sided form real-valued instead. The spectrum transform matrix will thus look like the lower right quadrant of Figure 3.24 on page 128 (i.e. quadrant ①), along with the right-hand portion of the \( k_y = 0 \) row and the bottom portion of the \( k_z = 0 \) column.

The extra information that must be included in the 1-sided spectrum mapping table can be deduced by reconsidering (3.295)–(3.300) in Section 3.7.1 and allowing for the possibility of conjugate spectral vector elements. Since the conjugate of a complex number simply negates its imaginary part, then a convenient form for describing both \( X_{Kx} \) and \( X_{Kx}^* \) is to introduce a sign indicator into its definition so that

\[ X_{Kx} = X_{Kx,r} + j s_x X_{Kx,i} \]  

(3.337)

where the sign indicator \( s_x \) is understood to be 1 to specify the \( X_{Kx} \) and \(-1\) to specify \( X_{Kx}^* \). Similarly for \( Z_{Kz} \) and \( Z_{Kz}^* \), the sign indicator \( s_z \) is introduced:

\[ Z_{Kz} = Z_{Kz,r} + j s_z Z_{Kz,i} \]  

(3.338)

Now when the product \( Y_{Kz} = X_{Kz} Z_{Kz} \) is formed by substituting (3.337) and (3.338), the following matrix form is the result:

\[
\begin{bmatrix}
Y_{Ky,r} \\
Y_{Ky,i}
\end{bmatrix} =
\begin{bmatrix}
X_{Kx,r} & -s_x s_z X_{Kx,i} \\
 s_x X_{Kx,i} & s_z X_{Kx,r}
\end{bmatrix}
\begin{bmatrix}
Z_{Kz,r} \\
Z_{Kz,i}
\end{bmatrix}.
\]

(3.339)
The matrix in (3.339) is a suitable “stamp” form for the 1-sided spectrum transform matrix (but it is not the final form – see Section 3.9.4 for that) since it separates the real-valued tuples of element $Z_{Kz}$ from its $s_z$ sign indicator, thus making it possible to include the sign indicator information for both $X_{Kx}$ and $Z_{Kz}$ into the spectrum mapping table (and subsequently into the spectrum transform matrix). Each spectrum mapping table entry for the 1-sided spectrum mapping table will thus consist of the 5-tuple $[k_y \ k_x \ s_x \ k_z \ s_z]$, and the algorithm that performs the discrete convolution operations on the 1-sided VFD Table will add VFDs, subtract the VFD for $Z$ from the VFD for $X$, and subtract the VFD for $X$ from the VFD for $Z$ for each atomic convolution product, and make the spectrum mapping table entries for all combinations with positive frequencies and appropriate 1-norms.

### 3.9.3 1-Sided Spectrum Mapping Table Construction Algorithm

Procedure `Construct-SMT` on page 124 emphasizes the elements of 2-sided spectrum mapping. Line 39 in the procedure refers to this section for the expansion of the algorithm to perform 1-sided spectrum mapping. Procedure `Construct-SMT-1s` shows the algorithmic steps that must be inserted at line 39 of Procedure `Construct-SMT` in lieu of the comment on that line.

```
1   // Insert this code at line 39 on page 124
2   if numsides == 1
3     then
4       if nzI_{kx} == nzI_{kz} and I_{kn} <> 1 // Rectification of non-DC terms
5         then
6           // Append an entry to the spectrum mapping table.
7         end
8       if nzI_{kx} > nzI_{kz} and I_{kz} > 1 // Conditionally subtract Z’s frequency from X’s
9         then
10           // Append an entry to the spectrum mapping table.
11         end
12       if nzI_{kz} > nzI_{kx} and I_{kx} > 1 // Conditionally subtract X’s frequency from Z’s
13         then
14           // Append an entry to the spectrum mapping table.
15       end
16     end
```
3.9.4 One-Sided Spectrum Transform Matrix

All of the development in Section 3.9.2 was done by considering spectral vector elements from a 2-sided spectral vector with a goal of identifying the information and final form for 1-sided spectrum mapping table entries. Section 3.9.2 deferred the final forms of the spectrum transform matrix stamps to this section, which begins by reiterating the result of (3.332),

\[
Y |_{k_y=3} = 2 \Re \left[ X_2 Z_1^* \right] + 2 \Re \left[ X_1^* Z_2 \right]
\]  \hfill (3.340)

and considering it in further depth. Notice that the contribution to the output of each atomic convolution product is twice the real part of the atomic convolution product, where the spectral vector elements appearing within the brackets correspond to those defined for a two-sided spectral vector. In procedure 3.5.4 on page 103, lines 12–15 and 26–29 indicate that the amplitudes of real-valued inputs are scaled by \( \frac{1}{2} A_0 \) as the input spectral content is split among positive and negative frequencies for 2-sided computations. Thus, the amplitude of the first atomic convolution product is given by

\[
|2 \Re [X_2 Z_1^*]| = 2 \left[ \frac{1}{2} A_{Kx} \frac{1}{2} A_{Kz} \right] = \frac{1}{2} A_{Kx} A_{Kz} .
\]  \hfill (3.341)

Notice though, that lines 38–39 on page 103 show no scaling of the input amplitudes for 1-sided computations, and thus the amplitude scale factors of \( \frac{1}{2} \) that are embedded into the double-sided vector and matrix constructions are omitted from the single-sided versions. For accurate results, it is thus necessary to introduce this scale factor into all atomic convolution products not involving spectral content at DC. Chang termed this scale factor \( \epsilon \). Thus (3.341) becomes

\[
|2 \Re [X_2 Z_1^*]| = \epsilon A_{Kx} A_{Kz} \quad \text{where} \quad \epsilon = \frac{1}{2} .
\]  \hfill (3.342)

Now consider what happens when DC terms are introduced into the atomic convolution products. Suppose first that \( K_x = 0 \) and \( K_z \neq 0 \). Then the output VFD will be \( K_y = K_z \) (i.e. it will not be \( K_y = 0 \), so the total output will be the result of summing what would be a positive and negative output in a 2-sided system) and the amplitude of the partial output will then be

\[
|2 \Re [X_0 Z_K]| = 2 \left[ A_{0x} \frac{1}{2} A_{Kz} \right] = A_{0x} A_{Kz} ,
\]  \hfill (3.343)

and thus the scale factor \( \epsilon \) for this situation will be 1. Now suppose \( K_x \neq 0 \) and \( K_z = 0 \). The output VFD will be \( K_y = K_x \), and again it will not be \( K_y = 0 \), so the total output will be the
result of summing what would be a positive and negative output in a 2-sided system so that the amplitude of the partial output will be

$$|2\Re\{X_{Kx}Z_0\}| = 2 \left| \frac{1}{2} A_{Kx} A_{0z} \right| = A_{Kx} A_{0z},$$

and again the scale factor $\epsilon$ for this situation will be 1.

Finally, consider the case of $K_x = 0$ and $K_z = 0$, which corresponds to the atomic multiplication of the two DC elements. Since this case does not correspond to one where spectral content from positive and negative frequencies would be summed in a 2-sided system, then the atomic convolution product $X_0 Z_0$ does not have the leading scale factor of two present in the other cases, so

$$|X_0 Z_0| = X_0 Z_0 = A_{0x} A_{0z},$$

and again the scale factor $\epsilon$ for this situation will be 1. Introducing the scale factor $\epsilon$ into the “stamp” in (3.339), the final form of the 4-element, 1-sided spectrum transform matrix stamp which is valid for $k_y \neq 0$ and $k_z \neq 0$ is given by

$$\begin{bmatrix} Y_{Ky,r} \\ Y_{Ky,i} \end{bmatrix} = \begin{bmatrix} \epsilon X_{Kx,r} & -s_x s_z \epsilon X_{Kx,i} \\ s_x \epsilon X_{Kx,i} & s_z \epsilon X_{Kx,r} \end{bmatrix} \begin{bmatrix} Z_{Kz,r} \\ Z_{Kz,i} \end{bmatrix}.$$  \hspace{1cm} (3.346)

This “stamp” exactly matches that described by Chang in [90], and it was universal because the construction algorithm in his work had an extra row for $K_y = 0$ and an extra column for $K_z = 0$ that were eliminated upon completion of the matrix construction. In this work, the construction algorithm comprehended these cases separately. The case of $K_z = 0$ corresponds to the DC term in $Z$ so that $Z_{Kz,i} = 0$ in (3.346), leading to the resulting 2-element stamp

$$\begin{bmatrix} Y_{Ky,r} \\ Y_{Ky,i} \end{bmatrix} = \begin{bmatrix} \epsilon X_{Kx,r} \\ s_x \epsilon X_{Kx,i} \end{bmatrix} Z_0.$$  \hspace{1cm} (3.347)

Also, the case of $K_y = 0$ corresponds to decimating the second row of (3.346) so that

$$Y_{Ky,r} = \begin{bmatrix} \epsilon X_{Kx,r} & -s_x s_z \epsilon X_{Kx,i} \end{bmatrix} Z_{Kz,i}.$$  \hspace{1cm} (3.348)

As a final note, (3.332) indicates that the real part of the result is to be taken. It must be emphasized that the final output of 1-sided systems consists only of the real part of the output spectral vector $Y$. 
3.9.5 1-Sided Spectrum Transform Matrix Construction Algorithm

Procedure CONSTRUCT-STM on page 130 emphasizes the elements of 2-sided spectrum transform matrix construction. Line 42 in the procedure refers to this section for the expansion of the algorithm to perform 1-sided spectrum transform matrix construction. Procedure CONSTRUCT-STM-1s on page 148 shows the algorithmic steps that must be inserted at line 42 of Procedure CONSTRUCT-STM in lieu of the comment on that line.
Construct-STM-1s(\ldots)

1 \triangleright \text{Insert this code at line 42 on page 130.}
2 \textbf{if} (numsides = 1) \triangleright X \text{ is 1-sided}
3 \quad \textbf{then}
4 \quad \textbf{for} row $\leftarrow 1 \ldots nrows$
5 \quad \textbf{do}
6 \qquad K_y \leftarrow \text{smt}(\text{row}, 1), \quad K_x \leftarrow \text{smt}(\text{row}, 2), \quad K_z \leftarrow \text{smt}(\text{row}, 4)
7 \qquad s_x \leftarrow \text{smt}(\text{row}, 3), \quad s_z \leftarrow \text{smt}(\text{row}, 5)
8 \qquad \textbf{if} (K_x \neq 0)
9 \qquad \quad \textbf{then}
10 \qquad \qquad X_{Kx,r} \leftarrow X(2K_x)
11 \qquad \qquad X_{Kx,i} \leftarrow X(2K_x + 1)
12 \qquad \quad \textbf{else}
13 \qquad \qquad X_{Kx,r} \leftarrow X(1)
14 \qquad \qquad X_{Kx,i} \leftarrow 0
15 \qquad \textbf{end}
16 \qquad \textbf{if} (K_x \neq 0) \text{ kwand } (K_z \neq 0)
17 \qquad \quad \textbf{then}
18 \qquad \qquad \epsilon \leftarrow 0.5
19 \qquad \quad \textbf{else}
20 \qquad \qquad \epsilon \leftarrow 1.0
21 \qquad \textbf{end}
22 \qquad \textbf{if} (K_y \neq 0) \text{ and } (K_z \neq 0)
23 \quad \textbf{then} \triangleright \text{Update 4 entries}
24 \qquad T_x(2K_y, 2K_z) \leftarrow T_x(2K_y, 2K_z) + \epsilon \cdot X_{Kx,r}
25 \qquad T_x(1 + 2K_y, 1 + 2K_z) \leftarrow T_x(1 + 2K_y, 1 + 2K_z) + s_x \cdot \epsilon \cdot X_{Kx,i}
26 \qquad T_x(2K_y, 1 + 2K_z) \leftarrow T_x(2K_y, 1 + 2K_z) - s_x \cdot s_z \cdot \epsilon \cdot X_{Kx,i}
27 \qquad T_x(1 + 2K_y, 1 + 2K_z) \leftarrow T_x(1 + 2K_y, 1 + 2K_z) + s_z \cdot X_{Kx,r}
28 \qquad \textbf{elseif} (K_y = 0) \text{ and } (K_z \neq 0)
29 \quad \textbf{then} \triangleright \text{Update two entries in row } K_y = 0
30 \qquad T_x(1, 2K_z) \leftarrow T_x(1, 2K_z) + \epsilon \cdot X_{Kx,r}
31 \qquad T_x(1, 1 + 2K_z) \leftarrow T_x(1, 1 + 2K_z) - s_x \cdot s_z \cdot \epsilon \cdot X_{Kx,i}
32 \qquad \textbf{elseif} (K_y \neq 0) \text{ and } (K_z = 0)
33 \quad \textbf{then} \triangleright \text{Update two entries in column } K_z = 0
34 \qquad T_x(2K_y, 1) \leftarrow T_x(2K_y, 1) + \epsilon \cdot X_{Kx,r}
35 \qquad T_x(1 + 2K_y, 1) \leftarrow T_x(1 + 2K_y, 1) - s_x \cdot s_z \cdot \epsilon \cdot X_{Kx,i}
36 \qquad \textbf{else} \ T_x(1, 1) \leftarrow T_x(1, 1) + \epsilon \cdot X_{Kx,r} \quad \triangleright \text{Update the center entry}
37 \qquad \textbf{end}
38 \quad \textbf{end}
39 \quad \textbf{end}
3.9.6 The Exponential Function

In his work Chang [90, 92] described an approximation for the exponential function as

\[
\exp_a(x) = \left[ 1 + \frac{x}{2^n} + \frac{1}{2!} \left( \frac{x}{2^n} \right)^2 + \ldots + \frac{1}{n!} \left( \frac{x}{2^n} \right)^n \right]^{2^n},
\]

(3.349)

where \( n \to \infty \) and the approximation error for \(|x| < 30\) was claimed to be less than 0.01% when \( n \geq 6 \). No citations were made to literature about this approximation, however, and the presentation of the form suitable for use in AOM appeared to omit some details. This subsection will attempt to fill in that missing detail.

First, the claim regarding the approximation error was investigated by implementing (3.349) in a Matlab® script and computing the relative error of \( \exp_a(x) \) to the built-in exponential function using the following:

\[
e(x) = \frac{|\exp_a(x) - \exp(x)|}{|\exp(x)|} \cdot 100\%.
\]

(3.350)

The results are shown in Figure 3.29, where it can indeed be seen that (3.349) has a relative error of less than 0.01% for \( n = 6 \) and \(|x| < 30\). For large-signal regimes, where \(|x| >> 1\), however, the results in Figure 3.29 suggest that a more suitable value of \( n \) would be 7 or 8, so
that when this function is implemented in iterative solver loops with terminating residuals of $10^{-3}$ or smaller, chances will be improved that the iterative loop will successfully terminate.

Second, some mathematical footing must be given to (3.349). A definitive paper with an irreverent title about exponentiating matrices first appeared in 1978 [91] and was updated in 2003 [182]. However, the paper is equally useful regarding exponentiating scalars as well. The Taylor series approximation $\exp_T(x)$ for $\exp(x)$ is well-known as

$$\exp_T(x) = 1 + x + \frac{1}{2}x^2 + \ldots + \frac{1}{n!}x^n,$$

where $n$ is an integer usually selected to meet some error criterion. A number of algorithms for approximating $\exp(x)$ also apply a scale factor, $m$ to the input prior to computing the Taylor expansion:

$$\exp_T(x) = \left[ \exp_T\left(\frac{x}{m}\right) \right]^m.$$

(3.352)

If $m$ is fixed at $m = 2^n$ and (3.352) is combined with (3.351), then $\exp_a(x)$ in (3.349) is the result.

In order to be useful in an AOM environment, product or quotient time-domain transfer functions are always sought, since a time domain product $x(t)z(t)$ corresponds to $X \ast Z$ and can be implemented as either $T_xZ$ or $T_zX$ given the associativity of discrete convolution (and similarly a time-domain quotient $x(t)/z(t)$ corresponds to $T_z^{-1}X$). Chang specified the output form of the exponential function for use with AOM as

$$Y = \left[ e_0 + \frac{X}{2^n} + \frac{1}{2} \left( \frac{T_x}{2^n} \right) \left( \frac{X}{2^n} \right) + \ldots + \frac{1}{n!} \left( \frac{T_x}{2^n} \right)^{n-1} \left( \frac{X}{2^n} \right)^n \right]^{2^n}.$$

(3.353)

It is fairly straightforward to see that $X/2^n$ is simply the input spectral vector with the $m = 2^n$ scale factor applied, and that

$$\frac{T_x}{2^n} = T \left[ \frac{X}{2^n} \right],$$

(3.354)

and so the computations that take place within the outer brackets of (3.353) follow. Left unexplained is how to raise the vector contents within those brackets to the power $2^n$. Begin by defining the contents within the outer brackets of (3.353) as $Y_1$:

$$Y_1 = e_0 + \frac{X}{2^n} + \frac{1}{2} \left( \frac{T_x}{2^n} \right) \left( \frac{X}{2^n} \right) + \ldots + \frac{1}{n!} \left( \frac{T_x}{2^n} \right)^{n-1} \left( \frac{X}{2^n} \right),$$

(3.355)
where $Y_1$ is computed using the method of (3.133) and (3.134) on page 58. Then create the spectrum transform matrix of $Y_1$ as $T_{y_1} = T [Y_1]$. If $y_1(t)$ is the corresponding time-domain function for $Y_1$, then it is evident that $y_2(t)$, the time domain square of $y_1(t)$, corresponds to $Y_2 = Y_1 * Y_1 = T_{y_1} Y_1$ in the frequency domain. But note that

$$
[y_1(t)]^{2^n} = [y_1^2(t)]^{2^{(n-1)}} = [y_2(t)]^{2^{(n-1)}},
$$

(3.356)

so that an instance of frequency domain convolution reduces the exponent by a factor of 2. Now let

$$
Y_2 = T_{y_1} Y_1,
$$

(3.357)

then create $T_{y_2} = T [Y_2]$ and let $Y_3 = T_{y_2} Y_2$. Note the correspondence to the time-domain function:

$$
[y_2(t)]^{2^{(n-1)}} = [y_2^2(t)]^{2^{(n-2)}} = [y_3(t)]^{2^{(n-2)}},
$$

(3.358)

where another instance of squaring has reduced the outer exponent by another factor of 2. In general, each instance of time domain squaring corresponds to a frequency domain convolution with the spectral vector result from the previous $n$, and the process ends at the $n^{th}$ step. For $n = 6$, and with $Y_1$ given by (3.355), the steps to the final output $Y$ are given by

$$
Y_2 = T_{y_1} Y_1, \quad T_{y_2} = T [Y_2], \quad m = 2^{(n-1)} = 32
$$

$$
Y_3 = T_{y_2} Y_2, \quad T_{y_3} = T [Y_3], \quad m = 2^{(n-2)} = 16
$$

$$
Y_4 = T_{y_3} Y_3, \quad T_{y_4} = T [Y_4], \quad m = 2^{(n-3)} = 8
$$

$$
Y_5 = T_{y_4} Y_4, \quad T_{y_5} = T [Y_5], \quad m = 2^{(n-4)} = 4
$$

$$
Y_6 = T_{y_5} Y_5, \quad T_{y_6} = T [Y_6], \quad m = 2^{(n-5)} = 2
$$

and, finally,

$$
Y = T_{y_6} Y_6, \quad m = 2^{(n-6)} = 1.
$$

(3.359)

In the course of this work, the length of spectral vectors has generally been dictated by the order of nonlinearity of the transfer function under consideration. However, $m = 2^6 = 64$, and the algorithm that computes the ceiling on the number of unique VFDs created (given a number of input tones and order of system nonlinearity) indicates that 64 GB (4 billion elements at 16 bytes per complex double) of storage would be needed for 6 input tones, while 1,264 GB (79
billion elements at 16 bytes) would be required for 7 tones. Thus, for the exponential function to be practical, a scheme of spectral truncation — not unlike that used in HB engines — must be employed. Chang chose to truncate his spectrums at the 2\textsuperscript{nd} order, and good results were obtained for the most part along with one anomalous result (at a higher input power) that might be attributable to overly aggressive spectral truncation. The effects of spectral truncation are illustrated in Section 4.4 but spectral truncation is not used any further in the work presented here. Strategies for choosing points of spectral truncation is specific to the problem at hand.

A few final observations are worth noting. First, note that the scaling factor $2^n$ reduces the magnitude of all of the spectral content in $X$ relative to the $e_0$ vector in (3.353). Since the presence of $e_0$ means that 1 will be added to the main diagonal of $Y_1$, the use of the scale factor shrewdly tends to assure that $Y_1$ will be diagonally-dominant and thus invertible. Second, note also that Chang’s implementation did not exponentiate matrices in the Taylor series sense described in [91]. The use of the matrix exponential function in Matlab\textsuperscript{®} with varying degrees of spectral truncation requires further research. Finally, it should be noted that the methods described here by Chang are oriented toward small, dense matrices while the AOM implementation in the present work is geared toward large-scale sparse matrices, so the method described in this section has not been implemented in the main body of this work. However, Chang’s methods are not precluded — it is possible to fool the AOM Toolbox into performing according to Chang’s methods by passing a dummy spectral vector consisting entirely of elements containing 1 (very easy to create with the Matlab\textsuperscript{®} \texttt{ones} command) to the AOM Toolbox function that creates the spectrum mapping table. This will force the creation of a full spectrum mapping table, which may then be easily reused to create all of the spectrum transform matrices indicated in (3.359) and its predecessors. An example driver script and function implementation can be found in Section E.2 of Appendix E. Note that care must be exercised in the choice of order of spectral truncation so that the capability of the computing platform is not exceeded. Refer to Section 3.4.5.1 on page 71 and particularly to Figure 3.13 on page 72, where $N$ in that figure determines the order of spectral truncation, which in combinatorial relationships with $Q$, the number of input tones, sets the size of the problem. Note that $n$ in Figure 3.29 does not affect the inherent size of the computational problem, but it does affect the accuracy of the results.
3.9.7 The Hyperbolic Tangent Function

The hyperbolic tangent function, in time-domain form, is given by

\[ \tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} = \frac{1 - \exp(-2x)}{1 + \exp(-2x)}, \tag{3.360} \]

and thus getting to a form that is useful for AOM involves the approximation in (3.349),

\[ \tanh(x) \approx v(x) = \tanh_a(x) = \frac{1 - \exp_a(-2x)}{1 + \exp_a(-2x)}. \tag{3.361} \]

In a form suitable for further manipulation in AOM, (3.361) is given by

\[ V = \frac{e_0 - Y}{e_0 + Y}, \tag{3.362} \]

where \( Y \) in (3.362) is the result obtained by substituting \(-2X\) for \( X \) in (3.355). Letting \( W = e_0 - Y \) and \( Z = e_0 + Y \),

\[ T_z = T[Z] = T[e_0 + Y], \tag{3.363} \]

and the final result is given by

\[ V = T_z^{-1}W. \tag{3.364} \]

Thus the evaluation of the hyperbolic tangent function requires all of the computational effort required to approximate an exponential function followed by a matrix inversion and a further matrix-vector product. The same issues of spectral truncation seen with the exponential function obviously apply to the hyperbolic tangent function. An example driver script and function implementation can be found in Section E.3 of Appendix E.

From the perspective of computational accuracy, the approximating function \( \tanh_a(x) \) function given in (3.361) is surprisingly accurate, considering that it is a function of the approximating exponential \( \exp_a(-2x) \). Figure 3.30a shows the error in \( v(x) = \tanh_a(x) \) relative to the built-in Matlab\textsuperscript{®} function. Note that the error floor appears around \( 10^{-14}\% \), which is not surprising since this is approximately 100 times the machine epsilon of \( 2.2 \cdot 10^{-16} \). Figure 3.30b shows the error inherent in the approximation \( \exp_a(-2x) \) alone. Compared to Figure 3.29, notice that the error at the extreme values of \( x \) is considerably higher than might be desired. However, since the approximation is a small value used in both numerator and denominator of a fraction when \( \exp_a(-2x) \) appears in \( \tanh_a(x) \), the error is effectively masked by the presence of the 1 in both numerator and denominator of \( \tanh_a(x) \).
Figure 3.30: Relative errors in function approximations.

(a): Relative Error in the approximation of $\tanh(x)$ for $n = 6, 7, 8$.

(b): Relative Error in the approximation of $\exp(-2x)$ for $n = 6, 7, 8$. 
3.9.8 Summary

The 1-sided spectrum transform matrix first developed by Chang was derived here as a simplification of the 2-sided spectrum transform matrix under the assumption that the 2-sided spectral vector used to create it had conjugate symmetry. In the absence of amplitude scale factors on the spectral vectors used in matrix-vector products by Chang, it was shown how the scale factor $\epsilon$ arises in the 1-sided spectrum transform matrices. Also, the algorithmic steps used by Chang to compute the exponential and hyperbolic tangent transfer functions were described in detail, with references to the appropriate supporting literature. Matlab® example driver scripts and functions can be found in Appendix E.

3.10 Concluding Remarks

This chapter has presented, for the first time, all of the mathematical theory underpinning the Arithmetic Operator Method in a detailed yet concise form. The keys to understanding the theory underpinning AOM are the realization that collections of sinusoids have collections of Dirac delta functions for Fourier transforms, and then understanding that the convolution of Dirac delta functions are other Dirac delta functions defined at frequencies which are the sums and differences of the original Dirac delta functions. These two facts, combined with the usual knowledge that time-domain multiplication corresponds to frequency-domain convolution, define the key bits of knowledge necessary to grasp the utility of AOM and contribute to its further development.

The act of convolution of Dirac delta functions for unique input frequencies can be distilled down to a vector form which is then used to define a unique set of vectors known as the Vector Frequency Description, which becomes the 1-to-1 basis upon which spectral vector elements (i.e. phasors) are defined. Regardless of the form of the spectral vector (1-sided real, 2-sided real, or 2-sided complex), each unique VFD in the VFD table corresponds to a phasor in the spectral vector. The spectrum mapping function implements the mechanics of Dirac delta function frequency addition and subtraction by adding and subtracting VFD table entries and creates a table indicating every valid convolution product (subject to a non-zero spectral vector element, for the case of sparse matrix implementation) in a form that can be directly used to construct a matrix for a given spectral vector which allows it to perform the convolution on another spectral vector when a matrix-vector product takes place.
Detailed algorithms and pseudocode were given for how to construct the VFD table, spectrum mapping table, and spectrum transform matrices. Storage considerations leading to limitations in the possible combinations of numbers of input tones and orders of transfer function nonlinearity were described, and a highly detailed example of VFD table construction for 3 commensurate input tones was provided. The correlation properties of spectral vectors based upon a VFD table were described, leading to the theorem that all spectral vector spectral content is uncorrelated when the input spectral content is uncorrelated. It was shown that spectrum transform matrices constructed from conjugate-symmetric 2-sided complex spectral vectors are Hermitian and therefore have an eigendecomposition, which opens up this method of solving systems of equations as an alternative to the usual linear system and Krylov subspace methods. The dynamic range of AOM was shown to be essentially unlimited, while the FFT was shown to have a definite dynamic range limitation. The mechanics used by Chang to implement exponential and hyperbolic tangent transfer functions in \textit{FREDA2} were described in detail, and suggestions for how to implement those methods in the AOM Toolbox were given along with reference implementations in Appendix E.

The AOM Toolbox in its present form is geared toward the investigation of problems with polynomial transfer functions processing a medium to large number of input tones. The essential limitations on problem sizes are governed by the combinatorial relationship of the number of input tones and the order of transfer function nonlinearity (or the order of spectral truncation, when spectral truncation is used), with reference particularly to Figure 3.13 on page 72 and the surrounding section. As a preview into the size of problems that can be modeled, in Chapter 5 a 15-tone input will be processed through a 5th order nonlinearity, in Chapter 6 a 57-tone input will be processed by a 3rd order nonlinearity, and in Chapter 7 inputs of 79 and 158 tones will be processed by a 3rd order nonlinearity.
Chapter 4

Validation of The AOM Toolbox
With a Logarithmic Amplifier Model

4.1 Introductory Remarks

Before using the AOM Toolbox to investigate larger-scale multitone problems, it will be useful to validate it on a smaller-scale problem. The validation will be done by simulating the response of a logarithmic amplifier to a two-tone input signal (with a DC bias) with the AOM Toolbox—with the AOM Toolbox simulating only the AC portion of the input signal — and then converting the AOM Toolbox output phasors into a sum of time-marching sinusoids which are merged after the AOM Toolbox simulation with the DC portion of the output signal. The results will be compared to a native Matlab® computation of the logarithmic amplifier response solely in the time domain as well as with the results of transient simulation of the circuit implementation of the amplifier using MicroSim PSpice Version 6.0. The effects of spectral truncation on the time-domain output will be observed and discussed.
4.2 Logarithmic Amplifier

4.2.1 Circuit Analysis

Logarithmic amplifiers are widely used in instrumentation circuits because of their ability to sense voltages over many orders of magnitude [183]. For example, a recently published work disclosed a logarithmic transimpedance amplifier capable of operation over 7 orders of magnitude [184]. Figure 4.1 shows a circuit diagram for the logarithmic amplifier which will be considered in this work. The amplifier in Figure 4.1 is ideal in the sense that the operational amplifier and the bipolar transistor in the feedback loop are considered to have ideal characteristics — particularly no frequency dependencies. Under these ideal assumptions, the voltage at the input terminals of the op amp tends to zero and the input currents at the op amp terminals are zero. Therefore, in Figure 4.1 the collector current in the bipolar junction transistor (BJT) is equal to the input current, \( I_{c1} = I_i \), and also \( V_o = -V_{be} \), where \( V_{be} \) is the voltage on the base-to-emitter junction of the bipolar transistor. Now, from the BJT constitutive relationship,

\[
I_{c1} = I_s \left[ \exp \left( \frac{V_{be}}{V_T} \right) - 1 \right] \approx I_s \exp \left( \frac{V_{be}}{V_T} \right),
\]

where \( I_s \) is the reverse saturation current of the transistor. In \( 4.1 \), \( V_T \) is the thermal voltage given by

\[
V_T = \frac{kT}{q},
\]

Figure 4.1: Circuit diagram of an ideal logarithmic amplifier.
where \( k \) is Boltzmann’s constant, \( T \) is the temperature in Kelvin and \( q \) is the unit electron charge. When \( T \) is 27 degrees Celsius (or approximately 300 K), \( V_T \approx 25.9 \text{ mV} \). Rearranging (4.1),

\[
\frac{I_{c1}}{I_s} = \exp \left( \frac{V_{be}}{V_T} \right) \\
\ln \left[ \frac{I_{c1}}{I_s} \right] = \frac{V_{be}}{V_T} \\
V_{be} = V_T \ln \left[ \frac{I_{c1}}{I_s} \right] \\
V_o = -V_T \ln \left[ \frac{I_{c1}}{I_s} \right].
\] (4.3)

Now, \( I_i = I_{c1} = \frac{V_i}{R} \), so that

\[
V_o = -V_T \ln \left[ \frac{V_i}{R I_s} \right].
\] (4.4)

### 4.2.2 AOM Behavioral Model and Input Signal Form

Since a suitable transfer function for use with the AOM Toolbox should be in the form of a simple or rational polynomial in the time-domain, a series expansion form for the natural logarithm must be determined. Since the natural logarithm is a transcendental function, it will have an infinite series expansion. From [124], there are three infinite series expansion forms to consider:

\[
\ln(x) = (x - 1) - \frac{1}{2}(x - 1)^2 + \frac{1}{3}(x - 1)^3 - \ldots, \quad (0 < x \leq 2), \quad (4.5)
\]

\[
\ln(1 + x) = x - \frac{1}{2}x^2 + \frac{1}{3}x^3 - \ldots, \quad -1 < x \leq 1, \quad (4.6)
\]

\[
\ln(x) = 2 \left[ \frac{x - 1}{x + 1} + \frac{1}{3} \left( \frac{x - 1}{x + 1} \right)^3 + \frac{1}{5} \left( \frac{x - 1}{x + 1} \right)^5 + \ldots \right], \quad (x > 0). \quad (4.7)
\]

While (4.7) is desirable for its wide region of convergence, in this work (4.6) is used instead due to its relative simplicity. Specifically, (4.6) may be expressed compactly by

\[
\ln(1 + x) = x - \frac{1}{2}x^2 + \frac{1}{3}x^3 - \ldots = \sum_{m=1}^{\infty} (-1)^{m+1} \frac{1}{m} x^m, \quad -1 < x \leq 1, \quad (4.8)
\]
and its Fourier transform — suitable for use as a transfer function in the AOM Toolbox— is given by

\[
\mathcal{F} \left[ \sum_{m=1}^{\infty} (-1)^{m+1} \frac{1}{m} x^m \right] = \sum_{m=1}^{\infty} (-1)^{m+1} \frac{1}{m} T_x^{m-1} X
\]

\[
= \sum_{m=1}^{\infty} (-1)^{m+1} \frac{1}{m} T_x C_x^{(m-1)} ,
\]

where \( X = \mathcal{F} [x] \). In order to be practical, the series must be terminated at some maximum index \( M \), rather than at infinity.

Let the input signal \( V_i \) be comprised of a constant DC portion, denoted by \( x_{dc} \), and an AC portion denoted by \( x_{ac}(t) \). Then

\[
V_i = x_{dc} + x_{ac}(t) = x_{dc} \left[ 1 + \frac{x_{ac}(t)}{x_{dc}} \right] .
\]

(4.10)

Note now that, aside from the scale factor \( x_{dc} \), (4.10) is in the \((1 + x)\) form required by (4.6).

If (4.10) is now substituted into (4.4), the following is obtained:

\[
V_o = -V_T \ln \left[ \frac{V_i}{RI_s} \right]
\]

\[
= -V_T \ln \left[ \frac{x(t)}{RI_s} \right]
\]

\[
= -V_T \ln \left[ \frac{x_{dc} \left( 1 + \frac{x_{ac}(t)}{x_{dc}} \right)}{RI_s} \right]
\]

\[
= -V_T \left[ \ln(x_{dc}) + \ln \left( \frac{1}{RI_s} \right) + \ln \left( 1 + \frac{x_{ac}}{x_{dc}} \right) \right] .
\]

(4.11)

Let \( y_{dc} \) and \( y_{ac}(t) \) denote the DC and time-varying parts of (4.11); then

\[
y_{dc} = -V_T \ln \left( \frac{1}{RI_s} \right) \]

(4.12)

and

\[
y_{ac}(t) = -V_T \ln \left( 1 + \frac{x_{ac}}{x_{dc}} \right) \approx -V_T \ln (1 + x) ,
\]

(4.13)

where \( x \) in (4.6) is given by \( x = x_{ac}(t)/x_{dc} \). \( y_{dc} \) in (4.12) will not be computed at all using the AOM Toolbox since it is a constant value in all of the time-domain output vectors. \( y_{ac}(t) \) will be computed by the AOM Toolbox by substituting (4.6), truncated at \( M \) terms, into (4.13),

\[
y_{ac}(t) = -V_T \ln (1 + x) \approx -V_T \sum_{m=1}^{M} (-1)^{m+1} \frac{1}{m} x^m ,
\]

(4.14)
and then taking the Fourier Transform (using (4.9)) so that $\mathcal{F}[y_{ac}(t)]$ is given by

$$Y_{ac} \approx -V_T \sum_{m=1}^{M} (-1)^{m+1} \frac{1}{m} T_x C_x^{(m-1)}.$$  \hfill (4.15)

At the completion of computations using the AOM Toolbox, $Y_{ac}$ will be a vector of complex phasor coefficients for a set of delta functions corresponding 1-to-1 with the VFD table. In the time-domain, these complex coefficients will determine the amplitude and phase for a sum of sinusoidal signals which will be added to the DC terms in (4.11).

Specifically, if the AOM Toolbox computations are done with two-sided spectra, then the conjugate symmetry of real signals will cancel the imaginary portion of the time-domain expansion, so the appropriate time-domain form for $y_{ac}$ is:

$$y_{ac}(t) = \sum_{k=-K}^{K} |Y_{ac,k}| \left[ \cos(2\pi f_k t + \phi_k) + j \sin(2\pi f_k t + \phi_k) \right]. \hfill (4.16)$$

On the other hand, if the AOM Toolbox computations are done with one-sided spectra, then it is necessary to use only the real part of the output phasors in the construction of $y_{ac}$,

$$y_{ac}(t) = \sum_{k=0}^{K} \Re[Y_{ac,k} e^{j2\pi f_k t}] = \sum_{k=0}^{K} |Y_{ac,k}| \cos(2\pi f_k t + \phi_k). \hfill (4.17)$$

In both (4.16) and (4.17), note that the index variable $k$ refers to the $k$th VFD and its corresponding complex coefficient (i.e. phasor) in $Y_{ac}$, so that $Y_{ac,k} = |Y_{ac,k}| e^{j\phi_k}$. With the output of the AOM Toolbox converted to either of the time-domain forms in (4.16) or (4.17), a time-marching simulation may be carried out to an arbitrary time duration, then combined with (4.12) to form the complete time-domain output that may be compared with results produced by other means.

One question that remains open regards the determination of when the series expansion in (4.15) is terminated, i.e. what value of $M$ is to be chosen. This is done when the time-domain waveform meets a pre-defined tolerance setting such that adding another term to the series expansion in (4.15) changes the resulting waveform at each time-domain point by less than the tolerance value. To see how this is done, one final bit of notation must be introduced into (4.16) and (4.17). Let $y_{ac,(\tilde{m})}$ denote the expansion of $y_{ac}$ in those equations where the spectral vector $Y_{ac}$ (as computed by (4.15)) has been expanded to the $\tilde{m}$th term. Specifically let the $\tilde{m}$th-term expansion of $Y_{ac}$ in (4.15) be denoted by

$$Y_{ac,(\tilde{m})} = -V_T \sum_{m=1}^{\tilde{m}} (-1)^{m+1} \frac{1}{m} T_x C_x^{(m-1)}, \hfill (4.18)$$
so that (4.16) may be written as
\[
y^{(m)}_{ac}(t) = \sum_{k=-K}^{K} \left| Y^{(m)}_{ac,k} \right| \left[ \cos(2\pi f_k t + \phi_k) + j \sin(2\pi f_k t + \phi_k) \right].
\] (4.19)
while (4.17) may be written as
\[
y^{(m)}_{ac}(t) = \sum_{k=0}^{K} \Re \left[ Y^{(m)}_{ac,k} e^{j2\pi f_k t} \right] = \sum_{k=0}^{K} \left| Y^{(m)}_{ac,k} \right| \cos(2\pi f_k t + \phi_k).
\] (4.20)
The iteration error \( e_{\tilde{m}} \) is an absolute error quantity defined as maximum absolute change in the value of the time-domain waveform over all the time-domain points between iterations, so that
\[
e_{\tilde{m}} = \max_t \left| y^{(m)}_{ac}(t) - y^{(m-1)}_{ac}(t) \right|.
\] (4.21)
The terminating value, \( M \), on expanding \( Y_{ac} \) is defined when \( e_{\tilde{m}} \) is less than the predefined tolerance value \( itol \), i.e.
\[
e_M = \max_t \left| y^{(M)}_{ac}(t) - y^{(M-1)}_{ac}(t) \right| \leq itol.
\] (4.22)
The final time-domain result of using the AOM Toolbox to compute \( V_o \) for the circuit in Figure 4.1 on page 158 will be given by combining \( y^{(M)}_{ac}(t) \) from (4.22) with \( y_{dc} \) in (4.12):
\[
V_{o,aom}(t) = y^{(M)}_{ac}(t) + y_{dc}.
\] (4.23)
This will be compared to the result of computing \( V_o \) entirely in the time-domain using the built-in Matlab® function for the natural logarithm as computed by (4.11) and the definition for \( x(t) \) given in Section 4.2.3. This result will be termed \( V_{o,ref} \). The actual error \( e_A \) between the results produced using the AOM Toolbox and the time-domain solution in Matlab® will be computed at each time-domain point, so that
\[
e_A(t) = \left| \frac{V_{o,ref}(t) - V_{o,aom}(t)}{V_{o,ref}(t)} \right| = \left| \frac{\ln \left( 1 + \frac{x_{ac}(t)}{x_{dc}} \right) - y^{(M)}_{ac}(t)}{V_{o,ref}(t)} \right|.
\] (4.25)
A similar error vector \( e_S(t) \) will be defined for comparing the PSpice output for \( V_o \), which will be termed \( V_{o,ps} \) to the AOM Toolbox results:
\[
e_S(t) = \left| \frac{V_{o,ps}(t) - V_{o,aom}(t)}{V_{o,ps}(t)} \right|.
\] (4.26)
4.2.3 Time-Domain Input Signal

In Section 4.2.2, the required form of the input signal was determined to be \( V_i = x_{dc} \left[ 1 + \frac{x_{ac}(t)}{x_{dc}} \right] \), where the conditions of (4.6) require that \( x_{dc} \) be chosen such that \( |x_{dc}| > \max |x_{ac}(t)| \). Now, if \( x_{ac} \) is a sum of \( Q \) sinusoids of equal amplitude \( A \), then

\[
x_{ac}(t) = \sum_{q=1}^{Q} A \cos(2\pi f_q t + \phi_q)
\]

so that \( \max_t |x_{ac}(t)| \leq QA \), and therefore \( x_{dc} \) is selected so that \( x_{dc} > QA \). For the purpose of the work presented here, two input tones will be used, so \( Q = 2 \), and the amplitude \( A \) of the tones will be 2 volts. \( x_{dc} \) is selected to be 10 volts, and with this selection it can be seen that \(-1 < \frac{\max |x_{ac}(t)|}{x_{dc}} \leq 1\) under all circumstances, so therefore use of (4.6) to describe the natural logarithm is valid. The frequencies of the two input signals will be \( f_1 = 900 \text{ MHz} \) and \( f_2 = 1 \text{ GHz} \). Since the output of a PSpice simulation will be used for comparison, the most accurate initial conditions for PSpice will be assured by using sines for the input voltage sources so that the initial input voltages will be 0, and so therefore both \( \phi_1 \) and \( \phi_2 \) will be set to \(-90\) degrees in the AOM Toolbox and a Matlab® reference simulated entirely in the time-domain.

4.2.4 Selection of Order of Spectral Truncation

It has already been noted that since the natural logarithm is a transcendental function, it has an infinite series expansion which must be truncated at some maximal index \( M \), rather than at infinity as shown in (4.9) on page 160 and that for the problem under consideration here \( M \) will be defined when the iteration tolerance \( itol \) in (4.22) on the previous page is met. \( M \) thus represents the maximum nonlinear order, in the time-domain, to which the function is expanded. Ideally, a VFD table terminated at the \( M^{th} \) nonlinear order would also be constructed, but when \( M \) is large, this may well be impractical, since the size of the VFD is a combinatorial function of both the number of input tones and the nonlinear order. Consider, for example, the computing limits shown in Section 3.4.5, where Figure 3.13 on page 72 shows that 10 input tones through a 10th order nonlinearity is a problem too large for a computer with 16 GBytes of memory to handle. Thus it is necessary to introduce spectral truncation — a conscious limit on the degree of spectral richness determined by a limiting order of non-linearity in the frequency-domain — to the problem of modeling the natural logarithm function. This is done by selecting a
limiting order of non-linearity, \( N \), for use in the construction of the VFD Table, where \( N \) is not necessarily the same as \( M \), the order of the series expansion in the time-domain.

For the purpose of validating the AOM Toolbox, \( N \) will be set to 5. Other choices for \( N \) will be explored in Section 4.4. With \( f_1 \) set to 900 MHz and \( f_2 \) to 1 GHz as noted in Section 4.2.3, the 2-sided Frequency-Sorted VFD table for the logarithmic amplifier modeling problem is given in Table 4.1. It contains 61 entries. Note that spectral content is allowed over a range of \( \pm 5 \) GHz, with all combinations of harmonics and intermodulation permitted within that range.

<table>
<thead>
<tr>
<th>Frequency Index ( k_x ) ((k_y, k_z))</th>
<th>VFD</th>
<th>1-Norm ( |\eta_k^T|_1 )</th>
<th>( f_1 ) Weight ( \eta_{k,1} )</th>
<th>( f_2 ) Weight ( \eta_{k,2} )</th>
<th>Output Frequency (GHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-30 )</td>
<td>( \eta_{-30}^T )</td>
<td>5</td>
<td>0</td>
<td>(-5)</td>
<td>(-5)</td>
</tr>
<tr>
<td>(-21 )</td>
<td>( \eta_{-21}^T )</td>
<td>5</td>
<td>(-1)</td>
<td>(-4)</td>
<td>(-4.9)</td>
</tr>
<tr>
<td>(-23 )</td>
<td>( \eta_{-23}^T )</td>
<td>5</td>
<td>(-2)</td>
<td>(-3)</td>
<td>(-4.8)</td>
</tr>
<tr>
<td>(-24 )</td>
<td>( \eta_{-24}^T )</td>
<td>5</td>
<td>(-3)</td>
<td>(-2)</td>
<td>(-4.7)</td>
</tr>
<tr>
<td>(-22 )</td>
<td>( \eta_{-22}^T )</td>
<td>5</td>
<td>(-4)</td>
<td>(-1)</td>
<td>(-4.6)</td>
</tr>
<tr>
<td>(-29 )</td>
<td>( \eta_{-29}^T )</td>
<td>5</td>
<td>(-5)</td>
<td>0</td>
<td>(-4.5)</td>
</tr>
<tr>
<td>(-20 )</td>
<td>( \eta_{-20}^T )</td>
<td>4</td>
<td>0</td>
<td>(-4)</td>
<td>(-4)</td>
</tr>
<tr>
<td>(-13 )</td>
<td>( \eta_{-13}^T )</td>
<td>4</td>
<td>(-1)</td>
<td>(-3)</td>
<td>(-3.9)</td>
</tr>
<tr>
<td>(-15 )</td>
<td>( \eta_{-15}^T )</td>
<td>4</td>
<td>(-2)</td>
<td>(-2)</td>
<td>(-3.8)</td>
</tr>
<tr>
<td>(-14 )</td>
<td>( \eta_{-14}^T )</td>
<td>4</td>
<td>(-3)</td>
<td>(-1)</td>
<td>(-3.7)</td>
</tr>
<tr>
<td>(-19 )</td>
<td>( \eta_{-19}^T )</td>
<td>4</td>
<td>(-4)</td>
<td>0</td>
<td>(-3.6)</td>
</tr>
<tr>
<td>(-26 )</td>
<td>( \eta_{-26}^T )</td>
<td>5</td>
<td>1</td>
<td>(-4)</td>
<td>(-3.1)</td>
</tr>
<tr>
<td>(-12 )</td>
<td>( \eta_{-12}^T )</td>
<td>3</td>
<td>0</td>
<td>(-3)</td>
<td>(-3)</td>
</tr>
<tr>
<td>(-7 )</td>
<td>( \eta_{-7}^T )</td>
<td>3</td>
<td>(-1)</td>
<td>(-2)</td>
<td>(-2.9)</td>
</tr>
<tr>
<td>(-8 )</td>
<td>( \eta_{-8}^T )</td>
<td>3</td>
<td>(-2)</td>
<td>(-1)</td>
<td>(-2.8)</td>
</tr>
<tr>
<td>(-11 )</td>
<td>( \eta_{-11}^T )</td>
<td>3</td>
<td>(-3)</td>
<td>0</td>
<td>(-2.7)</td>
</tr>
<tr>
<td>(-28 )</td>
<td>( \eta_{-28}^T )</td>
<td>5</td>
<td>(-4)</td>
<td>1</td>
<td>(-2.6)</td>
</tr>
</tbody>
</table>

Continued on next page
Table 4.1 (continued)

<table>
<thead>
<tr>
<th>Frequency Index $k_x$ $(k_y, k_z)$</th>
<th>VFD</th>
<th>1-Norm $|\eta_k^T|_1$</th>
<th>$f_1$ Weight $\eta_{k,1}$</th>
<th>$f_2$ Weight $\eta_{k,2}$</th>
<th>Output Frequency (GHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-16$ $\eta_{-16}^T$</td>
<td>4</td>
<td>1</td>
<td>-3</td>
<td>-2.1</td>
<td></td>
</tr>
<tr>
<td>$-6$ $\eta_{-6}^T$</td>
<td>2</td>
<td>0</td>
<td>-2</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>$-3$ $\eta_{-3}^T$</td>
<td>2</td>
<td>-1</td>
<td>-1</td>
<td>-1.9</td>
<td></td>
</tr>
<tr>
<td>$-5$ $\eta_{-5}^T$</td>
<td>2</td>
<td>-2</td>
<td>0</td>
<td>-1.8</td>
<td></td>
</tr>
<tr>
<td>$-17$ $\eta_{-17}^T$</td>
<td>4</td>
<td>-3</td>
<td>1</td>
<td>-1.7</td>
<td></td>
</tr>
<tr>
<td>$-25$ $\eta_{-25}^T$</td>
<td>5</td>
<td>2</td>
<td>-3</td>
<td>-1.2</td>
<td></td>
</tr>
<tr>
<td>$-9$ $\eta_{-9}^T$</td>
<td>3</td>
<td>1</td>
<td>-2</td>
<td>-1.1</td>
<td></td>
</tr>
<tr>
<td>$-2$ $\eta_{-2}^T$</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>$-1$ $\eta_{-1}^T$</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>-0.9</td>
<td></td>
</tr>
<tr>
<td>$-10$ $\eta_{-10}^T$</td>
<td>3</td>
<td>-2</td>
<td>1</td>
<td>-0.8</td>
<td></td>
</tr>
<tr>
<td>$-27$ $\eta_{-27}^T$</td>
<td>5</td>
<td>-3</td>
<td>2</td>
<td>-0.7</td>
<td></td>
</tr>
<tr>
<td>$-18$ $\eta_{-18}^T$</td>
<td>4</td>
<td>2</td>
<td>-2</td>
<td>-0.2</td>
<td></td>
</tr>
<tr>
<td>$-4$ $\eta_{-4}^T$</td>
<td>2</td>
<td>1</td>
<td>-1</td>
<td>-0.1</td>
<td></td>
</tr>
<tr>
<td>$0$ $\eta_0^T$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$4$ $\eta_4^T$</td>
<td>2</td>
<td>-1</td>
<td>1</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>$18$ $\eta_{18}^T$</td>
<td>4</td>
<td>-2</td>
<td>2</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>$27$ $\eta_{27}^T$</td>
<td>5</td>
<td>3</td>
<td>-2</td>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td>$10$ $\eta_{10}^T$</td>
<td>3</td>
<td>2</td>
<td>-1</td>
<td>0.8</td>
<td></td>
</tr>
<tr>
<td>$1$ $\eta_1^T$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.9</td>
<td></td>
</tr>
<tr>
<td>$2$ $\eta_2^T$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$9$ $\eta_9^T$</td>
<td>3</td>
<td>-1</td>
<td>2</td>
<td>1.1</td>
<td></td>
</tr>
<tr>
<td>$25$ $\eta_{25}^T$</td>
<td>5</td>
<td>-2</td>
<td>3</td>
<td>1.2</td>
<td></td>
</tr>
<tr>
<td>$17$ $\eta_{17}^T$</td>
<td>4</td>
<td>3</td>
<td>-1</td>
<td>1.7</td>
<td></td>
</tr>
<tr>
<td>$5$ $\eta_5^T$</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1.8</td>
<td></td>
</tr>
<tr>
<td>$3$ $\eta_3^T$</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1.9</td>
<td></td>
</tr>
<tr>
<td>$6$ $\eta_6^T$</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>$16$ $\eta_{16}^T$</td>
<td>4</td>
<td>-1</td>
<td>3</td>
<td>2.1</td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page
4.3 Results

The logarithmic amplifier described in Section 4.2.1 on page 158 was simulated for duration of 20 nsec, using 1 psec time-steps, in three different environments: From input to output in frequency-domain using the AOM Toolbox within Matlab® — with subsequent output inversion to the time-domain, entirely in the time-domain in Matlab® using the built-in function for the natural logarithm, and as a transient simulation in Microsim PSpice 6.0 with the circuit of Figure 4.1 on page 158.

The AOM Toolbox simulation was done using (4.23) on page 162 with $y_{ac}(t)$ as defined
in (4.19) so that the spectral vectors were two-sided complex form and were based on the 5th order \((N = 5)\) VFD Table given in Table 4.1 on page 164. The iteration tolerance \(itol\) in (4.22) on page 162 was set to \(10^{-6}\), and for this choice of \(itol\), the series expansion was terminated at the 12th term, so \(M = 12\). The Matlab\textsuperscript{®} time-domain reference was simulated using (4.11) on page 160, and the PSpice simulation was conducted with the Spice \textit{RELTOL} parameter set to \(10^{-6}\).

Figure 4.2 shows the results of simulating the logarithmic amplifier in these three environments. The waveforms in the figure essentially overlay each other, indicating good agreement among the results produced by the three environments. It can be seen that the envelope of the waveform has a period of about 10 nsec corresponding to a frequency of 100 MHz, which is the expected beat frequency behavior in the baseband for two input signals with 100 MHz spacing. The 20 nsec time duration is 20 periods of the 1 GHz input and a little more than 20 periods for the 900 MHz input. Since the waveforms overlay each other to within visual resolution in Figure 4.2, greater insight into the minor differences in the results can be obtained by considering the relative error quantities of (4.25) and (4.26). Figure 4.3 shows the very low relative error between the results produced by the AOM Toolbox and the Matlab\textsuperscript{®}.
Figure 4.3: Relative error between AOM Toolbox and Matlab® time-domain results.

As can be seen in this figure, the peak error occurs at roughly $0.045 \cdot 10^{-4} = 4.5 \cdot 10^{-6}$. This is greater than the $itol$ value, but this is to be expected since $itol$ is used to compare a series expansion of the same function while $e_A$ is the relative error between the series expansion and the actual natural logarithm function. (The unusual choice of axis scale factor $10^{-4}$ will become apparent in subsequent figures.)

Figure 4.4 on the following page shows the relative error between the results produced by the AOM Toolbox and those produced by the PSpice transient simulation.
Figure 4.4: Relative error between AOM Toolbox and PSpice transient simulation results.

Figure 4.5: Relative error between Matlab® reference and PSpice transient simulation results.
The error in the results here are slightly oscillatory, and the oscillatory behavior is most likely attributable to oscillation in the predictor-corrector numerical methods used to numerically solve the system of differential equations in the time-domain within the PSpice engine. Comparing Figure 4.3 on page 168 and Figure 4.4 on the preceding page it can be seen that the error in the AOM Toolbox results relative to the PSpice output is on the order of 100 times greater than the error in the AOM Toolbox results relative to the Matlab® time-domain reference. As a matter of assurance that the oscillatory behavior in Figure 4.4 on the previous page is contained entirely within the PSpice results, the relative error $e_R(t)$ between the PSpice and Matlab® time-domain reference results were computed as

$$e_R(t) = \left| \frac{V_{o,ref}(t) - V_{o,ps}(t)}{V_{o,ref}(t)} \right|,$$

(4.28)

and the result of the computation is shown in Figure 4.5 on the preceding page. Since the Matlab® reference result was computed directly rather than as the numerical solution of a set of time-marching differential equations, the residual oscillatory behavior can be definitively attributed entirely to the PSpice engine.

### 4.4 Effect of Spectral Truncation

The results presented in Section 4.3 truncated the frequency-domain spectra at the 5th order, which led to the VFD Table, Table 4.1 in Section 4.2.4. This produced the visually indistinguishable results shown in Figure 4.2 on page 167. It can easily be speculated that if the spectrum were truncated at a lower order of non-linearity, then the results in the time-domain wouldn’t agree quite as well. Indeed this is true, as will be illustrated here.

#### 4.4.1 Spectral Truncation to the Third Order

Consider the logarithmic amplifier modeling problem discussed in previous sections, but limit the order of spectral truncation to the 3rd order. The VFD Table for this case is shown in Table 4.2 on the following page. It contains 25 entries.
Table 4.2: Log Amplifier VFD Table for 2 Tones in a 3rd Order Nonlinearity

<table>
<thead>
<tr>
<th>Frequency Index $k_x$</th>
<th>VFD</th>
<th>$1-$Norm $|\eta^T_k|_1$</th>
<th>$f_1$ Weight $\eta_{k,1}$</th>
<th>$f_2$ Weight $\eta_{k,2}$</th>
<th>Output Frequency (GHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-12$</td>
<td>$\eta_{-12}^T$</td>
<td>3</td>
<td>0</td>
<td>-3</td>
<td>-3</td>
</tr>
<tr>
<td>$-7$</td>
<td>$\eta_{-7}^T$</td>
<td>3</td>
<td>-1</td>
<td>-2</td>
<td>-2.9</td>
</tr>
<tr>
<td>$-8$</td>
<td>$\eta_{-8}^T$</td>
<td>3</td>
<td>-2</td>
<td>-1</td>
<td>-2.8</td>
</tr>
<tr>
<td>$-11$</td>
<td>$\eta_{-11}^T$</td>
<td>3</td>
<td>-3</td>
<td>0</td>
<td>-2.7</td>
</tr>
<tr>
<td>$-6$</td>
<td>$\eta_{-6}^T$</td>
<td>2</td>
<td>0</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>$-3$</td>
<td>$\eta_{-3}^T$</td>
<td>2</td>
<td>-1</td>
<td>-1</td>
<td>-1.9</td>
</tr>
<tr>
<td>$-5$</td>
<td>$\eta_{-5}^T$</td>
<td>2</td>
<td>-2</td>
<td>0</td>
<td>-1.8</td>
</tr>
<tr>
<td>$-9$</td>
<td>$\eta_{-9}^T$</td>
<td>3</td>
<td>1</td>
<td>-2</td>
<td>-1.1</td>
</tr>
<tr>
<td>$-2$</td>
<td>$\eta_{-2}^T$</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>$-1$</td>
<td>$\eta_{-1}^T$</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>-0.9</td>
</tr>
<tr>
<td>$-10$</td>
<td>$\eta_{-10}^T$</td>
<td>3</td>
<td>-2</td>
<td>1</td>
<td>-0.8</td>
</tr>
<tr>
<td>$-4$</td>
<td>$\eta_{-4}^T$</td>
<td>2</td>
<td>1</td>
<td>-1</td>
<td>-0.1</td>
</tr>
<tr>
<td>0</td>
<td>$\eta_0^T$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>$\eta_4^T$</td>
<td>2</td>
<td>-1</td>
<td>1</td>
<td>0.1</td>
</tr>
<tr>
<td>10</td>
<td>$\eta_{10}^T$</td>
<td>3</td>
<td>2</td>
<td>-1</td>
<td>0.8</td>
</tr>
<tr>
<td>1</td>
<td>$\eta_1^T$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.9</td>
</tr>
<tr>
<td>2</td>
<td>$\eta_2^T$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>$\eta_9^T$</td>
<td>3</td>
<td>-1</td>
<td>2</td>
<td>1.1</td>
</tr>
<tr>
<td>5</td>
<td>$\eta_5^T$</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1.8</td>
</tr>
<tr>
<td>3</td>
<td>$\eta_3^T$</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1.9</td>
</tr>
<tr>
<td>6</td>
<td>$\eta_6^T$</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>$\eta_{11}^T$</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>2.7</td>
</tr>
<tr>
<td>8</td>
<td>$\eta_8^T$</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2.8</td>
</tr>
<tr>
<td>7</td>
<td>$\eta_7^T$</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2.9</td>
</tr>
<tr>
<td>12</td>
<td>$\eta_{12}^T$</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Figure 4.6: Results of spectral truncation at the 3rd order.

Figure 4.7: Relative error for 3rd order spectral truncation.
In this section, comparisons will be made only to the Matlab® time-domain reference output; the PSpice output will not be considered. The result of simulating the logarithmic amplifier with \( N = 3 \) and \( itol = 10^{-6} \) is shown in Figure 4.6 on the previous page. The AOM Toolbox simulation stopped at \( M = 11 \) terms in the series expansion, and Figure 4.7 on the preceding page shows the relative error between the time-domain waveforms produced by the AOM Toolbox output and the Matlab® reference. Good agreement is still seen between the AOM Toolbox result and the Matlab® reference in Figure 4.6 on the previous page. The maximum absolute error is roughly \( 1.4 \cdot 10^{-4} \) as shown in Figure 4.7 on the preceding page.

### 4.4.2 Spectral Truncation to the Second Order

Consider the logarithmic amplifier modeling problem again, but limit the order of spectral truncation to the 2nd order. The VFD Table for this case is shown in Table 4.3. It contains 13 entries.

<table>
<thead>
<tr>
<th>Frequency Index ( k_x ) ( (k_y, k_z) )</th>
<th>VFD</th>
<th>1-Norm ( | \eta_k^T |_1 )</th>
<th>( f_1 ) Weight ( \eta_{k,1} )</th>
<th>( f_2 ) Weight ( \eta_{k,2} )</th>
<th>Output Frequency (GHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-6)</td>
<td>( \eta_{-6}^T )</td>
<td>2</td>
<td>0</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>(-3)</td>
<td>( \eta_{-3}^T )</td>
<td>2</td>
<td>-1</td>
<td>-1</td>
<td>-1.9</td>
</tr>
<tr>
<td>(-5)</td>
<td>( \eta_{-5}^T )</td>
<td>2</td>
<td>-2</td>
<td>0</td>
<td>-1.8</td>
</tr>
<tr>
<td>(-2)</td>
<td>( \eta_{-2}^T )</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>(-1)</td>
<td>( \eta_{-1}^T )</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>-0.9</td>
</tr>
<tr>
<td>(-4)</td>
<td>( \eta_{-4}^T )</td>
<td>2</td>
<td>1</td>
<td>-1</td>
<td>-0.1</td>
</tr>
<tr>
<td>0</td>
<td>( \eta_0^T )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>( \eta_4^T )</td>
<td>2</td>
<td>-1</td>
<td>1</td>
<td>0.1</td>
</tr>
<tr>
<td>1</td>
<td>( \eta_1^T )</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.9</td>
</tr>
<tr>
<td>2</td>
<td>( \eta_2^T )</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>( \eta_5^T )</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1.8</td>
</tr>
<tr>
<td>3</td>
<td>( \eta_3^T )</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1.9</td>
</tr>
</tbody>
</table>

*Continued on next page*
As before, comparisons will be made only to the Matlab® time-domain reference output; the PSpice output will not be considered. Unlike previous results, the results presented here were not created by meeting an iteration tolerance itol parameter. Instead, the series expansion in the AOM Toolbox was forcibly terminated at the second term, so $M = 2$. Thus the results here are those obtainable for the minimum possible nonlinearity – $2^{nd}$ order time-domain expansion and $2^{nd}$ order frequency-domain spectral truncation, so $N = 2$. The results are shown in Figure 4.8, where the effects of spectral truncation are visible at the peaks and troughs of the waveforms. Figure 4.9 on the following page shows the relative error between the time-domain waveforms produced by the AOM Toolbox output and the Matlab® reference. The maximum absolute error is roughly $9.5 \cdot 10^{-4}$.
4.4.3 Further Discussion on Spectral Truncation

The results presented in Sections 4.4.1 and 4.4.2 generally show that good agreement with intrinsic time-domain functions is possible when the AOM Toolbox is used with low-order spectral truncation. However, it should not be concluded that a 2nd order spectral truncation is generally acceptable because $\max |x_{ac}/x_{dc}| \leq 0.4$, whereas the expansion for the natural logarithm was defined for absolute values up to 1. In order to set a minimum acceptable spectral truncation order, it is necessary to move the input amplitudes up to approach the model limit and also to consider not the iterative tolerance parameter, but the relative error or actual error $e_A$.

In turn, this reveals one other point that must be further emphasized, and that is that while adding more terms to the time-domain series expansion (i.e. increasing $M$ in previous sections) will allow just about any iteration tolerance value to be met, this reveals nothing about how closely the series expansion mirrors the transcendental function because the iteration tolerance parameter does not govern the number of phasors or spectral vector elements in the spectral vectors that are expanded in the time-domain. To see this, Figure 4.10 on the next page shows that the iteration tolerance parameter $itol$ can be reduced to an arbitrarily low level —
Figure 4.10: Iteration tolerance for at various orders of spectral truncation.

Figure 4.11: Actual error at various orders of spectral truncation.
regardless of the order of spectral truncation — by simply adding terms to the series expansion, i.e. by simply increasing $M$. On the other hand, when the actual error $e_A$ is considered, the effect of spectral truncation becomes readily apparent, in that the error between the series expansion and the transcendental function reaches a terminal floor value below which adding additional terms to the series expansion makes no difference. Figure 4.11 on the preceding page shows this phenomenon, and it is worth noting that the error floors appear at relatively high absolute values. For comparison, the straight-line $itol$ curves are shown. Thus in order to reduce the actual error $e_A$, it is necessary is to increase the order of spectral truncation, i.e. to increase $N$, which in turn means a larger VFD Table and a greater degree of spectral richness in the frequency-domain.

### 4.5 Concluding Remarks

The AOM Toolbox has been shown to produce results in substantial agreement with two reference implementations when its output is converted to a time-domain form and an appropriate degree of spectral richness is permitted. The comparison is to results produced by a Matlab® implementation of a corresponding transcendental function strictly in the time-domain and to the output of a PSpice transient simulation based on a netlist of a functionally-equivalent circuit to that which was behaviorally modeled in Matlab®. Thus it can be concluded that the basic functional ingredients of the AOM Toolbox—VFD Table generation, input spectral vector construction, spectrum mapping and transform matrix construction, and function evaluation— all work together properly to produce correct results.

The issue of spectral truncation addressed here is appropriate for further research work, and the work presented here suggests a framework for further investigation to characterize the AOM Toolbox for series expansions of other transcendental functions: After identifying an appropriate series expansion and an input stimulus appropriate to the transcendental function being modeled, AOM Toolbox results can be produced for various orders of spectral truncation. Those AOM Toolbox results can then be expanded in the time-domain on a time-marching sequence that spans several periods of the lowest-frequency signal in the input stimulus, and the actual error floors can be determined for each order of spectral truncation. In order for these error floors to be useful in a production modeling environment, it is important to consider input stimulus levels that approach the input’s valid maximum values in order to be sure that the
error floor represents the maximum error to be expected at a given order of spectral truncation.
Chapter 5

Modeling the Nonlinear Response to Multitones with Uncorrelated Phase

5.1 Introductory Remarks

One of the fundamental questions in RF and microwave simulation is whether or not the response of nonlinear systems to input tones that are uncorrelated is correctly captured in simulation. Simulation schemes that explicitly or implicitly require time-domain representations enforce correlation of the tones in their simulated responses, and pure time-domain simulation environments introduce the further problem of truncating the simulated time duration in order to maintain computing tractability. Both of these artifacts of time-domain simulation render both transient and harmonic balance environments unsuitable for simulating the nonlinear response of multiple uncorrelated input tones.

This chapter will present the results of behavioral modeling both the broadband and narrowband responses of a nonlinear amplifier to multitone inputs with uncorrelated phases. The goal of the work in this chapter is to show that the AOM Toolbox can predict the average power output of a nonlinear amplifying device in response to uncorrelated input tones in a single simulation — something which is not possible in simulation environments which enforce correlation, such as time-domain environments and other frequency-domain environments which also enforce correlation of spectral components occurring at the same frequency. Simulated
results produced by the AOM Toolbox will be compared to measured results of an amplifier for which a memoryless polynomial transfer function was previously extracted using two-tone techniques.

The input stimulus for measuring the response of the amplifier was a multitone oscillator which was custom-built for this work with the absence of phase locking in order to guarantee random phase behavior of the collection of oscillators. Due to minor differences in the amplitudes of each of the oscillators, their amplitude characteristics were measured separately and reproduced in the simulation environment in order to assure simulation accuracy. A short description of a multitone oscillator will be given first, followed by the details of the signal chain which were characterized and impressed upon the simulation environment, and finally a comparison of simulated and measured results.

5.2 Measurement Apparatus

5.2.1 Uncorrelated Phase Multitone Signal Generator

To facilitate this research, an uncorrelated-phase multitone signal generator was created from an ensemble of five assemblies, with each assembly containing three oscillators created from off-the-shelf parts. A detailed description of the three oscillator assembly and its characterization is given in Appendix B; a block diagram of such an assembly is shown in Figure 5.1 on the next page. An unusual aspect of the assembly is that the voltage-controlled oscillators are not controlled within phase-lock loops, but instead have their output frequencies set by simple variable resistance networks that permit a tuning range from approximately 410 to 490 MHz. The presence of 20 dB attenuators on each oscillator output, as shown in Figure 5.1, is necessary to sufficiently isolate each oscillator from injection pulling \[185\] by the others. When the oscillators are spaced at least 1 MHz apart, the artifacts of injection pulling are observed to be below −90 dBm when a 10 dB attenuator is included at the assembly’s combiner output. Despite lack of phase locking, it was found that the standard deviation of the stabilized oscillation frequency was less than 30 kHz when the assembly was housed in a crude oven — a well-sealed cardboard shipping box with a plastic sheeting lid — that typically reached a steady-state quiescent thermal environment approximately one-half hour after operating power was applied.

The outputs of five of these boxed assemblies are conducted via 0.5 meter segments of SMA cables, combined, then lowpass filtered to block harmonics of the oscillator outputs from
Figure 5.1: Simplified block diagram of a three oscillator assembly.

Further transmission. A block diagram of the complete multitone system is shown in Figure 5.2 on the following page.

5.2.2 Laboratory Equipment Setup

A block diagram of the laboratory setup is shown in Figure 5.3 on the next page and Figure 5.4 on the following page is a photo of measurements in progress. For this work, the 15 oscillators of the multitone system were set to 1 MHz nominal increments between 443 and 457 MHz. The boxed assemblies were allowed to dwell for approximately one hour with re-tuning of the oscillation frequencies at 20 minute intervals in order to stabilize them around the nominal targets. The input attenuator permits adjustment of the input power level seen by the amplifier under test, and the power attenuator on the amplifier output protects the spectrum analyzer from input overload damage. For the purposes of the work performed here, the input attenuator value was selected such that peak power of the 15 tones (assuming perfect phase alignment) at the amplifier input never exceeded $-4 \text{ dBm}$. A 0.6 meter segment of SMA cable connects the input attenuator to the amplifier under test. Measurements were taken with a 3 dB input attenuator and a 10 dB, 20 watt power attenuator. The amplifier under test is a Mini-Circuits ZHL-5W-1, a Class-A amplifier with a frequency response between 5 and 500 MHz and an
Figure 5.2: Block diagram of the multitone system of 5 three-tone assemblies. A lowpass filter is in series at the output.

Figure 5.3: Block diagram of the laboratory setup.

Figure 5.4: Laboratory photo of measurements in progress.
approximate gain of 45 dB in the vicinity of the 450 MHz center frequency of operation. The amplifier output connects to the power attenuator via a 1 meter segment of SMA cable, and the power attenuator is directly connected to the spectrum analyzer. The spectrum analyzer is an HP-8565E operating under the control of a LabView® Virtual Instrument that was developed to support this work. Each measurement sweep of the HP-8565E captures 601 points of data over a 60 MHz bandwidth thus giving a measurement resolution of 100 kHz.

5.2.3 Signal Chain Characterization for Simulation

Figure 5.5 on the next page shows the elements through which the signal from one of the multitone oscillator assemblies passes to the amplifier under test, followed by the elements after the amplifier under test to a spectrum analyzer.
Figure 5.5: Block diagram of the uncorrelated phase signal measurement chain. Only one oscillator assembly is shown, and it is understood that the 5:1 combiner takes in the outputs of four other oscillator assemblies.
Only a single oscillator assembly is shown, and it is understood that the 5:1 combiner takes in four other oscillator assembly sources not shown in the diagram. S-parameter measurements were taken over the range of 410–490 MHz for each of the 2-port passive elements in Figure 5.5 on the preceding page, and 5 separate 2-port measurements were taken for the 5:1 combiner (with unused inputs terminated in a matching impedance). Measurement reference planes are indicated by dashed vertical lines in Figure 5.5. One reference plane is at the output of each 3-oscillator assembly, with a second reference plane at the input to the amplifier under test. Figure 5.6 on the next page shows the characterized average output power levels (corresponding to the first reference plane in 5.5) of the collection of 15 oscillators used in this work. There are slight deviations in these levels within the manufacturer’s specifications. These deviations are reflected in an apparent lack of flatness in the amplified output (and matching simulation) results to be presented. Figure 5.7 on page 187 shows the signal levels as they appear at the amplifier input.

All of the components comprising the Input Signal Path in Figure 5.5 had their S-parameters converted to forward T-parameters [187], then a cascade of the T-parameters formed by the products of each device’s T-parameter matrix at each input frequency. The final T-parameter matrices were then inverted to yield S-parameters for the cascade of devices on the input chain. A similar procedure was followed for the Output Signal Path yielding S-parameter matrices for the output signal path at each frequency from 410–490 MHz. Since the amplifier under test was considered to have only non-zero $S_{21}$, only the $S_{21}$ parameters for the cascaded chains of devices were used in simulation.

### 5.3 Results and Discussion

The transfer function for the amplifier under test was previously extracted using narrowband extraction techniques with two tones in the vicinity of 450 MHz [188, 189]. The extracted transfer function was a polynomial with non-zero odd-order coefficients up to the 15th order and valid for input signals with a total input power under 0 dBm. For use in the modeling environment here, the transfer function was truncated to the 5th order; thus discarding the odd-order terms from 7 to 15 — terms which were expected to produce output power levels well below the noise floor of the available measuring equipment and also have little incremental effect upon the simulated results. Two intermodulation analysis scenarios were considered. The
Figure 5.6: Characterized average output power levels from the 15 oscillators before combining. The power levels shown here correspond to the reference plane between the oscillator assembly and the Input Signal Path component cascade in Figure 5.5 on page 184.
Figure 5.7: Characterized average output power levels from the 15 oscillators after combining. The power levels shown here correspond to the reference plane between the Input Signal Path component cascade and the AUT in Figure 5.5 on page 184.
first scenario considered the adjacent-band intermodulation (IM) distortion in the vicinity of a collection of 15 uncorrelated phase carrier signals. The second scenario considered the in-band IM distortion produced by the nonlinear amplification of a set of 14 uncorrelated phase carrier signals in the immediate vicinity of one carrier, the center frequency carrier at 450 MHz, which had been turned off. This is a common measurement scenario in the cable television industry [5], and it is one for which the industry is seeking predictive computer-aided analysis tools.

5.3.1 Adjacent-Band IM Analysis Example

Figure 5.8 on the next page shows the results of a typical adjacent-band IM measurement compared to the results simulated by the computer-aided environment when power computations assume that input carriers consist of uncorrelated phases, i.e. the power computation is done by squaring the magnitude of all phasors appearing at the same numerical frequency and summing in lieu of the typical method of assuming that phasors are correlated and squaring the magnitude of a coherent or vectorial sum. Good agreement can be seen in the linear and adjacent IM bands of Figure 5.8 with the possible exception of the far edges of the IM bands around 430 and 470 MHz. Figures 5.9 on page 191 and 5.10 on page 192 show detail views of the linear and left adjacent sideband of Figure 5.8. (The simulated 5th order IM falls well below the noise floor of the measuring equipment, thus justifying the decision to truncate the transfer function in the computer model.) Note that when the numbers of input tones (or carriers) is larger than a few, the number of nonlinearly created phasors appearing at the same frequency can become quite large.

Table 5.1 on the following page shows the number of unique phasors created (by virtue of different VFD table entries) for each of the frequencies from 415 to 450 MHz. (Due to transfer function symmetry, a reversed version of the table gives the numbers of nonlinearly created phasors from 450 to 485 MHz.) It can be seen that in the adjacent IM band there are cases where more than 1,000 phasors are being averaged, so it should be expected that power computations based on coherent or vectorial sums due to enforced correlation will vary greatly from power computations that assume uncorrelated phases.
Figure 5.8: Simulated and measured results for adjacent-band IM.

Table 5.1: Numbers of phasors averaged at adjacent-band IM frequencies.

<table>
<thead>
<tr>
<th>Frequency (MHz)</th>
<th>Number of Phasors</th>
<th>Frequency (MHz)</th>
<th>Number of Phasors</th>
</tr>
</thead>
<tbody>
<tr>
<td>415</td>
<td>1</td>
<td>433</td>
<td>718</td>
</tr>
<tr>
<td>416</td>
<td>2</td>
<td>434</td>
<td>817</td>
</tr>
<tr>
<td>417</td>
<td>5</td>
<td>435</td>
<td>917</td>
</tr>
<tr>
<td>418</td>
<td>9</td>
<td>436</td>
<td>1022</td>
</tr>
<tr>
<td>419</td>
<td>16</td>
<td>437</td>
<td>1125</td>
</tr>
<tr>
<td>420</td>
<td>25</td>
<td>438</td>
<td>1229</td>
</tr>
<tr>
<td>421</td>
<td>39</td>
<td>439</td>
<td>1327</td>
</tr>
<tr>
<td>422</td>
<td>56</td>
<td>440</td>
<td>1420</td>
</tr>
<tr>
<td>423</td>
<td>80</td>
<td>441</td>
<td>1502</td>
</tr>
<tr>
<td>424</td>
<td>109</td>
<td>442</td>
<td>1572</td>
</tr>
<tr>
<td>425</td>
<td>147</td>
<td>443</td>
<td>1715</td>
</tr>
<tr>
<td>426</td>
<td>192</td>
<td>444</td>
<td>1775</td>
</tr>
<tr>
<td>427</td>
<td>249</td>
<td>445</td>
<td>1824</td>
</tr>
</tbody>
</table>

Continued on next page
The simulated results of Figure 5.8 on the previous page can be contrasted with results typical of that for environments that enforce correlation upon uncorrelated phase input tones in their computational engines. Such environments compute the average power at a particular frequency by forming a vectorial sum of phasors and then squaring the magnitude of the resulting phasor. When the phases of the input tones are considered to be independent and identically distributed random variables, their joint probability distribution will tend to be Gaussian-distributed, and thus the ensemble average of the output of 30 simulations can be considered to give definitive results \[2\]. Thus a total of 30 separate simulations were run using the AOM Toolbox, followed by power computations using enforced correlation in order to create comparative data. Each simulation used a randomly-selected set of 15 phases or phase regimes, and each simulation used a different phase regime. Figures 5.11 on page 193, 5.12 on page 194, and 5.13 on page 195 show the results of these enforced correlation simulations, with the results of five enforced correlation phase simulations shown in each plot in the figures. For comparison, the uncorrelated phase simulation results from Figure 5.8 on the previous page are included. Note that no single simulation result using enforced correlation is in agreement with the uncorrelated phase simulation results, and that generally there are large variations in the predicted adjacent-band power levels in each simulation result when correlation is enforced.

It is only possible for the enforced correlation simulation results — typical of those produced by HB methods — to approach the uncorrelated phase results by averaging the results of many enforced correlation simulations, effectively a form of Monte Carlo analysis. Figures 5.14 on page 196, 5.15 on page 196, and 5.16 on page 197 show the effect of averaging enforced correlation simulations 1 through 8, 1 through 16, and 1 through 30, respectively. As

<table>
<thead>
<tr>
<th>Frequency (MHz)</th>
<th>Number of Phasors</th>
<th>Frequency (MHz)</th>
<th>Number of Phasors</th>
</tr>
</thead>
<tbody>
<tr>
<td>428</td>
<td>315</td>
<td>446</td>
<td>1875</td>
</tr>
<tr>
<td>429</td>
<td>382</td>
<td>447</td>
<td>1910</td>
</tr>
<tr>
<td>430</td>
<td>459</td>
<td>448</td>
<td>1943</td>
</tr>
<tr>
<td>431</td>
<td>538</td>
<td>449</td>
<td>1957</td>
</tr>
<tr>
<td>432</td>
<td>627</td>
<td>450</td>
<td>1967</td>
</tr>
</tbody>
</table>
Figure 5.9: Detail view of the linear response (i.e. the in-band output) in Figure 5.8. Excellent agreement is seen. The slight frequency offset of some of the measured points from the simulated results reflects slightly non-stationary behavior of some of the 15 oscillators.

When only 8 simulations are averaged, better results are achieved from averaging 16 simulations, while averaging 30 simulations produces results nearly identical to those produced by the single uncorrelated phase simulation. The successive improvement in the average power output results — with convergence toward the true average power as the number of separate simulations in the ensemble increases — is how enforced correlation simulation environments achieve their final accurate results.

Figures 5.17 on page 197 and 5.18 on page 198 show detail views of the linear and left adjacent IM band for the average of all 30 enforced correlation simulations, with the uncorrelated phase simulation result shown for reference. There is excellent agreement in the linear response region and good agreement in the adjacent IM band.

An alternative view of the convergence of ensemble averages of enforced correlation simulations toward the results produced by the AOM Toolbox for the uncorrelated phase simulation can be seen in Figure 5.19 on page 199, which shows the relative error in the computed power between the ensemble average of 8, 16, and 30 enforced correlation simulations and the
Figure 5.10: Detail view of the lower adjacent-band response in Figure 5.8. Good agreement can be seen. The slight frequency offset of some of the measured points from the simulated results reflects slightly non-stationary behavior of some of the 15 oscillators transformed by a predominantly 3rd order nonlinearity.

single uncorrelated phase simulation. Specifically, let $p_{ec}$ denote the power of an ensemble average at a particular frequency and let $p_{unc}$ denote the power of the single uncorrelated phase simulation. Then the relative error $e_r$ is given by

$$ e_r = \frac{p_{ec} - p_{unc}}{p_{unc}}. $$

(5.1)
Figure 5.11: Results of enforced correlation simulations 1-10 with the results from uncorrelated phase simulation for reference.
Figure 5.12: Results of enforced correlation simulations 11-20 with the results from uncorrelated phase simulation for reference.
Figure 5.13: Results of enforced correlation simulations 21-30 with the results from uncorrelated phase simulation for reference.
Figure 5.14: Results of averaging 8 enforced correlation simulations (numbers 1–8) with the results from uncorrelated phase simulation for reference.

Figure 5.15: Results of averaging 16 enforced correlation simulations (numbers 1–16) with the results from uncorrelated phase simulation for reference.
Figure 5.16: Results of averaging all 30 enforced correlation simulations with the results from uncorrelated phase simulation for reference.

Figure 5.17: Linear response detail of the average. A detail view of the linear response of the results of averaging all 30 enforced correlation simulations is shown. The results from uncorrelated phase simulation are shown for reference.
Figure 5.18: Left adjacent IM band response detail of the average. A detail view of the left adjacent IM band response of the results of averaging all 30 enforced correlation simulations is shown. The results from uncorrelated phase simulation are shown for reference.
Figure 5.19: Relative error between ensemble and a single simulation. The relative error between ensemble averages of 8, 16, and 30 enforced correlation simulations and the single uncorrelated phase simulation is shown. Note that the high relative error readings can be misleading because the relative error is a ratio of small numbers.
5.3.2 Narrow-Band IM Analysis Example

In this example, the stimulus consisted of 14 of the same 15 carriers used in Section 5.3.1. The carrier at 450 MHz was turned off, and measurements were made of the narrow-band IM in the vicinity of the 450 MHz signal. Due to the narrow-band nature of the measurement, it was not assumed that the carriers were set at 1 MHz nominal increments as in the previous example. Instead, two measurements were made, with the first made over a range of 442.5 to 457.5 MHz solely for the purpose of obtaining the instantaneous oscillating frequencies from the linear response. The second measurement, made immediately following the first, measured the narrow-band IM over a 1.2 MHz bandwidth centered about 450 MHz. Since 601 points were captured in each narrow-band measurement, the spacing of spectral content in the measured result here is 2 kHz. Table 5.2 shows the nominal and actual measured frequencies, where the actual frequencies were those used in the AOM Toolbox environment to obtain the simulated results.

<table>
<thead>
<tr>
<th>Nominal Frequency (MHz)</th>
<th>Measured Frequency (MHz)</th>
<th>Nominal Frequency (MHz)</th>
<th>Measured Frequency (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>443</td>
<td>443.000</td>
<td>451</td>
<td>451.050</td>
</tr>
<tr>
<td>444</td>
<td>444.000</td>
<td>452</td>
<td>452.000</td>
</tr>
<tr>
<td>445</td>
<td>445.000</td>
<td>453</td>
<td>452.975</td>
</tr>
<tr>
<td>456</td>
<td>445.950</td>
<td>454</td>
<td>453.950</td>
</tr>
<tr>
<td>447</td>
<td>446.975</td>
<td>455</td>
<td>454.950</td>
</tr>
<tr>
<td>448</td>
<td>448.000</td>
<td>456</td>
<td>455.950</td>
</tr>
<tr>
<td>449</td>
<td>449.050</td>
<td>457</td>
<td>456.950</td>
</tr>
</tbody>
</table>

Note that the measured frequencies of oscillation are not commensurate, and thus it would be challenging to perform a harmonic balance simulation using them. For example, at the 1 MHz nominal spacing, a reasonably accurate HB analysis could probably be performed with as few
as 32 to 64 points, but when the input tones are permitted to occur at 25 kHz increments, 600 points minimum are necessary, and probably 1,024 points would be used. However, the measured frequencies in Table 5.2 are handled by the AOM Toolbox with no more difficulty than the nominal ones. In the AOM Toolbox environment, 15 carrier frequencies were again considered as the input, but the amplitude of the center tone at 450 MHz was set to zero. Figure 5.20 shows the results of the AOM Toolbox simulation in the narrowband region around 450 MHz. Good agreement is seen in the area above the noise floor of the measuring equipment, allowing for the fact that continuous drift (of up to 30 kHz) in the carrier frequencies of the sources results in a “fill-in” effect in the measured results, where the measurement interval is only 2 kHz. The AOM Toolbox also predicts 5th order spectral content below the noise floor of the measuring equipment. Table 5.3 provides further data on the results produced by the AOM Toolbox, and here it can be seen that all of the simulated results falling below the noise floor of the measurements are comprised entirely of 5th order spectral content.
<table>
<thead>
<tr>
<th>Frequency (MHz)</th>
<th>Simulated Power (dBm)</th>
<th>Number of 3rd Order Phasors</th>
<th>Number of 5th Order Phasors</th>
</tr>
</thead>
<tbody>
<tr>
<td>449.750</td>
<td>-133.4304</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>449.775</td>
<td>-121.7008</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>449.800</td>
<td>-119.6205</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>449.825</td>
<td>-118.4364</td>
<td>0</td>
<td>17</td>
</tr>
<tr>
<td>449.850</td>
<td>-71.3629</td>
<td>1</td>
<td>68</td>
</tr>
<tr>
<td>449.875</td>
<td>-68.9911</td>
<td>2</td>
<td>67</td>
</tr>
<tr>
<td>449.900</td>
<td>-64.7140</td>
<td>7</td>
<td>155</td>
</tr>
<tr>
<td>449.925</td>
<td>-68.1083</td>
<td>5</td>
<td>152</td>
</tr>
<tr>
<td>449.950</td>
<td>-63.0783</td>
<td>12</td>
<td>254</td>
</tr>
<tr>
<td>449.975</td>
<td>-64.2181</td>
<td>6</td>
<td>234</td>
</tr>
<tr>
<td>450.000</td>
<td>-62.3139</td>
<td>12</td>
<td>256</td>
</tr>
<tr>
<td>450.025</td>
<td>-63.1690</td>
<td>7</td>
<td>185</td>
</tr>
<tr>
<td>450.050</td>
<td>-61.4909</td>
<td>13</td>
<td>178</td>
</tr>
<tr>
<td>450.075</td>
<td>-66.4031</td>
<td>4</td>
<td>96</td>
</tr>
<tr>
<td>450.100</td>
<td>-70.5952</td>
<td>4</td>
<td>93</td>
</tr>
<tr>
<td>450.125</td>
<td>-113.0054</td>
<td>0</td>
<td>42</td>
</tr>
<tr>
<td>450.150</td>
<td>-112.7395</td>
<td>0</td>
<td>48</td>
</tr>
<tr>
<td>450.175</td>
<td>-120.1977</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>450.200</td>
<td>-120.7672</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>450.225</td>
<td>-127.0383</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>450.250</td>
<td>-127.4075</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

### 5.3.3 Phase Invariance of Simulated Average Power Computations

With the comparison of simulated results to measurements in Section 5.3.1 establishing the validity of using the AOM Toolbox to simulate the average power of the nonlinear response to multitone inputs with uncorrelated phase in one simulation, an alternative perspective can be
gained by comparing the average power output spectral densities for each of the 30 simulations. If the average power is truly invariant of the input phases, then plotting the average power spectral density as computed in each of the 30 simulations should yield results substantially in agreement.

Figure 5.21 shows the results of plotting the average output power computed in each of the 30 simulations on one plot when 1-sided spectral vectors are used (as they were for all of the results in Section 5.3.1). Reasonably good agreement is seen, but there are noticeable differences in the average power levels in the distortion sidebands. To quantify these differences, for each output frequency, the ensemble of simulations was searched to find the simulations having the minimum and maximum power level, and their difference is shown in the plot in Figure 5.22 on the following page. Ironically, the greatest differences in the power levels occur within the linear region, although the variations are so tiny compared to the linear output power levels that they are not noticeable in Figure 5.21. The variations in the outermost edges of the adjacent band 3rd order IM distortion in Figure 5.21 can be attributed to the fact that, at these points, very few phasors are being averaged.

![Figure 5.21: Average power spectral density results from 30 simulations in Section 5.3.1 using 1-sided spectral vectors. In order not to obscure the plurality of spectral density curves, marks designating the power levels of the individual curves at each frequency are omitted.](image-url)
Figure 5.22: Absolute difference between the minimum and maximum powers at each computed frequency within the ensemble of 30 simulation results when 1-sided spectral vectors are used.

When complex-valued 2-sided spectral vectors are deployed in the 30 simulations of Section 5.3.1 the average power spectral density is nearly identical for each simulation. Figure 5.23 on the following page again shows the result of plotting the average output power computed in each of the 30 simulations, but here, 2-sided spectral vectors are used. It is not possible to discern distinct power spectral density curves because the results are nearly indistinguishable. Figure 5.24 on page 206 shows the difference between the minimum and maximum power levels within the ensemble at each output frequency. Here the differences are minute, and they are confined to the linear response region and well below the machine epsilon of $2.2 \cdot 10^{-16}$.

As was noted earlier, at the outer edges of adjacent IM bands in Figure 5.22 very few phasors are being averaged. Furthermore, those phasors were computed from 1-sided spectral vectors and 1-sided spectrum transform matrices where the spectrum transform matrices included additional conjugate phasor entries (compared to the 2-sided form) in order to account for subtractive intermodulation. Thus one possible explanation for the comparatively large power excursions in the 1-sided spectral vector simulations are differences created by the addition of products involving the conjugate phasors. Since the magnitudes of the numbers involved in these calculations is already small, introducing more sums involving the conjugates
may cause sign cancellation that does not occur in the 2-sided case. It should be noted that previous work on smaller-scale problems, such as the logarithmic amplifier modeling problem of Chapter 4 did not reveal noticeable differences between 1-sided and 2-sided results.

Figure 5.23: Average power spectral density results from 30 simulations in Section 5.3.1 using 2-sided spectral vectors. In contrast to Figure 5.21 on page 203, the results here overlay each other indistinguishably.
Figure 5.24: Absolute difference between the minimum and maximum powers at each computed frequency within the ensemble of 30 simulation results when 2-sided spectral vectors are used. The differences are minute.
5.3.4 Separating Correlated and Uncorrelated Distortion

In Section 3.5.5, it was shown that the spectral content corresponding to each VFD is uncorrelated from spectral content at all other VFDs, and also that spectral content occurring at the same VFD was correlated, regardless of the order of nonlinearity at which the spectral content is created. These properties make separation of correlated and uncorrelated IM a simple matter for the AOM Toolbox.

Specifically, to separate 3rd order correlated IM from uncorrelated in-band IM, it is only necessary to exclude the 3rd order phasor occurring at the same VFD as the linear output from the computation of average power at the same frequency; all of spectral content occurring at every other VFD with same numerical frequency will be uncorrelated IM. The 3rd order output spectral content from Section 5.3.1 was separated in this fashion, with the results shown in Figure 5.25. Figure 5.26 on the next page shows a magnified view of co-channel distortion. Good agreement can be seen between the results predicted here and those obtained by applying analytical cross-correlation methods to a memoryless nonlinearity in [158]. Specifically, the correlated power is several dB higher than the uncorrelated power at each frequency in the

![Figure 5.25: Adjacent and co-channel band view for measured source amplitude. The average power along with separation of co-channel correlated and uncorrelated 3rd order IM power are shown when the input is the set sources in Figure 5.7.](image)
Figure 5.26: Magnification of the co-channel band view for measured source amplitude. A magnification of the co-channel band showing correlated and uncorrelated 3rd order IM power is shown when the input is the set of sources in Figure 5.7.

co-channel band. In [158], the correlated power is flat across the band, while here there is a 1.5 dB variation. This is due to the fact that there is a 0.6 dB variation in the amplitudes of the 15-tone physical source used and an approximate 0.2 dB in-band loss variation in the cable and combiner chain (and reflected in the simulation environment here) from the sources to the amplifier input, while in [158] the sources were of equal amplitude.

In order to facilitate a better qualitative comparison of the results here with those in [158], the simulation producing the results shown in Figure 5.25 was re-run with one slight alteration — rather than using the characterized exact amplitudes of real oscillators as shown in Figure 5.7 on page 187, the input amplitudes the simulated oscillators were set to the average value of the set in Figure 5.7. Figure 5.27 on the next page shows the total response of the amplifier model to this equal-amplitude stimulus, while Figure 5.28 on page 210 shows a detail of the co-channel band. Other than the power scale, Figure 5.27 on the next page is a qualitative replica of Figure 12 appearing in [158]. Furthermore, it is worth noting that the power difference between the correlated and uncorrelated IM power in Figure 5.28 at the center frequency of 450 MHz is 4.8 dB. This power difference between correlated and uncorrelated IM power for
Figure 5.27: Adjacent and co-channel band view for average source amplitude. The average power along with separation of co-channel correlated and uncorrelated 3rd order IM power are shown when the input is a set of fictitious sources with input amplitudes equal to the average of those in Figure 5.7.

A 15-tone stimulus is in good agreement with the 4.43 dB difference predicted by the analysis for a 3-tone stimulus in Section 3.5.5.3.
Figure 5.28: Magnification of the co-channel band view for average source amplitude. A magnification of the co-channel band showing correlated and uncorrelated 3rd order IM power is shown when the input is a set of fictitious sources with input amplitudes equal to the average of those in Figure 5.7.
5.3.5 Computational Considerations

The use of the AOM Toolbox as a simulation environment entails two sets of computational costs. The first is a *set-up tabulation* cost that includes the computation of the VFD and Spectrum Mapping Tables. These two tables are solely a function of the vector of input frequencies, but are independent of the amplitudes and phases of the input signals, so their costs can be amortized over multiple simulations. The second kind of computational cost, called the *evaluation cost*, is incurred on each execution of the environment, and includes the cost of creating the input spectral vector (negligible, so it is omitted), the Spectrum Transform Matrix, and the matrix-vector multiplications necessary to evaluate the nonlinear transfer function.

Table 5.4 shows these costs in the form of time of computation on an AMD Athlon X2 4400 platform with 4 GBytes of DRAM running a 64-bit version of Matlab® under a Linux operating system (and with most of the CPU time devoted to Matlab®) for the two example scenarios given in Sections 5.3.1 and 5.3.2. The *set-up tabulation* times of roughly 23 minutes dominate in both cases, but note that this cost was incurred only once in order to compute the 30 coherent phase simulations used in the adjacent-band IM modeling scenario. The *evaluation* times for both examples are between 9 and 10 minutes.

<table>
<thead>
<tr>
<th>Computation Item</th>
<th>Adjacent-Band IM Computing time (sec)</th>
<th>Narrow-Band IM Computing time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VFD Table</td>
<td>199.92</td>
<td>187.74</td>
</tr>
<tr>
<td>Spectrum Mapping Table</td>
<td>1217.40</td>
<td>1147.20</td>
</tr>
<tr>
<td><strong>Set-Up Cost</strong></td>
<td>1417.32</td>
<td>1334.94</td>
</tr>
<tr>
<td>Spectrum Transform Matrix</td>
<td>576.65</td>
<td>548.49</td>
</tr>
<tr>
<td>Function Evaluation</td>
<td>0.53</td>
<td>0.49</td>
</tr>
<tr>
<td><strong>Evaluation Cost</strong></td>
<td>577.18</td>
<td>548.98</td>
</tr>
<tr>
<td><strong>Totals (1 run)</strong></td>
<td>1994.50</td>
<td>1883.92</td>
</tr>
</tbody>
</table>
The very low *Function Evaluation* times shown in Table 5.4 are attributable to Matlab®’s superior sparse matrix implementation and to the fact that the Spectrum Transform matrices for both problems were very sparse – the matrix for adjacent-band IM example had only 0.39% non-zeros, while the matrix for the narrow-band IM example had only 0.36% non-zeros. It is also worth noting that the adjacent-band IM example, in which the input signals were nominally spaced at 1 MHz intervals, had 123,276 unique VFD table entries, but only 356 unique numerical frequencies. For the narrow-band IM example, there were 123,063 unique VFD table entries mapping to 3,994 unique numerical frequencies as a result of using the more exacting measured carrier frequencies (with 25 kHz measurement increments) in lieu of the nominal 1 MHz spacing.

### 5.4 Concluding Remarks

The major result of the work in this chapter concerns the simulation of circuits excited by multiple uncorrelated tones. By creating a vector description (the VFD table) of the frequency-domain and using this underlying vector description as a further decomposition of the numerical frequency, it is possible to appropriately combine uncorrelated nonlinear interactions of the input tones to correctly model the response of the system, since spectral vectors based upon a VFD table will create uncorrelated output spectral content when the inputs are uncorrelated.

For the case of adjacent-band IM distortion modeling, it was shown that the results produced by the AOM Toolbox are essentially identical to those produced by averaging the results of multiple simulations performed using conventional techniques that enforce phase-correlation of the tones. Such simulations are performed with random initial phases of the tones, and by averaging the ensemble of results of multiple simulations the same results as a single AOM Toolbox simulation are obtained. The phase-invariance of the power spectral densities of the simulations using the AOM Toolbox was shown by plotting each simulation result on a single figure. Agreement was reasonably good for the 1-sided spectral vector version, while the results for 2-sided complex spectral vectors were indistinguishable. A second example of narrow-band IM distortion modeling accurately predicted the in-band IM that might be seen in cable television channel excited by intermodulation among 14 other channels. The AOM Toolbox has proven to be useful for predicting the results of broad-band and narrow-band intermodulation scenarios.

The easy separation of correlated and uncorrelated output spectral content with the
AOM Toolbox is a natural consequence of using the VFD table to define the output frequency locations. The results of separating the in-band correlated and uncorrelated IM distortion were presented from the first simulation were shown, and good agreement was seen with previous efforts.

It should be noted that the transfer function used in this work was in the form of a simple polynomial, and that a rational polynomial form might not be expected to produce such conclusive results because of the necessity to invert a spectrum transform matrix (for the denominator polynomial) and left multiply it by the numerator output spectral vector, thus introducing a numerical effect akin to enforced correlation. If a transfer function in rational polynomial form is amenable to division, the resulting quotient polynomial could be deployed effectively. Another possibility might be to assure that the matrix to be inverted is strongly diagonally-dominant so as to minimize any numerically-enforced correlation. A smaller scale version of this problem could probably be solved using the eigendecomposition technique noted in Section 3.7.2.
Chapter 6

Modeling the Response of a Nonlinear Amplifier to a Linear FM Chirp

6.1 Introductory Remarks

In this chapter, the AOM Toolbox will be applied to modeling the transmission of a linear Frequency Modulation (LFM) Chirp signal. Such signals are not sinusoidal in the sense of oscillating at fixed frequencies, but instead are formed by linearly increasing or decreasing the instantaneous frequency of a pulse envelope. They are popular in radar systems because LFM pulses received due to target reflection can be compressed in time by filters on the radar receiver, which effectively increases the range of radar detection \[190, 191\]. The results produced by the AOM Toolbox will be compared to the discrete Fourier Transform of time-domain results produced using the \textit{fREEDA}™ simulator. The linear FM chirp source will be nonlinearly amplified by a Filtronics LMA-411 PHEMT amplifier \[192\] and then bandpass-filtered. Results will be presented before and after the filter. Owing to the fact that linear FM chirp signals are inherently non-stationary, the notion of correlated and uncorrelated distortions being produced makes little sense, so the results presented here will be in terms of total output. The results in this chapter also make effective use of the \(z\)-domain filter modeling methodology reported in
Appendix A

6.2 Mathematical Analysis

6.2.1 Fourier Analysis of a Linear FM Chirp Signal

One of the unexpected challenges in this work was assuring that the spectra of the inputs to both the AOM Toolbox and fREEDA™ occurred at the same power levels. This was complicated by the fact that the readily available literature \[4, 193\] on LFM chirp signals omits several of the steps necessary to find the Fourier transform of the time-domain form of an LFM chirp. Thus it was necessary to derive this transform and then compare the results it predicted with those obtained by performing the FFT on the time-domain form of the input signal. As will be seen, the effects of accounting for necessary scale factors leads to input stimulus signals occurring at very low power levels.

The starting point for this work is the linear FM chirp signal, which is given in the time-domain by

\[
s(t) = A \Pi \left( \frac{t}{\tau} \right) \cos \left[ 2\pi f_0 t + \frac{\mu}{2} t^2 + \phi_0 \right],
\]

(6.1)

where \( A \) is the amplitude, \( \tau \) the duration of the pulse signal, \( f_0 \) the carrier frequency of the signal, and \( \mu \) is a constant known as the chirp rate which is given by

\[
\mu = \frac{2\pi \Delta f}{\tau},
\]

(6.2)

where \( \Delta f \) is the range of frequencies over which chirping takes place. For an upchirp or linearly increasing frequency modulation, the quantities \( A, \tau, \) and \( \Delta f \) will all be positive, while for a downchirp, \( \Delta f \) will be negative. (6.1) is the same as that given in \[4\] except that it uses Hertzian rather than radian frequencies. Note that the presence of the \( \Pi \) function (see \( 3.39 \) on page \( 35 \)) in (6.1) necessarily defines the signal \( s(t) \) over the time-domain limits \(-\tau/2 \leq t \leq \tau/2\). In order to focus on just the linear FM chirp signal, a complex envelope form \( g(t) \) will be analytically transformed, where

\[
g(t) = A \Pi \left( \frac{t}{\tau} \right) e^{j(2\pi f_0 t + \phi_0)}
\]

(6.3)

and so

\[
s(t) = \Re \left[ g(t) e^{j(2\pi f_0 t + j\phi_0)} \right].
\]

(6.4)
The Fourier Transform $G(f)$ of $g(t)$ will thus define the complex baseband form of the linear FM chirp signal. $G(f)$ is defined by

$$G(f) = \int_{-\infty}^{\infty} g(t)e^{-j2\pi ft}dt$$

$$= \int_{-\infty}^{\infty} A\Pi\left(\frac{t}{\tau}\right)e^{j\frac{\mu}{2}t^2}e^{-j2\pi ft}dt$$

$$= \int_{-\frac{\tau}{2}}^{\frac{\tau}{2}} A e^{j\frac{\mu}{2}t^2}e^{-j2\pi ft}dt.$$  \hspace{1cm} (6.5)

The key to further progress at this point is to complete the square in the exponent, as suggested in [194]. Let the exponent in the complex exponential in (6.5) be denoted

$$p(t) = \frac{\mu}{2}t^2 - 2\pi ft.$$  \hspace{1cm} (6.6)

The form sought is

$$p(t) = at^2 + bt = a\left(t + \frac{b}{2a}\right)^2 - \frac{b^2}{4a},$$  \hspace{1cm} (6.7)

and by comparing (6.7) to (6.6) it is evident so that $a = \mu/2$ and $b = -2\pi f$. Substituting these into (6.6),

$$p(t) = \frac{\mu}{2} \left( t - \frac{2\pi f}{\mu} \right)^2 - \frac{(2\pi f)^2}{2\mu}$$

$$= \frac{\mu}{2} \left[ t - \frac{2\pi f}{\mu} \right]^2 - \frac{2\pi^2 f^2}{\mu}$$

$$= \left[ \sqrt{\frac{\mu}{2}} \left( t - \frac{2\pi f}{\mu} \right) \right]^2 - \frac{2\pi^2 f^2}{\mu}.$$  \hspace{1cm} (6.8)

Note that the quantity $-2\pi^2 f^2/\mu$ does not contain $t$ and so therefore its complex exponential is a constant with respect to the variable of integration (time), so that it may be moved outside the integral in (6.5). Also, at this point, define the quantity being squared in (6.8) as $x$, so that

$$x^2 = \left[ \sqrt{\frac{\mu}{2}} \left( t - \frac{2\pi f}{\mu} \right) \right]^2$$  \hspace{1cm} (6.9)

and thus

$$x = \sqrt{\frac{\mu}{2}} \left( t - \frac{2\pi f}{\mu} \right).$$  \hspace{1cm} (6.10)
\( x \) will define a change of variable of integration, so that
\[
dx = \sqrt{\frac{\mu}{2}} dt
\] (6.11)
and therefore
\[
dt = \sqrt{\frac{2}{\mu}} dx
= \sqrt{\frac{2\tau}{2\pi \Delta f}} dx
= \sqrt{\frac{\tau}{\pi \Delta f}} dx
= \sqrt{\frac{\tau^2}{\pi \Delta f \tau}} dx
= \tau \sqrt{\frac{1}{\pi \Delta f \tau}} dx.
\] (6.12)
The limits of integration will change as well. The lower limit will be
\[
x_1 = x|_{t=-\tau/2}
= \sqrt{\frac{\mu}{2}} \left( \frac{\tau}{2} - \frac{2\pi f}{\mu} \right)
= -\sqrt{\frac{\pi \Delta f}{\tau}} \cdot \tau \left( \frac{1}{2} + \frac{f}{\Delta f} \right)
= -\sqrt{\pi \Delta f \tau} \cdot \frac{1}{2} \left( 1 + \frac{2f}{\Delta f} \right),
\] (6.13)
and the upper limit will be
\[
x_2 = x|_{t=\tau/2}
= \sqrt{\frac{\mu}{2}} \left( \frac{\tau}{2} - \frac{2\pi f}{\mu} \right)
= \sqrt{\frac{\pi \Delta f}{\tau}} \cdot \tau \left( \frac{1}{2} - \frac{f}{\Delta f} \right)
= \sqrt{\pi \Delta f \tau} \cdot \frac{1}{2} \left( 1 - \frac{2f}{\Delta f} \right).
\] (6.14)
Substituting the results of (6.10)–(6.14) into (6.5) and simplifying,

\[
G(f) = A e^{-j\frac{2\pi^2 f^2}{\mu}} \int_{x_1}^{x_2} e^{i2\pi^2 f^2 \cdot \frac{2}{\pi} x^2} dx
\]

\[
= A \tau \sqrt{\frac{1}{\pi \Delta f \tau}} e^{-j\frac{2\pi^2 f^2}{\mu}} \int_{x_1}^{x_2} e^{i2\pi^2 f^2 \cdot \frac{2}{\pi} x^2} dx
\]

\[
= A \tau \sqrt{\frac{1}{\pi \Delta f \tau}} e^{-j\frac{2\pi^2 f^2}{\mu}} \int_{x_1}^{x_2} [\cos x^2 + j \sin x^2] dx .
\] (6.15)

At this point, the integrands in (6.15) can be identified as having Fresnel form. The Fresnel integrals are given by [195]

\[
C(v) = \int_{0}^{v} \cos \left(\frac{\pi}{2} \xi^2 \right) d\xi
\]

\[
S(v) = \int_{0}^{v} \sin \left(\frac{\pi}{2} \xi^2 \right) d\xi
\]

in real form, and by

\[
F(v) = C(v) + jS(v) ,
\] (6.16)

in complex form. These integrals have no closed form analytical solutions and must be evaluated numerically. They are also odd functions, so that \(C(-v) = -C(v)\) and \(S(-v) = -S(v)\).

To make the integral in (6.15) conformable for numerical evaluation, the following change of variable must be introduced:

\[
\frac{\pi}{2} \xi^2 = x^2
\]

so that

\[
\xi^2 = \frac{2}{\pi} x^2
\]

and

\[
d\xi = \sqrt{\frac{2}{\pi}} dx \Rightarrow dx = \sqrt{\frac{\pi}{2}} d\xi
\] (6.17)

The limits of integration thus become

\[
\xi_1 = -\sqrt{\frac{2}{\pi} \sqrt{\pi \Delta f \tau}} \cdot \frac{1}{2} \left(1 + \frac{2f}{\Delta f} \right)
\]

\[
= -\sqrt{\frac{\Delta f \tau}{2}} \left(1 + \frac{2f}{\Delta f} \right)
\] (6.18)
and

$$\xi_2 = \sqrt{\frac{2}{\pi}} \sqrt{\pi \Delta f \tau} \cdot \frac{1}{2} \left(1 - \frac{2f}{\Delta f}\right)$$

$$= \sqrt{\frac{\Delta f \tau}{2}} \left(1 - \frac{2f}{\Delta f}\right),$$

(6.19)

and now substituting (6.17)–(6.19) into (6.15),

$$G(f) = A\tau \sqrt{\frac{1}{\pi \Delta f \tau}} e^{-j\frac{2\pi^2 f^2}{\mu}} \int_{\xi_1}^{\xi_2} [\cos \left(\frac{\pi}{2} \xi^2\right) + j \sin \left(\frac{\pi}{2} \xi^2\right)] \sqrt{\frac{\pi}{2}} d\xi$$

$$= A\tau \sqrt{\frac{1}{\pi \Delta f \tau}} \sqrt{\frac{\pi}{2}} e^{-j\frac{2\pi^2 f^2}{\mu}} \int_{\xi_1}^{\xi_2} [\cos \left(\frac{\pi}{2} \xi^2\right) + j \sin \left(\frac{\pi}{2} \xi^2\right)] d\xi$$

$$= A\tau \sqrt{\frac{1}{\Delta f \tau}} e^{-j\frac{2\pi^2 f^2}{\mu}} \int_{\xi_1}^{\xi_2} [\cos \left(\frac{\pi}{2} \xi^2\right) + j \sin \left(\frac{\pi}{2} \xi^2\right)] \sqrt{\frac{\pi}{2}} d\xi$$

$$= A\tau \sqrt{\frac{1}{\Delta f \tau}} e^{-j\frac{2\pi^2 f^2}{\mu}} \times$$

$$\left\{ - \int_{-\xi_1}^{\xi_1} [\cos \left(\frac{\pi}{2} \xi^2\right) + j \sin \left(\frac{\pi}{2} \xi^2\right)] d\xi + \int_{0}^{\xi_2} [\cos \left(\frac{\pi}{2} \xi^2\right) + j \sin \left(\frac{\pi}{2} \xi^2\right)] d\xi \right\}$$

$$= A\tau \sqrt{\frac{1}{\Delta f \tau}} e^{-j\frac{2\pi^2 f^2}{\mu}} \left[ \left(-\frac{[C(-\xi_1) + jS(-\xi_1)]}{\sqrt{2}} + \frac{[C(\xi_2) + jS(\xi_2)]}{\sqrt{2}} \right) \right]$$

$$= A\tau \sqrt{\frac{1}{\Delta f \tau}} e^{-j\frac{2\pi^2 f^2}{\mu}} \left[ \left[\frac{C(\xi_1) + jS(\xi_1)}{\sqrt{2}} \right] + \frac{[C(\xi_2) + jS(\xi_2)]}{\sqrt{2}} \right]$$

$$= A\tau \sqrt{\frac{1}{\Delta f \tau}} e^{-j\frac{2\pi^2 f^2}{\mu}} \left[ \frac{[C(\xi_1) + C(\xi_2)]}{\sqrt{2}} + j \frac{[S(\xi_1) + S(\xi_2)]}{\sqrt{2}} \right],$$

(6.20)

where the final form in (6.20) can be numerically evaluated. Note that the frequency variable $f$ that appears in (6.20) (and (6.19)) is the complex envelope or baseband frequency, which will range from DC to $\Delta f$. Therefore, the result in (6.20) is for positive frequencies only. In order to create a two-sided baseband signal vector for $G(f)$, the non-DC spectral content must be conjugated and flipped, and prepended to the positive-frequency result in (6.20). The final frequency-domain form $S(f)$ must include modulation to $f_0$ and the phase shift $\phi_0$, so that

$$S(f) = G(f) \ast \delta \left[\cos(2\pi f_0 t + \phi_0)\right]$$

$$= G(f) \ast \frac{e^{j\phi_0} \delta(f - f_0) + e^{-j\phi_0} \delta(f + f_0)}{2}$$

$$= \frac{e^{j\phi_0} G(f - f_0) + e^{-j\phi_0} G^*(f + f_0)}{2},$$

(6.21)

where $G(f)$ is the two-sided form of the baseband signal and $G^*(f)$ is its complex conjugate.
6.2.2 Discrete Fourier Transform Scale Factors

In Section 3.2.8 on page 40, it was found that the Fourier transform of sinusoids are a pair of delta functions so that the time interval over which the transform takes place is immaterial. Such is not the case for a linear frequency modulation signal, and it is thus necessary to be cognizant of a scale factor that must be applied to the FFT output in order to make accurate comparisons of transformed time-domain results with those obtained using frequency-domain methods.

Recall from (3.43) on page 36 that the Fourier Transform of an arbitrary time-domain signal \(x(t)\) is given by

\[
X(f) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi ft} dt .
\]  

(6.22)

Now suppose that \(x(t)\) is over-sampled in the time-domain at instants \(t_m\) ranging from \(t = 0\) to an end time of \(t = T\) so that there are no aliasing effects in the sampled form and any implicit window effects are minimized \[196\]. Suppose that there are \(M\) samples total. Then the discrete form of the transform in (6.22) is given by \[197\],

\[
X(f_k) = \sum_{m=0}^{M} x(t_m) e^{-j2\pi f_k t_m} \left( \frac{T}{M} \right) ,
\]  

(6.23)

where \(t_m = mT/M\) for \(m = 0 \ldots M - 1\) and \(f_k = k/T\) for \(k = 0 \ldots M - 1\). Comparing (6.23) to (6.22), it can be seen that the quantity \(dt\) in (6.22) equates to the quantity \(T/M\) in (6.23). The FFT returned by Matlab\textsuperscript{®} is similar, but omits the \(T/M\) scale factor that appears in (6.23). In sinusoidal analysis, it is common to use a scale factor of \(1/M\) \[198\], but linear FM chirps are not sinusoidal. For accuracy in comparisons, then, it is necessary to multiply the output of Matlab\textsuperscript{®}’s FFT by \(T/M\). Both fREEDA\textsuperscript{™} and Matlab\textsuperscript{®} use the FFTW \[199\] library to perform the FFT, and both produce the same improperly scaled output. For the results reported here, the fREEDA\textsuperscript{™} time-domain will be imported into Matlab\textsuperscript{®}, transformed and scaled.

6.3 AOM and fREEDA Modeling Environment Setup

The system to be simulated in this chapter consists of a linear FM chirp source driving an X-band MMIC amplifier, with the amplifier output filtered by a 10\textsuperscript{th} order Butterworth bandpass
filter. These elements are typical of those found in a radar transmission chain, with the chain being completed by a power amplifier (which is not modeled here) following the filter. A block diagram of the system to be simulated is shown in Figure 6.1. Two modeling instances will be presented. In the first, the linear FM chirp source will be varied over a 500 MHz chirp range centered about 10 MHz and the bandpass filter for this case will have a 600 MHz passband. In the second instance, the linear FM chirp source will be varied over a 1 GHz chirp range centered about 10 MHz and the bandpass filter for this case will have a 1.2 GHz passband.

6.3.1 fREEDA Circuit Models

The Filtronics LMA-411 MMIC fREEDA™ circuit model used in this work was derived from previous model extraction work done at another university [200], and the Linear FM chirp source and bandpass filter models were written by the dissertation author. The Butterworth bandpass filter models are implemented in the $z$-domain using the methodology disclosed in detail in Appendix A. The passband, its flatness, and the number of poles in the filter are specified as netlist parameters for the filter model. The amplitude of the chirp, center frequency of operation, and chirp range are specified as netlist parameters for the linear FM chirp source.

6.3.2 AOM Behavioral Models

The linear FM chirp source used in the AOM Toolbox modeling consisted of 57 selected data points chosen from a 401-point vector of data created from the native frequency-domain description of the signal as given by (6.20)–(6.21) on page 219, and the Butterworth bandpass filter models were implemented using a function written for the AOM Toolbox which takes the passband, its flatness, and the number of poles as function parameters. The Filtronics LMA-411 MMIC AOM Toolbox model used in this work is an odd-order complex power series behavioral model.
model with a linear gain of 18 dB that was derived from swept power AM/AM and AM/PM characteristics as reported in [20], where the authors reported a model with odd-order coefficients up to the 9th order. For the work here, the model was simplified to the 3rd order. The power series coefficients are given in Table 6.1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Real Part</th>
<th>Imaginary Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Gain $a_1$</td>
<td>-1.07003</td>
<td>7.80747</td>
</tr>
<tr>
<td>3rd Order Gain $a_3 (V^{-2})$</td>
<td>2.28416</td>
<td>-2.16753</td>
</tr>
</tbody>
</table>

### 6.3.3 Butterworth Bandpass Filter Model

Two bandpass filters were implemented in the fREEDA™ and AOM Toolbox toolbox simulations here. Both filters were 10th order Butterworth bandpass filters with center frequencies of 10 GHz. For the simulation of the 500 MHz chirp, the filter was implemented with a passband of 600 MHz and a passband flatness specification of 1 dB. For the simulation of the 1 GHz chirp, the filter was implemented with a passband of 1.2 GHz and a passband flatness specification of 1 dB. Figure 6.2 shows the magnitude response (in dB) of the two filters, and Figure 6.3 shows the phase angles introduced by the filters. The phase is plotted in unwrapped form with the 0 degree point set at the 10 GHz center frequency.
Figure 6.2: Attenuation (in dB) of the bandpass filters used in radar signal chain modeling. The filter with the 600 MHz passband is used to filter the response of the MMIC to the 500 MHz chirp source, while the filter with the 1.2 GHz passband is used to filter the response of the MMIC to the 1 GHz chirp source.

Figure 6.3: Phase change (in degrees) introduced by the 600 MHz and 1.2 GHz bandpass filters.
6.4 Results and Discussion

The results of simulating responses to the 500 MHz and 1 GHz chirps will be presented in separate subsections. The chirp input $X(f)$ (as shown in Figure 6.1 on page 221) will be shown first in plotted form, followed by views of the MMIC output $W(f)$ and the bandpass filter output $Y(f)$ (as shown in Figure 6.1 on page 221).

6.4.1 500 MHz Chirp Signal and Response

Figure 6.4 shows the magnitude spectrum of a 500 MHz linear FM chirp signal centered at 10 GHz. In the figure, a 401-point plot of the magnitude spectrum of the analytical Fourier transform is shown as a solid line, and the diamond marks show the 57 points selected used to form the magnitude of the input signal to the AOM Toolbox. Note that the 57 selected points covers all of the chirp output and the top portion of the chirp signal’s skirts. The scaled FFT of the $fREEDA™$ transient simulation output for $X(f)$ is also shown in Figure 6.4.

Figure 6.4: Magnitude of the 500 MHz linear FM chirp input. The discrete points used by the AOM Toolbox are illustrated along with the analytical Fourier transform and the FFT of the a linear FM chirp source in $fREEDA™$. 
Figure 6.5: Magnitude detail of the peak of the 500 MHz LFM chirp input. The details in the peak of Figure 6.4 are shown.

Figure 6.6: Phase of the 500 MHz linear FM chirp input. The smooth quadratic behavior predicted by (6.20) can be seen.
A detail view of just the peak portion of the chirp input is given in Figure 6.5 on the previous page. The fREEDA™ transient simulation used a time step of 5 picoseconds over a record of 131,072 points. The width of the chirp pulse was 380 nsec. Note that the greater density of points in the fREEDA™ source allows a greater resolution in capturing the oscillatory nature of the chirp signal source at frequencies well away from the chirp range. The phase of the chirp source is shown in Figure 6.6 on the preceding page and it shows the quadratic behavior indicated by the complex exponential factor appearing (before the Fresnel integral) in (6.20) on page 219. The phase of the fREEDA™ source is not shown here since time delays were introduced into in the fREEDA™ records.

6.4.1.1 MMIC Amplifier Response to a 500 MHz Chirp

In this subsection the simulated outputs in the vicinity of the chirp frequencies will be compared to each other and to the response predicted by adding the linear gain of the MMIC to a 401-point vector representation of the analytical Fourier transform of the chirp input signal. Figure 6.7 shows the predicted output magnitude of the MMIC \( W(f) \) in Figure 6.1 on page 221 in the vicinity of the chirp obtained by adding the 18 dB linear gain to the 401-point analytical Fourier transform of the linear FM chirp input. The simulated results produced by the AOM Toolbox (diamond marks) and the FFT of fREEDA™ transient simulation output are also shown. The results shown are the total output for the AOM Toolbox at the input frequencies. Figure 6.8 shows a detail view of Figure 6.7 in the vicinity of the chirp peak. There is excellent agreement — to within 1 dB here — between the analytically predicted results and the two simulations. However, close inspection of Figure 6.8 reveals that the response of the MMIC in the FFT of the fREEDA™ transient output has a slight upslope with increasing frequency. This upslope appears to be an inherent feature of the transient response of the MMIC as it can be seen also in Figure A.17 of Appendix A on page 322 which compares the measured response of the LMA-411 MMIC to the FFT of fREEDA™ simulation results to a narrow-bandwidth linear FM chirp signal.

Due to the presence of an initial output transient from the fREEDA™ model of the MMIC (because of capacitive coupling), the fREEDA™ simulation was run for an additional 10 nsec and the initial 10 nsec of output discarded so that the data records from fREEDA™ consisted of 131,072 points.

An alternate view of the MMIC output is given in Figure 6.9 on page 229 where the
Figure 6.7: Magnitude of the MMIC output $W(f)$ in the chirp range. The AOM Toolbox output in the chirp range is compared to the FFT of the fREEDA™ transient simulation output in the chirp range. The Fourier transform of the analytical input is scaled by the linear gain is also shown for reference.

FFT of the fREEDA™ output for the MMIC can be seen from DC up to the FFT’s Nyquist Frequency of 100 GHz. Note the presence of visible spectral content at the 2nd–5th harmonics of 10 GHz along with some additional content at the 7th harmonic. Since the fREEDA™ output data may be viewed as sampled data, it is important to consider the possible effects of aliasing on the data. Figure 6.10 on page 230 shows a detail view of the 7th harmonic output of Figure 6.9 along with a frequency-expanded, magnitude-adjusted version of the linear output of Figure 6.9. When aliasing is present in sampled data, the effect is to fold spectral content at frequencies beyond the Nyquist Frequency back onto the spectrum such that the validity of upper portions of the spectral content approaching the Nyquist frequency should be considered suspect. However, the qualitative similarity of the 7th harmonic content (particularly the oscillations at 7 times that of the adjusted linear overlay) to the adjusted linear output suggests that aliasing is not present in the FFT of the fREEDA™ data at least through the 7th harmonic. In contrast, the complete absence of anything appearing to be 8th or 9th harmonic content suggests that there may be some aliasing at frequencies beyond that of the 7th harmonic content. Since the AOM Toolbox model was confined to odd-order modeling and truncated at...
Figure 6.8: Magnitude detail of the MMIC chirp peak output of $W(f)$. The AOM Toolbox output in the vicinity of the chirp peak is compared to the FFT of the fREEDA™ transient simulation output in the vicinity of the chirp peak. The Fourier transform of the analytical input is scaled by the linear gain is also shown for reference.

the 3rd order, most of the spectral content visible in the fREEDA™ output is not modeled in the AOM Toolbox results, but the (appropriately scaled) fREEDA™ data that is compared to AOM Toolbox output may be considered free of aliasing.

In addition to the linear output, the 3rd order MMIC output is modeled using the AOM Toolbox, but direct comparison with the FFT of the fREEDA™ 3rd output requires some additional scaling of the fREEDA™ data. Recalling that the LFM chirp input is a single pulse while the use of the FFT on such data assumes it is periodically extended, then the scale factor $dt = T/M$ that appears in (6.23) on page 220 to the LFM chirp will properly scale the input signal to give a correct linear output. To compare 3rd order output, however, it is necessary to apply a scale factor of $T^3/M$ to the FFT of the fREEDA™ 3rd order harmonic data in order to make an accurate comparison with the results produced by the AOM Toolbox. This means that the 3rd order harmonic data appearing in Figure 6.9 must be arithmetically scaled by a factor of $20 \log_{10} T^2$ in order to accurately reflect a 3rd order response. Figure 6.11 on page 230 compares the AOM Toolbox 3rd order harmonic response to the FFT of the fREEDA™ 3rd order
harmonic response when this additional scale factor is applied to the fREEDA™ FFT data. There is about 10 dB of remaining discrepancy in the fREEDA™ and AOM Toolbox results. A possible explanation for this is that the in-band distortion model extraction technique used in [201] was not adequate to model the harmonic distortion of the MMIC, and that further accuracy improvements in the models suggests the use of parallel-connected modeling blocks. One other contributing factor may be a divergence — even in the absence of aliasing — between the discrete Fourier transform of a non-periodic function and its true transform. In [203], it is suggested that only the first 1/8 of points in a DFT should be considered highly accurate when the transformed signal is not periodic. Thus for the modeling problem here consisting of a sampling frequency of 200 GHz, then, only spectral content below 25 GHz can be considered to be highly accurate, so discrepancies in the vicinity of 3rd order harmonic centered at 30 GHz must be expected.
Figure 6.10: Detailed view of the 7th harmonic of the fREEDA™ MMIC $W(f)$ output as computed by the FFT of the fREEDA™ transient simulation output along with a frequency-expanded, magnitude-adjusted view of the linear output for qualitative spectral comparison.

Figure 6.11: Detailed view comparing the (arithmetically scaled, by a factor of $20 \log_1 0T^2$ 3rd harmonic fREEDA™ MMIC $W(f)$ output (as computed by the FFT of the fREEDA™ transient simulation output) to the 3rd harmonic output computed by the AOM Toolbox.
6.4.1.2 Bandpass Filtering the 500 MHz Chirp Response

In this subsection, the bandpass filter output \( Y(f) \) of the 3\(^{rd} \) order intermodulation in the bands adjacent to the chirp frequency range will be examined along with the response in the vicinity of the 3\(^{rd} \) order harmonic. In particular, the responses \( W(f) \) and \( Y(f) \) (see Figure 6.1 on page 221) produced by the AOM Toolbox will be compared to observe the effects of the bandpass filtering, and the \( f\text{REEDA}^\text{™} \) output of the bandpass filter will be presented.

Figure 6.12 shows the MMIC output \( W(f) \) and the bandpass filter output \( Y(f) \) produced by the AOM Toolbox in the bands adjacent to the linear output. Due to the very low power level of the linear output, the adjacent band IM distortion occurs around 300 dB down from the linear output in the AOM Toolbox. More important than the absolute power levels, however, is the observation that the bandpass filter is working exactly as expected — it matches the MMIC output in a range of about 600 MHz centered about 10 GHz, but attenuates the MMIC output rapidly outside the specified passband. Figure 6.13 on the next page shows the 3\(^{rd} \) harmonic MMIC output \( W(f) \) and the 3\(^{rd} \) harmonic bandpass filter output \( Y(f) \) produced by the AOM Toolbox around 30 GHz. Again, the bandpass filter drastically attenuates any MMIC output that is present, as expected. Figure 6.14 on the following page shows the

![Figure 6.12: Magnitude of the adjacent band outputs.](image)
Figure 6.13: Magnitude of the 3rd harmonic outputs.

Figure 6.14: Magnitude of the whole-spectrum fREEDA™ output $Y(f)$. 
$f\text{REEDA}^{TM}$ output $Y(f)$ over the whole spectrum. Compared to Figure 6.9 on page 229 note the dramatic attenuation of z-domain bandpass filter on the spectrum outside the passband. Finally, note that it is not possible to compare the adjacent-band 3rd order IM distortion of the AOM Toolbox and $f\text{REEDA}^{TM}$ results because the linear response of the chirp in the adjacent bands of the $f\text{REEDA}^{TM}$ output in Figure 6.9 on page 229 occurs at much higher levels than the 3rd order adjacent-band IM distortion in Figure 6.12 on page 231. This is to be expected since the FFT of the $f\text{REEDA}^{TM}$ response is of the total transient response.

6.4.2 1 GHz Chirp Signal and Response

Figure 6.15 on the following page shows the magnitude spectrum of a 1 GHz linear FM chirp signal centered at 10 GHz. In the figure, a 401-point plot of the magnitude spectrum of the analytical Fourier transform is shown as a solid line, and the diamond marks show the 57 points selected used to form the magnitude of the input signal to the AOM Toolbox. Note that the 57 selected points covers all of the chirp output and the top portion of the chirp signal’s skirts. The scaled FFT of the $f\text{REEDA}^{TM}$ transient simulation output for $X(f)$ is also shown in Figure 6.15. A detail view of just the peak portion of the chirp input is given in Figure 6.16 on page 235. The $f\text{REEDA}^{TM}$ transient simulation used a time step of 5 picoseconds over a record of 131,072 points. The width of the chirp pulse was 380 nsec. Note that the greater density of points in the $f\text{REEDA}^{TM}$ source allows a greater resolution in capturing the oscillatory nature of the chirp signal source at frequencies well away from the chirp range. The phase of the chirp source is shown in Figure 6.17 on page 235 and it shows the quadratic behavior indicated by the complex exponential factor appearing (before the Fresnel integral) in (6.20) on page 219. The phase of the $f\text{REEDA}^{TM}$ source is not shown here since time delays were introduced into the $f\text{REEDA}^{TM}$ records.

6.4.2.1 MMIC Amplifier Response to a 1 GHz Chirp

In this subsection the simulated outputs in the vicinity of the chirp frequencies will be compared to each other and to the response predicted by adding the linear gain of the MMIC to a 401-point vector representation of the analytical Fourier transform of the chirp input signal. Figure 6.18 on page 236 shows the predicted output magnitude of the MMIC ($W(f)$ in Figure 6.1 on page 221) in the vicinity of the chirp obtained by adding the 18 dB linear gain to the 401-point
analytical Fourier transform of the linear FM chirp input. The simulated results produced by the AOM Toolbox (diamond marks) and the FFT of $fREEDA^\text{™}$ transient simulation output are also shown. The results shown are the total output for the AOM Toolbox at the input frequencies. Figure 6.19 on page 237 shows a detail view of Figure 6.18 in the vicinity of the chirp peak. There is excellent agreement here between the analytically predicted results and the two simulations. There is excellent agreement — to within 1 dB here — between the analytically predicted results and the two simulations. However, close inspection of Figure 6.19 reveals that the response of the MMIC in the FFT of the $fREEDA^\text{™}$ transient output has a slight upslope with increasing frequency. This upslope appears to be an inherent feature of the transient response of the MMIC as it can be seen also in Figure A.17 on page 322 of Appendix A on page 322 which compares the measured response of the LMA-411 MMIC to the FFT of $fREEDA^\text{™}$ simulation results to a narrow-bandwidth linear FM chirp signal.
Figure 6.16: Magnitude detail of the peak of the 1 GHz LFM chirp input. The details in the peak of Figure 6.15 are shown.

Figure 6.17: Phase of the 1 GHz linear FM chirp input. The smooth quadratic behavior predicted by (6.20) in the region of the chirp peak can be seen.
Figure 6.18: Magnitude of the MMIC output $W(f)$ in the chirp range. The AOM Toolbox output in the chirp range is compared to the FFT of the $f$REEDA™ transient simulation output in the chirp range. The Fourier transform of the analytical input is scaled by the linear gain is also shown for reference.
Figure 6.19: Magnitude detail of the MMIC chirp peak output of $W(f)$. The AOM Toolbox output in the vicinity of the chirp peak is compared to the FFT of the fREEDA™ transient simulation output in the vicinity of the chirp peak. The Fourier transform of the analytical input is scaled by the linear gain is also shown for reference.
Due to the presence of an initial output transient from the \textit{fREEDA}™ model of the MMIC (because of capacitive coupling), the \textit{fREEDA}™ simulation was run for an additional 10 nsec and the initial 10 nsec of output discarded so that the data records from \textit{fREEDA}™ consisted of 131,072 points.

An alternate view of the MMIC output is given in Figure 6.20 on the following page where the FFT of the \textit{fREEDA}™ output for the MMIC can be seen from DC up to the FFT’s Nyquist Frequency of 100 GHz. Note the presence of visible spectral content at the 2\textsuperscript{nd}–5\textsuperscript{th} harmonics of 10 GHz along with some additional content at the 7\textsuperscript{th} harmonic. Since the \textit{fREEDA}™ output data may be viewed as sampled data, it is important to consider the possible effects of aliasing on the data. Figure 6.21 on the next page shows a detail view of the 7\textsuperscript{th} harmonic output of Figure 6.20 on the following page along with a frequency-expanded, magnitude-adjusted version of the linear output of Figure 6.20. When aliasing is present in sampled data, the effect is to fold spectral content at frequencies beyond the Nyquist Frequency back onto the spectrum such that the validity of upper portions of the spectral content approaching the Nyquist frequency should be considered suspect. However, the qualitative similarity of the 7\textsuperscript{th} harmonic content (particularly the oscillations at 7 times that of the adjusted linear overlay) to the adjusted linear output suggests that aliasing is not present in the FFT of the \textit{fREEDA}™ data at least through the 7\textsuperscript{th} harmonic. In contrast, the complete absence of anything appearing to be 8\textsuperscript{th} or 9\textsuperscript{th} harmonic content suggests that there may be some aliasing at frequencies beyond that of the 7\textsuperscript{th} harmonic content. Since the AOM Toolbox model was confined to odd-order modeling and truncated at the 3\textsuperscript{rd} order, most of the spectral content visible in the \textit{fREEDA}™ output is not modeled in the AOM Toolbox results, but the (appropriately scaled) \textit{fREEDA}™ data that is compared to AOM Toolbox output may be considered free of aliasing.
Figure 6.20: Magnitude of the whole-spectrum MMIC $W(f)$ output as computed by the FFT of the fREEDA™ transient simulation output.

Figure 6.21: Detailed view of the 7th harmonic of the fREEDA™ MMIC $W(f)$ output as computed by the FFT of the fREEDA™ transient simulation output along with a frequency-expanded, magnitude-adjusted view of the linear output for qualitative spectral comparison.
In addition to the linear output, the 3rd order MMIC output is modeled using the AOM Toolbox, but direct comparison with the FFT of the fREEDA™ 3rd output requires some additional scaling of the fREEDA™ data. Recalling that the LFM chirp input is a single pulse while the use of the FFT on such data assumes it is periodically extended, then the scale factor \( dt = T/M \) that appears in (6.24) on page 220 to the LFM chirp will properly scale the input signal to give a correct linear output. To compare 3rd order output, however, it is necessary to apply a scale factor of \( T_3^3/M \) to the FFT of the fREEDA™ 3rd order harmonic data in order to make an accurate comparison with the results produced by the AOM Toolbox. This means that the 3rd order harmonic data appearing in Figure 6.20 must be arithmetically scaled by a factor of \( 20 \log_{10} T^2 \) in order to accurately reflect a 3rd order response. Figure 6.22 compares the AOM Toolbox 3rd order harmonic response to the FFT of the fREEDA™ 3rd order harmonic response when this additional scale factor is applied to the fREEDA™ FFT data. There is about 10 dB of remaining discrepancy in the fREEDA™ and AOM Toolbox results. A possible explanation for this is that the in-band distortion model extraction technique used in [201] was not adequate to model the harmonic distortion of the MMIC, and that further accuracy improvements in the models suggests the use of parallel-connected modeling blocks. One other contributing factor may be a divergence — even in the absence of aliasing — between the discrete Fourier transform of a non-periodic function and its true transform. In [203], it is suggested that only the first 1/8 of points in a DFT should be considered highly accurate when the transformed signal is not periodic. Thus for the modeling problem here consisting of a sampling frequency of 200 GHz, then, only spectral content below 25 GHz can be considered to be highly accurate, so discrepancies in the vicinity of 3rd order harmonic centered at 30 GHz must be expected.

### 6.4.2.2 Bandpass Filtering the 1 GHz Chirp Response

In this subsection, the bandpass filter output \( Y(f) \) of the 3rd order intermodulation in the bands adjacent to the chirp frequency range will be examined along with the response in the vicinity of the 3rd order harmonic. In particular, the responses \( W(f) \) and \( Y(f) \) (see Figure 6.1 on page 221) produced by the AOM Toolbox will be compared to observe the effects of the bandpass filtering, and the fREEDA™ output of the bandpass filter will be presented.

Figure 6.23 on the next page shows the MMIC output \( W(f) \) and the bandpass filter output \( Y(f) \) produced by the AOM Toolbox in the bands adjacent to the linear output. Due to the very low power level of the linear output, the adjacent band IM distortion occurs around 300
Figure 6.22: Detailed view comparing the (arithmetically scaled, by a factor of $20\log_{10} T^2$) 3rd harmonic $f\text{REEDA}^+\text{MMIC} W(f)$ output (as computed by the FFT of the $f\text{REEDA}^+$ transient simulation output) to the 3rd harmonic output computed by the AOM Toolbox.

Figure 6.23: Magnitude of the adjacent band outputs.
dB down from the linear output in the AOM Toolbox. More important than the absolute power levels, however, is the observation that the bandpass filter is working exactly as expected — it matches the MMIC output in a range of about 600 MHz centered about 10 GHz, but attenuates the MMIC output rapidly outside the specified passband. Figure 6.24 shows the 3rd harmonic MMIC output \( W(f) \) and the 3rd harmonic bandpass filter output \( Y(f) \) produced by the AOM Toolbox around 30 GHz. Again, the bandpass filter drastically attenuates any MMIC output that is present, as expected. Figure 6.25 on the next page shows the \( f\text{REEDA}™ \) output \( Y(f) \) over the whole spectrum. Compared to Figure 6.20 on page 239, note the dramatic attenuation of z-domain bandpass filter on the spectrum outside the passband. Finally, note that it is not possible to compare the adjacent-band 3rd order IM distortion of the AOM Toolbox and \( f\text{REEDA}™ \) results because the linear response of the chirp in the adjacent bands of the \( f\text{REEDA}™ \) output in Figure 6.20 on page 239 is at much higher levels than the 3rd order adjacent-band IM distortion in Figure 6.23 on the preceding page. This is to be expected since the FFT of the \( f\text{REEDA}™ \) response is of the total transient response.
6.4.3 Computational Considerations

Since both the 500 MHz chirp and the 1 GHz chirp instances used 57 points of spectral data for the number of tones in the AOM input and both cases were limited to a 3\textsuperscript{rd} order nonlinear transfer function, times of computation were modest. Both simulations completed in just over 11 minutes on an AMD Opteron platform with 16 GBytes of DRAM running a 64-bit version of Matlab\textsuperscript{®} under a Linux operating system. Times of computation for the AOM Toolbox toolbox functions were equal within 6 seconds, with the VFD Table requiring 52 seconds, the Spectrum Mapping Table taking 327 seconds, the Spectrum Transform Table taking 297 seconds, and the sparse block evaluation competing in 0.1 seconds. The total time of computation was slightly more than 11 minutes.

6.5 Concluding Remarks

The AOM Toolbox and the FFT of fREEDA™ simulated responses of a nonlinear X-band MMIC device and a cascaded bandpass filter to two different linear FM chirp sources were found to be in excellent agreement with analytical predictions in the band of the linear response.
Furthermore, the AOM Toolbox bandpass filters showed a desired sharp response in the adjacent band according to a stringent passband flatness specification, and the \textit{fREEDA}™ passband filters showed drastic levels of attenuation — almost down to the noise floor of the FFT (for the case of 131,072 points) used to produce the frequency-domain magnitude spectrum. The 3\textsuperscript{rd} order responses of the AOM Toolbox MMIC model were found to slightly diverge from the \textit{fREEDA}™ results in the 3\textsuperscript{rd} harmonic portion of the frequency spectrum, and this is probably attributable to a deficiency in the ability of the extracted in-band power series model to accurate model harmonic distortions along with a possible expected discrepancy in the accuracy of the FFT data which develops after the first 1/8 fraction of the FFT points. The results here also demonstrate some inherent limits into the insight possible when considering the FFT of transient simulation data. Since transient simulation results of the sort produced by \texttt{Spice} engines are total responses including all orders of nonlinearity, when the FFT is taken, the linear response to signals such as Linear FM chirps tends to overwhelm any 3\textsuperscript{rd} order adjacent-band IM that might occur. However, since the AOM Toolbox separates the linear response and the responses at all orders of nonlinearity into unique output vectors, it is possible when using the AOM Toolbox to inspect the adjacent-band IM response from odd-order nonlinearities separately from the linear output. This capability could be exploited, for example, by permitting radar system designers to behaviorally model the response of driving radar transmitters to the edge of nonlinearity with linear FM chirp signals in a fashion analogous to that of the 3\textsuperscript{rd} order intercept point in two-tone analysis.

Beyond the additional insight available when the AOM Toolbox is used for radar transmission system modeling, the \textit{z}-domain filters used to model the Butterworth filters in \textit{fREEDA}™ are also a new contribution to knowledge in that they are synthesized from analog (not digital) frequency specifications passed as netlist parameters. Internal to \textit{fREEDA}™, the \textit{z}-domain filter models are synthesized using the time step specified in the netlist. By comparison, a leading commercial simulator synthesizes digital filters outside of the transient simulation environment and takes in the filter coefficients as netlist parameters; consequently whenever the time-step is changed, the filter must be synthesized again and the new filter coefficients passed in the netlist. Further details on the \textit{z}-domain filtering implementation in \textit{fREEDA}™ can be found in Appendix A on page 294.
Chapter 7

Modeling the Multicarrier Response of a CATV Trunk Amplifier

7.1 Introductory Remarks

This chapter will present the results of a large-scale behavioral modeling problem — that of modeling the response of an amplifier typically used in cable television (CATV) distribution networks. Amplifiers used in CATV signal propagation chains are used to amplify modulated carriers for 10–20 channels in minimal subscription situations, up to 50–60 channels for typical subscriptions, and over 100 channels in subscriptions with full channel lineups. Looking into the future, the trends are for greater numbers of channels, and so a computer-aided analysis tool that can model the response of such amplifiers to hundreds of carriers will become a necessity. Presently, there are no tools capable of this degree of large-scale modeling, but this chapter demonstrates how the AOM Toolbox may be able to fill that demand.

The work here begins with the data for extracting a behavioral model first proposed by Naishadham [204], but then separates the frequency dependence at each order of nonlinearity from a fixed gain value so that a Parallel Hammerstein model may be used to demonstrate the ability of the AOM Toolbox to model frequency-dependent memory effects. The frequency-dependent effects will be modeled by lowpass Butterworth filter models with characteristics selected from amplifiers which are also used in CATV applications. The model thus developed
here will have no real-world analogue, but since the emphasis here is on demonstrating the modeling potential of the AOM Toolbox for use in large-scale multicarrier problems, it will suffice.

### 7.2 The CATV Frequency Plan

For the purposes of this work, the frequency plan adopted is that noted by Naishadham \[204\], which was termed an Incremental Frequency Plan (IRC). The analog channels in this plan have carrier frequencies \( f_q \) given by

\[
 f_q = \begin{cases} 
 55.25 + 6(q - 1), & \text{for } q = 1, 2, 3 \\
 77.25 + 6(q - 4), & \text{for } q = 4, 5 \\
 109.25 + 6(q - 6), & \text{for } q = 6, 7, \ldots ,
\end{cases} \tag{7.1}
\]

where \( f_q \) is in MHz. In \[204\], the plan terminates at \( q = 79 \); here it will be extended. The plan used here is noted because it differs from that given in \[205\], which summarizes the channel allocation plan for North America. In \[205\], the IRC CATV channel plan has carriers located at 12,500 kHz offsets to that given in (7.1). The plan in (7.1) has carriers located at essentially the same nominal frequencies as the standard NTSC broadcast channel allocation plan for CATV \[206\], where up to 131 channels are defined. In \[205\] the channel plans show the channel with highest frequency in the range of 996–1002 MHz. Dividing 1002 by 6 and excluding the band below 54 MHz, this allows for a target of 158 channels. (Unfortunately, due to the channels in (7.1) for \( q = 4, 5 \) being offset from the uniform 6 MHz increments of the other channels, using 158 channels as the target results in a maximum carrier frequency at 1021.25 MHz. Due to long simulation times, this oversight will not be corrected and the 158 channel plan noted here will stand for the purposes of reporting results. For the record, though, 154 channels would be the correct number to have a maximum carrier frequency of 997.25, which is where the carrier would be situated in a 6 MHz channel spanning 996–1002 MHz.) Note that the channel allocation plan in (7.1) could not be used for over-air broadcasting as the channel assignments beginning at 109.25 MHz (for \( q = 6, 7, \ldots \)) in (7.1) would interfere with the FCC allocations for aeronautical navigation \[206\].
7.3 Forms of Distortion Considered

There are several forms of distortion phenomena caused by unwanted nonlinear behavior of amplifiers in CATV broadcasting, but the main focus here will be on two types known to limit performance: Composite Second Order (CSO) and Composite Triple Beat (CTB). CSO distortion is second-order intermodulation distortion occurring in the vicinity of a carrier frequency and CTB is third-order intermodulation distortion of carriers which occurs at a carrier frequency. The CSO distortion additionally is bifurcated two ways, with higher-power broadband distortion products occurring at ±1.25 MHz accompanied by lower-power broadband distortion products occurring at ±0.75 MHz (but with one interesting spike in power levels, as will be seen in the results). CSO and CTB are the focus of attention because their power levels are magnified due to a plurality of instances or beats of carrier intermodulation which all occur at the same frequency.

There are several other forms of distortion that occur in CATV trunk amplifiers. Harmonic distortions are caused by the second-order and third-order self- mixing of carriers. Because only one instance of these occur at each of the harmonic frequencies, their power levels are very low relative to the CSO and CTB distortion. Harmonic distortion results will be used to illustrate the effects of order-dependent filtering in a separate subsection of the results. Finally, a form of 3rd order intermodulation distortion involving the mixture of two additive instances of one carrier and one subtractive instance of a second carrier also occurs at power levels well below CTB, so it will also not be considered.

7.4 Forms of Distortion Not Considered

Cross modulation distortion (XMOD) is caused by the interaction of a carrier frequency with the horizontal frequency control signals of another channel [207]. The horizontal frequency control signals are embedded in the baseband content of each television signal that is modulated by the carrier frequency. Broadband XMOD modeling is presently beyond the capability of the AOM Toolbox to handle, although it may be possible to investigate XMOD for a small number of channels of modulated spectral content.
7.5 Nonlinear Amplifier Model

The model used in this work is shown in Figure 7.1, where the asterisk-like operators preceding the gain blocks denote convolution. The behavioral topology is classified as the Parallel Hammerstein variety [208, 209], where frequency-independent gain blocks for each order of nonlinearity are followed by filter blocks to account for order-dependent memory effects. It was chosen for this work because of its similarity to a Volterra model that was successfully used in previous CATV amplifier modeling efforts [210, 211]. One other report in the literature suggests that better fits of measured CATV data to models are found when filter blocks follow the memoryless nonlinearities [212]. Here the model is an amalgam of parameters from several locations. The gain parameters $G_m$ are derived from the model given in [204], and the filters $K_m$ are Butterworth filters with specifications following the passband specifications from real CATV amplifiers [213-215]. The transfer function for the amplifier model is given by

$$Y(f) = Y_1(f) + Y_2(f) + Y_3(f) = K_1(f)W_1(f) + K_2(f)W_2(f) + K_3(f)W_3(f) = K_1(f)G_1X(f) + K_2(f)G_2X(f) \ast X(f) + K_3(f)G_3X(f) \ast X(f) \ast X(f),$$

where $\ast$ denotes convolution, the $W_m(f)$ are the memoryless outputs before the order-dependent memories implemented by the filters $K_m(f)$, and the $Y_m(f)$ are the order-dependent outputs including memory effects. In the results shown later, the CSO and CTB distortion will be shown at the $W_m(f)$ levels, so as not to mix memory effects with distortion created by the
memoryless nonlinearities. The memory effects of the filters will be illustrated separately with respect to the linear and harmonic distortion outputs.

### 7.5.1 Input Signal

The carriers in the input signal for the 79-channel and 158-channel plans have uniform levels set to approximately $V \approx 15.85$ mV in amplitude. Within the CATV industry, input levels are typically specified in the units of decibels referred to 1 mV — or dBmV — where

$$V_{dBmV} (dBmV) = 20 \log_{10} \left( \frac{V}{0.001} \right). \tag{7.5}$$

Equation (7.5) gives $V_{dBmV} = 24$ dBmV for $V = 0.01585$ V.

### 7.5.2 Gain Parameters

The gain parameters $G_1$, $G_2$, and $G_3$ in Figure 7.1 on the previous page were extracted from the model reported in [204]. The linear power gain reported was 20 dB, so $G_1 = 10$. $G_2$ and $G_3$ were extracted by averaging the scaled antilogarithm of the reported Nonlinearity Distortion Parameters $NLD2(f)$ and $NLD3(f)$ shown in Figure 7.2. Figures 7.3 on page 251 and 7.4 on page 251 show the antilogarithm of the quantities in Figure 7.2. The parameters $D_2(f)$ and $D_3(f)$ are related to measured CSO and CTB levels in [216] and are related to the gain coefficients as follows:

$$D_2(f) = \frac{G_2(f) A_c}{G_1}, \tag{7.6}$$

$$D_3(f) = \frac{3G_3(f) A_c^2}{2G_1}, \tag{7.7}$$

where $A_c$ is the carrier amplitude and is assumed to be the same for all carriers. $D_2(f)$ and $D_3(f)$ have a mild upslope in their frequency dependency, but the variation from minimum to maximum values has only about a 7 dB effect. Therefore, for the purposes of simplifying the amplifier model here, $G_2$ and $G_3$ will be set to constant values by using the average values of $D_2(f)$ and $D_3(f)$, $\bar{D}_2(f)$ and $\bar{D}_3(f)$, respectively, so that

$$G_2 = \frac{G_1 \bar{D}_2(f)}{A_c}, \tag{7.8}$$

$$G_3 = \frac{2G_1 \bar{D}_3(f)}{3A_c^2}. \tag{7.9}$$
Under ideal circumstances, these parameters are frequency-independent, but in practice they exhibit mild frequency dependency. Table 7.1 summarizes the extracted gain parameters used in this work.

Table 7.1: Gain Parameters for the gain blocks in Figure 7.2

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Gain $G_1$</td>
<td>10</td>
</tr>
<tr>
<td>$2^{nd}$ Order Gain $G_2$</td>
<td>$V^{-1}$</td>
</tr>
<tr>
<td>$3^{rd}$ Order Gain $G_3$</td>
<td>$V^{-2}$</td>
</tr>
</tbody>
</table>

Figure 7.2: Nonlinearity Distortion Parameters used to derive $G_2$ and $G_3$. 

\[
\begin{align*}
\text{NLD2}(f) &= -20\log_{10}[D2(f)] \\
\text{NLD3}(f) &= -20\log_{10}[D3(f)]
\end{align*}
\]
Figure 7.3: Nonlinearity Distortion Parameter $D_2(f)$.

Figure 7.4: Nonlinearity Distortion Parameter $D_3(f)$.
7.5.3 Filter Parameters

The filters $K_1(f)$, $K_2(f)$, and $K_3(f)$ in Figure 7.1 were implemented as two-pole lowpass Butterworth filters using band edge specifications given by three commercial CATV amplifier products. Table 7.2 shows the parameters for the filters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>$K_1(f)$</th>
<th>$K_2(f)$</th>
<th>$K_3(f)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Freescale</td>
<td>RFHIC 2F1G18P</td>
<td>RFHIC 2F1G20P</td>
<td>MWH8182CN</td>
</tr>
<tr>
<td>Edge Frequency (MHz)</td>
<td>1000</td>
<td>1000</td>
<td>860</td>
</tr>
<tr>
<td>Edge Attenuation (dB)</td>
<td>-0.6</td>
<td>-0.8</td>
<td>-0.8</td>
</tr>
</tbody>
</table>

Figure 7.5 shows the magnitude response and Figure 7.6 shows the phase response of filters $K_1(f)$, $K_2(f)$, and $K_3(f)$. For a two-pole filter, the magnitude response should roll off
asymptotically at about 12 dB per octave. The data in Figure 7.6 is plotted only from DC to 2 GHz so that the filter curves can be distinguished, but the magnitude data was generated out to 3 GHz, and inspection of the data for filter \( K_1 \) showed a roll off of 11 dB from 1.5 to 3 GHz, so the filters are synthesized properly in the AOM Toolbox.

7.6 Results

The results of simulating the CATV amplifier model will be presented in three sections. The first and second sections will show the CSO and CTB distortions created before filtering for the 79 and 158 channel plans, respectively. The final section will show the response of the filters to the linear output and the 2nd and 3rd order harmonic distortions.

7.6.1 79 Channel Plan CSO/CTB Results

The results in this section are distortion products occurring within the \( W_m(f) \) outputs in Figure 7.1 on page 248 for a channel plan with 79 carriers as input. CSO results that occur in \( W_2(f) \) are given first, followed by a section on the CTB results that occur in \( W_3(f) \). Note that with the inputs set to a level of 24 dBmV, the linear output for all channels was flat at 44
dBmV. The linear output is not shown here.

### 7.6.1.1 79 Channel CSO Results

Figure 7.7 shows the output power spectral density of the Composite Second Order outputs occurring at frequencies located ±1.25 and ±0.75 MHz away from the carrier frequencies of the 79 channels. Figure 7.8 on the next page shows the CSO count of beats occurring at ±1.25 MHz and is in good agreement with the results reported in [204] with the exception of the area of channels 4 and 5. The otherwise good agreement in the beat counts lends validity to the power spectral density results shown in Figure 7.7. (There are no power spectral density plots in [204].) The notch in the data in both figures around channels 4 and 5 occurs because channels 4 and 5 are offset a further 4 MHz from the rest of the channels, and further insight can be obtained by examining the numbers of beats for each kind of CSO. Figure 7.9 on page 256 shows the CSO beats occurring at ±0.75 MHz.

![Diagram of CSO power spectral density](image-url)
While CSO combinations occurring at 1.25 MHz below the carrier are absent in the vicinity of channels 4 and 5, there is a large upspike in the CSO located 0.75 MHz above the carriers in the vicinity of channels 4 and 5. Inspection of the VFD Tables for this modeling effort show that this occurs because subtractive combinations of all of other channels (other than channels 4 and 5) that fall around channels 4 and 5. Outside of the vicinity of channels 4 and 5, CSO that occurs 0.75 MHz above other carriers is caused by subtractive combinations of the carriers for channels 4 and 5 with the carriers for other channels. This leads to 1 or 2 beats in the ±0.75 MHz CSO other than around channels 4 and 5.

7.6.1.2 79 Channel CTB Results

Figure 7.10 on page 257 shows the output power spectral density of the Composite Triple Beat outputs occurring at the carrier frequencies of the 79 channels. Two forms of CTB are shown here. The Uncorrelated CTB is CTB that is formed by the mixture of three carriers other than the carrier at the channel in which the distortion occurs and is the sort of CTB that is widely reported elsewhere in the literature. The Correlated CTB is CTB that occurs as the result of phase (and frequency) cancellation among all of the other carriers in combination with the carrier frequency itself and is similar to the correlated IM distortion reported in Chapter [6].
The numbers of uncorrelated CTB beats on the 79 channel plan is shown in Figure 7.11 on the following page, and this agrees with the results in [204] other than around channels 4 and 5. In the vicinity of channels 4 and 5, the situation again is that the beats creating CTB at those frequencies must necessarily include one of channel 4 or 5, so the numbers of combinations of carriers forming valid beats is limited.

Figure 7.9: ±0.75 MHz CSO beats for a 79 channel plan.
Figure 7.10: CTB power spectral density for a 79 channel plan.

Figure 7.11: CTB beats for a 79 channel plan.
7.6.2 158 Channel Plan CSO/CTB Results

The results in this section are distortion products occurring within the $W_m(f)$ outputs in Figure 7.1 on page 248 for a channel plan with 158 carriers as input. CSO results that occur in $W_2(f)$ are given first, followed by a section on the CTB results that occur in $W_3(f)$. Just as in the 79 channel case, the inputs were set to a level of 24 dBmV, so the linear output for all channels was flat at 44 dBmV. The linear output is not shown here.

7.6.2.1 158 Channel CSO Results

Figure 7.12 shows that the CSO results for 158 channels are similar to those for 79 channels, with simulated distortion power levels somewhat higher than the 79 channel case due to the presence of a greater number of beats, as is indicated in Figures 7.13 on the following page and 7.14 on the next page. The gap in CSO ±1.25 MHz beats around channels 4 and 5 remains, and the upspike in 0.75 MHz CSO beats above the carriers around channels 4 and 5 is present as well.

Figure 7.12: CSO power spectral density for a 158 channel plan.
Figure 7.13: ±1.25 MHz CSO beats for a 158 channel plan.

Figure 7.14: ±0.75 MHz CSO beats for a 158 channel plan.
7.6.2.2 158 Channel CTB Results

Figure 7.15 shows the output power spectral density of the Composite Triple Beat outputs occurring at the carrier frequencies of the 158 channels. As in Section 7.6.1.2 on page 255.

Uncorrelated and Correlated forms of CTB are shown, and an increase the distortion power level is present due to the increased numbers of beats. The numbers of uncorrelated CTB beats on the 158 channel plan is shown in Figure 7.16 on the next page. The results here show roughly a 4-fold growth in the number of CTB beats corresponding to a doubling of the numbers of channels for the 79 channel plan. These results speculatively suggest that there is an upper limit on the numbers of channels in this frequency plan given an allowable level of distortion, but further data is needed on more channel plans to establish firm trends of distortion levels as a function of the number of channels.

7.6.3 158 Channel Plan Filter Results

In the previous sections, the focus was on forms of distortion created by the memoryless nonlinearities. These distortions were present in the signals $W_m(f)$ in Figure 7.1 on page 248. In this section, the focus will be to demonstrate that the filters $K_m(f)$, which implement the memory
in the amplifier model under consideration, produce a response in line with that predicted in Section 7.5.3 on page 252. To achieve this, the linear output and 2nd and 3rd order harmonic distortion produced by the memoryless blocks in Figure 7.1 on page 248 will be examined before filtering — as they appear in the \( W_m(f) \) outputs — and after filtering as they appear in the outputs \( Y_m(f) \).

Referring to Figure 7.1 on page 248, Figure 7.17 on the following page shows the linear output before and after the filter \( K_1(f) \), Figure 7.18 on the next page shows the 2nd order harmonic output before and after the filter \( K_2(f) \), and Figure 7.19 on page 263 shows the 3rd order harmonic output before and after filter \( K_3(f) \).

### 7.6.4 Computational Considerations

Using a tabular format similar to that of Table 5.4 on page 211, the costs (in time) of computing the results of this chapter are presented. These results were computed on an AMD Opteron 275 platform with 16 GBytes of DRAM running a 64-bit version of Matlab\textsuperscript{®} under a Linux operating system. The number of VFD Table entries (corresponding to the number of unique phasors) for a frequency plan of 79 channels was 670,079, and the spectrum transform matrix
Figure 7.17: Effect of filtering the linear output for a 158 channel plan.

Figure 7.18: Effect of filtering the 2nd order harmonic for a 158 channel plan.
Figure 7.19: Effect of filtering the 3rd order harmonic for a 158 channel plan.

contained $4.4 \cdot 10^{-4}\%$ non-zero entries. For a doubling of the number of channels to 158, the number of VFD Table entries increased almost 10-fold to 5,309,433 — thus illustrating the combinatorial growth in VFDs as a function of numbers of input frequencies — and the sparse spectrum transform matrix had $5.6 \cdot 10^{-5}\%$ non-zero entries. The total time of computation for a CATV frequency plan of 79 channels is about 1 hour and 14 minutes, while for a frequency plan of 158 channels, the time required is much longer — about 61 hours and 10 minutes, or about 2.54 days. Most of the time — about 1.83 days — was spent filling the sparse spectrum transform matrix. Since this code is fairly straightforward to implement, a strong case for rewriting the spectrum transform matrix construction function in a high-level language can be made. Note that the function evaluation times — basically the time spent doing sparse matrix-vector multiplication — was very low in both cases.
Table 7.3: Costs of computation for CATV amplification.

<table>
<thead>
<tr>
<th>Computation Item</th>
<th>79 Channel Plan Computing Time (sec)</th>
<th>158 Channel Plan Computing Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VFD Table</td>
<td>165.85</td>
<td>1536.40</td>
</tr>
<tr>
<td>Spectrum Mapping Table</td>
<td>1624.20</td>
<td>60158.85</td>
</tr>
<tr>
<td>Spectrum Transform Matrix</td>
<td>2633.05</td>
<td>158514.82</td>
</tr>
<tr>
<td>Function Evaluation</td>
<td>0.29</td>
<td>3.72</td>
</tr>
<tr>
<td><strong>Totals</strong></td>
<td>4423.39</td>
<td>220213.7</td>
</tr>
</tbody>
</table>

It was discovered during this work that the platform was operating in the Advanced Configuration and Power Interface (ACPI) thermal throttling state T1 [217], which specifies a duty cycle where the processor clock operates 87.5% of the time and is masked the other 12.5% of the time. With the Opteron 275 specified to have a nominal clock frequency of 2.2 GHz, this means that it effectively operated as a 1.925 GHz processor. Note that this was not desired behavior — the ACPI State T0 (clock operation 100% of the time and no throttling at all) was desired — but the platform operating system refused commands to move to that state. At the time of the completion of this work, correspondence was underway with Tyan (the platform’s manufacturer) on the problem, which may be rooted in tables of ACPI information passed to the operating system by the platform’s Basic Input/Output System (BIOS).

### 7.7 Concluding Remarks

This chapter demonstrated the utility of the AOM Toolbox for simulating the nonlinear response of an amplifier to large numbers of non-harmonically related multicarriers. The Composite Second Order (CSO) and Composite Triple Beat (CTB) beat counts and average distortion power spectral densities were computed across the frequency band for non-zero carriers spanning the frequency plan. Building upon the results introduced in Chapter 5, the average distortion results here were computed by assuming that the phases of the carriers were uncorrelated (valid for real CATV broadcast amplification since each channel originates from different sources) and using only a single simulation. Good agreement was seen for the case of 79 channels compared to previous results [204], which lends credibility to the 158 channel results here despite the
absence of measurements or other references.

It was seen that the standard NTSC frequency plan used for analog television signal propagation over CATV trunk cables has two interesting dips in the 1.25 MHz spaced CSO and the CTB distortion around channels 4 and 5 (as designated by (7.1) on page 246) along with an upspike in the 0.75 MHz offset CSO distortion in the vicinity of the same channels. These deviations in the distortion behaviors have not been previously reported. The results here demonstrate progress toward creating a practical environment for modeling the response of broadband amplifiers to collections of large numbers of tones or modulated carriers. Further progress in this area might ultimately be applied toward economizing the design of transmission systems using such amplifiers, which have historically been Class A amplifiers operating at low gains.

One disappointing result here is the cost of computation for the 158 channel results. The reported time to construct the spectrum transform matrix is lengthy here. Handling larger numbers of channels in a tolerable turnaround time will require re-implementation of at least the spectrum transform matrix construction code in a high-level language.
Chapter 8

Conclusion

8.1 Summary

The objective of the work in this study was two-fold — to develop a firm mathematical foundation for the Arithmetic Operator Method and to implement the method in the form of a Matlab® toolbox which might serve as a springboard for further applied research. Both objectives were achieved.

The first objective — laying a firm mathematical foundation — was achieved by starting with the first principles of Fourier analysis of generalized functions and comprehensively showing how collections of discrete sinusoids passed through polynomials in the time-domain transforms into collections of Dirac delta functions undergoing repetitive convolution in the frequency domain. This is the first time that such an extensive mathematical analysis has been done to conclusively demonstrate that the frequency-domain techniques implemented by the Arithmetic Operator Method are rooted in frequency-domain convolution as the Fourier transform of time-domain multiplication. However, this was not the only fundamental mathematical result that was obtained. In this work, the Vector Frequency Description (VFD) was introduced, and it was shown that when the VFD is used as the basis for defining spectral vector elements, then output spectral content at each spectral vector element had a phase that was uncorrelated from every other spectral vector element (assuming the input spectral vector content was of uncorrelated phase). While spectral content at different frequencies is generally
known to be uncorrelated, in this work it was shown that output spectral vector elements have uncorrelated phase even when two spectral vector elements had the same numerical frequency. Use of the VFD thus made it possible to easily separate correlated and uncorrelated spectral output, and this was demonstrated for the first time in a frequency-domain simulation environment. One other fundamental mathematical result presented for the first time focused on a critical property of the spectrum transform matrix: When a spectrum transform matrix is constructed from a conjugate-symmetric (necessarily two-sided and thus a representation of a real signal) spectral vector, then the spectrum transform matrix is Hermitian. As a consequence, eigendecomposition becomes an available avenue for the direct solution of problems instead of the usual method of iteratively solving systems of linear equations or the use of Krylov subspace methods. Finally, a detailed explanation of how to compute the hyperbolic tangent and exponential functions (first implemented by Chang) with the Arithmetic Operator Method was given.

Implementation of the Arithmetic Operator Method in the form of the AOM Toolbox also achieved the goal of furnishing a toolbox written predominantly in Matlab® (with two minor excursions into C-language codes to furnish capability not available in Matlab®) that can serve as a launching point for further research — by researchers in the field of electrical engineering as well as those in other disciplines in the physical sciences where there is an interest in modeling nonlinear systems in the frequency domain. Mechanical stochastic modeling is one possible discipline of immediate application. Differing from previous work, the implementation of AOM in the AOM Toolbox extended the spectra on which it could operate to both sides of the frequency-domain and then made it possible for the double-sided spectra to be expressed in real-valued form (two spectral vector elements per phasor) or complex-valued form (one spectral vector element per phasor). The complex-valued form is a critical ingredient for facilitating solutions using eigendecomposition. The AOM Toolbox also implemented the construction of spectrum transform matrices exclusively in sparse form, using the sparse matrix techniques built into Matlab®. This facilitates the investigation of large-scale problems beyond those possible in previous work and also renders moot the usual large difference in storage requirements between single-sided and double-sided analysis, since the storage requirements for sparse matrix implementations are essentially the same regardless of whether the analysis is single-sided or double-sided. A related improvement is a dramatic reduction in the computational effort required to produce a spectrum mapping table: For a VFD table with $K$ elements
(i.e. a spectral vector with $K$ phasors) in the previous work, $O(K^3)$ operations were required to perform spectrum mapping, whereas in the AOM Toolbox this has been reduced to $O(K^2)$. Without this reduction in computational complexity, exploitation of Matlab®’s sparse matrix capability would not be possible. The AOM Toolbox algorithms were described in pseudocode form in the body of the dissertation and listings with embedded documentation are furnished in an appendix. Table 8.1 provides a summary comparison of the capabilities made available by the AOM Toolbox with those available in other well-established environments.

<table>
<thead>
<tr>
<th>Metric</th>
<th>AOM Toolbox</th>
<th>Transient Simulation</th>
<th>Harmonic Balance Simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic Range</td>
<td>Nearly unlimited</td>
<td>40 dB (Spice) 125 dB (fREEDA™)</td>
<td>Approximately 120 dB</td>
</tr>
<tr>
<td>Requires Fourier Transforms</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Potential for separating Correlated from Uncorrelated Output?</td>
<td>Yes, by design of the VFD Table</td>
<td>Only with time-intensive post-simulation correlation computations</td>
<td>No</td>
</tr>
<tr>
<td>Enforces Correlation on Uncorrelated Phase Sources?</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Output vectors available for each order of nonlinearity?</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Average output power of nonlinear device to uncorrelated phase multitones in 1 simulation?</td>
<td>Yes</td>
<td>No, Requires Monte Carlo Techniques</td>
<td>No, Requires Monte Carlo Techniques</td>
</tr>
</tbody>
</table>

Applied research with the AOM Toolbox was begun in Chapter 4 of this work by verifying the correctness of results produced by the AOM Toolbox through comparison to time-domain results produced by other means. This was followed by an investigation into the response of a memoryless nonlinear amplifier to multitone inputs with uncorrelated phase
in Chapter 5, through the use of the VFD table to define spectral vector content, it was shown that it is possible to accurately predict the average output power spectral density under these circumstances in a single simulation. This is something that is simply not possible in time-domain environments due to enforced phase correlation, and has only been produced in a frequency-domain environment by one other group of researchers (Carvalho and Pedro), who obtained their results by creating and managing an average power vector to which incremental power contributions were added during the convolution process. Going beyond the results of those researchers, however, the results here also furnished the correlated and uncorrelated IM output from a single simulation as a result of using the VFD table to define the spectral content. Also, by explicitly running an ensemble of simulations with differing input phase regimes and observing the same average output power spectral density, it was thus shown in this work that the average output power spectral density of a memoryless nonlinear amplifier in response to multitone inputs with uncorrelated phase is invariant of the input phase.

In Chapter 6, the response of a nonlinear amplification chain to two Linear FM chirp inputs was investigated using the AOM Toolbox and results were compared to the Fourier transform of results from an implementation of the same chain using fREEDA™, with good agreement seen except in the 3rd order harmonic response, where the likely discrepancy is inadequate modeling of harmonic response in the complex power series model used by the AOM Toolbox. Chapter 6 also demonstrated the use of bandpass filtering in the AOM Toolbox with comparisons — where possible — to an equivalent z-domain filter output from the FFT of a fREEDA™ transient simulation. The ability of the z-domain filters to attenuate over broad frequency bands was also demonstrated.

The results of Chapter 5 were used to good effect in Chapter 7, where an investigation of distortion produced by the response of a broadband television broadcast amplifier to multicarrier inputs was conducted using only one input phase regime. Good agreement among distortion counts (or beats) was seen for a 79-channel frequency plan with previous results, buttressing the credibility of results presented here for a 158-channel frequency plan. Long times of computation for the 158-channel plan, however, indicate that continued development of the AOM Toolbox should also include rewriting some toolbox functions in high-level languages.
8.2 Recommendations for Further Study

There are many avenues upon which the body of work presented here may be further researched and developed.

Further Research in the AOM Toolbox

Presently, exact counts are not available for the number of elements in the Vector Frequency Description table, as a function of the number of input tones and order of nonlinearity. Instead, a formula for the upper bound of the counts has been presented. Further investigation into the combinatorics (by those with a gift for combinatorial analysis) of the Vector Frequency Description may result in exact counts.

The correlation properties of spectral vectors based upon the Vector Frequency Description has been established for simple polynomial transfer functions in this work, but further work must be done to determine how useful it is for rational polynomial transfer functions. When eigendecomposition is used as the means to a solution, there is no issue with rational polynomial functions since the decomposition leads to an inversion of a diagonal matrix and the spectral content will remain uncorrelated by this technique. One issue with eigendecomposition is that it requires non-sparse spectrum transform matrices, and so its utility for large-scale multitone problems may be limited. Sparse techniques don’t preclude the use of linear systems solutions, but when linear algebraic techniques are used, a non-diagonal spectrum transform matrix will necessarily be inverted (i.e the divisor polynomial function must ultimately be cast as a spectrum transform matrix which is then inverted and pre-multiplied by the vector result for the numerator polynomial). When a dense matrix is inverted, every element in the inverted matrix becomes a function of all of the elements of the original matrix. From the point of view of applied probability, then, the act of dense matrix inversion essentially enforces statistical correlation upon the data processed by a transfer function. Fully dense spectrum transform matrices are atypical in the AOM Toolbox, however. Banded matrices are more typical, and so the question to be investigated is how much degradation to the output is caused by enforced correlation when direct solutions of linear systems are used. This could be investigated by comparing the results of eigendecomposition to that of direct linear systems solutions for small-scale problems. Favorable results here would buttress the case for using of direct solutions of linear systems for large-scale problems where eigendecomposition would not be practical. Related to
the research task of investigating the effect of inverting sparse spectrum transform matrices on correlation properties should be the application of sparse direct methods of matrix inversion, which have recently received a lot of attention as alternatives to Krylov subspace methods \[218\]. Once the phenomenon of enforced correlation in inverted sparse spectrum transform matrices is better understood, a useful next step would be the integration of the VFD into a frequency-domain circuit simulation environment.

An alternate or concurrent avenue is the continued development of the behavioral simulation environment. Within the Matlab® environment itself, one possible area that might be immediately investigated is the use of a sparse representation for elements of the Vector Frequency Description. Presently, elements of the VFD are 8-bit signed quantities, and there is no provision for a sparse representation. This choice of representation permitted the scaling of the VFD to handle large numbers of input tones while preserving the ability to directly observe the contents of the VFD table in Matlab®. However, when very low orders of nonlinearity are considered — typically the case when problems with large numbers of input tones are used — then the VFDs will necessarily be sparse since their maximal 1-norms are dictated by the order of nonlinearity. For example, a 200-tone input passed through a 3rd order nonlinearity will have VFDs with at most 3 non-zero tuples, and in this case would require at most 72 bytes per VFD to implement in Matlab®’s sparse form, whereas 200 bytes are presently used. In its present Matlab® implementation, the AOM Toolbox is reasonably suited for investigating the response of a single nonlinear block to large-scale multitone inputs, and it is also capable of investigating the response of a cascade of nonlinear blocks to a small number of input tones (see Appendix D). However, it is not well-suited to investigating the response of a cascade of nonlinear blocks to a large number of input multitones. To enable that capability, the codes for construction of the VFD and Spectrum Mapping Tables and the code for constructing the Spectrum Transform Matrix should be rewritten in a high-level language, preferably as Matlab Executable (MEX) files, so that they can form high-performance drop-in replacements for the existing routines. The author’s personal experience with MEX files has shown that MEX file replacements typically execute with a speedup of 40 to 400 times that of their Matlab M-file versions.

In the realm of functional enhancements, the AOM Toolbox is useful for investigating the interaction of multitone carriers of modulated low-frequency spectra, then the AOM Toolbox is not able to analyze the problem. Thus the
extension of the AOM Toolbox to encompass envelope transient simulation – using AOM for the nonlinear analysis (in lieu of Harmonic Balance techniques) and time-marching transient simulation in for the baseband spectral content – should be considered and has already been proposed by N.B. Carvalho [219]. Since it would not be necessary to use two differing time scales, and since the analysis in the carrier frequency ranges would be conducted entirely in the frequency domain, an AOM-ET environment would not suffer computationally from the inherent stiffness of the systems of differential equations, as has been seen in envelope transient systems using HB methods [53, 54]. It is also likely that an AOM-ET environment would have far greater dynamic range than the existing Envelope Transient environments.

Further Research in the z-domain

In the area of z-domain filters for transient simulation in fREEDA™, one possible area of further research is the development of models to simulate the behavior of transmission paths other than the forward path, which is all that the present models can do. Such an undertaking would be more complex and probably require the use of strategies such as pole-zero placement [220] and iterative optimization, since the behavior of reflected and reverse transmission paths will usually not follow the formulaic development that holds for the forward path. However, the forward transmission models developed in this work will serve as useful architectural coding templates from which to derive the other signal path models.

Another area for further research in z-domain filtering would be the implementation of filter structures other than the canonical forms presented here. In the work done here, the emphasis was on bandpass Infinite Impulse Response (IIR) Butterworth and Chebychev Type I filters that are relatively straightforward to physically realize (albeit with possible problems in component tolerances), but not amenable to transient simulation using inverted S-domain behavioral models. Since the coding infrastructure is now in place, however, it should be possible to implement Finite Impulse Response (FIR) filters, particularly the lattice structures [202], which do not have physically realizable counterparts. The State-Space structures [221], are a natural fit since they mesh perfectly with the state variable methodology that underpins fREEDA™.

Research has already been conducted into the effects of finite word-length number systems on the accuracy of the implementations of filters [222, 223], and this research has been useful when implementations are done on embedded systems with number systems of limited
precision. For low-order filters, the impact within \textit{fREEDA}\textsuperscript{™} of implementation using the IEEE-754 number system may be safely ignored for the most part. However, when the filter order is high, leading to many cascaded stages, then it may be beneficial to order the cascaded elements in a way that minimizes the effect of the underlying number system. This may permit filter orders higher than those reported in this work to be reliably realized within the limits of the IEEE-754 number system.

Finally, it should be noted that other researchers have successfully modeled amplifiers with memory in the time domain using \textit{z}-domain implementations \cite{224}, and there is nothing precluding such model implementations in \textit{fREEDA}\textsuperscript{™}. Alternatively, amplifier models with memory may be implemented in \textit{fREEDA}\textsuperscript{™} using the time-delay mechanisms inherent in the ADOL-C library \cite{225}. 
BIBLIOGRAPHY


277

MTT-S International Microwave Symposium Digest, vol. 1, 2000, pp. 453–456. [Citation: Page 14]


[55] A. Passinsky and V. Veremey, “Envelope transient analysis: Key to first-pass successful wireless systems and IC designs,” in *Wireless Symposium Portable By Design*, San Jose, California, 2000. [Citation: Page 15]


[81] ———, “Simulation of intermodulation distortion in mesfet circuits with arbitrary frequency separation of tones,” in *1986 IEEE MTT-S International Microwave Symposium Digest*, 1986, pp. 547–550. [Citation: Page 17]


[101] H. Karami and K. Sheshyekani, “Modified arithmetic operator method (AOM),” June 2007, reports improvements in index frequency and spectrum mapping algorithms from an early prototype of the AOM Toolbox. [Citation: Page 25]


[107] G. Baruffa and G. Reali, “A fast algorithm to find generic odd and even intermodulation products,” *IEEE Transactions on Wireless Communications*, vol. 6, no. 10, pp. 3749–3759, October 2007. [Citation: Page 26]


[112] S. A. Maas, Nonlinear Microwave and RF Circuits, 2nd ed. Artech House, 2003. [Citation: Page 26]


[115] J. C. Pedro and N. B. Carvalho, Intermodulation Distortion in Microwave and Wireless Circuits. Artech House, 2003. [Citation: Page 26]


[128] L. Schwartz, “Théorie des distributions, Tome II,” *Actualités Scientifiques et Industrielles*, no. 1122, 1951, published in French. [Citation: Page 34]

[129] ——, “Théorie des distributions, Tome I,” *Actualités Scientifiques et Industrielles*, no. 1245, 1957, published in French. [Citation: Page 34]


[132] M. J. Lighthill, *Introduction To Fourier Analysis And Generalized Functions*. Cambridge, UK: Cambridge University Press, 1958. [Citation: Page 34]

[133] H. P. Hsu, *Applied Fourier Analysis*. Harcourt Brace Jovanovich, 1984. [Citation: Page 34]


[136] R. D. A. Maurice, *Convolution And Fourier Transforms for Communications Engineers*. Halstead Press, 1976. [Citation: Page 35]


[139] ——, *Signal Analysis*. McGraw-Hill, 1977. [Citation: Page 36]


[160] J. Hall, North Carolina State University, October 2003, private communication. [Citation: Page 118]


[175] G. Strang, Computational Science and Engineering. Wellesley-Cambridge Press, 2007. [Citation: Page 137]

[176] G. W. Stewart, Afternotes goes to Graduate School. SIAM, 1998. [Citation: Page 137]


[181] A. Quarteroni and F. Saleri, Scientific Computing with MATLAB. Berlin, Germany: Springer-Verlag, 2003. [Citation: Page 139]


[198] D. Disco, “A guide to the FFT,” August 9, 2004, from the MATLAB Central File Exchange, a guide to the use of the FFT algorithm. [Citation: Page 220]


[209] R. Raich, “Nonlinear system identification and analysis with applications to power amplifier modeling and power amplifier predistortion,” Ph.D. dissertation, Georgia Institute of Technology, Atlanta, Georgia, 2004. [Citation: Page 248]

[211] ——, “Analysis and modeling of intermodulation distortion in wide-band Cable TV channels,” *IEEE Transactions on Communications*, vol. 35, no. 5, pp. 568–572, May 1987. [Citation: Page 248]


[213] Freescale Semiconductor, “Document Number: MHW8182CN,” April 2006, revision 3. [Citation: Page 248]

[214] RFHIC, “2F1G18P Datasheet,” version 1.1. [Citation: Page 248]

[215] ——, “2F1G20P Datasheet,” version 1.0. [Citation: Page 248]


[219] N. B. Carvalho, Instituto de Telecomunicações, Universidade de Aveiro, Portugal, 2007, private communication. [Citation: Page 272]


[222] L. B. Jackson, “Roundoff noise bounds derived from coefficient sensitivities for digital filters,” *IEEE Transactions on Circuits and Systems*, vol. 23, no. 8, pp. 481–484, August 1976. [Citation: Page 272]


[227] A. Walker, J. Hall, and M. Buff, “Multi-slice behavioral model,” April 2003. [Citation: Page 294]


[240] E. A. Guillemin, Synthesis of Passive Networks. John Wiley & Sons, 1957. [Citation: Page 311]


[244] M. B. Steer, North Carolina State University, 2006, private communication. [Citation: Page 325]


[246] K. G. Gard, North Carolina State University, 2004, private communication. [Citation: Page 370]


[250] ——, “Generalized superposition,” *Information and Control*, vol. 11, no. 5/6, pp. 528–536, November-December 1967. [Citation: Page 374]


[253] M. B. Steer and C. E. Christoffersen, “Circuit Simulation: ECE718 Class Notes,” January 2001. [Citation: Page 414]
Appendices
Appendix A

Z-Domain Filtering in a Transient Microwave Circuit Simulator

A.1 Background

The radar modeling discussed in Chapter 6 of this dissertation was compared to the Fourier transform of time-domain results produced by the fREEDA™ simulator. However, the fREEDA™ environment itself required some incremental developments in the realm of narrow-band filtering in order to fulfill its role as a comparative reference. In this appendix, differing from the main body of the dissertation, the frequency variable is \( \omega \) (rather than \( f \)), where \( \omega = 2\pi f \).

In transient circuit simulators filters are traditionally modeled using Laplace inverse pole-zero descriptions or using compact models of circuit elements. Accurate modeling typically requires small time-steps and the simulation problem is analogous to capturing small differences of large numbers [226]. Furthermore, the inverse Laplace methodology occasionally results in the creation of non-causal models. Also, since S-domain filters were previously found to be less than robust even in the case of loose specifications [223], the principal investigator of the fREEDA™ project determined that fREEDA™ could benefit from the presence of discrete-time behavioral filter models rooted in the z-domain, since future radar circuits and systems would be subject to even tighter specifications and higher levels of noise and interference during chirped operation. A further benefit is that microwave filters which do not fit into the lumped-
circuit paradigm of circuit simulators, such as parallel coupled-line bandpass filters \cite{228}, can be conveniently modeled as lumped elements in behavioral form.

A discrete-time filter is similar to a digital filter in that outputs are defined only at discrete points in time, but unlike a digital filter, a discrete-time filter makes no attempt to quantize the amplitude of the signals it processes \cite{202}. However, the theory \cite{229} establishing the basic equivalence of analog and digital signal processing through the use of the \textit{bilinear z-transform} (BZT) method holds without regard to amplitude quantization. Thus when time is uniformly discretized, it is possible to substitute a discrete-time equivalent filter for any linear filter provided that the Nyquist criterion \cite{120} is observed.

Although transient circuit simulation is intended to simulate continuous-time systems, internally the simulator environments are inherently discrete-time in nature. The simulation consists of solving systems of equations at discrete time intervals and using an interpolating function, usually the trapezoidal or Backward Euler rule \cite{13}, to integrate between the two time intervals. Thus it would seem natural to implement discrete-time filters, using the time step as the sampling interval for the discrete-time filter. Unfortunately, this has not been possible in many Spice-based simulators because the simulator engine dynamically varies the time step to meet algorithmic constraints intended to assure local convergence \cite{11}. Many Spice users are unaware that the time steps they specify in netlists are merely \textit{reporting intervals} \cite{230} while the simulator engine may step to many intermediate points between reporting intervals. In \textit{fREEDA™}, however, it is possible to run transient simulations with fixed time steps, so discrete-time filters will function in \textit{fREEDA™}, with the sampling rate of the filter fixed by the time step of the simulation.

At least one commercial simulator environment has permitted the definition of discrete-time filters outside the main simulator engine, with the resulting filter coefficients passed as model parameters. The weakness of this approach is that the filter must be re-synthesized and the coefficients passed as model parameters whenever the time step is changed. Here, for the first time, a discrete-time filtering model, which takes the traditional frequency-based parameters used to specify \textit{S}-domain filters for synthesis (and synthesizes the complete filter at the beginning of each simulation), has been seamlessly implemented for transient simulation in a circuit simulator.
A.2 The MNAM Conditioning Problem

The range of filters that can be modeled in a typical commercial transient simulator is shown in Figure A.1 for a Butterworth bandpass filter modeled using all-pole elements. This figure indicates regions where valid results are obtained with results valid for lower orders and higher bandwidths. When driven by a linear FM chirp source, a generic radar signal for example,

![Figure A.1: Synthesis limitations of a leading commercial simulator compared with the model developed here and implemented in fREEDA™.](image)

a convergent solution can only be obtained for a filter with a fractional bandwidth greater than three percent and when the order of the lowpass prototype of the filter order is less than 10 or so. The valid operating region is somewhat wider for sinusoidal excitation. The root cause of the problem is ill-conditioning of the Modified Nodal Admittance Matrix (MNAM) developed from the associated discrete model during iterative simulation. The problem is exacerbated by the large range of voltage and current levels in high Q filters.

Even though higher-frequency operation of modern circuits increases the possibility
for ill-conditioned MNAMs, the potential for a given circuit to have an ill-conditioned MNAM was known to the creators of Spice. For example, consider the simple two-stage ladder circuit shown in Figure A.2. Here, a 50 MHz sinusoidal voltage source drives the ladder, which is terminated in a 50-ohm load. In this circuit, the first stage has element values \( L_1 = 1 \mu\text{H} \) and \( C_1 = 1 \text{pF} \), and the second stage element values are multiplied by a scale factor, \( \psi \), which ranges over several orders of magnitude. The frequency-domain MNAM for this illustrative circuit is easily obtained using the techniques described in [232]. Figure A.3 on the following page shows (the logarithm of) the condition number of the MNAM for this circuit as a function of (the logarithm of) the scaling factor \( \psi \). For large scaling factors – indicating especially stiff circuits – the condition number grows without bound leading to invalid simulation results. Note that the character of the curve here does not change significantly with a change in driving point frequency; the knee of the curve simply moves to the left or right.

The matter of ill-conditioning is arguably worse in the time domain. This can be seen by considering how the construction of the time-domain MNAM entries unavoidably involves entries with largely varying orders of magnitude. To see this, consider a single second-order factor of a tenth-order lowpass prototype Butterworth filter,

\[
H_{LP}(s) = \frac{k_0 s + k_1}{s^2 + b_0 s + 1},
\]  

(A.1)

where \( k_0 \) and \( k_1 \) are the residues of each of the rational fractions resulting from a partial fraction expansion and the \( b_0 \) factors can be found in [233] for a variety of filter types. This filter is
subsequently converted to a bandpass filter through the substitution $H_{BP}(S) = H_{LP}\left(\frac{S^2 + \omega_0^2}{B S}\right)$, where $B$ is the 3-dB bandwidth and $\omega_0$ is the center frequency. When this substitution is made, the result is

$$H_{BP}(S) = \frac{k_0 B S^3 + B^2 k_1 S^2 + k_0 B \omega_0^2 S}{S^4 + b_0 B S^3 + (2\omega_0^2 + B^2) S^2 + b_0 B \omega_0^2 S + \omega_0^4}.$$  \text{(A.2)}

Creation of the time-domain MNAM entries for the filter is accomplished via analytic Laplace transform inversion of (A.2) and the addition of state variables to account for the extra time-domain derivatives. As a result, the unit coefficient of $S^4$ in (A.2) and the very large factor $\omega_0^4$ occupy separate MNAM entries with no way of resolving large disparities in their magnitudes. Thus the ill-conditioning within the time-domain MNAM is induced by the intrinsic center frequency of operation and is unrelated to whether or not the overall system is stiff due to the interaction of other components. As an example of the atrocious ill-conditioning that can occur, a conventional implementation in \textit{fREEDA}™ of a tenth-order Butterworth bandpass filter

![Figure A.3: Condition number of MNAM for the two-stage ladder circuit for various ladder scale factors at 50 MHz.](image)
with a center frequency of 1 MHz and a passband bandwidth of 100 kHz (i.e. 10%) yielded an
MNAM with a condition number of 2.9 × 10^{43}.

Here a strategy is presented for modeling filters of high order (up to order 83) and
fractional bandwidths as low as 0.1%. The central concept is using a (pre-warped) z-domain
approximation. The filter model makes only unit entries in the MNAM while filling in the
right-hand side known vector of the matrix-vector circuit equation, thus avoiding the creation
(or exacerbation) of MNAM ill-conditioning. This technique was implemented in the \textit{fREEDA}\textsuperscript{™} circuit simulator [234] (see also publication 2 in Section 1.4.2 on page 10), where the environment
has been modified to accommodate Spice-like transient circuit simulation for some elements and
z-domain simulation for others. The procedure is described for lowpass and bandpass filters
and results presented for transient simulation of bandpass filters.

### A.3 Z-Domain Filter Transformation

The process of modeling a lowpass filter begins with the specification of the cutoff frequency,
passband flatness, stopband attenuation, and often insertion loss. For a bandpass filter the
process begins with the specification of passband and stopband frequencies, passband flatness,
stopband attenuation, and often insertion loss. The bandpass specifications are then trans-
formed to lowpass specifications, and from there well-defined \cite{220} steps using BZT yield an
equivalent discrete-time bandpass filter:

1. Determine the pre-warped frequency specifications from the analog filter specifications
2. Determine the lowpass transfer function in the s-domain using the pre-warped
   frequency specifications, then, for bandpass filters only, apply the lowpass-to-
   bandpass frequency transformation

\begin{equation}
S \leftrightarrow \frac{S^2 + \omega_o^2}{S}, \quad (A.3)
\end{equation}

which translates the filter behavior into a passband in S geometrically centered at
\( \omega_o \) (skip this step for lowpass filters).

3. Perform the bilinear transformation,

\begin{equation}
S \leftrightarrow \frac{z - 1}{z + 1}, \quad (A.4)
\end{equation}
to convert the filter from the $S$-domain to the discrete-time frequency domain (i.e. $z$-domain) and algebraically reduce the transfer function to a form that can be realized.

Steps 2 and 3 will be described in detail in the following sections.

Step 1, frequency pre-warping, converts a continuous-time frequency $\omega_a$ to its discrete-time equivalent $\omega_p$ through application of the formula

$$\omega_p = \tan\left[\frac{\omega_a T}{2}\right]. \quad (A.5)$$

Here $T$ is the fixed step time in circuit simulation. However, from the discrete-time filtering perspective, $T$ also functions as the sampling interval and is related to the sampling frequency $\omega_s$ of the discrete-time filter by

$$T = \frac{2\pi}{\omega_s}. \quad (A.6)$$

Substituting (A.6) into (A.5) leads to

$$\omega_p = \tan\left[\frac{\omega_a}{\pi\omega_s}\right]. \quad (A.7)$$

Application of the Nyquist criterion requires that $\max(\omega_a) < \omega_s/2$, and after applying the criterion it can be seen that a one-to-one relationship exists between $\omega_a$ and $\omega_p$.

From here forward, all frequencies will be assumed to be pre-warped. Since the construction of bandpass filters includes the determination of properties from a lowpass form, the lowpass filter form will be discussed in detail first.

A.4 The Butterworth Lowpass Filter Model

Continuous-time Butterworth lowpass filters are all-pole filters having maximal flatness in the passband [233, 234, 235, 236]. Such filters may have an even or odd number of poles, and the poles are always equally-spaced and lie on a semi-circle about the origin in the left half of the continuous-time frequency domain (i.e. Laplace or $s$-domain). An even number of poles always occur in conjugate pairs so that a pole pair has a second-order polynomial form in the denominator of its $s$-domain transfer function. The single odd pole occurs only on the negative real axis. By cascading pairs of poles with the odd singleton pole, an arbitrary order Butterworth filter may be constructed.
Construction of a Butterworth lowpass discrete-time filter begins with the netlist parameters passed to the simulator engine. These parameters specify frequencies corresponding to the s-domain, i.e. frequencies that are not pre-warped (but are immediately pre-warped before any further computations). The parameters may explicitly specify the $-3$ dB frequency and number of poles of the filter; otherwise the passband and stopband edge frequencies implicitly specify the cutoff frequency and the number of poles. Figure A.4 shows the parameters that are usually explicitly passed. In Figure A.4, $\varepsilon$ and $\lambda$ are the passband and stopband flatness parameters, respectively. Usually, these are specified in the netlist in negative dB form, and in that case

\[
\varepsilon_{dB} = 10 \log_{10} \left( \frac{1}{1 + \varepsilon^2} \right),
\]

\[
\lambda_{dB} = 10 \log_{10} \left( \frac{1}{1 + \lambda^2} \right),
\]

Figure A.4: Typical parameters for a Butterworth lowpass filter.
so that

\[
\varepsilon = \sqrt{10^{-\frac{\varepsilon dB}{20}} - 1}
\]

\[
\lambda = \sqrt{10^{-\frac{\lambda dB}{20}} - 1}.
\]  

(A.10)  

(A.11)

Also in Figure A.4 on the previous page, \(\omega_p\) and \(\omega_s\) are the edge frequencies for the passband and stopband corresponding to the points in the magnitude response where \(\varepsilon\) and \(\lambda\) are defined. Given the nature of Butterworth filters, the following equations relate the \(-3\) dB cutoff frequency \(\omega_c\) and the number of poles, \(N\):

\[
\frac{1}{1 + \varepsilon^2} = \frac{1}{1 + \left(\frac{\omega_p}{\omega_c}\right)^{2N}} \quad \Rightarrow \quad \left[\frac{\omega_p}{\omega_c}\right]^{2N} = \varepsilon^2
\]

(A.12)

\[
\frac{1}{1 + \lambda^2} = \frac{1}{1 + \left(\frac{\omega_s}{\omega_c}\right)^{2N}} \quad \Rightarrow \quad \left[\frac{\omega_s}{\omega_c}\right]^{2N} = \lambda^2
\]

(A.13)

Combining the two preceding equations to eliminate \(\omega_c\) yields the following expression,

\[
N = \left[\frac{\log_{10}\left(\frac{\varepsilon}{\lambda}\right)}{\log_{10}\left(\frac{\omega_p}{\omega_s}\right)}\right],
\]

(A.14)

for the number of poles in the filter, where the \([\ ]\) and \([\ ]\) brackets denote that \(N\) is to be rounded up to the nearest integer. With \(N\) determined, the \(-3\) dB frequency \(\omega_c\) may be determined from either (A.12) or (A.13).

Following [235], a continuous-time Butterworth lowpass filter of order \(N\) has poles \(p_k\) located at the following locations for \(k = 0, 1, \ldots N - 1,\)

\[
u_k = \omega_c\left[\sin \left(\frac{(2k + 1)\pi}{2N}\right)\right]
\]

(A.15)

\[
u_k = \omega_c\left[\cos \left(\frac{(2k + 1)\pi}{2N}\right)\right]
\]

(A.16)

\[
p_k = u_k + jv_k
\]

(A.17)

\[
\overline{p}_k = p_{N-k} = u_k - jv_k,
\]

(A.18)

where \(\overline{p}_k\) denotes the complex conjugate of pole \(p_k\). When \(N\) is odd, the pole at index \(k = (N - 1)/2\) is purely real. Thus the transfer function \(H(s)\) of an \(N\)-pole Butterworth lowpass
filter is given by

\[
H(s) = \begin{cases} 
  A \prod_{k=0}^{N-1} \frac{1}{(s - p_k)(s - \overline{p_k})} & \text{for } N \text{ even,} \\
  \frac{A}{(s + p_r)} \prod_{k=0}^{N-1} \frac{1}{(s - p_k)(s - \overline{p_k})} & \text{for } N \text{ odd.}
\end{cases}
\quad (A.19)
\]

In (A.19), \( p_r \) is the magnitude of a pole that occurs on the negative real axis, and \( A \) is a constant that sets the filter gain. For unity gain

\[
A = \begin{cases} 
  \prod_{k=0}^{N-1} |p_k|^2 & \text{for } N \text{ even,} \\
  p_r \prod_{k=0}^{N-1} |p_k|^2 & \text{for } N \text{ odd.}
\end{cases}
\quad (A.20)
\]

Let the transfer function for an arbitrary pole pair be given by

\[
H_k(s) = \frac{1}{a_k s^2 + b_k s + c_k}.
\quad (A.21)
\]

Substituting (A.15)–(A.18) into (A.21),

\[
H_k(s) = \frac{1}{s^2 + 2\omega_c \left[ \sin \left( \frac{(2k+1)\pi}{2N} \right) \right] s + \omega_c^2},
\quad (A.22)
\]

so that

\[
a_k = 1
\quad (A.23)
\]

\[
b_k = 2\omega_c \left[ \sin \left( \frac{(2k+1)\pi}{2N} \right) \right]
\quad (A.24)
\]

\[
c_k = \omega_c^2
\quad (A.25)
\]

Now, applying the bilinear \( z \)-transform, \( s = (z - 1)/(z + 1) \) to (A.22) results in \( H_k(s) \Rightarrow H_k(z) \) so that

\[
H_k(z) = \frac{z^2 + 2z + 1}{(a_k + b_k + c_k)z^2 + 2(-a_k + c_k)z + (a_k - b_k + c_k) + 1 + 2z^{-1} + z^{-2}}.
\quad (A.26)
\]

\[
= \frac{(a_k + b_k + c_k) + 2(-a_k + c_k)z^{-1} + (a_k - b_k + c_k)z^{-2}}{(a_k + b_k + c_k) + 2(-a_k + c_k)z^{-1} + (a_k - b_k + c_k)z^{-2}}.
\quad (A.26)
\]
For the case of a pole singleton on the negative real axis, when \( k = (N - 1)/2 = K \),

\[
H_K(s) = \frac{1}{b_K s + c_K},
\]

(A.27)

where

\[
b_K = 1 \quad \quad \quad \quad \quad \quad c_K = \omega_c .
\]

(A.28)

(A.29)

Applying the bilinear Z-transform,

\[
H_K(z) = \frac{z + 1}{(b_K + c_K)z + (-b_K + c_K)}
\]

\[
= \frac{1 + z^{-1}}{(b_K + c_K) + (-b_K + c_K)z^{-1}} .
\]

(A.30)

Finally, implementation in time-domain form is facilitated by normalizing the denominator of the transfer function of the \( k^{th} \) block so that the \( z^0 \) term is 1. This is done by dividing the terms in the denominator of the pole pairs by \((a_k + b_k + c_k)\), and by dividing the denominator of the pole singleton by \((b_K + c_K)\). In the synthesis process, these constant factors are then collected to form a scaling constant for the entire filter:

\[
A_d = \begin{cases} 
A \prod_{k=0}^{\frac{N}{2}-1} \frac{1}{(a_k + b_k + c_k)}, & \text{for } N \text{ even,} \\
\frac{A}{(b_K + c_K)} \prod_{k=0}^{\frac{N-1}{2}} \frac{1}{(a_k + b_k + c_k)}, & \text{for } N \text{ odd.}
\end{cases}
\]

(A.31)

Thus the final form for the transfer function of a pole pair, with the denominator constant factored out, is

\[
H_k(z) = \frac{1 + 2z^{-1} + z^{-2}}{1 + 2\left(\frac{a_k - c_k}{a_k + b_k + c_k}\right)z^{-1} + \left(\frac{a_k - b_k + c_k}{a_k + b_k + c_k}\right)z^{-2}} .
\]

(A.32)

For the pole singleton, the transfer function with the denominator constant factored out is

\[
H_K(z) = \frac{1 + z^{-1}}{1 + \left(\frac{-b_k + c_k}{b_K + c_K}\right)z^{-1}} .
\]

(A.33)

The discrete-time form of the filter, in a block cascade implementation, is shown in Fig A.5 on the following page. Both (A.32) and (A.33) may be expressed as

\[
H_k(z) = \frac{\alpha_{k0} + \alpha_{k1}z^{-1} + \alpha_{k2}z^{-2}}{1 + \beta_{k1}z^{-1} + \beta_{k2}z^{-2}} ,
\]

(A.34)
where the coefficients in (A.34) are defined in Table A.1 by collecting the terms from (A.32)–(A.33). If $N_k(z)$ is defined as the numerator of (A.34) and $D_k(z)$ is defined as the denominator of (A.34), then (A.34) can be expressed in a form, called a canonc form, requiring the minimum number of storage elements, by creating an intermediate function $W_k(z)$ as follows [237]:

$$H_k(z) = \frac{Y_k(z)}{W_k(z)} \cdot \frac{W_k(z)}{X_k(z)} = N_k(z) \frac{1}{D_k(z)} \Rightarrow \quad (A.35)$$

$$N_k(z) = \frac{Y_k(z)}{W_k(z)} \Rightarrow \quad (A.36)$$

$$Y_k(z) = \left(\alpha_{k0} + \alpha_{k1}z^{-1} + \alpha_{k2}z^{-2}\right) \cdot W_k(z) \quad (A.36)$$

$$1 \quad \frac{W_k(z)}{X_k(z)} \Rightarrow \quad (A.37)$$

$$W_k(z) = X_k(z) - \left(\beta_1 z^{-1} + \beta_2 z^{-2}\right) \cdot W_k(z). \quad (A.37)$$

Table A.1: $z$-domain Butterworth lowpass filter block coefficient definitions

<table>
<thead>
<tr>
<th>Canonical coefficient</th>
<th>Pole Pair</th>
<th>Pole Singleton</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k, N$ even or $k \neq K, N$ odd</td>
<td>$k = K, N$ odd</td>
<td></td>
</tr>
<tr>
<td>$\alpha_{k0}$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$\alpha_{k1}$</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$\alpha_{k2}$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$\beta_{k1}$</td>
<td>$\frac{a_k-b_k+c_k}{a_k+b_k+c_k}$</td>
<td>$\frac{-b_k+c_k}{b_k+c_k}$</td>
</tr>
<tr>
<td>$\beta_{k2}$</td>
<td>$\frac{a_k-b_k+c_k}{a_k+b_k+c_k}$</td>
<td>0</td>
</tr>
</tbody>
</table>

Finally, taking the inverse $z$-transform of (A.36) and (A.37),

$$y_k(n) = \alpha_{k0} \cdot w_k(n)$$

$$+ \alpha_{k1} \cdot w_k(n-1) + \alpha_{k2} \cdot w_k(n-2) \quad (A.38)$$

$$w_k(n) = x_k(n)$$

$$- \beta_{k1} \cdot w_k(n-1) - \beta_{k2} \cdot w_k(n-2). \quad (A.39)$$
The results in (A.38) and (A.39) define the internal operations of one of the cascaded blocks in Figure A.5 on the previous page. Figure A.6 shows the implementation of one block. The complete discrete-time filter is composed only of adders, multipliers, and delay elements – indicated by $Z^{-1}$ – to store previous samples. Real-time implementation of the block in Figure A.6 can be done with hardware adders, multipliers, and registers, but it is often implemented in real-time software as well, and in this case the delay elements will be implemented in low-latency main memory or registers within a microprocessor. In “offline” implementations operating on previously sampled and stored data, software-only implementations are typical, and these are described in [238]. The behavior of software implementing the block in Figure A.6 is an ordered two-step process at each time step:

1. Compute $w(n)$ and $y(n)$ from $x(n)$ and the previously stored values $w(n-1)$ and $w(n-2)$.

2. Update the stored element values: $w(n-2) \leftarrow w(n-1)$ and $w(n-1) \leftarrow w(n)$.

The circuit simulation environment presents some difficulties to the software implementation. First, the data presented are not real-time in nature (or sampled and stored “offline” versions of real-time data). Within a circuit simulator, data are Newton-iterated at every time step to solve a system of equations. The real-time input $x(n)$ in Figure A.6 is replaced by the Newton-iterated input $x^{(j)}(n)$. Therefore, at each Newton iteration, $w(n)$ becomes $w^{(j)}(n)$ and $y(n)$ becomes $y^{(j)}(n)$, but $w(n-1)$ and $w(n-2)$ are still used in computations at each Newton iteration.

Figure A.6: Typical discrete-time filter block
iteration. However, for proper operation of the filter, \( w(n - 1) \) and \( w(n - 2) \) must not be updated until convergence of the Newton iteration. To facilitate this, it is necessary to save the value of \( w^{(j)}(n) \) from one Newton iteration to the next. This is illustrated in Figure A.7 where the extra storage block \( \Delta j \) indicates storage of the previous iterate of \( w^{(j)}(n) \) within the same time step. Figure A.8 on the next page shows the conditions under which \( w(n - 1) \)

\[
\begin{align*}
   & x^{(j)}(n) \quad + \quad w^{(j)}(n) \quad \Delta j \quad Z^{-1} \\
   & \quad \quad -\beta_1 \quad -\beta_2 \quad w(n-1) \quad Z^{-1} \\
   & \quad \quad \quad \quad \quad \alpha_0 \quad \alpha_1 \quad \alpha_2 \quad y^{(j)}(n)
\end{align*}
\]

Figure A.7: Discrete-time filter block permitting Newton iteration

and \( w(n - 2) \) are updated. Here, the simulator has reached convergence during the previous Newton iteration and has advanced to the initial iteration of a new time step. A software flag in \texttt{fREEDA™} indicates this condition to the simulated filter implementation. At this point, the simulated filter will update the values of \( w(n - 1) \) and \( w(n - 2) \) before any computation takes place. The value of \( w^{(j)}(n) \) stored by \( \Delta j \) thus represents the final value of \( w(n) \) at step \( n \), so it is the correct value to store in delay element \( w(n - 1) \) at step \( n + 1 \). The behavior of the software implementing the blocks in Figs. A.7 A.8 at step \( n + 1 \) is given by the following:

1. If the “first iteration” flag is set, then update the stored element values: \( w(n - 2) \leftarrow w(n - 1) \) and \( w(n - 1) \leftarrow w^{(j)}(n) \)
2. Compute \( w^{(j)}(n + 1) \) and \( y^{(j)}(n + 1) \) from \( x^{(j)}(n + 1) \) and the previously stored values \( w(n) \) and \( w(n - 1) \).
3. Update the stored iterate value: \( w^{(j-1)}(n+1) \leftarrow w^{(j)}(n+1) \).

### A.5 The Chebychev Type I Lowpass Filter Model

Continuous-time Chebychev lowpass filters are all-pole filters having a defined level of ripple — rooted in Chebychev polynomials — in the passband and a flat stopband. Such filters may have an even or odd number of poles, and the poles lie on an ellipse about the origin in the left half of the continuous-time frequency domain (i.e. Laplace or s-domain), with the shortest distance on the ellipse to the origin being on the real axis. An even number of poles always occur in conjugate pairs so that a pole pair has a second-order polynomial form in the denominator of its s-domain transfer function. The single odd pole occurs only on the negative real axis. By cascading pairs of poles with the odd singleton pole, an arbitrary order Chebychev Type I filter may be constructed.

Construction of a Chebychev Type I lowpass discrete-time filter begins with the netlist parameters passed to the simulator engine. These parameters specify frequencies corresponding to the s-domain, i.e. frequencies that are not pre-warped (but are immediately pre-warped before any further computations). The parameters must specify the passband and stopband...
edge frequencies, the passband ripple, and the stopband attenuation, from which the number of poles is derived. Figure A.9 shows the parameters that are usually explicitly passed. In Figure A.9, $\varepsilon$ is the allowable passband ripple and $\lambda$ is the stopband flatness parameter. Usually, these are specified in the netlist in negative dB form, and in that case

$$
\varepsilon_{dB} = 10 \log_{10} \left( \frac{1}{1 + \varepsilon^2} \right) \quad (A.40)
$$

$$
\lambda_{dB} = 10 \log_{10} \left( \frac{1}{1 + \lambda^2} \right) \quad , \quad (A.41)
$$

so that

$$
\varepsilon = \sqrt{10^{-\frac{\varepsilon_{dB}}{10}} - 1} \quad (A.42)
$$

$$
\lambda = \sqrt{10^{-\frac{\lambda_{dB}}{10}} - 1} \quad . \quad (A.43)
$$

Also in Figure A.9, $\omega_p$ and $\omega_s$ are the edge frequencies for the passband and stopband corresponding to the points in the magnitude response where $\varepsilon$ and $\lambda$ are defined. For a Chebychev...
Type I filter, $\omega_p$ defines the highest frequency at which the ripple specification is met. Given the nature of Chebychev Type I filters, the number of poles, $N$ is given by

$$N = \left\lfloor \frac{\cosh^{-1} \left( \frac{1}{\varepsilon} \right)}{\cosh^{-1} \left( \frac{1}{\omega_p} \right)} \right\rfloor,$$  \hspace{1cm} (A.44)

where $\cosh^{-1}$ denotes the inverse hyperbolic cosine function, and where the $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ brackets denote that $N$ is to be rounded up to the nearest integer.

Following [235], a continuous-time Chebychev Type I lowpass filter of order $N$ has poles $p_k$ located at the following locations for $k = 0, 1, \ldots, N - 1$,

\begin{align*}
    u_k &= -\sinh(v_0) \left[ \sin \left( \frac{(2k + 1)\pi}{2N} \right) \right] \omega_p \hspace{1cm} (A.45) \\
    v_k &= \cosh(v_0) \left[ \cos \left( \frac{(2k + 1)\pi}{2N} \right) \right] \omega_p \hspace{1cm} (A.46) \\
    p_k &= u_k + jv_k \hspace{1cm} (A.47) \\
    \bar{p}_k &= p_{N-k} = u_k - jv_k \hspace{1cm} (A.48)
\end{align*}

where $\bar{p}_k$ denotes the complex conjugate of pole $p_k$ and the quantity $v_0$ is given by

$$v_0 = \frac{\sinh^{-1}(1/\varepsilon)}{N},$$  \hspace{1cm} (A.49)

with $\sinh^{-1}$ denoting the inverse hyperbolic sine function. When $N$ is odd, the pole at index $k = (N - 1)/2$ is purely real. Thus the transfer function $H(s)$ of an $N$-pole Chebychev Type I lowpass filter is given by

$$H(s) = \begin{cases} 
    A \prod_{k=0}^{N-1} \frac{1}{(s - p_k)(s - \bar{p}_k)} & \text{for } N \text{ even,} \\
    A \prod_{k=0}^{N-1} \frac{1}{(s + p_r)(s - \bar{p}_k)} & \text{for } N \text{ odd.} 
\end{cases} $$ \hspace{1cm} (A.50)

In (A.50), $p_r$ is the magnitude of a pole that occurs on the negative real axis, and $A$ is a constant that sets the filter gain. For unity gain

$$A = \begin{cases} 
    \prod_{k=0}^{N-1} |p_k|^2 & \text{for } N \text{ even,} \\
    p_r \prod_{k=0}^{N-1} |p_k|^2 & \text{for } N \text{ odd.} 
\end{cases} \hspace{1cm} (A.51)$$
Let the transfer function for an arbitrary pole pair be given by

\[ H_k(s) = \frac{1}{a_k s^2 + b_k s + c_k} \]  

(A.52)

Substituting (A.45)-(A.48) into (A.52) leads to

\[ a_k = 1 \]  

(A.53)

\[ b_k = 2\omega_p \sinh(v_0) \left[ \sin \left( \frac{(2k + 1)\pi}{2N} \right) \right] \]  

(A.54)

\[ c_k = \left[ \sinh^2(v_0) \sin^2 \left( \frac{(2k + 1)\pi}{2N} \right) + \cosh^2(v_0) \cos^2 \left( \frac{(2k + 1)\pi}{2N} \right) \right] \omega_p^2 \]  

(A.55)

For the case of a pole singleton on the negative real axis, when \( k = (N - 1)/2 \equiv K \),

\[ H_K(s) = \frac{1}{b_K s + c_K} \]  

(A.56)

where

\[ b_K = 1 \]  

(A.57)

\[ c_K = \omega_p \sinh(v_0) \]  

(A.58)

From this point forward, the task of completing the construction of a Chebychev Type I lowpass filter mirrors that of completing the construction of a Butterworth lowpass filter, but with (A.53)-(A.55) substituted into (A.26) and (A.57)-(A.58) substituted into (A.30).

A.6 The Bandpass Filter Model

Construction of a particular instance of a discrete-time bandpass filter begins with the netlist parameters passed to the simulator engine. These parameters specify frequencies corresponding to the s-domain, i.e. frequencies that are not pre-warped. The parameters may explicitly specify the center frequency and number of poles of the filter; otherwise the passband and stopband edge frequencies shown in Figure A.10 implicitly specify the center frequency and number of poles, which are then pre-warped and used to compute the number of poles, \( N \), by following steps described in [220, 239].

The process of synthesizing a bandpass filter begins with converting the passband frequencies into an equivalent set of frequencies for a lowpass prototype filter [240]. First, the
The parameters ε and λ carry over to the lowpass prototype filter without change. From this point, the synthesis process proceeds through the steps noted in Section A.4 or Section A.5 until
the values of filter coefficients \(a_k, b_k,\) and \(c_k\) are established by (A.53)–(A.55) for Butterworth filters or by (A.57)–(A.58) for Chebychev Type I filters.

Recalling the transfer function for the \(k^{th}\) pole pair of a lowpass filter,

\[
H_k(s) = \frac{1}{a_k s^2 + b_k s + c_k},
\]

the frequency transformation (A.3) is applied to (A.62) yielding the bandpass \(H_k(S)\) response:

\[
H_k(S) = \frac{S^2}{a_k S^4 + b_k S^3 + C_k S^2 + D_k S + E_k},
\]

where \(C_k = 2\omega_0^2 a_k + c_k,\) \(D_k = b_k\omega_0^2,\) and \(E_k = a_k\omega_0^4.\) Next, the BZT \(S \leftrightarrow (z-1)/(z+1)\) is applied to (A.63) yielding

\[
H_k(z) = \frac{1 - 2z^{-2} + z^{-4}}{F_k + G_k z^{-1} + H_k z^{-2} + K_k z^{-3} + M_k z^{-4}},
\]

with \(F_k\ldots M_k\) in Table A.2.

### Table A.2: Transfer function denominator coefficients

<table>
<thead>
<tr>
<th>Canonical coefficient</th>
<th>Lowpass Pole</th>
<th>Lowpass Singleton</th>
</tr>
</thead>
<tbody>
<tr>
<td>(k, N) even (or k \neq K,) (N) odd</td>
<td>(a_k + b_k + C_k + D_k + E_k) (b_k + c_k + b_k\omega_0^2)</td>
<td>(a_k + b_k + C_k - D_k + E_k) (0)</td>
</tr>
<tr>
<td>(k = K,) (N) odd</td>
<td>(6a_k - 2C_k + 6E_k) (b_k - c_k + b_k\omega_0^2)</td>
<td>(-4a_k + 2b_k - 2D_k + 4E_k) (0)</td>
</tr>
<tr>
<td>(K_k)</td>
<td>(-4a_k + 2b_k - 2D_k + 4E_k) (0)</td>
<td>(a_k - b_k + C_k - D_k + E_k) (0)</td>
</tr>
<tr>
<td>(M_k)</td>
<td>(a_k - b_k + C_k - D_k + E_k) (0)</td>
<td></td>
</tr>
</tbody>
</table>
A lowpass pole singleton transformation can be derived similarly with $a_k$ set to zero:

$$H_k(z) = \frac{1 - z^{-2}}{F_k + G_k z^{-1} + H_k z^{-2} + K_k z^{-3} + M_k z^{-4}}, \quad (A.65)$$

with $F_k \ldots M_k$ in Table A.2. Realization of (A.64) and (A.65) in canonical form [237] requires normalization of the $F_k$, and so for purposes of realizing the filter, the canonical form of $H_k(z)$ is given by $H_{ck}(z) = H_k(z)/F_k$ so that

$$H_{ck}(z) = \frac{\alpha_{k0} + \alpha_{k1}z^{-2} + \alpha_{k2}z^{-4}}{1 + \beta_{k1}z^{-1} + \beta_{k2}z^{-2} + \beta_{k3}z^{-3} + \beta_{k4}z^{-4}}. \quad (A.66)$$

The coefficients in (A.66) are given in Table A.3 and are obtained by collecting the scaled terms from (A.64) and (A.65). The $1/F_k$ scaling factors are combined into a discrete-time multiplier coefficient $A_d$, where

$$A_d = \begin{cases} 
\frac{1}{A} \prod_{k=0}^{N} \frac{1}{F_k}, & \text{for } N \text{ even} \\
\frac{1}{A} \prod_{k=0}^{N-1} \frac{1}{F_k}, & \text{for } N \text{ odd} 
\end{cases} \quad (A.67)$$

The discrete-time form of the filter, in a block cascade implementation, is shown in Figure A.11.

$$x(n) \xrightarrow{A_d} H_{c1}(z) \xrightarrow{} H_{c2}(z) \xrightarrow{\cdots} H_{ck}(z) \xrightarrow{\cdots} H_{cN}(z) \xrightarrow{} y(n)$$

Figure A.11: Discrete-time bandpass filter in cascade form.
Following the development given in [237], the canonical form of a filter block that implements (A.66) is the form that requires the fewest storage elements to implement. This form is created by introducing an intermediate function $W_k(z)$. If $X_k(z)$ and $Y_k(z)$ are the input and output, respectively, of the $k$th cascaded block of the filter, then the intermediate function $W_k(z)$ is introduced by

$$
H_{ck}(z) = \frac{Y_k(z)}{X_k(z)} = \frac{Y_k(z)}{W_k(z)} \cdot \frac{W_k(z)}{X_k(z)} ,
$$

where, by substitution of the numerators and denominators of (A.66), the following relationships are obtained:

$$
\frac{Y_k(z)}{W_k(z)} = \alpha_{k0} + \alpha_{k1}z^{-2} + \alpha_{k2}z^{-4}
$$

(A.69)

$$
\frac{W_k(z)}{X_k(z)} = \frac{1}{1 + \sum_{i=1}^{4} \beta_{ki}z^{-i}} .
$$

(A.70)

Rearranging (A.69) and (A.70), the final $z$-domain form of the equations implementing the canonical form of the filter block are given by

$$
Y_k(z) = [\alpha_{k0} + \alpha_{k1}z^{-2} + \alpha_{k2}z^{-4}] W_k(z)
$$

(A.71)

$$
W_k(z) = X_k(z) - \sum_{i=1}^{4} \beta_{ki}z^{-i}W_k(z) ,
$$

(A.72)
and, after inversion from the z-domain to the time-domain, the final form of the block filter equations are obtained:

\[ y_k(n) = \alpha_{k0} w_k(n) + \alpha_{k1} w_k(n - 2) + \alpha_{k2} w_k(n - 4) \]  
\[ w_k(n) = x_k(n) - \sum_{i=1}^{4} \beta_{ki} w_k(n - i) . \]  

(A.73)  

(A.74)

In (A.73) and (A.74) the index variable \( n \) refers to the discretized time increment, i.e. \( t = nT \). In order to facilitate Newton iteration at a given time step, it is necessary to subject the present values of the variables \( x_k(n), w_k(n), \) and \( y_k(n) \) to Newton iteration. Letting a superscripted \((j)\) denote the iterates, then, (A.73) and (A.74) are modified as follows:

\[ y_k^{(j)}(n) = \alpha_{k0} w_k^{(j)}(n) + \alpha_{k1} w_k^{(j)}(n - 2) + \alpha_{k2} w_k^{(j)}(n - 4) \]  
\[ w_k^{(j)}(n) = x_k^{(j)}(n) - \sum_{i=1}^{4} \beta_{ki} w_k^{(j)}(n - i) . \]  

(A.75)  

(A.76)

The block diagram of the \( k \)th cascaded block of the filter, reflecting (A.75) and (A.76) and in canonical form, is shown in Figure A.12 on the following page. One new element in Figure A.12 (and not present in [237]) is the storage element \( \Delta j \) required to store values of the Newton iterates of \( w_k^{(j)}(n) \) within a time step. Upon convergence at a particular time step the \( Z^{-1} \) storage elements holding \( w_k(n) \) are updated at the initial iterate of the next time step.

One drawback to the discrete-time filter implementation concerns entries that must be made in the Jacobian matrix to facilitate Newton iteration \([156]\). In conventional implementations of device models, Jacobian entries \([14, 15]\) are determined through the evaluation of equations resulting from the analytic partial differentiation of continuous-time constitutive equations. Given the inherently discrete-time nature of the filter described here, that method is not available. Instead, the Jacobian entries are formed from a ratio of the difference of the current and previous output iterates to the current and previous input iterates — effectively an implementation of the Secant Method. This has the effect of reducing the rate of convergence of the Newton iterations from quadratic (i.e. error reduced at a quadratic rate) to superlinear with a resulting rate of \((1 + \sqrt{5})/2 \approx 1.62\) \([241]\).
Figure A.12: Typical discrete-time filter block in canonical form.
A.7 A 1.7 GHz Coaxial Resonator Filter

A discrete-time Butterworth bandpass filter was implemented in fREEDA™ [234] using the methodology of the previous section. Validation used a 5-section (having a 5th order lowpass prototype) filter with coaxial resonators having a maximally-flat transfer characteristic. The filter has a $-3$ dB passband of 15.6 MHz, a center frequency of 1.7 GHz (a fractional bandwidth of 0.9%), and an insertion loss of 1.2 dB. The passband and stopband edge frequencies and insertion loss were extracted from the $S_{21}$ magnitude of the measured filter. These were used as model netlist parameters.

The first step in validation is verifying the frequency response of the simulated filter. This was accomplished by running a series of transient simulations with sinusoidal sources at 0 dBm power levels, then transforming the results to the frequency domain and collecting the filter response for the series at the excitation frequencies. Simulations were run for a time period long enough to assure that initial transients had died out, and the FFT was taken on 8192 points of ‘trailing data.’ Figure A.13 compares the results achieved by fREEDA™ with analytic frequency response calculations and the measured $S_{21}$ magnitude data for the coaxial filter. Agreement is seen in the vicinity of the passband and down the stopband edges to about

![Figure A.13: Simulated (referenced to 0 dBm) and measured frequency response results for a 5-section coaxial filter.](image-url)
Next, the filter was excited by a linear FM chirp signal with a 200 MHz chirp range centered about 1.7 GHz, and the FFT of the result was taken. The power level of chirp source was normalized such that the flat portion of the chirp was at 0 dBm. The linear FM chirp signal is the type of waveform used in radar, a system that is difficult to characterize using harmonic balance and traditional circuit simulators. Figure A.14 shows that excellent agreement is seen between the response of the simulated filter to the simulated chirp signal and the measured $S_{21}$ magnitude data for the coaxial filter. The filter order and fractional bandwidth are noted in Figure A.1 where it is seen that the filter characteristics cannot be simulated using a conventional transient circuit simulation technique. Furthermore the high dynamic range of fREEDA™ enables the filter response to be modeled over a very wide dynamic range.

![Graph showing response of simulated filter to a Linear FM chirp signal.](image-url)

Figure A.14: Response of simulated filter to a Linear FM chirp signal.
A.8 A 9 GHz Parallel-Coupled Line Filter

Before the work in Section A.7 was completed, an earlier effort to validate the discrete time Butterworth behavioral filter model in freeda™ was conducted using a filter that was fabricated to be part of a radar transmission chain. The radar transmission system was simulated in freeda™, and the FFT of the transient simulation of the radar transmission system was compared to laboratory measurements of a chain containing the fabricated filter. A block diagram of the laboratory setup is shown in Figure A.15. The complete transmission system under test consisted of a Filtronics LMA-411 X-band MMIC (assembled onto a carrier with end launch SMA connectors), used as a driver amplifier, followed by the fabricated bandpass filter.

Fabricated Bandpass Filter

A 9 GHz edge coupled bandpass filter with a 3 dB bandwidth target of 270 MHz (i.e. 3% fractional bandwidth) was designed and fabricated using 1 μm thick microstrips of sputtered copper on an 40 mil thick alumina substrate with a permittivity of 9.8. The design procedure for such filters can be found in [242, 243]. The filter structure was patterned using a bi-layer liftoff process and connections were made using end launch SMA connectors. The band stop frequencies were specified such that 10 poles would be required in the bandpass implementation, or 5 poles in the lowpass prototype form from which the bandpass filter would result. Given normal process parameters in the fabrication of the filter, it was expected that there would be 4-5 dB of insertion loss. The layout of the filter is shown in Figure A.16.

Unfortunately, the fabricated filter had 14 dB of insertion loss due to a non-optimal sputtering process where the resistivity of the copper obtained from the process was 5 to 6
times that of base metal. Hence the effective thickness was approximately 0.15 to 0.2 \( \mu \text{m} \). Such a lossy filter has limited practical utility, but it is still useful for demonstrating the utility of the discrete-time Butterworth filter synthesis capability in \textit{fREEDA™}. However, the greater insertion loss does have consequences for the filter that \textit{fREEDA™} would synthesize: The depth of the filter skirts were effectively far more shallow than would be necessary for a 10\textsuperscript{th} order filter, thus reducing the effective filter order. Given the measured frequency specifications, \textit{fREEDA™} synthesized a 4\textsuperscript{th} order bandpass filter. One positive note is that the fabricated filter had a 250 MHz 3 dB bandwidth (i.e. 2.78\% fractional bandwidth).

### Comparison of Simulated and Measured Results

The transmission system was simulated in \textit{fREEDA™} with a narrowband pulsed linear FM chirp source spanning 80 MHz with a center frequency of 8.875 GHz (with the center frequency chosen to lie on the lower edge of the filter’s passband). Laboratory measurements with an
equivalent source were conducted. The simulated results were transformed to the frequency domain using the FFT, and the measured data were thresholded to remove points near the noise floor. Figure A.17 shows that good agreement is seen between measured and simulated results except in areas dominated by the noise floor of the laboratory equipment. Note that the upward tilt in the passband of the response is to be expected, since the measurement was taken around the left edge of the passband over a chirp range of 80 MHz, while the passband of the entire filter is 250 MHz. (The chirp source was limited to an 80 MHz bandwidth.) In addition, what appears to be asymmetry in the skirts is the result of two different attenuating phenomena: The left-side skirt is the result of the filter characteristics suppressing some of the chirp input, while the right side skirt is passed through the filter with little suppression. The two small sidelobes in the measured results are artifacts of the signal generating equipment and cannot be attributed to the fabricated filter.

Figure A.17: Measured and simulated results near the lower edge of the passband.
A.9 Concluding Remarks

A simulation model for maximally-flat and Chebychev discrete-time filters suitable for transient circuit simulation was presented. The models are effectively implemented in the $z$-domain so that a much higher order filter can be modeled than that achievable with a more conventional implementation based on a pole-zero $s$-domain transfer function. Owing to the robustness of the fREEDA™ transient simulator engine, Newton iteration convergence is achieved using a fixed time step, making the simulation of a discrete-time behavioral filter model possible. In order to deploy discrete-time filter models in other transient simulator environments, it is necessary to use a fixed time step and avoid the default adaptive time stepping of the Spice engine [11].
Appendix B

Uncorrelated Phase Multitone Signal Generator

B.1 Problem Statement

The investigation of the response of amplifiers to uncorrelated-phase multitone signals requires a signal source where the phase of each tone is uncorrelated from the others. Sophisticated signal-generating equipment such as the Agilent ESG and PSG product lines are capable of producing multitones at uniformly-spaced frequency intervals with random initial phases, but since the generated signal is ultimately in the form of a time-domain sequence, the tones are necessarily correlated and the phase relationships between tones remain fixed for the duration of the generated signal. Thus, measurements of the response of a nonlinear amplifier to this type of stimulus give valid results only for the coherent phase regime chosen for the input tones. In order to approximate the results for a multitone uncorrelated phase source, Monte Carlo techniques must be used where the phases of the multitone signal are considered a jointly Gaussian random variable. A series of approximately 30 measurements must be taken — each with a different source phase regime — and the average of the ensemble of outputs would then approximate the response of the amplifier to an uncorrelated phase multitone source.

For this work, it was decided to create an uncorrelated phase multitone source based on combining the outputs of a plurality of voltage-controlled oscillators (VCO), with each
VCO having frequency-tuning networks independent of the others. Thus the signal-generating equipment used for the work would have inherently uncorrelated phase relationships, assuming that injection-pulling was not an issue. In order to assure that the developed uncorrelated phase multitone signal generator did not exhibit undesirable features during applications for this research, the task of creating the generator consisted of its design and characterization, with the characterization data serving to identify areas of both valid and undesirable behavior during operation. During research work, measurements would be limited to the area of valid behavior.

Design Requirements

The basic requirements for the uncorrelated phase signal generator were:

- Hardware-configurable number of tones (VCOs), but with a maximum greater than 12
- Minimum degree of injection-pulling when applied to research situations
- VCO outputs tunable in a frequency range suited to previously modeled amplifiers
- Hardware-configurable attenuation permitting a wide range combined output power levels

Source stability of the kind guaranteed by placing the VCOs in a Phase-Locked Loop (PLL) was not viewed as a requirement. Instead, the stationarity of the source frequencies would be characterized after construction with the expectation that VCO output frequencies would stabilize to within a narrow range.

Design Details

Each tone output of the multitone signal generator was comprised of a Mini-Circuits ZX95-535 VCO and the supporting circuitry shown in Figure B.1 on page 329. This circuitry appears three times on each assembly as indicated by the blocks \( OSC_1 - OSC_3 \) in Figure B.2 on page 330. A basic 3-tone signal generator apparatus was created by combining the outputs of three of the VCOs through attenuators \( A_1 - A_3 \) and combiner \( S_1 \) as shown in Figure B.2 on page 330. Attenuator \( A_4 \) provides a measure of overall power level control out of the assembly. A photo of a typical three oscillator apparatus is given in Figure B.3 on page 331 and a bill of mate-
MATERIALS FOR A THREE OSCILLATOR ASSEMBLY IS GIVEN IN TABLE B.1. EXCEPT FOR ATTENUATORS $A_1$–$A_4$ AND FEEDTHROUGH ADAPTERS $F_1$–$F_4$, ALL OF THE PARTS ARE FASTENED TO THE UNDERLYING PERF BOARD WITH SECTIONS OF INSULATED 18 GAUGE WIRE. THIS SCHEME PERMITS LOOSENING THE VCO FROM ITS NORMALLY TIGHTLY-FASTENED STATE AND PERMITS SUBSTITUTION OF ALTERNATE ATTENUATORS ON THE VCO OUTPUTS. IN ADDITION, THE PRESENCE OF THE SMA MALE-MALE FEEDTHROUGH ADAPTERS $F_1$–$F_3$ ENABLE EASY REMOVAL OF AN OSCILLATOR FROM THE SIGNAL CHAIN; THIS WAS USEFUL FOR INSTALLING A TERMINATOR ON A COMBINER INPUT, WHICH FACILITATED THE 14-TONE NARROWBAND IM MEASUREMENTS IN SECTION 5.3.2.

### Table B.1: Bill of Materials for 3-Oscillator Assembly in Figure B.2

<table>
<thead>
<tr>
<th>Designation</th>
<th>Qty</th>
<th>Manufacturer</th>
<th>Mfr P/N</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O_1$</td>
<td>3</td>
<td>Mini-Circuits</td>
<td>ZX95-535</td>
<td>VCO, 300-520 MHz, tunable</td>
</tr>
<tr>
<td>$L_1$</td>
<td>3</td>
<td>API-Delevan</td>
<td>1025-44K</td>
<td>Inductor, 10 uH, 144 mA</td>
</tr>
<tr>
<td>$C_1$</td>
<td>3</td>
<td>Panasonic</td>
<td>EEU-FC1H100</td>
<td>Cap., Elec. Alum., 10 uF</td>
</tr>
<tr>
<td>$P_1$</td>
<td>3</td>
<td>CTS</td>
<td>026TB32R102B1A1</td>
<td>Pot., 1k $\Omega$, 1 turn, 5 watt</td>
</tr>
<tr>
<td>$P_1$ (alt)</td>
<td>3</td>
<td>Vishay-Spectrol</td>
<td>534-1-1-102</td>
<td>Potentiometer, 1k $\Omega$, 10 turn, 2 watt</td>
</tr>
<tr>
<td>$P_2$</td>
<td>3</td>
<td>CTS</td>
<td>026TB32R101B1A1</td>
<td>Pot., 100 $\Omega$, 1 turn, 5 watt</td>
</tr>
<tr>
<td>$P_2$ (alt)</td>
<td>3</td>
<td>Vishay-Spectrol</td>
<td>534-1-1-101</td>
<td>Potentiometer, 100 $\Omega$, 10 turn, 2 watt</td>
</tr>
<tr>
<td>$P_3$</td>
<td>3</td>
<td>CTS</td>
<td>026TB32R5A0B1A1</td>
<td>Pot., 5 $\Omega$, 1 turn, 5 watt</td>
</tr>
<tr>
<td>$R_{pu}$</td>
<td>3</td>
<td>Phoenix Passive</td>
<td>5073NW47R00J12AFX</td>
<td>Resistor, Metal film, 47 $\Omega$, 1 watt</td>
</tr>
<tr>
<td>$R_{pd}$</td>
<td>3</td>
<td>Phoenix Passive</td>
<td>5073NW2K200J12AFX</td>
<td>Resistor, Metal film, 2.2k $\Omega$, 1 watt</td>
</tr>
<tr>
<td>9</td>
<td>Keystone Electronics</td>
<td>8555</td>
<td>Knob, 6.25 mm shaft diameter</td>
<td></td>
</tr>
<tr>
<td>$TW\ P_1$–</td>
<td>3</td>
<td>L-Com</td>
<td>BC25</td>
<td>tinned 6 in. twisted</td>
</tr>
</tbody>
</table>

Continued on next page
Table B.1 (continued)

<table>
<thead>
<tr>
<th>Designation</th>
<th>Qty</th>
<th>Manufacturer</th>
<th>Mfr P/N</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TWP3</td>
<td>3</td>
<td>L-Com</td>
<td>BA832</td>
<td>wire pair with male BNC</td>
</tr>
<tr>
<td>T1</td>
<td>1</td>
<td>L-Com</td>
<td>BA250</td>
<td>BNC-tee, 3 female</td>
</tr>
<tr>
<td>T2</td>
<td>1</td>
<td>L-Com</td>
<td></td>
<td>BNC-tee, 1 male 2 female</td>
</tr>
<tr>
<td>A1−A3</td>
<td>3</td>
<td>Mini-Circuits</td>
<td>BW-S20W2+</td>
<td>Attenuator, SMA, 20 dB, 2 watt</td>
</tr>
<tr>
<td>F1−F3</td>
<td>3</td>
<td>Mini-Circuits</td>
<td>SM-SM50</td>
<td>Feedthrough, SMA male-male, 50 Ω</td>
</tr>
<tr>
<td>S1</td>
<td>1</td>
<td>Mini-Circuits</td>
<td>ZFSC-3-4-S</td>
<td>3:1 Combiner, SMA, 1-520 MHz</td>
</tr>
<tr>
<td>F4</td>
<td>1</td>
<td>Pasternack</td>
<td>PE-9068</td>
<td>Right angle, SMA male-female</td>
</tr>
<tr>
<td>A4</td>
<td>1</td>
<td>Mini-Circuits</td>
<td>BW-S10W2+</td>
<td>Attenuator, SMA, 10 dB, 2 watt</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Radio Shack</td>
<td>276-1396</td>
<td>Perf board, 6 in. by 8 in.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Radio Shack</td>
<td></td>
<td>Hookup wire, 18 gauge</td>
</tr>
</tbody>
</table>

Characterization: Injection Pulling

A test for observable effects of injection pulling was conducted on the first 3-oscillator assembly built. This assembly (unit No. 1) used the CTS 100 Ω potentiometer for part designation P2 as shown in Table B.1. For the purposes of this test, oscillator OSC3 shown in Figure B.2 was disconnected by removing F3, and connecting A3 and a 50 Ω terminator to power combiner S port 3, thus leaving only OSC1 and OSC2 connected. In addition, OSC1 and OSC2 were disconnected from the 12 V supply indicated in Figure B.2 and instead connected independently to two separate lead acid lamp batteries (Interstate Battery model DRY1425) nominally designated as 12 volt batteries (with 4200 mAh capacity), but which supplied 12.9 volts under the lightly loaded conditions here. Thus any signal coupling path through the power supply connections was eliminated, and so under these conditions, the only possible way
in which one oscillator could affect the other would be through injection pulling induced through the oscillator’s output port (i.e. through the $S_{12}$ signal path).

Two injection pulling experiments were conducted. In the first experiment, attenuators $A_1$ and $A_2$ (see Fig B.2) were set to 10 dB. In the second, they were set to 20 dB. In both experiments, one oscillator was held at a nominal oscillating frequency of 450 MHz while the other was tuned to a series of frequencies incrementally higher than 450 MHz. The minimum frequency spacing attempted was 5 kHz. At this spacing, visible and quasi-stable injection pulling in the form of spectrally-rich sideband beats to the two nominal oscillating frequencies could be observed. Attempts to reduce the spacing below 5 kHz resulted in the merging of the two oscillators into a single apparent spectrum with drastically reduced sideband character. For spacings greater than 5 kHz, it was possible to observe the fundamental outputs of the two oscillators along with intermodulation-like spurs at frequency increments equal to the frequency spacing of the fundamentals. The spurs were symmetrical in amplitude about the location of the two fundamentals and, for minimum spacing of the fundamentals, occurred at amplitudes equal to the attenuator values used for $A_1$ and $A_2$ in Figure B.2 on page 330. As the frequency spacing was increased, the amplitudes of the spurs fell off, dropping below the approximate $-95$ dBm measurement floor at the highest spacing of 1 MHz.

Figure B.4 on page 334 shows the results of the injection pulling experiment. As can be seen in Figure B.4a on page 334 when only 10 dB of attenuation is used between the oscillators and the combiner, there are measurable spurs occurring at 1, 2, and 3 times the spacing of the fundamentals, and that for low frequency spacings, the power level difference between the fundamental and the first spur is equal to the value of attenuators $A_1$ and $A_2$. On the other hand, when a 20 dB attenuator is used for $A_1$ and $A_2$ in the oscillator assembly, spurs are visible only at 1 and 2 times the spacing of the fundamentals, and the spurs fall off rapidly with increasing power levels. Note that the first injection pulling spur is at roughly $-90$ dBm for a spacing of 750 kHz and is below the noise floor at 1 MHz. From this result, it was concluded that a 1 MHz nominal minimum spacing between oscillator tones should be maintained to eliminate measurable injection pulling when the assembly is deployed as an uncorrelated-phase multitone signal generator.
Figure B.1: Schematic of circuitry supporting one VCO.
Figure B.2: Block Diagram of an assembly of three oscillators.
Figure B.3: Photo of a 3 oscillator assembly.
Characterization: Frequency Stability

Since the oscillators on each assembly are not stabilized within a phase-lock loop, the absolute frequency at which an oscillator stabilizes cannot be predicted, but with adequate control of all of the variables other than tuning voltage, stability with a small degree of non-stationarity will be obtained. It was found in this work that it is not possible to expect frequency stability when an assembly was operating in an open-air laboratory environment (room 410 of the Monteith Research Center at NC State University). The source of the instability was suspected to be turbulent air flows in the laboratory due to ceiling-mounted air conditioning ducts. To investigate this hypothesis, each assembly was housed in a crude oven-like enclosure – a sealed cardboard shipping box with a lid formed of 1/4 millimeter thick plastic sheeting. The enclosure had no external temperature controls, but with the outside ambient airflows eliminated, operating assemblies housed in such boxes could be expected to reach thermal equilibrium.

The following sections show the behavioral statistics of running each assembly for 30 minutes outside of any enclosure, i.e. in an open air environment, to running the same assembly for 30 minutes inside of the sealed cardboard box enclosure. For each assembly, the nominal frequency of oscillator 1 was set to 449 MHz, oscillator 2 to 450 MHz, and oscillator 3 to 451 MHz. The oscillators on each assembly were tuned to these nominal frequencies (with a spectrum analyzer used to monitor the frequencies), then allowed to dwell for 3 minutes before a 30 minute automated measurement suite was run. 90 automated measurements were taken at a rate of 3 measurements per minute. For each assembly, the following information was accumulated for each oscillator:

- Histogram showing counts of measured frequencies
- Running standard deviations of amplitude and frequency vs. measurement number
- Measured and running mean frequency
- Measured and running mean amplitude

The data on the following pages is organized such that a side-by-side comparison of data from the same oscillator assembly outside the box, given by subfigure (a) in each case, can be easily made with data under operation in the sealed cardboard box, given by subfigure (b). In the histograms of the following pages, it will be noted generally that none of the measured frequencies occurs at the nominally tuned frequency; this is because of the negative frequency per unit temperature coefficient \[ \text{245} \] which results in a downward drift in the frequency of the
oscillator with increasing temperature over the initial dwell time.

The most important data in all of these figures is the behavior of the frequency running standard deviation, shown as $\sigma_F$ in the figures to follow. This is a figure of merit that furnishes the deviation in frequency considering the current sample and all previous samples. From the point power is first applied and for several minutes, the running frequency standard deviation grows as the oscillator drifts. When thermal equilibrium is achieved, the oscillating frequency is expected to stabilize, and the running standard deviation will undergo a change of derivative value and stabilize to its equilibrium value. However, it can be seen that when the oscillator assemblies operate in an open-air environment that this does not generally happen, and instead the running standard deviation in these cases grows without bound in the worst cases, or stagnates (reaches a maxima) without a change in derivative toward a stable value in the best cases. When the assemblies are placed in the cardboard boxes, however, all of them either undergo the change in derivative value or stabilize at a maxima. In all cases, though, stabilized deviations in oscillating frequency is the desired — and achieved — result when the assemblies are housed in cardboard boxes.

**Concluding Remarks**

This appendix has presented the development and characterization of a multitone uncorrelated phase signal generator comprised of 15 discrete voltage-controlled oscillators. The Bill of Materials for creating a board holding 3 of the oscillators was furnished, and it was seen that despite the lack of phase-locking, the oscillators could be made to assume stable oscillating frequencies by controlling the ambient environment in which the oscillators operated and allowing a sufficient period of ambient dwell time. An injection-pulling experiment was performed where it was seen that when oscillator outputs are kept at least 1 MHz apart, then the effects of injection pulling are minimal.
Figure B.4: Results of two injection pulling experiments.
Three-Tone Assembly No. 1 Data Collated
Figure B.5: Histogram of oscillator frequency locations for Assembly 1.

(a): Open-air environment.  
(b): In sealed cardboard enclosure.
Figure B.6: Amplitude and Frequency Running Standard Deviation for Assembly 1.

(a): Open-air environment.

(b): In sealed cardboard enclosure.
Figure B.7: Instantaneous and Running Average Amplitude for Assembly 1.
Figure B.8: Instantaneous and Running Average Frequency for Assembly 1.

(a): Open-air environment.

(b): In sealed cardboard enclosure.
Three-Tone Assembly No. 2 Data Collated
(a): Open-air environment.

(b): In sealed cardboard enclosure.

Figure B.9: Histogram of oscillator frequency locations for Assembly 2.
Figure B.10: Amplitude and Frequency Running Standard Deviation for Assembly 2.

(a): Open-air environment.

(b): In sealed cardboard enclosure.
Figure B.11: Instantaneous and Running Average Amplitude for Assembly 2.
Figure B.12: Instantaneous and Running Average Frequency for Assembly 2.
Three-Tone Assembly No. 3 Data Collated
Figure B.13: Histogram of oscillator frequency locations for Assembly 3.
Figure B.14: Amplitude and Frequency Running Standard Deviation for Assembly 3.

(a): Open-air environment.

(b): In sealed cardboard enclosure.
Figure B.15: Instantaneous and Running Average Amplitude for Assembly 3.

(a): Open-air environment.

(b): In sealed cardboard enclosure.
(a): Open-air environment.

(b): In sealed cardboard enclosure.

Figure B.16: Instantaneous and Running Average Frequency for Assembly 3.
Three-Tone Assembly No. 4 Data Collated
Assembly # 4, Oscillator # 1; Peak Frequencies for 90 samples

Assembly # 4, Oscillator # 2; Peak Frequencies for 90 samples

Assembly # 4, Oscillator # 3; Peak Frequencies for 90 samples

(a): Open-air environment.

Figure B.17: Histogram of oscillator frequency locations for Assembly 4.

(b): In sealed cardboard enclosure.
(a): Open-air environment.  
(b): In sealed cardboard enclosure. 

Figure B.18: Amplitude and Frequency Running Standard Deviation for Assembly 4.
Figure B.19: Instantaneous and Running Average Amplitude for Assembly 4.

(a): Open-air environment.

(b): In sealed cardboard enclosure.
Figure B.20: Instantaneous and Running Average Frequency for Assembly 4.
Three-Tone Assembly No. 5 Data Collated
Assembly # 5, Oscillator # 1; Peak Frequencies for 90 samples

Assembly # 5, Oscillator # 2; Peak Frequencies for 90 samples

Assembly # 5, Oscillator # 3; Peak Frequencies for 90 samples

(a): Open-air environment.

(b): In sealed cardboard enclosure.

Figure B.21: Histogram of oscillator frequency locations for Assembly 5.
(a): Open-air environment.

(b): In sealed cardboard enclosure.

Figure B.22: Amplitude and Frequency Running Standard Deviation for Assembly 5.
Figure B.23: Instantaneous and Running Average Amplitude for Assembly 5.

(a): Open-air environment.

(b): In sealed cardboard enclosure.
Figure B.24: Instantaneous and Running Average Frequency for Assembly 5.

(a): Open-air environment.

(b): In sealed cardboard enclosure.
Three-Tone Assembly No. 6 Data Collated
Figure B.25: Histogram of oscillator frequency locations for Assembly 6.

(a): Open-air environment.

(b): In sealed cardboard enclosure.
(a): Open-air environment.

(b): In sealed cardboard enclosure.

Figure B.26: Amplitude and Frequency Running Standard Deviation for Assembly 6.
Assembly # 6, Oscillator # 1

Assembly # 6, Oscillator # 2

Assembly # 6, Oscillator # 3

(a): Open-air environment.

(b): In sealed cardboard enclosure.

Figure B.27: Instantaneous and Running Average Amplitude for Assembly 6.
(a): Open-air environment.

(b): In sealed cardboard enclosure.

Figure B.28: Instantaneous and Running Average Frequency for Assembly 6.
Three-Tone Assembly No. 7 Data Collated
Assembly # 7, Oscillator # 1; Peak Frequencies for 90 samples

Assembly # 7, Oscillator # 2; Peak Frequencies for 90 samples

Assembly # 7, Oscillator # 3; Peak Frequencies for 90 samples

(a): Open-air environment.
(b): In sealed cardboard enclosure.

Figure B.29: Histogram of oscillator frequency locations for Assembly 7.
(a): Open-air environment.

(b): In sealed cardboard enclosure.

Figure B.30: Amplitude and Frequency Running Standard Deviation for Assembly 7.
Figure B.31: Instantaneous and Running Average Amplitude for Assembly 7.

(a): Open-air environment.

(b): In sealed cardboard enclosure.
Figure B.32: Instantaneous and Running Average Frequency for Assembly 7.

(a): Open-air environment.

(b): In sealed cardboard enclosure.
Appendix C

Mathematical Foundations for AOM

C.1 Preface

This appendix includes the content (with corrections) of the article "Mathematical Foundations of Frequency-Domain Modeling of Nonlinear Circuits using the Arithmetic Operator Method," which was published in the International Journal of RF and Microwave Computer-Aided Engineering, volume 13, number 6, November 2003. The publication reported on a prototype behavioral modeling implementation of AOM in Matlab® that was based closely upon algorithms in the circuit simulation implementation reported by Chang [90]. The corrections include:

1. Removal of a citation to a paper by Gard [155] as a prior user of AOM. Gard did not use AOM in his work [246].
2. The stamps of (C.20) on page 380 apply to only to single-sided spectra. See Section 3.7.1 on page 127 for the double-sided stamps.
3. \( T_w \) in (C.49) on page 396 is not formed as described in the journal paper, but is a valid approach requiring further research. See Section 3.9.7 on page 153 for the details on how \( W \) is computed. If \( T_w \) is needed, it is created via \( T_w = T[W] \).
4. Appendix A (on page 488 of the journal paper) incorrectly discussed linear phasor analysis as using the real part of \( X(\omega) \) for \( x(t) \). This is corrected here in Section C.8 on page 397 and the correction appears in blue ink.
5. Appendix A (on page 488 of the journal paper) was missing closing right braces in
(A.4). This has been corrected in (C.53) on page 398 and the correction appears in blue ink.

In addition to the corrections, the journal paper was intended to be generalized, but was unfortunately limited to mixing examples with second-order frequency sets — i.e. second-order nonlinear processes. The introductory content in Section 3.3 on page 44 of this dissertation gives an overview of AOM using a second-order nonlinearity, but without claims of being comprehensive. The comprehensive explanation of the growth of spectral content with nonlinear order (up to the maximal order or the order of spectral truncation) can be found in Section 3.4 on page 64. Also, since the journal paper content here appears as an appendix in an overall dissertation, the sections of the journal paper that were termed appendices have been given the moniker Postscript and appear as sections after the main content. Finally, differing from the main body of this dissertation, in this appendix the frequency variable is $\omega$ (rather than $f$), where $\omega = 2\pi f$.

C.2 Introduction

RF and microwave engineering is concerned with both the spectral representation of signals and their time-domain representation or waveforms. The preference by microwave engineers for the frequency-domain perspective is principally the result of two characteristics. One of these is that many RF and microwave components are linear so that in the frequency domain simple arithmetic can be used to combine signals and otherwise manipulate the interaction of signals with the linear circuitry. The second characteristic that favors use of the spectral representation is that conventional communication and radar systems are inherently narrow band with the spectrum of a signal occupying a bandwidth that is a small percentage of the center frequency of the signal. Thus a communication signal appears as a sinusoidal signal that slowly varies in amplitude and phase with distortion and mixing products appearing as similar slowly-varying quantities. Thus it is of particular importance for the RF and microwave designer to understand the spectral content of the signal and the interaction of the signal with the nonlinear circuitry as well as with the linear circuitry.

Communications and radar technology has evolved with narrow band spectrum assignments and so it is critical that circuits are designed to have well defined spectral responses, with limited spectral content outside the allocated information-carrying frequency band. Any
additional energy in neighboring channels interferes with the proper operation of other wireless links. Unfortunately it is extremely difficult to estimate low-level interference using transient circuit analysis as is used with digital and broadband analog circuits. Additionally it is particularly difficult to characterize distortion by examining the time-domain waveform although the instantaneous time-domain behavior is, fundamentally, the source of distortion. Fortunately, numerical techniques have been developed that enable the response of a narrow band signal in a nonlinear circuit to be calculated with the necessary precision. The most commonly used technique is referred to as Harmonic Balance wherein the steady-state response of a signal described by a discrete spectrum is found. In the Harmonic Balance method, a circuit is partitioned into linear and nonlinear parts with the characteristics of linear elements handled efficiently in the frequency domain. The nonlinear elements are described in the time-domain and the interface to the discrete spectral representation of signals is achieved using Fourier transforms. Both the quality of the time-domain input and the Fourier transform method may have a dramatic impact upon the efficiency of harmonic balance methods, and the numerical error introduced through repeated Fourier transforms as the methods iterate ultimately limits the dynamic range available for solutions. Spectral-domain methods operating entirely in the frequency domain also exist and here an instantaneous representation of the nonlinear characteristics is utilized as this has representations in both the time and frequency domains. In essence, in spectral-domain methods, a transform is utilized to relate the time- and frequency-domain device characteristics. In the Spectral Balance method, as with Harmonic Balance, the coefficients of the discrete spectral representation, generally phasors, are iteratively updated to arrive at a steady-state response. However, Spectral Balance methods do not require the use of Fourier transforms within the iterative process as do Harmonic Balance methods. These techniques have served the design community well although there are restrictions on the types of signals that can be considered and design insight is lost in purely numerical solutions. This paper addresses this problem by formally introducing a type of mathematics called the Arithmetic Operator Method (AOM) that manipulates signals with discrete spectra by operating entirely in the frequency domain. In essence AOM is the formal mathematical description of the transformation of any analytic functional operation on waveforms rendering them as operations on discrete frequency spectra. This is achieved by noting that any analytic function (as defined in [247]) can be implemented using the four basic arithmetic operations of addition, subtraction, multiplication and division. AOM has versions of these arithmetic operators for operating on discrete spectra
and yielding results that are also discrete spectra. The procedure is akin to convolution which can be illustrated for the mixer shown in Figure C.1 (where the mixer is an ideal multiplier). The inputs to the mixer are the waveforms $x(t)$ and $z(t)$, and $y(t)$ is the output waveform. It is well known that if $x(t)$, $y(t)$, and $z(t)$ are described by a set of phasors restricted to having a uniformly spaced spectral content, then the spectral representation of $y(t)$ is the circular convolution of the spectral representations of $x(t)$ and $z(t)$. AOM eliminates the uniformly spaced spectral restriction of circular convolution. Furthermore, if the spectral representations of $y(t)$ and $z(t)$ are known, then AOM enables the spectral representation of $x(t) = y(t)/z(t)$ to be determined. Thus AOM has application to system identification and here there is no convolution direct analog. Oppenheim [248] referred to a similar concept and called the procedure deconvolution.

The purpose of this paper is to present and expand on AOM for manipulating discrete spectra in linear and nonlinear systems. In earlier publications [90, 92] the concept of AOM was introduced and applied to Spectral Balance analysis. It has been successfully used to characterize non-linear microwave electronic circuits by Carvalho and Pedro [97, 99]. In this paper a tutorial exposition is presented with a mathematically rigorous development for the first time. Throughout, a three-tone example is used to illustrate AOM and this serves to clearly distinguish the method from circular convolution.

C.3 Mathematical Foundation of AOM

AOM is a technique for implementing operations on one or more signals by manipulating their spectra. The mathematical basis is the Generalized Superposition principle described by
Oppenheim in [249, 250] and Chapter 10 of [248]. Superposition can be used only with Linear Time-Invariant (LTI) systems and these are easily modeled using a linear transformation on a vector space, i.e. using matrix algebra. The Generalized Superposition principle, however, has utility beyond the class of LTI systems, being applicable to certain nonlinear systems while maintaining the utility of matrix algebra. Multiplication in the time domain — and thus convolution in the frequency domain — is inherently nonlinear, but real systems that perform time-domain multiplication exhibit the properties required for Generalized Superposition. A key requirement is that the frequency-domain functions be associative when convolved. While not all functions that can be convolved are associative [251], convolution of discrete spectra is associative, and this forms the core mathematical basis of AOM. It is well known that the multiplication of two time-domain waveforms, say \( y(t) = x(t)z(t) \), can be implemented in the frequency domain as a convolution operation:

\[
Y(f) = X(f) * Z(f) \tag{C.1}
\]

In AOM the frequency components of \( X(f) \) and \( Z(f) \) can be arbitrary including having different complex amplitudes for corresponding positive and negative frequencies so that \( x(t) \) and \( z(t) \) could be digitally modulated signals.

### C.3.1 Signal Representation

The purpose of this section is to develop a vector representation of a quasi-periodic signal — this is called a spectral vector. Both single-sided and double-sided spectral representations of a signal can be used. For the purposes of development here, a double-sided spectra has spectral content at positive and negative frequency pairs, but the amplitudes and phases of the pairs need not correspond, see Figure [C.2](a). A single-sided spectrum does not contain positive and negative frequency pairs and is not restricted to non-negative frequencies as shown in Figure [C.2](b). The preferred choice of spectral representation depends on whether the spectrum of the time-domain signal is real-valued or complex-valued. With real-valued signals phasor representations can be used and this relates to the single-sided spectrum. Now, it is not necessary to confine the single-sided spectrum to just non-negative frequencies as situations occur when negative frequencies can have distinct amplitudes and phases, as with complex baseband signals in digital modulation. Another situation occurs with the multiplication of two tones at frequencies \( f_1 \) and \( f_2 \). Then the output of the multiplication contains a tone at
Figure C.2: Spectral representation of a three-tone signal: (a) double-sided spectrum where the complex amplitudes of the components at positive and negative frequencies are not necessarily related; and (b) single-sided spectrum, the preferred representation for real-valued signals.

frequency \( f_1 - f_2 \). If \( f_2 > f_1 \) then this tone will be negative. While the development below appears exhaustive, it is necessary for mathematical rigor. Note that the double-sided spectrum could optionally be used with real-valued signals, however the double-sided spectrum must be used with complex signals such as the envelope signal in digital modulation.

A real-valued quasi-periodic signal composed of a finite number of sinusoids is described by:

\[
x(t) = \sum_{n=0}^{N} x_n(t) = \sum_{n=0}^{N} |x_n| \cos(\omega_n t + \phi_n),
\]

where each cosinusoidal component has radian frequency \( \omega_n \), amplitude \( |x_n| \), and phase \( \phi_n \). This signal can be expanded as follows:

\[
x(t) = \sum_{n=0}^{N} |x_n| \left( \frac{e^{j(\omega_n t + \phi_n)} + e^{-j(\omega_n t + \phi_n)}}{2} \right)
= \frac{1}{2} \sum_{n=0}^{N} |x_n| e^{j(\phi_n + \omega_n t)} + \frac{1}{2} \sum_{n=0}^{N} |x_n| e^{j(-\phi_n - \omega_n t)}
= \frac{1}{2} \sum_{n=0}^{N} [X_{nr} + jX_{ni}] e^{j\omega_n t} + \frac{1}{2} \sum_{n=0}^{N} [X_{-nr} + jX_{-ni}] e^{j\omega_{-n} t},
\]

where \( \omega_{-n} = -\omega_n \), and \( X_n = X_{nr} + jX_{ni} \) is the phasor of the \( n^{th} \) (positive) frequency component of \( x_n(t) \). Taking the Fourier transform (indicated by \( \mathcal{F}[\cdot] \)) of \( x(t) \) yields:

\[
\mathcal{F}_X(\omega) = \mathcal{F}[x(t)] = \mathcal{F} \left[ \frac{1}{2} \sum_{n=0}^{N} [X_{nr} + jX_{ni}] e^{j\omega_n t} + \frac{1}{2} \sum_{n=0}^{N} [X_{-nr} + jX_{-ni}] e^{j\omega_{-n} t} \right]
= \frac{1}{2} \sum_{n=0}^{N} [X_{nr} + jX_{ni}] \delta(\omega - \omega_n) + \frac{1}{2} \sum_{n=0}^{N} [X_{-nr} + jX_{-ni}] \delta(\omega - \omega_{-n})
\]

(C.4)
This is a double-sided spectral representation indicated by the leading superscript $D$ to distinguish it from the single-sided representation introduced later. (In (C.4) the coefficient scale factor of $2\pi$ that appears in the frequency domain has been dropped — this is justified in Section C.8.) Observe that the coefficient of each $\delta(\omega - \omega_n)$ component in the frequency domain is simply $\frac{1}{2}X_n = \frac{1}{2}(X_{nr} + jX_{ni})$ for components on the right hand side of the spectrum, while on the left hand side of the spectrum, the coefficient of each $\delta(\omega - \omega_n)$ is:

$$\frac{1}{2}X_{-n} = \frac{1}{2}(X_{nr} - jX_{ni}) = \frac{1}{2}(X_{nr} - jX_{ni})^* = \frac{1}{2}X_n^*,$$

where $X_n^*$ denotes the complex conjugate of $X_n$. This complex conjugate symmetry in the phasor description applies only to real-valued signals. In general, for a complex-valued signal $X_{-n}$ and $X_n$ need not be related. By definition DC, or $n = 0$, corresponds to the origin in the frequency domain, so there is no imaginary part to the coefficient $X_0$, i.e. $X_0 = X_{0r}$ and the scale factor of $\frac{1}{2}$ does not apply to the DC term. Rewriting of (C.4) and listing the DC term separately leads to:

$$D X(\omega) = X_0\delta(\omega) + \frac{1}{2} \sum_{n=0}^{N} [X_n\delta(\omega - \omega_n) + X_{-n}\delta(\omega - \omega_n)]$$

$$= \sum_{n=0}^{N} \frac{1}{2}X_n\delta(\omega - \omega_n) + X_0\delta(\omega) + \sum_{n=0}^{N} \frac{1}{2}X_{-n}\delta(\omega - \omega_n).$$

(C.5)

In order to facilitate the use of matrix methods, it is convenient to describe a signal as a vector of complex-valued functions of real frequencies instead of a sum of Dirac $\delta$ functions as in (C.5). In general, when the signal is not necessarily real-valued but of arbitrary complex nature, the double-sided spectral vector is defined as:

$$D C X = [X_{-N} \quad X_{-(N-1)} \quad \ldots \quad X_{-1} \quad X_0 \quad X_1 \quad \ldots \quad X_{N-1} \quad X_N]^T$$

(C.6)

Here the leading subscript $C$ indicates that the vector elements are complex-valued functions of a set of real positive and negative frequencies. The frequency set that forms the vector basis for $D C X$ in (C.6) is denoted as:

$$D \omega = \{\omega_{-N}, \omega_{-(N-1)}, \ldots, \omega_{-1}, \omega_0, \omega_1, \ldots, \omega_{N-1}, \omega_N\}.$$  

(C.7)

For example, the spectral vector corresponding to the signal in (C.5) is:

$$D C X = \frac{1}{2} [X_{-N} \quad X_{-(N-1)} \quad \ldots \quad X_{-1} \quad 2X_0 \quad X_1 \quad \ldots \quad X_{N-1} \quad X_N]^T$$

(C.8)

When the signals of interest are sinusoids, as in (C.8), the complex elements of the spectral vector are the phasors of the sinusoidal signals. Thus the time-domain arithmetic addition of
two signals can be represented as vector operations in the discrete frequency domain:

\[ y(t) = x(t) + z(t) \leftrightarrow D^c Y = D^c X + D^c Z, \]  

(C.9)

and subtraction as:

\[ y(t) = x(t) - z(t) \leftrightarrow D^c Y = D^c X - D^c Z, \]  

(C.10)

where the \( y \) and \( z \) quantities are defined in the same manner as the \( x \) quantities.

Some operations on complex matrices require iterative solution techniques and so real vectors and matrices are preferred. The real-valued double-sided spectral vector of \( x(t) \) is defined as

\[ D^c X = [X_{(-N)r}, X_{(-N)i}, \ldots, X_{(-1)r}, X_{(-1)i}, X_0, X_{1r}, X_{1i}, \ldots, X_{Nr}, X_{Ni}]^T, \]  

(C.11)

where \( X_{kr} = \Re[X_k], X_{ki} = \Im[X_k] \), and \( \Re[\cdot] \) indicates that the real part is taken while \( \Im[\cdot] \) indicates that the imaginary part is taken. The frequency set that forms the vector basis of \( D^c X \) in (C.11) is denoted as:

\[ D^c \omega = \{\omega_{(-N)r}, \omega_{(-N)i}, \ldots, \omega_{(-1)r}, \omega_{(-1)i}, \omega_0, \omega_{1r}, \omega_{1i}, \ldots, \omega_{Nr}, \omega_{Ni}\}. \]  

(C.12)

The set in (C.12) has redundant entries in the sense that \( \omega_{kr} = \omega_{ki} \) for all non-zero frequencies, but this notation will be maintained so that each element in the frequency set is paired with a component in the corresponding spectral vector. With \( D^c Y \) and \( D^c Z \) similarly defined, the real-valued frequency-domain arithmetic operator transform pairs are

\[ y(t) = x(t) + z(t) \leftrightarrow D^c Y = D^c X + D^c Z, \]  

(C.13)

and

\[ y(t) = x(t) - z(t) \leftrightarrow D^c Y = D^c X - D^c Z. \]  

(C.14)

Note that spectral vectors must be described using a common frequency set. Thus, if \( D^c X \) and \( D^c Y \) have different spectral content, then a union set of frequencies must be chosen as the vector basis so that the vector addition and subtraction operations will form a closed set.
C.3.2 Multiplication as Convolution in the Frequency Domain

The next stage in complexity of arithmetic operations on two signals is multiplication. Time-domain multiplication corresponds to frequency-domain convolution, thus

$$y(t) = x(t)z(t) \leftrightarrow \mathcal{D}Y(\omega) = \mathcal{D}X(\omega) \ast \mathcal{D}Z(\omega).$$  \hspace{1cm} (C.15)

where \( \ast \) denotes convolution. When \( x(t) \) and \( z(t) \) are composed of sums of sinusoids

$$\mathcal{D}Y = \left\{ X_0\delta(\omega) + \frac{1}{2} \sum_{n=0}^{N} [X_n\delta(\omega - \omega_n) + X_{-n}\delta(\omega - \omega_{-n})] \right\} \ast$$

$$\left\{ Z_0\delta(\omega) + \frac{1}{2} \sum_{n=0}^{N} [Z_n\delta(\omega - \omega_n) + Z_{-n}\delta(\omega - \omega_{-n})] \right\}. \hspace{1cm} (C.16)$$

The aim of this section is to represent time-domain multiplication of signals with possibly non-harmonically related components as matrix multiplication. The result when the discrete frequency components are harmonically-related or discretely-spaced is trivial as then convolution is just circular convolution and the matrix representation of the operation is straightforward. Substituting (C.5) in (C.16) yields:

$$\mathcal{D}Y(\omega) = \sum_{\omega_k} \mathcal{D}X(\omega) \mathcal{D}Z(\omega - \omega_k).$$

$$\mathcal{D}Y(\omega) = \mathcal{D}X(\omega) \ast \mathcal{D}Z(\omega)$$

$$\mathcal{D}Y(\omega) = \sum_{\omega_k} \mathcal{D}X(\omega_k) \mathcal{D}Z(\omega - \omega_k)$$

$$= \sum_{\omega_k} \left[ \left[ \frac{1}{2}X_{-N}\delta(\omega_k - \omega_{-N}) + \ldots + \frac{1}{2}X_{-2}\delta(\omega_k - \omega_{-2}) + \frac{1}{2}X_{-1}\delta(\omega_k - \omega_{-1}) + X_0\delta(\omega_k) + \frac{1}{2}X_1\delta(\omega_k - \omega_1) + \frac{1}{2}X_2\delta(\omega_k - \omega_2) + \ldots + \frac{1}{2}X_N\delta(\omega_k - \omega_N) \right] \right].$$

$$= \sum_{\omega_k} \left[ \left[ \frac{1}{2}Z_{-N}\delta[\omega - (\omega_k - \omega_{-N})] + \ldots + \frac{1}{2}Z_{-2}\delta[\omega - (\omega_k - \omega_{-2})] + \frac{1}{2}Z_{-1}\delta[\omega - (\omega_k - \omega_{-1})] + Z_0\delta(\omega - \omega_k) + \frac{1}{2}Z_1\delta[\omega - (\omega_k - \omega_1)] + \frac{1}{2}Z_2\delta[\omega - (\omega_k - \omega_2)] + \ldots + \frac{1}{2}Z_N\delta[\omega - (\omega_k - \omega_N)] \right] \right].$$

(C.17)

The only stipulation on the signals is that \( \omega_0 = 0 \), otherwise the frequency set is completely arbitrary and represents any number of incommensurate tones.

Two observations can be made concerning the frequency-domain convolution above. First, for a total of \( 2N \) distinct input frequencies the output \( \mathcal{D}Y(\omega) \) has components at DC
and 2N positive and negative frequencies (4N + 1 frequencies total), with the minimum radian frequency being \( \omega_{-2N} = -2\omega_N \) and the maximum \( \omega_{2N} = 2\omega_N \). Second, due to the sifting property of the \( \delta \) function, for each distinct value of frequency indicated by the index \( k \) (i.e. for each \( \omega_k \)), only one term in \( ^D X(\omega_k) \) will be non-zero. This term then forms a set of products with all of the components of \( ^D Z(\omega - \omega_k) \), which retains the independent variable \( \omega \). Put another way, each sifted term in \( ^D X(\omega_k) \) forms product terms with all of the terms in \( ^D Z(\omega) \), but shifted in the frequency domain to the frequency dictated by the argument of the \( \delta \) functions for \( ^D Z(\omega) \) in (C.17). In the course of the convolution operation, there may be a number of non-zero products for different values of the frequency index \( k \) for which the \( \delta \) functions in \( ^D Z(\omega) \) happen to evaluate to the same frequency. The convolution operation calls for these terms to add linearly, conforming to the notion of generalized superposition.

### C.3.3 Matrix Representation

In the matrix representation of frequency-domain convolution the signals \( ^D X(\omega) \), \( ^D Y(\omega) \), and \( ^D Z(\omega) \) have the same set of basis frequencies. Then, \( ^D Y(\omega) \) can be related to \( ^D X(\omega) \) and \( ^D Z(\omega) \) through a matrix transformation. This is illustrated in Section C.9 by way of an example for the case of three incommensurate tones. The result is the introduction of a spectral transformation matrix \( ^D C T_x \) defined as the matrix representation of the spectral vector \( ^D C X \) such that \( ^D C T_x \) is conformable for matrix multiplication with the spectral vector \( ^D C Z \). Thus the convolution operation is represented as a matrix-vector operation so that the multiplication transform pair is:

\[
y(t) = x(t)z(t) \leftrightarrow ^D C Y = ^D C T_x^D C Z . \tag{C.18}
\]

\( ^D C T_x \) is reasonably easily developed following a simple pattern as a result of the double sided spectrum (as illustrated in Appendix B). In a later section the single-sided version of the spectral transformation is introduced and an algorithm for developing it is presented. When discrete spectra only are being considered, the single-sided version is preferred due to its smaller memory requirements, efficiency of computation, and more easily-understood results. The double-sided version must be used with digital modulation but this is not discussed further here. In terms of real vector-matrix operations (C.18) can be written

\[
y(t) = x(t)z(t) \leftrightarrow ^D Y = ^D T_x^D Z , \tag{C.19}
\]
where now $D\mathbf{T}_x$ is formed by replacing each complex entry of $D\mathbf{T}_x$ in (C.18) by a submatrix so that

$$
\begin{bmatrix}
D\mathbf{T}_x(2i,2j-1) & D\mathbf{T}_x(2i-1,2j) \\
D\mathbf{T}_x(2i,2j) & D\mathbf{T}_x(2i-1,2j) \\
\end{bmatrix} =
\begin{bmatrix}
\Re\mathcal{C}_x(i,j) & -\Im\mathcal{C}_x(i,j) \\
\Im\mathcal{C}_x(i,j) & \Re\mathcal{C}_x(i,j) \\
\end{bmatrix}
$$

for $i, j \neq 0$

$$
\begin{bmatrix}
D\mathbf{T}_x(0,2j-1) & D\mathbf{T}_x(0,2j) \\
D\mathbf{T}_x(0,2j) & D\mathbf{T}_x(0,2j) \\
\end{bmatrix} =
\begin{bmatrix}
\Re\mathcal{C}_x(0,j) & -\Im\mathcal{C}_x(0,j) \\
\Im\mathcal{C}_x(0,j) & \Re\mathcal{C}_x(0,j) \\
\end{bmatrix}
$$

for $i = 0, j \neq 0$

$$
\begin{bmatrix}
D\mathbf{T}_x(2i-1,0) \\
D\mathbf{T}_x(2i,0) \\
\end{bmatrix} =
\begin{bmatrix}
\Re\mathcal{C}_x(i,0) \\
\Im\mathcal{C}_x(i,0) \\
\end{bmatrix}
$$

for $i \neq 0, j = 0$

$$
\begin{bmatrix}
D\mathbf{T}_x(0,0) \\
\end{bmatrix} =
\begin{bmatrix}
\Re\mathcal{C}_x(0,0) \\
\end{bmatrix}
$$

for $i = 0, j = 0$.

(C.20)

The submatrices in (C.20) come from considering the product of complex numbers $X_k = X_k + jX_ki$ and $Z_l = Z_l + jZ_li$; for example when the complex multiplication of $X_1$ and $Z_2$ is performed, the following result is obtained:

$$
X_1Z_2 = (X_1 + jX_1i) \cdot (Z_2 + jZ_2i)
$$

$$
= X_1Z_2 - X_1iZ_2i + j(X_1Z_2i + X_1iZ_2i).
$$

(C.21)

In (C.18) the specific reference to the set $D\omega$ defining the frequency ordering has been dropped, but it is understood. In this case of double-sided spectra, the frequency set is:

$$
D\mathcal{C} = \{ \omega_{-2N}, \ldots, \omega_{-N}, \omega_{-(N-1)}, \ldots, \omega_{-1}, \omega_0, \omega_1, \ldots, \omega_N, \ldots, \omega_{2N} \}.
$$

(C.22)

Finally, observe that due to the associative nature of the discrete convolution process, (C.18) could just as easily have been written:

$$
D\mathcal{C}\mathbf{Y} = D\mathcal{C}\mathbf{T}_z D\mathcal{C}\mathbf{X}.
$$

(C.23)

since $x(t)z(t) = z(t)x(t)$.

The frequency-domain arithmetic operator for division is developed in the following manner. Consider the multiplication transform pair

$$
z(t) = x(t)y(t) \leftrightarrow D\mathcal{C}\mathbf{Z} = D\mathcal{C}\mathbf{T}_z D\mathcal{C}\mathbf{Y}.
$$

(C.24)
This can be rearranged in the following way:

\[
z(t) = x(t) \cdot y(t) \leftrightarrow \frac{D}{C}Z = \frac{D}{C}T_x^D Y
\]

\[
z(t)/x(t) = x(t) \cdot y(t)/x(t) \leftrightarrow \frac{D}{C}T_x^{-1}Z = \frac{D}{C}T_x^{-1}T_x^D Y
\]

\[
y(t) = z(t)/x(t) \leftrightarrow \frac{D}{C}Y = \frac{D}{C}T_x^{-1}D Z .
\] (C.25)

Similarly,

\[
y(t) = z(t)/x(t) \leftrightarrow \frac{D}{Y} = \frac{D}{T_x^{-1}D} Z .
\] (C.26)

With the four mathematical operations of addition \([C.9]\), subtraction \([C.10]\), multiplication \([C.18]\), and division \([C.26]\), any analytic operation can be implemented. However division requires inversion of a matrix and it is problematic to use complex-valued matrices as iterative techniques are required. The next section develops a real-valued convolution matrix and finally a single-sided implementation is presented that yields the final preferred computationally efficient form.

### C.4 Arithmetic Operator Method For Real Signals

#### C.4.1 Single-Sided Spectrum

The previous section presented the Arithmetic Operator Method (AOM) on the double-sided domain, \(D\omega\), that includes both positive and negative frequencies. When signals are real and can be represented as sums of sinusoids it is possible to exploit mathematical symmetry to develop a computationally more convenient and efficient implementation of AOM that eliminates the need to use negative frequencies in the spectral vectors and spectrum transform matrix. The reduced frequency set will be indicated by dropping the leading superscript \(D\) and replacing it with \(S\) to indicate a single-sided frequency domain. Thus \(D\omega\) becomes \(S\omega\). Referring to \([C.17]\), \(S\omega\) is created by dropping the negative elements of the frequency set. Thus,

\[
\hat{S}\omega = \left\{ \omega_0, \omega_1, \ldots, \omega_{N-1}, \omega_N \right\}
\] (C.27)
and real signals considered are represented as:

\[
x(t) = \sum_{k} |X_k| \cos(\omega_k t + \phi_k)
\]

\[
= \sum_{k} \left( \frac{1}{2} X_k e^{j\omega_k t} + \frac{1}{2} X_{-k} e^{-j\omega_k t} \right)
\]

\[
= \sum_{k} \left( \frac{1}{2} X_k e^{j\omega_k t} + \frac{1}{2} X_k^* e^{-j\omega_k t} \right)
\]  \hspace{1cm} (C.28)

Now, noting that \( X_k = X_{kr} + jX_{ki} \) and that \( X_k^* = X_{kr} - jX_{ki} \), observe that the phase of the second term in (C.28) is opposite to that of the first term. Thus,

\[
x(t) = \sum_{k} \left( \frac{1}{2} X_k e^{j\omega_k t} + \frac{1}{2} X_k^* e^{-j\omega_k t} \right)
\]

\[
= \sum_{k} \left( \frac{1}{2} |X_k| e^{j(\omega_k t + \phi_k)} + \frac{1}{2} |X_k| e^{-j(\omega_k t + \phi_k)} \right)
\]  \hspace{1cm} (C.29)

This simple phase relationship permits the reduction of the frequency domain to a single-sided spectrum. Comparing (C.29) to (C.28), it should be noted that the proper amplitude relationship is maintained only if the positive frequency coefficients are scaled up by a factor of 2. Note that at DC, \( k = 0 \), the coefficient of the signal is \( X_0 \). Proper phase relationships are maintained by substituting the complex conjugate of the coefficient at frequency \( \omega_k \) whenever an instance of the negative frequency \( \omega_{-k} = -\omega_k \) is required. For example, the frequency of second order intermodulation is always the result of addition and subtraction of two incommensurate frequencies. Thus any vector operations involving the subtracted frequency would use the complex conjugate term, \( X_k^* \), for this frequency in any vector computations. To carry out this operation, only the bottom right quarter of the double-sided complex convolution matrix (i.e. \( \frac{D}{C} \mathbf{T}_z \)) is used. A scale factor for the center column of the double-sided convolution matrix, which is the DC portion of the matrix, must also be included. In all of the matrix multiplication operations, this column’s elements are multiplied by \( X_0 \). \( X_0 \) is not scaled in (C.28), but the effect of the matrix multiplication in this column creates symmetrical convolution products on both sides of the spectrum. Thus, for correct results, all of the non-DC entries in this column are scaled by a factor of 2. For the three-tone \((\omega_1, \omega_2, \omega_3)\) example presented in Appendix B the single-sided frequency set is

\[
S_{C}^\omega = \{ \omega_0, \omega_1, 2\omega_1, \omega_2 + \omega_1, \omega_2, \omega_2 + \omega_1, 2\omega_2, \omega_3 + \omega_2, \omega_3 + \omega_1, \omega_3, \omega_3 + \omega_1, \omega_3 + \omega_2, 2\omega_3 \} \]  \hspace{1cm} (C.30)
and the spectral vector of the input signal is

$$\hat{S}_C^TX = \begin{bmatrix} X_0 & X_1 & 0 & 0 & X_2 & 0 & 0 & 0 & 0 & X_3 & 0 & 0 & 0 \end{bmatrix}^T. \quad (C.31)$$

Now, noting that $\omega_{-x} = -\omega_x$ and $X_{-n} = X_n^*$, the single-sided convolution matrix is:

$$\hat{S}_C^T\mathbf{T}_x = \begin{bmatrix} X_0 & \frac{1}{2}X_1^* & 0 & 0 & \frac{1}{2}X_2^* & 0 & 0 & 0 & 0 & \frac{1}{2}X_3^* & 0 & 0 & 0 \\ X_1 & X_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2}X_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2}X_1^* & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ X_2 & 0 & 0 & 0 & X_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2}X_2 & 0 & 0 & \frac{1}{2}X_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2}X_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2}X_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2}X_1^* & 0 & 0 & 0 & 0 & 0 & 0 \\ X_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & X_0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2}X_3 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2}X_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2}X_3 & 0 & 0 & 0 & \frac{1}{2}X_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2}X_3 & 0 & 0 & 0 & 0 \end{bmatrix}. \quad (C.32)$$

With the scale factor correction described above the single-sided spectral vectors are phasors. Note that the order of the frequencies in $\hat{S}_C\omega$ is not important except that the first entry must be $\omega_0$ or DC. However, matrices and vectors must use the same ordering in order to be conformable for matrix multiplication.

**C.4.2 Definition of the Spectrum Transform Matrix**

The real-valued Spectrum Transform Matrix can now be constructed for the example case of mixing three tones as it is implemented in AOM. This is done by beginning with the convolution matrix in (C.32) and using the replacement strategy of (C.20). The result is given in (C.33) on page 384.
\[
T_x = \begin{bmatrix}
X_0 & \frac{1}{2}X_{1r} & -\frac{1}{2}X_1^* & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{2}X_2^* & 0 & 0 & 0 & 0 & 0
X_{1r} & X_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
X_{1s} & 0 & X_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
0 & \frac{1}{2}X_{1r} & -\frac{1}{2}X_1^* & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
0 & \frac{1}{2}X_{1s} & -\frac{1}{2}X_1^* & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
X_{2r} & 0 & 0 & 0 & 0 & 0 & X_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
0 & \frac{1}{2}X_{2r} & -\frac{1}{2}X_2^* & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
0 & \frac{1}{2}X_{2s} & -\frac{1}{2}X_2^* & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2}X_{2r} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2}X_{2s} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2}X_2^* & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
X_{3r} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2}X_{3r} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
X_{3s} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2}X_{3s} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
0 & \frac{1}{2}X_{3r} & -\frac{1}{2}X_3^* & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
0 & \frac{1}{2}X_{3s} & -\frac{1}{2}X_3^* & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2}X_{3r} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2}X_{3s} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2}X_3^* & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2}X_3^* & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2}X_3^* & 0 & 0 & 0 & 0 & 0 & 0 & 0
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2}X_3^* & 0 & 0 & 0 & 0 & 0 & 0
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2}X_3^* & 0 & 0 & 0 & 0 & 0
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2}X_3^* & 0 & 0 & 0 & 0
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2}X_3^* & 0 & 0 & 0
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2}X_3^* & 0 & 0
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2}X_3^* & 0
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2}X_3^*
\end{bmatrix}
\]
Since all of the non-zero frequencies in (C.30) now correspond to two real values in the Spectral Vector, the frequency set $\hat{S}_C \omega$ is single-valued for DC, but doubled-valued otherwise. So as to maintain an element-level correspondence between frequency set elements and elements in the Spectral Vector, a variant on the frequency set, which will be denoted simply as $\omega$ (leading superscripts dropped), is defined such that each non-zero frequency redundantly appears twice in the set. Thus if $\omega_n$ is a non-zero frequency in the set $\hat{S}_C \omega$ in (C.30), then in the set $\omega$ there will be two elements denoted as $\omega_{nr}$ and $\omega_{ni}$, and thus the frequency set $\omega$ is given by:

$$\omega = \{ \omega_0, \omega_1r, \omega_1i, 2\omega_1r, 2\omega_1i, (\omega_2 - \omega_1)_r, (\omega_2 - \omega_1)_i, \omega_2r, \omega_2i, (\omega_2 + \omega_1)_r, (\omega_2 + \omega_1)_i, 2\omega_2r, 2\omega_2i, (\omega_3 - \omega_2)_r, (\omega_3 - \omega_2)_i, (\omega_3 - \omega_1)_r, (\omega_3 - \omega_1)_i, \omega_3r, \omega_3i, (\omega_3 + \omega_1)_r, (\omega_3 + \omega_1)_i, (\omega_3 + \omega_2)_r, (\omega_3 + \omega_2)_i, 2\omega_3r, 2\omega_3i \}. \quad (C.34)$$

For $Z$ the result is:

$$Z = \begin{bmatrix} z_0 & z_1r & z_1i & 0 & 0 & 0 & 0 & z_2r & z_2i & 0 & 0 & 0 & 0 & 0 & 0 & z_3r & z_3i & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T. \quad (C.35)$$

There are several properties of the Spectrum Transform Matrix to observe: First, for $K$ non-zero frequencies, the Spectrum Transform Matrix has dimensions $2K + 1 \times 2K + 1$, where the $+1$ item is due to the DC component, which has no imaginary part. Note also that the Spectral Vector has $2K + 1$ elements. Second, both the Spectrum Transform Matrix and its conformable Spectral Vector contain only real-valued elements. Third, where a negative frequency phasor might be required in the Spectrum Transform Matrix (as called for with the two-sided convolution matrix), the complex conjugate of the corresponding positive frequency phasor element is substituted. Finally, and most importantly, a Spectrum Transform Matrix $T_x$ for a given Spectral Vector $X$ will always contain $X$ as its first column. Therefore, once the frequency domain of interest is specified, the Spectrum Transform Matrix for solving a given problem is unique.

### C.4.3 Generalized Application of the Spectrum Transform Matrix and Spectral Vector

The Spectrum Transform Matrix as described in the previous section implements the frequency-domain convolution for time domain multiplication. However it is straightforward to extend the
utility of the Arithmetic Operator Method. Consider $y(t)$ as an arbitrary polynomial transfer function of $x(t)$, for example,

$$y = a_0 + a_1 x + a_2 x^2 + \ldots + a_M x^M = \sum_{m=0}^{M} a_m x^m \ .$$

(C.36)

Examining the term for $m = 2$ more closely, this can be expressed as $y(t)|_{m=2} = a_2 x^2 = a_2 x \cdot x$, from which it follows that $a_2 x \cdot x \leftrightarrow a_2 T_x X$. The resulting matrix product $T_x X$ can then be substituted into the term for $m = 3$ to obtain $a_3 x \cdot x^2 \leftrightarrow a_3 T_x(T_x X) = a_3 T_x^2 X$. In general,

$$y = \sum_{m=0}^{M} a_m x^m \leftrightarrow Y = \sum_{m=0}^{M} a_m T_x^{m-1} X \ .$$

(C.37)

Alternatively, the spectral vector $Y$ can be extracted as the first column of its spectrum transform matrix $T_y$:

$$T_y = \sum_{m=0}^{M} a_m T_x^m \ .$$

(C.38)

When the maximum order $M$ of the polynomial in (C.36) is known in advance, more efficient computation will result from using the form

$$Y = \sum_{m=0}^{M} a_m T_x^{m-1} X = a_0 + (a_1 I + T_x(a_2 I + T_x(a_3 I + \ldots + T_x(a_{M-1} I + T_x(a_M))))))X \ .$$

(C.39)

In (C.39) $I$ is an identity matrix with the same dimensions as $T_x$ and $a_0$ remains a scalar quantity. The time-domain form $y(t)$ can be recovered by simply evaluating the spectral vector as a function of time at its various components, summing the results, and adding the scalar quantity $a_0$ as a final step.

Another interesting application of $T_x$ is in implementing time-domain division by operating on the frequency-domain spectra of signals. Consider the division operation $y(t) = z(t)/x(t)$. This is easily implemented in AOM through its frequency-domain equivalent operation using the inverse of $T_x$:

$$Y = T_x^{-1} Z \ .$$

(C.40)

Applications of AOM using $T_x$ in matrix multiplication are known as spectral multiplication, while applications of AOM using $T_x^{-1}$ in matrix multiplication are known as spectral division. Using spectral addition, subtraction, multiplication, and division any analytic function in the time-domain can be transformed into a spectral operation.
C.5 Computer-Aided Method For Developing The Spectrum Transform Matrix

This paper has already described the construction of a Spectrum Transform Matrix for an example of multiplying two 3-tone signals as a reduction from a generalized, double-sided, convolution matrix, using ad hoc reasoning. Chang and Steer originally described an algorithm for automatically developing the Spectrum Transform Matrix, but the description used ad hoc methods to determine the frequencies of interest. The computer-aided development of the Spectrum Transform Matrix is presented here. This begins with the introduction of the Basic Intermodulation Product Description (BIPD), a method for producing a table, called the BIPD table, that describes the relationships among frequency components. The BIPD table is then used to drive the development of another table, called the Spectrum Mapping Table. The Spectrum Mapping Table is derived by anticipating all combinations of the frequency components (plus intermodulation distortion products) taken two at a time that will result in incremental components in the set of output frequencies specified by the BIPD table. (The interaction of two tones at a time is a consequence of spectral addition, subtraction, multiplication, and division operating on spectral components two at a time.) The BIPD Table will be described first followed by development of the Spectrum Mapping Table, which in essence describes the interaction of frequency components taken two at a time. Finally the development of the Spectrum Transform Matrix from the Spectrum Mapping Table is described. The development of the computer-aided method here is limited to non-negative single-sided spectra, but can be extended to a generalized form for double-sided complex spectra.

C.5.1 Basic Intermodulation Product Description Table

The Basic Intermodulation Product Description (BIPD) table is a list of frequency indices that defines the anticipated frequency domain of the output signal $y(t)$. Along with DC, the entries in the BIPD table form the radian frequency set $\mathbb{S} \omega$. That is,

$$\mathbb{S} \omega = \{\text{DC and all entries in the BIPD table}\}$$  \hspace{1cm} (C.41)

The choice of the frequency components considered is arbitrary with the choice affecting the accuracy of the signal representation. The frequencies of interest are generally determined by considering the excitation signals and the degree of nonlinearity. The frequency components
being considered can be manually selected, or an automatic procedure such as the rectangular or triangular truncation schemes [86] can be used. See Section C.10 for an example of automatic frequency selection using triangular truncation. The BIPD table can also be set manually.

An Intermodulation Product Description (IPD) describes the relationship of a frequency component in terms of other frequency components in the chosen spectrum. For example, if a frequency is defined by the relationship

\[ f_{k_y} = n_1^{(k_y)}f_1 + n_2^{(k_y)}f_2 + \cdots + n_n^{(k_y)}f_n \]  

(C.42)

then the set of \( n_i \) values constitutes an IPD. Each frequency may have a large number of IPD’s, but only one, the BIPD, uniquely identifies a frequency component by describing it in terms of the incommensurate excitation tones. The frequency selection scheme proceeds as follows: For a choice of \( n \) frequencies the BIPD table is generated by creating a table of \( n \) columns (denoted \( n_1, n_2, \ldots, n_n \), with one column for each input frequency) where each row in the table contains a unique combination of index numbers (coefficient weightings) for each frequency \( f_1, f_2, \ldots, f_n \).

With each row of the table designated by the frequency index \( k_y \), then the entries at each column \( n_i \) within each row are known as index numbers and are designated \( n_i^{(k_y)} \). As an example consider the spectrum produced by an incommensurate two-tone signal. This will be used to illustrate the construction of the BIPD table. (This example differs from the earlier three tone example used in the development of the mathematical foundation as two tones will suffice for illustrative purposes here.) The first step in the process is to choose the frequency spectrum utilized in the analysis. The complete BIPD Table for the spectrum of Figure C.3 is given in Table C.1.

![Figure C.3](image-url)

Figure C.3: The approximate spectrum resulting from the excitation of a nonlinear system by incommensurate tones \( f_1 \) and \( f_2 \).
By convention, the first few entries in the BIPD table in Table C.1 correspond to the incommensurate tones. (Note that for these entries \(|n| = 1\) and \(k_x = i\), where \(i\) is the subscript for \(n_i\) in the column containing index number 1.) Thus there are 12 non-zero output radian frequencies of interest in this example: \(\omega_1, \omega_2, \omega_2 - \omega_1, \omega_2 + \omega_1, 2\omega_1 - \omega_2, 2\omega_2 - \omega_1, 2\omega_1, 2\omega_2, 2\omega_1 + \omega_2, \omega_1 + 2\omega_2, 3\omega_1, \) and \(3\omega_2\). Thus during construction, the Spectrum Transform Matrix will have twice this number of rows and columns for each frequency (for real and imaginary parts). Since the DC component has no imaginary part, the second row and column will simply be deleted from the matrix after executing the algorithm to create it. Thus for the case of DC and 12 non-zero frequencies, the Spectrum Transform Matrix will be 25 by 25.

### Table C.1: BIPD Table for spectrum of Figure C.3

| \(k_y\) | \(|n|\) | \(n_1\) | \(n_2\) |
|---|---|---|---|
| 1 | 1 | 1 | 0 |
| 2 | 1 | 0 | 1 |
| 3 | 2 | -1 | 1 |
| 4 | 2 | 1 | 1 |
| 5 | 2 | 2 | 0 |
| 6 | 2 | 0 | 2 |
| 7 | 3 | 2 | -1 |
| 8 | 3 | -1 | 2 |
| 9 | 3 | 2 | 1 |
| 10 | 3 | 1 | 2 |
| 11 | 3 | 3 | 0 |
| 12 | 3 | 0 | 3 |

### C.5.2 Spectrum Mapping Table

The Spectrum Mapping Table describes all of the possible vector frequency component combinations of two inputs, \(x(t)\) and \(z(t)\), that will map back onto the BIPD table. As an example
of how an entry in the Spectrum Mapping Table is identified, suppose that a component of $X(\omega)$ has frequency index $k_x = 7$, so that $n_1^{(7)} = 2$ and $n_2^{(7)} = -1$, and that a component of $Z(\omega)$ has frequency index $k_z = 3$, so $n_1^{(3)} = -1$ and $n_2^{(3)} = 1$. When a vector addition of the components’ index numbers is performed, the result is $n_1^{k_y} = n_1^{(7)} + n_1^{(3)} = 2 - 1 = 1$ and $n_2^{k_y} = n_2^{(7)} + n_2^{(3)} = -1 + 1 = 0$. For this example, the vector sum of the index numbers maps back onto the BIPD table, since $n_1^{k_y} = 1$ and $n_2^{k_y} = 0$ are the index numbers for frequency index $k_y = 1$, that is $n_1^{k_y} = n_1^{(1)}$ and $n_2^{k_y} = n_2^{(1)}$.

Construction of the Spectrum Mapping Table is done systematically by setting the frequency index for the output, $k_y$, to each of the values in the BIPD table, then determining all of the vector combinations of index numbers for $k_x$ and $k_z$ which equate to the index numbers for $k_y$. Every valid combination is entered into the Spectrum Mapping Table, and often there are many entries for each $k_y$ corresponding to different intermodulation components. At this point, the development of the computer-aided method is limited to non-negative frequencies and single-sided spectra (but in the future will be extended to a generalized form for double-sided complex spectra). Among the valid combinations that must be considered are those instances of intermodulation where one frequency is subtracted from another, with the result being still a positive frequency. Referring back to the Spectrum Transform Matrix shown in (C.32), these combinations are the cases where an entry in the matrix includes the complex conjugate of a spectral vector element. In order to account for these instances, the algorithm for constructing the Spectrum Mapping Table also attempts combinations of frequency mnemonics where the index numbers are multiplied by $-1$. Any valid combinations will also be entered into the Spectrum Mapping Table. This sign multiplier, which can be 1 or $1$, is denoted by $s_x$ for $k_x$ and by $s_z$ for $k_z$. Thus the Spectrum Mapping Table entry for the case of $k_y = 1$, $k_x = 7$, and $k_z = 3$ mentioned above will be as shown in Table C.2.

<table>
<thead>
<tr>
<th>$k_y$</th>
<th>$k_x$</th>
<th>$s_x$</th>
<th>$k_z$</th>
<th>$s_z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

Through use of the BIPD table and the sign multiplier, all valid combinations of non-
zero frequencies in the Spectrum Mapping Table are taken into account. The BIPD table does not include DC terms, so instances where DC terms are mixed with non-zero frequencies, or when two DC terms are mixed, are taken into account at the time the Spectrum Mapping Table is constructed. Referring back to the definition of the spectral vectors in (C.16) and (C.17), $X_0$ and $Z_0$ denote the DC components, and $Y_0$ was defined as a DC term in the output by stipulating that $\omega_0 = 0$. Thus appearances of DC terms in the convolution matrix products can be taken into account by adding $k_y = 0$, $k_x = 0$, and $k_z = 0$ to the combinations that must be tested at the time the Spectrum Mapping Table is created. Mixing of the pure DC terms in both vectors is taken into account by including an entry in the table where $k_y = 0$, $k_x = 0$, $s_x = 1$, $k_z = 0$, and $s_z = 1$. Also, rectification is accounted for by including entries in the table for $k_y = 0$ where $s_x = 1$, $s_z = -1$, and $k_x = k_y$. Entries of this sort are included for every frequency mnemonic in the BIPD table. This completes the entries in the Spectrum Mapping Table for $k_y = 0$. In order to account for the mixing of DC terms with non-zero frequencies, $k_x = 0$ and $k_z = 0$ are included with their frequency indices defined to be 0 when performing vector addition of frequency indices. Because of the single-sided frequency limitation, the possibilities of $s_x = -1$ and $s_z = -1$ are excluded when a non-zero frequency is being mixed with a DC term, i.e. the algorithm does not permit entries in the Spectrum Mapping Table when $k_x = 0$ and $s_z = -1$ or when $k_z = 0$ and $s_x = -1$. (Note that the value of the sign for non-zero frequencies is of no concern since the BIPD table contains only single-sided frequencies by construction and only valid combinations of frequency indices, including the signs, which map back into the BIPD table are permitted.)

For the two-tone example with 12 entries in the BIPD table, the corresponding Spectrum Mapping Table contains 175 entries. For illustrative purposes, the portions of that Spectrum Mapping Table for DC ($k_y = 0$) and for $k_y = 7$ are given in Table C.3.

Table C.3: Portions of the Spectrum Mapping Table created from BIPD Table in Table C.1 for $k_y = 0$ (DC) and $k_y = 7$

<table>
<thead>
<tr>
<th>$k_y$</th>
<th>$k_x$</th>
<th>$s_x$</th>
<th>$k_z$</th>
<th>$s_z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
</tr>
</tbody>
</table>

Continued on next page
Table C.3 (continued)

<table>
<thead>
<tr>
<th>$k_y$</th>
<th>$k_x$</th>
<th>$s_x$</th>
<th>$k_z$</th>
<th>$s_z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>1</td>
<td>6</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>1</td>
<td>7</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>1</td>
<td>8</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>9</td>
<td>1</td>
<td>9</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>1</td>
<td>10</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>11</td>
<td>1</td>
<td>11</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>12</td>
<td>1</td>
<td>12</td>
<td>-1</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>-1</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>-1</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>-1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>1</td>
<td>8</td>
<td>-1</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>-1</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>-1</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>-1</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>-1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>1</td>
<td>6</td>
<td>-1</td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td>1</td>
<td>4</td>
<td>-1</td>
</tr>
</tbody>
</table>

C.5.3 Spectrum Transform Matrix

The Spectrum Transform Matrix is developed from the Spectrum Mapping Table. It is based on the phasor result of the operation specified by a single Spectrum Mapping Table entry.
Referring to (C.21), this is given by

\[
Y'_{k_y} = (\varepsilon X_{k_z r} + j\varepsilon s_x X_{k_z i}) \cdot (\varepsilon Z_{k_z r} + j\varepsilon s_x Z_{k_z i}) \\
= \varepsilon X_{k_z r} Z_{k_z r} - s_x s_z \varepsilon X_{k_z i} Z_{k_z i} \\
+ j(s_x \varepsilon X_{k_z i} Z_{k_z r} + s_x \varepsilon X_{k_z r} Z_{k_z i}),
\]

(C.43)

where the nomenclature developed to describe the BIPD and Spectrum Mapping Tables is used. The quantity \( \varepsilon \) captures the special handling required for terms involving DC. It is defined as follows. Referring to the Spectrum Transform Matrix shown in (C.33), note that the matrix contains scale factors of \( \frac{1}{2} \) on every term in the matrix except for the first column and wherever \( X_0 \) appears. Since the first column entries in the Spectrum Transform Matrix are always multiplied by \( Z_0 \) (i.e. the matrix is constructed so that its columns are conformable to the spectral vector’s rows), this means that \( \varepsilon = 1 \) whenever \( k_z = 0 \). Also, any entry in the matrix containing \( X_0 \) corresponds to a case where \( k_x = 0 \). For all other entries in the Spectrum Transform Matrix, \( \varepsilon = \frac{1}{2} \). Now, in terms of real and imaginary parts a Spectrum Transform Matrix entry becomes:

\[
\begin{bmatrix}
Y'_{k_y r} \\
Y'_{k_y i}
\end{bmatrix} = \begin{bmatrix}
\varepsilon X_{k_z r} & -\varepsilon s_x s_z X_{k_z i} \\
\varepsilon s_x X_{k_z i} & \varepsilon s_x X_{k_z r}
\end{bmatrix} \begin{bmatrix}
Z_{k_z r} \\
Z_{k_z i}
\end{bmatrix}
\]

(C.44)

with appropriate modification to handle DC, as in (C.20).

It is important to note the primes on \( Y \) in (C.33) and (C.44). This emphasizes that only one mixing product from the Spectrum Mapping Table has been shown here. The matrix in (C.44) is the submatrix of the Spectrum Transform Matrix at the \( k_y \)th row and the \( k_z \)th column. The construction of the Spectrum Transform Matrix is done by first initializing all entries to zero and then adding via superposition to the matrix based upon the specifications of the Spectrum Mapping Table. The dimensions of the Spectrum Transform Matrix are square and initialized to 2 times the number of entries in the BIPD table — accounting for the fact that complex phasor products are stored as two real-valued numbers — plus 2 additional rows and columns for the DC terms. The Spectrum Transform Matrix construction is complete when all of the entries in the Spectrum Mapping Table have been read and corresponding elements in the Spectrum Transform Matrix have been updated using superposition. Since the DC terms do not have imaginary parts the second row and second column of the Spectrum Transform Matrix are deleted when the algorithm to create the Spectrum Transform Matrix is completed.
The final matrix is thus square with dimensions equal to twice the number of BIPD table entries plus 1.

The algorithm for developing the Spectrum Transform Matrix can now be defined. If the variable \( \text{Nrows} \) is defined as the number of entries (rows) in the Spectrum Mapping Table, and the Spectrum Mapping Table is organized as an \( \text{Nrows} \) by 5 matrix with the variable name \( \text{SpMapTab} \), then the algorithm presented in Listing C.1 develops the Spectrum Transform Matrix.

Listing C.1: Matlab® Pseudo-Code for Developing the Spectrum Transform Matrix

```
1  Tx = zeros(2K+2,2K+2); % Initialize all elements of matrix to zero
2  for row = 1:N
3    ky = sp_map_tab(row,1);
4    kx = sp_map_tab(row,2);
5    sx = sp_map_tab(row,3);
6    kz = sp_map_tab(row,4);
7    sz = sp_map_tab(row,5);
8    X_\text{k}\text{\_x}\_r = X\text{\_new}(2*kx+1);
9    X_\text{k}\text{\_x}\_i = X\text{\_new}(2*kx+2);
10   if ( kx == 0 | kz == 0 )
11      epsilon = 0.5;
12    else
13      epsilon = 1.0;
14    end % end if condition
15   Tx(2*ky+1, 2*kz+1) = ...  
16   Tx(2*ky+1, 2*kz+1) + epsilon*X_\text{k}\text{\_x}\_r;
17   Tx(2*ky+1, 2*kz+1+1) = ... 
18   Tx(2*ky+1, 2*kz+1+1) = sx*sz*epsilon*X_\text{k}\text{\_x}\_i;
19   Tx(2*ky+1+1, 2*kz+1) = ... 
20   Tx(2*ky+1+1, 2*kz+1) + sx*epsilon*X_\text{k}\text{\_x}\_i;
21   Tx(2*ky+1+1, 2*kz+1+1) = ... 
22   Tx(2*ky+1+1, 2*kz+1+1) + sz*epsilon*X_\text{k}\text{\_x}\_r;
23  end % end for loop
```

The variables appearing in the pseudocode in Table 4 correspond to those shown in Equation \( \text{C.44} \). Assuming that there are \( K \) distinct non-zero frequencies in the spectral vector, then note that at the completion of the algorithm, the matrix has \( 2K + 2 \) rows and \( 2K + 2 \) columns. On completion of the algorithm, the second row and column are deleted, leaving \( T_x \) in its final form as a \( 2K + 1 \) by \( 2K + 1 \) matrix.
C.6 Discussion

The functionality of nonlinear circuits and systems can be viewed as a transform process: transforming an input waveform to an output waveform or, alternatively and in the steady-state, transforming a phasor vector representation of the input signal to a phasor vector representation of the output signal. This transformation view is commonly adopted in frequency-domain descriptions of nonlinear processes. (In linear circuits and systems the transformation is quite simple with the matrix operation reducing to a diagonal matrix of linear elements denoting the independence of frequency components.) The Arithmetic Operator Method (AOM) is a formal mathematics for manipulating signals described by their discrete spectra and implementing this transformation process. Coupled with an iterative procedure, a spectral balance scheme based on AOM can be realized for the frequency-domain analysis of nonlinear circuits \[111\]. Efficient analysis of nonlinear circuits is accomplished with exceptionally wide dynamic ranges \[90\] since AOM reduces to linear analysis for very small signals. AOM implements complex operations as efficient matrix-vector operations: All of the AOM operations discussed in this paper have been implemented in Matlab® and will be made available for download from the internet in the near future.

The full suite of matrix operations available in Matlab® can be utilized in system and circuit modeling. For example, consider the input-output relation

\[ y(t) = x^q(t) , \]  

where \( q \) is a non-integer. In spectral vector representation this becomes

\[ Y = T_x^{q^{-1}} X , \]  

where \( T_x^{q^{-1}} \) is a matrix raised to a non-integer power. This operation is directly supported in Matlab®. In evaluating functions in AOM it is not necessary, and generally it is neither efficient nor accurate, to use Taylor series expansions. Most analytic functions can be evaluated using standard matrix operations or established numerical algorithms. Consider the exponential function, \( e^x \), which is commonly encountered in modeling semiconductor devices. The numerically efficient approximation for the exponential is

\[ e^x = \left[ 1 + \frac{x}{2^n} + \frac{1}{2} \left( \frac{x}{2^n} \right)^2 + \cdots + \frac{1}{n} \left( \frac{x}{2^n} \right)^n \right]^{2n} , \quad n \to \infty . \]
The approximation error for \( n \geq 6 \) is less than 0.01% over the range \(-30 < x < 30\). In turn the polynomial in square brackets in (C.47) can be efficiently evaluated using the procedure described by (C.39).

In behavioral modeling of RF and microwave systems the hyperbolic tangent function is commonly used. The function also appears in modeling semiconductor devices such as MESFETs and HEMTs. With \( x(t) \) as the input signal, the hyperbolic tangent is expressed as

\[
w(t) = \tanh(x(t)) = \frac{1 - y(t)}{1 + y(t)},
\]

(C.48)

where \( y(t) = e^{-2x(t)} \). This function is readily evaluated in AOM yielding the frequency components of \( w(t) \) given the frequency components of \( x(t) \). The spectral vector \( W \) is extracted as the first column of the Spectrum Transform Matrix \( T_w \), where

\[
T_w = \frac{I - T_y}{I + T_y} = \frac{I - e^{-2T_x}}{I + e^{-2T_x}},
\]

(C.49)

where \( T_x \) is the Spectrum Transform Matrix of \( x(t) \) and \( I \) is the identity matrix with dimensions identical to \( T_y \).

### C.7 Conclusion

The mathematical basis of the Arithmetic Operator Method (AOM) for the modeling of steadystate nonlinear circuits and systems was presented. AOM enables one to describe the response of an algebraically analytic nonlinear system to a sum of sinusoidal tones using matrix-vector arithmetic, i.e. with linear algebra. Currently the focus of AOM application is behavioral modeling and system identification. The operation of AOM in behavioral modeling is particularly easy to visualize for a unilateral system model. In the frequency domain the system model, of say an amplifier, has an input with a spectral vector \( X \) and an output signal with a spectral vector \( Y \). System modeling can then can be viewed as mapping \( X \) to \( Y \) and AOM performs this transformation entirely in the frequency domain using matrix-vector operation. For multilateral systems with multiple spectral vector inputs and outputs, it is necessary to develop spectrum transform matrices for each input-output port pair.
C.8 Postscript - Conversion Factors in Fourier Transforms

The purpose of this appendix is to specify the transformation between frequency and time domain representations used in the paper. Consider two time-domain signals, \( x(t) \) and \( z(t) \), both of which are composed of a finite number of sinusoids of arbitrary frequency content. For the purposes of the development here, assume that \( x(t) \) and \( z(t) \) have spectral content at the same frequencies, although with different amplitudes and phases. This is not restrictive, since if \( x(t) \) and \( z(t) \) were composed of sinusoids of different frequencies, both could be described both using a common (i.e. union) set of frequencies with some amplitude coefficients in \( x(t) \) and \( z(t) \) both set to 0. Thus, the development that follows for \( x(t) \) will also apply to \( z(t) \). The signal \( x(t) \) can be expressed as the sum of a set of sinusoidal signals as

\[
x(t) = \sum_{n=0}^{N} x_n(t) = \sum_{n=0}^{N} |x_n| \cos(\omega_n t + \phi_n) = \sum_{n=0}^{N} |x_n| \frac{e^{j(\omega_n t + \phi_n)} + e^{-j(\omega_n t + \phi_n)}}{2},
\]

where each component is expressed as a cosine function with frequency \( \omega_n = 2\pi f_n \), amplitude \( |x_n| \), and phase \( \phi_n \). Euler’s equation \( e^{j\theta} = \cos \theta + j \sin \theta \) is now used to express the cosine function as the sum of two complex exponentials. (This form will allow use of a phasor form for the cosine function later in the development. However, it must be stressed that since a nonlinear system is being modeled, it is not mathematically correct to simply take the Fourier transform \( X(\omega) = \mathcal{F} \{ x(t) \} \) and then use the real part, \( \Re \{ X(\omega) \} \), for further analysis. The representation here is exact, but one obtains the benefit of phasor arithmetic through the formulation.) This result can be further refined as

\[
\sum_{n=0}^{N} |x_n| \frac{e^{j(\omega_n t + \phi_n)} + e^{-j(\omega_n t + \phi_n)}}{2} = \frac{1}{2} \sum_{n=0}^{N} |x_n| e^{j\phi_n} e^{j\omega_n t} + \frac{1}{2} \sum_{n=0}^{N} |x_n| e^{-j\phi_n} e^{-j\omega_n t} \\
= \frac{1}{2} \sum_{n=0}^{N} [X_{nr} + jX_{ni}] e^{j\omega_n t} + \frac{1}{2} \sum_{n=0}^{N} [X_{nr} - jX_{ni}] e^{-j\omega_n t}.
\]

(C.51)
Again Euler’s equation has been used to express the amplitude and phase as a complex number pair, \(X_{nr} \pm jX_{ni}\). The Fourier transform of \(x(t)\) is

\[
X(\omega) = \mathcal{F}[x(t)] = \mathcal{F}\left[ \frac{1}{2} \sum_{n=0}^{N} [X_{nr} + jX_{ni}] e^{j\omega_n t} + \frac{1}{2} \sum_{n=0}^{N} [X_{nr} - jX_{ni}] e^{-j\omega_n t} \right] \\
= \frac{1}{2} \sum_{n=0}^{N} [X_{nr} + jX_{ni}] [2\pi \delta(\omega - \omega_n)] + \frac{1}{2} \sum_{n=0}^{N} [X_{nr} - jX_{ni}] [2\pi \delta(\omega + \omega_n)] .
\]

(C.52)

Now, from the definition of the inverse Fourier transform,

\[
x(t) = \mathcal{F}^{-1}[X(\omega)] = \frac{1}{2\pi} \int_{-\infty}^{+\infty} X(\omega) e^{j\omega t} d\omega \\
= \frac{1}{2\pi} \int_{-\infty}^{+\infty} \left\{ \frac{1}{2} \sum_{n=0}^{N} [X_{nr} + jX_{ni}] [2\pi \delta(\omega - \omega_n)] + \frac{1}{2} \sum_{n=0}^{N} [X_{nr} - jX_{ni}] [2\pi \delta(\omega + \omega_n)] \right\} e^{j\omega t} d\omega \\
= \frac{1}{2\pi} (2\pi) \int_{-\infty}^{+\infty} \left\{ \frac{1}{2} \sum_{n=0}^{N} [X_{nr} + jX_{ni}] [\delta(\omega - \omega_n)] + \frac{1}{2} \sum_{n=0}^{N} [X_{nr} - jX_{ni}] [\delta(\omega + \omega_n)] \right\} e^{j\omega t} d\omega \\
= \int_{-\infty}^{+\infty} \left\{ \frac{1}{2} \sum_{n=0}^{N} [X_{nr} + jX_{ni}] [\delta(\omega - \omega_n)] + \frac{1}{2} \sum_{n=0}^{N} [X_{nr} - jX_{ni}] [\delta(\omega + \omega_n)] \right\} e^{j\omega t} d\omega .
\]

(C.53)

Note that a factor of \(2\pi\) that appears when applying the strict definition of the Fourier transform. This same factor is then eliminated in the inverse Fourier transform. In this paper this domain conversion factor will be ignored, since the ultimate result will be a time domain signal, and the adoption enables simplified interpretation of the frequency components as phasors.

### C.9 Postscript - Three Tone AOM

The development of convolution as matrix-vector operations is considered here by way of an example for signals \(x(t)\) and \(z(t)\) with a DC component (\(\omega_0 = 0\)) and three non-commensurate tones with frequencies \(f_1\), \(f_2\), and \(f_3\) corresponding to the radian frequencies \(\omega_1\), \(\omega_2\), and \(\omega_3\). The double sided spectrum of this signal is shown in Figure 3. This example demonstrates the representation of convolution as a matrix operation and that it is not restricted to circular convolution. This representation is fundamental to AOM. (A two-tone example would be barely
distinguishable from circular convolution.) In the frequency domain the signals are represented as

\[ DX(\omega) = X_0\delta(\omega) + \frac{1}{2} \sum_{n=1}^{3} \left[ X_n\delta(\omega - \omega_n) + X_{-n}\delta(\omega - \omega_{-n}) \right], \tag{C.54} \]

and

\[ DZ(\omega) = Z_0\delta(\omega) + \frac{1}{2} \sum_{n=1}^{3} \left[ Z_n\delta(\omega - \omega_n) + Z_{-n}\delta(\omega - \omega_{-n}) \right]. \tag{C.55} \]

The product of the two signals, \( y(t) = x(t)z(t) \), becomes a convolution operation in the frequency domain:

\[ DY(\omega) = DX(\omega) * DZ(\omega) \]

\[ = \sum_{\text{All } \omega_k} DX(\omega_k)DZ(\omega - \omega_k) \]

\[ = \sum_{\text{All } \omega_k} \left[ \begin{array}{c}
\frac{1}{2}X_{-3}\delta(\omega_k - \omega_{-3}) + \frac{1}{2}X_{-2}\delta(\omega_k - \omega_{-2}) + \\
\frac{1}{2}X_{-1}\delta(\omega_k - \omega_{-1}) + X_0\delta(\omega_k) + \\
\frac{1}{2}X_1\delta(\omega_k - \omega_1) + \frac{1}{2}X_2\delta(\omega_k - \omega_2) + \\
\frac{1}{2}X_3\delta(\omega_k - \omega_3)
\end{array} \right] \]

\[ = \sum_{\text{All } \omega_k} \left[ \begin{array}{c}
\frac{1}{2}Z_{-3}\delta[\omega - (\omega_k - \omega_{-3})] + \frac{1}{2}Z_{-2}\delta[\omega - (\omega_k - \omega_{-2})] + \\
\frac{1}{2}Z_{-1}\delta[\omega - (\omega_k - \omega_{-1})] + Z_0\delta(\omega - \omega_k) + \\
\frac{1}{2}Z_1\delta[\omega - (\omega_k - \omega_1)] + \frac{1}{2}Z_2\delta[\omega - (\omega_k - \omega_2)] + \\
\frac{1}{2}Z_3\delta[\omega - (\omega_k - \omega_3)]
\end{array} \right] \tag{C.56} \]

The components of \( DY(\omega) \) in (C.56) for each \( \omega_k \) are denoted as \( DY(\omega)|_k \) and these are given as follows in (C.57)–(C.63) on pages 400–402, respectively.
\[ D_Y(\omega) |_{\kappa=3} = \frac{1}{2} X_{-3} \left\{ \begin{array}{l} \frac{1}{2} Z_{-3} \delta [\omega - (\omega_3 + \omega_3)] + \frac{1}{2} Z_{-2} \delta [\omega - (\omega_3 + \omega_2)] + \\
\frac{1}{2} Z_{-1} \delta [\omega - (\omega_3 + \omega_1)] + Z_0 \delta (\omega - \omega_3) + \\
\frac{1}{2} Z_1 \delta [\omega - (\omega_3 + \omega_1)] + \frac{1}{2} Z_2 \delta [\omega - (\omega_3 + \omega_2)] + \\
\frac{1}{2} Z_3 \delta [\omega - (\omega_3 + \omega_3)] \\
\end{array} \right\} \\
= \frac{1}{2} X_{-3} Z_{-3} \delta [\omega - 2\omega_3] + \frac{1}{4} X_{-3} Z_{-2} \delta [\omega - (\omega_3 + \omega_2)] + \\
\frac{1}{4} X_{-3} Z_{-1} \delta [\omega - (\omega_3 + \omega_1)] + \frac{1}{2} X_{-3} Z_0 \delta (\omega - \omega_3) + \\
\frac{1}{4} X_{-3} Z_1 \delta [\omega - (\omega_3 + \omega_1)] + \frac{1}{4} X_{-3} Z_2 \delta [\omega - (\omega_3 + \omega_2)] + \\
\frac{1}{4} X_{-3} Z_3 \delta (\omega) \quad (C.57) \]

\[ D_Y(\omega) |_{\kappa=2} = \frac{1}{2} X_{-2} \left\{ \begin{array}{l} \frac{1}{2} Z_{-3} \delta [\omega - (\omega_2 + \omega_3)] + \frac{1}{2} Z_{-2} \delta [\omega - (\omega_2 + \omega_2)] + \\
\frac{1}{2} Z_{-1} \delta [\omega - (\omega_2 + \omega_1)] + Z_0 \delta (\omega - \omega_2) + \\
\frac{1}{2} Z_1 \delta [\omega - (\omega_2 + \omega_1)] + \frac{1}{2} Z_2 \delta [\omega - (\omega_2 + \omega_2)] + \\
\frac{1}{2} Z_3 \delta [\omega - (\omega_2 + \omega_3)] \\
\end{array} \right\} \\
= \frac{1}{2} X_{-2} Z_{-3} \delta [\omega - (\omega_2 + \omega_3)] + \frac{1}{4} X_{-2} Z_{-2} \delta [\omega - 2\omega_2] + \\
\frac{1}{4} X_{-2} Z_{-1} \delta [\omega - (\omega_2 + \omega_1)] + \frac{1}{2} X_{-2} Z_0 \delta (\omega - \omega_2) + \\
\frac{1}{4} X_{-2} Z_1 \delta [\omega - (\omega_2 + \omega_1)] + \frac{1}{4} X_{-2} Z_2 \delta (\omega) + \\
\frac{1}{4} X_{-2} Z_3 \delta [\omega - (\omega_2 + \omega_3)] \quad (C.58) \]

\[ D_Y(\omega) |_{\kappa=1} = \frac{1}{2} X_{-1} \left\{ \begin{array}{l} \frac{1}{2} Z_{-3} \delta [\omega - (\omega_1 + \omega_3)] + \frac{1}{2} Z_{-2} \delta [\omega - (\omega_1 + \omega_2)] + \\
\frac{1}{2} Z_{-1} \delta [\omega - (\omega_1 + \omega_1)] + Z_0 \delta (\omega - \omega_1) + \\
\frac{1}{2} Z_1 \delta [\omega - (\omega_1 + \omega_1)] + \frac{1}{2} Z_2 \delta [\omega - (\omega_1 + \omega_2)] + \\
\frac{1}{2} Z_3 \delta [\omega - (\omega_1 + \omega_3)] \\
\end{array} \right\} \\
= \frac{1}{2} X_{-1} Z_{-3} \delta [\omega - (\omega_1 + \omega_3)] + \frac{1}{4} X_{-1} Z_{-2} \delta [\omega - (\omega_1 + \omega_2)] + \\
\frac{1}{4} X_{-1} Z_{-1} \delta [\omega - 2\omega_1] + \frac{1}{2} X_{-1} Z_0 \delta (\omega - \omega_1) + \\
\frac{1}{4} X_{-1} Z_1 \delta (\omega) + \frac{1}{4} X_{-1} Z_2 \delta [\omega - (\omega_1 + \omega_2)] + \\
\frac{1}{4} X_{-1} Z_3 \delta [\omega - (\omega_1 + \omega_3)] \quad (C.59) \]
\[ D\dot{Y}(\omega)_{|k=0} = X_0 \left\{ \begin{array}{l} \frac{1}{2} Z_{-3} \delta[\omega - (\omega_1 + \omega_3)] + \frac{1}{2} Z_{-2} \delta[\omega - (\omega_1 + \omega_2)] + \\ \frac{1}{2} Z_{-1} \delta[\omega - (\omega_1 + \omega_1)] + Z_0 \delta(\omega - \omega_1) + \\ \frac{1}{2} Z_1 \delta[\omega - (\omega_1 + \omega_1)] + \frac{1}{2} Z_2 \delta[\omega - (\omega_1 + \omega_2)] + \\ \frac{1}{2} Z_3 \delta[\omega - (\omega_1 + \omega_3)] \end{array} \right\} \]

\[ = \frac{1}{2} X_0 Z_{-3} \delta[\omega - (\omega_0 + \omega_3)] + \frac{1}{2} X_0 Z_{-2} \delta[\omega - (\omega_0 + \omega_2)] + \\ \frac{1}{2} X_0 Z_{-1} \delta[\omega - (\omega_0 + \omega_1)] + X_0 Z_0 \delta(\omega - \omega_0) + \\ \frac{1}{2} X_0 Z_1 \delta(\omega - \omega_1) + \frac{1}{2} X_0 Z_2 \delta(\omega - \omega_2) + \\ \frac{1}{2} X_0 Z_3 \delta(\omega - \omega_3) \]  

(C.60)

\[ D\dot{Y}(\omega)_{|k=1} = \frac{1}{2} X_1 \left\{ \begin{array}{l} \frac{1}{2} Z_{-3} \delta[\omega - (\omega_1 + \omega_3)] + \frac{1}{2} Z_{-2} \delta[\omega - (\omega_1 + \omega_2)] + \\ \frac{1}{2} Z_{-1} \delta[\omega - (\omega_1 + \omega_1)] + Z_0 \delta(\omega - \omega_1) + \\ \frac{1}{2} Z_1 \delta[\omega - (\omega_1 + \omega_1)] + \frac{1}{2} Z_2 \delta[\omega - (\omega_1 + \omega_2)] + \\ \frac{1}{2} Z_3 \delta[\omega - (\omega_1 + \omega_3)] \end{array} \right\} \]

\[ = \frac{1}{4} X_1 Z_{-3} \delta[\omega - (\omega_1 + \omega_3)] + \frac{1}{4} X_1 Z_{-2} \delta[\omega - (\omega_1 + \omega_2)] + \\ \frac{1}{4} X_1 Z_{-1} \delta(\omega + \omega_1) + \frac{1}{4} X_1 Z_0 \delta(\omega - \omega_1) + \frac{1}{4} X_1 Z_1 \delta(\omega - 2\omega_1) + \\ \frac{1}{4} X_1 Z_2 \delta[\omega - (\omega_1 + \omega_2)] + \frac{1}{4} X_1 Z_3 \delta[\omega - (\omega_1 + \omega_3)] \]  

(C.61)

\[ D\dot{Y}(\omega)_{|k=2} = \frac{1}{2} X_2 \left\{ \begin{array}{l} \frac{1}{2} Z_{-3} \delta[\omega - (\omega_2 + \omega_3)] + \frac{1}{2} Z_{-2} \delta[\omega - (\omega_2 + \omega_2)] + \\ \frac{1}{2} Z_{-1} \delta[\omega - (\omega_2 + \omega_1)] + Z_0 \delta(\omega - \omega_2) + \\ \frac{1}{2} Z_1 \delta[\omega - (\omega_2 + \omega_1)] + \frac{1}{2} Z_2 \delta[\omega - (\omega_2 + \omega_2)] + \\ \frac{1}{2} Z_3 \delta[\omega - (\omega_2 + \omega_3)] \end{array} \right\} \]

\[ = \frac{1}{4} X_2 Z_{-3} \delta[\omega - (\omega_2 + \omega_3)] + \frac{1}{4} X_2 Z_{-2} \delta(\omega + \omega_2) + \frac{1}{4} X_2 Z_{-1} \delta[\omega - (\omega_2 + \omega_1)] + \\ \frac{1}{4} X_2 Z_0 \delta[\omega - \omega_2] + \frac{1}{4} X_2 Z_1 \delta[\omega - (\omega_2 + \omega_1)] + \frac{1}{4} X_2 Z_2 \delta(\omega - 2\omega_2) + \\ \frac{1}{4} X_2 Z_3 \delta[\omega - (\omega_2 + \omega_3)] \]  

(C.62)
\[ D Y(\omega) |_{k=3} = \frac{1}{2} X_3 \left\{ \begin{array}{l}
\frac{1}{2} Z_{-3} \delta[\omega - (\omega_3 + \omega_{-3})] + \frac{1}{2} Z_{-2} \delta[\omega - (\omega_3 + \omega_{-2})] + \\
\frac{1}{2} Z_{-1} \delta[\omega - (\omega_3 + \omega_{-1})] + Z_0 \delta(\omega - \omega_{-3}) + \\
\frac{1}{2} Z_1 \delta[\omega - (\omega_3 + \omega_1)] + \frac{1}{2} Z_2 \delta[\omega - (\omega_3 + \omega_2)] + \\
\frac{1}{2} Z_3 \delta[\omega - (\omega_3 + \omega_3)]
\end{array} \right\} \\
= \frac{1}{4} X_3 Z_{-3} \delta(\omega) + \frac{1}{4} X_3 Z_{-2} \delta[\omega - (\omega_3 + \omega_{-2})] + \frac{1}{4} X_3 Z_{-1} \delta[\omega - (\omega_3 + \omega_{-1})] + \\
\frac{1}{4} X_3 Z_0 \delta(\omega - \omega_3) + \frac{1}{4} X_3 Z_1 \delta[\omega - (\omega_3 + \omega_1)] + \frac{1}{4} X_3 Z_2 \delta[\omega - (\omega_3 + \omega_2)] + \\
\frac{1}{4} X_3 Z_3 \delta(\omega - 2\omega_3) .
\] (C.63)

The domain of the output \( D Y(\omega) \) can be described in set form as the set of frequencies containing non-zero convolution products:

\[ D \omega = \{ 2\omega_{-3}, \omega_{-3} + \omega_{-2}, \omega_{-3} + \omega_{-1}, \omega_{-3} + \omega_{-1} + \omega_1, \omega_{-3} + \omega_1, 2\omega_{-2}, \omega_{-2} + \omega_{-1}, \omega_{-2}, \\
\omega_{-2} + \omega_1, 2\omega_{-1}, \omega_{-1} + \omega_0, \omega_1, 2\omega_1, \omega_2 + \omega_{-1}, \omega_2, \omega_2 + \omega_1, 2\omega_2, \omega_3 + \omega_{-2}, \omega_3 + \omega_{-1}, \\
\omega_3, \omega_3 + \omega_1, \omega_3 + \omega_2, 2\omega_3 \} \] (C.64)

which contains both positive and negative frequencies in addition to DC and relates to the double-sided spectrum denoted with the leading superscript \( D \). It is now possible to express \( D Y(\omega) \) as a vector, termed a double-sided spectral vector, using the set \( D \omega \) as its basis. Col-
lecting components of the same frequency in Equations (C.57) to (C.63) yields:

\[
\begin{bmatrix}
Y(2\omega_3) \\
Y(\omega_3 + \omega_2) \\
Y(\omega_3 + \omega_1) \\
Y(\omega_3) \\
Y(\omega_3 + \omega_1) \\
Y(\omega_3 + \omega_2) \\
Y(2\omega_2) \\
Y(\omega_2 + \omega_1) \\
Y(\omega_2) \\
Y(\omega_2 + \omega_1) \\
Y(2\omega_1) \\
Y(\omega_1) \\
\end{bmatrix}
= \begin{bmatrix}
\frac{1}{4}X_3Z_3 \\
\frac{1}{4}X_3Z_2 + \frac{1}{4}X_2Z_3 \\
\frac{1}{4}X_3Z_1 + \frac{1}{4}X_1Z_3 \\
\frac{1}{4}X_3Z_0 + \frac{1}{4}X_0Z_3 \\
\frac{1}{4}X_3Z_1 + \frac{1}{4}X_1Z_3 \\
\frac{1}{4}X_3Z_2 + \frac{1}{4}X_2Z_3 \\
\frac{1}{4}X_2Z_2 \\
\frac{1}{4}X_2Z_1 + \frac{1}{4}X_1Z_2 \\
\frac{1}{4}X_2Z_0 + \frac{1}{4}X_0Z_2 \\
\frac{1}{4}X_2Z_1 + \frac{1}{4}X_1Z_2 \\
\frac{1}{4}X_2Z_2 \\
\frac{1}{4}X_2Z_1 + \frac{1}{4}X_1Z_2 \\
\frac{1}{4}X_2Z_0 + \frac{1}{4}X_0Z_2 \\
\frac{1}{4}X_3Z_2 + \frac{1}{4}X_2Z_3 \\
\frac{1}{4}X_3Z_1 + \frac{1}{4}X_1Z_3 \\
\frac{1}{4}X_3Z_2 + \frac{1}{4}X_2Z_3 \\
\frac{1}{4}X_3Z_3 \\
\end{bmatrix}
\]

(C.65)

If \( \frac{D}{C}Z(\omega) \) is constructed as a double-sided spectral vector using the set of output frequencies \( D_\omega \) as its basis, then using the three-tone input of Figure 2(b) (without intermodulation com-
ponents) this becomes:

\[
\begin{bmatrix}
Z(2\omega_3) \\
Z(\omega_3 + \omega_2) \\
Z(\omega_3 + \omega_1) \\
Z(\omega_3) \\
Z(\omega_3 + \omega_1) \\
Z(\omega_3 + \omega_2) \\
Z(\omega_2 + \omega_1) \\
Z(\omega_2) \\
Z(\omega_2 + \omega_1) \\
Z(\omega_2) \\
Z(\omega_3 + \omega_2) \\
Z(\omega_3 + \omega_1) \\
Z(\omega_3) \\
Z(\omega_3 + \omega_1) \\
Z(\omega_3 + \omega_2) \\
Z(\omega_3) \\
Z(\omega_3)
\end{bmatrix}
= 
\begin{bmatrix}
0 \\
0 \\
0 \\
\frac{1}{2}Z_3 \\
0 \\
0 \\
0 \\
\frac{1}{2}Z_2 \\
0 \\
0 \\
0 \\
0 \\
\frac{1}{2}Z_3 \\
0 \\
0 \\
0 \\
0 \\
0
\end{bmatrix}
\]

To generalize the notions of spectral vectors and conformable convolution matrices, consider the three-tone double-sided spectrum of Figure C.4 below resulting from the three incommensurate tones \(\omega_1, \omega_2, \omega_3\). Let the phasors \(X_n\) and \(Z_n\) correspond to the component at frequency \(f_n\).
Then the generalized spectral vector $\frac{D}{C}Z$ is:

$$\frac{D}{C}Z(\omega) = \frac{1}{2} \begin{bmatrix} Z_{-12} & Z_{-11} & Z_{-10} & Z_{-9} & Z_{-8} & Z_{-7} & Z_{-6} & Z_{-5} & Z_{-4} & 2Z_0 \\ Z_{4} & Z_{5} & Z_{1} & Z_{-2} & Z_{3} & Z_{6} & Z_{7} & Z_{8} & Z_{9} & Z_{10} & Z_{11} & Z_{12} \end{bmatrix}^T.$$  
(C.67)

Also, a conformable convolution matrix, called a transformation matrix here because of its general applicability, $\frac{D}{C}T_x$, is given in (C.9) on page 406.
$$D T_x = $$

$$\begin{array}{c|cccccccccccc}
\text{T} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{C} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
X_0 & 2X_0 & X_1 & X_2 & X_3 & X_4 & X_5 & X_6 & X_7 & X_8 & X_9 & X_{10} & X_{11} & X_{12} \\
X_1 & 2X_1 & X_2 & X_3 & X_4 & X_5 & X_6 & X_7 & X_8 & X_9 & X_{10} & X_{11} & X_{12} \\
X_2 & X_3 & X_4 & X_5 & X_6 & X_7 & X_8 & X_9 & X_{10} & X_{11} & X_{12} \\
X_3 & X_4 & X_5 & X_6 & X_7 & X_8 & X_9 & X_{10} & X_{11} & X_{12} \\
X_4 & X_5 & X_6 & X_7 & X_8 & X_9 & X_{10} & X_{11} & X_{12} \\
X_5 & X_6 & X_7 & X_8 & X_9 & X_{10} & X_{11} & X_{12} \\
X_6 & X_7 & X_8 & X_9 & X_{10} & X_{11} & X_{12} \\
X_7 & X_8 & X_9 & X_{10} & X_{11} & X_{12} \\
X_8 & X_9 & X_{10} & X_{11} & X_{12} \\
X_9 & X_{10} & X_{11} & X_{12} \\
X_{10} & X_{11} & X_{12} \\
X_{11} & X_{12} \\
X_{12} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}$$

(C.68)
The pattern in (C.9) can be followed to develop $D_C T_x$ for larger problems. (An algorithm for filling this matrix is presented in the body of the paper for a single-sided spectrum.) With the matrix construction $D_C T_x$ describing $D_C X(D_\omega)$, then $y(t) = x(t) z(t)$ is represented by the matrix operation:

$$D_C Y = D_C T_x D_C Z.$$  \hfill (C.69)

### C.10 Postscript - Automatic Development of BIPD Table Using Frequency Truncation

The Basic Intermodulation Product Description (BIPD) table is a list of frequency indices that defines the anticipated frequency domain of the output signal $y(t)$. Along with DC, the entries in the BIPD table form the radian frequency set $S_\omega$. That is,

$$S_\omega = \{ \text{DC and all entries in the BIPD table} \}.$$  \hfill (C.70)

The BIPD table can be generated automatically using, for example, a triangular truncation scheme. The frequency selection scheme proceeds as follows: For a choice of $n$ frequencies and an arbitrary nonlinear order, $N$, the BIPD table is generated by creating a table of $n$ columns (denoted $n_1, n_2, \ldots, n_n$, with one column for each input frequency) where each row in the table contains a unique combination of index numbers (coefficient weightings) for each frequency $f_1, f_2, \ldots, f_n$. Valid entries in the table are subject to the constraint that the sum of the absolute values of the index numbers across the row, denoted as $|n|$, is less than or equal to $N$, the order of nonlinearity. If each row of the table is designated with the frequency index $k_x$, then the index number entries in each column $n_i$ within each row are designated $n_i^{(k_x)}$. Thus the constraint on the sum of absolute values of index numbers across a row can be stated as:

$$\sum_{i=1}^{n} |n_i^{(k_x)}| = |n| \leq N.$$  \hfill (C.71)

A second constraint is that the sum of weighted frequency products in each row must be single-sided with the choice of non-negative frequencies being most convenient. Thus, the following relationship holds:

$$\sum_{i=1}^{n} n_i^{(k_x)} \cdot f_i \geq 0.$$  \hfill (C.72)
The construction of a BIPD table is thus a combinatorial problem subject to the constraints given in \((C.71)\) and \((C.72)\). Often the input signal is a non-commensurate two-tone signal, so this will be used to illustrate the construction of a BIPD table. Consider the case of a down-conversion mixer where a narrowband RF signal and a local oscillator signal are to be mixed through an intentional nonlinear transfer function that is known to have non-linear behavior up to the third order. Let the input signal have the form \(x(t) = X_1 \cos \omega_1 t + X_2 \cos \omega_2 t\), assuming that \(\omega_2 > \omega_1\) (with \(\omega_1 = 2\pi f_1\) and \(\omega_2 = 2\pi f_2\)), and let the mixer have a nonlinear behavior conforming to the polynomial model \(y = a_0 + a_1 x + a_2 x^2 + a_3 x^3\), so for this example \(n = 2\) and \(N = 3\).

Using the two input frequencies and the constraints of \((C.71)\) and \((C.72)\), the BIPD table, Table \(C.4\), is constructed. By convention, the first few entries in the BIPD table correspond to the incommensurate excitation tones. (Note that for these entries \(|n| = 1\) and \(k_x = i\), where \(i\) is the subscript for \(n\) in the column containing index number 1.) Thus there are 12 non-zero output frequencies of interest in this example: \(\omega_1, \omega_2, \omega_2 - \omega_1, \omega_2 + \omega_1, 2\omega_1 - \omega_2, 2\omega_2 - \omega_1, 2\omega_1, 2\omega_2, 2\omega_1 + \omega_2, \omega_1 + 2\omega_2, 3\omega_1,\) and \(3\omega_2\).

Table \(C.4\): BIPD Table for 3rd Order Non-Linear Transfer Function

| \(k_y\) \(k_x, k_z\) | \(|n|\) | \(n_1\) | \(n_2\) |
|-----------------|-----|-----|-----|
| 1 \((1, 1)\)     | 1   | 1   | 0   |
| 2 \((1, 0)\)     | 1   | 0   | 1   |
| 3 \((2, -1)\)    | 2   | -1  | 1   |
| 4 \((2, 1)\)     | 2   | 1   | 1   |
| 5 \((2, 2)\)     | 2   | 2   | 0   |
| 6 \((2, 0)\)     | 2   | 0   | 2   |
| 7 \((3, 2)\)     | 3   | 2   | -1  |
| 8 \((3, -1)\)    | 3   | -1  | 2   |
| 9 \((3, 2)\)     | 3   | 2   | 1   |
| 10 \((3, 1)\)    | 3   | 1   | 2   |
| 11 \((3, 3)\)    | 3   | 3   | 0   |

*Continued on next page*
Table C.4 (continued)

<table>
<thead>
<tr>
<th>$k_y$</th>
<th>$n$</th>
<th>$n_1$</th>
<th>$n_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_x, k_z$</td>
<td>12</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>
Appendix D

AOM Toolbox User’s Guide

D.1 Introductory Remarks

This appendix is intended to furnish an overview of the operation of the AOM Toolbox through an illustrative example of second-order mixing. The intention of this section is to serve as a guide to establish familiarity with the mechanics of using the AOM Toolbox for the new user, not as a comprehensive reference that will define the bounds of the suitability of the AOM Toolbox for analyzing specific problems. It should also be understood that the AOM Toolbox comes with no warranty whatsoever.

D.2 Installation

A Matlab® Version 7 environment is required to use the AOM Toolbox. The AOM Toolbox was developed and refined over a period of time as the required problem sizes were scaled up, ultimately reaching the limits of existing computing platforms. Compressed archives are available for installation on two platforms: Windows (.zip form) and Linux (.tar form). Mac users should choose the compressed archive form most suitable to their operating platform. (None of the AOM Toolbox was developed on a Mac environment, so its use there is untested.) Since the exact internet location of the AOM Toolbox archive files may change from time to time, the archive files may be found by using an internet search engine that searches World Wide
Web content. Use the search terms “Hart” and “AOM Toolbox” to find the archive files.

The archive files should be uncompressed and installed in a folder or directory which is then added to the Matlab® path. On Linux systems, the appropriate directory to place the tar file is /home/userid/matlab, and running the Linux command tar -zxvf aom_tb.tar.gz will automatically create the subdirectory ./aom_tb and install the toolbox m-files in the subdirectory. On Windows platforms, the appropriate folder will generally be C:\MATLAB or a variation depending upon the particular release.

Before the first use of the AOM Toolbox, it will be necessary to compile two small Matlab® Executable (MeX) files for manipulating matrices of 8-bit integer elements (something not available in Matlab®) which are written in C. These two MeX files are called by the AOM Toolbox toolbox function aom_vfd, and attempts to use the AOM Toolbox before compilation will cause aom_vfd to issue an error message. If the user’s platform has never been used to do compilation of MeX files, it will be necessary to run the mex -setup command from the Matlab® command line and select a C compiler. Most Linux installations will have the Gnu Compiler Collection (GCC) available, while many Windows installations will have the Matlab® LCC compiler available. (The code may be compiled on another user’s platform, if necessary, but the platform on which the compiler takes place should have the same operating system and use a release of Matlab® Version 7.) After completing the mex -setup, change directories to the ./aom_tb directory on Linux or the \aom_tb on Windows and issue the commands mex aom_cperml.c followed by mex aom_matmuli.c to compile the C codes. After compilation, look for executable files. For example, on a 64-bit Linux platform, look for files with extension .mexa64, and on a 32-bit Windows platform, look for files with extension or .dll.

D.3 Usage Overview

The AOM Toolbox is designed to manipulate discrete spectra in the frequency-domain by implementing frequency-domain convolutional techniques on (time-domain) polynomial transfer functions. That is, given a time-domain nonlinear transfer function expressible in polynomial form, the AOM Toolbox implements the complimentary operation (convolution) in the frequency-domain. The AOM Toolbox achieves the end result through a divide and conquer approach to modeling the nonlinear transfer function in the frequency domain. The toolbox heavily exploits the sparse matrix handling capability of Matlab® to achieve its ends.
Figure D.1 on the following page shows a generic flow-chart of the steps required to analyze a problem using the AOM Toolbox with direct analysis methods. Further details on each of the boxes of the flow-chart:

- **Input Signal Description and Nonlinear Transfer Function**: Define the input signal amplitude, phase, and frequency vectors. The signal must be discrete, and the typical transfer function will be a polynomial.

- **Compute Vector Frequency Description Table**: Call the toolbox function `aom_vfd` (see Listing F.1 in Section F.2.1 on page 446), which takes as input the vector of input frequencies, the order of nonlinearity (or spectral truncation, when it differs from the order of nonlinearity of the transfer function), and whether the VFD Table is to be one-sided or two-sided and returns the VFD Table and its associated frequency index vector.

- **Create Input Spectral Vector**: Call the toolbox function `aom_makespv` (see Listing F.5 in Section F.2.5 on page 463) which takes as input the user-defined amplitude and phase vectors, the one-sided or two-sided parameter, and whether the vector elements are to be real or complex and returns a spectral vector of phasors in the specified form with the input amplitudes and phases seeded at the appropriate frequency index locations.

- **Compute Spectrum Mapping Table**: Call the function `aom_sspecmap` (see Listing F.6 in Section F.2.6 on page 466), which takes as input the VFD Table, the one-sided or two-sided parameter, the frequency index vector, and (optionally) the input spectral vector and returns a sparse spectrum mapping table that exploits the sparsity of the spectral vector (i.e. no entries are made in the table for atomic convolution products giving a zero result). If an empty matrix is passed for the spectral vector, the code assumes non-zero spectral content only in the phasor locations corresponding to the frequency index vector locations between 1 and $Q$, i.e. only for the input amplitudes and phases.

- **Compute Spectrum Transform Matrix**: Call the toolbox function `aom_sspctrans` (see Listing F.7 in Section F.2.7 on page 476), which takes as input the input spectral vector, the spectral vector, and returns the spectrum transform matrix.

- **Evaluate Transfer Function**: Call the toolbox function `aom_sblockeval` (see Listing F.8 in Section F.2.8 on page 479), which takes as input the input spectral vector, the
Figure D.1: Typical analysis flow for solving a problem with the AOM Toolbox.
spectrum transform matrix, and the transfer function, and returns a 2-dimensional array of output vectors corresponding to the output spectral vector at each order of nonlinearity in the transfer function.

- **Output Results** Perform any further manipulation required on the output spectral vectors and plot or store results. This is step is outside the scope of the AOM Toolbox implementation and is problem-specific.

## D.4 Application Example

### D.4.1 Problem Description

A specific example of using the AOM Toolbox will be given by considering the following textbook problem, where portions have been redacted so that the problem may be used for illustrative purposes here while retaining instructional value to its author:

The system in Figure [D.2] is a superheterodyne receiver that converts an information carrying signal at the RF (or radio frequency) to a much lower frequency IF for further processing. The frequency conversion is done in two stages as this results in better noise performance and relaxes requirements on filtering. The IF will also contain the same information as the original signals. The RF contains two signals at 504.5 MHz and 504.4 MHz of equal amplitude — 1 V. The first mixer has a 500 MHz LO (local oscillator) and the second mixer has a 5 MHz LO; both mixers have magnitude 2 V. Specify filters to suppress nonlinear distortion in this system. The filters should eliminate intermodulation components at the IF and maximize the linear response between the 504.5 MHz RF and the 500 kHz IF, and between the 504.4 MHz RF and the 600 kHz IF. You will need to develop intermod tables...

These mixers are microwave mixers and usually at higher frequencies the RF and LO are combined before being applied to the nonlinear element. So the model to use for each mixer is that given in Figure [D.3]. Here, for both mixers the RF and LO are summed and then applied to a nonlinear block.

Mixer A is described by

\[ y = x + 0.35x^2 \]  \hspace{1cm} (D.1)

while Mixer B is described by

\[ z = 0.5 + y + 0.5y^2, \]  \hspace{1cm} (D.2)

and for illustrative purposes the output of Mixer A will be passed through an ideal bandpass filter (also in the form of an AOM Toolbox function) which will be specified. The input to
Mixer A will be termed $X$, the output from Mixer A (and subsequent input to Mixer B) will be termed $Y$, and the output from Mixer B will be termed $Z$. No filter will be specified for the output of Mixer B, although little imagination will be required to specify one of the ideal bandpass variety. In the sections that follow, portions of the Matlab® script used to analyze the problem will be presented, and the results of using them shown. The beginning portion of the script is shown in Listing D.1, which furnishes some introductory comments and sets two global variables dictating that one-sided real-valued spectral vectors will be used.

Listing D.1: Comment lines and global variables in AOM Toolbox mixer example.

1 % AOM implementation of Exercise 6.10 of ECE718 notes, Ca 2001.
2 % Mixer A with y = x + 0.35*x^2 for two RF tones (@ 504.4 and 504.5 MHz)
3 % of 1 V amplitude and a mixer signal (@ 500 MHz) of 2 V amplitude.
4 % y is the IF output. Bandpass filter it and use it as an input to
5 % Mixer B where z = 0.5 + y + 0.5y^2 for y and another mixer signal
6 % (@ 5 MHz) of amplitude 2 V.
7 % (c) Frank P. Hart, 2008
8 clear variables; clc;
D.4.2 Analysis of Mixer A

The analysis will proceed by considering Mixer A’s inputs and nonlinear function first. The transfer function in (D.1) is a 2\textsuperscript{nd} order nonlinearity and this specified in Listing D.2 by variable \texttt{MixA\_ost} on line 15. The input consists of the sum of three discrete sinusoids and so it meets the requirement for using the AOM Toolbox. The input frequency and amplitude vectors will be constructed such that \( f_1 = 500 \) MHz, \( f_2 = 504.4 \) MHz, and \( f_3 = 504.5 \) MHz. The input frequencies are specified in the vector variable \texttt{fXin\_A} on line 19, the corresponding amplitudes are shown in the vector \texttt{Xin\_A} on line 18, and the corresponding phases in the vector \texttt{Xin\_phid}. The number of input tones is certainly determined by length of the previous three vectors, but is also checked by the input variable \texttt{Xin\_nt} on line 16. A final input, \texttt{Xin\_dc} on line 17, specifies the DC content, if any, that is to be used to construct the signal. Usually, this is set to zero since input circuits to real-world devices are usually capacitively-coupled to block DC.

Listing D.2: Define input signal and transfer function for Mixer A.

\begin{verbatim}
14 \% Define the Mixer A input...
15 MixA\_ost = 2; \% Mixer A order of spectral truncation (nonlinearity)
16 Xin\_nt = 3; \% Number of input tones
17 Xin\_dc = 0.0; \% No DC on input
18 Xin\_A = [2 1 1]; \% 2 V @ 500 LO, 1 V on RF signals
19 fXin\_A = [500 504.4 504.5]*1e6;
20 Xin\_phid = [0 0 0]; \% Assume 0-degree phase
21 A\_coeff = [0.35 1 0]; \% y = 0.35*x^2 + 1.0*x + 0;
\end{verbatim}

The transfer function in (D.1) is a 2\textsuperscript{nd} order nonlinearity, and so the VFD Table will be constructed using an order of spectral truncation equal to 2. Listing D.3 shows the lines of Matlab\textsuperscript{®} code that call the functions to create the VFD Table (lines 24–25) and spectral vector (line 26), and a separate line (line 27) of code to create a vector of the numeric frequencies.
Listing D.3: Call the functions to create the VFD Table and spectral vector.

```matlab
23 % Call the AOM routines for Mixer A...
24 [Xht_nvfd, Xhash_tables, X_nvfd, X_ffix, X_fvfd, X_ffix, X_ntof, X_fton] ... = aom_vfd(Xinnt, MixAost, fXinA, G_num_sides, 'X');
25 Xin_sv = aom_makespv(Xin_A, Xin_dc, Xin_phid, X_ffix, G_spvec_kind);
26 X_ffreq = cast(X_fvfd, 'double')*fXin_A';
```

The call to `aom_vfd` on lines 24–25 of Listing D.3 passes variables defined in Listing D.2 as inputs and returns several variables as outputs. The contents of output variables `Xht_nvfd` and `X_hash_tables` are the positive portion of the VFD Table in 1-norm sorted form and the hash tables corresponding to the vector index locations in that 1-norm sorted form of the VFD Table. These two variables are keys to facilitating the fast and accurate construction of the spectrum mapping table. The variables `X_nvfd` and `X_ffix` are the 1-norm sorted VFD Table and its corresponding frequency index vector. These may differ from the table used for hashing in that the hash table is always 1-sided (and only corresponds to positive frequencies) while `X_nvfd` may be two-sided if `G_num_sides` is set to 2. The variables `X_fvfd` and `X_ffix` are the frequency-sorted VFD Table and its corresponding frequency index vector. Finally, the variables `X_ntof` and `X_fton` are cross-referencing index vectors: For a given index into `X_nvfd`, using the same index into `X_ntof` will return the index of the same VFD in `X_fvfd`. `X_fton` works in the opposite fashion — for a given index into `X_fvfd`, using the same index into `X_fton` returns the index of the same VFD in `X_nvfd`.

The next line (26) of Listing D.3 calls the AOM Toolbox routine `aom_makespv`. This routine allocates storage for the input spectral vector such that the number of phasor locations is equal to the number of VFD Table entries (which it deduces from the `X_ffix` as an input variable), and it initializes the input spectral vector with the input amplitudes, phases, and the input DC value, if any. The form of the input spectral vector that is output by `aom_makespv` is determined by the inputs `X_ffix` (determines whether the spectral vector is 1-sided or 2-sided by inspection of the first element) and `G_spvec_kind` (determines whether the elements will be real-valued or complex-valued). Line 27 of Listing D.3 creates a vector of numeric frequencies corresponding to the frequency-sorted VFD table. This vector will be useful for plotting results. Table D.1 shows the VFD Table created by the call to `aom_vfd` in Listing D.3.
Table D.1: Mixer A VFD Table for 3 Tones in a 2nd Order Nonlinearity

<table>
<thead>
<tr>
<th>Frequency (%)</th>
<th>Index $k_x$ $(k_y, k_z)$</th>
<th>VFD $\eta^T_k$</th>
<th>Norm $|\eta^T_k|_1$</th>
<th>$f_1$ Weight $\eta_{k,1}$</th>
<th>$f_2$ Weight $\eta_{k,2}$</th>
<th>$f_3$ Weight $\eta_{k,3}$</th>
<th>Output Frequency (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\eta^T_0$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>$\eta^T_9$</td>
<td>2</td>
<td>0</td>
<td>$-1$</td>
<td>1</td>
<td>0</td>
<td>0.1</td>
</tr>
<tr>
<td>8</td>
<td>$\eta^T_8$</td>
<td>2</td>
<td>$-1$</td>
<td>1</td>
<td>0</td>
<td>4.4</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>$\eta^T_7$</td>
<td>2</td>
<td>$-1$</td>
<td>0</td>
<td>1</td>
<td>4.5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$\eta^T_1$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>500</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$\eta^T_2$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>504.4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$\eta^T_3$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>504.5</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>$\eta^T_{10}$</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$\eta^T_4$</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1004.4</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>$\eta^T_6$</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1004.5</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>$\eta^T_{11}$</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1008.8</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>$\eta^T_5$</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1008.9</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>$\eta^T_{12}$</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1009</td>
<td></td>
</tr>
</tbody>
</table>

Listing D.4: Call the functions for spectrum mapping, transform matrix, and function evaluation for Mixer A.

28 `SMT_\chi = aom_sspectran(X_{ht\_nvfd}, X_{hash\_tables}, X_{nfix}, fXin_{\chi}, X_{fvd}, X_{ntof}, X_{in\_sv}, MixA_{ost}, G_{num\_sides});`
29 `STM_\chi = aom_sspectran(X_{in\_sv}, SMT_\chi, X_{ffix});`
30 `Y_{sv} = aom sblokeval(X_{in\_sv}, STM_\chi, X_{ffix}, A_{coeff}, 'Y_{sv}');`
31 `Yo = sum(Y_{sv}, 2); % sum spectral vectors from all the non-linear orders; no DC...`
32 `Y_{cv} = aom svtoev(Yo, X_{ffix});`

Listing D.4 shows the calls to the toolbox functions to create the spectrum mapping table and spectrum transform matrix followed by a call to the toolbox function to evaluate the transfer function for Mixer A. Line 28 calls `aom sspecmap` with the hashing 1-sided VFD.
table `X_ht_nvfd`, the hash tables `X_hash_tables`, the 1-norm sorted frequency index `X_nf`x, the input frequency vector `fXin_A`, the frequency-sorted VFD Table `X_fvfd`, the 1-norm sort to frequency sort cross-reference vector `X_ntof`, the input spectral vector `Xin_sv`, the order of spectral truncation `MixA_cost`, and the global variable governing whether spectral vectors are 1-sided or 2-sided `G_num_sides` as inputs. The sparse spectrum mapping table `SMT_X` is returned.

In line 28, a non-empty spectral vector is passed in as a variable, but if an empty vector ([ ] in Matlab®) were passed, then `aom_specmap` would assume that only the input phasor locations (corresponding to VFD table locations with a 1-norm of 1) were non-zero. Line 29 shows the call to `aom_spectrans`, which takes the spectral vector `X_sv` and spectrum mapping table `SMT_X` as inputs (and, optionally, a quoted text variable which is echoed by a Matlab® waitbar) and returns the sparse spectrum transform matrix `STM_x`. Line 30 shows the call to `aom_sblockeval`, the toolbox routine which takes the spectral vector, the spectrum transform matrix, and the coefficients of the nonlinear transfer function `A_coeff` (and, optionally, a quoted text variable which is echoed by a Matlab® waitbar) as inputs and returns a set of output spectral vectors equal to the order of nonlinearity of `A_coeff`. Here, the transfer function is a 2nd order nonlinearity, so `Y_sv` will be a two-dimensional array contain two spectral vectors corresponding to the linear output and the 2nd order output. `Y_sv(:,1)` will contain the linear output and `Y_sv(:,2)` will contain the 2nd order output. In line 31, the two spectral vectors are summed, and in line 32 the real-valued spectral vector is converted to a complex form so that a 1-to-1 correspondence is established between output frequency and output spectral content. Note here that since the inputs are purely real (phase is 0) and the transfer function is real, the outputs will be real also (conveniently so, in this case). Thus the results can be plotted in one plot rather than the more general complex form requiring plotting real and imaginary parts (or magnitude and phase).

The results of the application of AOM using the AOM Toolbox on Mixer A are shown in Figures [D.4 on the next page][D.5 on the following page][D.6 on page 421]. Figure D.4 on the following page shows the down-converted IF (Intermediate Frequency) outputs at 4.4 and 4.5 MHz from Mixer A which occur as a result of 2nd order subtractive intermodulation, Figure [D.5 on the next page] shows the linear output from Mixer A (a consequence of the pass-through of the inputs as dictated by (D.1)), and Figure [D.6 on page 421] shows the 2nd order harmonic and additive intermodulation distortion.
Figure D.4: IF output from Mixer A.

Figure D.5: RF (linear) output from Mixer A.
D.4.3 Ideal Bandpass Filtering of Mixer A Output

Listing D.5 shows the call to `aom_ideal_bpf` on line 36, using the band-edge frequency parameters given in line 35. These band-edge parameters specify the beginning and end frequencies, inclusively, of the filter. The other inputs to `aom_ideal_bpf` are the output spectral vector $Y_0$, the vector of frequencies $X_{ffreq}$ at which $Y_0$ is defined (recall that the input and output spectral vectors are defined in the AOM Toolbox using the same VFD table), and the frequency index vector $X_{ffix}$.

Listing D.5: Ideal bandpass filtering of the output of Mixer A.

```matlab
34  \% Ideal bandpass filter the output from block A
35  start_freq = .43e6; stop_freq = 46e6;
36  Y_bpf = aom_ideal_bpf(Y_0, X_fffreq, X_fffix, start_freq, stop_freq);
37  Y_bpf_cv = aom_svtocv(Y_bpf, X_fffix);
```

Figure D.7 on the next page shows the result of filtering the results in Figure D.4 on page 420 with the ideal bandpass filter. It can be seen that the desired IF outputs are passed unchanged,
but that the DC and undesired baseband IM components near DC have been filtered out.

D.4.4 Analysis of Mixer B

The analysis of Mixer B largely follows the same template of coding as that for Mixer A, so this section will be described with greater brevity. Listing D.6 shows the portions of the Matlab® script which set up the inputs to the AOM Toolbox toolbox codes for computing the output $Z$ as indicated in Figure D.3 on page 415. Lines 40 and 41 use the Matlab® `find` function to locate the non-zero spectral content output by the ideal bandpass filter; these are then concatenated with the local oscillator signal amplitude and frequency in lines 48 and 49. The coefficients for the transfer function of Mixer B are defined in line 50, and the remaining variables in lines 44–47 are similar to those defined for Mixer A.

Listing D.6: Define input signal and transfer function for Mixer B.

```matlab
39  % Apply ideal thresholding to the result from Mixer A and create the
40  % input form for Mixer B
41  Yin = Y_{bpf_cv}(\text{find}(Y_{bpf_cv} > 0))'; % form [A1 A2];
42  fYin = X_{ffreq}(\text{find}(Y_{bpf_cv} > 0))'; % form [f1 f2];
```

Figure D.7: IF output from Mixer A.
Define the Mixer B input:

- `MixB_ost = 2;` Mixer order of spectral truncation (nonlinearity)
- `Yin_int = 3;` Number of input tones
- `Yin_dc = 0.0;` No DC on input
- `Yin_phid = [0 0 0];` Assume 0-degree phase
- `Yin_B = [Yin, 2];` Yin is the bpf-ed output from Mixer A with 2 V LO
- `fYin_B = [fYin, 5e6];` fYin_B are from Mixer A and at 5 MHz for LO
- `B_coeff = [0.5 1 0.5];` \( z = 0.5y^2 + 1.0y + 0.5 \);

Listing D.7 shows the calls to the AOM Toolbox toolbox functions involved in computing all of the quantities pertaining to Mixer B. Due to the similarity of the code here to that for Mixer A, detailed explanation for each line will be omitted. However, line 60 shows one difference that must be highlighted: On this line, the spectral vector outputs `Z_sv` are summed for assignment to `Zo`, but also included here is a call to `aom_unitdc`. `aom_sblockeval` evaluates output spectral vectors from the linear output through the orders of nonlinearity specified in the transfer function, but it does not include the effect of a DC offset that may be included in the transfer function. The toolbox function `aom_unitdc` fills this need. When `aom_unitdc` is called with two arguments, it returns a unit vector of a form compatible with the input spectral vector, with a 1 in the DC location and zeros elsewhere. If the transfer function coefficient vector is passed as a third argument, then `aom_unitdc` scales the unit vector by the DC coefficient in the transfer function.

Listing D.7: Calls to the AOM Toolbox functions for Mixer B:

```matlab
52 % Call the AOM routines for Mixer B...
53 [Yht_nvfd, Yhash_tables, Y_nvfd, Y_nfix, Y_fvfd, Y_ffix, Y_ntof, Y_fton] ... = aom_nvfd(Yin_int, MixB_ost, fYin_B, G_num_sides, 'Y');
55 Yin_sv = aom_makespv(Yin_B, Yin_dc, Yin_phid, Y_ffix, G_spvec_kind);
56 Y_ffreq = cast(Y_fvfd, 'double')*fYin_B;
57 SMT_Y = aom_sspectrans(Yht_nvfd, Yhash_tables, Y_nfix, fYin_B, Y_fvfd, Y_ntof, Yin_sv, MixB_ost, G_num_sides);
58 STM_Y = aom_sspecmap(Yht_nvfd, Yhash_tables, Y_nfix, fYin_B, Y_fvfd, Y_ffix, Y_etof, Y_ntof, MixB_ost, G_num_sides);
59 Z_sv = aom_sblockeval(Yin_sv, STM_Y, Y_ffix, B_coeff, 'Z_sv');
60 Zo = sum(Z_sv, 2) + aom_unitdc(Yin_sv, Y_ffix, B_coeff);
61 Z_cv = aom_svtoce(Zo, Y_ffix);
```
Table D.2 shows the VFD Table created by the call to `aom_vfd` in Listing D.7.

Table D.2: Mixer B VFD Table for 3 Tones in a 2nd Order Nonlinearity

<table>
<thead>
<tr>
<th>Frequency Index $k_x$ $(k_y, k_z)$</th>
<th>VFD</th>
<th>1-Norm $|\eta_k^T|_1$</th>
<th>$f_1$ Weight $\eta_{k,1}$</th>
<th>$f_2$ Weight $\eta_{k,2}$</th>
<th>$f_3$ Weight $\eta_{k,3}$</th>
<th>Output Frequency (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\eta_0^T$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>$\eta_8^T$</td>
<td>2</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0.1</td>
</tr>
<tr>
<td>9</td>
<td>$\eta_9^T$</td>
<td>2</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>7</td>
<td>$\eta_7^T$</td>
<td>2</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>0.6</td>
</tr>
<tr>
<td>1</td>
<td>$\eta_1^T$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4.4</td>
</tr>
<tr>
<td>2</td>
<td>$\eta_2^T$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4.5</td>
</tr>
<tr>
<td>3</td>
<td>$\eta_3^T$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>$\eta_{10}^T$</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>8.8</td>
</tr>
<tr>
<td>4</td>
<td>$\eta_4^T$</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>8.9</td>
</tr>
<tr>
<td>11</td>
<td>$\eta_{11}^T$</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>$\eta_6^T$</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>9.4</td>
</tr>
<tr>
<td>5</td>
<td>$\eta_5^T$</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>9.5</td>
</tr>
<tr>
<td>12</td>
<td>$\eta_{12}^T$</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>

The results of the application of AOM using the AOM Toolbox on Mixer B are shown in Figures D.8 on the next page, D.9 on the following page, and D.10 on page 426. Figure D.8 on the following page shows the down-converted IF (Intermediate Frequency) outputs at 500 and 600 kHz from Mixer B which occur as a result of 2nd order subtractive intermodulation, Figure D.9 on the next page shows the linear output from Mixer B (a consequence of the pass-through of the inputs as dictated by (D.2)), and Figure D.10 on page 426 shows the 2nd order harmonic and additive intermodulation distortion.
Figure D.8: IF output from Mixer B.

Figure D.9: RF (linear) output from Mixer B.
D.4.5 Output Processing

Listing D.8 shows all of the Matlab® code involved in plotting the output shown in Figures D.4 on page 420 through D.10 shown above.

Listing D.8: Matlab® code for plotting output.

```matlab
63    figure (1);
64    stem (X_freq/1e6, Y_cv);
65    xlim ([0 5]); ylim ([0 1.1]);
66    xlabel ('Frequency (MHz)'); ylabel ('Amplitude (volts)');
67    figure (2);
68    stem (X_freq/1e6, Y_cv);
69    xlim ([499 505]); ylim ([0 2.2]);
70    xlabel ('Frequency (MHz)'); ylabel ('Amplitude (volts)');
71    figure (3);
72    stem (X_freq/1e6, Y_cv);
73    xlim ([998 1010]); ylim ([0 0.8]);
74    xlabel ('Frequency (MHz)'); ylabel ('Amplitude (volts)');
75    figure (4);
76    stem (X_freq/1e6, Y_bpf_cv);
77    xlim ([0 5]);
```

Figure D.10: 2nd order harmonic and IM output from Mixer B.
D.5 Known Limitations and Workarounds

Not all of the output variables from \texttt{aom\_vfd} are used elsewhere, but many of them are. It was intended that the 1-norm sorted VFD table and the frequency-sorted VFD table could be used interchangeably, but unfortunately, using the 1-norm sorted VFD table to construct a spectrum mapping table has not been done, so only the frequency-sorted forms are known to work.

It is also possible that if complex spectral vector forms are specified but all-real input vector data is given (i.e. all input phases are zero or 180 degrees), then Matlab\textsuperscript{®} will not allocate storage for complex elements and this may cause some of the code embedded in AOM Toolbox routines to break. When complex spectral vector forms are wanted, be sure that the input phases are distributed around the unit circle such that the imaginary portion of the spectral vector is created. When in doubt, check the class of the output spectral vector returned by \texttt{aom\_makespv}.

D.6 Concluding Remarks

This appendix has furnished an overview of the installation of the AOM Toolbox, given an overview of how it is applied, and applied it to solving a specific problem involving two cascaded nonlinear mixers. A complete Matlab\textsuperscript{®} script that calls the AOM Toolbox functions and produces plotted output was discussed in detail, with particular emphasis on those lines that
call AOM Toolbox functions. Known limitations of the AOM Toolbox were noted, and a workaround for one problem situation was given.
Appendix E

FREDA2 Transcendental Function Examples

E.1 Introductory Remarks

This Appendix furnishes examples of how to use the AOM Toolbox to compute exponential and hyperbolic tangent functions according to the methods first developed by Chang and discussed in Section 3.9. Example driver scripts and function files which demonstrate the two-tone response of the functions are presented. Results for both functions are shown with orders of spectral truncation set to 2 and 3 in order to further demonstrate the effects of spectral truncation.

E.2 Exponential Function

E.2.1 Discussion and Results

The exponential function is evaluated in the Matlab function file aom_exp_fcn.m using the methodology of Section 3.9.6 on page 149. The function file specifically implements (3.355)–(3.359) on page 150 using the AOM Toolbox, except that the variable $n$ as it appears in (3.355)–(3.359) is declared in the Matlab driver script as structure variable field in.Neo and set to a value of 8 on line 31 of Listing E.1 in Section E.2.2 on page 432. Thus (3.359) has
Illustrative results are given here for a 2-tone input signal consisting of equal-amplitude sinusoids at 700 and 800 MHz at −5 dBm. The numerical frequencies were chosen so that both frequency mixing and the visual effects of spectral truncation — its presence and absence — can be easily observed in the plotted output. Figure E.1 shows the exponential function response over a 20 nsec interval, with the response computed using both the AOM Toolbox with 2nd order spectral truncation (followed by subsequent time-domain expansion of the output) and entirely in the time-domain using the built-in Matlab® exponential function for reference. Agreement is reasonably good, but there is a slight divergence at the peaks which is more readily seen in the detail view in Figure E.2 on the next page.

Figures E.3 on the following page and E.4 on page 432 repeat these computations, but this time the order of spectral truncation is set to the 3rd order. As can be seen in Figure E.4 on page 432, increasing the order of spectral truncation leads to much better agreement in the computed results.
Figure E.2: AOM Toolbox Detailed view of exponential function response to a 2-tone input with 2\textsuperscript{nd} order spectral truncation.

Figure E.3: AOM Toolbox Exponential function response to a 2-tone input with 3\textsuperscript{rd} order spectral truncation.
E.2.2 AOM Exponential Driver Script

Listing E.1 is a Matlab® script file that drives the AOM Exponential Function Matlab® function file. Input amplitudes, phases, and frequencies are set in the driver script and results are plotted after the AOM Exponential function is called.

Listing E.1: AOM Exponential driver script.

```matlab
1 % AOM Exponential Function Driver
2 %
3 % North Carolina State University
4 % Copyright Frank P. Hart, 2005−2008
5 %
6 % Driver file for example exp() evaluation per the FREDA2 methodology...
7
clear variables; format long; clc;
8
9 % Power level of input tones
10 Pwr_in = −5; % dBm
11 Ampl_in = aom_dBmToV(Pwr_in);
12
13 % Plot or not...
```
% Take a snapshot of the time
t_start = cpu time;

% Input Spectral Vector Parameters
in.dc = 0.0;
in.A = [Ampl, in, Ampl, in];
in.phid = [90.0, 90.0];
in.f = 1e9*[0.7, 0.8]; % Non-zero frequency input vector for log amp
in.nt = 2; % Number of tones (defined as length(f));
in.Nst = 2; % Order of spectral truncation
in.Nst = 3; % Order of spectral truncation
in.Neo = 8; % Order of series expansion
in.num_sides = 2; % Number of sides to spectrum
in.spvec_kind = 'real'; % Real or complex
%in.spvec_kind = 'comp';

% Call the logamp computational function
out = aom_exp_fcn(in);
Yo = out.Ysv;
fvfd = out.fvfd;

% Use VFD to build matrix containing time domain frequency information
num_freq = size(fvfd, 1);
freq = zeros(length(fvfd),1); % allocate space for freq
for k = 1:length(fvfd)
    freq(k) = cast(fvfd(k,:), 'double')*transpose(in.f);
end;
if in.num_sides == 1
    nposfreq = num_freq-1;
    Yo_cv = aom_svtocv(Yo, X_fifix);
    freq_dci = 1;
    Y_dci = 1;
else % 2-sided
    nposfreq = (num_freq)/2;
    freq_dci = nposfreq + 1;
    Y_dci = (length(Yo))/2 + 1;
end;
%initialize output
yout = Yo(Y_dci)*ones(size(t));
for k=1:nposfreq % fbipd now has DC, so -1 here % Now add time
    freq_ind = freq_dci+k;
    % Right half spectrum --- Do for both 1-sided and 2-sided...
    if isreal(Yo)
        Yr = Yo(Y_dci+2*k-1); Yi = Yo(Y_dci+2*k);
    else
        Yr = real(Yo(Y_dci+k)); Yi = imag(Yo(Y_dci+k));
    end;
    [Ymag,Yphi] = aom_doatan(Yr,Yi);
    yout = yout + Ymag*cos(2*pi*freq(freq_ind)*t - Yphi); % Add results
to time domain signal. Vector op on each t.
if in.num_sides == 2 % Left half spectrum --- Do only for 2-
    freq_ind = freq_dci-k;
    if isreal(Yo)
        Yr = Yo(Y_dci-2*k+1); Yi = Yo(Y_dci-2*k);
    else
        Yr = real(Yo(Y_dci-k)); Yi = imag(Yo(Y_dci-k));
    end;
    [Ymag,Yphi] = aom_doatan(Yr,Yi);
    yout = yout + Ymag*cos(2*pi*freq(freq_ind)*t - Yphi);
end
end %end for loop to add freq. content to yout
% find the exact error of the expansion vs. the built-in
% transcendental function...
[aerr, aerrindex] = max(abs(yout-yref));
aerrct = 100*aerr;
% find the max relative error
[rerr, rerrindex] = max(abs((yout-yref)./yref));
rerrct = 100*rerr;

t_end = cputime - t_start;
fprintf('
');
fprintf(['Time elapsed executing code = ', num2str(t_end), '
']);
% Plot output voltages... vout_aom and vout_ref
```matlab
if plot_out
   fignum = fignum + 1;
   figure(fignum);
   plot(t/1e-9,yout,t/1e-9,yref);
   hla = legend('Y{out} using AOM Toolbox Function','Y{ref} Matlab Built-in Exp Function');
   set(hla,'Location','Best','FontSize',12);
   %title(['Log Amp: 0.9 GHz @ ',num2str(in.A(1)),'V & 1.0 GHz @ ',...%
      num2str(in.A(2)),' V with N ',num2str(in.N),'; Signal Plot']);
   ylim([0.65 1.65]);
   xlim([-0.843 -0.813]);
   xlabel('Time (nsec)');
   ylabel('Output (volts)');
end;
% Plot output zoomed in on a peak to show spectral truncation effect
if plot_out
   fignum = fignum + 1;
   figure(fignum);
   plot(t/1e-9,yout,t/1e-9,yref);
   hla = legend('Y{out} using AOM Toolbox Function','Y{ref} Matlab Built-in Exp Function');
   set(hla,'Location','North','FontSize',12);
   ylim([1.25 1.5]);
   xlim([10.1 10.55]);
   xlabel('Time (nsec)');
   ylabel('Output (volts)');
end;
return;
```

### E.2.3 AOM Exponential Function

Listing E.2 is a Matlab® function file that evaluates the exponential function of its inputs using the AOM Toolbox.

**Listing E.2: AOM Exponential function.**

```matlab
function out = aom_exp_fcn(in)
% aom_exp_fcn executes core AOM Toolbox functions to obtain exponential function output
%
% Input: in - structure variable (see below)
% Output: out - structure variable (see below)
```
% This function is analogous that first implemented by C.R. Chang (ca. 1990)

% Take a snapshot of the time
{\texttt{t_start = cputime;}}

% Input Spectral Vector Parameters and create Spectral Vector
\begin{verbatim}
{\texttt{dc = in.dc;}}  \quad \% DC value
\texttt{A = in.A;} \quad \% non-DC Input Amplitudes
\texttt{phid = in.phid;} \quad \% Input phases (in degrees)
\texttt{f = in.f;} \quad \% Non-zero frequency input vector for log amp
\texttt{nt = in.nt;} \quad \% Number of tones (defined as length(f))
\texttt{Nst = in.Nst;} \quad \% Order of spectral truncation (determines VFD size)
\texttt{Neo = in.Neo;} \quad \% Order of expansion of the exponential function
\texttt{num_sides = in.num_sides;} \quad \% Number of sides to spectrum
\texttt{spvec_kind = in.spvec_kind; \% Spectral vector kind}
\end{verbatim}

% Create the VFD table and the spectral vector for the input...
\begin{verbatim}
{\texttt{[ht_nvfd, hash_tables, nvfd, nfreq_index, fvfd, ffreq_index, ntf, fton] ...}}
\texttt{= aom_vfd(nt, Nst, f, num_sides, 'X');}
\texttt{Xsv = aom_makespv(A, dc, phid, ffreq_index, spvec_kind);}
\end{verbatim}

% Create a "full" spectrum mapping table using the dummy full Xones vector...
\begin{verbatim}
{\texttt{ffreq = cast(fvfd, 'double')*f';}}
\texttt{SMTX = aom_sspecmap(ht_nvfd, hash_tables, nfreq_index, f, fvfd, ntf, num_sides);}
\end{verbatim}

if spvec_kind == 'real'
\begin{verbatim}
\texttt{Xones = ones(size(Xsv)); \quad \% Dummy "full" vector to force full spectrum map...}
\end{verbatim}
else \% complex
\begin{verbatim}
\texttt{Xones = complex(1,1)*ones(size(Xsv)); \quad \% Dummy "full" vector to force full spectrum map...}
\end{verbatim}
end;

% Scale the spectral vector by the "2^n" factor where Neo here corresponds to Chang's "n"
\begin{verbatim}
\texttt{Xsv_scaled = Xsv/2^Neo;} \quad \% Scale spectral vector for exponentiation
\texttt{STMx = aom_sspectrans(Xsv_scaled, SMTX, ffreq_index); \% Create the Tx from Xsv_scaled...}
\end{verbatim}
E.3 Hyperbolic Tangent Function

E.3.1 Discussion and Results

The hyperbolic tangent function is evaluated in the Matlab® function file `aom_tanh_fcn.m` using the methodology of Section 3.9.6 on page 149. The function file specifically implements (3.361)–(3.364) on page 153 using the AOM Toolbox. The AOM Toolbox implementation of the hyperbolic tangent function uses the AOM Toolbox implementation of the exponential function for its computation, and so the variable $n$ as it appears in (3.355)–(3.359) on page 151 is declared in the Matlab® driver script as structure variable field $\text{in.\mbox{Neo}}$ and set to a value of 8 on line 31 of Listing E.3 in Section E.3.2 on page 440. Thus (3.359) has 8 steps in the implementation given here, although users can increase or decrease the number of steps by simply adjusting...
Figure E.5: AOM Toolbox Hyperbolic tangent function response to a 2-tone input with 2nd order spectral truncation.

Illustrative results are given here for a 2-tone input signal consisting of equal-amplitude sinusoids at 700 and 800 MHz at −5 dBm. The numerical frequencies were chosen so that both frequency mixing and the visual effects of spectral truncation — its presence and absence — can be easily observed in the plotted output. Figure E.5 shows the exponential function response over a 20 nsec interval, with the response computed using both the AOM Toolbox with 2nd order spectral truncation (followed by subsequent time-domain expansion of the output) and entirely in the time-domain using the built-in Matlab® exponential function for reference. Agreement is reasonably good, but there is a slight divergence at the peaks which is more readily seen in the detail view in Figure E.6 on the following page.

Figures E.7 on the next page and E.8 on page 440 repeat these computations, but this time the order of spectral truncation is set to the 3rd order. As can be seen in Figure E.8 on page 440, increasing the order of spectral truncation leads to much better agreement in the computed results.
Figure E.6: AOM Toolbox Detailed view of hyperbolic tangent function response to a 2-tone input with 2\textsuperscript{nd} order spectral truncation.

Figure E.7: AOM Toolbox Hyperbolic tangent function response to a 2-tone input with 3\textsuperscript{rd} order spectral truncation.
E.3.2 AOM Hyperbolic Tangent Driver Script

Listing E.3 is a Matlab® script file that drives the AOM Hyperbolic Tangent Function Matlab® function file. Input amplitudes, phases, and frequencies are set in the driver script and results are plotted after the AOM Hyperbolic Tangent function is called.

Listing E.3: AOM Hyperbolic tangent driver script.

```matlab
1  % AOM Tanh Function Driver
2  %
3  % North Carolina State University
4  % Copyright Frank P. Hart, 2005–2008
5  %
6  % Driver file for example tanh() evaluation per the FREDA2 methodology...
7  clear variables; format long; clc;
8  % Power level of input tones
9  Pwr_in = -5, % dBm
10  Ampl_in = aomdbmtov(Pwr_in);
11  % Plot or not...
```

Figure E.8: AOM Toolbox Detailed view of hyperbolic tangent function response to a 2-tone input with 3rd order spectral truncation.
% Take a snapshot of the time
% start = cputime;

% Input Spectral Vector Parameters
in.dc = 0.0;
in.A = [Amp1 in Amp1 in];
in.phid = [90.0 90.0];
in.f = 1e9*[0.7 0.8]; % Non-zero frequency input vector for log amp
in.nt = 2; % Number of tones (defined as length(f));
in.Nst = 2; % Order of spectral truncation
in.Nst = 3; % Order of spectral truncation
in.Neo = 8; % Order of series expansion
in.num_sides = 2; % Number of sides to spectrum
in.spvec_kind = 'real'; % Real or complex
in.spvec_kind = 'comp';

% Call the logamp computational function
out = aom_tanh_fcn(in);
Yo = out.Ysv;
fvfd = out.fvfd;

% Time-domain simulation parameters...
STOP_TIME = 2e-9; % end sim at this time (set-up to match data output from Spice)
TIME_STEP = 1e-12; % time step for sim (set-up to match data output from Spice)
t(1:STOP_TIME/TIME_STEP+1)=0:TIME_STEP:STOP_TIME; %Create time vector from 0 to stop_time in increments of time_step

% We eventually want to plot the aom results overlaid with the results obtained from
% the time domain equation. This code produces the time domain input and output reference
% signals...
% h = waitbar(0,'Computing The Output Signal Using Actual Time Domain Analysis');
xin = in.dc*ones(1,size(t,2));
for q=1:in.nt
    xin = xin + in.A(q)*cos(2*pi*in.f(q)*t-pi*(in.phid(q)/180.0));
end
yref = tanh(xin);

% Use VFD to build matrix containing time domain frequency information
num_freq = size(fvfd,1);

% Create frequency vector from freq-sorted bipd table and locate dc index
freq = zeros(length(fvfd),1); % allocate space for freq
for k = 1:length(fvfd)
    freq(k) = cast(fvfd(k,:), 'double')*transpose(in.f);
end;
if in.num_sides == 1
    nposfreq = num.freq-1;Ycv = aomsvtocv(Yo, X_ffix);
    freq_dci = 1;
    Y_dci = 1;
else  % 2-sided
    nposfreq = (num.freq-1)/2;
    freq_dci = nposfreq + 1;
    Y_dci = (length(Yo)-1)/2 + 1;
end;

%initialize output
yout = Yo(Y_dci)*ones(size(t));
for k=1:nposfreq  % fbipd now has DC, so -1 here  % Now add time
domain component for each freq of the term
    freq_ind = freq_dci+k;
% Right half spectrum  —  Do for both 1-sided and 2-sided...
    if isreal(Yo)
        Yr = Yo(Y_dci+2*k-1); Yi = Yo(Y_dci+2*k);
    else
        Yr = real(Yo(Y_dci+k)); Yi = imag(Yo(Y_dci+k));
    end;
[Ymag,Yphi] = naom_doatan(Yr,Yi);
yout = yout + Ymag*cos(2*pi*freq(freq_ind)*t - Yphi);  % Add results
to time domain signal.  Vector op on each t.
if in.num_sides == 2  % Left half spectrum  —  Do only for 2-sided...
    freq_ind = freq_dci-k;
    if isreal(Yo)
        Yr = Yo(Y_dci-2*k+1); Yi = Yo(Y_dci-2*k);
    else
        Yr = real(Yo(Y_dci-k)); Yi = imag(Yo(Y_dci-k));
    end;
[Ymag,Yphi] = naom_doatan(Yr,Yi);
yout = yout + Ymag*cos(2*pi*freq(freq_ind)*t - Yphi);  %
end
end %end for loop to add freq. content to y_ac

% find the exact error of the expansion vs. the built-in
transcendental function...
[aerror, aerrortx] = max(abs(yout-yref));
aerrpct = 100*aerror;
% find the max relative error
[rerror, rerortx] = max(abs((yout-yref)./yref));
rerpct = 100*rerror;
t_end = cputime - t_start;
fprintf('
');
fprintf(['Time elapsed executing code = ', num2str(t_end), '
']);
% Plot output voltage time series... yout (AOM) and yref (Matlab)
if plot_out
    fignum = fignum + 1;
    figure(fignum);
    plot(t/1e-9,yout,t/1e-9,yref);
    hla = legend('Y{\text{\textit{out}}} using AOM Toolbox Function','Y{\text{\textit{ref}}} Matlab
    Built-in Tanh Function');
    set(hla,'Location','North','Fontsize',12);
    ylim([-0.4 0.6]);
    xlabel('Time (nsec)');
    ylabel('Output (volts)');
end;

% Plot output zoomed in on a peak to show spectral truncation effect
if plot_out_st
    fignum = fignum + 1;
    figure(fignum);
    plot(t/1e-9,yout,t/1e-9,yref);
    hla = legend('Y{\text{\textit{out}}} using AOM Toolbox Function','Y{\text{\textit{ref}}} Matlab
    Built-in Tanh Function');
    set(hla,'Location','North','Fontsize',12);
    ylim([0.20 0.4]);
    xlim([10.1 10.55]);
    xlabel('Time (nsec)');
    ylabel('Output (volts)');
end;
return;

E.3.3 AOM Hyperbolic Tangent Function

Listing E.4 is a Matlab® function file that evaluates the hyperbolic tangent function of its inputs using the AOM Toolbox.

Listing E.4: AOM Hyperbolic tangent function.

```matlab
function out = aom_tanh_fcn(in)
% aom_tanh_fcn executes core AOM Toolbox functions to obtain
% exponential function output
%
% Input: in - structure variable (see below)
% Output: out - structure variable (see below)
%
% North Carolina State University
% Copyright Frank P. Hart, 2005-2008
%
% This function is analogous that first implemented by C.R. Chang (ca.
% 1990)
```
% Take a snapshot of the time
\texttt{t\_start = cputime;}  

% Input Spectral Vector Parameters and create Spectral Vector
\texttt{dc = in\_dc;} % DC value
\texttt{A = in\_A;} % non-DC Input Amplitudes
\texttt{phid = in\_phid;} % Input phases (in degrees)
\texttt{f = in\_f;} % Non-zero frequency input vector for log amp
\texttt{nt = in\_nt;} % Number of tones (defined as length(f))
\texttt{Nst = in\_Nst;} % Order of spectral truncation (determines VFD size)
\texttt{Neo = in\_Neo;} % Order of expansion of the exponential function
\texttt{num\_sides = in\_num\_sides;} % Number of sides to spectrum
\texttt{spvec\_kind = in\_spvec\_kind;} % Spectral vector kind
\texttt{ffreq = cast(fvfd,'double')*f';}

% Create a "full" spectrum mapping table using the dummy full Xones vector...
% The "full" spectrum mapping table will be used to build all spectrum transform matrices
\texttt{if spvec\_kind == 'real'
  Xones = ones(size(Xsv)); % Dummy "full" vector to force full spectrum map...
\texttt{else % complex
  Xones = complex(1,1)*ones(size(Xsv)); % Dummy "full" vector to force full spectrum map...
\texttt{end;}

\texttt{SMT\_X = aom\_sspecmap(h\_nvfd, hash\_tables, nfreq\_index, f, fvfd, ntf, Xones, Nst, num\_sides);}  

% Scale \texttt{-2x} the spectral vector by the "2^n" factor where Neo here corresponds to Chang's "n"
\texttt{Xsv\_scaled = \texttt{-2}*Xsv/2^\texttt{Neo}; % Could also express as \texttt{-Xsv/2^{\texttt{Neo-1}}}

\texttt{STM\_x = aom\_sspectrans(Xsv\_scaled,SMT\_X,ffreq\_index);} % Create the Tx from Xsv\_scaled...

% Create the Taylor series expansion coefficients for computing the part of the exponential
% within the square brackets in the [ ]^\texttt{(2^n)} expression...
\texttt{T\_coeff = [0];

\texttt{for} n = 1:Neo
```
51    T_coff = [1/n, T_coff]; % build in polyval form (same as used by
52    aom_sblockeval.m) ...
53  end;
54
55  % Evaluate the exponential exp(-2x)
56  Ysb = aom_sblockeval(Xsv_scaled,STMx,ffreq_index,T_coff,'Ysv'); %
57  % create out spec vecs excl DC...
58  Yob = sum(Ysb,2); % sum spectral vectors from all the non-linear
59  % orders; no DC...
60  Ysv = Yob + aom_unitdc(Xsv_scaled,ffreq_index); % add in the DC...
61  for n = 1:Neo
62      Ystr = ['Y',num2str(n)];
63      STM_Ysv = aom_sspectrans(Ysv,SMTX,ffreq_index,Ystr); % Use the "
64            full" spec. map table to STM[Ysv]...
65      Yn = STM_Ysv*Ysv;
66      Ysv = Yn;
67  end;
68
69  % Create the num/den spec vecs and the den spectrum transform matrix
70  Wsv = aom_unitdc(Xsv_scaled,ffreq_index) - Ysv;
71  Zsv = aom_unitdc(Xsv_scaled,ffreq_index) + Ysv;
72  STM_Zsv = full(aom_sspectrans(Zsv,SMTX,ffreq_index,'Zsv'));
73
74  % Invert the spectrum transform matrix and get output...
75  % Use of backslash tells Matlab to choose 'best' way to solve
76  Vsv = STM_Zsv\Wsv;
77  out.Ysv = Vsv; % V = tanh(X) output spectral vector
78  out.nvfd = nvfd; % 1-norm sorted vfd table
79  out.fvfd = fvfd; % frequency-sorted vfd table
80  out.Xsv = Xsv; % input X spectral vector
81
82  %keyboard;
83  return;
```
Appendix F

Listings of MATLAB Codes

F.1 Introductory Remarks

This Appendix furnishes listings of the AOM Toolbox and all of the Matlab® scripts, and driver (with called function) codes used in this dissertation.

F.2 Listings of the AOM Toolbox Codes

This section contains listings of the core AOM Toolbox codes.

F.2.1 AOM VFD Table Generation Function

Listing F.1 is the AOM Toolbox function to create VFD tables.

```
function [ht_nvfd, hash_tables, nvfd, nfreq_index, fvd, ffreq_index, ntof, fton, t_ex, nvfd_freq, nvfd_sum, nvfd_hash, vfd_dupes, end_dupes, hash_hits] = ...
aom_vfd(Q, N, f, num_sides, X_namestr, ht_filename)
% aom_vfd creates a Vector Frequency Description (VFD) Table
% aom_vfd returns the VFD table (a column vector of row vectors) given
% specified input frequencies, the order of system nonlinearity or
```
spectral truncation, and an indicator of whether the system under
consideration
will have 1-sided or 2-sided spectra.

Since VFD tables grow combinatorially (as functions of Q and N) and
can be very
large, the construction algorithm builds associative memories in the
form of
hash tables that are used to quickly determine whether a "candidate"
VFD is already
included (or not) in the VFD table. The completed hash tables are
used later by
the spectrum mapping table algorithm.

The inputs are:
Q: The number of input tones. (integer)
N: The highest of system nonlinearity or a user-chosen order of
nonlinear
spectral truncation. (integer)
f: a row vector of size 1 by Q that contains frequencies f1..fQ. (double)
num_sides: the number of sides to the spectral vector (1 or 2);
this input is optional and defaults to 1. (integer 1 or 2)
X_namestr: An optional user-supplied string denoting the name of the
spectral vector for which the VFD is being built. Default value
is 'X'. (string)
ht_filename: An optional user-supplied string denoting the name of a
file which
be used to hold the completed hash tables. (string)

The recommended outputs (for use in applications) are:
hn_vfd: 1-sided nvfd (1-norm sorted VFD table) suitable for use
with hash
tables in spectrum mapping.
hash_tables: cell array of hash tables
nvfd: 1-norm sorted VFD table (column array of Q-tuples of int8)
nfreq_index: array of indices for the nvfd form of the VFD table (column array of doubles)
fvfd: frequency-sorted VFD table (column array of Q-tuples of int8)
ffreq_index: array of indices for the fvfd form of the VFD table (column array of doubles)
ntof: vector cross-referencing the 1-norm sorted to frequency-sorted
VFD table index (column array of doubles)
fton: vector cross-referencing the frequency-sorted to 1-norm sorted
VFD table index (column array of doubles)
t_exec: time of execution of the toolbox function (double)

The additional outputs are diagnostic of the internal toolbox
function operation and
may be ignored by application users.

North Carolina State University
AOM Toolbox Ver. 2.00
Copyright Frank P. Hart, 2005-2008
% Check for aom_cperml and aom_matmuli...
exist_cperml = exist('aom_cperml', 'file');
if exist_cperml ≠ 3
    error('aom_vfd :: Required Mex file aom_cperml not found. You may need to compile C-source...');
end;
exist_matmuli = exist('aom_matmuli', 'file');
if exist_matmuli ≠ 3
    error('aom_vfd :: Required Mex file aom_matmuli not found. You may need to compile C-source...');
end;

% Check that Q = length(f) and kick out if not...
if Q ≠ length(f)
nvfd = -1;
error('vfd: Number of tones parameter, Q, not equal to length of f vector');
return;
end;

% Hashing vector dotted with vfd entries to form nvfd_hash vector
if nargin ≠ 6
    h_filename = 'aom_hash_tables.mat';
end;
if exist(h_filename)
delete(h_filename);
end;
hash_tables = cell(N, 1);
hsdev = sqrt(1/9); % make sigma 1/3
randn('state', 0); % initialize randn to default state to force repeatability...
h = [1:1:Q] .* randn(1, Q) * hsdev; % vector of hashing values...
hT = transpose(h); % ... and its transpose
phi_gr = (sqrt(5.0)-1)/2.0;
A_hash = phi_gr * 2^32; % produces golden ratio for *(*)-bit address
hash_table_ratio = 0.875; % 7/8 selected by trial and error
hash_tables{1} = [Q, h, phi_gr, A_hash, hash_table_ratio]; % Make sure specmap uses h, not hT...

% Perf monitoring counters
ivfs_dupes = 0;
nvfs_dupes = 0;
ivfp_dupes = 0;
nvfp_dupes = 0;
end_dupes = 0;
% Initialize some necessary matrices and vectors
basis = eye(Q, 'int8');  % Set up the vfd basis vectors for the
incommensurates
shift = [ basis(Q,:) ; basis(1:Q-1,:)]';  %[ basis(2:Q,:) ; basis(1,:)] ';
% Set up upper circulant shift matrix [Tdone] -- this must be made
pad = zeros(Q,1,'int8');  % col of zeros to pad vfd_int matrices for n
from 3...N. [Tdone]

% Norm ordering index vector -- for each n, 1≤n≤N, nfirst(n) gives the
index to
% the first element of vfd having nonlinearity equal to n (i.e. 1-norm
= n).
nfirst = zeros(N,1);
if N == 1 nfirst = zeros(2,1); end;
nfirst(1) = 1;
nfirst(2) = 1 + Q;
f = f(:)'; fT = f';

% Determine nvfd... for the right-hand-side, excluding DC
waitstr = ['Computing Vector Frequency Description '];
if nargin == 5 waitstr = ['Computing Vector Frequency Description for
',X,namestr]; end;
waitprog=0;
h1 = waitbar(waitprog,waitstr); pos_w1=get(h1,'Outerposition'); % x,y,
width,height

% Call aom_countvfsds to get vfd counts...
[vfd_total, vfd_countvect, vfd_count_w0] = aom_countvfsds(Q,N);

for n = 1:1:N  % Put in big loop so can do other stuff below...
if n == 1  % n=1	nvfd = basis;  % Initialize the vfd table to the basis vector
matrix
end;
if n ≥ 2
nvfd_end_ptr = size(nvfd,2);
nfirst(n) = 1 + nvfd_end_ptr;  % Index of first entry for n...
Q_int = nfirst(n)-nfirst(n-1);  % # of tones in VFD from previous
iterate on n
nm_int = floor(Q_int/Q);  % # of "whole" intermediate
matrices of dimensions Q x Q
nm_rem = mod(Q_int,Q);  % remainder rows to be "padded
" in last intermediate matrix
n
if nm_rem ≠ 0
nm_tot = nm_int + 1;  % "whole" plus one "padded
" matrix
else
nm_tot = nm_int;  % all "whole" matrices
end

% Initialize secondary waitbar and position on screen below
primary
i f  Q > N
    shftstr = [ ' ' ];
    shftprog = 0;
    h2 = waitbar(shftprog, shftstr);
    pos_w2=[pos_w1(1) pos_w1(2)-pos_w1(3) pos_w1(4) pos_w1(4)];
    set(h2, 'Outerposition', pos_w2, 'doublebuffer', 'on');
end;

% Set up the hash table for this order of non-linearity...
clear hash_table
uab = ceil(hash_table_ratio*log2(vfd_countvect(n)));
hash_length = 2^uab;
nab = ceil((1/hash_table_ratio)*log2(vfd_countvect(n)));
hash_width = 2^nab;
hash_hits = zeros(1*hash_width,1); % diagnostic tool
clear hash_null; hash_null = cell(1,1);
hash_null(1,1) = {zeros(1,hash_width)};
hash_table = repmat(hash_null, hash_length,1);
clear hash_null;

nvfd_ones = ones(1,vfd_countvect(n),'int8');
nvfd_addl = aom_matmuli(pad, nvfd_ones);
nvfd = [nvfd , nvfd_addl];
clear nvfd_ones;
clear nvfd_addl;

for nm = 1:nm_tot % do for "whole" plus one "padded" matrix...
    svfd = shift;
    % Construct intermediate operand matrix
    mat_ind = nfirst(n-1)+(nm-1)*Q; % Index within nvfd to first row of this iteration's mat_int
    if nm < nm_int % create "whole" matrix
        mat_int = nvfd(:,mat_ind:mat_ind+Q-1);
    else % into the remainder rows; create padded matrix
        mat_int = nvfd(:,mat_ind:mat_ind+nm_rem-1); % Start with the remainder rows
        for k = 1:Q-nm_rem
            mat_int = [mat_int , pad]; % Recursively pad to fill out Q rows
        end;
    end;
    if Q > N
        shftstr = [ '* n=' , num2str(n) , ' mm=' , num2str(mm) , ' of ' , num2str(nm_tot) , ' +*'];
        shftprog=mm/nm_tot;
        waitbar(shftprog, h2, shftstr);
    end;
end;

% Now use intermediate operand matrix +/- svfd to determine new vfd entries...
m=1;
while m < Q
for svfd_sign = 1:−2:−1

% add or subtract svfd (do this since * not possible on
int8s)
if (svfd_sign == −1) vfd_int = [mat_int' − svfd]';
else vfd_int = [mat_int' + svfd]'; %
end;

% vector version of norm check
vfd_int_nrm = sum(abs(vfd_int));
vfd_int_nrm_pass = (vfd_int_nrm == cast(n,'int8'));
vfd_int = vfd_int(:,find(vfd_int_nrm_pass));

% vector version of freq check
if (svfd_sign == −1)
vfd_int_freq = f*cast(vfd_int,'double');
vfd_int_freq_pass = (vfd_int_freq ≥ 0);
vfd_int = vfd_int(:,find(vfd_int_freq_pass)); % vfd
now pruned to meet norm and frequency test
end;

vfd_int_dbl = cast(vfd_int,'double'); % use with
weighted vfd hash
% Create vfd_int_nzrows as a cell array of arrays of
find() output.
% vfd_int_nzrows is an array of cells each holding the
% find() output of non−zero tuple indices within all
% entries in vfd_int.
clear vfd_int_nzrows;
vfd_int_nzrowsc = cell(size(vfd_int,2),1);
vfd_int_hval = zeros(size(vfd_int,2),1);
for k = 1: size(vfd_int,2)
clear vfd_int_nzrows;
vfd_int_nzrows = find(vfd_int(:,k));
vfd_int_hval(k) = abs(h(vfd_int_nzrows)*vfd_int_dbl(vfd_int_nzrows,k));
vfd_int_nzrowsc{1} = vfd_int_nzrows;
end;

% k_vect = 1:1:size(vfd_int,2);
for k = size(vfd_int,2):−1:1
  do_nvfd_ck = 1; % assume will check nvfd for
dupes...
clear vfd_int_nzrows;
vfd_int_nzrows = vfd_int_nzrowsc{k};
if k ≠ 1
  lrow_ind = ones(1,k−1,'int8'); lrow_ind = logical
  (lrow_ind);
  for l = 1:length(vfd_int_nzrows)
    lrow_ind = lrow_ind & (vfd_int(vfd_int_nzrows(l),1:k−1) ==
    vfd_int(vfd_int_nzrows(l),k));
    row_ind = find(lrow_ind);
  end
end

%
if isempty(row_ind)  % no match means kth
    vfd_int entry is unique
    break;  % ... so no need to
    continue row checking further...
end;
end;
if ~isempty(row_ind)
    k_vect(k) = [];
    ivfs_dupes = ivfs_dupes + 1;
do_nvfd_ck = 0;
end;
end;
do_nvfd_ck & (nfirst(n) < nvfd_end_ptr)
    hash_tabl_addr = floor(hash_length*rem(vfd_int_hval(k)*A_hash,1))+1;
    if (hash_tabl_addr < 1) | (hash_tabl_addr > hash_length)
        error('aom_vfd: hash table address out of
            bounds');
    end;
    hash_nvfd_list = hash_table{hash_tabl_addr};
    hash_nvfd_addr = find(hash_nvfd_list ~= 0);
    if ~isempty(hash_nvfd_addr)
        hash_nvfd = nvfd(:,hash_nvfd_addr);
        num_rows_ck = size(hash_nvfd,2);
        hash_hits(num_rows_ck) = hash_hits(num_rows_ck
            ) + 1;
        lrow_ind = ones(1,num_rows_ck,'int8');
        lrow_ind = logical(lrow_ind);
        for l = 1:length(vfd_int_nzrows)
            lrow_ind = lrow_ind & (hash_nvfd(
                vfd_int_nzrows(l),1:num_rows_ck) ==
                vfd_int(vfd_int_nzrows(l),k));
            row_ind = find(lrow_ind);
            if isempty(row_ind)  % no match means kth
                vfd_int entry is unique
                break;  % ... so no need to
                continue row checking further...
            end;
        end;
    else
        row_ind = [];
    end;
if ~isempty(row_ind)  % if row_ind has 1 or
    more elements here, vfd_int(k,:) is a dupe,
    delete...
    k_vect(k) = [];
    nvfs_dupes = nvfs_dupes + 1;
end;
clear hash_nvfd;
clear hash_nvfd_list;
clear hash_nvfd_addr;
clear lrow_ind;
```matlab
end; % do_nvfdck
end; % for k = size(vfd_int...)

% update the hash table
nvfd_length = size(nvfd,2);
for k = 1:length(k_vect) % the vfd_int entries
    remaining in k_vect get added to nvfd...
    vfd_int_nzrows = vfd_int_nzrowsc(k_vect(k));
    hash_table_addr = floor(hash_length*rem(vfd_int_hval(k_vect(k)) + A_hash,1)) + 1;
    hash_table_list = hash_table{hash_table_addr}; % get
    the list of nvfd addresses from hash t.
    hash_tabl_zeros = find(hash_table_list == 0,1,'first');
    if ~isempty(hash_tabl_zeros) % put the address into
    the first available zero location
        hash_table_list(hash_tabl_zeros(1)) = nvfd_end_ptr + k;
    else % all initialized locations full, concatenate
    address to list...
        hash_table_list = [hash_table_list, (nvfd_end_ptr + k)];
        hash_hits_current_dim = size(hash_hits,1);
        hash_hits_new_dim = size(hash_table_list,2);
        if hash_hits_new_dim > hash_hits_current_dim
            hash_hits_temp = zeros(hash_hits_new_dim,1);
            hash_hits_temp(1:hash_hits_current_dim,:) =
                hash_hits(1:hash_hits_current_dim,:);
            clear hash_hits;
            hash_hits = hash_hits_temp;
            clear hash_hits_temp;
        end;
        hash_table{hash_table_addr} = hash_table_list;
    end;

% update the nvfd table
nvfd_next_end_ptr = nvfd_end_ptr + length(k_vect);
if nvfd_end_ptr + length(k_vect) <= nvfd_length % normal
    update
    nvfd(:,nvfd_end_ptr + 1 : nvfd_end_ptr + length(k_vect)) = vfd_int(:,k_vect);
    nvfd_end_ptr = nvfd_end_ptr + length(k_vect);
elseif nvfd_next_end_ptr - nvfd_length < length(k_vect)
    kavail = nvfd_length - nvfd_end_ptr;
    for k = 1:kavail
        nvfd(:,nvfd_end_ptr + k) = vfd_int(:,k_vect(k));
    end;
    for k = kavail+1:1
        k_vect(k) = [];
    end;
    nvfd = [nvfd, vfd_int(:,k_vect)];
    nvfd_end_ptr = size(nvfd,2);
else
```
nvfd = [nvfd, vfd_int(:, k_vect)];

nvfd_bot_ptr = size(nvfd, 2);

end;
clear vfd_int;
clear vfd_int dbl;
clear vfd_int nzcolsc;
clear k_vect;
end;
svfd = aom_cperml(svfd, shift);
m = m+1;
end; %while m<Q...
end; %for nm=...
if Q > N
close(h2);
end;
hash_tables{n} = hash_table;
save(ht_filename, 'hash_tables');
clear hash_table;
if nvfd_end_ptr < nvfd_length
nvfd_complete = nvfd(:, 1:nvfd_end_ptr);
clear nvfd;
nvfd = nvfd_complete;
clear nvfd_complete;
end;
end; %if n>2...
waitprog = n/N; % Update the waitbar
waitbar(waitprog);
end; % for n=1:1:N...
close(h1); % Get rid of waitbar
% Have positive-sided nvfd and nfirst excluding DC here.

% Clear out variables no longer needed...
clear svfd vfd_int mat int basis shift; % Need pad, don't clear it!

% Create fvd and ffreq_index for 1-sided spectrum
nvfd = transpose(nvfd);
[nvfd_rows, nvfd_cols] = size(nvfd);
nfreq_index = [1:1:nvfd_rows]'; % set freq. index for 1-norm sort
nfreq = zeros(nvfd_rows, 1); % if unable to reserve this, we're sunk...
for n = 1:1:nvfd_rows % can't cast entire nvfd as double - getting 'out of memory'
nfreq(n) = cast(nvfd(n,:), 'double')*fT; % so preallocate nfreq and fill sequentially
end;
naug_mat = [nfreq nfreq_index]; % nfreq_index same as nperm when 0-pad not included
faug_mat = sortrows(naug_mat, 1);
ffreq = faug_mat(:, 1);
ffreq_index = round(faug_mat(:, 2));
clear naug_mat faug_mat % done with these; free up the memory
fvfd(nfreq_index, :) = nvfd(ffreq_index(nfreq_index), :);
ftyon = ffreq_index; % true here but not at end of routine.
ntof_aug = sortrows([nfreq_index; fton],2);
ntof = ntof_aug(:,1); % Permutation vector mapping 1-norm sort index to freq sort index

% Use 1-sided nvfd and nfirstr from above to create 2-sided objects if necessary...
hvfd = nvfd; % capture nvfd before padding with DC. Reflects exact match to hash tables.

if num_sides == 2
    % Create nvfd and nfirstr for 1-sided spectrum
    nfreq_index = [1:1:length(nvfd)]'; % set freq. index for 1-norm sort
    nvfd = [-flipud(nvfd); pad'; nvfd]; % Add DC to top of nvfd table
    %ssnvfd = nvfd;
    nfreq_index = [-flipud(nfreq_index); 0; nfreq_index];
    fvd = [-flipud(fvd); pad'; fvd];
    ffreq_index = [-flipud(ffreq_index); 0; ffreq_index];
    nfirstr = nfirstr + 1; % Increment values of nfirstr in prep for including DC
    nf = size(nvfd,1); % number of vfd frequencies including DC
    nfirstr = [1; nfirstr]; % first occurrence of 1-norm in nvfd; zero 1-norm is first element
    if N > 1
        nfirstr = [nfirstr; nf+1]; % first occurrence of 1-norm in nvfd;
        zero 1-norm is first element
    end;
    nutfmid = length(nutf) + 1;
    nutf = [-flipud(nutf)+ntofmid; nutfmid; nutf + nutfmid];
    fton = [-flipud(fton)+ntofmid; nutfmid; fton + nutfmid];
    end;

if num_sides == 1
    % Create nvfd and nfirstr for 1-sided spectrum
    nfreq_index = [1:1:length(nvfd)]'; % set freq. index for 1-norm sort
    nvfd = [pad'; nvfd]; % Add DC to top of nvfd table
    nfreq_index = [0; nfreq_index];
    fvd = [pad'; fvd];
    ffreq_index = [0; ffreq_index];
    nfirstr = nfirstr + 1; % Increment values of nfirstr in prep for including DC
    nf = size(nvfd,1); % number of vfd frequencies including DC
    nfirstr = [1; nfirstr]; % first occurrence of 1-norm in nvfd; zero 1-norm is first element
    if N > 1
        nfirstr = [nfirstr; nf+1]; % first occurrence of 1-norm in nvfd;
        zero 1-norm is first element
    end;
    nutf = nutf + ones(length(nutf),1);
    nutf = [1; nutf];
fton = fton + ones(length(fton),1);
fton = [1; fton];
end;

vfd_dupes = zeros(4,1);
vfd_dupes(1) = ivfs_dupes;
vfd_dupes(2) = nvfs_dupes;
vfd_dupes(3) = ivfp_dupes;

tex = cputime - t_begin;
return;

F.2.2 AOM VFD Count Estimation Function

This function is called by the aom_vfd to estimate the storage reserved by aom_vfd to build a VFD table, but is available for end user usage as well.

Listing F.2: AOM Toolbox VFD Count Estimation Function.

function [count_total, count_vect, count_w0_tuples] = aom_countvfsds(Q, N)
% aom_countvfsds combinatorially (over-) estimates the number of single sided VFD entries
% This routine is called by aom_vfd.m and is used to furnish an overestimate of
% the number of entries in the VFD table. aom_vfd uses these results to pre-allocate
% the space required to hold the VFD table, thus speeding the performance of the code.
% Inputs: Q - Number of input tones
% N - Max order of nonlinearity or spectral truncation
% Outputs: count_total - Total number of unique vfd tuples for all response orders
% count_vect - Total number of unique vfd tuples for n = 1 ..N (not including n=0);
% The index into array is response order; array contents the total.
% count_w0_tuples - Total number of tuples containing
count_total = 0; % initialize, set below.
if Q < N
    count_vect = zeros(N,1); count_w0_tuples = zeros(Q-1,1);
    for non_zeros = 1:Q-1 % compute subtotals for any tuples containing zeros...
        count_w0_tuples(non_zeros) = local_binom(Q+non_zeros-1,non_zeros);
    end;
    %
    for n = 1:N
        count_subt = 0; count_bin = 0;
        if n < Q-1 nzlim = n;
        else nzlim = Q-1;
        end;
        for nz = 1:nzlim
            count_vect(n) = count_vect(n) + 2^(nz-1)*count_w0_tuples(nz);
            count_subt = count_subt + count_w0_tuples(nz);
        end;
        count_bin = local_binom(n+Q-1,n);
        if n < Q binlim = n;
        else binlim = Q;
        end
        count_vect(n) = count_vect(n) + 2^(binlim-1)* ( count_bin - count_subt );
    end;
    count_total = sum(count_vect) + 1; % add 1 for n = 0
end;
if Q > N
    count_vect = zeros(N,1); count_w0_tuples = zeros(N,1);
    for non_zeros = 1:N % compute subtotals containing zeros...
        count_w0_tuples(non_zeros) = local_binom(Q+non_zeros-1,non_zeros);
    end;
    %
    for n = 1:N
        count_subt = 0; count_bin = 0;
        for nz = 1:n
            count_vect(n) = count_vect(n) + 2^(nz-1)*count_w0_tuples(nz);
            count_subt = count_subt + count_w0_tuples(nz);
        end;
        count_bin = local_binom(n+Q-1,n);
        count_vect(n) = count_vect(n) + 2^(n-1)* ( count_bin - count_subt );
    end;
    count_total = sum(count_vect) + 1;
end;
return;
function binout = local_binom(a,b);
    binout = nchoosek(a,b);
    return;

function binout = local_binom2(n,k);
    % C(n,k) = [n*(n-1)* *(n-k+1)]/[k*(k-1)* *(1)]
    %diff = n-k;
    num_vect = n:-1:n-k+1;
    num_vect_length = length(num_vect);
    num_fact = [factor(num_vect(1)), 1];
    if num_vect_length > 1
        for i = 2:num_vect_length
            num_fact = [num_fact, factor(num_vect(i))];
        end;
    end;
    den_vect = k:-1:1;
    den_vect_length = length(den_vect);
    den_fact = factor(den_vect(1));
    if den_vect_length > 1
        for j = 2:den_vect_length
            den_fact = [den_fact, factor(den_vect(j))];
        end;
    end;
    den_fact_length = length(den_fact);
    for j = den_fact_length:-1:1
        curr_fact = den_fact(j);
        curr_fact_loc = find((num_fact == curr_fact),1,'first');
        if isempty(curr_fact_loc)
            % this should never happen!
            error('aom_countvdfs :: Denominator prime factor not found in numerator');
        else % this should be routine...
            num_fact(curr_fact_loc) = [];
            den_fact(j) = [];
        end;
    end;
    if ~isempty(den_fact)
        error('aom_countvdfs :: Denominator prime factors not empty after cancellation');
    else
        binout = prod(num_fact);
    end;
    return;
F.2.3 AOM Int8 Column Permutation Function

This is a small piece of compiled C-code that is seen by Matlab® as a compiled Matlab® executable function. It is called by aom_vfd and is used to permute columns in matrices containing 8-bit integer entries.

Listing F.3: AOM Toolbox VFD Matrix Column Permutation Function.

```c
/* aom_cperml – Matlab Executable Function to do ’char’ integer permutation w/ perm matrix.
   This C-code implements the Matlab equivalent of:
   function c = a*b
   when b is a permutation matrix (i.e. it has only one ’1’ in each row and column — no more!).

   [a_rows, a_cols] = size(a);
   [b_rows, b_cols] = size(b);
   if a_cols ~= b_rows
     error(’aom_cperml :: matrices not conformable for multiplication’);
   end;
   c = zeros(a_rows, b_cols, ’int8’);
   for i = 1:a_rows
     for j = 1:b_cols
       for k = 1:a_cols
         c(i,j) = c(i,j) + a(i,k)*b(k,j);
       end;
     end;
   end;
   return;

   Arguments that must be passed from Matlab: a, b
   Returned: c
   a is any matrix
   b is assumed to be a permutation matrix (you’ve been warned!)
   Format of call in Matlab:
   c = aom_cperml(a, b);
   Types: a, b, c = ie x je int8 (from Matlab explore function)
   Caution with indices:
```
Ex: Matlab 1.60 -->
C++ 0.59
So Matlab 2.59 -->
C++ 1.58

#include "mex.h"
#define NDIMS 2
#define ONE '1'

void matcperml(char *c, char *a, char *b,
int mrows, int mind, int pcols)
{
    int i, jcol, j, k;
    /* i=0 in C++ --> i=1 in Matlab */
    for (j=0; j<pcols; j++) {
        /* For all cols in b */
        jcol = 0;
        /* Init the target copy column to 0... */
        for (i=0; i<mrows; i++) {
            /* Find the row in column j with a 1... */
            if ( *(b+i+j*mind) == 1 ){
                jcol = i;
                break;
            }
        }
        for (k=0; k<mrows; k++) {
            *(c+k+j*mrows) = *(a+k+jcol*mrows);
        }
    }
}

/* Matlab gateway function. */
void mexFunction(
int nlhs, mxArray *plhs [] ,
int nrhs, const mxArray *prhs [] )
{
    char *a, *b, *c;
    int a_cols, b_rows, dims [2];
    /* Check for proper number of arguments. */
    if (nrhs! =2)
        mexErrMsgTxt("aom_cperml : Two inputs required.");
    if (nlhs! =1)
        mexErrMsgTxt("aom_cperml : One output required.");
    /* Create pointers to the input matrices. */
a = mxGetPr(prhs[0]);
b = mxGetPr(prhs[1]);

    /* Get dimensions of input matrices. */
dims [0] = mxGetM(prhs [0] );
dims [1] = mxGetN(prhs [1] );
a_cols = mxGetM(prhs [0] );
b_rows = mxGetM(prhs [1] );
if (a_rows != a_cols)
    mexErrMsgTxt("aom_permil :: Permutation matrix not square.");

if (a_cols != b_rows)
    mexErrMsgTxt("aom_permil :: Matrices not conformable for permutation.");

/* Allocate space for the output matrix. */
plhs[0] = mxCreateNumericArray(NDIMS, dims, mxINT8_CLASS, mxREAL);
c = mxGetPr(plhs[0]);

/* Call matcperrml... */
matcperrml(c, a, b, dims[0], a_cols, dims[1]);
}

F.2.4 AOM Int8 Matrix Multiplication Function

Listing F.4 is a small piece of compiled C-code that is seen by Matlab® as a compiled Matlab® executable function. It is called by aom_vfd and is used to perform outer product multiplications on vectors of 8-bit integers.

Listing F.4: AOM Toolbox VFD Matrix Column Permutation Function.
/* aom_matmuli - Matlab Executable Function to do 'char' integer matrix multiply
This C-code implements the Matlab equivalent of:
function c = aom_matmuli(a, b)
[ a_rows, a_cols ] = size(a);
[ b_rows, b_cols ] = size(b);
if a_cols != b_rows
    error('aom_matmuli :: matrices not conformable for multiplication ');
end;
c = zeros(a_rows, b_cols, 'int8 ');
for i = 1:a_rows
    for j = 1:b_cols
        for k = 1:a_cols
            c(i,j) = c(i,j) + a(i,k)*b(k,j);
        end;
    end;
end;*/
Arguments that must be passed from Matlab: a, b

Returned: c

Format of call in Matlab:

\[ c = \text{aom\_matmuli}(a, b); \]

Types: \( a, b, c \) — int8 (from Matlab explore function)

Caution with indices:
Ex: Matlab 1..60 \( \rightarrow \)
\[ C++ \quad 0..59 \]
So Matlab 2..59 \( \rightarrow \)
\[ C++ \quad 1..58 \]

*/
#include "mex.h"
#define NDIMS 2

void matmuli(
    char *c, char *a, char *b,
    int mrows, int nind, int pcols)
{
    int i, j, k;
    /* i=0 in C++ \( \rightarrow \) i=1 in Matlab */
    for (j=0; j<pcols; j++) {
        for (i=0; i<mrows; i++) {
            for (k=0; k<nind; k++) {
                *(c+i+j*mrows) = *(c+i+j*mrows) +
                (*(a+i+k*mrows) * *(b+k+j*nind));
            }
        }
    }
}

/* Matlab gateway function. */
void mexFunction(
    int nlhs, mxArray *plhs[],
    int nrhs, const mxArray *prhs[])
{
    char *a, *b, *c;
    int a\_cols, b\_rows, dims[2];

    /* Check for proper number of arguments. */
    if (nrhs!=2)
        mexErrMsgTxt("aom\_matmuli :: Two inputs required.");
    if (nlhs!=1)
        mexErrMsgTxt("aom\_matmuli :: One output required.");

    /* Create pointers to the input matrices. */
463

a = mxGetPr(prhs[0]);
b = mxGetPr(prhs[1]);

/* Get dimensions of input matrices. */
dims[0] = mxGetM(prhs[0]);
dims[1] = mxGetN(prhs[1]);
a_cols = mxGetN(prhs[0]);
b_rows = mxGetM(prhs[1]);
if (a_cols != b_rows)
    mexErrMsgTxt("aom_matmuli:: Matrices not conformable for multiplication.");

/* Allocate space for the output matrix. */
plhs[0] = mxCreateNumericArray(NDIMS, dims, mxINT8_CLASS, mxREAL);
c = mxGetPr(plhs[0]);

/* Call matmuli... */
matmuli(c, a, b, dims[0], a_cols, dims[1]);
}

F.2.5 AOM Spectral Vector Creation Function

Listing F.5 is the AOM Toolbox function to create a spectral vector given input amplitudes, phases, and frequencies.

Listing F.5: AOM Toolbox Spectral Vector Creation Function.
function [X, t_ex] = aom_makespv(A, dc, phid, ffreq_index, spvec_kind_in)
% aom_makespv creates a spectral vector given input spectral data
% aom_makespv creates a spectral vector given vectors of input amplitude,
% and frequency characteristics, the form of the spectral vector,
% and the frequency–sorted frequency index vector, which is used to
% identify and ”seed” the input spectral data.
% Inputs:
% A – vector of amplitudes for the non-zero input frequencies (doubles)
% dc – scalar containing the DC value for the spectral vector (double)
% phid – vector of input phases, in degrees, corresponding to A (double)
ffreq_index = vector of frequency indices sorted by increasing frequency (double)
spvec_kind_in = 'real' or 'complex' to denote spectral vector form

Outputs:
X = spectral vector corresponding to the input spectral data
t_ex = time of execution of function call

North Carolina State University
AO M Toolbox Ver. 2.00
Copyright Frank P. Hart, 2005–2008

Snapshot time for returning time of execution t_ex
t_begin = cputime;
fprintf(['Entering Spectral Vector Creation Code...
']);
eps_thresh = 1e-15; % threshold for zero-ing out attempts by cos and sine to
                % return eps in cases of cos(n*pi/2) or sin(n*pi).

%if nargin ≠ 5
if (nargin < 4) | (nargin > 5)
    error('makespv :: The number of input arguments required is 4 or 5.
');
end;

if fffreq_index(1) == 0
    num_sides = 1;
else
    num_sides = 2;
end;
if (nargin == 4)
    spvec_kind = 'real';
end;
if (nargin == 5)
    if ((length(spvec_kind_in) == 6) & (spvec_kind_in == 'complex'))
        spvec_kind = 'comp';
    elseif ((spvec_kind_in == 'real') | (spvec_kind_in == 'comp'))
        spvec_kind = spvec_kind_in; % do nothing
    else
        error('makespv :: Invalid specification for kind of spectral vector.');
    end;
end;
if (num_sides == 1) & (spvec_kind == 'comp')
    error('makespv :: One-sided complex spectral vectors are not permitted.');
end;
phi_r = (phi_d/180.0)*pi;  % convert from degrees to radians before passing to cos & sin

% does only 'sine' function right now.... add more to do others later...

if (num_sides == 2) & (spvec_kind == 'comp')
    X_length = length(ffreq_index);
    X_re = zeros(X_length,1);
    X_im = zeros(X_length,1);
    X_dc_el = (X_length-1)/2 + 1;
    X_re(X_dc_el) = dc;
    ffreq_half_length = (length(ffreq_index)-1)/2;
    ffreq_dc = ffreq_half_length + 1;
    for n = 1:length(A)  % Do for each input tone...
        k = find([ffreq_index-n*ones(length(ffreq_index),1) == 0])-ffreq_dc;
        Re_phi = cos(phi_r(n));
        if abs(Re_phi) < eps_thresh  % Do not allow assignment of eps when on pi-boundary
            Re_phi = 0.0;
        end;
        Im_phi = sin(phi_r(n));
        if abs(Im_phi) < eps_thresh
            Im_phi = 0.0;
        end;
        ReX = (1.0/2.0)*A(n)*Re_phi;
        ImX = (1.0/2.0)*A(n)*Im_phi;
        X_re(X_dc_el + k) = (1.0/2.0)*A(n)*Re_phi;
        X_im(X_dc_el + k) = (1.0/2.0)*A(n)*Im_phi;
        X_re(X_dc_el - k) = -(1.0/2.0)*A(n)*Re_phi;
        X_im(X_dc_el - k) = (1.0/2.0)*A(n)*Im_phi;
    end;
    X = complex(X_re,X_im);
end;

if (num_sides == 2) & (spvec_kind == 'real')
    X_length = 2*length(ffreq_index)-1;  % X's length is 2x(ffreq_index) - 1.
    X = zeros(X_length,1);
    X_dc_el = (X_length-1)/2 + 1;
    X(X_dc_el) = dc;
    ffreq_half_length = (length(ffreq_index)-1)/2;
    ffreq_dc = ffreq_half_length + 1;
    for n = 1:length(A)  % Do for each input tone...
        k = find([ffreq_index-n*ones(length(ffreq_index),1) == 0])-ffreq_dc;
        ReX = cos(phi_r(n));
        if abs(ReX) < eps_thresh  % Do not allow assignment of eps when on pi-boundary
            ReX = 0.0;
        end;
        ImX = sin(phi_r(n));
        if abs(ImX) < eps_thresh
            ImX = 0.0;
        end;
    end;
end;
F.2.6 AOM Spectrum Mapping Table Function

Listing F.6 is the AOM Toolbox function to create a spectrum mapping table.

Listing F.6: AOM Toolbox Spectrum Mapping Table Function.

1 function [specmap, t_ex] = aom_ssspecmap (ht_nvfd, hash_tables_in, 
2 nfreq_index, l_in, fvd, fperm, X, N, num_sides_in) 
3 % aom_ssspecmap returns a spectrum mapping table given a VFD table and 
4 related info 
5
% aom_specmap returns a spectrum mapping table given a frequency-
% sorted VFD
% table (fvfd), the 1-sided, 1-norm sorted hash table corresponding
% exactly to the
% hash tables in the hash_tables_in cell array, the 1-norm sorted
% frequency
% index vector nfreq_index, the input frequency vector f_in, the input
% spectral
% vector X (or optionally, []), the order of maximum system
% nonlinearity or
% order of spectral truncation (N), and the number of sides indicator
% (1 or 2)
% variable. If [] is passed instead of an actual spectral vector, then
% code will use the norm of the fvfd entries to determine the
% locations
% corresponding to linear inputs and will assume these — and only
% these —
% are non-zero. This facility allows the creation of a single
% spectrum mapping
% the input spectral content. The code will perform the mechanics of
discrete
% frequency convolution (i.e. adding and subtracting VFDs for aligned
% spectral
% content), then determine if the resulting frequency location for
% each "atomic
% convolution" product (or phasor multiplication) is valid. If valid,
% an entry
% will be made in the spectrum mapping table.
% The spectrum mapping table takes two forms depending upon whether
% the number
% of sides is 1 or 2. For 2-sided spectral content, only VFD addition
% is performed
% and the the three frequency indices (which index the space of the
% spectral
% vectors as well as the row and column spaces of the spectrum
% transform matrix)
% are recorded in the table for valid combinatoins, while for 1-sided
% spectral
% content, the VFDs are subtracted from each other and two fields
% corresponding
% to the sign of each VFD in the subtraction operation are added to the
% spectrum
% mapping table entry. When two VFDs involved in an atomic
% multiplication are
% added (or subtracted), the resulting VFD is checked for compliance
% with the
% 1-norm limit, is negated if it corresponds to a negative frequency
% in a 2-sided
% problem, and the VFD location is then identified using the 1-sided
% nvfd hash tables.
Inputs: ht_nvfd = 1-sided (and excluding DC) hash table with indices corresponding to those in the hash_tables_in data structure; created by aom_vfd.m
hash_tables_in = hash tables for quickly looking up the location of a valid VFD and translate to a valid output frequency index.
nfreq_index = 1-norm sorted frequency index vector (DC at center for 2-sided spectral vectors)
f_in = The input frequency vector, which is dotted with an atomic product to determine whether the output of a 2-sided convolution is positive or negative frequency
fvfd = frequency-sorted VFD table which has its entries added (or subtracted) corresponding to discrete spectral convolution fperm = translates from 1-norm sort frequency index location to frequency-sorted frequency index location
X = input spectral vector
N = order of nonlinearity or spectral truncation
num_sides_in = (1 or 2) indicator of 1-sided or 2-sided spectral vectors

Outputs: specmap = Table of 3-tuples (for 2-sided spectra) or 5-tuples (for 1-sided spectra) (all doubles)
t_ex = Time of execution of the toolbox function (double)

count_total = 0; % initialize, set below.
t_begin = cputime;
fprintf(['Entering Spectrum Mapping Table Generation Code...
']);
global hash_tables;
global ht;
global phi_gr;
global A_hash;
global hash_table_ratio;
global num_sides;
global ikn;
global idx;
global fT;
global dc_el;
global nl_nvfd_lut;
global nl_nfi;
fT = transpose(f_in);

num_sides = num_sides_in;
if num_sides == 1
dc_el = 1;
else % num_sides == 2...
dc_el = (1/2)*(length(nfreq_index)-1) + 1;
end;

nl_nvfd_lut = ht_nvfd(1:length(f_in),:);
nl_nfi = [1:1:length(f_in)]';
hash_tables = hash_tables_in;

ht_data_arr = hash_tables{1};
Q = ht_data_arr(1);
ht = ht_data_arr(2:Q+1);
phi_gr = ht_data_arr(Q+2);
A_hash = ht_data_arr(Q+3);
hash_table_ratio = ht_data_arr(Q+4);

if nargin < 9
    error('sspecmap :: Number of input arguments must be 9. ');
end;

% Get VFD table size using the function VFD.
[num_freqs, num_tones]=size(fvd);
%
% Works for 1-s and 2-s when sorted for 2-s
freq_index = sortrows(nfreq_index); % Order okay for 1-S, must sort
for 2-S!
%
% Create a vector called Xmag indicating whether the phasor
% coefficient of X at frequency index kx is 0 or not.
if num_sides == 1
    Xmag=zeros(1,num_freqs);
    if isempty(X)
        fnorm = sum(abs(fvfd),2);
        fl_ind = find(fnorm == 1);
        Xmag(fl_ind) = 1;
    else
        if isreal(X)
            Xmag(1)=sign(abs(X(1))); % DC term
            for kx = 1:num_freqs-1
                Xmag(kx+1)=sign(X(2*kx)'+X(2*kx+1)'-2); % Non-zero freqs
            end;
        else % complex spectral vector
            Xmag=sign(abs(X));
        end;
    end;
else % num_sides == 2
    Xmag=zeros(1,num_freqs);
if isempty(X)
    fnorm = sum(abs(fvfd), 2);
    f1_ind = find(fnorm == 1);
    Xmag(f1_ind) = 1;
else
    if isreal(X) % real spectral vector
        X_dc_el = (length(X)-1)/2 + 1; % Middle element of X is DC
        Xmag_dc_el = (num_freqs-1)/2 + 1; % Middle element of Xmag is DC
        Xmag(Xmag_dc_el) = sign(abs(X(X_dc_el)));
        for kx = 1:(num_freqs-1)/2
            Xmag(Xmag_dc_el+kx) = sign(X(X_dc_el+2*kx-1)+2*X(X_dc_el+2*kx-1)^2);
            Xmag(Xmag_dc_el-kx) = sign(X(X_dc_el-2*kx+1)+2*X(X_dc_el-2*kx+1)^2);
        end;
        else % complex spectral vector
            Xmag = sign(abs(X));
        end;
end;
end;
end;

specmap = [];
    % The spectrum mapping
table - try '!', if necessary
    if num_sides == 1
        specmap_entry = [0 0 0 0 0]; % Single entry in the specmap
        - initialize
    else % num_sides == 2, so no signs to worry about...
        specmap_entry = [0 0 0];
    end;
fvfd_length = size(fvfd, 1);
nz_Xmag = find(Xmag); % get the non-zero indices in Xmag
nz_Xmag_length = length(nz_Xmag); % length of non-zero indices
vector
[specmap, specmap_bot_ptr] = local_alloc_specmap(specmap, nz_Xmag_length,
    fvfd_length, num_sides);
specmap_ptr = 1; % incr by one after each write...
cp_vfd_test = zeros(1, num_tones); % Convolution product VFD to test
for membership
    if isempty(X)
        waitstr = ['Computing Spectrum Mapping Table'];
    else
        waitstr = ['Computing Spectrum Mapping Table for ', inputname(7)];
    end;
waitpct = 0;
waitprog = 0;
h = waitbar(waitprog, waitstr);
if num_freqs == 1
    else
        ikn_max = 2*num_freqs-1;
for ikn = 1:1:ikn_max
    nz_ikx_first = [];
nz_i_kx_last = []; nz_i_kx = []; if ikn <= num_freqs % for convolving "leading" frequency domain spikes
  ixxstart = 1; ixxend = ikn;
  if ixxend > nz_Xmag(1)
    nz_i_kx_first = find(nz_Xmag >= ixxstart,1,'first');
    nz_i_kx_last = find(nz_Xmag <= ixxend,1,'last');
  end;
  else % ikn>num_freqs % for convolving "trailing" frequency domain spikes
    ixxstart = ikn-num_freqs+1;
    ixxend = num_freqs;
    if ixxstart <= nz_Xmag(end)
      nz_i_kx_first = find(nz_Xmag >= ixxstart,1,'first');
      nz_i_kx_last = find(nz_Xmag <= ixxend,1,'last');
    end;
  end;
  nz_i_kx = nz_Xmag(nz_i_kx_first:nz_i_kx_last);
%
% use this test to govern rest of loop execution...
if ~isempty(nz_i_kx)
  nz_i_kz = ikn - nz_i_kx + 1;
  fvfd_nz_i_kx = fvfd(nz_i_kx,1:num_tones); % Find the non-zero indices of X
  fvfd_nz_i_kz = fvfd(nz_i_kz,1:num_tones); % ... and the corresponding indices of Z
  cp_vfd_nrm = sum(abs(fvfd_nz_i_kx) + abs(fvfd_nz_i_kz)),2);
  nleN_i_kx = find(cp_vfd_nrm <= N);
  if ~isempty(nleN_i_kx)
    for inx = 1:length(nleN_i_kx)
      idx = nleN_i_kx(inx);
      cp_vfd_sum = fvfd_nz_i_kx + fvfd_nz_i_kz; % sum
      cp_vfd_xmz = fvfd_nz_i_kx - fvfd_nz_i_kz; % diff x minus z
      cp_vfd_zmx = fvfd_nz_i_kz - fvfd_nz_i_kx; % diff z minus x
      if cp_vfd_nrm(idx) < N % skip if norm requirement not met
        db_i_kx = freq_index(nz_i_kx(idx));
        db_i_kz = freq_index(nz_i_kz(idx));
        iky = ck_vfdd(cp_vfd_sum(idx,1:num_tones),ht_nvfd,num_tones,fperm);
        if iky == -1
          error('specmap: Invalid ky entry returned from check of vfd table.');
        end; % if iky...
        if num_sides == 1
          sx = 1; sz = 1;
          specmap_entry = [' freq_index(iky) freq_index(nz_i_kx(idx))
                         sx freq_index(nz_i_kz(idx)) sz'];
        else % num_sides == 2, so no signs...
          specmap_entry = [' freq_index(iky) freq_index(nz_i_kx(idx))
                           freq_index(nz_i_kz(idx)) ']';
      end; % if cp_vfd_nrm(idx) < N
    end; % for inx = 1:length(nleN_i_kx)
  end; % if ~isempty(nz_i_kx)
end;

if specmap_ptr == specmap_bot_ptr
    [specmap, specmap_bot_ptr] = local_alloc_specmap(specmap, nzXmag_length, fvfd_length, num_sides);
    specmap(:,specmap_ptr) = specmap_entry;
else
    specmap(:,specmap_ptr) = specmap_entry;
end;

specmap_ptr = specmap_ptr + 1;

% ... plus do one of the three below (only when 1-sided)
if num_sides == 1
    % Case 1 — Rectification (and not of DC terms since ikn ≠ 1)
    if ((nz_ikx(idx) == nz_ikz(idx)) & (ikn ≠ 1))
        sx = 1; sz = -1; % No need to test, maps to DC, so freq_index(iky)=0
        specmap_entry = [ 0 freq_index(nz_ikx(idx)) sx freq_index(nz_ikz(idx)) sz ];
        if specmap_ptr == specmap_bot_ptr
            [specmap, specmap_bot_ptr] = local_alloc_specmap(specmap, nzXmag_length, fvfd_length, num_sides);
            specmap(:,specmap_ptr) = specmap_entry;
        else
            specmap(:,specmap_ptr) = specmap_entry;
        end;
        specmap_ptr = specmap_ptr + 1;
    end;
% Case 2 — Subtract Z's freq from X's for non-DC Z
    if ((nz_ikz(idx) > nz_ikx(idx)) & (nz_ikx(idx) > 1))
        iky = ck_vfdm(cp_vfdzmx(idx,1:num_tones), htnvfd, num_tones, fperm);
        if iky == -1
            error('specmap: Invalid ky entry returned from check of vfd table. ');
        end; % if iky ==
        sx = 1; sz = -1;
        specmap_entry = [ freq_index(iky) freq_index(nz_ikx(idx)) sx freq_index(nz_ikz(idx)) sz ];
        if specmap_ptr == specmap_bot_ptr
            [specmap, specmap_bot_ptr] = local_alloc_specmap(specmap, nzXmag_length, fvfd_length, num_sides);
            specmap(:,specmap_ptr) = specmap_entry;
        else
            specmap(:,specmap_ptr) = specmap_entry;
        end;
        specmap_ptr = specmap_ptr + 1;
    end; % Case 2
% Case 3 — Subtract X's freq from Z's for non-DC X
    if ((nz_ikz(idx) > nz_ikx(idx)) & (nz_ikx(idx) > 1))
        iky = ck_vfdm(cp_vfdzmx(idx,1:num_tones), htnvfd, num_tones, fperm);
        if iky == -1
            error('specmap: Invalid ky entry returned from check of vfd table. ');
        end; % if iky ==
        sx = 1; sz = -1;
        specmap_entry = [ freq_index(iky) freq_index(nz_ikx(idx)) sx freq_index(nz_ikz(idx)) sz ];
        if specmap_ptr == specmap_bot_ptr
            [specmap, specmap_bot_ptr] = local_alloc_specmap(specmap, nzXmag_length, fvfd_length, num_sides);
            specmap(:,specmap_ptr) = specmap_entry;
        else
            specmap(:,specmap_ptr) = specmap_entry;
        end;
        specmap_ptr = specmap_ptr + 1;
    end; % Case 3
```matlab
end;  \% if iky ==..
sx = -1; sz = 1;
specmap_entry = [ freq_index(iky) freq_index(nz lkx(idx)) sz freq_index(nz lkz(idx))];
if specmap_ptr == specmap_bot ptr
    [specmap, specmap_bot ptr] = local_alloc specmap (specmap, nz Xmag length, fvd_length, num sides);
specmap(:, specmap_ptr) = specmap_entry;
else
    specmap(:, specmap_ptr) = specmap_entry;
end;
specmap_ptr = specmap_ptr + 1;
end; \% Case 3
end; \% if num_sides == 1, do extra cases
end; \% if < N ... for iky = ..
end; \% if ~isempty(nle N lkx)
end; \% if ~isempty(nz lkx)...
if (waitpct + 1) <= fix(100*ikn/ikn_max)
    waitpct = waitpct + 1;
    waitprog = waitpct/100;
end waitbar(waitprog);
end; \% for ikn = ..
waitbar(1);
% Clean up spare table entries if over--allocated
if specmap_ptr-1 < specmap_bot ptr
    specmap_complete = specmap(:, 1:specmap_ptr-1);
clear specmap;
specmap = transpose(specmap_complete);
clear specmap_complete;
else
    specmap = transpose(specmap);
end;
if num_sides == 1 \% sort rows of specmap to min memory thrash in
    spectrans construction
    specmap = sortrows(specmap,[4 1]); \% sort by kz (column first),
    then ky (row w/in col)
else \% num_sides == 2, no signs to sort...
    specmap = sortrows(specmap,[3 1]);
end;
close(h);printf(["Exiting Spectrum Mapping Table Generation Code...

          t ex = cputime - t begin;
return;
function ck_vfd out=ck_vfd m(cp vfd test, ht_nvfd, num tones, fperm)
global hash_tables; \% try as global to avoid passing large data
    structure on each call..
global ht;
global phi gr;
global Ahash;
global hash_table ratio;
global num_sides;
```
global ikn;
global idx;
global FT;
global dc_el;
global n1_nvfd_lut;
global n1_nfi;

cp_vfd_out = -1;
cp_vfd_norm = round(cast(sum(abs(cp_vfd_test)),'double'));
cp_vfd_neg = 1;
if cast(cp_vfd_test,'double')*FT < 0;
    cp_vfd_neg = -1;
    cp_vfd_test = -cp_vfd_test;
end;
if cp_vfd_norm == 0
    ck_vfd_out = fperm(dc_el);
elseif cp_vfd_norm == 1
    for n1_nvfd_lut_ind = 1:size(n1_nvfd_lut,1)
        if n1_nvfd_lut(n1_nvfd_lut_ind,:) == cp_vfd_test
            break;
        end;
    end;
    if num_sides == 1
        ck_vfd_out = fperm(dc_el + n1_nvfd_lut_ind);
    elseif num_sides == 2 && cp_vfd_neg == -1
        ck_vfd_out = fperm(dc_el - n1_nvfd_lut_ind);
    else % 2-sided and positive
        ck_vfd_out = fperm(dc_el + n1_nvfd_lut_ind);
    end;
elseif % use hash tables...
    hash_table = hash_tables{cp_vfd_norm};
    hash_length = length(hash_table);
    cp_vfd_nzi = find(cp_vfd_test);
    cp_vfd_hval = abs(ht(cp_vfd_nzi)*transpose(cast(cp_vfd_test( cp_vfd_nzi),'double')));
    hash_table_addr = floor(hash_length*rem(cp_vfd_hval*A_hash,1))+1;
    if (hash_table_addr < 1) || (hash_table_addr > hash_length)
        error('specmap :: hash table address out of bounds');
    end;
    hash_nvfd_vect = hash_table{hash_table_addr};
    hash_nvfd_nonz_ind = find(hash_nvfd_vect == 0);
    if ~isempty(hash_nvfd_nonz_ind)
        hash_nvfd_addr = hash_nvfd_vect(hash_nvfd_nonz_ind);
        for i = 1:length(hash_nvfd_addr)
            if ht_nvfd(hash_nvfd_addr(i),:) == cp_vfd_test
                if num_sides == 1
                    ck_vfd_out = fperm(dc_el + hash_nvfd_addr(i));
                elseif num_sides == 2 && cp_vfd_neg == -1
                    ck_vfd_out = fperm(dc_el - hash_nvfd_addr(i));
                else % 2-sided and positive
                    ck_vfd_out = fperm(dc_el + hash_nvfd_addr(i));
                end;
                break;
            end;
function [specmap, specmap_length] = local_alloc_specmap(specmap, Xmag_nz, fvd_length, num_sides);
% initial call; alloc size equal to length of fvd
alpha = 0.25;  % Guess on factor (between 0 and 1) denoting number of conv prods with correct norm.
specmap_length = fix(alpha*Xmag_nz*fvd_length);
if num_sides == 1
    specmap = zeros(5, specmap_length);
else
    specmap = zeros(3, specmap_length);
end;
else
    curr_length = size(specmap, 2);
    new_length = 2*size(specmap, 2);
    if num_sides == 1
        new_specmap = zeros(5, new_length);
    else
        new_specmap = zeros(3, new_length);
    end;
    new_specmap(:, 1:curr_length) = specmap;
clear specmap;
specmap = new_specmap;
specmap_length = size(specmap, 2);
clear new_specmap;
end;
return;

F.2.7 AOM Spectrum Transform Matrix Function

Listing F.7 is the AOM Toolbox function to create a spectrum transform matrix.
Listing F.7: AOM Toolbox Spectrum Transform Matrix Function.

```matlab
function [Tx,t_ex]=aom_spectrans(X,specmap,ffreq_index,Tx_namestr);
% aom_spectrans creates a sparse spectrum transform matrix
% aom_spectrans creates a sparse spectrum transform matrix using Matlab's
% sparse matrix format. The code requires an input spectral vector and
% spectrum mapping table and the frequency–sorted index vector, and it
% takes an optional string describing the object being built.

% Inputs: X – spectral vector
% specmap – spectrum mapping table
% ffreq_index – frequency–sorted index vector
% Tx_namestr – string denoting the matrix being built

% Outputs: Tx – spectrum transform matrix corresponding to X
% t_ex – time of toolbox function execution

% North Carolina State University
% AOM Toolbox Ver. 2.00
% Copyright Frank P. Hart, 2005–2008

% Snapshot time for returning time of execution t_ex
dt = cputime();
fprintf(['Entering Spectrum Transform Matrix Generating Code...

if (nargin < 3) | (nargin > 4)
    error('sspectrans: Number of input arguments must be at least 3 and
            no more than 4');
end;

if ffreq_index(1) == 0 % Determine whether system is 1–S or 2–S
    num_sides = 1; % 1–S if DC is the first element
else num_sides = 2; % ...otherwise 2–S
end;

waitstr = ['Computing Spectrum Transform Matrix from ',inputname(1)];
if nargin == 4 waitstr = ['Computing Spectrum Transform Matrix ',
    Tx_namestr,' from ',inputname(1)]; end;
waitprog_count=0;
h = waitbar(waitprog_count,waitstr);
if num_sides == 2 % 2–Sided spectrum. X_dci is non–zero and sign(k –) can be + or –
	X_length = size(X,1);
	X_dci = (X_length-1)/2 + 1; % DC is the middle element of X (whether real or complex)
	Nrows = size(specmap,1); %Find num of rows in spectrum mapping table
else
    if Nrows > 100 waitprog_incr = Nrows/100; % Use this to hold the
```
waitbar updates to 100
else waitprog_incr = 1;
end;
if isreal(X)
    Tx = spalloc(X_length, X_length, Nrows);
    for row = 1:Nrows
        ky = specmap(row, 1); kx = specmap(row, 2); kz = specmap(row, 3);
        X_kx_r = X(X_dc + 2*kx.*sign(kx)); % real part of the kx component of X
        if kx > 0
            X_kx_i = X(X_dc + 2*kx); % imag part of the kx component of X.
        else
            X_kx_i = 0;
        end;
        if (ky > 0) & (kz > 0) % Non-DC, update 4 entries
            Tx(X_dc + 2*ky.*sign(ky), X_dc + 2*kz.*sign(kz)) = ...;
            Tx(Xdc + 2*ky.*sign(ky), Xdc + 2*kz.*sign(kz)) + X_kx_r;
            Tx(Xdc + 2*ky.*sign(ky), Xdc + 2*kz) = ...;
            Tx(Xdc + 2*ky, Xdc + 2*kz.*sign(kz)) = ...;
            Tx(Xdc + 2*ky, Xdc + 2*kz) = ...;
            Tx(Xdc + 2*ky, Xdc + 2*kz) + X_kx_r;
        elseif (ky > 0) & (kz == 0) % Column entries at DC column
            Tx(Xdc + 2*ky.*sign(ky), Xdc) = ...;
            Tx(Xdc + 2*ky, Xdc) + X_kx_r;
            Tx(Xdc + 2*ky, Xdc) = ...;
            Tx(Xdc + 2*ky, Xdc) + X_kx_r;
        elseif (ky == 0) & (kz > 0) % Row entries at DC row
            Tx(Xdc, Xdc + 2*kz.*sign(kz)) = ...;
            Tx(Xdc, Xdc + 2*kz.*sign(kz)) + X_kx_r;
            Tx(Xdc, Xdc + 2*kz) = ...;
            Tx(Xdc, Xdc + 2*kz) = ...;
        else % kx == 0 and kz == 0, i.e., DC element
            Tx(Xdc, Xdc) = Tx(Xdc, Xdc) + X_kx_r;
        end;
    if row > waitprog_incr*waitprog_count
        waitprog = row/Nrows;
        if waitprog_incr == 1
            waitprog_count = waitprog*100;
        else waitprog_count = waitprog_count + 1;
        end;
        waitbar(waitprog_count/100);
    end;
else % complex spectral vector yielding complex Tx
    max_X_i = max(imag(X));
    X_i_zero = (max_X_i == 0.0);
    if X_i_zero % all zeros, so seed a phantom im value...
        X_dc_re = real(X(X_dc));
        X(X_dc) = complex(X_dc_re, eps);
        X = X + complex(0.0, eps);
end;
% for each row of the sp_map_tab
ky = specmap(row,1); kx = specmap(row,2); kz = specmap(row,3);
X_kx_re = real(X(X_dci+kx)); X_kx_im = imag(X(X_dci+kx));
Tx(X_dci+ky, X_dci+kz) = Tx(X_dci+ky, X_dci+kz) + complex(X_kx_re, X_kx_im);
if row > waitprog_incr*waitprog_count
    waitprog = row/Nrows;
    if waitprog_incr == 1
        waitprog_count = waitprog*100;
    else waitprog_count = waitprog_count + 1;
    end;
    waitbar(waitprog_count/100);
end; %for row=1:Nrows
end; %if isreal(X)
end; % if num_sides == 2
if num_sides == 1 % Single-sided spectra - done as special case of 2-sided code with X_dci=sign(k)=1
    Nrows = size(specmap,1); %Find num of rows in spectrum
    mapping table
    if Nrows > 100 waitprog_incr = Nrows/100; % Use this to hold the
    waitbar updates to 100
    else waitprog_incr = 1;
    end;
    Tx = spalloc(size(X,1),size(X,1),Nrows);
    if isreal(X)
        for row = 1:Nrows % for each row of the sp_map_tab
            ky = specmap(row,1);
            kx = specmap(row,2); sx = specmap(row,3);
            kz = specmap(row,4); sz = specmap(row,5);
            if kx != 0
                X_kx_r = X(2*kx); % real part of the kx component of X.
                X_kx_i = X(2*kx+1); % imag part of the kx component of X
            else % kx == 0
                X_kx_r = X(1);
                X_kx_i = 0.0;
            end;
            if ( (kx != 0)&(kz != 0) ) % Don't use this scale factor on
                epsilon = 0.5;
            else
                epsilon = 1.0;
            end %if kx!=0 and ky!= 0
            if (ky != 0) & (kz != 0) % Non-DC, update 4 entries
                Tx(2*ky , 2*kz) = ... 
                Tx(2*ky , 2*kz) + epsilon*X_kx_r;
                Tx(1+2*ky , 2*kz) = ... 
                Tx(1+2*ky , 2*kz) + sx*epsilon*X_kx_i;
                Tx(2*ky , 1+2*kz) = ...
Listing F.8 is the AOM Toolbox function that evaluates a nonlinear polynomial function.

Listing F.8: AOM Toolbox Sparse Block Evaluation Function.

```matlab
function [Tx, t_ex] = aom_spectrans(X, specmap, ffrq_index, Tx_namestr);
% aom_spectrans creates a sparse spectrum transform matrix
% aom_spectrans creates a sparse spectrum transform matrix using
```
sparse matrix format. The code requires an input spectral vector and spectrum mapping table and the frequency-sorted index vector, and it takes an optional string describing the object being built.

% Inputs: X - spectral vector
% specmap - spectrum mapping table
% ffreq_index - frequency-sorted index vector
% Tx_namestr - string denoting the matrix being built

% Outputs: Tx - spectrum transform matrix corresponding to X
% t_ex - time of toolbox function execution

% North Carolina State University
% AOM Toolbox Ver. 2.00
% Copyright Frank P. Hart, 2005-2008
%
% Snapshot time for returning time of execution t_ex
% begin = cputime;

fprintf(['nEntering Spectrum Transform Matrix Generating Code...\n']);

if (nargin < 3) | (nargin > 4)
    error('ssspectrans: Number of input arguments must be at least 3 and no more than 4');
end;

nside = 1;
if ffreq_index(1) == 0
    nside = 1;
end;

waitstr = ['Computing Spectrum Transform Matrix from ',inputname(1)];

if nargin == 4
    waitstr = ['Computing Spectrum Transform Matrix',
               ' from ',inputname(1)];
end;

waitprog_count=0;
h = waitbar(waitprog_count,waitstr);

if nside == 2
    Xdci = (X_length-1)/2 + 1; % DC is the middle element of X (whether real or complex)
else
    Xdci = size(X,1);
end;

Nrows = size(specmap,1); % Find num of rows in spectrum mapping table
if Nrows > 100
    waitprog_incr = Nrows/100; % Use this to hold the waitbar updates to 100
else
    waitprog_incr = 1;
end;

if isreal(X)
    Tx = spalloc(X_length,X_length,Nrows);
    for row = 1:Nrows
        ky = specmap(row,1); kx = specmap(row,2); kz = specmap(row,3)
X_{kx,r} = X(X_{dc}i+2+kx\cdot\text{sign}(kx)) \quad \% \text{real part of the } kx \text{ component of } X

\text{if } kx \neq 0
    X_{kx,i} = X(X_{dc}+2+kx) \quad \% \text{imag part of the } kx \text{ component of } X
\text{else}
    X_{kx,i} = 0;
\end{cases};
\end{align}

\begin{align}
\text{if } (ky \neq 0) & \land (kz \neq 0) \quad \% \text{Non-DC, update 4 entries}
\begin{align}
    & Tx(X_{dc}i+2+ky\cdot\text{sign}(ky), X_{dc}i+2+kz\cdot\text{sign}(kz)) = \
    & Tx(X_{dc}i+2+ky\cdot\text{sign}(ky), X_{dc}i+2+kz\cdot\text{sign}(kz)) + X_{kx,r};
    & Tx(X_{dc}i+2+ky\cdot\text{sign}(ky), X_{dc}i+2+kz) = \
    & Tx(X_{dc}i+2+ky\cdot\text{sign}(ky), X_{dc}i+2+kz) - X_{kx,i};
    & Tx(X_{dc}i+2+ky, X_{dc}i+2+kz\cdot\text{sign}(kz)) = \
    & Tx(X_{dc}i+2+ky, X_{dc}i+2+kz) + X_{kx,r};
    & Tx(X_{dc}i+2+ky, X_{dc}i) = Tx(X_{dc}i+2+ky, X_{dc}i) + X_{kx,i};
\end{align}
\text{elseif } (ky \neq 0) & \land (kz = 0) \quad \% \text{Column entries at DC column}
\begin{align}
    & Tx(X_{dc}i+2+ky\cdot\text{sign}(ky), X_{dc}i) = \
    & Tx(X_{dc}i+2+ky\cdot\text{sign}(ky), X_{dc}i) + X_{kx,r};
    & Tx(X_{dc}i+2+ky, X_{dc}i) = \
    & Tx(X_{dc}i+2+ky, X_{dc}i) - X_{kx,i};
\end{align}
\text{elseif } (ky = 0) & \land (kz \neq 0) \quad \% \text{Row entries at DC row}
\begin{align}
    & Tx(X_{dc}i, X_{dc}i+2+kz\cdot\text{sign}(kz)) = \
    & Tx(X_{dc}i, X_{dc}i+2+kz\cdot\text{sign}(kz)) + X_{kx,r};
    & Tx(X_{dc}i, X_{dc}i+2+kz) = \
    & Tx(X_{dc}i, X_{dc}i+2+kz) - X_{kx,i};
\end{align}
\text{else } \% kx = 0 \text{ and } kz = 0, \text{ i.e., DC element}
\begin{align}
    & Tx(X_{dc}i, X_{dc}i) = Tx(X_{dc}i, X_{dc}i) + X_{kx,r};
\end{align}
\end{cases};
\end{align}

\text{if } row > \text{waitprog_incr} * \text{waitprog_count}
\begin{align}
    & \text{waitprog} = \text{row}/\text{Nrows};
    & \text{if } \text{waitprog_incr} == 1
    & \begin{align}
        & \text{waitprog_count} = \text{waitprog} * 100;
    \end{align}
    & \text{else } \text{waitprog_count} = \text{waitprog_count} + 1;
    & \text{end};
    & \text{waitbar(\text{waitprog_count}/100)};
    & \text{end}; \% \text{for row} = 1:\text{Nrows}
\end{align}
\text{else } \% \text{complex spectral vector yielding complex } Tx
\begin{align}
    & \text{max}X_{im} = \text{max}(\text{imag}(X));
    & X_{im_zero} = (\text{max}X_{im} == 0.0);
    & \text{if } X_{im_zero} \% \text{all zeros, so seed a phantom im value...}
    & X_{dc ре} = \text{real}(X(X_{dc}i));
    & X(X_{dc}i) = \text{complex}(X_{dc ре}, \text{eps});
    & X = X + \text{complex}(0.0, \text{eps});
    & \text{end};
\end{align}
\begin{align}
    & Tx = \text{spalloc}(\text{X_length}, \text{X_length}, \text{Nrows});
    & \text{for row} = 1:\text{Nrows} \quad \% \text{for each row of the sp_map_tab}
    & \text{ky} = \text{specmap(row,1)}; \text{ kx} = \text{specmap(row,2)}; \text{ kz} = \text{specmap(row,3)};
    & \begin{align}
        & X_{kk ре} = \text{real}(X(X_{dc}i+kx)); X_{kk им} = \text{imag}(X(X_{dc}i+kx));
        & Tx(X_{dc}i+ky, X_{dc}i+kz) = Tx(X_{dc}i+ky, X_{dc}i+kz) + \text{complex}(}
\[ X_{\kx \text{re}}, X_{\kx \text{im}} \];

if row > waitprog_incr \cdot \text{waitprog\_count}
  \begin{align*}
    \text{waitprog} &= \text{row}/\text{Nrows}; \\
    \text{if } & \text{waitprog\_incr }= 1 \\
    \text{waitprog\_count} &= \text{waitprog } \cdot 100; \\
    \text{else } & \text{waitprog\_count} = \text{waitprog\_count }+ 1; \\
  \end{align*}
end;

\text{waitbar}(\text{waitprog\_count}/100);
end;
\%for row = 1:Nrows
end; \%if isreal(X)
end; \%if num\_sides == 2
\%
\% Single\_-sided spectra – done as special case of 2\_-sided code with X_{\text{dci}}=\text{sign} (k) = 1
\begin{align*}
  \text{Nrows} &= \text{size}(\text{specmap} , 1); \\
  \% \text{Find num of rows in spectrum} \\
  \text{if } \text{Nrows } > 100 \text{ waitprog\_incr }= \text{Nrows}/100; \% \text{Use this to hold the} \\
  \text{waitbar updates to 100} \\
  \text{else } & \text{waitprog\_incr }= 1; \\
\end{align*}
\%
\text{Tx} = \text{spalloc}(\text{size}(X,1), \text{size}(X,1), \text{Nrows});
\text{if isreal(X)}
\begin{align*}
  \%\text{for each row of the sp\_map\_tab} \\
  \text{for row } = 1: \text{Nrows} \\
  \text{ky} &= \text{specmap(row,1)}; \\
  \text{kk} &= \text{specmap(row,2)}; \text{ sx} &= \text{specmap(row,3)}; \\
  \text{kz} &= \text{specmap(row,4)}; \text{ sz} &= \text{specmap(row,5)}; \\
  \text{if } & \text{kk }\neq 0 \\
  X_{\kx \text{re}} &= X(2 \cdot \text{kk}); \% \text{real part of the kk component of X.} \\
  X_{\kx \text{im}} &= X(2 \cdot \text{kk }+ 1); \% \text{imag part of the kk component of} \\
  X \\
  \text{else } & \% \text{kk }= 0 \\
  X_{\kx \text{re}} &= X(1); \\
  X_{\kx \text{im}} &= 0.0; \\
\end{align*}
\%
\text{if } ( (\text{kk }\neq 0) \& (\text{kk }\neq 0) ) \% \text{Don't use this scale factor on} \\
\text{2\_-sided spectra} \\
\text{epsilon} &= 0.5; \\
\text{else} \\
\text{epsilon} &= 1.0; \\
\text{end} \%\text{if kk}! = 0 \text{ and ky}! = 0
\begin{align*}
  \text{if (ky }\neq 0) \& (\text{kk }\neq 0) \% \text{Non-DC, update 4 entries} \\
  \text{Tx} & (2 \cdot \text{ky}, 2 \cdot \text{kk}) = \ldots \\
  & \text{Tx} (2 \cdot \text{ky}, 2 \cdot \text{kk}) + \text{epsilon} \cdot X_{\kx \text{re}}; \\
  \text{Tx} (1+2 \cdot \text{ky}, 2 \cdot \text{kk}) = \ldots \\
  & \text{Tx} (1+2 \cdot \text{ky}, 2 \cdot \text{kk}) + \text{sx} \cdot \text{epsilon} \cdot X_{\kx \text{im}}; \\
  \text{Tx} (2 \cdot \text{ky}, 1+2 \cdot \text{kk}) = \ldots \\
  & \text{Tx} (2 \cdot \text{ky}, 1+2 \cdot \text{kk}) + \text{sx} \cdot \text{sz} \cdot \text{epsilon} \cdot X_{\kx \text{im}}; \\
  \text{Tx} (1+2 \cdot \text{ky}, 1+2 \cdot \text{kk}) = \ldots \\
  & \text{Tx} (1+2 \cdot \text{ky}, 1+2 \cdot \text{kk}) + \text{sz} \cdot X_{\kx \text{re}}; \\
\text{elseif (ky }\neq 0) \& (\text{kk }= 0) \% \text{Column entries at DC column} \\
  \text{Tx} & (2 \cdot \text{ky}, 1) = \ldots \\
  & \text{Tx} (2 \cdot \text{ky}, 1) + \text{epsilon} \cdot X_{\kx \text{re}}; \\
  \text{Tx} (1+2 \cdot \text{ky}, 1) = \ldots
Listing F.9 is the AOM Toolbox function that creates and (optionally) scales a unit DC spectral vector for use in function evaluations.

Listing F.9: AOM Toolbox Unit DC Spectral Vector Function.

function [Ydc, t_ex] = aom_unitdc(X, ffreq_index, polycoeff);

% aom_unitdc creates a unit vector suitable for use as a DC term
%
% aom_unitdc creates a spectral vector containing a unit value
% that is suitable for adding unit DC offsets. When a polynomial
% transfer function is optionally supplied, the unit value is
% scaled by the polynomial coefficient a_0.
%
% Inputs: X - spectral vector
% ffreq_index - frequency-sorted index vector
% polycoeff - polynomial coefficients; a_0 is used to scale
% Snapshot time for returning time of execution t_ex
% Begin
\begin{verbatim}
    t_begin = cputime;

tprintf(['Entering Output DC Evaluation Code...\n']);
if ( nargin < 2 ) | ( nargin > 3 )
    error('unitdc :: Number of input arguments must be at least 2 and no more than 3');
end;

% Set the scale factor
if ( nargin == 3 )
a_0 = polycoeff(end);
else
    a_0 = 1.0;
end;

% Figure out if one-sided or two...
if ffreq_index(1) == 0 % Determine whether system is 1-S or 2-S
    num_sides = 1; % 1-S if DC is the first element
else
    num_sides = 2; % ...otherwise 2-S
end;

% Check for real/complex X, kick out if necessary
X_real = isreal(X);
if ( num_sides == 1 ) & ~X_real % X complex - can't have 1-sided
    error('unitdc :: One-sided complex spectral vector not permitted.');
end;

% Length of spectral vectors and DC element location
sv_length = size(X,1);
if num_sides == 1
    sv_dci = 1;
else
    sv_dci = (sv_length-1)/2 + 1;
end;

% Allocate storage for outputs...
Ydc = zeros(sv_length,1); % storage for linear, second, etc responses in cols...

% Set the DC value....
Ydc(sv_dci) = a_0;

tprintf(['Exiting Output DC Evaluation Code...\n']);
\end{verbatim}

\texttt{t_ex = cputime - t_begin;}
\texttt{return;}

F.2.10  AOM Ideal Bandpass Filter Function

Listing F.10 is the AOM Toolbox function that performs ideal bandpass filtering.

```
function Y = aom_ideal_bpf(X, ffreq, ffreq_index, arg4, arg5)

% aom_ideal_bpf performs ideal bandpass filtering of spectral vectors
% aom_ideal_bpf is an ideal bandpass filtering function that takes
% a spectral vector and its corresponding ffreq and ffreq_index along
% with two additional arguments and returns the filtered spectral
% vector as an output. The two additional arguments are interpreted
% as follows:

1) If arg5 > arg4 and both arg4 and arg5 are greater than 100,
   then arg4 and arg5 are treated as the ideal band-edges.
   Spectral content at arg4 and arg5 and between will be passed, the rest
   rejected.
   Prototype form for this case:
   function Y = aom_ideal_bpf(X, ffreq, ffreq_index, f_start, f_stop)

2) If arg5 < arg4 and arg5 < 100, then arg4 is treated as a center
   frequency and arg5 is treated as a percentage bandwidth about
   the center frequency.
   function Y = aom_ideal_bpf(X, ffreq, ffreq_index, f_cen, bw_pct)

3) If arg5 < arg4 and arg5 > 100, then arg4 is treated as a center
   frequency and arg5 is treated as an absolute bandwidth about the
   center frequency.
   function Y = aom_ideal_bpf(X, ffreq, ffreq_index, f_cen, bw_abs)

For ideal lowpass operation, specify start_freq = 0.0 and stop_freq
   = the cutoff frequency.

% Inputs:
X - spectral vector in column format (variable # rows, 1 column)
ffreq - column format vector of frequencies correspondig to
  phasors in X
ffreq_index - column format vector of indices for ffreq
f_start (arg4) - starting frequency for passband
f_stop (arg5) - stopping frequency for passband
f_cen (arg4) - center frequency of passband
bw_pct (arg5) - bandwidth percentage about f_cen
bw_abs (arg5) - absolute bandwidth about f_cen

% Outputs:
Y - spectral vector in column format
```
if nargin < 5
  error('ideal_bpf :: The number of input arguments required is 5.');
end;

if ffreq_index(1) == 0
  num_sides = 1;
else
  num_sides = 2;
end;

if arg5 < arg4
  if arg5 < 100.0
    center_freq = arg4;
    bandwidth = (arg5/100.0)*center_freq;
    start_freq = center_freq - bandwidth/2.0;
    stop_freq = center_freq + bandwidth/2.0;
  else
    center_freq = arg4;
    bandwidth = arg5;
    start_freq = center_freq - bandwidth/2.0;
    stop_freq = center_freq + bandwidth/2.0;
  end;
else
  start_freq = arg4;
  stop_freq = arg5;
end;

H = ones(size(ffreq));

if num_sides == 1
  if start_freq == 0.0
    H(1) = 1;
else
    H(1) = 0;
end;
for k = 2:length(ffreq)
  if ffreq(k) < start_freq
    H(k) = 0;
  elseif ffreq(k) > stop_freq
    H(k) = 0;
  else
    ; % do nothing
  end
end;
else
  ffi_dc_el = (length(ffreq)-1)/2 + 1;
if start_freq == 0.0
    H ffi_dc_el = 1;
else
    H ffi_dc_el = 0;
end;
for k = 1:ffi_dc_el-1
    if ffreq(k) < -stop_freq
        H(k) = 0;
    elseif ffreq(k) > -start_freq
        H(k) = 0;
    else
        ;
    end
end;
for k = ffi_dc_el+1:length(ffreq)
    if ffreq(k) < -start_freq
        H(k) = 0;
    elseif ffreq(k) > stop_freq
        H(k) = 0;
    else
        ; % do nothing
    end
end;
end;
X_real = isreal(X);
if ~X_real & (num_sides == 1)
    error('ideal_bpf :: One-sided complex spectral vectors not permitted.');
end;

% If spec vector is real, go to complex form for filtering
if X_real
    Xc = aom_svtocv(X, ffreq_index);
else
    Xc = X;
end;
Yc = zeros(size(Xc));
Yc = H.*Xc;

% Done with filtering, if X was real put Y back into real form.
if X_real
    Y = aom_cvtsv(Yc, ffreq_index);
else
    Y = Yc;
end;
return;
F.2.11  AOM Butterworth Bandpass Filter Function

Listing F.11 is the AOM Toolbox function that performs Butterworth bandpass filtering.

Listing F.11: AOM Toolbox Butterworth Bandpass Filter Function.

function [Y, fls, fhs] = aom_butter_bpf(X, ffreq, ffreq_index, arg4, arg5, arg6, arg7, arg8, arg9)
% aom_butter_bpf is a selectable order (default 2) Butterworth filtering
% function that takes a spectral vector and its corresponding ffreq
% and ffreq_index along with a passband edge frequency and its
% corresponding
% attenuation level (usually a flatness specification) as inputs and
% returns
% the filtered spectral vector as an output. The number of poles may
% be
% implicitly (5 arg prototype) or explicitly (6 arg prototype) or may
% be
% determined by also supplying the band-reject edge frequency and its
% attenuation.
% Function prototype forms:
%
% function Y = aom_butter_bpf(X, ffreq, ffreq_index, arg4, arg5, arg6, arg7, arg8, arg9)
% function Y = aom_butter_bpf(X, ffreq, ffreq_index, fls, flp, fhp, fhs, pbfdb, sbadb)
% function Y = aom_butter_bpf(X, ffreq, ffreq_index, fo, pbw, Np, pbfdb)
% function Y = aom_butter_bpf(X, ffreq, ffreq_index, fo, pbw, Np)
% If 6 arguments are passed, pbfdb is assumed to be -3.
% fls and fhs are optionally returned and are useful for fREEDA
% comparison when fo,pbw,Np used...
% Inputs:
X  - spectral vector in column format (variable # rows, 1 column)
ffreq  - column format vector of frequencies correlated to X
ffreq_index  - column format vector of indices for ffreq
fls (arg4)  - low-side stopband edge frequency for which sbadb is
specified
flp (arg5)  - low-side passband edge frequency for which pbfdb is
specified
fhp (arg6)  - high-side passband edge frequency for which pbfdb
is specified
fhs (arg7)  - high-side passband edge frequency for which sbadb
is specified
pbfdb (arg8)  - attenuation (vs. DC) at the passband edge
frequencies
sbadb (arg9)  - attenuation (vs. DC) at the stopband edge
frequencies
30  %  fo (arg4) = center frequency of passband
31  %  pbw (arg5) = passband bandwidth (% if < 100 or Hz otherwise)
32  %  Np (arg6) = Number of lowpass prototype poles
33  %
34  % North Carolina State University
35  % AOM Toolbox Ver. 2.00
36  % Copyright Frank P. Hart, 2005–2008
37  %
38
39  % error checks for proper function prototypes...
40  if ~((nargin == 9 | nargin == 6 | nargin == 7))
41     error ('butter_bpf :: Must specify 6, 7, or 9 input arguments');
42  end
43
44  if nargin == 9
45      fls = arg4; flp = arg5; fhp = arg6; fhs = arg7; pbfdb = arg8; sbadb = arg9;
46      if pbfdb > 0
47         error ('butter_bpf :: Passband flatness must be specified in negative dB');
48      end
49      if sbadb > 0
50         error ('butter_bpf :: Stopband relative amplitude must be specified in negative dB');
51      end
52
53  if fhp == 0.0
54     error ('butter_bpf :: Upper passband frequency limit must be specified');
55  end
56
57  if fhs == 0.0
58     error ('butter_bpf :: Upper stopband frequency limit must be specified');
59  end
60
61  % Convert specs in dB to absolutes...
62  if pbfdb == -3.0
63      epsilon = 1.0;
64  else
65      epsilon = sqrt(10.0^(-pbfdb/10.0)-1);
66  end
67
68  lambda = sqrt(10.0^(-sbadb/10.0)-1);
69  % Convert specified edge frequencies (in Hz) to radian frequencies
70  wls = 2*pi*fls; wlp = 2*pi*flp; whp = 2*pi*fhp; whs = 2*pi*fhs;
71  % Determine the prototype LP filter 's pass- and stop-band frequencies
72  wo2 = wlp*whp; % omega–naught squared
73  wo = sqrt(wo2);
74  wp = (whp^2.0 - wo2)/whp; % prototype passband, would be same if
75     (wo2 - wlp^2.0)/wlp
76  wsh = (whs^2.0 - wo2)/whs;
77  wsl = (wo2 - wls^2.0)/wls;
78  if wsh > wsl % determine stop band
79      ws = wsl; % ws = wsh;
80  else
81      ws = wsh; % ws = wsl;
% Now determine the prototype LP filter's order Np and cutoff freq
Np = ceiling(log10(epsilon/lambda)/log10(wp/ws));

if nargin == 6 pbfdb = -3.0; end;
if nargin == 7 pbfdb = arg7; end;
if ( nargin == 6 | nargin == 7)
fo = arg4; pbw = arg5; Np = arg6;
if pbfdb > 0
    error('butter_bpf :: Passband flatness must be specified in negative dB');
end
if fo == 0
    error('butter_bpf :: Passband center frequency must be specified');
end;
if pbw == 0
    error('butter_bpf :: Passband bw (pct if < 100, Hz else) must be specified');
end;
if Np == 0
    error('butter_bpf :: Number of poles in lowpass prototype must be specified');
end;
if pbfdb == -3.0
    epsilon = 1.0;
else
    epsilon = sqrt(10.0^(-pbfdb/10.0)-1);
end
if pbw < 100
    wp = (pbw/100)*2*pi*fo;
else
    wp = 2*pi*pbw;
end
wo2 = (2*pi*fo)^2;
sbadb = -10;
lambda = sqrt(10.0^(-sbadb/10.0)-1);
flp = -pbw/2.0 + sqrt((pbw*pbw/4) + fo*fo);
fhp = pbw/2.0 + sqrt((pbw*pbw/4) + fo*fo);
sbw = pbw*(lambda/epsilon)^(1/Np);
ws = wp*(lambda/epsilon)^(1/Np);
fls = -sbw/2.0 + sqrt((sbw*sbw/4) + fo*fo);
fhls = sbw/2.0 + sqrt((sbw*sbw/4) + fo*fo);
end;
wc = wp/(epsilon^(1.0/Np));

% Determine whether order is even or odd, set boolean var Nodd == mod(N,2)
if mod(Np,2) == 0
    Nodd = 0;
else
    Nodd = 1;
end
% Set max iterate for pole (pair) computations
if Nodd
    kmax = (Np+1)/2;
else
    kmax = Np/2;
end

% poles
for k = 0:(Np-1)/2;
    ind = k+1;
    u(ind) = wc*(-sin((2*k+1)*pi/(2*Np)));
    v(ind) = wc*(cos((2*k+1)*pi/(2*Np)));
    p(ind) = u(ind) + complex(0,1)*v(ind);
end;

pc = conj(p);
A = wc^Np;

% Do the frequency translation from lowpass to bandpass...
sl = complex(0,1)*2*pi*ffreq;
s = (sl.*s1 + wo2*ones(size(s1)))./s1;
H = A;
for k = 0:(Np-1)/2
    ind = k+1;
    if k * (Np-1)/2
        H = H.*(1./((s - p(ind)).*(s - pc(ind))));
    else
        H = H.*(1./s - p(ind)));
end;

magH = abs(H);
phH = exp(complex(0,1)*angle(H));

% Figure out if spectral vector is 1 sided or 2...
if ffreq_index(1) == 0
    num_sides = 1;
else
    num_sides = 2;
end;

X_real = isreal(X);
if ~X_real & (num_sides == 1)
    error('butter_bpf :: One-sided complex spectral vectors not permitted.');
end;

% If spec vector is real, go to complex form for filtering
if X_real
    Xc = aom_svtocv(X, ffreq_index);
else
    Xc = X;
end;

Yc = zeros(size(Xc));
[m,n] = size(Xc);
for i = 1:n
    Yc(:,i) = magH.*Xc(:,i).*phH;
end;

% Done with filtering, if X was real put Y back into real form.
if X_real
    Y = aom_cvtosv(Yc, ffreq_index);
else
    Y = Yc;
end;
return;

F.2.12 AOM Butterworth Lowpass Filter Function

Listing F.12 is the AOM Toolbox function that performs Butterworth lowpass filtering.

Listing F.12: AOM Toolbox Butterworth Lowpass Filter Function.

function Y = aom_butter_lpf(X, ffreq, ffreq_index, fe, ea_db, arg6, arg7)
  % aom_butter_lpf performs Butterworth lowpass filtering of spectral vectors
  % aom_butter_bpf is a selectable order (default 2) Butterworth filtering
  % function that takes a spectral vector and its corresponding ffreq
  % and ffreq_index along with a passband edge frequency and its corresponding
  % attenuation level (usually a flatness specification) as inputs and returns
  % the filtered spectral vector as an output. The number of poles may be
  % implicitly (5 arg prototype) or explicitly (6 arg prototype) or may be
  % determined by also supplying the band-reject edge frequency and its attenuation.
  % Function prototype forms:
  %
  % function Y = aom_butter_lpf(X, ffreq, ffreq_index, fe, ea_db, fe_r, ra_db)
  % function Y = aom_butter_lpf(X, ffreq, ffreq_index, fe, ea_db, Np)
  % function Y = aom_butter_lpf(X, ffreq, ffreq_index, fe, ea_db)
  % When only 5 arguments are passed, Np is set to 2.
Inputs:

- X - spectral vector in column format (variable # rows, 1 column)
- ffreq - column format vector of frequencies correlated to X
- ffreq.index - column format vector of indices for ffreq
- f.e - scalar - edge frequency at which passband attenuation is defined
- ea_db - scalar - attenuation (vs. DC) at the band-edge frequency
- f.r - scalar - edge frequency at which rejection band is defined
- ra_db - scalar - attenuation (vs. DC) at the band-reject frequency

Outputs:

- Y - filtered spectral vector

North Carolina State University

AOM Toolbox Ver. 2.00

Copyright Frank P. Hart, 2005–2008

if nargin == 7
    f.r = arg6; ra_db = arg7;
    if ra_db > 0
        error('butter_lpf :: The band-reject edge attenuation must be a negative dB value');
    end;
    ra = sqrt(10^(-ra_db/10)-1);
    if f.r <= 0
        error('butter_lpf :: The band-reject edge attenuation frequency must be positive');
    end;
    ea = sqrt(10^(-ea_db/10)-1);
    Np = ceil( log10(ea/ra) / log10(f.e/f.r) );
elseif nargin == 6
    ea = sqrt(10^(-ea_db/10)-1);
    Np = arg6;
elseif nargin == 5
    ea = sqrt(10^(-ea_db/10)-1);
    Np = 2;
elseif nargin < 5
    error('butter_lpf :: The number of input arguments required is 5 or 6.');
end;

if Np < 1
    error('butter_lpf :: The number of poles must be ≥ 1');
end;
if ea_db >= 0
    error('butter_lpf :: The band-pass edge attenuation must be a negative dB value');
end;
if f.e <= 0
    error('butter_lpf :: The band-pass edge attenuation frequency must be positive');
omega_e = 2*pi*f_e;
if ea_db == -3
    omega_c = omega_e;
else
    omega_c = omega_e/(ea*(1/Np));
end;

% poles
for k = 0:(Np-1)/2;
    ind = k+1;
    u(ind) = omega_c*(-sin((2*k+1)*pi/(2*Np)));
    v(ind) = omega_c*(cos((2*k+1)*pi/(2*Np)));
    p(ind) = u(ind) + complex(0,1)*v(ind);
end;

pc = conj(p);
%A = abs(p)'^2;
A = omega_c^Np;
s = complex(0,1)*2*pi*ffreq;
H = A;
for k = 0:(Np-1)/2
    ind = k+1;
    if k < (Np-1)/2
        H = H.*(1./(s - p(ind)).*(s - pc(ind)));
    else
        H = H.*(1./(s - p(ind)));
    end;
end;

magH = abs(H);
phH = exp(complex(0,1)*angle(H));

% Figure out if spectral vector is 1 sided or 2...
if ffreq_index(1) == 0
    num_sides = 1;
else
    num_sides = 2;
end;

X_real = isreal(X);
if ~X_real & (num_sides == 1)
    error('butter_lpf :: One-sided complex spectral vectors not permitted.');
end;

% If spec vector is real, go to complex form for filtering
if X_real
    Xc = aom_svtocv(X, ffreq_index);
else
    Xc = X;
end;

Yc = zeros(size(Xc));
Yc = magH.*Xc.*phH;
if X real
    Y = aom_cvtsv(Yc, ffreq_index);
else
    Y = Yc;
end;

return;

F.2.13 AOM Spectral to Complex Vector Function

Listing F.13 is an AOM Toolbox function that converts a spectral vector from real-valued to complex-valued form. It is called by the AOM Toolbox filtering functions and is available for end-user use as well.

Listing F.13: AOM Toolbox Spectral to Complex Vector Function.

function [cv, t_ex] = aom_cvtsv(real_vector, ffreq_index)
% aom_cvtsv converts a real-valued spectral vector to complex-valued form
% aom_cvtsv takes in a real-valued spectral vector and returns a complex-valued spectral vector equivalent by combining the two real-valued spectral vector elements for each non-zero frequency into a single complex-valued element (i.e. a phasor).
% The conversion routine gives the spectral element at DC a zero imaginary value by construction. Use of this routine permits complex vector manipulation by native Matlab functions. Use the aom_cvtsv routine to convert back to the real-valued AOM spectral vector form.

% Inputs:
% real_vector = real-valued spectral vector
% ffreq_index = frequency-sorted frequency index vector

% Outputs:
% cv = complex-valued spectral vector
% t_ex = execution time of the toolbox function

% North Carolina State University
% AOM Toolbox Ver. 2.00
% Copyright Frank P. Hart, 2005–2008

% Snapshot time for returning time of execution t_ex
27 t_begin = cputime;
28 fprintf(['Enter Real to Complex Spectral Vector Conversion Code...'
29 'n']);
30 error(nargcchk(1,2,nargin)); % Squawk if other than 1 or 2 input args.
31 end;
32
33 if nargin == 1 % If freq_index not passed, assume
34     num_sides = 1;
35 elseif freq_index(1) == 0 % Single-sided when first element of freq_index is 0
36     num_sides = 1;
37 else
38     num_sides = 2;
39 end;
40
41 sv_length = length(real_vector); % length of spectral vector
42 sv_length = mrows;
43 cv_freqs = (sv_length-1)/2.0; % # of non-zero freqs
44 cv_length = cv_freqs + 1; % add 1 to include DC
45 av = zeros(cv_length,ncols); % initialize real part
46 bv = zeros(cv_length,ncols); % initialize imag part
47
48 if num_sides == 1 % Single-sided
49     for n = 1:ncols
50         av(1,n) = real_vector(1,n); % First element is DC.
51         bv(1,n) = 0.0; % Im part of DC is zero.
52         for k = 1:cv_freqs
53             av(k+1,n) = real_vector(2*k,n);
54             bv(k+1,n) = real_vector(2*k+1,n);
55         end;
56     end;
57     cv = complex(av,bv);
58 end;
59
60 if num_sides == 2 % Double-sided
61     sv_dc_el = (sv_length-1)/2 + 1; % Middle element is DC
62     cv_dc_el = (cv_length-1)/2 + 1;
63     for n = 1:ncols
64         av(cv_dc_el,n) = real_vector(sv_dc_el,n);
65         bv(cv_dc_el,n) = 0.0;
66         for k = 1:cv_freqs/2
67             av(cv_dc_el+k,n) = real_vector(sv_dc_el+2*k-1,n);
68             bv(cv_dc_el+k,n) = real_vector(sv_dc_el+2*k,n);
69         end;
70     end;
71 end;

F.2.14 AOM Complex to Spectral Vector Function

Listing F.14 is an AOM Toolbox function that converts a complex-valued spectral vector to real-valued form. It it called by the AOM Toolbox filtering functions and is available for end-user use as well.


function [sv, t_ex] = aom_cvtosv(comp_vector, ffreq_index)
% aom_cvtosv converts a complex-valued spectral vector to real-valued form
% aom_cvtosv takes in a complex-valued spectral vector and returns
% a real-valued spectral vector equivalent by dividing each
% complex-valued phasor element for non-zero frequencies into two real-valued
% spectral vector elements. The conversion routine ignores imaginary values
% occurring at DC when they are in the vicinity of the machine epsilon as they
% are assumed to be artifacts of finite-precision arithmetic. (But larger
% imaginary values in the DC element cause an error and stop execution.)
% Inputs:
% comp_vector - complex-valued spectral vector
% ffreq_index - frequency-sorted frequency index vector
% Outputs:
% sv - real-valued spectral vector
% t_ex - execution time of the toolbox function
% North Carolina State University
% AOM Toolbox Ver. 2.00
% Copyright Frank P. Hart, 2005–2008

% Snapshot time for returning time of execution t_ex
% t_begin = cputime;

fprintf([ 'Entering Complex to Real Spectral Vector Conversion Code...

error(nargchk(1,2,nargin)); % Squawk if other than 1 or 2 input args.

% Modified so that sequence of Y vectors can be handled.
[mrows,ncols] = size(comp_vector);
if (mrows < 1) | (ncols < 1)
    error('cvtosv:: Complex vector must have at least two rows and/or columns. ');
end;

if nargin == 1 % If ffreq_index not passed, assume single-sided
    num_sides = 1; % May change later to double-sided default.
elseif ffreq_index(1) == 0 % Single-sided when first element of
    num_sides = 1;
else
    num_sides = 2;
end;

% cv_length = length(comp_vector); % length of complex vector DC+
nonzero freqs
% cv_length = mrows;
sv_length = 2*cv_length - 1; % only 1 element for DC
% cv_freqs = (sv_length-1)/2.0; % # of non-zero freqs
sv = zeros(sv_length,ncols); % initialize spectral vector

if num_sides == 1 % Single-sided
    for n = 1:ncols
        sv(1,n) = real(comp_vector(1,n)); % First element is DC
        if abs(imag(comp_vector(1,n))) > 1e-15 % Threshold the imag DC term in case some operation done...
            warning('cvtosv:: Complex vector has a non-zero DC imaginary part. ');
        end
    end;
    for k = 2:cv_length % see if can vectorize this per Higham.
        sv(2*(k-1),n) = real(comp_vector(k,n));
        sv(2*(k-1)+1,n) = imag(comp_vector(k,n));
    end;

if num_sides == 2  % Double-sided
        cv_dc_el = (cv_length-1)/2 + 1;
        sv_dc_el = (sv_length-1)/2 + 1;  % Middle element is DC
        for n = 1:ncols
            sv(sv_dc_el,n) = real(comp_vector(cv_dc_el,n));
            if abs(imag(comp_vector(cv_dc_el,n))) >= 1e-15  % Threshold the
                imag DC term in case some operation done...
                    warning('cvtosv :: Complex vector has a non-zero DC imaginary
                        part.');
                    warning('cvtosv :: Imaginary part will be chopped in
                        conversion.');
            end;
            for k = 1:cv_freqs/2
                sv(sv_dc_el+2*k-1,n) = real(comp_vector(cv_dc_el+k,n));
                sv(sv_dc_el+2*k,n) = imag(comp_vector(cv_dc_el+k,n));
                sv(sv_dc_el-2*k-1,n) = real(comp_vector(cv_dc_el-k,n));
                sv(sv_dc_el-2*k,n) = imag(comp_vector(cv_dc_el-k,n));
            end;
        end;
    end;
end;
end;

fprintf(['Exiting Complex to Real Spectral Vector Conversion Code...
']);
t_ex = cputime - t_begin;
return;

F.2.15 AOM Phasor Conversion Function

Listing F.15 is an AOM Toolbox function that converts complex numbers in Cartesian coordinates to polar form with higher accuracy than the built-in Matlab® atan() function by avoiding divide-by-zero and other boundary conditions that may result in finite-precision arithmetic error.

Listing F.15: AOM Toolbox Phasor Conversion Function.

function [mag,phase] = aom_doatan(Zr,Zi)
    % aom_doatan converts complex numbers to polar form with improved
    % accuracy vs. Matlab’s atan()
    % aom_doatan improves the accuracy of the built-in Matlab atan() function by identifying
5 % some exceptional inversion conditions (e.g. divide by zero) and
6 % returning the exact
7 % value rather than relying upon the output of atan(), which is
8 % subject to finite-precision
9 % error.
10 % Inputs: Zr – real part of the complex number to be inverted (or x–
11 % coordinate)
12 % Zi – imaginary part of the complex number to be inverted (or y–
13 % coordinate)
14 % Outputs: mag – magnitude of the complex number
15 % phase – angle of the complex number
16 % North Carolina State University
17 % AOM Toolbox Ver. 2.00
18 % Copyright Frank P. Hart, 2005–2008
19 %
20 mag = (Zrˆ2 + Ziˆ2)ˆ0.5; % Determine magnitude
21 if (mag ≠ 0)
22    if (Zr==0) & (Zi>0) % Non-zero phasor, zero real part, non
23       non-zero + im part
24       phase = pi/2; % ...so angle is 90 degrees
25    elseif (Zr==0) & (Zi<0) % Non-zero phasor, zero real part, non
26       non-zero - im part
27       phase = -pi/2; % ...so angle is -90 degrees
28    elseif (Zr<0) & (Zi==0) % Non-zero phasor, non-zero - real
29       part, zero im part
30       phase = pi; % ...so angle is 180 degrees
31    elseif (Zr>0) & (Zi==0) % Non-zero phasor, non-zero + real
32       part, zero im part
33       phase = 0; % ...so angle is 0 degrees
34 else % (abs(Zr)>0) & (Zi≠0) % Non-zero phasor, both real and im
35       phase = atan(Zi/(Zr+sign(Zr)*eps)); % ...so add eps
36       with same sign as real and call atan...
37 end
38 else % mag == zero, set phase to zero...
39    phase = 0;
40 end
41 return;

F.2.16 AOM Power to Voltage Conversion Function
Listing F.16 is an AOM Toolbox utility that converts a power specification in dBm to an
amplitude specification in volts.

Listing F.16: AOM Toolbox Power to Voltage Conversion Function.
Listing F.17: AOM Toolbox Voltage to Power Conversion Function.

Listing F.17 is an AOM Toolbox utility that converts a voltage specification in volts to a power specification in dBm.

```matlab
function A_dBm = aom_vtodbm(A_v, R_ref)
% aom_vtodbm returns the power in dBm given an input voltage
% Inputs: A_v - Input voltage amplitude
% R_ref - Reference impedance (optional, defaults to 50 ohms)
% Outputs: A_dBm - Output power in dBm (referenced to 1 milliwatt)
% This routine takes an input voltage amplitude and returns the power of
% that voltage in dBm. The reference impedance is assumed to be 50 ohms
% North Carolina State University
% AOM Toolbox Ver. 2.00
% Copyright Frank P. Hart, 2005-2008
% R_ref = 50.0;
if nargin == 2
    if isnumeric(R_ref)
        R_ref = 50.0;
    end
end
A_dBm = 10*log10( (A_v^2)/(R_ref^2) );
```

F.2.17 AOM Voltage to Power Conversion Function

Listing F.17 is an AOM Toolbox utility that converts a voltage specification in volts to a power specification in dBm.
Listing F.18: Script demonstrating the dynamic range of the AOM Toolbox.

1 % This script computes the response to a two-tone input of a transfer
2 % function modelling the AR 10W1000C amplifier. Used to demonstrate
3 % the dynamic range of AOM.
4 %
5 % North Carolina State University
6 % Copyright Frank P. Hart, 2005–2008
7 %
8 clear all; clc; format long;
9 t_start = cputime;
10 % Transfer function coefficients...from A. Walker's one-tone
11 measurements...
12 polyreal = [...
13 -2.3228863e+001 0 ... % 3rd order + zero 2nd order...
14 +1.0045465e+002 0]; % 1st order + zero DC ...
15 polyimag = [...
16 +8.5100580e+001 0 ... % 3rd order + zero 2nd order...
17 +1.6632706e+000 0]; % 1st order + zero DC ...
18

F.3 Listings of Matlab Codes used in Chapter 3

This section contains listings of scripts and supporting utilities used to create the content in Chapter 3.

F.3.1 AOM Dynamic Range Demonstration Script

Listing F.18 is a script that demonstrates the nearly-unlimited dynamic range of the AOM Toolbox.
polycoeff = complex(polyreal,polyimag);
N_next = length(polycoeff)-1;  % Order of nonlinearity of block in blockeval

% Create the Vector Frequency Description Table
% Define two-tone frequency vector with 450 MHz carrier and
% input signal (to be power swept) 10 kHz away...
f = [450e6 450.01e6];  % frequency vector (row)
phid = [90 90];  % phase vector (degrees)
num_sides = 1;  % Number of sides to spectrum
nt = 2;  % Number of tones
ht_filename = [pwd,'/ar_450_dynrange_hash_tables.mat'];
[ht_nvfd, hash_tables, nvfd, nfreq_index, fvf, ffreq_index, ntf, fton, vfd_time] = ... 
aom_vfd(nt, N_next, f, num_sides, 'X', ht_filename)

% Create frequency vector from freq-sorted vfd table
ffreq = zeros(size(fvf,1),1);
for n = 1:length(fvf)
    ffreq(n) = cast(fvf(n,:), 'double') * transpose(f);
end;

% Create a dummy Amplitude vector, dummy spectral vector to get spectrum
% mapping table for repeated re-use...
dc = 0;  A = [1 1];
X = aom_makespv(A, dc, phid, ffreq_index);  % Make dummy spectral vector using A & phid
spmapstab = aom_spcecmap(ht_nvfd, hash_tables, nfreq_index, f, fvf, ntf, X, N_next, num_sides);

% Define the Spectral Vector constant and swept characteristics...
% Define Carrier power and tone power limits...
carr_power = -10.0;  % -10 dBm constant
carr_ampl = aom_dbmtov(carr_power);
tone_power = 0:-5:-300;
tone_ampl = aom_dbmtov(tone_power);

Y_thrlow = zeros(1,length(tone_power));
Y_thrhigh = zeros(1,length(tone_power));
Y_swpfu = zeros(1,length(tone_power));

for n = 1:length(tone_power)
    Aampl = [carr_ampl tone_ampl(n)];
    X = aom_makespv(Aampl, dc, phid, ffreq_index);
    Tx = aom_spsectrans(X,spmapstab,ffreq_index,'Tx');
    [Y, tbe] = aom_sblockeval(X,Tx,ffreq_index,polycoeff,'Y');
    Y_compthr = aom_svtocv(Y(:,3),ffreq_index);
    Y_amplthr = abs(Y_compthr);
    Y_compfun = aom_svtocv(Y(:,1),ffreq_index);
    Y_amplfun = abs(Y_compfun);
    Y_thrlow(n) = aom_vtodbm(Y_amplthr(3));  % index 3 is (2,-1) fvf
    Y_thrhigh(n) = aom_vtodbm(Y_amplthr(6));  % index 6 is (-1,2) fvf
% A R 10W1000C a m p l i f i e r 2–t on e time domain script.

F.3.2 FFT Dynamic Range Reference Script

Listing F.19 is an script that demonstrates the dynamic range of the FFT routines in Matlab® for comparison to the AOM Toolbox.

Listing F.19: Script demonstrating the dynamic range of the FFT.
clear variables; clc; format long;

This script computes the response to a two-tone input of a transfer
function modelling the AR amplifier in the time domain and
then takes an FFT. It is intended to illustrate the limitations
of the FFT compared to the pure FD computations of ar_450_dynrange.m

t_start = cputime;

% Transfer function coefficients...from A. Walker's one-tone
measurements...

polyreal = [...
-2.3228863e+001 0 ... % 3rd order + zero 2nd order...
+1.0045465e+002 0]; % 1st order + zero DC ...

polyimag = [...
+8.5100580e+001 0 ... % 3rd order + zero 2nd order...
+1.6632706e+000 0]; % 1st order + zero DC ...

polycoeff = complex(polyreal,polyimag);
polymag = abs(polycoeff);
polyphase = (180/pi)*angle(polycoeff);
polymag(1) = -polymag(1);

if polyphase(1) > 0
    polyphase(1) = polyphase(1) - pi;
else
    polyphase(1) = polyphase(1) + pi;
end;

carr_power = -10.0; % -10 dBm constant
carr_ampl = naom_dbmtov(carr_power);
tone_power = 0:-1:-300;
tone_ampl = naom_dbmtov(tone_power);

carr_freq = 450e6; % carrier at 450 MHz
tone_freq = 450.01e6; % tone at 450 MHz plus 10 kHz
carr_per = 1.0/carr_freq;
tone_per = 1.0/tone_freq;

N = 16*(1024)*(1024); % 2^21 points, power of 2
Nf = 500;
Fs = N*Nf; % Sampling freq appx = 10.48 GHz.; chosen so \Delta f =
fs/N is 10 kHz.
Ts = 1.0/Fs; % Sampling interval appx. 95 ps and change
f = [0:1:N-1]*Nf; % get a 1 kHz stepping
t = [0:1:N-1]*Ts; % Time series
carr_find = find(f == 450e6);
tone_find = find(f == 450.01e6);
out3m_find = find(f == 449.99e6);
out3p_find = find(f == 450.02e6);
lin_carr_tser = carr_ampl*cos(2*pi*carr_freq*t+polyphase(3)); %
carrier, linear phase
54 thr_carr_tser = carr_ampl*cos(2*pi*carr_freq*t+polyphase(1)); %
carrier, third order phase
55 lin_tone_tser_unsc = cos(2*pi*tone_freq*t+polyphase(3)); % tone,
linear phase
56 thr_tone_tser_unsc = cos(2*pi*tone_freq*t+polyphase(1)); % tone,
third order phase
57
58 P1t = zeros(1,length(tone_power)); % storage for FFT power at 450.01
 Mhz % Tone Power from Linear Input Signal
59 P1c = zeros(1,length(tone_power)); % storage for FFT power at 450.00
 Mhz % Tone Power from Linear Output Signal
60 P1t = zeros(1,length(tone_power)); % storage for FFT power at 450.01
 Mhz % Tone Power from Linear Output Signal
61 Poc = zeros(1,length(tone_power)); % storage for FFT power at 450.00
 Mhz % Carrier Power from Total Output Signal
62 Pot = zeros(1,length(tone_power)); % storage for FFT power at 449
 .99 Mhz % Lo--side IM3 from Total Output Signal
63 Poin3m = zeros(1,length(tone_power)); % storage for FFT power at 450
 .02 Mhz % Hi--side IM3 from Total Output Signal
64
65 h = waitbar (0,'Computing Power Sweeps');
66 for p = 1:length(tone_ampl)
67 lin_tone_tser = tone_ampl(p)*lin_tone_tser_unsc;
68 thr_tone_tser_unsc = tone_ampl(p)*thr_tone_tser_unsc;
69 lin_tser = polymag(3)* (lin_carr_tser + lin_tone_tser);
70 thr_tser = polymag(1)* (thr_carr_tser + thr_tone_tser).*3;
71 out_tser = lin_tser + thr_tser;
72 tone_fser = fft(lin_tone_tser)/N;
73 lin_fser = fft(lin_tser)/N;
74 out_fser = fft(out_tser)/N;
75
76 P1t(p) = 10* log10((abs(tone_fser(tone_find)).^2)/.1) + 3;
77 P1c(p) = 10* log10((abs(lin_fser(carr_find)).^2)/.1) + 3;
78 P1t(p) = 10* log10((abs(lin_fser(tone_find)).^2)/.1) + 3;
79 Poc(p) = 10* log10((abs(out_fser(carr_find)).^2)/.1) + 3;
80 Pot(p) = 10* log10((abs(out_fser(tone_find)).^2)/.1) + 3;
81 Poin3m(p) = 10* log10((abs(out_fser(out3m_find)).^2)/.1) + 3;
82 Poin3p(p) = 10* log10((abs(out_fser(out3p_find)).^2)/.1) + 3;
83
84 waitbar(p/length(tone_ampl));
85 pause(1);
86 end;
87 close(h);
88 plot(tone_power,P1c,'k--',tone_power,Poc,'k:',tone_power,Pit,'r-',
tone_power,Pit,'r--',tone_power,Pot,'r:',tone_power,Poin3m,'sb:',
tone_power,Poin3p,'dm--');
89 hl = legend(['Carrier Power at Linear Out','Carrier Power at Total Out'
 , 'Tone Power at Linear Input','Tone Power at Linear Out','Tone
Power at Total Out','Lo--Side IM3 at Total Out','Hi--Side IM3 at
Total Out']);
90 set(hl,'Fontname','Helvetica','Fontsize',12,'Location','North');
91 save ar_450_tdfft % .mat , save just power data later.
92 return;
F.3.3 Dynamic Range Plotting Script

Listing F.20 is a script that plots the results produced by the scripts in Listings F.18 and F.19.

Listing F.20: Script to plot dynamic range demonstration results.

```matlab
% Script to load .mat files of dynamic range data and plot

% North Carolina State University
% Copyright Frank P. Hart, 2005–2008

clear variables; clc;

load ar_450_dynrange.mat % Load AOM simulator data
load ar_450_tdiff.mat % Load time-domain simulator data

plot (aom_tone_power, aom_Y3lo, 'sb--', ..., 
aom_tone_power, aom_Y3hi, 'dm:', ..., 
tone_power, P1c, 'k--', ..., 
tone_power, Poc, 'k:', ..., 
tone_power, Pit, 'r--', ..., 
tone_power, P1t, 'r:', ..., 
tone_power, Pot, 'r:', ..., 
tone_power, Poin3m, 'b:', ..., 
tone_power, Poin3p, 'm--');
xl = legend ('AOM Lo–Side IM3', 'AOM Hi–Side IM3', 'Carrier Power at Linear Out', 'Carrier Power at Total Out', 'Tone Power at Linear Input', 'Tone Power at Linear Out', 'Tone Power at Total Out', 'Lo–Side IM3 at Total Out', 'Hi–Side IM3 at Total Out');
set (xl, 'Fontname', 'Helvetica', 'Fontsize', 10, 'Location', 'SouthEast');
figure (2);
plot (aom_tone_power, aom_Y3lo, '–db', ..., 
aom_tone_power, aom_Y3hi, '–sk', ..., 
tone_power, P1c, '–k', ..., 
tone_power, Poc, '–sk', ..., 
tone_power, Pit, '–k', ..., 
tone_power, P1t, '–k', ..., 
tone_power, Poin3m, ':b', ..., 
tone_power, Poin3p, '––k');
xlim([-303 max(tone_power)]);
```
F.4 Listings of Matlab Codes used in Chapter 4

This section contains listings of scripts and supporting utilities used to create the content in Chapter 4.

F.4.1 Logarithmic Amplifier Driver Script

Listing F.21 is a script that drives the function file to simulate the logarithmic amplifier.

```
Listing F.21: Script to drive the logarithmic amplifier function.

1 % AOM Log Amplifier Output Simulator
2 %
3 % North Carolina State University
4 % Copyright Frank P. Hart, 2005–2008
5 %
6 % This script plots the simulated output of a logarithmic amplifier using AOM
7 clear variables; format long; clc;
8
9 % Plot or not...
10 fignum = 0;
11 plot_spy = 0;
12 plot_5out = 1;
13 plot_5sp = 0;
14 plot_5revref = 0;
15 plot_5resp = 0;
16 plot_5revosp = 0;
17 plot_3out = 1;
18 plot_3revref = 1;
19
20 % Take a snapshot of the time
21 t_start = cputime;
```
% Error tolerances for sim loops...
in.itol = []; % Don't use relative error tolerance;
NUMTERMS governs loops
in.itol = 1e-1; % Use this tolerance to terminate expansion loops

% Input Spectral Vector Parameters and create Spectral Vector
in.dc = 10.0;
in.A = [ 2.0 2.0 ];
in.phid = [90.0 90.0 ];
in.f = 1e9*[0.9 1.0 ]; % Non-zero frequency input vector for log amp
in.nt = 2; % Number of tones (defined as length(f));
\%
in.N = 5; % Order of transfer function non-linearity
in.N = 2; % Order of transfer function non-linearity
in.N = 2; % Order of transfer function non-linearity
\%
in.num_sides = 2; % Number of sides to spectrum
in.spvec_kind = 'real'; % Real or complex
in.spvec_kind = 'comp';
in.savetable = 0;
in.savetabfn = 'n5vfd.tex ';
in.savetabfn = 'n3vfd.tex ';
in.savetabfn = 'n2vfd.tex ';

% Parameters governing the simulation loop
\%
in.NUM_TERMS = 2;
in.NUM_TERMS = 1;
in.MAX_TERMS = 100; % STOP series expansion at this number of terms if still out of tolerance
\%
in.STOP_TIME = 20e-9; % end sim at this time (set-up to match data output from Spice)
in.TIME_STEP = 1e-12; % time step for sim (set-up to match data output from Spice)
\%
% Call the logamp computational function
out = aom_logamp_fcn(in);
\%
% Recover output structure data...
V_t = out.V_t;
R = out.R;
L_s = out.L_s;
Tx = out.Tx;
sparsepct = out.sparsepct;
term_count = out.term_count;
t = out.t;
y_ac = out.y_ac;
y_exact = out.y_exact;
vfd_time = out.vfd_time;
specmap_time = out.specmap_time;
spectrans_time = out.spectrans_time;
iterloop_time = out.iterloop_time;
NUMTERMS = out.NUMTERMS;
ierror = out.ierror;
errorindex = out.errorindex;
aerror = out.aerror;
aerrorindex = out.aerrorindex;

fprintf('
');
fprintf([' Time to compute vfd = ',num2str(out.vfd_time),'
']);
fprintf([' Time to compute specmap = ',num2str(out.specmap_time),'
']);
fprintf([' Time to compute spectrans = ',num2str(out.spectrans_time),'
']);
fprintf([' Time to compute series exp = ',num2str(out.iterloop_time),'
']);
fprintf('
');
fprintf([' Number of sides to spectrum = ',num2str(in.numsides),'
']);
fprintf([' Order of sys non-linearity = ',num2str(in.N),'
']);
fprintf([' Pct. non-zero entries in Tx = ',num2str(out.sparsepct),'
']);
fprintf([' Number of terms in series = ',num2str(out.NUMTERMS),'
']);
fprintf([' Residual Error = ',num2str(out.ierror),'
']);
fprintf([' Exact Error = ',num2str(aerror),'
']);
fprintf([' Index of max exact error = ',num2str(aerrorindex),'
']);

% Computation of ln(1+xac/xdc) using AOM is now complete; result stored in y_ac.
% Now add to the constant terms of the transfer function of the logarithmic amplifier
% to get the final output: Vsd = -Vsd*[ln(dc) + ln[1/(R*Is)] + ln[1 + (xac/Xdc)]

vout_aom = zeros(1,size(t,2));
vout_aom = -Vsd*[log(in.dc) + log(1/(R*Is)) + y_ac];

% We eventually want to plot the aom results overlaid with the results obtained from
% the actual time domain equation

for k=1:in.nt
    vin = vin + in.A(k)*cos(2*pi*in.f(k)*t-pi/2);
end

vout_ref = -Vsd*log(vin/(R*Is));
t_end = cputime - t_start;
fprintf('
');
fprintf([' Time elapsed executing code = ',num2str(t_end),'
']);

% Sparsity plot of Tx
if plot_spy
```matlab
511  fignum = fignum + 1;
512  figure(fignum);
513  spy(Tx);
514  end;
515
516  imshow output voltages... vout_aom and vout_ref
517  if plot5out
518      fignum = fignum + 1;
519      figure(fignum);
520      plot(t/1e-9,vout_aom,t/1e-9,vout_ref);
521      hla = legend('V{\omega}_{o,aom} from AOM Toolbox','V{\omega}_{o,ref} Matlab Time-Domain Reference');
522      set(hla,'Location','North','Fontsize',12);
523      ylim([-0.8425 -0.8125]);
524      xlabel('Time (nsec)');
525      ylabel('Output V{\omega} (volts)');
526  end;
527
528  spice_dat = aom_get_spice_data('2k1Logamp.out');
529  t_sp = transpose(spice_dat(:,1));
530  vout_sp = transpose(spice_dat(:,2));
531  clear spice_dat;
532
533  imshow output voltages... vout_aom and vout_ref
534  if plot5sp
535      fignum = fignum + 1;
536      figure(fignum);
537      plot(t/1e-9,vout_aom,t/1e-9,vout_sp,t/1e-9,vout_sp);
538      hlb = legend('V{\omega}_{o,aom} from AOM Toolbox','V{\omega}_{o,ps} Microsim PSpice Transient');
539      set(hlb,'Location','North','Fontsize',10);
540      ylim([-0.8425 -0.8125]);
541      xlabel('Time (nsec)');
542      ylabel('Output V{\omega} (volts)');
543  end;
544
545  imshow the relative error of vout_aom vs. vout_ref
546  if plot5revrref
547      fignum = fignum + 1;
548      figure(fignum);
549      revrref = abs((vout_aom - vout_ref)./vout_ref)/1e-4;
550      plot(t/1e-9,revrref,-'k');  % plot as a percentage...
551      xlabel('Time (nsec)');
552      ylabel('Relative error \{e_{A}\} (x 10^{-4})');
553  end;
554
555  imshow the relative error of vout_aom vs. vout_sp
556  if plot5resp & (length(t) == length(t_sp))
557      fignum = fignum + 1;
558      figure(fignum);
559      reosp = abs((vout_aom - vout_sp)./vout_sp)/1e-4;
560      plot(t/1e-9,reosp,:'k');  % plot as a percentage...
561      xlabel('Time (nsec)');
562      ylabel('Relative error \{e_{S}\} (x 10^{-4})');
```

F.4.2 Logarithmic Amplifier Function

Listing F.22 is a function file that simulates logarithmic amplifier.

```
function out = aom_logamp_fcn(in)
% aom_logamp_fcn executes core AOM Toolbox functions to obtain
% logarithm function output
```
% Input: in - structure variable (see below)
% Output: out - structure variable (see below)
% North Carolina State University
% Copyright Frank P. Hart, 2005–2008

% Physical constants (source: Sears and Zemansky's Univ. Physics, 10th ed.)
k = 1.380658e-23; % Boltzman's Constant
T = 273.15 + 27; % Temp is degrees Kelvin, assume 27 deg C (that is ambient in Spice)
q = 1.60217733e-19; % Unit electronic charge of an electron (Coulombs)

% Other quantities particular to the compression amplifier
V_t = k*T/q; % "Thermal" voltage
R = 1000; % R is the input resistance, 1 kOhms here.
I_s = 1e-16; % I_s is the reverse saturation current of the BJT transistor, 100e-18 here.

% Take a snapshot of the time
t_start = cputime;

% Error tolerances for sim loops...
itol = in.itol;

% Input Spectral Vector Parameters and create Spectral Vector
dc = in.dc; % DC value
A = in.A; % non-DC Input Amplitudes
phid = in.phid; % Input phases (in degrees)
f = in.f; % Non-zero frequency input vector for log amp
nt = in.nt; % Number of tones (defined as length(f))
N = in.N; % Order of transfer function non-linearity
num_sides = in.num_sides; % Number of sides to spectrum
spvec_kind = in.spvec_kind; % Spectral vector kind

% Parameters governing the simulation loop
NUM_TERMS = in.NUM_TERMS; % NUM_TERMS=1 is an initial value only now for the series expansion
MAX_TERMS = in.MAX_TERMS; % STOP series expansion at this number of terms if still out of tolerance
STOP_TIME = in.STOP_TIME; % end sim at this time (set–up to match data output from Spice)
TIME_STEP = in.TIME_STEP; % time step for sim (set–up to match data output from Spice)

if ~isempty(itol) & (NUM_TERMS # 1)
    error('anm_logampfcn :: must have NUMTERMS == 1 when tolerance is specified');
end;
erreur = 1000; % initial error (over-estimate)

(1:STOP_TIME/STEP+1)=0:TIME_STEP:STOP_TIME; %Create time vector from 0 to stop time in increments of time_step

% Create the VFD Table
[ht_nvfd, hash_tables, nvfd, nfreq_index, fvd, ffreq_index, ntf, fton,
  vfd_time] = ...
aom_vfd(nt, N, f, num_sides, 'X');

X = aom_makespv(A, dc, phid, ffreq_index, spvec_kind); % Make arbitrary spectral vector using A & phid
X = X/dc; % For log amp, non-zero freqs are scaled by the DC value...
if num_sides == 1
  X_dci = 1; % DC element index in 1-sided vector
else
  X_dci = (length(X)-1)/2 + 1; % DC element index in 2-sided vector
end
if isreal(X)
  X(X_dci) = 0.0; % For log, DC is separated out of the AOM calculations
else
  X(X_dci) = complex(0.0,0.0);
end;

num_freq = size(fvfd,1);

% Create frequency vector from freq-sorted vfd table and locate dc index
freq = zeros(length(fvfd),1); % allocate space for freq
for k = 1:length(fvfd)
  freq(k) = cast(fvfd(k,:), 'double')*transpose(f);
end;

% stop here to save vfd and freq
%keyboard;
if isfield(in, 'savetable ') & (in.savetable == 1)
  frc = aom_vfd_to_LT(fvfd, ffreq_index, freq/1e9, in.savetabfn);
  %save(in.savetabfn, 'fvfd', 'freq', 'ffreq_index ');
end;

if num_sides == 1
  nposfreq = num_freq-1;
  freq_dci = 1;
else
  nposfreq = (num_freq-1)/2;
  freq_dci = nposfreq + 1;
end;

% Create spectrum mapping table
[spmaptab, specmap_time] = aom_sspecmap(ht_nvfd, hash_tables,
nfreq_index, f, fvd, ntoi, X, N, num_sides);
% Create spectrum transform matrix
[Tx, spectrans_time] = aom_spectrans(X, spmatlab, ffreq_index, 'Tx');
% Determine % sparsity of Tx
sparsect = 100.0*nnz(Tx)/prod(size(Tx));
% Create y_exact (ac portion of exact function) for reference use in
determining error...
arg_exact = ones(1,length(t)); % The '1' part of 1+x
for k=1:nt
  %arg_exact = arg_exact + (1/dc)*A(k)*sin(2*pi*f(k)*t); % Add in
  the 'x' part of 1+x
  arg_exact = arg_exact + (1/dc)*A(k)*cos(2*pi*f(k)*t-(pi/180)*phid(k)); % Add in the 'x' part of 1+x
end
y_exact = log(arg_exact); %
Now have log(1+x) for compare to ACM results...
y_last = 1000*ones(1,length(t)); % Init
to a ridiculous ref value to make sure loop starts...
y_ac = zeros(1,STOP_TIME/TIME_STEP+1); % y_ac
contains time domain values
waitstr=['Computing Infinite Series Output'];
waiterror=0;
h = waitbar(waiterror,waitstr);
term_sign = 1; % initialize alternating sign for ln(1+x) expansion
term_count = 1; % initialize term_count = final value determined by error tol
new_term = X;
tic;
if isempty(itol)
  loop_cont_cond = term_count < NUM_TERMS+1;
else
  loop_cont_cond = (ierror > itol) & (term_count < MAX_TERMS+1);
end;
%while ((ierror > itol)&(term_count < MAX_TERMS+1)) % Add terms to
% expansion until tolerance met or max terms hit...
while loop_cont_cond % Add terms to expansion until tolerance met or
max terms hit...
  Yac = term_sign*new_term/term_count; % Yac is an output spectral vector for current term_count...
  if isreal(X)
    y_ac = y_ac + Yac(X_dci)*ones(1,length(t)); % Add dc
component to running total of time domain value
  else
    y_ac = y_ac + real(Yac(X_dci))*ones(1,length(t)); % Add
dc component to running total of time domain value
  end;
for k=1:nposfreq % fvd now has DC, so -1 here % Now add time
domain component for each freq of the term
freq
ind = freq
dci + k;

% Right half spectrum — Do for both 1-sided and 2-sided...
if isreal(X)
    Yr = Yac(X

        dci + 2\*k-1); Yi = Yac(X

        dci + 2\*k);
else
    Yr = real(Yac(X

        dci + k)); Yi = imag(Yac(X

        dci + k));
end;
[Ymag, Yphi] = aom

doatan(Yr, Yi);
y
ac = y
ac + Ymag*cos(2*\pi*freq(freq
ind)*t - Yphi); % Add results to time domain signal. Vector op on each t.
if num
sides == 2 % Left half spectrum — Do only for 2-sided...
    freq
ind = freq
dci - k;
    if isreal(X)
        Yr = Yac(X

        dci - 2\*k+1); Yi = Yac(X

        dci - 2\*k);
    else
        Yr = real(Yac(X

        dci - k)); Yi = imag(Yac(X

        dci - k));
    end;
    [Ymag, Yphi] = aom

doatan(Yr, Yi);
y
ac = y
ac + Ymag*cos(2*\pi*freq(freq
ind)*t - Yphi); %
end
end %end for loop to add freq. content to y
ac
[ierror, ierrorindex] = max(abs(y
ac - y
last));
y
last = y
ac;
if (ierror > itol)
    waiterror = 1/abs((log10(itol/ierror)));
else
    waiterror = 1;
end
waitbar(waiterror);
term
sign = -term
sign; % prepare ingredients for next term in the infinite series expansion...
new
term = Tx*new
term; % Perform Tx matrix multiply to raise to the next power
term
count = term
count + 1;
if isempty(itol)
    loop
cont
cond = term
count < NUM
TERMS+1;
else
    loop
cont
cond = (ierror > itol) & (term
count < MAX
TERMS+1);
end;
end %end while loop — either tolerance met or MAX
TERMS reached...
iter
oop
ime = toc;
close(h); % Get rid of waitbar
% find the exact error of the expansion vs. the built-in transcendental function...
[aerror, aerrorindex] = max(abs(y
ac - y
exact));
% Create output structure...
out.V
_t = V
_t;
out.R = R;
out.L
s = L
s;
out.Tx = Tx;
out.sparsepct = sparsepct;
F.4.3 Spice Data Import Function

Listing F.23 is a function file that imports Spice-compatible transient simulation output data into Matlab®. The imported data is compared to output from the AOM Toolbox.

Listing F.23: Function file for reading Spice data.

```matlab
function spice_dat = aom_get_spice_data(dat_file_name);
% aom_get_spice_data reads Spice output into Matlab data structures
% Get Spice Transient Data
% This script file reads in the automatically generated Spice output data file, locates the table of transient data, and reads it into a two-dimensional vector spice_dat, where each row is of the form [time-step f(time-step)] and f is some function of time that has been output as the result of a transient simulation.
% The purpose of this function is to allow the easy comparison of Spice results and results from AOM to be compared on the same MATLAB plot.
% Note that this function expects output in a form that does not include .PLOT data to the right. It is compatible only with Spice decks including .PRINT statements without .PLOT statements.
% North Carolina State University
% Copyright Frank P. Hart 2001-2008
```
% Attempt to open for read only
[fid, fidmessage] = fopen('dat_file_name', 'r');
if fid == -1 % if pass this, got a good file...
    fprintf(1, 'Cannot open
    
    Message from fopen command: ');
    fprintf(1, fidmessage);
    fprintf(1, 'Returning a -1 to caller...
    return;
    end;

% found_tran=0;
while -(found_tran)
    input_line = fgetl(fid);
    if length(input_line) > 28
        ast = input_line(2:5);
        tr = input_line(11:19);
        an = input_line(21:28);
        if isequal(ast, '****') & isequal(tr, 'TRANSIENT') & isequal(an, 'ANALYSIS')
            found_tran=1; % found beginning of .tran data, break out of
        end;
    end;
    if found_tran==0
        % this shouldn't happen...
        spice_dat = -1;
        return;
    end;
end

% advance file pointer to first line of data
for n=1:9 input_line=fgetl(fid);
end;

end_vect=0;
spice_dat ='';
while -(end_vect)
    time = str2num(input_line(4:12));
    vout = str2num(input_line(15:24));
    if length(spice_dat)==0
        spice_dat = [time vout];
    else
        spice_dat = [spice_dat; time vout];
    end;
    input_line=fgetl(fid);
    if length(input_line)==0
        end_vect=1;
    end;
end;
F.4.4  VFD to LaTeX Export Function

Listing F.24 is a function file that exports VFD tables and related information in a form suitable for formatting by the \texttt{\LaTeX Longtable} environment.

Listing F.24: Function file for exporting VFD information.

```matlab
function rc = aom_vfd_to_LT(vfd, ffreq_index, ffreq, out_file_name)
% aom_vfd_to_LT - function to create \LaTeX Longtable-compatible table from VFD info
% aom_vfd_to_LT creates \LaTeX-compatible source code suitable for use with
% the Longtable package. It is intended to automate documentation of
% AOM-generated VFD tables in documentation.
% Inputs:
% vfd - nVfd or fVfd table
% ffreq_index - frequency-sorted frequency index
% ffreq - numeric frequencies
% out_file_name - name of output file for the \LaTeX source
% North Carolina State University
% Copyright Frank P. Hart, 2005–2008
% rc = 0;
add_struts = 1;  % include '\T\B' strut
add_hline = 1;  % include 'hline\%' at the end of the line
if add_struts == 1
    struts = [' \T\B'];
else
    struts = []; 
end;
ffl = length(ffreq_index);
for i = 1:ffl
    ffil(i) = length(num2str(ffreq_index(i)));
end;
max_ffil = max(ffil);
ffis = cell(ffl,1);
for i = 1:ffl
    ffil = []; 
```
for j = 1:max_ffil-ffil(i)
    ffls = [ffls , ' '];
end;
ffis{i} = [',num2str(ffreq(i)),',ffils ,struts , ' & ' ]; % freq.
index
ffis{i} = [ffis{i}, ',num2str(ffreq(i)),',ffils , ' & ' ];% VFD
end;
%keyboard;
%return;

vfdnorm = sum(abs(vfd), 2);
for i = 1:ffl
    vfdnl(i) = length(num2str(vfdnorm(i)));
end;
max_vfdnl = max(vfdnl);
Vfds = cell(1, ff,1);
for i = 1:ffl
    vfdnls = [];
    for j = 1:max_vfdnl- vfdnl(i)
        vfdnls = [vfdnls , ' '];
    end;
    vfdnls{i} = [',num2str(vfdnorm(i)),', vfdnls , ' & ' ];
end;

vfdw = size(vfd, 2);
for i = 1:ffl
    vfdl(i) = 0;
    for j = 1:vfdw
        vfdl(i) = vfdl(i) + length(num2str(vfd(i, j)));
    end;
end;
max_vfdl = max(vfdl);
Vfds = cell(1, ff,1);
for i = 1:ffl
    vfdls = [];
    for j = 1:max_vfdl- vfdl(i)
        vfdls = [vfdls , ' '];
    end;
    vfdls{i} = [Vfds{i}, ',num2str(vfd(i, j)), ' & ' ];% VFD tuples
end;
Vfds{i} = [vfdls{i}, vfdls];
end;
%keyboard;

for i = 1:ffl
    ffrl1(i) = length(num2str(ffreq(i)));
end;
max_ffrl = max(ffrl1);
ffreqs = cell(1, ff,1);
for i = 1:ffl
    ffrls = [];
end;
F.5 Listings of Matlab Codes used in Chapter 5

This section contains listings of scripts and supporting utilities used to create the content in Chapter 5.

F.5.1 Uncorrelated Phase Multitone Driver Script

Listing F.25 is a script that drives the function file to simulate the nonlinear response of an amplifier to multitone waves with uncorrelated phase.

```
function driveUncorrelatedPhaseMultitone

% Uncorrelated phase multitone driver AOM simulation script...

% This script drives the unc_mtoneFcn.m file, which does the AOM computations
```

Listing F.25: Script to drive the uncorrelated phase multitone function.
% involved in modeling the nonlinear response to multitone inputs with
% uncorrelated phase. The driver configures the structure input
variable
% called 'options' which directs the function file on what to compute
and
% the form of spectral vectors to use.

% North Carolina State University
% Copyright Frank P. Hart, 2005–2008
%

clear variables; clc;

%num_ph_runs = 30; % number of random phase runs...
num_ph_runs = 1;
A_phid_src = zeros(num_ph_runs+1,15); % leaves the zero-phase at the
top...
rand('state', 0);
for i = 2:num_ph_runs+1 % deal with Matlab indices
    A_phid_src(i,:) = 360.0*(rand(1,15)-0.5); % Rand is uniformly
distributed between 0 and 1; subtract half...
end;
% Other input options
options.n tones_in = '15';
options.amplifier_in = 'MC';
options.desired_NL_order_in = '05';
options.comb_attn_in = '03';
options.freq_in = [];
options.sim_num_in = [];
options.sim_kind_in = 'cun';
fprintf('
');
for i = 1:num_ph_runs+1
    j = i-1;
    options.phase_num_in = j;
    options.phid_in = A_phid_src(i,:);
    fprintf('Beginning random phase simulation number \n',j);
    rc = uncintone_fcn(options);
    fprintf('Ending random phase simulation number \n',j);
end;
return;

F.5.2 Uncorrelated Phase Multitone Function

Listing F.26 is a function file that simulates the response of a nonlinear amplifier to multitones
with uncorrelated phase.

Listing F.26: Function file for uncorrelated phase multitones.
function rc = unc_mtone_fcn( options );
% unc_mtone_fcn computes the nonlinear response of uncorrelated phase multitones
% The amplifier modeled is a Mini-Circuits ZHL-5W.
% Input: options — structure variable (see below)
% Output: rc — return code (int)
% North Carolina State University
% Copyright Frank P. Hart, 2005–2008

% Initialize return code
rc = 0;

% Start time
start = cputime;

% Plot options
plot_input_ampl_volts = 1; % Plot input ampl using stem()
plot_input_ampl_dbm = 1;
plot_output_ampl_lin = 0;
plot_output_ampl_all = 0; % Plot all sv output ampls using stem();
plot_output_ampl_thr = 0; % Plot 3rd ord out ampl only using stem();
plot_output_ampl_fif = 0;
plot_pmag_output = 0;
plot_output_random = 0; % Plot out ampls in root of sum of sq mag of ph
plot_output_coherent_only = 1; % Plot output ampl using mag of coh phasor sum
plot_output_measured_only = 1; % Plot meas output on same plot as coh
plot_output_meas_coh = 0;
plot_coherent_magnitude_error = 1;
plot_coherent_phase_error = 0;

% Unpack the options from the structure input variable...
num_tones_in = options.num_tones_in;
amplifier_in = options.amplifier_in;
desired_NL_order_in = options.desired_NL_order_in;
comb_attn_in = options.comb_attn_in;
freq_in = options.freq_in; % Needed for ctb version.
sim_num_in = options.sim_num_in; % Ditto above.
phid_in = options.phid_in;
phase_num_in = options.phase_num_in;
sim_kind_in = options.sim_kind_in;

[num_tones, amplifier, desired_NL_order, comb_attn, freq, phid, phase_num, sim_num, sim_kind] = ...
local_check_set_parameters(num_tones_in, amplifier_in,
desired_NL_order_in, comb_attn_in,

freq_in, phid_in, phase_num_in, sim_num_in, sim_kind_in);

nt = str2num(num_tones); % Need numeric form for vfd creation...

% Character switches for storage files not set by
local_check_set_parameters...
tone_spacing = '1e3'; % Choose tone spacing (integer kHz); 1 MHz space.
center_freq = '450'; % Choose center frequency (integer MHz)

% Directory locations
entry_dir = pwd; % Store the directory we entered from and return there
[home_dir, slash] = local_get_home_slash; % Figure out 'home' directory and 'slash' for the computer we're on...
work_dir = [home_dir, slash, 'work']; % Location of the MATLAB work directory...
amp_dir = [home_dir, slash, amplifier, '450Data']; % Directory containing data for AR and MC amplifiers...
src_dir = [home_dir, slash, 'AOMuncmtone', slash, 'contraps_wbox_30min_ampl_char']; % Measured source data...
PNA_dir = [home_dir, slash, 'AOMcontrap_VNA5230_data']; % Signal path S -parameter measurement data...
curr_dir = [pwd, slash];
AOM_contrap_dir = curr_dir;
matfile_str = [sim_kind_ch, '_mtone_', num_tones, 'T_', tone_spacing, 'K_',
center_freq, 'M', desired_NL_order, 'O_'];

if ~isempty(sim_num_ch)
simnum_str = ['S_', sim_num_ch, '']
end;

% Call local function to get signal corrected to amplifier input....
% Reference plane is contraption output...
[A_ampl_src] = local_get_source_amplitude(num_tones, src_dir, slash);
if sim_kind_ch == 'ctb'
A_ampl_src(8) = 0.0;
end;

% Initialize rand state...
rand('state', 0);

% Call local function to get random phase assignment....
if ~isempty(phid)
A_phid_src = phid;
else %
[A_phid_src] = local_get_source_phase(A_ampl_src);
end;

% Nominal frequency locations if not passed in...
if isempty(nt)
switch nt
case 15
freq = [443:1:457]*1e6;
case 3
end;
freq = [449:1:451]*1e6;
otherwise
end;
end;
%return;

% Create the scatter transfer (T) and S-parameter matrix for each source tone....
% Reference plane at contraption elbow output, so overall T and S matrix includes
% effects of 10 dB attn, 1.5 foot cable, combiner, VLF-490 filter, 6 dB (AR) or
% 10 dB (MC) attenuator, and 2.0 foot cable. Matrices are stored in
% S(:, :, toneind),
% where toneind ranges from 1 to numtones, and the first :, :
% corresponds to the parameter
% subscripts.
sxfr_str = [currdir, matfile_str, combattn, 'db_input_sxfr.mat'];
if exist(sxfr_str)
  load(sxfr_str);
else
  [S_xfer, T_xfer] = local_get_input_scatter_transfer(combattn, freq, 
PNA_dir, slash);
save(sxfr_str, 'S_xfer', 'T_xfer');
end;

% Scatter transfer source signal from input reference plane to amplifier input....
% The output amplitudes and phases will appear at the input to the amplifier...
[A_ampl_xfer, A_phid_xfer, A_magdbm_xfer] = local_get_ampinput( 
A_ampl_src, A_phid_src, S_xfer);
if sim_kind == 'iuc'
  A_ampl_mean = mean(A_ampl_xfer); % find the mean amplitude
  A_ampl_xfer = A_ampl_mean*ones(1, length(A_ampl_xfer)); % set all tones to the mean..
end;

% Some plotting info
freqscale = 1.0e6; % scale for neat plots
fc = 450e6;
df = 1e6;
bbw = 60e6;
fmin = (fc - bbw/2)/freqscale;
fmax = (fc + bbw/2)/freqscale;
ymin = 0.0;
ymindbm = -100.0;
fscale = freq/freqscale;
freq_scale = freq/freqscale;

% Plot the magnitude of the input amplitude (volts)
if plot_input_ampl_vols
  plot_spec = 'dk';
  [fignum, h_ampl_vols] = local_plot_input_ampl_vols(freq_scale,
A_magdbm_xfer, plot_spec, freqscale, fc, df, fmin, fmax, fignum);
end;

if plot_input_ampl_dbm
    plot_spec = 'k-';
    negdbmlim = -100.0;
    [fignum, h_ampl_dbm] = local_plot_input_ampl_dbm(freq_scale,
        A_magdbm_xfer, plot_spec, negdbmlim, freqscale, fc, df, fmin, fmax,
        fignum);
end;

% Plot the magnitude of the input amplitude (dBm)
if plot_magnitude
    plot_spec = 'k-';
    negdbmlim = -100.0;
    [fignum, h_magnitude] = local_plot_magnitude(freq_scale,
        A_magnitude, plot_spec, negdbmlim, freqscale, fc, df, fmin, fmax,
        fignum);
end;

% Call local function (below) to get amplifier coefficients...
[polyreal, polyimag] = local_get_amp_coeffs(amplifier, amp_dir,
    desired_NL_order);
polycoeff = complex(polyreal, polyimag);
if sim_kind.ch == 'cun' % Make 2-s complex...
    num_sides = 2;
    spvec_kind = 'comp';
else
    num_sides = 1;
    spvec_kind = 'real';
end;
N_next = length(polycoeff)-1; % Order of nonlinearity of block in blockeval

% Move to directory containing AOM files...
cd(AOM_contrap.dir);

% Create the Vector Frequency Description Table
if ~isempty(sim_num.ch)
    vfd_str = [curr_dir, matfile_str, simnum_str, 'vfd.mat'];
else
    vfd_str = [curr_dir, matfile_str, 'vfd.mat'];
end;
if exist(vfd_str)
    load(vfd_str);
else
    if ~isempty(sim_num.ch)
        ht_filename = [curr_dir, matfile_str, simnum_str, 'hash_tables.mat'];
    else
        ht_filename = [curr_dir, matfile_str, 'hash_tables.mat'];
    end;
    [ht_nvfd, hash_tables, nvfd, nfreq_index, fvd, ffreq_index, ntof,
        fton, vfd_time] = ...
        aom_vfd(nt, N_next, freq, num_sides, 'X', ht_filename);
    save(vfd_str, 'ht_nvfd', 'nvfd', 'nfirst', 'nfreq_index', 'fvd',
        'ffreq_index', 'ntof', 'fton', 'vfd_time', 'nvfd_freq', 'nvfd_sum',
        'nvfd_hash', 'nvfd_dups', 'end_dups', 'hash_hits', 'hash_tables');
end;

% Create the Spectral Vector
dc = 0.0;
[X, makespv_time] = aommakespv(Aamp_xfer, dc, Aphi_xfer, ffreq_index, spvec_kind); % Make arbitrary spectral vector using A & phid

% Create frequency vector from freq--sorted vfd table
ffreq = zeros(size(fvfd,1),1);
for n = 1:length(fvfd)
    ffreq(n) = cast(fvfd(n,:), 'double')*transpose(freq);
end;

% Create spectrum mapping table
if ~isempty(simnum_ch)
    specmap_str = [curr_dir, matfile_str, simnum_str,'specmap.mat'];
else
    specmap_str = [curr_dir, matfile_str, 'specmap.mat'];
end
if exist(specmap_str)
    load(specmap_str);
else
    [spmaptab, t_ex_specmap] = aomsspecmap(ht_nvfd, hash_tables, nfreq_index, freq, fvfd, ntf, X, Nnext, num_sides);
    save(specmap_str, 'spmaptab', 't_ex_specmap');
end;
if exist('t_ex_specmap')
    specmap_time = t_ex_specmap;
end;

% Create sparse spectrum transform matrix
if ~isempty(simnum_ch)
    spectransstr = [curr_dir, matfile_str, simnum_str, comb_attn, 'db_spectrans.mat'];
else
    spectransstr = [curr_dir, matfile_str, comb_attn, 'db', phase_num_ch, 'ph_spectrans.mat'];
end
if exist(spectrans_str)
    load(spectrans_str);
else
    [Tx, spectrans_time] = aomsspectrans(X, spmaptab, ffreq_index, 'Tx');
    save(spectrans_str, 'Tx', 'spectrans_time');
end;

% Determine % sparsity of Tx
sparsect = 100.0*nnz(Tx)/prod(size(Tx));

% Evaluate through sparse replacement for block
[Ydut, t_be] = aomsblockeval(X, Tx, ffreq_index, polycoeff, 'Ydut');

% Need amplifier to choose correct pwr cable, PNA_dir to get s2p file, ffreq to
% get frequencies to recover for S21, ffreq_index to determine if 1s or 2s, and Ydut...
% Must convert Ydut to complex vectors, do S21 multiplication, convert back to return...
\%Ysim = local_get_pwr_cbl_shift( amplifier, PNA_dir, slash, ffreq, 
ffreq_index, Ydut);
Ysim = local_get_output_scatter_transfer( sim_kind_ch, amplifier, 
PNA_dir, slash, ffreq, ffreq_index, Ydut);
ffreq_fscale = ffreq/ffreqscale;

\% Statistics from run...
fprintf(\'\n\');
fprintf(\'\'Time to compute vfd
 = \',num2str(vfd_time),\'\n\']);
fprintf(\'\'Time to compute spec vector
 = \',num2str(makepv_time),\'\n\']);
fprintf(\'\'Time to compute specmap
 = \',num2str(specmap_time),\'\n\']);
fprintf(\'\'Time to compute spectrans
 = \',num2str(spectrans_time),\'\n\'])->
fprintf(\'\n\');
fprintf(\'\'\Order of non-linearity
 = \',num2str(N_next),\'\n\']);
fprintf(\'\'\Pct. non-zero entries in Tx
 = \',num2str(sparsepct),\'\n\']);
fprintf(\'\n\');
fprintf(\'\'Time to compute blockeval
 = \',num2str(t_be),\'\n\']);

if isreal(Ysim)
Y_comp_volt_lin = aom_svtocv(Ysim(:,1),ffreq_index);
Y_ampl_volt_lin = abs(Y_comp_volt_lin)+eps;
Y_ampl_dbm_lin = aom_vtodbm(Y_ampl_volt_lin);
Y_comp_volt_thr = aom_svtocv(Ysim(:,3),ffreq_index);
Y_ampl_volt_thr = abs(Y_comp_volt_thr)+eps;
Y_ampl_dbm_thr = aom_vtodbm(Y_ampl_volt_thr);
Y_comp_volt_fif = aom_svtocv(Ysim(:,5),ffreq_index);
Y_ampl_volt_fif = abs(Y_comp_volt_fif)+eps;
Y_ampl_dbm_fif = aom_vtodbm(Y_ampl_volt_fif);
Y_ampl_volt = [Y_ampl_volt_lin, Y_ampl_volt_thr, Y_ampl_volt_fif];
Y_ampl_dbm = [Y_ampl_dbm_lin, Y_ampl_dbm_thr, Y_ampl_dbm_fif];
else
Y_comp_volt_lin = Ysim(:,1);
Y_ampl_volt_lin = abs(Y_comp_volt_lin)+eps;
Y_ampl_dbm_lin = aom_vtodbm(Y_ampl_volt_lin);
Y_comp_volt_thr = Ysim(:,3);
Y_ampl_volt_thr = abs(Y_comp_volt_thr)+eps;
Y_ampl_dbm_thr = aom_vtodbm(Y_ampl_volt_thr);
Y_comp_volt_fif = Ysim(:,5);
Y_ampl_volt_fif = abs(Y_comp_volt_fif)+eps;
Y_ampl_dbm_fif = aom_vtodbm(Y_ampl_volt_fif);
Y_ampl_volt = [Y_ampl_volt_lin, Y_ampl_volt_thr, Y_ampl_volt_fif];
Y_ampl_dbm = [Y_ampl_dbm_lin, Y_ampl_dbm_thr, Y_ampl_dbm_fif];
end;

% Power Plotting visualization – composite segment for each frequency
pthresh = -120;
refres = 50;
[pfreq, pfreqi, corfreqi, uncfreqi] = aom_getpfreqcu(ffreq, fvd);
[pmagcoh, pmagran, vmagcoh, vmagran, pcohunc, pcochcor, vcochcor] =
aom_getpowerc(Y_sim, pfreq, pfreqi, corfreqi, uncfreqi, ffreq_index, refres);

% Plot the outputs of getpmag
if plot_pmag_output
  if ~isempty(sim_num_ch)
    pmag_str = [curr_dir, matfile_str, simnum_str, comb_attn, 'db',
                amplifier, '_pmagran.mat'];
  else
    pmag_str = [curr_dir, matfile_str, comb_attn, 'db', amplifier, '_pmagran.mat'];
  end
  pmagran_dbm = 10*log10((pmagran+eps)/(.001));
  fignum = fignum + 1;
  figure(fignum);
  plot(pfreq/freqscale, pmagran_dbm+6, '-b'); % scale amplitude by 3 dB when complex 2-s, so power by 6 dB
  xlim([fmin fmax]);
  ylim([-120 0]);
  hold on;
  pmag_str = ['contrap_15T_1e3K_150M_05O_03db_MC_pmagran.mat'];
  load(pmag_str);
  pmagran_dbm = 10*log10((pmagran+eps)/(.001));
  if ~isempty(sim_num_ch)
    pmagph_str = [curr_dir, matfile_str, simnum_str, comb_attn, 'db',
                  phase_num_ch, 'phi', amplifier, '_pmagranfreq.mat'];
    allvars_str = [curr_dir, 'allvars_', matfile_str, simnum_str, comb_attn,
                   'db', amplifier, '_pmagranfreq.mat'];
  else
    pmagph_str = [curr_dir, matfile_str, comb_attn, 'db', phase_num_ch, 'phi',
                  amplifier, '_pmagranfreq.mat'];
    allvars_str = [curr_dir, 'allvars_', matfile_str, comb_attn, 'db',
                   phase_num_ch, 'phi', amplifier, '_pmagranph.mat'];
  end
  pmagran_dbm = 10*log10((pmagran+eps)/(.001));
  pcochcor_dbm = 10*log10((pcochcor+eps)/(.001));
  pcochunc_dbm = 10*log10((pcochunc+eps)/(.001));
  save(pmagph_str, 'pmagcoh', 'pmagran', 'vmagcoh', 'vmagran', 'Y_sim', 'pfreq',
       'pfreqi', 'ffreq_index', 'refres', ...
       'pthresh', 'pmagran_dbm', 'freqscale', 'freq', 'fvd', 'corfreqi',
       'uncfreqi', 'pcochcor', 'pcochunc', ...
       'pcochcor_dbm', 'pcochunc_dbm', 'vcochcor'); % Change this to save everything. Comment here and simplify call.
  save(allvars_str);
  cd(entry_dir);
  return;
end

%-------------------------------- End of main part of function...
function [A_ampl_src] = local_get_source_amplitude(num_tones, src_dir, slash)

% Number of tones input determines construction of A_ampl_src
num_tones = str2num(num_tones);
num_contr = num_tones/3;
A_ampl_dbm = zeros(1,num_tones);
A_ampl_src = zeros(1,num_tones);
if num_tones == 15
    contrap = [3 6 4 5 7]; % order of contraption tones, e.g c3 -> 43, 44, 45...
elseif num_tones == 9
    contrap = [6 4 5];
e elseif num_tones == 3
    contrap = 4;
end;
for k = 1:num_contr
    src_file = ['c',num2str(contrap(k)),',
    -box 30min ampl char peak figdata.mat'];
    src_loc = [src_dir,slash,src_file];
clear p_data; % clear the cell array variable containing the
    characterized amplitudes...
    load(src_loc);
    for i = 1:3 % do for each oscillator...
        clear mean_amp;
        mean_amp = p_data{i,6}; % running amplitude mean is the 6th
        array for each oscillator (i)
        A_ampl_dbm(3*(k-1)+i) = mean_amp(end); % use last average
            recorded...
    end;
end;
clear p_data;
clear mean_amp;
A_ampl_src = aomdlbmtov(A_ampl_dbm);
return;

function [A_phid_src] = local_get_source_phase(A_ampl_src)
A_phid_src = 360.0*(rand(size(A_ampl_src))-0.5); % Rand is
    uniformly distributed between 0 and 1; subtract half...
return;

function Begin Local Function to get scatter transfer
matrices...
function \([S\_xfer, T\_xfer] = \) local\_get\_input\_scatter\_transfer\((comb\_attn, freq, \) PNA\_dir, slash\()\)

% Recover number of tones...
num\_tones = length\(freq\);
num\_contr = num\_tones/3;
T\_xfer = zeros\((2,2,\) num\_tones\);
S\_xfer = zeros\((2,2,\) num\_tones\);
if num\_tones == 15
    contrap = [3 6 4 5 7]; % order of contraption tones, e.g c3 -> 443, 444, 445...
    combinp = [5 2 4 3 1]; % combiner input used by respective contraption...
elseif num\_tones == 9
    contrap = [6 4 5];
elseif num\_tones == 3
    contrap = 4;
end;

for k = 1: num\_contr % do for each contraption
    for i = 1:3 % do for each contraption’s oscillators...
        toneind = 3\(k-1\)+i;
%
        % passage through 10 dB attenuator on contraption output...
        cx\_attn\_dev = ['c','num2str\(\) (contrap\(k\))','\_attn\_10\_db']; % file
        with 10 dB attenuator data
        cx\_attn\_loc = [PNA\_dir, slash, cx\_attn\_dev, '.mat'];
        load\(\) (cx\_attn\_loc, '-mat');
        devfreq = eval\(\) ([cx\_attn\_dev, '.Freq'; ']);
        freqind = find\(\) (devfreq == freq\(toneind\));
        cx\_attn\_T\((1,1)\) = eval\(\) ([cx\_attn\_dev, '.T11\(freq\)\ind; ']);
        cx\_attn\_T\((1,2)\) = eval\(\) ([cx\_attn\_dev, '.T12\(freq\)\ind; ']);
        cx\_attn\_T\((2,1)\) = eval\(\) ([cx\_attn\_dev, '.T21\(freq\)\ind; ']);
        cx\_attn\_T\((2,2)\) = eval\(\) ([cx\_attn\_dev, '.T22\(freq\)\ind; ']);
        eval\(\) ('clear ', cx\_attn\_dev);
%
        % passage through 1.5 foot cable from contraption attenuator to combiner...
        cx\_cbl15\_dev = ['c','num2str\(\) (contrap\(k\))','\_cbl15']; % file
        with 1.5 foot cable data
        cx\_cbl15\_loc = [PNA\_dir, slash, cx\_cbl15\_dev, '.mat'];
        load\(\) (cx\_cbl15\_loc, '-mat');
        devfreq = eval\(\) ([cx\_cbl15\_dev, '.Freq'; ']);
        freqind = find\(\) (devfreq == freq\(toneind\));
        cx\_cbl15\_T\((1,1)\) = eval\(\) ([cx\_cbl15\_dev, '.T11\(freq\)\ind; ']);
        cx\_cbl15\_T\((1,2)\) = eval\(\) ([cx\_cbl15\_dev, '.T12\(freq\)\ind; ']);
        cx\_cbl15\_T\((2,1)\) = eval\(\) ([cx\_cbl15\_dev, '.T21\(freq\)\ind; ']);
        cx\_cbl15\_T\((2,2)\) = eval\(\) ([cx\_cbl15\_dev, '.T22\(freq\)\ind; ']);
        eval\(\) ('clear ', cx\_cbl15\_dev);
%
        % passage through 5:1 combiner...
        ix\_comb\_dev = ['i','num2str\(\) (combinp\(k\))','\_s\_comb\_zbsc5']; % file for combiner path data
        ix\_comb\_loc = [PNA\_dir, slash, ix\_comb\_dev, '.mat'];
        load\(\) (ix\_comb\_loc, '-mat');
devfreq = eval([ix_comb_dev,'.Freq;']);
defreqind = find(devfreq == freq(toneind));
ix_comb_T(1,1) = eval([ix_comb_dev,'.T11(freqind);']);
ix_comb_T(1,2) = eval([ix_comb_dev,'.T12(freqind);']);
ix_comb_T(2,1) = eval([ix_comb_dev,'.T21(freqind);']);
ix_comb_T(2,2) = eval([ix_comb_dev,'.T22(freqind);']);
eval(['clear',ix_comb_dev]);

% passage through VLF 490 filter...
cs_vlf_dev = ['cs_attn_vlf_490_42']; % file with filter data
cx_vlf_loc = [PNA_dir,slash,cs_vlf_dev,'.mat'];
load(cx_vlf_loc, 'A1');
devfreq = eval(['cs_vlf_dev','Freq;']);
defreqind = find(devfreq == freq(toneind));
cs_vlf_T(1,1) = eval(['cs_vlf_dev','T11(freqind);']);
cs_vlf_T(1,2) = eval(['cs_vlf_dev','T12(freqind);']);
cs_vlf_T(2,1) = eval(['cs_vlf_dev','T21(freqind);']);
cs_vlf_T(2,2) = eval(['cs_vlf_dev','T22(freqind);']);
eval(['clear','cs_vlf_dev']);

% passage through 6/10 dB attenuator....
switch comb_attn
  case {'03'}
    cs_attn_dev = ['cs_attn_3db']; % file with 3 dB attenuator data
  case {'06'}
    cs_attn_dev = ['cs_attn_6db']; % file with 6 dB attenuator data
  case {'10'}
    cs_attn_dev = ['cs_attn_10db']; % file with 10 dB attenuator data
  case {'20'}
    cs_attn_dev = ['cs_attn_20db']; % file with 20 dB attenuator data
  case {'30'}
    cs_attn_dev = ['cs_attn_30db']; % file with 30 dB attenuator data
  case {'40'}
    cs_attn_dev = ['cs_attn_40db']; % file with 40 dB attenuator data
  otherwise
    cs_attn_dev = ['cs_attn_6db']; % file with 6 dB attenuator data
end;
cs_attn_loc = [PNA_dir,slash,cs_attn_dev,'.mat'];
load(cs_attn_loc, 'A1');
devfreq = eval(['cs_attn_dev','Freq;']);
defreqind = find(devfreq == freq(toneind));
cs_attn_T(1,1) = eval(['cs_attn_dev','T11(freqind);']);
cs_attn_T(1,2) = eval(['cs_attn_dev','T12(freqind);']);
cs_attn_T(2,1) = eval(['cs_attn_dev','T21(freqind);']);
cs_attn_T(2,2) = eval(['cs_attn_dev','T22(freqind);']);
eval(['clear','cs_attn_dev']);
533
534% passage through 2.0 foot cable...
535cs_cbl20_dev = ['cs_cbl20'] ; % file with cable data
536cs_cbl20_loc = [PNA_dir, slash, cs_cbl20_dev, '.mat' ];
537load(cs_cbl20_loc, 'mat');
538devfreq = eval(['cs_cbl20_dev', '.Freq' ]); % file with cable data
539freqind = find(devfreq == freq(toneind));
540cs_cbl20_T(1,1) = eval(['cs_cbl20_dev', '.T11(freqind)' ]); % find position
541cs_cbl20_T(1,2) = eval(['cs_cbl20_dev', '.T12(freqind)' ]); % find position
542cs_cbl20_T(2,1) = eval(['cs_cbl20_dev', '.T21(freqind)' ]); % find position
543cs_cbl20_T(2,2) = eval(['cs_cbl20_dev', '.T22(freqind)' ]);
544eval(['clear ', cs_cbl20_dev]);
545
546% Formation of Scattering Transfer Matrix...
547T_xfer(:,:,toneind) = cs_cbl20_T * cs_attn_T * cs_vlf_T *
548ix_comb_T * cs_cbl15_T * cs_attn_T;
549S_xfer(1,1,toneind) = T_xfer(2,1,toneind)/T_xfer(2,2,toneind);
550S_xfer(1,2,toneind) = 1.0/T_xfer(2,2,toneind);
551S_xfer(2,1,toneind) = (T_xfer(2,2,toneind)*T_xfer(1,1,toneind))
552/T_xfer(2,2,toneind);
553S_xfer(2,2,toneind) = T_xfer(2,1,toneind)/T_xfer(2,2,toneind);
554%
clear p_data; % clear the cell array variable containing the
555characterized amplitudes...
556end;
557
558%-------------------------------------------------- End Local Function to get scatter transfer
559matrices ... ---------------------------
560
561%-------------------------------------------------- Begin local Function to transform contraption
562output to amplifier... ---
563% "xfer" refers to the amplifier side of the S-parameters; "src"
564% refers to the source side...
565function [A_ampl_xfer, A_phid_xfer, A_magdbm_xfer] =
566local_get_ampinput(A_ampl_src, A_phid_src, S_xfer)
567num_tones = length(A_ampl_src);
568A_ri_src = A_ampl_src.*complex( cos((pi/180.0)*A_phid_src), sin((pi
569/180.0)*A_phid_src) );
570B_ri_xfer = zeros(size(A_ri_src));
571for i = 1:num_tones
572A = [A_ri_src(1); 0];
573B = S_xfer(:,i)*A;
574B_ri_xfer(i) = B(2);
575end;
576% Output of S-parm block becomes the amplifier input...
577A_ampl_xfer = abs(B_ri_xfer);
578A_magdbm_xfer = aom_vtodbm(A_ampl_xfer);
579A_phid_xfer = (180.0/pi)*angle(B_ri_xfer);
580return;
581%-------------------------------------------------- End local Function to transform contraption
582output to amplifier... ---
% Local Function to transfer amp output through power cable and attenuator...

% Need amplifier to choose correct pwr cable and attenuator, PNA\_dir to get s2p file, ffreq to
% get frequencies to recover for S21, f freq\_index to determine if 1s or 2s, and Y\_dut...
% Must convert Y\_dut to complex vectors, do S21 multiplication, convert back to return...

function Y\_out = local\_get\_output\_scatter\_transfer(sim\_kind\_ch, amplifier, PNA\_dir, slash, ffreq, f freq\_index, Y\_dut);
    if isreal(Y\_dut) \% allow for complex Y\_dut...
        Y\_real = 1;
        Y\_dut\_cv = aom\_svtocv(Y\_dut, f freq\_index);
    else \% complex case, already in "phasor" form...
        Y\_real = 0;
        Y\_dut\_cv = Y\_dut;
    end;
    if sim\_kind\_ch == 'iuc' \% skip phase shift, apply 10^-1 ideal scale for attenuator...
        attn\_scale = sqrt(10.0^-1); \% must take sqrt to bury this scale factor in Y\_out\_cv...
        Y\_out\_cv = attn\_scale*Y\_dut\_cv;
        if Y\_real
            Y\_out = aom\_cvtosv(Y\_out\_cv, f freq\_index);
        else
            Y\_out = Y\_out\_cv;
        end;
    else \% sim\_kind\_ch is 'unc' or 'ctb' both using measured devices...
        switch amplifier
        case {'AR'}
            \% passage through 2.0 foot N-type power cable...
            pwr\_cbl20\_dev = ['pwr\_cbl20\_N'] ; \% file with cable data
            pwr\_cbl20\_loc = [PNA\_dir, slash, pwr\_cbl20\_dev, '.mat'];
            load(pwr\_cbl20\_loc, '-mat');
            cbl\_freq = eval(['pwr\_cbl20\_dev', '.Freq']);
            cbl\_T11 = eval(['pwr\_cbl20\_dev', '.T11']);
            cbl\_T12 = eval(['pwr\_cbl20\_dev', '.T12']);
            cbl\_T21 = eval(['pwr\_cbl20\_dev', '.T21']);
            cbl\_T22 = eval(['pwr\_cbl20\_dev', '.T22']);
            eval(['clear ', pwr\_cbl20\_dev]);
            \% passage through Narda 10db/20w power attenuator
            pwr\_atttn\_dev = ['pwr\_atttn\_10db20w_narda'] ; \% file with cable data
            pwr\_atttn\_loc = [PNA\_dir, slash, pwr\_atttn\_dev, '.mat'];
            load(pwr\_atttn\_loc, '-mat');
            atttn\_T11 = eval(['pwr\_atttn\_dev', '.T11']);
            atttn\_T12 = eval(['pwr\_atttn\_dev', '.T12']);
            atttn\_T21 = eval(['pwr\_atttn\_dev', '.T21']);
            atttn\_T22 = eval(['pwr\_atttn\_dev', '.T22']);
            eval(['clear ', pwr\_atttn\_dev]);
        case {'MC'}
            \% passage through 3.0 foot SMA cable...
            cs\_cbl30\_dev = ['cs\_cbl30'] ; \% file with cable data
        end;
    end;
end;
\begin{verbatim}
cs_cbl30_loc = [PNA_dir, slash, cs_cbl30_dev, '.mat'];
load(cs_cbl30_loc, '-mat');
cbldfreq = eval(['cs_cbl30_dev', '.Freq', '']);
cblT11 = eval(['cs_cbl30_dev', '.T11', '']);
cblT12 = eval(['cs_cbl30_dev', '.T12', '']);
cblT21 = eval(['cs_cbl30_dev', '.T21', '']);
cblT22 = eval(['cs_cbl30_dev', '.T22', '']);
eval(['clear', 'cs_cbl30_dev']);
end;

% passage through Narda 10db/20w power attenuator
pwr_attn_dev = ['pwr_attn_10db_20w_narda''']; % file with
cable data
pwr_attn_loc = [PNA_dir, slash, pwr_attn_dev, '.mat'];
load(pwr_attn_loc, '/A1mat');
attnT11 = eval(['pwr_attn_dev', '.T11', '']);
attnT12 = eval(['pwr_attn_dev', '.T12', '']);
attnT21 = eval(['pwr_attn_dev', '.T21', '']);
attnT22 = eval(['pwr_attn_dev', '.T22', '']);
eval(['clear', pwr_attn_dev']);

otherwise
end; % switch

outT11 = attnT11.*cblT11 + attnT12.*cblT21;
outT12 = attnT11.*cblT12 + attnT12.*cblT22;
outT21 = attnT21.*cblT11 + attnT22.*cblT21;
outT22 = attnT21.*cblT12 + attnT22.*cblT22;
outS21 = (outT22.*outT11 - outT12.*outT21).*outT22;

if ~Y_real
    outS21 = [flipud(conj(outS21)); outS21];
cbldfreq = [-flipud(cbldfreq); cbldfreq];

end;

outS21_Y = ones(size(ffreq)); % Init to ones, not zeros...
for i = 1:length(cbldfreq)
    freqind = find(ffreq == cbldfreq(i));
    outS21_Y(freqind) = outS21(i);
end;

[num_freqs, num_orads] = size(Y_dut_cv);
for j = 1:num_orads
    Y_out_cv(:, j) = Y_dut_cv(:, j).*outS21_Y;
end;

if Y_real
    Y_out = aom_cvtosv(Y_out_cv, ffreq_index);
else
    Y_out = Y_out_cv;
end;

return;

%---------------------------- End local Function to transform contraption
to output to amplifier... ---

%---------------------------- Local Function to get amplifier coefficients...

function [polyreal, polyimag] = local_get_amp_coeffs(amplifier,

\end{verbatim}
amp\_dir, desired\_NL\_order)

switch amplifier
case \{'AR'\}

\%
Multinomial coefs from file...
%
polyreal = [...] +2.5947478e+001 0 ... % 5th order + zero 4th order...
−2.3228863e+001 0 ... % 3rd order + zero 2nd order...
+1.0045465e+002 0]; % 1st order + zero DC ...
%
polyimag = [...] −1.2533859e+002 0 ... % 5th order + zero 4th order...
+8.5100580e+001 0 ... % 3rd order + zero 2nd order...
+1.6632706e+000 0]; % 1st order + zero DC ...
%
case \{'MC'\}
%
Transfer function coefficients...
% from A. Walkers's One-tone measurements...
% Instantaneous multitone coefs from A-file...
cd(amp\_dir);
load G\_coefa.mat % load multinomial coefs...
MC\_real = flipr(rG\_coefa');
MC\_imag = flipr(iG\_coefa');
for i = 1:2:length(MC\_real)
    MC\_real(i) = 0.0; % null the even order entries!
    MC\_imag(i) = 0.0; % null the even order entries!
end;
polyreal = flipr(MC\_real(1:2*str2num(desired\_NL\_order)+1));
polyimag = flipr(MC\_imag(1:2*str2num(desired\_NL\_order)+1));
otherwise
    error('contrap15 :: Invalid amplifier!');
end;
return; % from local function local\_get\_amplifier\_coeffs...

%------------------- End local function to get amplifier coefficients... ---

%------------------- Begin local function to set home dir and slash... ---

function [home\_dir, slash] = local\_get\_home\_slash
% Use 'computer' or 'isp\_c' or 'isunix' and 'is\_student'
if ispc & is\_student % running student version on MB\&D or Prescott
    home\_dir = '\C:\MATLAB\SV701';
    slash = '\\';
elseif ispc & ~is\_student % running R2006a an NCSU Win PC academic - Nighthawk
    home\_dir = '\C:\MATLAB\R2006a\'; % Looks correct, may need to create dirs
    slash = '\\';
elseif isunix & is\_student % running student version on Twin Cities
    home\_dir = '/usr/local/matlab7'; % not sure this is correct.
    slash = '/';
else
    isunix & ~is\_student
    [status, hostname] = unix('hostname');
if hostname(1:5) == 'boreg' % Running R2006a on Boregon
    home_dir = '/local/home/fphart/matlab'; % not sure this is correct.
    slash = '/';
elseif hostname(1:5) == 'italy' % Running R2007b on Italy
    home_dir = '/home/fphart/matlab';
    slash = '/';
else
    error('Unknown computer');
end;
else % Add other computers here...
    error('Unknown computer');
end;
return;

%--------------------- End local function to set home dir and slash...

%--------------------- Begin local function to check/set passed parameters...
function [num tones, amplifier, desired NL order, comb attn, freq, phid, phase num ch, sim num ch, sim kind ch] = ...
    local check set parameters(num tones in, amplifier in, desired NL order in, comb attn in,...
    freq in, phid in, phase num in, sim num in, sim kind in)
% Choose number of tones
if class(num tones in) == 'char'
    num tones in = num2str(num tones in);
end;
switch num tones in
    case '03'
        num tones = '3'; % Single 3–tone contraption through filtered cable.
    case '09'
        num tones = '9';
    case '15'
        num tones = '15'; % 15 tones through 5 combined contraptions with filter and 6–db attn on comb output.
    otherwise
        error('Invalid number of tones specified');
end;

% Choose amplifier: AR or MC.
switch amplifier in
    case 'AR'
        amplifier = 'AR';
    case 'MC'
        amplifier = 'MC';
    otherwise
        error('Invalid amplifier specified');
end;

% Choose desired nonlinear order: 5 to 15
% N.B.: AR strictly limited to 5th order.
if class(desired NL order in) == 'char'
desired_NL_order_in = num2str(desired_NL_order_in);
end;
switch desired_NL_order_in
  case '05'
    desired_NL_order = '05';
  case '07'
    desired_NL_order = '07';
  case '09'
    desired_NL_order = '09';
  case '11'
    desired_NL_order = '11';
  case '13'
    desired_NL_order = '13';
  case '15'
    desired_NL_order = '15';
  otherwise
    error('Invalid nonlinear order specified');
end;
if amplifier == 'AR'
  desired_NL_order = '05';
end;

% Choose desired combiner output attenuator
% This attn fixes input power level to amp.
if class(comb_attn_in) ≠ 'char'
  comb_attn_in = num2str(comb_attn_in);
end;
switch comb_attn_in
  case '03'
    comb_attn = '03';
  case '06'
    comb_attn = '06';
  case '10'
    comb_attn = '10';
  case '20'
    comb_attn = '20';
  otherwise
    error('Invalid attenuator value specified');
end;

% basic checks on passed-in freq
if isempty(freq_in)
  freq = freq_in;
elseif (size(freq_in) ≠ [1,str2num(num_tones)])
  error('Freq vector size not same as num_tones');
elseif class(freq_in) ≠ 'double'
  error('Freq vector not doubles');
else
  freq = freq_in;
end;

% basic checks on passed-in phid
if isempty(phid_in)
  phid = phid_in;
```matlab
elseif (size(phid_in) ≠ [1, str2num(num_tones)])
    error('Phase vector size not same as num_tones');
elseif class(phid_in) ≠ 'double'
    error('Phase vector not doubles');
else
    phid = phid_in;
end;

% ... and sim_num_in...
if isempty(sim_num_in)
    sim_num_ch = ' ';
elseif sim_num_in < 10
    sim_num_ch = ['0', num2str(sim_num_in)];
else
    sim_num_ch = num2str(sim_num_in);
end;

% ... and phase_num_in...
if isempty(phase_num_in)
    phase_num_ch = ' ';
elseif phase_num_in < 10
    phase_num_ch = ['0', num2str(phase_num_in)];
else
    phase_num_ch = num2str(phase_num_in);
end;

% Choose sim type: 'unc' -> uncorrelated multitone; 'ctb' -> narrowband
% CTB; 'iuc' -> ideal unc (skip sig chain)
switch sim_kind_in
    case 'unc'
        % uncorrelated multitone over 5th bandpass (410–490
        % MHz) centered at 450 MHz using characterized simulated
        % source
        sim_kind_ch = 'unc';
    case 'ctb'
        % CTB measurement over narrowband centered at 450
        % MHz, with 450 MHz carrier turned off
        sim_kind_ch = 'ctb';
    case 'iuc'
        % same as 'unc' except source is idealized to
        % average of the carrier powers
        sim_kind_ch = 'iuc';
    case 'cun'
        % same as 'unc' except simulation is done as 2-sided
        % complex rather than 1-sided real
        sim_kind_ch = 'cun';
    otherwise
        error('Invalid sim kind specified');
end;

return;

%---------------------------------- End local function to check/set passed
% parameters... --

%---------------------------------- Begin local function to plot input amplitude in
count_vols... --
function [fignum_out, handle_out] = local_plot_input_ampl_vols(
    freq_scale, A Amplifier, plot_spec, freqscale, fc, df, fmin, fmax, fignum_in)
% Plot the magnitude of the input amplitude (volts)
% if plot_input_ampl_volts
    fignum_out = fignum_in + 1;
    handle_out = figure(fignum_out);
    if isempty(plot_spec)
        plot_spec = 'k-';
    end;
    stem(freq_scale, abs(A_ampl_xfer), plot_spec);
    ymax = max(abs(A_ampl_xfer));
    ymin = 0.0;
    axis([fmin fmax ymin ymax]);
    xlim([fmin fmax]);
    num_tones = length(freq_scale);
    title(['Input Amplitudes for ',num2str(num_tones), ' tones centered at ',
           num2str((fc/freq_scale), '%11.4g'), ' MHz with ',
           num2str((df/freq_scale), '%11.4g'), ' MHz spacing']);
    xlabel('Frequency (MHz)');
    ylabel('Amplitude (Volts)');
%end;
return;

%------------------------------- End local function to plot input amplitude in
volts... --

%------------------------------- Begin local function to plot input amplitude in
dBm... --
function [fignum_out, handle_out] = local_plot_input_ampl_dbm(
    freq_scale, A_magdbm_xfer, plot_spec, negdbmlim, freqscale, fc, df, fmin, fmax, fignum_in)
% Plot the magnitude of the input amplitude (dBm)
% if plot_input_ampl_dbm
    fignum_out = fignum_in + 1;
    handle_out = figure(fignum_out);
    %plot(freq_scale, A_magdbm_xfer, 'd');
    [xx,yy] = local_stemdbm(freq_scale, A_magdbm_xfer, negdbmlim);
    if isempty(plot_spec)
        plot_spec = 'k-';
    end;
    plot(xx,yy, plot_spec);
    ymax = 0.0;
    ymin = negdbmlim;
    axis([fmin fmax ymin ymax]);
    xlim([fmin fmax]);
    num_tones = length(freq_scale);
    title(['Input Amplitudes for ',num2str(num_tones), ' tones centered at ',
           num2str((fc/freqscale), '%11.4g'), ' MHz with ',
           num2str((df/freqscale), '%11.4g'), ' MHz spacing']);
    xlabel('Frequency (MHz)');
    ylabel('Amplitude (dBm)');
%end;
return;
%------------------------------- End local function to plot input amplitude in
dBm... --

F.6 Listings of Matlab Codes used in Chapter 6

This section contains listings of scripts and supporting utilities used to create the content in Chapter 6.

F.6.1 Broadband Chirp Modeling Driver Script

Listing F.27 is a script that drives the function file to simulate the nonlinear response of an amplifier to a broadband Linear FM chirp.

Listing F.27: Script to drive the broadband chirp modeling function.

```matlab
% Broadband Radar Chirp AOM simulation driver script...
% North Carolina State University
% Copyright Frank P. Hart, 2005–2008

clear variables; clc;

% kind of spectral vectors?
sim_kind_ch = '2s_comp';

% number of sides
```
num_sides = 2;
%
\% parameters for chirp generation in the freq.-domain for a 500 MHz chirp
tau = 380e-9; \% Pulse width in time-domain
\% Chirp range
fc = 10.0e9; \% Carrier frequency of chirp
A = 0.2; \% Amplitude of input
fpts = 201; \% number of points in Fresnel evaluation
apts = 57; \% number of points in the AOM input signal (fewer than Fresnel)
f0 = 10e9; \% center freq of BPF
pbw = 600e6; \% passband bw of BPF
Np = 5; \% number of lowpass proto poles (2x this for bandpass)
pbfdb = -1; \% passband flatness
%
\% parameters for chirp generation in the freq.-domain for a 1000 MHz chirp
tau = 380e-9; \% Pulse width in time-domain
\% Chirp range
fc = 10.0e9; \% Carrier frequency of chirp
A = 0.2; \% Amplitude of input
fpts = 201; \% number of points in Fresnel evaluation
apts = 57; \% number of points in the AOM input signal (fewer than Fresnel)
f0 = 10e9; \% center freq of BPF
pbw = 1200e6; \% passband bw of BPF
Np = 5; \% number of lowpass proto poles (2x this for bandpass)
pbfdb = -1; \% passband flatness
%
if sim_kind_ch == '2s_comp'
    options.sim_kind_ch = sim_kind_ch;
    options.spvec_kind = 'comp';
    options.num_sides = 2;
    options.tau = tau;
    options.\%f = \%f;
    options.A = A;
    options.fpts = fpts;
    options.apts = apts;
    options.\%f = \%f;
    options.pbw = pbw;
    options.Np = Np;
    options.pbfdb = pbfdb;
end;
rc = bbc_aom_fcn(options);
F.6.2 Broadband Chirp Modeling Function

Listing F.28 is a function file that simulates the response of a nonlinear amplifier to a broadband Linear FM chirp signal.

Listing F.28: Function file for broadband chirp modeling.

```matlab
function rc = bbc_aom_fcn(options)
% bbc_aom_fcn executes core AOM Toolbox functions to obtain radar function output
% Amplifier (behaviorally) modeled is a Filtrons LMA-411 MMC
% Input: options — structure variable (see below)
% Output: rc — return code (int)
% North Carolina State University
% Copyright Frank P. Hart, 2005–2008
%
% Initialize return code
rc = 0;

% Plot switches...
plot_xin_mag = 1;
plot_xin_ph = 1;
plot_wout_pcoh = 1;
plot_all_wout_inband = 1;
plot_yout_pcoh = 1;
t_start = cputime;
fignum = 0;

% Input options fields...
sim_kind_ch = options.sim_kind_ch;
spvec_kind = options.spvec_kind;
um_sides_in = options.num_sides;
tau = options.tau;
\Delta f = options.\Delta f;
A = options.A;
fpts = options.fpts;
apts = options.apts;
nt = apnts;
```

```matlab
% Simulation time

t_end = cputime;
duration = t_end - t_start;
```

```matlab
% Execution time
execution_time = duration;
```

```matlab
% Output return code
rc = 1;
```

```matlab
% Returning result
return;
```
fo = options.fo;
pbw = options.pbw;
Np = options.Np;
pbfdb = options.pbfdb;
nt_str = num2str(apt3);
\Delta f_str = num2str(\Delta f/1e6);

% Store the directory we entered from and return there upon exit...
entry_dir = pwd;

% Number of sides to spectrum
if num_sides_in == 1
    num_sides = 1;
    bbc_str = 'bbc_1s';
end;
if num_sides_in == 2
    num_sides = 2;
    bbc_str = 'bbc_2s';
end;

% Figure out home directory and 'slash' for the computer we're on...
[home_dir, slash] = local_get_home_slash;

% Some reference directory locations
work_dir = [home_dir, slash,'AOM_bbchirp'];
matfile_str = [work_dir, slash, bbc_str, \Delta f_str, 'df', nt_str, 'fr', spvec_kind, '\_\_'];
% Add phase info?

% Generate the Fresnel integrals
f = linspace(0.0, \Delta f, fpts);
x1 = sqrt(\Delta f*tau/2.0)*(ones(1,fpts)+2.0*f/\Delta f);
x2 = sqrt(\Delta f*tau/2.0)*(ones(1,fpts)-2.0*f/\Delta f);
sx1 = zeros(1,fpts);
sx2 = zeros(1,fpts);
cx1 = zeros(1,fpts);
cx2 = zeros(1,fpts);

for i=1:fpts % use the Fresnels integral package from Matlab File Central...
    [cx1(i), sx1(i)] = fcs(x1(i));
    [cx2(i), sx2(i)] = fcs(x2(i));
end;
Gampl = A*tau*sqrt(1.0/(2.0*\Delta f*tau));
Gphase = exp(complex(0,1)*pi*tau*f.'/2/\Delta f);
Greal = cx1 + cx2;
Gfim = sx1 + sx2;
Gcomp = Gampl*Gphase.*complex(Greal,Gfim);

% Gmag1s = abs(Gcomp);
Gph1s = (180.0/pi)*angle(Gcomp);
Gmag2s = [flip1r(Gmag1s(2:fpts)),Gmag1s];
Gph2s = [flip1r(-(Gph1s(2:fpts))),Gph1s];
f2s = [flip1r(f(2:fpts)),f];
fc = 10e9;
mf2s = f2s + fc;
% 3rd order truncated coefficients from S. Luniya's UWB letter...
polyreal = [ 2.28416 0 -1.07003 0 ];
polyimag = [-2.16753 0 7.80747 0 ];
polycoeff = complex(polyreal,polyimag);
N_next = length(polycoeff) - 1;  % Order of nonlinearity of block in 
blockeval

fc_ind = find(mf2s == fc);  %
hs_fpts = (fc_ind - 1)/2;  % half-side # pts in modulated Fresnel,
   div by 2 to skip far edges
hs_apts = (apts - 1)/2;  % half-side # pts in AOM input signal
apts_incr = round(hs_fpts/hs_apts);
apts_ind_low = [];
for i = 1:hs_apts
    apts_ind_low = [apts_ind_low,fc_ind - apts_incr*i];
end;
apts_ind_high = [];
for i = 1:hs_apts
    apts_ind_high = [apts_ind_high,fc_ind + apts_incr*i];
end;
apts_ind = [flipr(apts_ind_low),fc_ind,apts_ind_high];

% Points at evenly spaced intervals...
aom_freq = mf2s(apts_ind);  % This is vector of freqs of length apts...
aom_mag = Gmag2s(apts_ind);
aom_phase = Gph2s(apts_ind);

if plot_xin_mag == 1;
    fignum = fignum + 1;
    figure(fignum);
    pmag = 20*log10(Gmag2s);
aom_pmag = pmag(apts_ind);
plot(mf2s,pmag,'-k',aom_freq,aom_pmag,'db');
xlabel('Frequency (GHz)');
ylabel('Power (dBW)');
end;

% Plot 1s phase.
if plot_xin_ph == 1;
    fignum = fignum + 1;
    figure(fignum);
PGl1s = unwrap((pi/180)*Gph1s);
PGl2s = (180/pi)*flipr(-(PGl1s(2:fpts)));
Paom_phase = PGl2s(apts_ind);
plot(mf2s,PGl2s,'-k',aom_freq,Paom_phase,'db');
grid;
xlabel('Frequency (GHz)');
ylabel('Phase (degrees)');
end;

% Create the Vector Frequency Description Table
vfd_str = [matfile_str,'vfd.mat'];

if exist(vfd_str)
    load(vfd_str);
else
    htfilename = [matfile_str,'hash_tables.mat'];
    [ht_nvfd, hash_tables, nvfd, nfreq_index, fvd, ffreq_index, ntof, 
        fton, vfd_time] = ...
        aom_vfd(nt, N_next, aom_freq, num_sides, 'X', htfilename);
    save(vfd_str,'ht_nvfd','nvfd','nfirst','nfreq_index','fvd','
        ffreq_index','ntof','fton','vfd_time','nvfd_freq','nvfd_sum','
        nvfd_hash','vfd_duplicates','end_duplicates','hash_hits','hash_tables');
end;

% Create frequency vector from freq-sorted vfd table
ffreq = zeros(size(fvd,1),1);
for n = 1:length(fvd)
    ffreq(n) = cast(fvd(n,:),'double')*transpose(aom_freq);
end;

% Create the Spectral Vector
dc = 0.0;
[X, makespvt ime] = aom_makespv(aom_mag, dc, aom_phase, ffreq_index,
    spvec_kind); % Make arbitrary spectral vector using A & phid

% Create spectrum mapping table
specmap_str = [matfile_str,'specmap.mat'];
if exist(specmap_str)
    load(specmap_str);
else
    [spmaptab, t_ex_specmap] = aom_specmap(ht_nvfd, hash_tables,
        nfreq_index, aom_freq, fvd, ntof, X, N_next, num_sides);
    save(specmap_str,'spmaptab','t_ex_specmap');
end;

if exist('t_ex_specmap')
    specmap_time = t_ex_specmap;
end;

% Create sparse spectrum transform matrix
spectrans_str = [matfile_str,'sspectrans.mat'];
if exist(spectrans_str)
    load(spectrans_str);
else
    [Tx, spectrans_time] = aom_spectrans(X,spmaptab,ffreq_index,'Tx');
    save(spectrans_str,'Tx','spectrans_time');
end;

% Determine % sparsity of Tx
sparsepct = 100.0*nnz(Tx)/prod(size(Tx));

% Evaluate through sparse replacement for block
[Wsim,t_be] = aom_sblockeval(X,Tx,ffreq_index,polycoeff,'Wsim');

% Statistics from run...
fprintf('\n');
fprintf([ 'Time to compute vfd = ', num2str(vfd_time), '\n' ]);
fprintf([ 'Time to compute spec vector = ', num2str(makespv_time), '\n' ]);  
fprintf([ 'Time to compute specmap = ', num2str(specmap_time), '\n' ]);  
fprintf([ 'Time to compute spectrans = ', num2str(spectrans_time), '\n' ]);  
fprintf([ '\n' ]);  
fprintf([ 'Number of sides to spectrum = ', num2str(numsides), '\n' ]);  
fprintf([ 'Order of sys non-linearity = ', num2str(Nnext), '\n' ]);  
fprintf([ 'Pct. non-zero entries in Tx = ', num2str(sparsept), '\n' ]);  
fprintf([ '\n' ]);  
fprintf([ 'Time to compute blockeval = ', num2str(tbe), '\n' ]);  

% lingain = 20*log10(abs(polycoeff(3)));  
pwrcorr = 20*log10(2) + 10*log10(2); % 6 db for adj 2-s to 1-s + 3 db  
for /2 in getfreqcu...  
pthresh = -120;  
refres = 1;  
[pfreq, pfreqi, corfreqi, uncfreqi] = aom_getfreqcu(ffreq, fvd);  
Wpmagcoh = aom_getpowerc(Wsim, pfreq, pfreqi, corfreqi, uncfreqi,  

% Bandpass filter and find power...  
[Ysim, fls, fhs] = aom_butter_bpf(Wsim, ffreq, ffreq_index, fo, pbw,  

if plot_wout_pcoh == 1  
fignum = fignum + 1;  
figure(fignum);  
plot(pfreq, Wpmagcoh-db+pwrcorr);  
end;  

if plot_all_wout_inband == 1  
fignum = fignum + 1;  
figure(fignum);  
pmag = 20*log10(Gmag2s);  
aom_pmag = pmag(aptinds);  
plot(mf2s, pmag+lingain, '-k', aom_freq, aom_pmag+lingain, 'db', pfreq,  

xlim([9.5e9 10.5e9]);  
ylim([-190 -140]);  
xlabel('Frequency (GHz)');  
ylabel('Power (dBW)');  
end;  

if plot_yout_pcoh == 1  
fignum = fignum + 1;  

end;
figure(fignum);
plot(pfreq,Wpmagcoh_db+pwr, Ypmagcoh_db+pwr);
end;

t_end = cputime - t_start;
fprintf('
');
fprintf(['Time elapsed executing code = ', num2str(t_end), '
']);

% save the workspace for use by plotting routines later...
all_fn = [matfile_str, 'all'];
save(all_fn);
cd(entry_dir);
return;

% Begin local function to set home dir and slash...

function [home_dir, slash] = local_get_home_slash
    % Use 'computer' or 'ispc' or 'isunix' and 'istudent'
    if ispc & istudent % running student version on MESSD or Prescott
        home_dir = 'C:\MATLAB\SV701';
        slash = '\';
    elseif ispc & ~istudent % running R2006a an NCSU Win PC academic - Nighthawk
        home_dir = 'C:\MATLAB\R2006a'; % Looks correct, may need to create dirs
        slash = '\';
    elseif isunix & istudent % running student version on Twin Cities
        home_dir = '/usr/local/matlab7'; % not sure this is correct.
        slash = '/';
    elseif isunix & ~istudent % running on either Boregon or Italy...
        [rc,hostname] = unix('hostname');
        if hostname(1:5) == 'italy'
            home_dir = '/home/fphart/matlab';
            slash = '/';
        else % hostname == 'boregon'
            home_dir = '/local/home/fphart/matlab';
            slash = '/';
        end;
    elseif % Add other computers here...
        error('Unknown computer');
    end;
    return;

% End local function to set home dir and slash...
F.7 Listings of Matlab Codes used in Chapter 7

This section contains listings of scripts and supporting utilities used to create the content in Chapter 7.

F.7.1 Multicarrier CATV Modeling Driver Script

Listing F.29 is a script that drives the function file to simulate the response CATV trunk amplifier to a multicarrier input.

Listing F.29: Script to drive CATV amplifier modeling function.

```matlab
% CATV Trunk Amplifier AOM simulation driver script...
%
% North Carolina State University
% Copyright Frank P. Hart, 2005–2008
%
clear variables; clc;

% kind of spectral vectors?
%sim_kind_ch = '1s_real';
%sim_kind_ch = '2s_real';
sim_kind_ch = '2s_comp';

% number of sides
%num_sides = 1;
num_sides = 2;

% 12, 79, 131, 158
%num_channels = '79';
num_channels = '158';

% random or coherent phase?
sim_phase_ch = 'coh';
if sim_phase_ch == 'coh'
    Aphid_src = zeros(1, str2num(num_channels)); % leaves the zero-phase at the top...
else % sim_kind_ch == 'ran'
    rand('state', 0);
    Aphid_src = 360.0*(rand(1, str2num(num_channels))–0.5); % Rand is uniformly distributed between 0 and 1; subtract half...
end;
fprintf('
');

if sim_kind_ch == '1s_real'
    options.sim_kind_ch = sim_kind_ch;
    options.sim_phase_ch = sim_phase_ch;
```
options.spvec_kind = 'real';
options.num_sides = 1;
options.num_channels = num_channels;
options.phid_in = A_phid_src;
options.ac_dbmv_in = 24;
end;

if sim_kind_ch == '2s_real'
options.sim_kind_ch = sim_kind_ch;
options.sim_phase_ch = sim_phase_ch;
options.spvec_kind = 'real';
options.num_sides = 2;
options.num_channels = num_channels;
options.phid_in = A_phid_src;
options.ac_dbmv_in = 24;
end;

if sim_kind_ch == '2s_comp'
options.sim_kind_ch = sim_kind_ch;
options.sim_phase_ch = sim_phase_ch;
options.spvec_kind = 'comp';
options.num_sides = 2;
options.num_channels = num_channels;
options.phid_in = A_phid_src;
options.ac_dbmv_in = 24;
end;

rc = catv_aom_fcn(options);
return;

F.7.2 Multicarrier CATV Modeling Function

Listing F.30 is a function file that simulates the response of a CATV trunk amplifier to a multicarrier input.

Listing F.30: Function file for CATV trunk amplifier modeling.

function rc = catv_aom_fcn(options);

% catv_aom_fcn models the behavior of a CATV trunk amplifier
% Input: options - structure variable (see below)
% Output: rc - return code (int)

% North Carolina State University
% Copyright Frank P. Hart, 2005–2008
% Initialize return code
rc = 0;

t_start = cputime;
fignum = 0;

% Unpack the options from the structure input variable...
sim_kind_ch = options.sim_kind_ch;
sim_phase_ch = options.sim_phase_ch;
spvec_kind = options.spvec_kind;
num_sides_in = options.num_sides;
num_channels_in = options.num_channels;
phid_in = options.phid_in;
ac_dbmv_in = options.ac_dbmv_in;

% Store the directory we entered from and return there
entry_dir = pwd;

% Number of sides to spectrum
if num_sides_in == 1
    num_sides = 1;
catv_str = 'catv_1s'
end;
if num_sides_in == 2
    num_sides = 2;
catv_str = 'catv_2s'
end;

% Figure out home directory and 'slash' for the computer we're on...
[home_dir, slash] = local_get_home_slash;

[num_tones, phid] = ...
local_check_set_parameters(num_channels_in, phid_in);
nt = str2num(num_tones); % Need numeric form for vfd creation...

% Some reference directory locations
work_dir = [home_dir, slash, 'AOM_catv'];
matfile_str = [work_dir, slash, catv_str, num_tones, 'Ch', sim_kind_ch, '
', sim_phase_ch, '
']; % Add phase info?

% Set all carrier amplitudes equal...
ac_dbmv = ac_dbmv_in; % 24 Used in Naishadham's paper
[A_ampl_src] = local_get_source_amplitude(nt, ac_dbmv);

% Initialize rand state...
rand('state', 0);

% Call local function to get random phase assignment....
if nargin == 3
    A_phid_src = phid;
else % nargin = 1, set phid internally...
    [A_phid_src] = local_get_source_phase(A_ampl_src);
% Nominal frequency locations...
% from Naish:
f = 55.25 + (q-1)*6; % q=1,3
%f = 77.25 + (q-4)*6; % q=4,5
%f = 109.25 + (q-6)*6; % q = 6,... nt

freq = []; freqscale = 1e6;
freq1 = [55.25:6:67.25]; % channels 2-4
freq2 = [77.25:6:83.25]; % channels 5,6
freq3 = 109.25 + [0:6:(nt-6)]; % channels 7-max
freq = [freq1,freq2,freq3]*freqscale;
chan1 = [2:1:4];
chan2 = [5:6];
chan3 = [7:1:nt+1];
chan = [chan1,chan2,chan3];

load naish_amp_parms G1 G2 G3;
polyreal = [G3 G2 G1 0];
polyimag = [0 0 0 0];
polycoeff = complex(polyreal,polyimag);
Nnext = length(polycoeff)-1; % Order of nonlinearity of block in blockeval

% Create the Vector Frequency Description Table
vfd_str = [matfile_str,'vfd.mat'];
if exist(vfd_str)
    load(vfd_str);
else
    htfilename = [matfile_str,'hash_tables.mat'];
    [ht_nvfd, hash_tables, nvfd, nfreq_index, fvfd, ffreq_index, ntfon, vfd_time] = ...
    aom_nvfd(nt, Nnext, freq, num_sides, 'X', htfilename);
    save(vfd_str, 'ht_nvfd', 'nvfd', 'nfirst', 'nfreq_index', 'fvfd', 'ffreq_index', 'ntfon', 'vfd_time', 'nvfd_freq', 'nvfd_sum', 'nvfd_hash', 'nvfd_duplicates', 'end_duplicates', 'hash_hits', 'hash_tables');
end;

% Create frequency vector from freq-sorted vfd table
ffreq = zeros(size(fvfd,1),1);
for n = 1:length(fvfd)
    ffreq(n) = cast(fvfd(n,:),'double')*transpose(freq);
end;

% Create the Spectral Vector
dc = 0.0;
[X, makespy_time] = aom_makespy(A_ampl_src, dc, A_phid_src, ffreq_index, spvec_kind); % Make arbitrary spectral vector using A & phid

% Create spectrum mapping table
specmap_str = [matfile_str,'specmap.mat'];
if exist(specmap_str)
    load(specmap_str);
else
    [spmaptab, t_exspecmap] = aom_specmap(ht_nvfd, hash_tables, nfreq_index, freq, fvd, ntol, X, N_next, num_sides);
    save(specmap_str, 'spmaptab', 't_exspecmap');
end;
if exist('t_exspecmap')
    specmap_time = t_exspecmap;
end;
spectrans_str = [matfile_str, 'spectrans.mat'];
if exist(spectrans_str)
    load(spectrans_str);
else
    [Tx, spectrans_time] = aom_spectrans(X, spmaptab, ffreq_index, 'Tx');
    save(spectrans_str, 'Tx', 'spectrans_time');
end;

% Determine % sparsity of Tx
spars pct = 100.0*nnz(Tx)/prod(size(Tx));

% Evaluate through sparse replacement for block
[W_sim, t_be] = aom_sblockeval(X, Tx, ffreq_index, polycoeff, 'W_sim');

% More plotting info...
ffreq_fscale = ffreq/freqscale;

% Statistics from run...
fprintf('
');
fprintf(['Time to compute vfd = ', num2str(vfd_time), '\n']);
fprintf(['Time to compute spec vector = ', num2str(makepv_time), '\n']);
fprintf(['Time to compute specmap = ', num2str(specmap_time), '\n']);
fprintf(['Time to compute spectrans = ', num2str(spectrans_time), '\n']);
fprintf('
');
fprintf(['Number of sides to spectrum = ', num2str(num_sides), '\n']);
fprintf(['Order of sys non-linearity = ', num2str(N_next), '\n']);
fprintf(['Pct. non-zero entries in Tx = ', num2str(sparsect), '\n']);
fprintf('
');
fprintf(['Time to compute b looseval = ', num2str(t_be), '\n']);

% Filter K1...
K1_f_e = 1000e6; K1_ea_db = -0.6; K1_Np = 2;
Y1_sim = aom_butterlpf(W_sim(:,1), ffreq, ffreq_index, K1_f_e, K1_ea_db, K1_Np);

% Filter K2...
K2_f_e = 1000e6; K2_ea_db = -0.8; K2_Np = 2;
Y2_sim = aom_butterlpf(W_sim(:,2), ffreq, ffreq_index, K2_f_e, K2_ea_db, K2_Np);

% Filter K3...
K3_f_e = 860e6; K3_ea_db = -0.6; K3_Np = 2;
Y3_sim = aom_butterlp(fsim(:,3), ffreq, ffreq_index, K3_ce, K3_ce_db, K3_Np);
Y_sim = [Y1_sim, Y2_sim, Y3_sim];

% For a 2-sided problem "solution" make up 1-sided ffreq, fvfd, and freq
% to fake-out the aom_getpfcatv.m routine.
if num_sides == 2
    if isreal(Wsim)
        Wsim_cv2s = aom_svtocv(Wsim, ffreq_index);
        Ysim_cv2s = aom_svtocv(Ysim, ffreq_index);
    else
        Wsim_cv2s = Wsim;
        Ysim_cv2s = Ysim;
    end
    ffreq2s = ffreq;
    fvfd2s = fvfd;
    ffi2s = ffreq_index;
    half_length_dc_index = (size(ffreq_index)-1)/2 + 1;
    ffreq = ffreq2s(half_length_dc_index:end);
    fvfd = fvfd2s(half_length_dc_index:end);
    ffreq_index = ffi2s(half_length_dc_index:end);
    Wsim_cv = Wsim_cv2s(half_length_dc_index:end);
    Wsim_cv_mplr = 2.0*ones(half_length_dc_index,1);
    Wsim_cv_mplr(1) = 1.0;
    for n = 1:size(Wsim_cv,2)
        Wsim_cv(:,n) = Wsim_cv_mplr.*Wsim_cv(:,n);
    end;
    W_sim = aom_cvtsv(Wsim_cv, ffreq_index);
% Ysim_cv = Ysim_cv2s(half_length_dc_index:end);
    Ysim_cv_mplr = 2.0*ones(half_length_dc_index,1);
    Ysim_cv_mplr(1) = 1.0;
    for n = 1:size(Ysim_cv,2)
        Ysim_cv(:,n) = Ysim_cv_mplr.*Ysim_cv(:,n);
    end;
    Y_sim = aom_cvtsv(Ysim_cv, ffreq_index);
end;

[pfreq, pfreqi, lin_pfi, cor_pfi, unc_pfi, ...
csom075_pfi, csop075_pfi, csom125_pfi, csop125_pfi, ...
cthic_pfi, ...
hd2_pfi, hd3_pfi, im2_pfi, im3_pfi, ...
rem_im2_pfi, rem_ctb_pfi] = aom_getpfcatv(ffreq, fvfd, freq);

% Compute Power of memoryless NL Output...
[lin_pfn, lin_pav] = local_get_pavg(lin_pfi, pfreq/freqscale, Wsim, ffreq_index, [1]);
[cor_pfn, cor_pav] = local_get_pcoh(cor_pfi, pfreq/freqscale, Wsim, ffreq_index, [3]);
[unc_pfn, unc_pav] = local_get_pavg(unc_pfi, pfreq/freqscale, Wsim, ffreq_index, [3]);
[cson075_pfn, cson075_pav] = local_get_pavg(cson075_pfi, pfreq/freqscale, Wsim, ffreq_index, [2]);
[csop075_pfn, csop075_pav] = local_get_pavg(csop075_pfi, pfreq/freqscale, W_sim, freq_index, [2]);
[csom125_pfn, csom125_pav] = local_get_pavg(csom125_pfi, pfreq/freqscale, W_sim, freq_index, [2]);
[csop125_pfn, csop125_pav] = local_get_pavg(csop125_pfi, pfreq/freqscale, W_sim, freq_index, [2]);
[ctbc_pfn, ctbc_pav] = local_get_pavg(ctbc_pfi, pfreq/freqscale, W_sim, freq_index, [2]);

% Compute Power of filtered NL Output...

[flin_pfn, flin_pav] = local_get_pavg(lin_pfi, pfreq/freqscale, Y_sim, freq_index, [1]);
[fhd2_pfn, fhd2_pav] = local_get_pavg(hd2_pfi, pfreq/freqscale, Y_sim, freq_index, [2]);
[fhd3_pfn, fhd3_pav] = local_get_pavg(hd3_pfi, pfreq/freqscale, Y_sim, freq_index, [3]);

% powers in dbmv...

lin_pav_dbmv = 10*log10(lin_pav/(.001)^2);
cor_pav_dbmv = 10*log10(cor_pav/(.001)^2);
unc_pav_dbmv = 10*log10(unc_pav/(.001)^2);
csom075_pav_dbmv = 10*log10(csom075_pav/(.001)^2);
csom075_pav_dbmv = 10*log10(csom075_pav/(.001)^2);
csom125_pav_dbmv = 10*log10(csom125_pav/(.001)^2);
csom125_pav_dbmv = 10*log10(csom125_pav/(.001)^2);
csom075_pav_dbmv = 10*log10(csom075_pav/(.001)^2);
ctbc_pav_dbmv = 10*log10(ctbc_pav/(.001)^2);

% Equivalent to指挥

flin_pav_dbmv = 10*log10(flin_pav/(.001)^2);
fhd2_pav_dbmv = 10*log10(fhd2_pav/(.001)^2);
fhd3_pav_dbmv = 10*log10(fhd3_pav/(.001)^2);
fhd3_pav_dbmv = 10*log10(fhd3_pav/(.001)^2);

figure(fignum);
plot(lin_pfn, lin_pav_dbmv, '-dk', ... % linear power
csom075_pfn, csom075_pav_dbmv, '-+b', ... % CSO100- power
csom075_pfn, csom075_pav_dbmv, '-*g', ... % CSO100+ power
% CSO125- power (IM of 3 different carriers)
csom125_pfn, csom125_pav, dbmv, 'sb', ... % CSO125- power
csop125_pfn, csop125_pav, dbmv, 'sg', ... % CSO125+ power
cbtc_pfn, cbtc_pav, dbmv, 'or', ... % CB power @ carrier freq (IM of 3 different carriers)
cor_pfn, cor_pav, dbmv, 'dm', ... % IM3 occurring at same VFD as linear output
unc_pfn, unc_pav, dbmv, 'dy', ... % IM3 occurring at VFDs other than linear output
hd2_pfn, hd2_pav, dbmv, 'xk', ... % HD2 (second order harmonic distortion)
hd3_pfn, hd3_pav, dbmv, 'ok', ... % HD3 (third order harmonic distortion)
rem_im2_pfn, rem_im2_pav, dbmv, ':sr', ... rem_ctb_pfn, rem_ctb_pav, dbmv, ':or');
xlabel('Frequency (MHz)');
ylabel('Number of pm 1.25 MHz CSO Beats');
h2 = legend('CSO 1.25 MHz below carrier', 'CSO 1.25 MHz above carrier');
set(h2, 'Fontsize', 12);

fignum = fignum + 1;
figure(fignum);
plot(csom075_pfn, csom075_pav, dbmv, ':bk', csop075_pfn, csop075_pav, 'ok');
xlabel('Frequency (MHz)');
ylabel('Number of pm 0.75 MHz CSO Beats');
h3 = legend('CSO 0.75 MHz below carrier', 'CSO 0.75 MHz above carrier');
set(h3, 'Fontsize', 12);
fignum = fignum + 1;
figure(fignum);
plot(ctbcf,ctbcN,':sk');
xlabel('Frequency (MHz)');
ylabel('Number of Distortion Products in CTB');
keyboard;
fignum = fignum + 1;
figure(fignum);
plot(lin_pfn,lin_pavdbmv,':sk',flipn_pfn,flipn_pavdbmv,'-sb',... %
    %linear
    hd2_pfn,hd2_pavdbmv,':db',fhd2_pfn,fhd2_pavdbmv,'-sb',... % 2
    %nd harm
    hd3_pfn,hd3_pavdbmv,':dg',fhd3_pfn,fhd3_pavdbmv,'-sg');
hf = legend(['Linear Output W_1(f)', 'Filtered Linear Output Y_1(f)',... 
    '2nd Harmonic Output W_2(f)', 'Filtered 2nd Harmonic Output Y_2(f)',... 
    '3rd Harmonic Output W_3(f)', 'Filtered 3rd Harmonic Output Y_3(f)']);
set(hf,'FontSize',12);
xlabel('Frequency (MHz)');
ylabel('Power (dBmv)');
save(all_fn,'matfile_str','all');
return;

function [inp_pfn ,inp_ncd] = local_get_Ncdisto(inp_pfi , pfreq ,
    ffreq_index);
    inp_pfn = [];
    inp_ncd = [];
    pf_length = size(pfreq,1);
    for n = 1:pf_length
        if ~isempty(inp_pfi{n})
            inp_pfn = [inp_pfn , pfreq(n)]; % add the numeric freq to the freq vector
            inp_ncd = [inp_ncd , length(inp_pfi{n})]; % get the vector of complex vector indices
        end;
    end;
return;

function [inp_pfn ,inp_pav] = local_get_pavg(inp_pfi , pfreq , Y,
    ffreq_index , ord_vect);
    inp_pfn = [];
    inp_pav = [];
    pf_length = size(pfreq,1);
    Ycv = aom_svtocv(Y, ffreq_index);
    for n = 1:pf_length
        pav = 0;
        if ~isempty(inp_pfi{n})
            inp_pfn = [inp_pfn , pfreq(n)]; % add the numeric freq to the freq vector
            cv_pfi = inp_pfi{n}; % get the vector of complex
vector indices;  
for p = 1:length(ord_vect)  % usually ord_vect is one  
    cv = Ycv(:,ord_vect(p));  % make a complex vector of  
    pav = pav + sum(abs(cv(cv_pfi)) .* abs(cv(cv_pfi)));  %  
    pav = pav + sum magnitude squares...  
end;  
inp_pav = [inp_pav, pav];  
end;  
end;  
return;

function [inp_pfn,inp_pco] = local_get_pcoh(inp_pfi, pfreq, Y,  
    ffreq_index, ord_vect);  
inp_pfn = [];  
inp_pco = [];  
pf_length = size(pfreq,1);  
Ycv = aom_svtovcv(Y, ffreq_index);  
for n = 1:pf_length  
    pco = 0;  
    vco = complex(0.0,0.0);  
    if ~isempty(inp_pfi(n))  
        inp_pfn = [inp_pfn, pfreq(n)];  % add the numeric freq to  
        cv_pfi = inp_pfi(n);  % get the vector of complex  
        vector indices;  
        for p = 1:length(ord_vect)  % usually ord_vect is one  
            cv = Ycv(:,ord_vect(p));  % make a complex vector of  
            the selected order...  
            vco = vco + sum((cv(cv_pfi)));  
        end;  
        pco = (abs(vco))^2;  
        inp_pco = [inp_pco, pco];  
    end;  
end;  
return;

%----------------------------------------------------------
%----------------------------------------------------------
%----------------------------------------------------------
%----------------------------------------------------------

%----------------------------------------------------------
% Local Function to get source amplitude...
%----------------------------------------------------------

function [A_ampl_src] = local_get_source_amplitude(nt, ac_dbmv)  
    % Number of tones input determines construction of A_ampl_src  
    A_ampl_src = local_dBmVtoV(ac_dbmv)*ones(1,nt);  
return;

%----------------------------------------------------------
%----------------------------------------------------------
%----------------------------------------------------------
%----------------------------------------------------------
% Begin local function to set source phase...
function [A\_phid\_src] = local\_get\_source\_phase(A\_ampl\_src)
    A\_phid\_src = 360.0\*(rand(size(A\_ampl\_src))\-0.5); % Rand is uniformly distributed between 0 and 1; subtract half...
return;
% End local function to set source phase...

% Begin local function to set home dir and slash...
function [home\_dir, slash] = local\_get\_home\_slash
    if ispc & istudent % running student version on MBS\&D or Prescott
        home\_dir = 'C:\MATLAB\SV701';
        slash = '/';
    elseif ispc & ~istudent % running R2006a an NCSU Win PC academic - Nighthawk
        home\_dir = 'C:\MATLAB\R2006a\'; % Looks correct, may need to create dirs
        slash = '/';
    elseif isunix & istudent % running student version on Twin Cities
        home\_dir = '/usr/local/matlab7'; % not sure this is correct.
        slash = '/';
    elseif isunix & ~istudent % running on either Boregon or Italy...
        [rc, hostname] = unix('hostname');
        if hostname(1:5) == 'italy'
            home\_dir = '/home/fphart/matlab';
            slash = '/';
        else % hostname == 'boregon'
            home\_dir = '/local/home/fphart/matlab';
            slash = '/';
        end;
    else % Add other computers here...
        error('Unknown computer');
    end;
return;
% End local function to set home dir and slash...

% Begin local function to check/set passed parameters...
function [num\_tones, phid] = ...
    local\_check\_set\_parameters(num\_tones\_in, phid\_in)
    % Choose number of tones
    if class(num\_tones\_in) == 'char'
        num\_tones\_in = num2str(num\_tones\_in);
    end;
    switch num\_tones\_in
    case '12' % VHF channels only
        num\_tones = '12';
    case '79' % VHF and UHF in the existing NTSC IRC plan
        num\_tones = '79';
    end;
case '131' % Highest defined frequency in the NTSC IRC plan (over cable)
    num_tones = '131';
case '158' % 1002 MHz / 6 - 9 lower channels = 158...
    num_tones = '158';
case '200' % Limits of VFD code on Boregon before performance thrashing...
    num_tones = '200';
otherwise
    error('Invalid number of tones specified');
end;

% basic checks on passed-in phid
if isempty(phid_in)
    phid = phid_in;
elseif (size(phid_in) ≠ [1,str2num(num_tones)])
    error('Phase vector size not same as num_tones');
elseif class(phid_in) ≠ 'double'
    error('Phase vector not doubles');
else
    phid = phid_in;
end;
return;
%____________________________________ End local function to check/set passed parameters...

% get amplitude in volts from dBmV input...
function vout = local_dBmVtoV(vindbmv)
    vout = 0.001*(10.0).^(vindbmv/20.0);
return;