

## ABSTRACT

DIGIROLAMO, CLAIRE M. Applications of GPOPS-II to Optimal Control Problems with Delays. (Under the direction of Stephen Campbell.)

In this thesis we explore the application of the MATLAB-based optimal control software GPOPS-II to boundary value problems and optimal control problems with constant delays in the state and control. GPOPS-II is a direct transcription software: it discretizes the optimal control problem and directly transcribes it into a large nonlinear programming problem (NLP). In particular, GPOPS-II uses a Legendre-Gauss-Radau  $hp$ -adaptive pseudospectral method to discretize the optimal control problem.  $p$  Pseudospectral methods are global orthogonal collocation schemes. GPOPS-II employs powerful grid refinement algorithms which allow the software to adapt both the number of mesh intervals  $h$  and the order of the interpolating polynomial  $p$  on each interval. The software is designed to exploit the sparsity of the resulting NLP.

While GPOPS-II is an optimal control solver, the software utilizes a boundary value solver philosophy in that it attempts to minimize the error along the entire solution interval. The software's flexible problem formulation allows the user to apply GPOPS-II to a wide array of problems, including using the software as an ODE integrator or a boundary value problem (BVP) solver. For the first time, this thesis examines applying GPOPS-II to certain types of delay differential equations (DDEs), delayed BVPs, as well as delayed optimal control problems (OCPs). These are classes of problems for which GPOPS-II was not designed. These applications are particularly important: currently, there are no delay BVP solvers for MATLAB, and there are very few optimal control solvers which accept delay problems. Delay problems have a wide array of applications and frequently appear in industrial problems, such as when a control responds to sensor data or in the modeling of physical systems.

At the beginning of this thesis, we provide the reader with the necessary background on DDEs and OCPs. We then discuss existing numerical techniques for solving DDEs and OCPs with and without delays. We introduce two new formulations for solving a delay problem in GPOPS-II: one which involves a linear interpolation of the delay term, and an iterative method. We first apply both methods to OCPs with delays in the state, as well as the state delayed BVP which results from the necessary conditions. We present numerical results which demonstrate we are able to successfully solve several types of problems with delays in the state using GPOPS-II. We present a sufficient condition for the convergence of the nonlinear optimizer on a BVP with delay in the state. We then present a condition for the convergence of the nonlinear optimizer for an OCP with delay in the state.

We then discuss solving OCPs and BVPs with delays in the control using GPOPS-II. We present numerical results which demonstrate we may expect GPOPS-II to perform well on these types of problems, provided the user has a sufficiently accurate starting value for the control. In the case where the control computed by GPOPS-II is excessively noisy, we demonstrate a technique which

allows the user to smooth the control. We provide all relevant MATLAB scripts for each type of problem discussed in this thesis in the Appendix.

Applications of GPOPS-II to Optimal Control Problems with Delays

by  
Claire M. Digirolamo

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

Applied Mathematics

Raleigh, North Carolina

2020

APPROVED BY:

---

Hien Tran

---

Ralph Smith

---

Tao Pang

---

Member 4 of Committee

---

Stephen Campbell  
Chair of Advisory Committee

## **DEDICATION**

To my family, whose tireless support made all this possible.

## **BIOGRAPHY**

Claire Marie Digirolamo was born in New Jersey to parents Lisa and Anthony Digirolamo. She has a younger sister, Amelia Digirolamo. She attended high school at Mount Saint Mary Academy in Watchung, NJ. There, she enjoyed participating on the Mock Trial team and as an editor of the school's literary magazine. She also participated in the school's band and chamber music group, in which she played the cello. She graduated Summa Cum Laude and went on to attend George Washington University, where she participated in the university's Honors Program.

While in college she served as a board member of her university's chapter of the Association for Women in Mathematics. She has been interested in science since childhood, and originally planned to study Physics in college. She attributes her switch to mathematics to an excellent Calculus professor her sophomore year. She graduated Magna Cum Laude with a B.S. in Mathematics. After college, she went on to attend graduate school at North Carolina State University in Raleigh, NC where she obtained both her M.S. and Ph.D. in Applied Mathematics.

## **ACKNOWLEDGEMENTS**

I would like to thank my advisor, Dr. Stephen Campbell, for all his help and support. My graduate school experience has not been an easy one, with several significant challenges to my health and the loss of two of my grandparents. Through it all, Dr. Campbell was never anything other than kind and understanding, even when my research progress slowed to crawl. His patience helped me believe I could keep working even when finishing my degree seemed nearly impossible, and I will be forever grateful.

In addition to Dr. Campbell's guidance, this thesis would not have been possible without the help of Dr. John Betts. Dr. Betts generously used his time to run several of our examples in the optimal control software he developed, SOSD. The data Dr. Betts sent gave us a way to judge the quality of our results; we would not have been able to validate our work without him. I would also like to thank my committee members, Dr. Tran, Dr. Smith, Dr. Pang, and Dr. Martin, for all their time and feedback. Special thanks are in order, as in addition to their own work, they took the time to help coordinate my final defense remotely when the University needed to close for the remainder of the semester.

Finally, I must thank my wonderful family, especially my parents. They have loved and supported me every step of way, even taking time off work and traveling to North Carolina to help when I got sick. They have seen me through every challenge, and I would not have been able to finish my education without them.

## TABLE OF CONTENTS

<b>LIST OF TABLES</b> .....	<b>viii</b>
<b>LIST OF FIGURES</b> .....	<b>ix</b>
<b>Chapter 1 INTRODUCTION</b> .....	<b>1</b>
1.1 Dissertation Outline .....	1
1.2 Delay Differential Equations .....	2
1.3 Optimal Control Problems With and Without Delays .....	4
1.4 Existing Numerical Methods .....	9
<b>Chapter 2 Delay Differential Equations</b> .....	<b>12</b>
2.1 Basic Theory .....	12
2.1.1 Existence Results .....	12
2.1.2 Analytic Solution Via Method of Steps .....	15
2.2 Formulation as a Partial Differential Equation .....	16
2.2.1 Reformulation of a Delay Differential Equation as a System of Partial Differential Equations .....	16
2.2.2 Method of Lines .....	17
2.3 Eigenvalues .....	19
2.4 Existing Solvers and Numerical Methods .....	22
2.4.1 MATLAB Delay Differential Equation Solver .....	22
2.4.2 Method of Steps Formulation with MATLAB Boundary Value Problem Solver ..	24
2.4.3 Waveform Relaxation .....	25
<b>Chapter 3 Optimal Control Problems Without Delays</b> .....	<b>27</b>
3.1 Derivation of Necessary Conditions .....	27
3.2 Numerical Methods .....	29
3.2.1 Solving the Necessary Conditions .....	29
3.2.2 Direct Transcription Methods .....	31
3.2.2.1 The Nonlinear Optimizer .....	31
3.2.2.2 Solution of the Optimal Control Problem .....	32
3.2.2.3 GPOPS-II: HP-Adaptive Pseudospectral Method .....	34
3.2.3 Control Parameterization .....	38
<b>Chapter 4 Optimal Control Problems with State Delays</b> .....	<b>41</b>
4.1 Derivation of Necessary Conditions for the Delayed Problem .....	41
4.2 Review of Existing Numerical Methods for Delayed Optimal Control Problems .....	44
4.2.1 Method of Steps .....	44
4.2.2 Control Parameterization .....	45
4.2.3 Direct Transcription .....	46
4.3 Previously Existing Solvers for Delayed Optimal Control Problems .....	47
4.3.1 SOS .....	47
4.3.2 Activate .....	48
4.4 New Use of GPOPS-II on Delayed Optimal Control Problems .....	48
4.4.1 Formulating a Delayed State Optimal Control Problem in GPOPS-II .....	48
4.4.1.1 Collocation Equations for Delayed State Optimal Control Problems	50

4.4.1.2	Case A: Boundary Value Problem for the Delayed Tracking Problem	52
4.4.2	An Iterative Formulation for the Delayed State Optimal Control Problem . . .	53
4.4.2.1	Case B: Boundary Value Problem for Waveform Optimization Problem	54
4.4.3	Error Estimates Between Problem Formulations . . . . .	55
4.4.3.1	Convergence of Delayed Problem . . . . .	55
4.4.3.2	Convergence of Waveform Optimization Problem . . . . .	56
<b>Chapter 5</b>	<b>Numerical Results . . . . .</b>	<b>58</b>
5.1	Use of GPOPS-II as a Delay Differential Equation Solver . . . . .	58
5.1.1	Using the MOS Formulation . . . . .	59
5.1.2	Solving a DDE Using Waveform Relaxation . . . . .	60
5.1.2.1	Example . . . . .	60
5.1.2.2	Comparison of GPOPS-II <sub>m</sub> as a DDE Solver to Matlab DDE Solver .	60
5.1.2.2.1	Solutions on a Short Interval . . . . .	62
5.1.2.2.2	Solutions on a Longer Interval . . . . .	62
5.1.3	Solving the Delayed Necessary Conditions with GPOPS-II <sub>m</sub> . . . . .	65
5.1.3.1	Nonlinear Example Using the Necessary Conditions . . . . .	67
5.1.4	GPOPS-II and the Method of Lines . . . . .	69
5.2	Use of GPOPS-II to Solve Delayed State Optimal Control Problems . . . . .	72
5.2.1	Use of GPOPS-II with Linear Interpolation (GPOPS-II <sub>m</sub> ) . . . . .	72
5.2.1.1	Nonlinear Example . . . . .	77
5.2.2	Use of GPOPS-II and Waveform Optimization (GPOPS-II <sub>low</sub> ) . . . . .	78
5.3	Control Parameterization Using Activate . . . . .	82
5.4	Analysis of Results . . . . .	87
5.4.1	Effect of Delay Coefficient on Solutions . . . . .	90
5.4.1.1	Solving the Necessary Conditions . . . . .	92
5.4.2	Changing the value on the state coefficient . . . . .	96
5.4.2.1	Necessary Conditions . . . . .	96
<b>Chapter 6</b>	<b>Control Delays . . . . .</b>	<b>100</b>
6.1	Control Delays and MOS . . . . .	101
6.2	Solving the Necessary Conditions . . . . .	101
6.2.1	Numerical Results . . . . .	104
6.3	Control Delays and GPOPS-II <sub>m</sub> . . . . .	106
6.3.1	Smoothing the Control . . . . .	109
6.3.1.1	Adding Additional Dynamic Equation . . . . .	109
<b>Chapter 7</b>	<b>Conclusions and Future Research . . . . .</b>	<b>112</b>
7.1	Conclusions and Contributions . . . . .	112
7.2	Future Research . . . . .	113
7.2.1	Smoothing the Control . . . . .	114
7.2.2	Identification of Delay . . . . .	114
7.2.3	Applications of Mesh Refinement for an OCP in Activate . . . . .	114
<b>APPENDIX</b>	<b>. . . . .</b>	<b>121</b>
Appendix A	Relevant Code . . . . .	122
A.1	MATLAB Scripts . . . . .	122
A.1.1	MOS Using GPOPS-II . . . . .	122

A.1.2	GPOPS-II and MOL . . . . .	126
A.1.3	GPOPS-II on the State Delayed Necessary Conditions . . . . .	130
A.1.4	GPOPS-II on Control Delayed Necessary Conditions . . . . .	133
A.1.5	GPOPS-II and Waveform Relaxation . . . . .	137
A.1.6	GPOPS-II <sub>m</sub> on State Delay Differential Equation . . . . .	143
A.1.7	GPOPS-II <sub>m</sub> on OCP with State Delay . . . . .	146
A.1.8	GPOPS-II <sub>m</sub> on OCP with Control Delay . . . . .	150
A.1.9	GPOPS-II <sub>m</sub> on OCP with Control Delay, Extra Variable . . . . .	153
A.1.10	GPOPS-II <sub>ow</sub> State Delay . . . . .	157
A.1.11	Control Parameterization and OCP with State Delay . . . . .	164
Index . . . . .		167

## LIST OF TABLES

Table 3.1	Cost obtained by control parameterization using cubic splines for different number of grid segments, $N$ . . . . .	39
Table 5.1	Mesh refinement study using MOL and GPOPS-II vs SOSD for Eq. 5.16. . . . .	71
Table 5.2	Values of the optimal cost, $J$ , for each $\tau$ and $c = \pm 1.2$ obtained by GPOPS-II, SOSD, and control parameterization. . . . .	75
Table 5.3	Costs from GPOPS-II and MOS for different values of $c$ in Eq. 5.41. . . . .	92
Table 5.4	Timing, in seconds, for different values of $c$ in Eq. 5.41. . . . .	92
Table 5.5	Costs from GPOPS-II and MOS for different values of $c$ in Eq. 5.41 using GPOPS-II on the necessary conditions. . . . .	95
Table 5.6	Timing, in seconds, for different values of $c$ in Eq. 5.41 using GPOPS-II on the necessary conditions. . . . .	95
Table 5.7	Costs from GPOPS-II and MOS for different values of $a$ in Eq. 5.42 using GPOPS-II. . . . .	98
Table 5.8	Timing, in seconds, for different values of $a$ in Eq. 5.42 using GPOPS-II. . . . .	98
Table 5.9	Costs from GPOPS-II and MOS for different values of $a$ in Eq. 5.42 using GPOPS-II on the necessary conditions. . . . .	99
Table 5.10	Timing, in seconds, for different values of $a$ in Eq. 5.42 using GPOPS-II on the necessary conditions. . . . .	99
Table 6.1	Run times, in seconds, and cost for Eq. 6.2 using GPOPS-II and the two phase problem formulation. . . . .	108

## LIST OF FIGURES

Figure 2.1	(a) Eigenvalues of Eq. 2.32 with $\tau = 1$ , $a = -.5$ , $c = -1.2$ and (b) $c = +1.2$ . . . .	21
Figure 2.2	(a) Eigenvalues of Eq. 2.32 with $\tau = 1.5$ , $a = -.5$ , $c = -1.2$ and (b) $c = +1.2$ . . .	21
Figure 2.3	(a) Eigenvalues of Eq. 2.32 with $\tau = .05$ , $a = -.5$ , $c = -1.2$ and (b) $c = +1.2$ . . .	21
Figure 2.4	Solution of Eq. 2.41 using MOS. . . . .	25
Figure 3.1	(a) State and (b) control obtained by GPOPS-II and control parameterization for Eq. 3.32 with cost Eq. 3.33 using linear splines and $N = 30$ segments. . . .	39
Figure 3.2	(a) State and (b) control obtained by GPOPS-II and control parameterization for Eq. 3.32 with cost Eq. 3.33 using cubic splines and $N = 30$ segments. . . .	39
Figure 3.3	(a) State and (b) control obtained by GPOPS-II and control parameterization for Eq. 3.32 with cost Eq. 3.33 using cubic splines and $N = 28$ segments for new initial value $u(0) = 6*(1:N+1)^{-1}$ . . . . .	40
Figure 5.1	MOS (a) State and (b) control from Eq. 4.19 with $c = +1.2$ , $\tau = 1$ . . . . .	59
Figure 5.2	MOS (a) State and (b) control from Eq. 4.19 with $c = -1.2$ , $\tau = 1$ . . . . .	59
Figure 5.3	(a) Solutions obtained by GPOPS-II and dde23 and (b) solutions obtained by ode45 and dde23 to Eq. 5.2 for $c = -1.2$ and $\tau = 1.0$ . . . . .	61
Figure 5.4	(a) Residuals from GPOPS-IIw and dde23 and (b) residuals from ode45 and dde23 for Eq. 5.2 with $c = -1.2$ and $\tau = 1.0$ . . . . .	61
Figure 5.5	(a) Asymptotically stable Eq. 5.3a solution in dde23 and GPOPS-IIw and (b) difference of solutions on $[0,4]$ with tolerances $10^{-7}$ . . . . .	62
Figure 5.6	(a) Barely asymptotically stable Eq. 5.4a solution in dde23 and GPOPS-IIw and (b) difference of solutions on $[0,4]$ with tolerances $10^{-7}$ . . . . .	63
Figure 5.7	(a) Unstable Eq. 5.5a solution in dde23 and GPOPS-IIw and (b) difference of solutions on $[0,4]$ with tolerances $10^{-7}$ . . . . .	63
Figure 5.8	(a) Slightly unstable Eq. 5.6a solution in dde23 and GPOPS-IIw and (b) difference of solutions on $[0,4]$ with tolerances $10^{-7}$ . . . . .	63
Figure 5.9	(a) Asymptotically stable Eq. 5.3a solution in dde23 and GPOPS-IIw and (b) difference of solutions on $[0, 23.95]$ with tolerances $10^{-7}$ . . . . .	64
Figure 5.10	(a) Barely asymptotically Eq. 5.4a solution in dde23 and GPOPS-IIw and (b) difference of solutions on $[0,23.95]$ with tolerances $10^{-7}$ . . . . .	64
Figure 5.11	(a) Unstable Eq. 5.5a solution in dde23 and GPOPS-IIw and (b) difference of solutions on $[0, 23.95]$ with tolerances $10^{-7}$ . . . . .	64
Figure 5.12	(a) Slightly unstable Eq. 5.6a solution in dde23 and GPOPS-IIw and (b) difference of solutions on $[0,23.95]$ with tolerances $10^{-7}$ . . . . .	65
Figure 5.13	(a) Asymptotically stable Eq. 5.4a solution from dde23 and GPOPS-IIw and (b) difference of solutions on $[0,20]$ for tolerance $10^{-8}$ . . . . .	65
Figure 5.14	(a) Unstable solution Eq. 5.5a from dde23 and GPOPS-IIw and (b) difference of solutions on $[0,23.95]$ for tolerance $10^{-8}$ . . . . .	66
Figure 5.15	(a) State and (b) control for necessary conditions Eq. 4.34 with $\tau = 1$ , $c = -1.2$ . . . .	66
Figure 5.16	(a) State and (b) control the modified Van der Pol oscillator with cost $J = 1.9866728$ . . . . .	68
Figure 5.17	Optimal state and control for modified Van der Pol Oscillator from [53]. . . . .	68
Figure 5.18	(a) State and (b) control obtained for Eq. 5.16 using GPOPS-II and MOL with $\tau = 1$ and $c = -1.2$ . . . . .	70

Figure 5.19	(a) State and (b) control obtained for Eq. 5.16 with grid size $N = 5, 10, 20$ and $\tau = 1, c = -1.2$ . . . . .	70
Figure 5.20	(a) State and (b) control obtained by GPOPS-IIIm, control parameterization, SOSD for $c = -1.2, \tau = 0.05$ with cost Eq. 5.8. . . . .	73
Figure 5.21	(a) State and (b) control obtained by GPOPS-IIIm, control parameterization, SOSD for $c = -1.2, \tau = 1$ with cost Eq. 5.8. . . . .	73
Figure 5.22	(a) State and (b) controls from GPOPS-IIIm using linear interpolation and splines on the delay term, compared to SOSD for $c = -1.2, \tau = 1$ with cost Eq. 5.8. . . . .	74
Figure 5.23	(a) State and (b) control obtained by GPOPS-IIIm, control parameterization, SOSD for $c = 1.2, \tau = 0.05$ with cost Eq. 5.8. . . . .	74
Figure 5.24	(a) State and (b) control obtained by GPOPS-IIIm, control parameterization, SOSD for $c = 1.2, \tau = 1$ with cost Eq. 5.8. . . . .	75
Figure 5.25	(a) State obtained by GPOPS-IIIm and dde23 for $c = 1.2$ and (b) $c = -1.2, \tau = 1$ with cost Eq. 5.8. . . . .	76
Figure 5.26	(a) State and (b) control for modified Van der Pol oscillator using GPOPS-IIIm with cost $J = 2.0237941$ . . . . .	77
Figure 5.27	Optimal state and control for modified Van der Pol Oscillator from [53]. . . . .	77
Figure 5.28	(a) States and (b) controls from GPOPS-IIow for Eq. 5.24 with $\tau = 1$ and $c = -1.2$ . . . . .	78
Figure 5.29	(a) Comparison of states and (b) controls produced by GPOPS-IIow, GPOPS-IIIm, and control parameterization for Eq. 5.19 with $\tau = 1$ and $c = -1.2$ . . . . .	79
Figure 5.30	(a) Comparison of states and (b) controls produced by GPOPS-IIow, GPOPS-IIIm, and control parameterization for Eq. 5.19 with $\tau = 1$ and $c = +1.2$ . . . . .	79
Figure 5.31	(a) Comparison of states and (b) controls produced by GPOPS-IIow, GPOPS-IIIm, and control parameterization for Eq. 5.19 with $\tau = 1.7$ and $c = +1.2$ . . . . .	80
Figure 5.32	(a) Comparison of states and (b) controls produced by GPOPS-IIow, GPOPS-IIIm, and control parameterization for Eq. 5.19 with $\tau = 1.7$ and $c = -1.2$ . . . . .	81
Figure 5.33	(a) Comparison of states and (b) controls produced by GPOPS-IIow, GPOPS-IIIm, and control parameterization for Eq. 5.19 with $\tau = 0.4$ and $c = -1.2$ . . . . .	81
Figure 5.34	(a) Comparison of states and (b) controls produced by GPOPS-IIow, GPOPS-IIIm, and control parameterization for Eq. 5.19 with $\tau = 0.4$ and $c = +1.2$ . . . . .	82
Figure 5.35	State from GPOPS-IIow and state obtained by dde23 integrating the delayed dynamics with $c = -1.2, \tau = 1$ . . . . .	82
Figure 5.36	Activate diagram using control parameterization to solve Eq. 5.26 with $c = +1.2$ and $\tau = 1$ . . . . .	84
Figure 5.37	Initialization script for Eq. 5.26 with $c = +1.2$ and $\tau = 1$ . . . . .	85
Figure 5.38	BobyqaOpt block parameters for Eq. 5.26 with $c = +1.2$ and $\tau = 1$ . . . . .	85
Figure 5.39	(a) State and control obtained from Activate for Eq. 5.26 and (b) state obtained by control parameterization and GPOPS-II with $c = +1.2$ and $\tau = 1$ . . . . .	86
Figure 5.40	(a) State and (b) control by GPOPS-IIIm and MOS with $c = .1$ for Eq. 5.41, cost $J = 7.3484273$ from GPOPS-IIIm, and $J = 7.3476357$ from MOS. . . . .	90
Figure 5.41	(a) State and (b) control of Eq. 5.41 with $c = .5$ , cost 3.7867743 from GPOPS-IIIm, and cost $J = 3.7859437$ from MOS. . . . .	91
Figure 5.42	(a) State and (b) control of Eq. 5.41 with $c = 1.0$ , cost 5.6303104 from GPOPS-IIIm, and cost $J = 5.5426332$ from MOS. . . . .	91
Figure 5.43	(a) State and (b) control of Eq. 5.41 with $c = 1.5$ , cost 13.9834065 from GPOPS-IIIm, and cost $J = 13.3242859$ from MOS. . . . .	91

Figure 5.44	(a) State and (b) control of Eq. 5.41 with $c = 2.0$ , cost 27.4879834 from GPOPS-IIIm, and cost $J = 25.4211318$ from MOS. . . . .	92
Figure 5.45	(a) State and (b) control of Eq. 5.41 with $c = .1, a = -.5$ , cost 7.3477053 from GPOPS-IIIm with necessary conditions and cost $J = 7.3476357$ from MOS. . .	93
Figure 5.46	(a) State and (b) control of Eq. 5.41 with $c = .5, a = -.5$ , cost 3.7861046 from GPOPS-IIIm with necessary conditions and cost $J = 3.7859437$ from MOS. . .	93
Figure 5.47	(a) State and (b) control of Eq. 5.41 with $c = 1, a = -.5$ , cost 5.5421608 from GPOPS-IIIm with necessary conditions and cost $J = 5.5426332$ from MOS. . .	94
Figure 5.48	(a) State and (b) control of Eq. 5.41 with $c = 1.5, a = -.5$ , cost 13.3243674 from GPOPS-IIIm with necessary conditions and cost $J = 13.3242859$ from MOS. . . . .	94
Figure 5.49	(a) State and (b) control of Eq. 5.41 with $c = 2.0, a = -.5$ , cost 25.4211284 from GPOPS-IIIm with necessary conditions and cost $J = 25.4211318$ from MOS. . . . .	94
Figure 5.50	(a) State and (b) control of Eq. 5.42 with $a = 0.1, c = -1.2$ , cost 22.2395200 from GPOPS-IIIm and cost $J = 21.5224898$ from MOS. . . . .	96
Figure 5.51	(a) State and (b) control of Eq. 5.42 with $a = 20.0, c = -1.2$ , cost 222.1349950 from GPOPS-IIIm and cost $J = 222.1272862$ from MOS. . . . .	97
Figure 5.52	(a) State and (b) control of Eq. 5.42 with $a = -0.1, c = -1.2$ , cost 29.2185129 from GPOPS-IIIm and cost $J = 28.3062769$ from MOS. . . . .	97
Figure 5.53	(a) State and (b) control of Eq. 5.42 with $a = -20.0, c = -1.2$ , cost 196.2159127 from GPOPS-IIIm and cost $J = 196.2104009$ from MOS. . . . .	97
Figure 5.54	(a) State and (b) control using GPOPS-IIIm on the necessary conditions for Eq. 5.42 with $a = -1.5$ and $c = -1.2$ and cost $J = 82.4499980$ from GPOPS-IIIm. . .	98
Figure 5.55	(a) State and (b) control using GPOPS-IIIm on the necessary conditions for Eq. 5.42 with $a = +0.1$ and $c = -1.2$ and cost $J = 21.5220325$ from GPOPS-IIIm. . .	99
Figure 6.1	(a) State and (b) control obtained using MOS for Eq. 6.2 and cost $J = 53.27062.101$	
Figure 6.2	(a) State and (b) control obtained using MOS for Eq. 6.3 and cost $J = 56.18724.102$	
Figure 6.3	(a) State and (b) control obtained by GPOPS-IIIm solving the necessary conditions for Eq. 6.2 with cost $J = 53.27083$ . . . . .	104
Figure 6.4	(a) State and (b) control obtained by GPOPS-IIIm solving the necessary conditions for Eq. 6.3 with cost $J = 55.2559463$ . . . . .	105
Figure 6.5	State obtained by GPOPS-II and obtained by ode4 using the control computed by GPOPS-IIIm solving the necessary conditions for Eq. 6.3. . . . .	106
Figure 6.6	(a) State and (b) control using GPOPS-IIIm for Eq. 6.2 with $J = 53.3702471$ . . .	106
Figure 6.7	(a) State and (b) control for Eq. 6.3 using GPOPS-IIIm with cost $J = 57.1385819.108$	
Figure 6.8	Comparison of states from GPOPS-IIIm and ode4 for Eq. 6.3. . . . .	109
Figure 6.9	(a) State and (b) control for Eq. 6.2 using GPOPS-IIIm and adding $\dot{u} = z$ to the dynamics, cost $J = 56.824463$ . . . . .	110
Figure 6.10	(a) State and (b) control for Eq. 6.3 using GPOPS-IIIm and adding $.01\ z\ ^2$ to cost, with $J = 61.665679$ . . . . .	110
Figure 6.11	(a) State and (b) control for Eq. 6.3 using GPOPS-IIIm and adding $\epsilon\ z\ ^2$ to cost, $\epsilon$ as in Eq. 6.15, with $J = 57.5017110$ . . . . .	111

## CHAPTER

# 1

# INTRODUCTION

## 1.1 Dissertation Outline

Chapter 1 contains our introduction. In Chapter 2, we discuss delay differential equations (DDEs) in-depth. We consider methods for obtaining the analytic solution of a DDE via the method of steps as well as ways to reformulate a DDE as a system of ordinary and partial differential equations. We discuss existing numerical methods for solving DDEs, including those in a built-in MATLAB DDE solver. In Chapter 3 we discuss numerical methods for optimal control problems without delays. We provide a discussion of solving the necessary conditions as well as direct transcription methods. We examine the *hp*-adaptive pseudospectral method implemented by the optimal control software GPOPS-II.

In Chapter 4, we examine optimal control problems with state delays and examine the extent to which GPOPS-II can be used to solve these delayed state optimal control problems. We derive the necessary conditions for two different problem formulations and demonstrate that for one formulation the delayed problem will converge to the undelayed case. We derive the collocation equations for using GPOPS-II and demonstrate a new problem formulation to allow GPOPS-II to solve a delayed state optimal control problem. We propose the use of incorrect Jacobians by GPOPS-II leads the software to produce a suboptimal solution for certain problems.

Chapter 5 contains our numerical results, and subsequent analysis. We show that using the necessary conditions, it is often possible to use GPOPS-II to obtain an accurate solution of a delayed

state optimal control problem. We also demonstrate that on more unstable problems with longer delays, using GPOPS-II iteratively is often a viable method for producing a suboptimal solution to the delayed optimal control problem. We implement control parameterization to solve a delayed state optimal control problem using the software Activate. We present new results on the convergence of GPOPS-II's nonlinear optimizer IPOPT both on delayed optimal control problems as well as the delayed boundary value problem.

In Chapter 6, we introduce the use of GPOPS-II on optimal control problems with control delays, and demonstrate that using GPOPS-II to solve the necessary conditions gives an accurate solution for certain types of problems. We discuss techniques for smoothing the control for the delayed optimal control problem. In Chapter 7 we make our conclusions, summarize the contributions of this thesis, and consider applications for further research.

## 1.2 Delay Differential Equations

A delay differential equation is a differential equation of the form

$$\dot{x}(t) = f(x(t), x_t, t), \quad (1.1)$$

where the change in the state,  $x$ , depends not only on the present state  $x(t)$  but also on past values  $x_t$  [8]. Referred to in early literature as differential equations with deviating arguments, one of the first known differential equations with a delayed argument originally appeared in the work of French mathematician and aristocrat Marquis de Condorcet in the late 1700s [58]. While mathematicians remained aware of delayed effects in mathematical systems, the theory required to work with these equations did not begin to be developed until the 20th century [36, 49]. Delay equations arise naturally in any system model which must account for reaction time, and therefore have a wide range of applications in engineering, biology, chemistry, and physics [25, 36, 49, 56].

Delays can be time-dependent with

$$\dot{x}(t) = f(x(t), x(t - \tau(t)), t). \quad (1.2)$$

Time-varying delays often appear in biological systems. For example, the modified Mackey-Glass model

$$\dot{x} = \frac{a x(t - \tau(t))}{1 + x^{10}(t - \tau(t))} - b x(t) \quad (1.3)$$

$$\tau(t) = \tau_0 + \int_0^t \xi(s) ds \quad (1.4)$$

introduces a time-varying delay term to model blood cell regeneration in leukemia patients [48].

Delay terms can also be state-dependent, with

$$\dot{x}(t) = f(x(t), x(t - \tau(t, x(t)))). \quad (1.5)$$

We consider DDEs with fixed, discrete delays. A general form of the linear problem with constant coefficients is given by [8]

$$a \dot{x}(t) + b \dot{x}(t - \tau) + c x(t) + d x(t - \tau) = f(t). \quad (1.6)$$

The general Eq. 1.6 can be categorized into three cases. For the case  $a \neq 0$  and  $b \neq 0$  we have a DDE of the *neutral type*,

$$a \dot{x}(t) + c x(t) + d x(t - \tau) = f(t), \quad (1.7)$$

where delays appear in derivatives of the state. If  $a = b = 0$  or  $c = d = 0$ , then the equation is said to be a *pure difference equation* [8], for example,

$$c x(t) + d x(t - \tau) = f(t). \quad (1.8)$$

In this thesis will we focus on the case  $a \neq 0$ ,  $b = 0$ , a DDE of the *retarded type*. A general nonlinear version with several delays takes the form

$$\dot{x}(t) = f(t, x(t), x(t - \tau_1), \dots, x(t - \tau_m)), \quad \tau_m \geq \dots \geq \tau_1 \geq 0, \quad (1.9)$$

where delays are constant and appear only in the state and not in the derivative [8]. In the remainder of this work, when we discuss a general DDE we are referring to an equation of the form given in Eq. 1.9.

We note that with the introduction of a delay term it is no longer sufficient to specify an initial condition to obtain a unique solution of the problem. Rather we need to specify a prehistory function,

$$\phi(t), \quad \phi \in \mathcal{C}([-\tau_m, 0], \mathbb{R}^n) \quad (1.10)$$

[56]. Thus we have an infinite-dimensional set of initial conditions and even a single, linear DDE is an infinite-dimensional problem [25]. We note that choosing a continuous prehistory function  $\phi(t)$  is one possibility; one can have a looser requirement that  $\phi(t)$  simply be integrable. We further discuss the relevant theory of DDEs in Chapter 2.

We note we may also have advances of the form

$$\dot{x} = x(t + \tau), \quad t \geq 0. \quad (1.11)$$

While we do not consider specific examples with advances in the dynamics, advances will appear in

our equations for the necessary conditions of our delayed optimal control problems. Because our focus is on direct transcription codes, which use a boundary value problem philosophy, there is no significant difference in the implementation of delays versus advances [12]. This will become apparent when we discuss and solve the necessary conditions in Chapters 4 and 5.

DDEs have a wide range of applications, from population dynamics to engineering. We present an industrial example from Kyrychko and Hogan [49]. High-speed milling is a common industrial cutting process using a rotating tool. Let  $x$  be the displacement of the cutting tool. Then the motion of the cutting tool can be modeled by the equation

$$\ddot{x}(t) + 2\xi\omega_n\dot{x}(t) + \omega_n^2x(t) = \frac{g(t)}{m}F_c(h(t)), \quad (1.12)$$

where  $K$  is an experimentally-determined parameter,  $\omega_n$  is the natural undamped frequency,  $\xi$  is the relative damping factor, and  $w$  is the constant chip width. The term  $F_c$  is a cutting force, usually modeled as a power law

$$F_c(h(t)) = K w(h(t))^{3/4} \quad (1.13)$$

and

$$g(t) = \begin{cases} 0 & \text{if there exists } j \in \mathbb{Z} : t_j \leq t \leq t_{j+1}^-, \\ 1 & \text{if there exists } j \in \mathbb{Z} : t_{j+1}^- \leq t < t_{j+1}, \end{cases} \quad (1.14)$$

where  $t_{j+1}^-$  are times at which the tool begins cutting,  $t_{j+1}$  the times at which the tool finishes cutting, and  $t_j$  are the times at which the tool starts free vibration. The delay arises in the term representing the chip thickness,  $h(t)$ , where

$$h(t) = h_0 + x(t - \tau) - x(t) \quad (1.15)$$

and  $h_0 = v_0\tau$  is the feed for a cutting period. The thickness  $h$  of the chip being cut depends on both the current tool tip position as well as its previous position [49].

### 1.3 Optimal Control Problems With and Without Delays

Consider a dynamical system without delays

$$\dot{x} = f(x(t), u(t), t) \quad (1.16)$$

and a cost functional

$$J = \phi(x(T), T) + \int_{t_0}^T L(x(t), u(t), t) dt, \quad x(t_0) = x_0, \quad (1.17)$$

where  $x(t)$  is the state and  $u(t)$  is the control input. The functional  $J$  is what we wish to minimize. The term  $\phi(x(T), T)$  is the final weighting function, dependent upon the state  $x$  at the final time  $T$ , and  $\int_{t_0}^T L dt$  is the cost on the trajectory and control. Consider also a terminal condition  $\psi(x(T), T) = 0$ . The *optimal control* is the input  $u^*$  which steers our system in Eq. 1.16 along a trajectory  $x^*$  which minimizes our cost functional  $J$  in Eq. 1.17 and satisfies our terminal condition [51].

The need for a theory of optimal control arose after World War II, when engineers began tackling the problems of flight dynamics in high speed aircraft and optimizing the trajectories of rockets [47, 63]. The field lacked many of the mathematical tools needed until Pontryagin and his colleagues formulated a maximum principle for a minimum time optimal control problem utilizing the calculus of variations. We give a statement of Pontryagin's maximum principle. Consider the Hamiltonian function

$$H(x(t), u(t), \lambda(t), t) = \lambda^T(t)f(x(t), u(t)) + L(x(t), u(t), t) \quad (1.18)$$

with  $x(t) \in \mathbb{R}^n$ ,  $u(t) \in D \subset \mathbb{R}^m$ , and  $\lambda(t) \in \mathbb{R}^n$  is the costate vector. The maximum principle states that for a control  $u^*(t)$  and a trajectory  $x^*(t)$  to be optimal, there must exist a nonzero variable vector  $\lambda^*(t)$  which satisfies the equation Eq. 1.19a with the additional conditions Eq. 1.19b and Eq. 1.19c:

$$-\dot{\lambda} = H_x(x^*(t), u^*(t), \lambda(t), t) \quad (1.19a)$$

$$\phi_T(x(T)) + H(x(T), u(T), \lambda(T), T) = 0 \quad (1.19b)$$

$$\lambda^T(T) = \phi_x(x(T)), \quad (1.19c)$$

$t_0 \leq t \leq t_1$ , where we only have Eq. 1.19c if  $x(T)$  is free, and which satisfies

$$H(x(t), u(t), \lambda(t), t) = \max_{u \in U} H(x(t), u(t), \lambda(t), t), \quad \forall t \in [t_0, t_1], \quad (1.20)$$

where we assume  $u(t)$  is piecewise continuous [63, 64]. For  $x(t_0)$  fixed, we will have  $\lambda(t_0)$  free. We note that, depending upon the sign on  $H$ , the principle may be called Pontryagin's minimum principle. We give a derivation of the necessary conditions for an unconstrained optimal control problem using this maximum principle in Chapter 3. With the preliminary theory in place, the field of optimal controls continued to grow throughout the 20th century. Optimal control theory has a wide variety of applications including trajectory optimization, production models in factories, and mechanical engineering [18, 47, 69].

In this work we primarily consider problems with quadratic cost, or performance index,

$$J = \phi(x(T), T) + \int_{t_0}^T (x - r)^T Q(x - r) + u^T R u dt. \quad (1.21)$$

In our case,  $r$  is a reference trajectory and Eq. 1.21 is a type of tracking problem. The idea of a quadratic performance index was originally introduced by Kalman in 1960 and eventually became known as a Linear Quadratic Regulator (LQR) [41]. Kalman was able to show that "the optimal controls were linear feedbacks of the state variables" for both multiple-input multiple output (MIMO) and linear time varying systems [18].

Below we present an example from Jahromi, Xie, and Bhat [40] of an LQR used to control damping in a semi-active car suspension system. A semi-active suspension utilizes a controllable damper with an actuator. The researchers sought to minimize the pitch and roll rates as well as the bounce velocity of the car body as the car passed over bumps in the road. The car is modeled as a four mass-spring system with seven degrees of freedom.

The equations of motion and the state space equations, in matrix form, are

$$M \ddot{x}_s + B \dot{x}_s + K x_s = F_{MR} + F_{road} \quad (1.22a)$$

$$\dot{z}_s = A z_s + B_{MR} u + W v \quad (1.22b)$$

$$Y = C z_s + D u, \quad (1.22c)$$

where  $x$  is the displacement of the suspension springs,  $u$  is the control,  $v$  represents the disturbance from the road, and  $M$  is the mass. Matrix  $B$  is damping,  $K$  is stiffness,  $F_{MR}$  is the actuating force, and  $F_{road}$  is the force due to the terrain of the road. Matrix  $A$  contains the dynamic properties of the system, the effect of the actuator is modeled by  $B_{MR}$ , and the effect of the profile of the road is modeled by  $W$ . The output of the system  $Y$  is given by the  $14 \times 14$  identity matrix  $C$ , and in the researchers' model the static effect in the matrix  $D$  is zero [40]. The performance index

$$J = \frac{1}{2} \int_0^\infty x^T(t) Q x(t) + u^T(t) R u(t) dt \quad (1.23)$$

sought to minimize the actuator energy needed, where  $Q \geq 0$  is symmetric positive semi-definite and  $R > 0$  is symmetric positive definite.

Delays are frequently incorporated into control systems with a variety of applications, such as modeling the incubation period or treatment delay of a disease [74], economic models [17], and transmission delays in networked control systems [39]. Much work has been done to formulate a minimum principle for optimality in optimal control problems with delays; we note only a small number of works here. A minimum (maximum) principle for optimal control problems with constant delays in the state was originally derived by Kharatishvili in 1961 [44]. Frankena derived a minimum principle for optimal control problems with delays in the state and control as well as inequality restrictions on both the state and control variables in 1975 [27]. In this work, we utilize the minimum principle as well as the necessary conditions derived by Göllmann, Kern, and Mauer [32, 33] for optimal control problems with delays in both the state and control and mixed inequality constraints.

Below we give a statement of the optimality conditions developed by Göllmann, Kern, and Mauer from [32].

Consider the general, not necessarily linear, delayed optimal control problem

$$\min J(x, u) = g(x(b)) + \int_a^b L(t, x(t), x(t-\tau), u(t), u(t-s)) dt, \quad (1.24)$$

where  $\tau \geq 0$  and  $s \geq 0$  are constant delays in the state and control respectively,  $x(t) \in \mathbb{R}^n$ ,  $u(t) \in \mathbb{R}^m$ . The cost functional in Eq. 1.24 is subject to the constraints

$$\dot{x} = f(t, x(t), x(t-\tau), u(t), u(t-s)) \quad \text{a.e. } t \in [a, b] \quad (1.25a)$$

$$x(t) = \phi(t), \quad t \in [a-\tau, a] \quad (1.25b)$$

$$u(t) = \psi(t), \quad t \in [a-s, a] \quad (1.25c)$$

$$w(x(T)) = 0 \quad (1.25d)$$

$$C(t, u(t), x(t)) \leq 0, \quad t \in [a, b], \quad (1.25e)$$

where  $\phi(t)$  and  $\psi(t)$  are the prehistory functions for the delayed state and control,  $w$  is the terminal condition, and “a.e.” stands for “almost everywhere”. The authors assume the following functions are twice continuously differentiable with respect to all arguments:

$$g : \mathbb{R}^n \rightarrow \mathbb{R}$$

$$f : [a, b] \times \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^m \rightarrow \mathbb{R}$$

$$w : \mathbb{R}^n \rightarrow \mathbb{R}^q, \quad 0 \leq q \leq n$$

$$C : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^p$$

$$L : [a, b] \times \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^m \rightarrow \mathbb{R}.$$

A state  $x$  and a control  $u$  are called admissible if they satisfy the constraints in Eq. 1.25 with  $(x, u) \in W^{(1,\infty)}([a, b], \mathbb{R}^n) \times L^\infty([a, b], \mathbb{R}^m)$  where  $W^{(1,p)}$  is the Sobolev space for  $p = \infty$  and  $L^\infty$  is the Lebesgue  $L^p$  space for  $p = \infty$ . The authors define the pair  $(\hat{x}, \hat{u})$  to be locally optimal or a weak minimum for the delayed optimal control problem if

$$J(\hat{x}, \hat{u}) \leq J(x, u) \quad (1.26)$$

holds for admissible  $(x, u)$  in a neighborhood of  $(\hat{x}, \hat{u})$  where  $\|x(t) - \hat{x}(t)\|, \|u(t) - \hat{u}(t)\| < \epsilon$  for small enough  $\epsilon > 0$ ,  $t \in [a, b]$ . In order to extend their the application of Pontryagin's minimum principle to their problem with mixed constraints, the authors need to assume a regularity condition. They say, for a locally optimal pair  $(\hat{x}, \hat{u})$  and  $t \in [a, b]$ , let  $J_0(t) := \{j \in \{1, \dots, p\} | C_j(t, \hat{x}(t), \hat{u}(t)) = 0\}$  be the

set of active indices for the inequality constraint in Eq. 1.25e. Then the authors assume the rank condition

$$\text{rank} \left( \frac{\partial C_j(t, \hat{x}(t), \hat{u}(t))}{\partial u} \right)_{j \in J_0(t)} = \#J_0(t). \quad (1.27)$$

The authors further assume that for delays  $\tau, s \geq 0$ ,  $(\tau, s) \neq (0, 0)$ , the ratio  $s/\tau \in \mathbb{Q}$  with  $\tau > 0$  or  $\tau/s \in \mathbb{Q}$  with  $s > 0$  where  $\mathbb{Q}$  is the set of rational numbers. The authors then replace the terminal condition in Eq. 1.25d with the mixed boundary condition  $w(x(a), x(b)) = 0$ . Define the Hamiltonian  $H$  and the augmented Hamiltonian  $\mathcal{H}$  as follows

$$H(t, x(t), y, u(t), v, \lambda) := L(t, x(t), y, u(t), v) + \lambda^T f(t, x(t), y, u(t), v) \quad (1.28)$$

$$\mathcal{H}(t, x(t), y, u(t), v, \lambda, \mu) := L(t, x(t), y, u(t), v) + \lambda^T f(t, x(t), y, u(t), v) + \mu^T C(t, x(t), u(t)), \quad (1.29)$$

where  $\lambda(t) \in \mathbb{R}^n$  and  $\mu(t) \in \mathbb{R}^p$  are multipliers,  $y(t) \in \mathbb{R}^n$  is the delayed state variable and  $v(t) \in \mathbb{R}^m$  is the delayed control variable [32].

To obtain the necessary conditions for the delayed OCP with mixed state and control constraints, the authors extend an approach originally developed by Guinn in 1976 [35] for an unconstrained problem with a delay in the state. The authors augment the delayed OCP to obtain an undelayed OCP with higher dimensionality, allowing them to use the necessary conditions for the undelayed problem [32, 35].

**Theorem 1.3.1.** *Let  $(\hat{x}, \hat{u})$  be locally optimal for Eq. 1.24 and Eq. 1.25 with delays satisfying the rationality assumption. Then there exist a costate (adjoint) function  $\hat{\lambda} \in W^{(1, \infty)}([a, b], \mathbb{R}^n)$ , a multiplier function  $\hat{\mu} \in L^{(1, \infty)}([a, b], \mathbb{R}^p)$ , and a multiplier  $\hat{v} \in \mathbb{R}^q$ , such that the following conditions hold for a.e.  $t \in [a, b]$ :*

1. *adjoint differential equation:*

$$\dot{\hat{\lambda}}^T = -\hat{\mathcal{H}}_x(t) - \chi_{[a, b-\tau]}(t) \hat{\mathcal{H}}_y(t + \tau), \quad (1.30)$$

which gives

$$\begin{aligned} \dot{\hat{\lambda}}^T = & -\mathcal{H}_x(t, \hat{x}(t), \hat{x}(t - \tau), \hat{u}(t), \hat{u}(t - s), \hat{\lambda}(t), \hat{\mu}(t)) \\ & - \chi_{[a, b-\tau]}(t) \mathcal{H}_y(t + \tau, \hat{x}(t + \tau), \hat{x}(t), \hat{u}(t + \tau), \hat{u}(t + \tau - s), \hat{\lambda}(t + \tau), \hat{\mu}(t + \tau)) \end{aligned} \quad (1.31)$$

where  $\hat{\mathcal{H}}_x(t)$  and  $\hat{\mathcal{H}}_y(t)$  denote the evaluation of the partial derivatives  $\mathcal{H}_x$  and  $\mathcal{H}_y$  along  $\hat{x}(t), \hat{x}(t - \tau), \hat{u}(t), \hat{u}(t - \tau), \hat{\lambda}(t), \hat{\mu}(t)$ ;

2. *transversality conditions:*

$$\hat{\lambda}(b)^T = g_x(\hat{x}(b)) + \hat{v}^T w_x(\hat{x}(b)) \quad (1.32)$$

3. *minimum condition for Hamiltonian:*

$$\begin{aligned}
& \hat{H}(T) + \chi_{[a,b-s]}(t)\hat{H}(t+s) = \\
& = H(t, \hat{x}(t), \hat{x}(t-\tau), \hat{u}(t), \hat{u}(t-s), \hat{\lambda}(t)) \\
& \quad + \chi_{[a,b-s]}(t)H(t+s, \hat{x}(t+s), \hat{x}(t+s-\tau), \hat{u}(t+s), \hat{u}(t), \hat{\lambda}(t+s)) \\
& \leq H(t, \hat{x}(t), \hat{x}(t-\tau)u, \hat{u}(t-s), \hat{\lambda}(t)) \\
& \quad + \chi_{[a,b-s]}(t)H(t+s, \hat{x}(t+s), \hat{x}(t+s-\tau), \hat{u}(t+s), u, \hat{\lambda}(t+s))
\end{aligned} \tag{1.33a}$$

for all  $u \in \mathbb{R}^m$  satisfying  $C(t, \hat{x}(t), u) \leq 0$ ;

4. *local minimum condition for augmented Hamiltonian:*

$$\hat{\mathcal{H}}_u + \chi_{[a,b-s]}(t)\hat{\mathcal{H}}_v(t+s) = 0 \tag{1.34}$$

5. *nonnegativity of multiplier and complementarity condition:*

$$\hat{\mu}(t) \geq 0 \text{ and } \hat{\mu}_i(t)C_i(t, \hat{x}(t), \hat{u}(t)) = 0, \quad i = 1, \dots, p. \tag{1.35}$$

A complete proof of Theorem 1.3.1 may be found in [32]. We give a derivation of the necessary conditions for an unconstrained optimal control problem with constant delay in the state in Chapter 4.

## 1.4 Existing Numerical Methods

Though mathematicians began working on optimal control theory in the early 20th century, the equations in their models were not solvable analytically. Mathematicians and engineers needed to develop numerical tools to solve their optimal control problems concurrently. The computational expense involved meant that solving many optimal control problems was not possible until the advent of digital computers after the end of World War II [18, 63].

Broadly, numerical methods for solving optimal control problems can be divided into *direct* and *indirect* methods. In an indirect method, one must first find the first order necessary conditions of the optimal control problem from the calculus of variations and Pontryagin's maximum (minimum) principle stated above. The necessary conditions are in the form of a multipoint boundary value problem, giving a system of nonlinear equations to be solved. Popular indirect methods include indirect shooting methods, which are vulnerable to sensitivities in the unknown initial conditions, and indirect collocation [67].

In this work, we will primarily consider direct methods of solving optimal control problems. In a direct method, sometimes referred to as a direct transcription method, one discretizes either the control or the state and control and transcribes the optimal control problem into the nonlinear programming problem (NLP) [78]

$$\min J(x) \tag{1.36}$$

subject to the constraints

$$c(x) = 0 \tag{1.37a}$$

$$z(x) \leq 0, \tag{1.37b}$$

$x \in \mathbb{R}^n$ ,  $c(x) \in \mathbb{R}^m$ , and  $z(x) \in \mathbb{R}^q$ . The first order optimality conditions are given by the Karush-Kuhn-Tucker (KKT) conditions [43, 76]

$$c_i(x) = 0, \quad i = 1, \dots, m \tag{1.38a}$$

$$z_i(x) \leq 0, \quad i = 1, \dots, q \tag{1.38b}$$

$$\mu_i \geq 0, \quad i = 1, \dots, q \tag{1.38c}$$

$$\mu_i z_i(x) = 0, \quad i = 1, \dots, q \tag{1.38d}$$

$$\nabla f(x) + \sum_{i=1}^m \lambda_i \nabla c_i(x) + \sum_{i=1}^q \mu_i \nabla z_i(x) = 0. \tag{1.38e}$$

Both direct and indirect methods require a differential equation solver to integrate the dynamics, but unlike an indirect method, direct methods utilize nonlinear optimization in order to find the optimal value of the control and minimize the cost subject to the dynamics and any other constraints. A key advantage of direct methods is they do not require the derivation of the necessary conditions, a process which can become extremely cumbersome for large or complex problems. Direct methods are able to avoid the sensitivity to changes in the initial conditions that result from the Hamiltonian nature of the necessary conditions [23].

There have been numerous papers written on numerical methods for optimal control problems with delays. However the focus of these papers is often on the theoretical foundations of the proposed method; numerical results are presented without the code that generated them, making the proposed methods difficult for the reader to implement [16, 24, 26, 45, 53, 54, 55, 62]. Given access to a delayed differential equation solver, one can sometimes parameterize the control and use a standard nonlinear optimizer to minimize the cost. This eliminates the need for a specialized software package, but in exchange one sacrifices the full functionality of a dedicated control solver. Optimal control packages are written to take advantage of the large, sparse NLPs which result from discretizing

the optimal control problem, greatly improving their speed of execution. These packages are also often equipped with powerful mesh refinement algorithms. GPOPS-II, for example, includes a mesh refinement algorithm which adjusts both the number of intervals and collocation points within each interval [59]. A user who implements their own control parameterization approach would need to write their own sparse solver and mesh refinement scheme.

Additionally, there are a limited number of commercially available optimal control solvers formulated to accept delays, including the Sparse Optimization Suite SOS [13], written in FORTRAN, or the open source PSOPT written in C++ [5]. In this work we seek to expand the application of the optimal control software GPOPS-II, written in MATLAB, by adapting the software to solve delayed optimal control problems. While GPOPS-II is not written to accept delay problems, we show it is possible to use GPOPS-II to solve certain types of delay problems. We present several different problem formulations, as well as the associated code snippets, to enable the reader to easily implement the results. We include a full list of code for each type of problem considered in the Appendix.

## CHAPTER

# 2

# DELAY DIFFERENTIAL EQUATIONS

In this chapter we give an overview of delay differential equations. We present existence and uniqueness results important for our work, as well as results on stability. We discuss existing solvers and methods for the numerical solution of delay differential equations, and introduce techniques for solving delay differential equations using GPOPS-II.

## 2.1 Basic Theory

We now give a brief overview of delay differential equations (DDEs). We include relevant existence results and stability properties for the work in this thesis. We discuss methods of obtaining an analytical solution to validate our work in later chapters as well as several useful numerical methods.

### 2.1.1 Existence Results

We begin with a result on the existence and uniqueness of a general, linear DDE with constant delay  $\tau$  given by Bellman and Cooke in [8]. We follow the notation and conventions established therein. Consider a DDE of the form

$$a x'(t) + b x(t) + c x(t - \tau) = f(t) \quad (2.1)$$

with initial condition

$$x(t) = \phi(t) \quad (2.2)$$

on  $0 \leq t \leq \tau$ . Let  $C^k(t_1, t_2)$  be the set of all functions with  $k$  continuous derivatives on  $(t_1, t_2)$ . The set  $C^k[t_1, t_2)$  contains functions  $f \in C^k(t_1, t_2)$  which have continuous  $k^{th}$  right-hand derivative at  $t_1$ . Similarly the set  $C^k(t_1, t_2]$  contains those functions  $f \in C^k(t_1, t_2)$  which have continuous  $k^{th}$  left-hand derivative at  $t_2$ . If a function  $f$  is in both  $C^k[t_1, t_2)$  and  $C^k(t_1, t_2]$ , then  $f \in C^k[t_1, t_2]$ . Then we have the following result from [8]:

**Theorem 2.1.1.** *Let  $f \in C^1[0, \infty)$  and  $\phi \in C^0[0, \tau]$ . Then there exists exactly one continuous function for  $t \geq 0$  which satisfies condition Eq. 2.2 and satisfies Eq. 2.1 for  $t > \tau$ . Furthermore, this function  $x$  is  $C^1(\tau, \infty)$  and  $C^2(2\tau, \infty)$ . If  $\phi \in C^1[0, \tau]$ , then  $x'$  is continuous at  $\tau$  if and only if*

$$a\phi'(\tau-0) + b\phi(\tau) + c\phi(0) = f(\tau). \quad (2.3)$$

We have  $x''$  continuous at  $2\tau$  if  $\phi \in C^2[0, \tau]$  and either Eq. 2.3 holds or  $c = 0$ . [8]

Though we note in the introduction that one may simply choose a  $\phi$  integrable instead of  $\phi$  continuous, one is no longer guaranteed the existence of a reasonably smooth, unique solution to the DDE. In the next section, we discuss a technique for solving a DDE called the *method of steps*, which can be used to prove the existence of a solution by construction. For those problems, we need not require  $\phi(t)$  be continuous at 0 in order to guarantee the existence and uniqueness of a solution.

The result in Theorem 2.1.1 is applicable to linear DDEs. We now present a more general result by Gorecki, Fuksa, Grabowski, and Korytowski from [34] on the continuous dependence of solutions; we utilize the conventions and notation established by the authors. Let  $C([a, b], \mathbb{R}^n)$  be the space of continuous functions from  $[a, b]$  into  $\mathbb{R}^n$  with norm  $|f| = \sup_{a \leq t \leq b} |f(t)|$ , and let  $h \geq 0, h \in \mathbb{R}$ . Consider the following DDE

$$\dot{x} = f(x_t, t) \quad (2.4)$$

on  $D \times [t_0, t_1]$  where  $D \subseteq C([-h, 0], \mathbb{R}^n)$ ,  $D$  open. The term  $x_t$  is the delay argument and  $f : D \times [t_0, t_1] \rightarrow \mathbb{R}^n$ , is a given function,  $f$  not necessarily linear. Let  $x_t = x(t + s)$ ,  $s \in [-h, 0]$ , and  $x_t \in C([-h, 0], \mathbb{R}^n)$  for  $x \in C([t_0 - h, t_1], \mathbb{R}^n)$ ,  $t_1 > t_0$ . The function  $x : [t_0 - h, t_1] \rightarrow \mathbb{R}^n$  is a solution to Eq. 2.4 if  $x \in C([t_0 - h, t_1], \mathbb{R}^n)$  and if Eq. 2.4 holds for almost all  $t \in [t_0, t_1]$ . The authors have the initial condition  $x_{t_0} = \phi$ ,  $\phi \in C([-h, 0], \mathbb{R}^n)$ .

The authors further require  $f$  satisfy the Carathéodory condition. Let  $f : \Omega \rightarrow \mathbb{R}^n$ , where  $\Omega \subseteq C([-h, 0], \mathbb{R}^n) \times \mathbb{R}$ ,  $\Omega$  open. Suppose  $f(\phi, \cdot)$  is measurable for every fixed  $\phi$ , and  $f(\cdot, t)$  is continuous for every fixed  $t$ . Then, if for every point  $(\phi, t) \in \Omega$ , there is a Lebesgue integrable function  $m$  and a neighborhood  $V(\phi, t)$  such that

$$|f(\psi, s)| \leq m(s), \quad (\psi, s) \in V(\phi, t), \quad (2.5)$$

then  $f$  satisfies the Carathéodory condition. We have now given sufficient background to present

the following theorem on continuous dependence from [34].

**Theorem 2.1.2.** *Let  $\Omega \subseteq C([-h, 0], \mathbb{R}^n) \times \mathbb{R}$ ,  $\Omega$  open. Let  $x^0$  be a solution of Eq. 2.4 with right hand side  $f^0 \in C(\Omega, \mathbb{R}^n)$ ,  $f$  passing through  $(\phi^0, t_0^0) \in \Omega$ . Assume  $x^0$  exists and is unique in  $[t_0^0 - h, b]$ ,  $b > t_0^0$ . Define a set*

$$W^0 = \{(x_t^0, t) : t \in [t_0^0, b]\} \quad (2.6)$$

*where  $V^0$  is a neighborhood of  $W^0$ , and let  $f$  be bounded in  $V^0$ . Supposed there is an arbitrary sequence  $(\phi^k, f^k, t_0^k)$ ,  $k = 1, 2, \dots$  where  $\phi^k \rightarrow \phi^0$ ,  $|f^k - f^0| \rightarrow 0$ , and  $t_0^k \rightarrow t_0^0$  as  $k \rightarrow \infty$ . Then there exists a  $k^0$  such that for all  $k \geq k^0$  there is a solution  $x^k$  of Eq. 2.4 in  $[t_0^k, b]$  which passes through  $(\phi^k, t_0^k)$  with right hand side  $f^k$ . Additionally, for every  $\epsilon > 0$ , there exists a  $k_1(\epsilon)$  such that  $x^k(t)$ ,  $k \geq k_1(\epsilon)$ , is determined on  $[t_0^0 - h + \epsilon, b]$  and  $x^k \rightarrow x^0$  uniformly on  $[t_0^0 - h + \epsilon, b]$ . [34]*

In our work, we primarily consider linear delayed optimal control problems. We now present an existence result from [34] for linear systems of delayed differential equations of the form

$$\dot{x} + \sum_{i=0}^N [A_i x(t - h_i) + B_i u(t - h_i)] + \int_{-h}^0 A(t, \tau) x(t + \tau) d\tau + \int_{-h}^0 B(t, \tau) u(t + \tau) d\tau = f(t), t \geq t_0 \quad (2.7)$$

with a control  $u$ . We note Eq. 2.7 includes a delay in the control; while we restrict much of our discussion to systems with delays in the state alone, we include the result for the more general case in Eq. 2.7 and we consider control delays in Chapter 6. We have the initial conditions

$$x(t) = \phi(t), \quad t \in [t_0 - h, t_0] \quad (2.8a)$$

$$\dot{x}(t) = \mu(t), \quad t \in (t_0 - h, t_0) \quad (2.8b)$$

$$u(t) = \nu(t), \quad t \in [t_0 - h, t_0] \quad (2.8c)$$

where  $\phi, \mu$ , and  $\nu$  are all given and square integrable on  $(t_0 - h, t_0)$ .

Given that we consistently work on finite real intervals, we may additionally assume  $f$  and  $u$  are square integrable on every  $[t_0, t]$ ,  $t > t_0$ . We allow  $A, B, A_i, B_i$  may be linear time-varying, and assume they are bounded and integrable on every finite subset of their domain. The authors note there is no requirement for continuity of the initial condition  $\phi$ , and therefore  $x$  is only required to be continuous for  $t > t_0$ , and continuous from the right at  $t_0$ . The authors use the Cartesian product

$$H = \mathbb{R}^n \times L^2(-h, 0; \mathbb{R}^n) \times L^2(-h, 0; \mathbb{R}^n) \times L^2(-h, 0; \mathbb{R}^m) \quad (2.9)$$

as their space of initial data where  $L^2$  is the Lebesgue  $L^p$  space with  $p = 2$ . Then we have the following result on existence from [34].

**Theorem 2.1.3.** *For all  $t_1, s, t_1 \geq s \geq t_0$ , every  $u \in L^2([s, t_1], \mathbb{R}^m)$ , every  $f \in L^2([s, t_1], \mathbb{R}^n)$ , and every initial condition  $\xi \in H$  there is a unique solution  $x(\cdot, \xi, u, f)$  of Eq. 2.7 in  $[s - h, t_1]$ . Additionally, the*

function  $(\xi, u, f) \mapsto x(\cdot, \xi, u, f)|_{[s, t_1]} \in C(s, t_1)$  is both linear and continuous. The solution to Eq. 2.7 is of the form:

$$x(t, \xi, u, f) = \Phi(t, s)x(s) - \int_{-h}^0 \Phi^1(t, s, \tau)x(t + \tau)d\tau - \int_{-h}^0 \Phi^2(t, s, \tau)u(t + \tau)d\tau - \int_s^t \Phi(t, \tau)[v(\tau) - f(\tau)]d\tau, \quad t \geq s, \quad (2.10)$$

where

$$v(t) = \sum_{i=0}^N B_i \begin{cases} u(t - h_i) & t - h_i \geq s \\ 0 & \text{else} \end{cases} \quad (2.11a)$$

$$+ \int_{-h}^0 B(t, \tau) \begin{cases} u(t + \tau) & t + \tau \geq s \\ 0 & \text{else} \end{cases} d\tau. \quad (2.11b)$$

Here,  $\Phi$  is the fundamental solution matrix of Eq. 2.7 and is defined by

$$\frac{\partial}{\partial t} \Phi(t, s) + \sum_{i=0}^N A_i \Phi(t - h_i, s) + \int_{-h}^0 A(t, \tau) \Phi(t + \tau, s) d\tau = 0, \quad t > s \quad (2.12)$$

with  $\Phi(s, s) = I$  and  $\Phi(t, s) = 0$  for  $t < s$ . We also have

$$\Phi^1(t, s, \tau) = \sum_{i=0}^N \begin{cases} \Phi(t, s + \tau + h_i) A_i(s + \tau + h_i) & \tau = s - t < -h_i \leq \tau \\ 0 & \text{else} \end{cases} \quad (2.13a)$$

$$+ \int_{-h}^{\tau} \Phi(t, s + \tau - \theta) A(s + \tau - \theta, \theta) d\theta \quad (2.13b)$$

and  $\Phi^2$  is defined similarly to  $\Phi^1$  with  $B_i$  instead of  $A_i$  and  $B$  instead of  $A$ .

### 2.1.2 Analytic Solution Via Method of Steps

Having presented key existence results for DDEs, we now turn to obtaining an analytic solution. The *method of steps* is a technique to obtain an analytic solution of a DDE by reducing it to an ordinary differential equation (ODE) on time intervals determined by the length of the delay. The technique was originally developed by Bellman in [7] as a way to alleviate computational storage problems when solving DDEs numerically. The method is simple to implement but becomes cumbersome when the delay is short relative to the time interval of interest due to the number of equations one would need to solve. While the method of steps is applicable to systems with multiple discrete,

constant delays it suffices to consider the case where only one delay is present.

Consider a DDE of the form

$$\dot{x} = f(t, x(t), x(t-\tau)), \quad 0 < t \leq T \quad (2.14a)$$

$$x(t) = \phi(t), \quad t \in [-\tau, 0] \quad (2.14b)$$

with constant delay  $\tau$ , prehistory function  $\phi(t)$ , and  $f$  not necessarily linear. We can rewrite Eq. 2.41 as an ODE system on  $[0, \tau]$  in the following manner:

$$\dot{x}_1(t) = f(t, x_1(t), \phi(t), u_1(t)), \quad i = 1, \quad t \in [0, \tau] \quad (2.15a)$$

$$\dot{x}_i(t) = f(t, x_i(t), x_{i-1}(t), u_i(t)), \quad i = 2, \dots, N, \quad t \in [0, \tau] \quad (2.15b)$$

$$x_1(0) = \phi(0), \quad i = 1 \quad (2.15c)$$

$$x_i(0) = x_{i-1}(\tau), \quad i = 2, \dots, N \quad (2.15d)$$

for  $N$  number of steps in the interval  $[0, T]$ .

We can see that as the length of the interval increases, or the number of delays increases, the size of the problem increases as well. Though the problem can get quite large, it can now be solved by any ODE integrator. Bellen and Zennaro note that, for any finite number of steps, a numerical method of global order  $p$  will perform on the process to order  $p$  [6]. For  $\phi(t)$  continuous, the MOS gives a unique solution that is globally-defined in forward time. This is because any initial discontinuity at  $t_0$  is propagated through the solution at successively higher derivatives; the solution becomes smoother on subsequent time intervals [56].

## 2.2 Formulation as a Partial Differential Equation

The method of steps allows one to reformulate a DDE as system of ODEs without a delay. We now discuss reformulating a DDE as a system of partial differential equations (PDEs), which allows one to use a broader range of numerical techniques to solve the resulting problem.

### 2.2.1 Reformulation of a Delay Differential Equation as a System of Partial Differential Equations

Consider the DDE given in the previous section,

$$\dot{x} = f(t, x(t), x(t-\tau)), \quad t \geq 0 \quad (2.16a)$$

$$x(t) = \phi(t), \quad t \in [-\tau, 0]. \quad (2.16b)$$

Let  $\phi \in C^1([-\tau, 0], \mathbb{R}^n)$  and  $f$  be globally Lipschitz on  $[0, \infty) \times \mathbb{R}^n \times \mathbb{R}^n$  and satisfy the requirements of Theorem 2.1.2. Then Eq. 2.16 has a unique solution  $x(t) \in C^1([-\tau, \infty), \mathbb{R}^n)$ . Let

$$u(t, \theta) = x(t + \theta), \quad \theta \in [-\tau, 0], \quad t \geq 0. \quad (2.17)$$

Then  $u(t, \theta)$  is the solution  $x$  on  $[t - \tau, t]$  and satisfies the PDE boundary value problem

$$\frac{\partial u}{\partial t} = \frac{\partial u}{\partial \theta}, \quad \theta \in [-\tau, 0], \quad t \geq 0 \quad (2.18a)$$

$$u(0, \theta) = \phi(\theta), \quad \theta \in [-\tau, 0] \quad (2.18b)$$

$$\frac{\partial u}{\partial t}(t, 0) = f(t, u(t, 0), u(t, -\tau)), \quad t \geq 0. \quad (2.18c)$$

We have rewritten our delay equation in Eq. 2.16 as a PDE system which we can solve with any suitable numerical technique.

### 2.2.2 Method of Lines

In the preceding section we demonstrate how a DDE can be rewritten as a PDE. We now show how the resulting PDE can be solved numerically using the method of lines (MOL). The method of lines relies on partially discretizing the PDE, generally in the state direction, in order to rewrite the PDE as a system of ODEs. In this section, we use MOL to reformulate a delayed optimal control problem. We first rewrite the delayed system as a PDE in terms of time,  $t$ , and our delayed state,  $\theta$ . We then approximate this PDE using the method of lines. This eliminates the delay from the equations, allowing one to use a greater variety of optimal control solvers.

Say we have the system

$$\dot{x} = Ax + Cx(t - \tau) + Bu \quad (2.19a)$$

$$x(t) = \phi(t), \quad t \in [-\tau, 0] \quad (2.19b)$$

$$J = \int_0^T (x - r)^T Q(x - r) + u^T R u \, dt \quad (2.19c)$$

$$J(0) = (0) \quad (2.19d)$$

with delay  $\tau$ . We assume  $A, B, C$  are linear time-varying, with  $Q \geq 0$  and  $R > 0$ . We first reformulate Eq. 2.19 as a PDE with respect to time and a delayed variable,  $\theta$ . Let  $U(t, \theta) = x(t + \theta), -\tau \leq \theta \leq$

$0, t \geq 0$ . Then we have

$$U_t = U_\theta, \quad -\tau \leq \theta < 0, \quad t \geq 0 \quad (2.20a)$$

$$U(0, \theta) = \phi(\theta), \quad -\tau \leq \theta \leq 0 \quad (2.20b)$$

$$U_t(t, 0) = AU(t, 0) + CU(t, -\tau) + Bu(t), \quad t \geq 0 \quad (2.20c)$$

$$J_t(t, 0) = (U(t, 0) - r)^T Q(U(t, 0) - r) + u^T Ru, \quad t \geq 0, \quad (2.20d)$$

where  $\phi(\theta)$  is the prehistory.

Now we have an initial boundary value problem and can use MOL with a centered difference to compute the partial derivatives with respect to the delay variable  $\theta$  and step size  $h$  to rewrite Eq. 2.20 as a system of ODEs in  $\theta$ . Let  $U(t, \theta_k) = U_k(t)$ ,  $k = 0 \dots N$ . Then we have:

$$U_{k+1}(t) = U_k(t) + hU'_k(t) + (h^2/2)U''_k(t) + (h^3/6)U'''_k(\xi_1) \quad (2.21a)$$

$$U_{k-1}(t) = U_k(t) - hU'_k(t) + (h^2/2)U''_k(t) - (h^3/6)U'''_k(\xi_2). \quad (2.21b)$$

Subtracting Eq. 2.21b from Eq. 2.21a we obtain

$$U'_k(t) = \frac{1}{2h} (U_{k+1}(t) - U_{k-1}(t)) - \frac{h^2}{6} U'''_k(\xi), \quad (2.22)$$

a second order approximation to the solution on the interior grid points. Then our system of ODEs is

$$U'_k = 1/2h (U_{k+1}(t) - U_{k-1}(t)), \quad k = 1 \dots N \quad (2.23a)$$

$$U'_N = AU_N(t) + CU_0 + Bu(t), \quad k = N \quad (2.23b)$$

$$J'_N = (U_N - r)^T Q(U_N - r) + u^T Ru, \quad k = N. \quad (2.23c)$$

We require a boundary condition along the line  $U(t, -\tau)$ , so an equation for  $U'_0(t)$ . We begin by taking a Taylor expansion about the points  $U(t + 2h)$  and  $U(t + h)$ . Then we have

$$U(t + 2h) = U(t) + 2hU'(t) + (4h^2/2)U''(t) + (8h^3/6)U'''(\xi_1) \quad (2.24a)$$

$$U(t + h) = U(t) + hU'(t) + (h^2/2)U''(t) + (h^3/6)U'''(\xi_2). \quad (2.24b)$$

Multiplying Eq. 2.24b by four and subtracting from Eq. 2.24a, we obtain

$$U(t + 2h) - 4U(t + h) = -3U(t) - 2hU'(t) + (2h^3/3)(U'''(\xi_1) - U'''(\xi_2)) \quad (2.25)$$

or

$$U'(t) = 1/2h (-U(t + 2h) + 4U(t + h) - 3U(t)) + (2h^3/3)U'''(\xi). \quad (2.26)$$

Thus our boundary condition becomes

$$U_0'(t) = 1/2h(-U_2(t) + 4U_1(t) - 3U_0(t)) + (2h^3/3)U'''(\xi) \quad (2.27)$$

and we have the system

$$U_0'(t) = 1/2h(-U_2(t) + 4U_1(t) - 3U_0(t)), \quad k = 0 \quad (2.28a)$$

$$U_k' = 1/2h(U_{k+1}(t) - U_{k-1}(t)), \quad k = 1 \dots N \quad (2.28b)$$

$$U_N' = AU_N(t) + CU_0 + Bu(t), \quad k = N \quad (2.28c)$$

$$J_N' = (U_N - r)^T Q(U_N - r) + u^T Ru, \quad k = N \quad (2.28d)$$

using a second order central difference formula for the interior points and a third order approximation for our boundary condition.

We implement the MOL using GPOPS-II for a specific example in Chapter 5, and demonstrate we are able to obtain an accurate solution for the case  $N = 5$ . Our results are remarkable considering the formula on the interior is only second order accurate and we use a low-dimensional approximation. A key fact to note is, for constant step size, our  $h = \tau/N$ . Then for a modest delay of  $\tau = 1$ , we have that our error is  $\mathcal{O}(.2^2)$  on the interior and  $\mathcal{O}(.2^3)$  on the boundary.

## 2.3 Eigenvalues

When using the method of lines, or method of steps, to solve a delay differential equation the stability of the resulting problem can affect the user's ability to obtain a numerical solution. In this section we discuss the eigenvalues of a DDE and their affect on the system's stability. Consider a linear delayed differential equation of the form

$$\dot{x} = A_0 x(t) + \sum_{i=1}^N A_i x(t - \tau_i). \quad (2.29)$$

The characteristic equation of Eq. 2.29 is

$$\det(\Delta(\lambda)) = 0, \quad (2.30)$$

where  $\Delta(\lambda)$  is the characteristic matrix of Eq. 2.29 with

$$\Delta(\lambda) = \lambda I - \left( A_0 + \sum_{i=1}^N A_i e^{-\lambda \tau_i} \right), \quad (2.31)$$

where we have substituted in  $x = y e^{-\lambda \tau_i}$ . Here  $y$  is a nonzero vector,  $\det(\Delta(\lambda))$  is the characteristic function of Eq. 2.29, and  $\lambda$  is a root of the characteristic function, so  $x = y e^{-\lambda \tau_i}$  is a solution to the DDE. The roots of Eq. 2.30 are the characteristic roots of the differential equation.

We now introduce several important definitions of stability from Michiels and Niculescu in [56]. We begin with the following important notion of asymptotic stability of the homogeneous solution to Eq. 2.29:

**Definition 1.** *The homogenous solution of Eq. 2.29 is asymptotically stable iff for all  $\epsilon > 0$  there exists a  $\delta > 0$  such that for all  $\phi \in C([- \tau_m, 0], \mathbb{R}^n)$ ,  $\|\phi\| < \delta \implies$  for all  $t \geq 0$   $\|x(t; \phi)\| < \epsilon$ , for all  $\phi \in C([- \tau_m, 0], \mathbb{R}^n)$   $\lim_{t \rightarrow +\infty} x(t; \phi) = 0$  [56].*

Here,  $\|\cdot\|$  is the supremum norm and  $x(t; \phi)$  denotes solution  $x$  with initial condition  $\phi$ . We note that for a linear DDE of the form in Eq. 2.29, asymptotic stability is equivalent to exponential stability. The following definition of exponential stability is from [56].

**Definition 2.** *The homogeneous solution  $x$  with initial condition  $\phi \in C([- \tau_m, 0], \mathbb{R}^n)$  is exponentially stable iff there are constants  $C > 0$ ,  $k > 0$  such that  $\|x(t; \phi)\| \leq C e^{-kt} \|\phi\|$ . [56]*

We note the following important key fact about the characteristic roots of Eq. 2.29 from [56]:

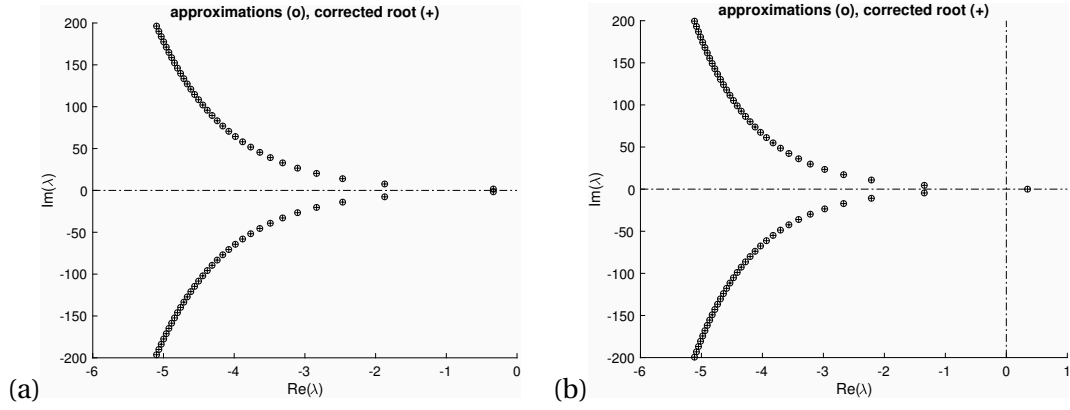
**Proposition 1.** *The homogenous solution of Eq. 2.29 is asymptotically stable iff all characteristic roots have negative real part; i.e. all characteristic roots of Eq. 2.29 are in the open left half plane.*

Now consider the scalar, linear DDE

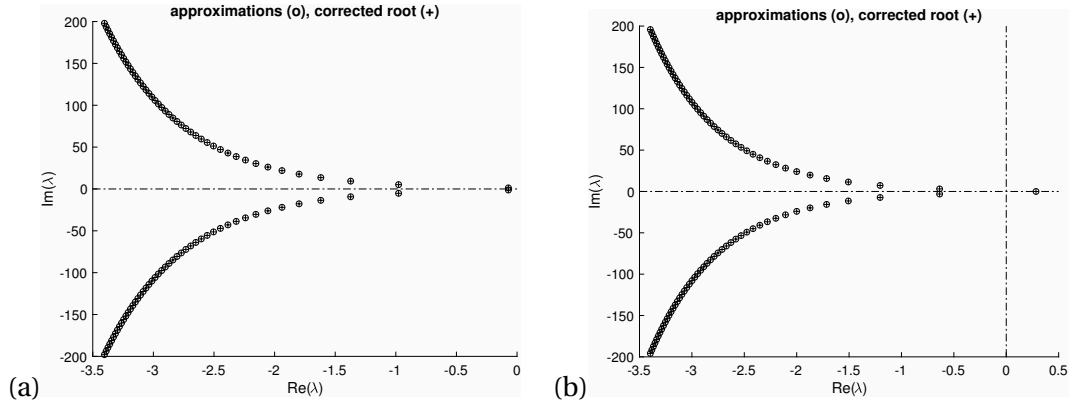
$$\dot{x} = ax + cx(t - \tau), \quad (2.32)$$

which we use for several computational examples in this work. The eigenvalues, and hence stability of the problem, will depend on the values of parameters  $a$  and  $c$  as well as the value of the delay  $\tau$ . We utilize the MATLAB `roots` package written by Wu and Michiels [82] to compute and plot the characteristic roots of Eq. 2.32 for several different values of  $c$  and  $\tau$ . The package allows the user to find the roots in a given right half-plane,  $Re(\lambda) > r$ , or to specify a rectangular region. We chose the region  $-20 < Re(\lambda) < 20$  and  $-200 < Im(\lambda) < 200$ .

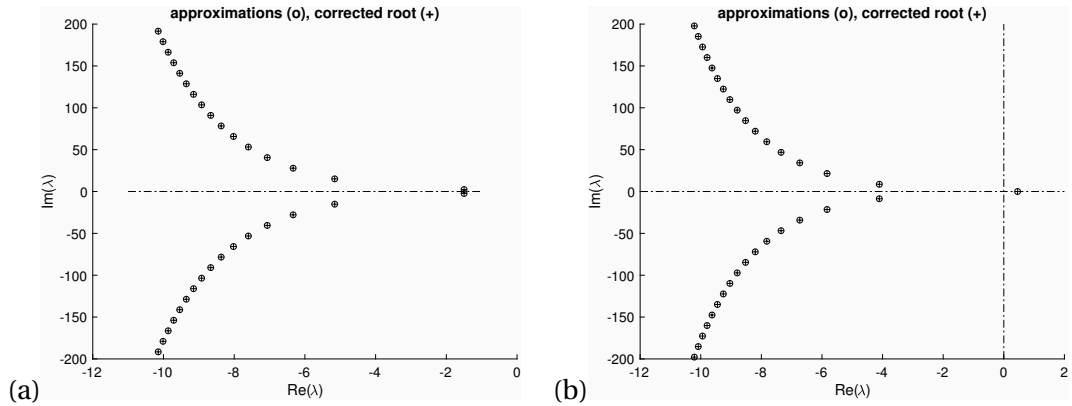
We first take  $a = -.5$ ,  $c = -1.2$ , and  $\tau = 1$ . We can see from Fig. 2.3(a) that our characteristic roots all have negative real part, indicating an asymptotically stable solution. We therefore may reasonably expect a decent numerical approximation to the solution. Fig. 2.3(b) demonstrates changing the sign on the delay term to  $c = +1.2$  introduces a small, positive characteristic root with  $Im(\lambda) = 0$ .



**Figure 2.1** (a) Eigenvalues of Eq. 2.32 with  $\tau = 1$ ,  $a = -0.5$ ,  $c = -1.2$  and (b)  $c = +1.2$ .



**Figure 2.2** (a) Eigenvalues of Eq. 2.32 with  $\tau = 1.5$ ,  $a = -0.5$ ,  $c = -1.2$  and (b)  $c = +1.2$ .



**Figure 2.3** (a) Eigenvalues of Eq. 2.32 with  $\tau = .05$ ,  $a = -0.5$ ,  $c = -1.2$  and (b)  $c = +1.2$ .

## 2.4 Existing Solvers and Numerical Methods

There are a variety of available numerical tools for solving DDEs. As we discuss in the preceding sections, one can utilize the method of steps or method of lines with any standard ODE or BVP solver. Another technique we implement is a method called *waveform relaxation*, which we discuss later in this section. Runge-Kutta methods have been implemented with success in various specialized DDE solvers including the MATLAB solver dde23 [72] and the FORTRAN-based DKL6 [21]. MATLAB has two other DDE solvers: ddesd and ddensd [70, 73]. For our purposes, we restrict our software discussions to the two main MATLAB solvers we utilized: bvp4c, and dde23 [46, 71].

### 2.4.1 MATLAB Delay Differential Equation Solver

We utilize the MATLAB DDE solver dde23 developed by Shampine and Thompson in [72] to solve our delayed dynamics in our control parameterization code and as the "true" solution to test GPOPS-II as a DDE integrator later in this work. The syntax of a function call to dde23 is as follows

```
sol = dde23(ddefun,lags,history,tspan),
```

where the output `sol` is a structure containing the mesh `sol.x`, the solution `sol.y` evaluated at the mesh points, and `sol.yp`, the derivative of the solution evaluated at the mesh points. The function `ddefun` contains the dynamics, `lags` is the vector of constant delay values, `history` is the prehistory function, and `tspan` is the time interval. The user can specify additional options such as tolerances and step size using the `options` argument and the function `ddeset`.

The solver utilizes the Runge-Kutte triple BS(3,2) developed by Bogacki and Shampine in [66]. We now present a general discussion of the dde23 solver strategy as originally given in [72], and follow the authors' notational conventions. We begin by defining the general formulas used in the Runge-Kutta triple. Let  $y_n$  be the approximation to  $y(x)$  at the mesh point  $x_n$ . Then the next approximation to the solution  $y_{n+1}$  is defined as

$$y_{n+1} = y_n + h_n \sum_{i=1}^s b_i f_{ni}, \quad (2.33)$$

where  $h_n$  is the step size and  $i$  is the index of stage  $f_{ni} = f(x_{ni}, y_{ni})$  at mesh point  $x_{ni} = x_n + c_i h_n$  and

$$y_{ni} = y_n + h_n \sum_{j=1}^{i-1} a_{ij} f_{nj}. \quad (2.34)$$

Then

$$y(x_{n+1}) = y(x_n) + h_n \Phi(x_n, y(x_n)) + e_n \quad (2.35)$$

where  $\Phi(x_n, y_n) = \sum_{i=1}^s b_i f_{ni}$  and  $e_n$  is the local truncation error which is  $O(h_n^{p+1})$  for  $f, y(x)$  sufficiently smooth. The triple uses a second, lower order formula for selecting the step size

$$y_{n+1}^* = y_n + h_n \sum_{i=1}^s b_i^* f_{ni} = y_n + h_n \Phi^*(x_n, y_n) \quad (2.36)$$

with local truncation error  $e_n^*$  that is  $O(h_n^p)$ . The third formula is

$$y_{n+\sigma} = y_n + h_n \sum_{i=1}^s b_i(\sigma) f_{ni}, \quad (2.37)$$

where the  $b_i(\sigma)$  are coefficient polynomials in  $\sigma$ . This gives a polynomial approximation to the value of the solution  $y(x_n + \sigma h_n)$  for  $\sigma \in [0, 1]$ . For the BS(2,3) triple the following two assumptions hold: first, that  $\sigma = 0$  gives  $y_n$  and  $\sigma = 1$  gives  $y_{n+1}$  and second, that the order of Eq. 2.37 is the same as Eq. 2.35. Then Eq. 2.37 is referred to as a *continuous extension* of Eq. 2.35, and Eq. 2.35 is considered the  $\sigma = 1$  case of Eq. 2.37. The authors define the local truncation error of the continuous extension as

$$y(x_n + \sigma h_n) = y(x_n) + h_n \Phi(x_n, y(x_n), \sigma) + e_n(\sigma) \quad (2.38)$$

where they assume for smooth  $y(x), f$  that  $\|e_n(\sigma)\| \leq C_1 h_n^{p+1}$  for some constant  $C_1$ .

For a DDE with linear, constant delays the authors have

$$f_{ni} = f(x_{ni}, y_{ni}, y(x_{ni} - \tau_1), \dots, y(x_{ni} - \tau_k)). \quad (2.39)$$

Consider the approximation  $S(x)$  to  $y(x)$  for all  $x \leq x_n$ . Then for  $h_n \leq \tau = \min(\tau_1, \dots, \tau_k)$  one has  $x_{ni} - \tau_j < x_n$  and

$$f_{ni} = f(x_{ni}, y_{ni}, S(x_{ni} - \tau_1), \dots, S(x_{ni} - \tau_k)) \quad (2.40)$$

is an explicit method for determining the stage and formulas Eq. 2.35, Eq. 2.36, and Eq. 2.37 are explicit as well. In this case the function  $S(x)$  is the prehistory. To obtain the value of the approximation at the next step,  $y_{n+\sigma}$ , the authors use the continuous extension to define  $S(x_n + \sigma h_n) = y_{n+\sigma}$  on  $[x_n, x_{n+1}]$ .

For the case  $h_n > \tau_j$ , for some  $j$ , the formulas for the triple are now implicit and the history function  $S(x)$  is evaluated in the span of the current step. The implicit formulas are evaluated using iteration. In dde23, the function  $S(x)$  is defined up to the current step  $x_n$ . The authors extend the definition of  $S(x)$  to  $(x_n, x_n + h_n]$  by calling the new function  $S^{(0)} = y_0$  for the first step, because the solution is discontinuous at the left endpoint of the interval,  $a$ . The prediction for  $S^{(0)}$  at the next step is the continuous extension of the previous step. The algorithms for adjusting step size in dde23 are taken from the MATLAB ordinary differential equation solver ode23 [71] and modified to accept implicit formulas. For results on convergence and error control, see [72].

The algorithm requires a choice of step size such that the discontinuities are mesh points, so none of the delay terms will have a low-order discontinuity in the span of the current step. This ensures that the Runge-Kutta method used will have the expected order to estimate the error. Only the smoothness of  $f$  need be considered because the authors utilize one-step methods. For more serious discontinuities, dde23 will restart the integration.

#### 2.4.2 Method of Steps Formulation with MATLAB Boundary Value Problem Solver

As we explain earlier in the chapter, a key advantage of the MOS formulation is the ability to use any BVP solver to find the solution to a DDE. In this work, we utilize the MOS with the MATLAB BVP solver bvp4c. We consider the following delay problem

$$\dot{x} = -x + 2x(t-1), \quad t \in [0, 4] \quad (2.41a)$$

$$x(t) = \sin(t), \quad t \in [-1, 0] \quad (2.41b)$$

with delay  $\tau = 1$  and prehistory  $\phi(t) = \sin(t)$ . Using the MOS we rewrite Eq. 2.41 as a system of ODES. We have :

$$\dot{x}_1(t) = -x_1(t) + 2\sin(t-1) + t, \quad x_1(0) = 0, \quad t \in [0, 1] \quad (2.42a)$$

$$\dot{x}_i(t) = -x_i(t) + 2x_{i-1}(t) + t, \quad x_i(0) = x_{i-1}(1), \quad i = 2, \dots, 4. \quad (2.42b)$$

We define a new ODE for each  $i$ ,  $i = 1, \dots, 4$ . The delay term in Eq. 2.42 is given by the prehistory  $\sin(t-1)$  for the first equation in  $x_1$  and in the following  $x_i$  equations the delay term is replaced by the solution on the previous interval. We treat Eq. 2.42 as a BVP where the righthand boundary condition for  $x_{i-1}$  is the lefthand boundary condition for  $x_i$ , in order to enforce continuity across the time intervals. We then solve Eq. 2.42 with bvp4c using the following code

```
solinit = bvpinit(linspace(0,1),[0,0,0,0]); %initial conditions
sol = bvp4c(@defun, @debc, solinit) %solve using bvp4c
```

```
function res = debc(ya,yb) % boundary conditions
res = [ya(1);
       ya(2)-yb(1);
       ya(3)-yb(2);
       ya(4) - yb(3)];
end
```

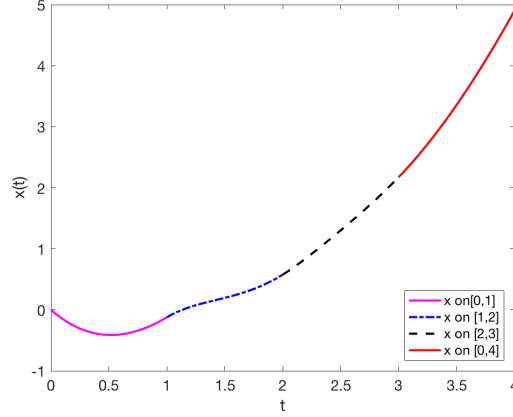
```
function dydt = defun(t,y)
dydt= [-y(1) + 2*sin(t-1) + t; %dynamics
```

```

-y(2) + 2*y(1) + (t+1);
-y(3) + 2*y(2) + (t+2);
-y(4) + 2*y(3) + (t+3) ];
end

```

Our solution plot in Fig. 2.4 demonstrates how one needs to translate the solutions onto the different intervals in order to construct the solution on the entire time interval.



**Figure 2.4** Solution of Eq. 2.41 using MOS.

### 2.4.3 Waveform Relaxation

In addition to the method of steps and dde23, another numerical technique we rely upon is waveform relaxation. Waveform relaxation is an iterative technique which can be used to solve DDEs. It was originally applied by Lelarsmee, Ruehli, and Sangiovanni-Vincentelli in [50] to the time domain analysis of systems of nonlinear differential equations modeling integrated circuits.

To apply waveform relaxation to a DDE, we begin with an initial estimate of the solution of the delayed equation. In subsequent iterations, we replace the delayed term at the current iterate with the value of the delayed term from the previous iterate. For example, consider the general DDE

$$\dot{x} = f(t, x(t), x(t-\tau)), \quad t \geq 0 \quad (2.43a)$$

$$x(t) = \phi(t), \quad t \in [-\tau, 0]. \quad (2.43b)$$

Then, given an initial estimate of the solution  $x_0(t)$ , we have the ODE

$$x'_{k+1} = f(t, x_{k+1}(t), x_k(t-\tau)), \quad k = 1 \dots N \quad (2.44)$$

with  $N$  iterates.

This approach is of particular interest to us because the method is no longer solving a DDE. In our work, we seek to utilize the software GPOPS-II to solve a delayed optimal control problem. Though designed to be an optimal control solver, GPOPS-II is a very effective ODE integrator. Before attempting to apply a waveform-type method to an optimal control problem, we test GPOPS-II on a linear DDE using waveform relaxation.

We consider problems of the form

$$\dot{x} = ax + cx(t - \tau) \quad (2.45a)$$

$$x(t) = \phi(t), \quad t \in [-\tau, 0]. \quad (2.45b)$$

Then we have the iteration

$$\dot{x}_{k+1} = f(x_{k+1}(t), x_k(\theta(t))) = ax_{k+1}(t) + cx_k(t - \tau), \quad x_{k+1}(0) = x_0 \quad (2.46)$$

where the delay argument is  $\theta(t) = t - \tau$ . In particular, Eq. 2.46 satisfies the assumptions (3.2)-(3.4) for superlinear convergence from [15] i.e. that  $f$  is Lipschitz continuous with respect to  $x_k(\theta(t))$ , satisfies a one-sided Lipschitz condition with respect to  $x_{k+1}(t)$ , and that  $\theta(t) \leq t$ ,  $\theta'(t) > 0$ . We may therefore expect the waveform relaxation method to converge for the types of problems we consider.

We note these conditions given in [15] do not guarantee the existence of a solution, only that the method will converge if such a solution exists. For our linear problem in Eq. 2.45 we are guaranteed a solution via Theorem 2.1.1. For detailed convergence analysis and error estimates, we refer the interested reader to [3, 4, 15, 83]. We compare the performance of GPOPS-II to using waveform relaxation with the MATLAB ODE integrator ode45, as well as the DDE integrator dde23, in Chapter 5 and see that we do indeed achieve rapid convergence.

## CHAPTER

# 3

# OPTIMAL CONTROL PROBLEMS WITHOUT DELAYS

In Chapter 2 we discuss the supporting theory and necessary background for DDEs. In this chapter, we review optimal control problems without delays, including numerical techniques, before turning to a discussion of delayed OCPs in Chapter 4. We begin by discussing some of the known theory for OCPs, as we will rely on these ideas later. We then discuss several different methods for solving OCPs, including an overview of the software GPOPS-II, and introduce our own control parameterization method.

## 3.1 Derivation of Necessary Conditions

We first give a proof of the known necessary conditions for optimality using Pontryagin's minimum principle and the calculus of variations. Consider the following general optimal control problem with fixed final time  $T$  in Bolza form:

$$\min J = \phi(x(T), T) + \int_{t_0}^T L(t, x, u) dt, \quad (3.1)$$

subject to the constraints

$$\dot{x} = f(t, x, u), \quad x(t_0) = x_0 \quad (3.2a)$$

$$0 = g(x(T), T), \quad (3.2b)$$

where  $L(t, x, u)$  is the weighting function of the trajectory and control,  $\phi$  is an endpoint weighting function, and  $g$  is our terminal constraint. We have  $x(t) \in \mathbb{R}^N$ ,  $u(t) \in \mathbb{R}^m$  and  $g(t) \in \mathbb{R}^p$ . We follow the conventions established in [51].

Define the Hamiltonian of the system as

$$H(t, x, u, \lambda) = L(t, x, u) + \lambda^T f(t, x, u), \quad (3.3)$$

where  $\lambda$  is our costate, with values  $\lambda(t) \in \mathbb{R}^n$ . Then our augmented cost functional is

$$\tilde{J} = \phi(x(T), T) + \nu^T g(x(T), T) + \int_{t_0}^T H(t, x, u, \lambda) - \lambda^T \dot{x} dt, \quad (3.4)$$

where  $\nu$  is the multiplier associated with our endpoint constraint,  $\nu(t) \in \mathbb{R}^p$ . Then, taking the derivative of  $\tilde{J}$  using Leibniz's rule [51] we obtain

$$\begin{aligned} d\tilde{J} = & (\phi_x + g_x^T \nu)^T dx \Big|_T + (\phi_t + g_t^T \nu) dt \Big|_T + g^T \Big|_T d\nu + (H - \lambda^T \dot{x}) dt \Big|_T - (H - \lambda^T \dot{x}) dt \Big|_{t_0} \\ & + \int_{t_0}^T H_x^T \delta x + H_u^T \delta u + (H_\lambda - \dot{x})^T \delta \lambda - \lambda^T \delta \dot{x} dt, \end{aligned} \quad (3.5)$$

where  $\delta x$ ,  $\delta u$ , and  $\delta \lambda$  are the variations in  $x$ ,  $u$ , and  $\lambda$  respectively. We note that  $\delta \dot{x}$  and  $\delta x$  are not independent increments, so we integrate by parts to obtain

$$-\int_{t_0}^T \lambda^T \delta \dot{x} dt = -\lambda^T \delta x \Big|_T + \lambda^T \delta x \Big|_{t_0} + \int_{t_0}^T \dot{\lambda}^T \delta x dt. \quad (3.6)$$

Now, from [51] we have the relation

$$dx(T) = \delta x(T) + \dot{x}(T) dT. \quad (3.7)$$

Substituting Eq. 3.7 and Eq. 3.6 into Eq. 3.5 and noting that  $t_0$ ,  $T$ , and  $x(t_0)$  are fixed, we have

$$d\tilde{J} = (\phi_x + g_x^T \nu - \lambda)^T dx \Big|_T + g^T \Big|_T d\nu + \int_{t_0}^T H_x^T \delta x + \dot{\lambda}^T \delta x + H_u^T \delta u + H_\lambda^T \delta \lambda - \dot{x}^T \delta \lambda dt. \quad (3.8)$$

Setting the coefficients of our independent increments equal to zero, we obtain our first order

necessary conditions

$$\dot{x} = f(t, x, u) \quad (3.9a)$$

$$-\dot{\lambda} = H_x(t, x, u) \quad (3.9b)$$

$$u \text{ minimizes } H(x, \lambda, u, t) \quad (3.9c)$$

$$x(t_0) = x_0 \quad (3.9d)$$

$$0 = g(x(T), T) \quad (3.9e)$$

with boundary conditions

$$(\phi_x + g_x^T \nu - \lambda)^T \Big|_T dx(T) = 0. \quad (3.10)$$

We note that if  $u$  is unconstrained, the minimization in Eq. 3.9c becomes  $H_u = 0$ .

## 3.2 Numerical Methods

In this introduction, we outline two general classes of numerical methods for optimal control problems: direct and indirect methods. We note that for both classes of numerical methods discussed here, a sufficiently accurate initial value is often required for the methods to perform well [10, 16, 67, 80]. We will see this sensitivity to the initial value illustrated later in our numerical examples in Chapter 5. We now give a more detailed discussion of various direct and indirect methods.

### 3.2.1 Solving the Necessary Conditions

As noted in the introduction, any indirect method for solving an optimal control problem involves solving the necessary conditions for optimality. In the previous section, we gave a derivation of the first order necessary conditions for optimality for a general unconstrained optimal control problem in Eq. 3.9 and Eq. 3.10. In rare cases when the cost and associated constraints are very simple, it is possible to obtain an analytic solution. However for most practical applications, a numerical approach is needed. Two main advantages for utilizing indirect methods is they are often very accurate and do not require solving a large NLP.

Indirect methods generally fall into one of two categories: indirect shooting methods or indirect collocation. In a shooting method, an initial guess is made for the value of the unknown boundary conditions. The Hamiltonian system is then integrated; if the solution differs from the known terminal conditions by more than a specified tolerance, the initial guess is adjusted and the integration repeated. Indirect shooting methods require relatively accurate initial guesses because of the sensitivity of the Hamiltonian system but are very easy to implement. A multiple shooting method subdivides the timespan into multiple intervals, and integrates the necessary conditions on each subinterval. Additional variables are introduced via continuity conditions on the state  $x$

and costate multiplier  $\lambda$  at each interior node. Integrating over a shorter time span helps mitigate the ill-conditioning of the system. [23, 67, 78]

In an indirect collocation method, one applies a collocation scheme to the two point BVP in terms of the state  $x$  and the costate  $\lambda$  which results from finding the necessary conditions. Consider, for example, a pseudospectral collocation scheme, where the collocation nodes are the roots of an orthogonal polynomial defined on  $[-1, 1]$ . First, the BVP is transferred down onto the interval  $[-1, 1]$ . The variables are then discretized at the chosen collocation points. Depending upon the collocation method chosen, the collocation points may be defined on  $(-1, 1)$ ,  $[-1, 1)$ ,  $(-1, 1]$ , or  $[-1, 1]$ . The state and costate are then approximated by a basis of polynomials

$$x(y) \approx \sum_{j=1}^N x_j P_j(y) \quad (3.11a)$$

$$\lambda(y) \approx \sum_{j=1}^N \lambda_j P_j(y), \quad (3.11b)$$

which are then evaluated at the  $y_i$  collocation points in  $[-1, 1]$ ,  $i = 1, \dots, N$ . Here  $P_j$  are the approximating polynomials, and  $\lambda_j, x_j$  are the coefficients for which the software needs to solve. Two common approaches are to use the roots of Legendre or Chebyshev polynomials as the collocation points.

The approximations are then substituted into the Hamiltonian system given above in Eq. 3.9 and Eq. 3.10. For our purposes, it suffices to consider a quadratic cost of the form

$$J = \phi(x(T), T) + \int_{t_0}^T x^T Q x + u^T R u \, dt \quad (3.12)$$

and dynamics of the form

$$\dot{x} = F(x) + B(x)u \quad (3.13)$$

with  $R > 0$ . For  $u$  unconstrained, one can use the condition  $H_u = 0$  to solve for  $u$  in terms of  $\lambda$  to obtain  $u = -\frac{1}{2}R^{-1}B^T(x)\lambda$ . This gives, on  $[-1, 1]$ , the system of collocation equations

$$\sum_{j=0}^N x_j \frac{dP_j}{dt}(y_i) - \frac{t_f - t_0}{2} F(x_j, \lambda_j, y_j) = 0 \quad (3.14a)$$

$$\sum_{j=0}^N \lambda_j \frac{dP_j}{dt}(y_i) + \frac{t_f - t_0}{2} \left( \frac{\partial L}{\partial x}(x_j, \lambda_j, y_j) + \lambda_j F(x_j, \lambda_j, y_j) \right) = 0 \quad (3.14b)$$

on a single interval with mixed boundary conditions dependent upon the form of  $g(x(T), T)$ . We note the term  $\frac{t_f - t_0}{2}$  is required to account for translating the problem down to the interval  $[-1, 1]$ .

In general, pseudospectral methods are global collocation schemes. Thus  $[t_0, t_f] = [t_0, T]$ , that is, the problem is approximated by a global polynomial along the entire time interval. Later, when we discuss GPOPS-II, we will introduce an  $hp$ -adaptive pseudospectral method. In that case, the entire time interval  $[t_0, T]$  is subdivided,  $t_0, t_1, \dots, t_N = T$ , and each individual subinterval  $[t_i, t_{i+1}]$  is translated down onto  $[-1, 1]$ . [67, 75, 78]

Because there are no control parameters to optimize, the problem formulation in Eq. 3.14 can be solved with a BVP solver, not necessarily dedicated optimal control software. Even if one uses a collocation code, the indirect approach is still more labor intensive for the user, because one must first find the necessary conditions to provide to the software. Depending upon the form of the terminal constraint  $g(x(T), T)$  in Eq. 3.9e, obtaining the correct boundary conditions for the problem can become quite complicated.

### 3.2.2 Direct Transcription Methods

In our introduction, we note that direct methods, or direct transcription methods, are a class of numerical methods for solving optimal control problems. Rather than writing down the necessary conditions for a minimum, the optimal control problem is instead directly transcribed as a nonlinear programming problem (NLP). We now consider different types of direct methods in greater detail. There are two parts to a direct transcription method: the discretization of the optimal control problem, and the nonlinear optimizer used to solve the resulting NLP. We note both the discretization and optimization are handled by the software; in this section we explain how the software itself works.

#### 3.2.2.1 The Nonlinear Optimizer

In this work, we focus on a class of nonlinear optimization methods known as *gradient methods*. Gradient methods are iterative methods where the iteration is of the form

$$x_{k+1} = x_k + \alpha_k d_k, \quad (3.15)$$

where  $\alpha_k$  is the step size and  $d_k$  is the search direction. Two popular gradient-based methods for nonlinear optimization are sequential quadratic programming, and interior point methods. Our work with the optimal control software GPOPS-II utilizes an interior point method.

In an interior point method, the constrained NLP

$$\min f(x) \quad (3.16)$$

subject to

$$c(x) = 0 \quad (3.17a)$$

$$x_L \leq x \leq x_U \quad (3.17b)$$

is replaced with a sequence of unconstrained problems

$$\min_{x \in \mathbb{R}^n} \beta(x, \mu) = f(x) - \mu_i \sum_{i=1}^n \ln(x_i), \quad (3.18)$$

subject to  $c(x) = 0$  where each  $\mu_i$  is a parameter in a sequence where the  $\mu_i$  are decreasing to zero. Interior point methods are sometimes referred to as barrier methods, where  $\beta(x, \mu)$  in Eq. 3.18 is the logarithmic barrier function and  $\mu$  is the barrier parameter. We note that Eq. 3.18 is only one choice of barrier function [9].

We utilize GPOPS-II with the open source interior point software IPOPT, an Interior Point OPTimizer. IPOPT uses a damped Newton's method to solve the primal-dual formulation of the barrier problem in Eq. 3.18 [80]; solving Eq. 3.18 is equivalent to applying a homotopy method to the primal-dual system

$$\nabla f(x) + \nabla c(x)\lambda - z = 0 \quad (3.19a)$$

$$c(x) = 0 \quad (3.19b)$$

$$XZe - \mu e = 0, \quad (3.19c)$$

where  $e$  is a vector of ones,  $\mu$  is now the homotopy parameter, and the dual variables  $\lambda$  and  $z$  are the associated Lagrange multipliers for the equality constraints and variable bounds. In this case  $X$  and  $Z$  are diagonal matrices where the diagonals are the associated vectors  $x$  and  $z$  [57, 68, 80]. The additional conditions  $x \geq 0$ ,  $z \geq 0$  coupled with Eq. 3.19 give the KKT conditions for the original NLP in Eq. 3.16 and Eq. 3.17.

Rather than using a merit function to determine the step size  $\alpha_k$  in Eq. 3.15, IPOPT uses a line-search filter method [79, 80]. A merit function seeks to decrease both the objective function and the infeasibility of the constraint, while a filter method treats these as two separate criteria [77]. The outline of the filter method used in IPOPT may be found in [79, 80]. The search direction  $d_k$  is determined from a linearization of the primal-dual system in Eq. 3.19.

### 3.2.2.2 Solution of the Optimal Control Problem

We now consider different methods of discretizing and solving the optimal control problem. The first, and most simple, is a *direct shooting method*. Direct shooting is a form of control parameterization.

Given a time grid  $t_0 < t_1 \leq \dots \leq t_m = t_f$  we parameterize the control  $u(t)$  with a known function,

$$u(t) \approx u_j(a_j, t), \quad j = 1 \dots m, \quad t \in [t_j, t_{j+1}]. \quad (3.20)$$

Let  $u(t) \in \mathbb{R}^q$ , and suppose we choose a polynomial of order  $p$  to parameterize the control. Then the vector of parameters is  $a_j \in \mathbb{R}^r$  where  $r = q(p+1)$ . Examples of common functions include piecewise linear functions and splines; later in this work we present the results of a control parameterization code where the control is parameterized with cubic splines. [42]

The user provides an initial estimate of the control parameters to a numerical integrator, used to integrate the dynamics and calculate the cost. These values are then passed to a nonlinear optimizer, and the process is repeated until the constraints have been met and the cost minimized within a user-specified tolerance. Because they do not require specialized control software, direct shooting methods are very easy to implement. However a very fine mesh or higher order approximating function may be required to obtain a sufficiently accurate solution, resulting in a high dimensional problem with a greatly increased computational cost. Additionally, a sufficiently "good" initial estimate of the parameters may be required to obtain an accurate solution.

In order to overcome some of the difficulties in a single shooting method, one may use a multiple shooting method. Now the time interval is discretized into finite subintervals,  $[t_j, t_{j+1}]$ ,  $j = 0 \dots m$ . We again parameterize the control as in Eq. 3.20. However, we integrate the dynamics and calculate the cost on each individual subinterval. The initial value of the state,  $x$ , on each subinterval becomes an additional parameter in the optimization.

Let our initial condition on each interval be  $x(t_j) = y_j$ . Then we have the additional continuity condition

$$y_{j+1} - x(t_{j+1}; y_j, a_j) = 0, \quad t \in [t_j, t_{j+1}], \quad (3.21)$$

where the state  $x$  evaluated at  $t_{j+1}$  depends on the initial condition  $y_j$  and the value of the parameter for the control  $a_j$ . Introducing an additional parameter into the optimization increases the dimension of the problem but integrating the dynamics on much shorter time intervals helps decrease the sensitivity to the initial estimate.

An alternative to direct shooting methods is direct collocation. In a direct collocation scheme, the collocation method is applied directly to the optimal control problem. Both the state and control are parameterized. Generally, direct collocation methods can be classified as either *local* or *global*. In a local collocation scheme the time interval is discretized into a finite number of subintervals,  $[t_j, t_{j+1}]$ ,  $j = 0, \dots, m$ .

The state and control are usually approximated by a piecewise continuous polynomial,

$$x(t) \approx \sum_{i=0}^n P_{ij}(t) x_{ij} \quad (3.22a)$$

$$u(t) \approx \sum_{i=0}^q P_{ij}(t) u_{ij}, \quad (3.22b)$$

where  $x_{ij}$  and  $u_{ij}$  are the coefficients to be determined,  $t \in [t_j, t_{j+1}]$ , where  $i$  is the index within each subinterval. Individual subintervals are again linked by a continuity condition on the states, where the terminal value of  $x$  on  $[t_{j-1}, t_j]$  is enforced as the initial condition of  $x$  on  $[t_j, t_{j+1}]$ . The number of subintervals may be increased to obtain convergence while the order of the approximating polynomial remains fixed. Continuity of the control  $u$  across subintervals may not be enforced; depending upon the application, a piecewise continuous control may be acceptable.

In a global collocation scheme, the time interval is discretized  $t_0 < t_1 < \dots < t_{m-1} < t_m = t_f$  and the solution is approximated by a global polynomial on the entire time interval. In contrast to a local collocation method, the number of intervals (i.e. one) remains fixed while the order of the approximating polynomial is increased. An *orthogonal* collocation scheme is one in which the collocation nodes are chosen as the roots of orthogonal polynomials. A *pseudospectral* collocation scheme is a global orthogonal collocation scheme.

### 3.2.2.3 GPOPS-II: HP-Adaptive Pseudospectral Method

GPOPS-II, a general purpose optimal control MATLAB software, is an  $hp$ -adaptive pseudospectral method [61]. Here,  $h$  refers to the number of mesh intervals, and  $h$ -methods achieve convergence by increasing the number of mesh intervals until a specified error tolerance is satisfied. In contrast,  $p$  refers to the order of the approximating polynomial and a  $p$ -method relies on increasing the degree of the approximating polynomial. As described in the previous section, a pseudospectral method is a global orthogonal collocation scheme. GPOPS-II uses a Radau pseudospectral method or Legendre-Gauss-Radau collocation scheme on each individual mesh interval [22, 29, 61]. The formulas for the LGR nodes can be found in [20] and are zeroes of

$$L_N + L_{N+1}, \quad (3.23)$$

where  $L_N$  is the  $N$ th order Legendre polynomial and the weights are given by

$$w_j = \frac{1}{(N+1)^2} \frac{1-y_j}{(L_N(y_j))^2}. \quad (3.24)$$

The approximating polynomials used are the Lagrange polynomials in Eq. 3.27 [61].

We present the general outline of how the Radau pseudospectral method is implemented for a general unconstrained optimal control problem,

$$\min J = \phi(x(T), T) + \int_{t_0}^{t_f} L(t, x, u) dt, \quad (3.25)$$

subject to the dynamics

$$\dot{x} = f(t, x(t), u(t)). \quad (3.26)$$

Later on, we give a more detailed explanation of the collocation equations for a specific form of the optimal control problem. For now we follow the notational conventions established in [28, 29, 30] and give the differential form of the collocation scheme on  $[-1, 1]$ .

In GPOPS-II, the user has the option to use the differential or integral form of the collocation equations, which were shown to be equivalent in [29]. Consider the time interval  $[t_0, t_f]$  and subdivide it into  $K$  intervals  $[T_k, T_{k+1}]$ . The LGR collocation points are defined on the half-open interval  $[-1, 1)$ , so each interval  $[T_k, T_{k+1}]$  is mapped down onto  $[-1, 1]$ . The authors define a new independent variable on  $[-1, 1]$ . Let  $y \in [-1, 1]$ , so  $t = \frac{T_{k+1} - T_k}{2}y + \frac{T_{k+1} + T_k}{2}$ . Thus a factor of  $\frac{T_{k+1} + T_k}{2}$  is carried through the approximation.

Let  $x^{(k)}(y_j)$  be the state after the change of variables from grid points  $t_i$  on  $[T_k, T_{k+1}]$  to collocation points  $y_j$  on  $[-1, 1]$  where  $k = 1, \dots, K$  is the index of the interval. Then one can approximate the dynamics at the collocation points  $y_j$ ,  $j = 1, \dots, N_k + 1$  in each interval on  $[-1, 1]$  using a basis of Lagrange polynomials,

$$x^{(k)}(y) \approx \sum_{j=1}^{N_k+1} x_j^{(k)} p_j^{(k)}(y), \quad p_j^{(k)}(y) = \prod_{l=0, l \neq j}^{N_k} \frac{y - y_l^{(k)}}{y_j^{(k)} - y_l^{(k)}}, \quad (3.27)$$

where  $N_k$  is the order of the approximating polynomial in the  $k$ th interval. The authors in [30] introduce an additional noncollocated point  $y_{N_k+1} = 1$  to account for the righthand endpoint of the interval in the state approximation.

The approximation is then differentiated and substituted it into the dynamics to obtain the equation

$$\sum_{j=1}^{N_k+1} x_j^{(k)} \frac{d p_j^{(k)}(y_i)}{d y} = \frac{T_{k+1} - T_k}{2} f(x_i^{(k)}, u_i^{(k)}, y_i^{(k)}), \quad i = 1, \dots, N_k, \quad (3.28)$$

where  $u_i^{(k)}$  is the approximation to the control on the  $k$ th interval. The authors of the software use the same variable for  $y_{N_k+1}^{(k)}$  and  $y_1^{(k+1)}$  to enforce continuity across intervals [60]. The cost  $J$  is

approximated by an LGR quadrature rule with

$$J \approx \phi(x_{N_k+1}^{(k)}, T_{k+1}) + \sum_{k=1}^K \sum_{j=1}^{N_k+1} \frac{T_{k+1} - T_k}{2} w_j^{(k)} L(y_j^{(k)}, x_j^{(k)}, u_j^{(k)}). \quad (3.29)$$

The scheme is called "adaptive" because GPOPS-II is able to change the number of mesh intervals and the order of the approximating polynomial. There are several different mesh refinement schemes available to the user, such as those outlined in [22] and [52]. We use the default 'hp-PattersonRao' mesh refinement scheme established by Patterson, Hager, and Rao in [59]. While the software option is called 'hp-PattersonRao', the authors refer to their mesh refinement algorithm as a *ph* scheme in [59] because the algorithm first seeks to increase the order of the approximating polynomial on a given interval to satisfy a user-specified tolerance and only subdivides the interval if the polynomial degree cannot be further increased.

In order to determine when to increase the order of the approximating polynomial or subdivide the mesh, the authors utilize a user-specified error tolerance  $\epsilon$  and an estimate of the relative error in the state,  $e^{(k)}$ . The authors obtain their estimate of the relative error by comparing the solution on the current mesh,  $x^{(k)}(y_l)$ , to a polynomial approximation,  $\hat{x}^{(k)}(\hat{y}_l)$ ,  $l = 1, \dots, N + 2$ , they construct with higher accuracy. The authors assume the solution to the optimal control problem is sufficiently smooth, such that a larger number of LGR collocation points yields a more accurate approximation of the dynamics. Then for the  $i$ th component of the state, the relative error  $e_i^{(k)}$  is

$$e_i^{(k)}(\hat{y}_l^k) = \frac{|\hat{x}_i^{(k)}(\hat{y}_l^{(k)}) - x_i^{(k)}(\hat{y}_l^{(k)})|}{1 + \max_j |x_i^{(k)}(\hat{y}_j^{(k)})|}, \quad (3.30)$$

where  $i = 1, \dots, n$ ,  $l = 1, \dots, N_k + 2$ , and  $j = 1, \dots, N_k + 2$  where  $N_k$  is the number of LGR points on a given interval and  $n$  is the dimension of the continuous-time state.

In order to determine how to refine the mesh, the user is asked to specify the following in the GPOPS-II main file

```
%-----%
%-----Provide Mesh Refinement Method and Initial Mesh -----%
%-----%

mesh.method          = 'hp-PattersonRao';
mesh.tolerance        = eps;
mesh.maxiteration     = Mmax;
mesh.colpointsmx      = Pmax;
mesh.colpointsmn      = Pmin;
```

where `mesh.method` is the mesh refinement method, `mesh.tolerance` is the tolerance  $\epsilon$ , `mesh.maxiteration` is the maximum number of mesh iterations, and `mesh.colpointsmax` and `mesh.colpointsmmin` are the maximum and minimum allowable number of collocation points in each interval respectively.

The argument of `mesh.colpointsmmin` must be at least two, and the user must set `mesh.colpointsmmax`  $\geq$  `mesh.colpointsmmin`. Let  $e_{\max}^{(k)}$  be the maximum relative error on the  $k$ th mesh interval  $S_k$  and  $e_{\max}^{(k)} > \epsilon$  so the mesh needs to be refined. The method in [59] first seeks to increase the degree of the approximating polynomial on  $S_k$  by checking if the predicted number of collocation points  $N_k$  needed on the next mesh iteration is larger than  $P_{\max}$ , the maximum polynomial degree specified by the user.

In order to determine how many new collocation points to add on the next mesh iteration, the authors use the fact that the convergence rate for a global collocation scheme is approximately  $O(N^{2.5-c})$  where  $N$  is the number of collocation points in the interval and  $c$  is the number of continuous derivatives in the solution. For a sufficiently smooth solution, the authors are able to take  $c = N$ . Thus, if the number of collocation points in the interval on the next iteration is  $N + P$  the error bound decreases by a factor of  $N^{-P}$ .

In order to obtain the desired tolerance  $\epsilon$ , the authors need  $e_{\max}^{(k)*} = e_{\max}^{(k)} (\epsilon / e_{\max}^{(k)})$ . They choose  $N_k$  where  $N_k^{-P} = \epsilon / e_{\max}^{(k)}$ . Then they use the expression for  $N^{-P}$  to solve for  $P$  and obtain

$$N^P = e_{\max}^{(k)} / \epsilon \implies P = \log_{N_k}(e_{\max}^{(k)} / \epsilon), \quad (3.31)$$

where  $P = \lceil \log_{N_k}(e_{\max}^{(k)} / \epsilon) \rceil$  if Eq. 3.31 is not an integer.

Say  $N_k \leq P_{\max}$ . Then GPOPS-II will increase the number of collocation points to  $N_k$  on the next iteration, and repeat the process until either  $e_{\max}^{(k)} < \epsilon$  for all  $k$  or until  $N_k > P_{\max}$ . Now suppose  $N_k > P_{\max}$ , so GPOPS-II can no longer increase the degree of the polynomial by adding more collocation points. Then GPOPS-II will seek to subdivide the mesh interval  $S_k$ . The authors have established two main criteria for the mesh division: the sum of the number of collocation points in the new subintervals must be equal to the predicted polynomial degree on the next mesh and the software must place  $P_{\min}$  collocation points in each new subinterval. Then the number of new subintervals created is  $\max\left(\left\lceil \frac{N_k}{P_{\min}} \right\rceil, 2\right)$  [59].

GPOPS-II has two NLP solver options: the sequential quadratic programming method SNOPT [31], which requires the user to purchase a license, and the open source interior point method IPOPT. For our work we use IPOPT, which we explain in Section 3.2.2.1. There are three main options for supplying derivatives to the NLP solver in GPOPS-II: a built-in sparse finite differencing scheme [60], automatic differentiation with the software ADiGator [81], or user-supplied analytic derivatives. Analytic derivatives must be passed to GPOPS-II using the assumed sparsity structure of the problem, making this approach unsuitable for our application.

The user has the option to specify the NLP solver use either first or second order derivatives. For our work, when using numerical differentiation we provide IPOPT with second order derivatives and we select GPOPS-II's sparse finite differencing scheme with central differences. We note the default setting in GPOPS-II is to supply only first derivatives to the NLP solver using the built-in sparse finite difference method.

### 3.2.3 Control Parameterization

When using GPOPS-II on delayed optimal control problems, we require another numerical solution to validate our results. We note that many direct transcription methods, such as shooting methods, parameterize the control. Direct transcription codes generally involve dedicated solvers and hundreds of grid points in the discretization. In this work when we refer to "control parameterization" we are referring to parameterizing the control with a relatively low number of mesh segments, solving the cost and constraint equations with a differential equation solver, and using a nonlinear optimizer to find the value of our parameters and the cost.

To verify our algorithm, we begin with the following example

$$\dot{x} = -0.5x + u, \quad x(0) = 0 \quad (3.32)$$

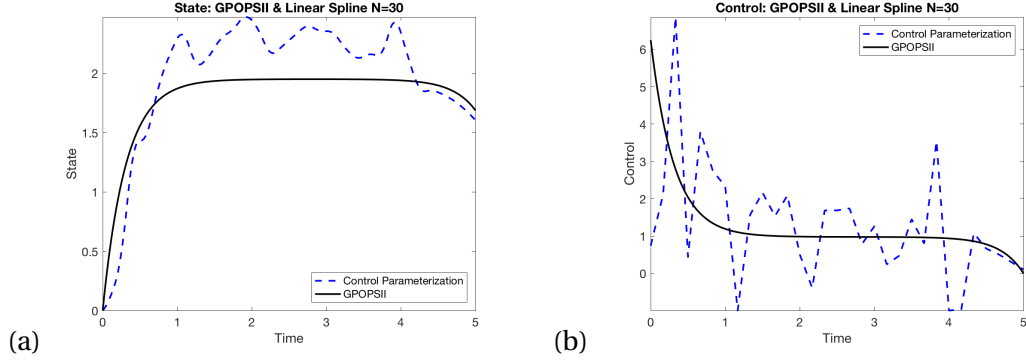
with the cost

$$J = \int_0^5 10(x-2)^2 + u^2 dt. \quad (3.33)$$

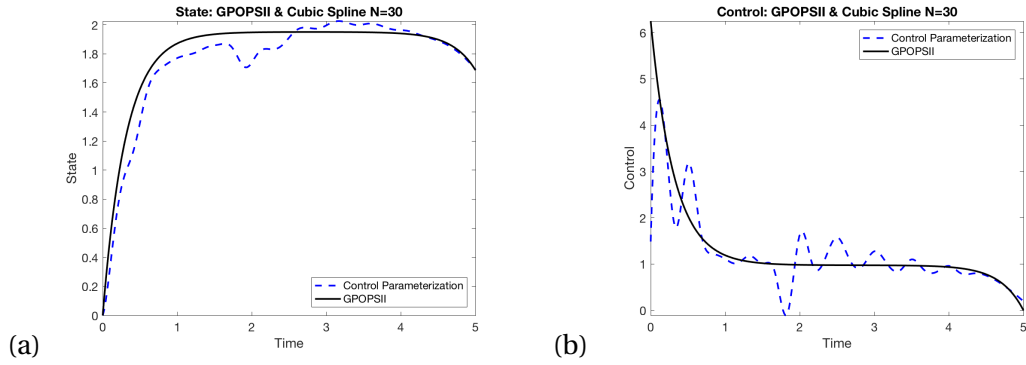
Given Eq. 3.32 is an undelayed control problem we are able to use GPOPS-II to establish a "true" solution. We set our tolerances at  $1e-7$  for both methods. We allow GPOPS-II five iterations and set the initial number of mesh intervals to  $N = 4$  and fix the order of our polynomial at  $P = 3$ . GPOPS-II finds the cost to be  $J = 18.7137100$ .

We now solve Eq. 3.32 using control parameterization. We use the MATLAB solver ode45 to find a solution to the system and then minimize the cost  $J$  over the control parameters  $u$  using the MATLAB nonlinear solver fmincon. We initially use linear splines to interpolate our control and divide our interval  $[0,5]$  into  $N = 30$  segments. We obtain the cost  $J = 31.2462859$ . We plot our state and control in Fig. 3.1. We can see the solutions obtained from control parameterization oscillate significantly. The state from control parameterization in Fig. 3.1(a) roughly follows the state obtained by GPOPS-II but the control in Fig. 3.1(b) does not approximate the control from GPOPS-II well at all.

To improve the quality of our approximation using control parameterization we keep our initial  $N = 30$  but now interpolate our control using cubic splines instead. We obtain cost  $J = 20.4920886$ . Fig. 3.2 shows that using cubic splines enables our code to approximate the answers obtained by GPOPS-II much more closely, however there is still a significant amount of oscillation.



**Figure 3.1** (a) State and (b) control obtained by GPOPS-II and control parameterization for Eq. 3.32 with cost Eq. 3.33 using linear splines and  $N = 30$  segments.



**Figure 3.2** (a) State and (b) control obtained by GPOPS-II and control parameterization for Eq. 3.32 with cost Eq. 3.33 using cubic splines and  $N = 30$  segments.

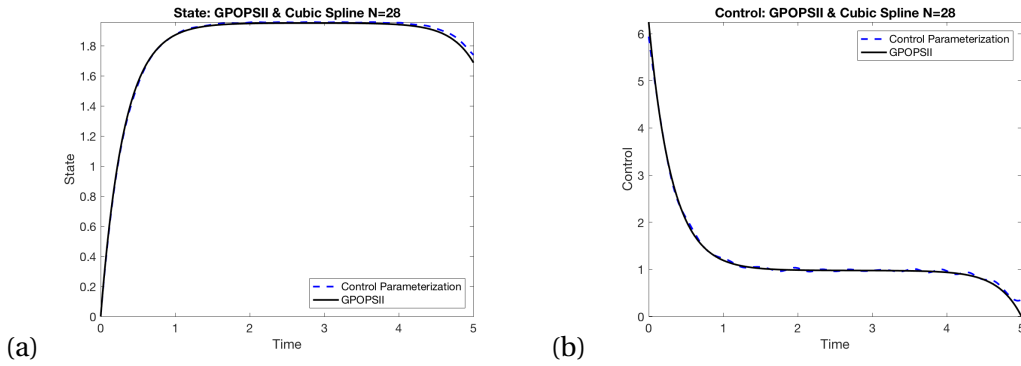
We attempt to find the initial number of segments  $N$  at which the quality of our approximation degrades. Table 3.1 shows the number of segments and the cost obtained by our control parameterization code. We can see the optimal number of initial segments is  $N = 28$ , as it gives us the best

**Table 3.1** Cost obtained by control parameterization using cubic splines for different number of grid segments,  $N$ .

$N$	J
27	19.7921053
28	19.5916043
29	19.6824370
30	20.4920886
32	20.2325944
35	20.3113542
36	25.1148795

approximation to our cost  $J = 18.7137100$  obtained from GPOPS-II.

Although  $N = 28$  gives us the best value of the cost, our approximation still has a significant amount of oscillation in the state and control. We therefore consider changing our initial iteration value for the control  $u$ . Previously we took our initial value to be  $u = 0 \ast (1 : N + 1)$ . Our new initial value is  $u(0) = 6 \ast (1 : N + 1)^{-1}$ , and we continue to use  $N = 28$ . Our new cost is  $J = 18.7325981$ , much closer to GPOPS-II's result of  $J = 18.7137100$ . Fig. 3.3 shows that our state and control closely resemble those obtained by GPOPS-II, demonstrating this problem's sensitivity to the initial value in the control. While adjusting the initial number of intervals  $N$  was helpful, it was not enough to obtain a good solution for poor choice of initial iteration value in the control.



**Figure 3.3** (a) State and (b) control obtained by GPOPS-II and control parameterization for Eq. 3.32 with cost Eq. 3.33 using cubic splines and  $N = 28$  segments for new initial value  $u(0) = 6 \ast (1 : N + 1)^{-1}$ .

Our computational results in this section demonstrate our control parameterization technique is a valid method for solving optimal control problems, however the process is very computationally intensive and the quality of the results is highly dependent upon the initial choice of parameters for  $u$ . We can see from both our costs and plots in Fig. 3.1 - Fig. 3.3 that even for the relatively easy problem in Eq. 3.32, Eq. 3.33 the initial value for the control is extremely important for obtaining an optimal solution. These results are not surprising; the choice of initial value in the control is often extremely important in the numerical solution of optimal control problems [10]. We therefore expect the initial choice for  $u$  to be extremely important later in this work when we consider optimal control problems with delays.

## CHAPTER

# 4

# OPTIMAL CONTROL PROBLEMS WITH STATE DELAYS

In Chapter 3, we discuss optimal control problems without delays. We outline several numerical methods, including those underlying the software GPOPS-II. We are now ready to begin our discussion of optimal control problems with delays. In this chapter, we focus on OCPs with state delays. We review a selection of existing numerical methods for solving delay OCP. We then introduce two new GPOPS-II problem formulations for solving OCPs with state delay. We discuss the necessary conditions for each problem formulation, and explain how these necessary conditions can affect our numerical results. We also introduce the collocation equations which result from an OCP with state delay, and discuss how the additional delay terms may affect the performance of GPOPS-II.

### 4.1 Derivation of Necessary Conditions for the Delayed Problem

In Chapter 1 we give a general statement of the necessary conditions for a problem with state and control delays in Theorem 1.3.1. We now derive the necessary conditions for the following optimal

control problem with a state delay

$$J = \int_0^T x^T Q x + u^T R u \, dt \quad (4.1a)$$

$$\dot{x} = Ax + Cx(t - \tau) + Bu \quad (4.1b)$$

$$x(t) = \phi(t), \quad t \in [-\tau, 0], \quad (4.1c)$$

where  $A, B, C$  are time-varying with piecewise smooth entries,  $R > 0$ , and  $Q \geq 0$ . Then our augmented cost is

$$\tilde{J} = \int_0^T x^T Q x + u^T R u + \lambda^T (Ax + Cx(t - \tau) + Bu - \dot{x}) \, dt \quad (4.2)$$

and our Hamiltonian is

$$H(t, x, x(t - \tau), u, \lambda) = x^T Q x + u^T R u + \lambda^T (Ax + Cx(t - \tau) + Bu). \quad (4.3)$$

In order to find the necessary conditions for a minimum we wish to take the derivative of our augmented cost functional and set this derivative to zero. Using Leibniz's rule to take the derivative of our augmented cost is straightforward for the increments  $\delta u$ ,  $\delta \lambda$  in  $u$  and  $\lambda$ . However, in order to take the derivative of the term  $\lambda^T Cx(t - \tau)$  with respect to  $\delta x$  in Eq. 4.3 we need to account for the delay.

Consider the integral

$$\int_0^T \lambda^T(t) Cx(t - \tau) \, dt = \int_0^\tau \lambda^T(t) Cx(t - \tau) \, dt + \int_\tau^T \lambda^T(t) Cx(t - \tau) \, dt. \quad (4.4)$$

The integral on  $[0, \tau]$  in Eq. 4.4 is defined by the prehistory  $\phi(t)$  in Eq. 4.1c, so its derivative with respect to the increment  $\delta x$  is zero. For the integral on  $[\tau, T]$  in Eq. 4.4 we have

$$\int_\tau^T \lambda^T(t) Cx(t - \tau) \, dt = \int_0^{T-\tau} \lambda^T(t + \tau) Cx(t) \, dt = \int_0^T \chi_{[0, T-\tau]} \lambda^T(t + \tau) Cx(t) \, dt, \quad (4.5)$$

where  $\chi_{[0, T-\tau]}$  is the characteristic function of  $[0, T - \tau]$ . Then the derivative of our cost functional  $J$  with respect to each of our increments  $\delta x$ ,  $\delta u$  and  $\delta \lambda$  is

$$d\tilde{J} = (H - \lambda^T \dot{x}) \, dt \Big|_T - (H - \lambda^T \dot{x}) \, dt \Big|_0 \quad (4.6a)$$

$$+ \int_0^T \left[ (2Ru + B^T \lambda)^T \delta u + (2Qx + A^T \lambda + \chi_{[0, T-\tau]} C^T \lambda(t + \tau))^T \delta x \right] \, dt \quad (4.6b)$$

$$+ \int_0^T \left[ -\lambda^T \delta \dot{x} + (Ax + Cx(t - \tau) + Bu - \dot{x})^T \delta \lambda \right] \, dt. \quad (4.6c)$$

We note that  $\delta x$  and  $\delta \dot{x}$  in Eq. 4.6 are not independent. To obtain independent increments we integrate by parts, which gives us

$$-\int_0^T \lambda^T \delta \dot{x} = -\lambda^T \delta x \Big|_T + \lambda^T \delta x \Big|_0 + \int_0^T \dot{\lambda}^T \delta x \, dt. \quad (4.7)$$

From [51] we have the relation

$$dx(T) = \delta x(T) + \dot{x}(T) dT. \quad (4.8)$$

Substituting Eq. 4.8 into Eq. 4.7 and then into Eq. 4.6 we obtain

$$d\tilde{J} = (H - \lambda^T \dot{x} + \lambda^T \dot{x}) dt \Big|_T - \lambda^T dx \Big|_T - (H - \lambda^T \dot{x} + \lambda^T \dot{x}) dt \Big|_0 + \lambda^T dx \Big|_0 \quad (4.9a)$$

$$+ \int_0^T (2Qx + A^T \lambda + \chi_{[0, T-\tau]} C^T \lambda(t+\tau))^T \delta x + \dot{\lambda}^T \delta x \, dt \quad (4.9b)$$

$$+ \int_0^T (2Ru + B^T \lambda)^T \delta u + (Ax + Cx(t-\tau) + Bu - \dot{x})^T \delta \lambda \, dt. \quad (4.9c)$$

Now  $\delta x$ ,  $\delta u$ , and  $\delta \lambda$  are independent increments. Since our control  $u$  is unconstrained, to obtain our necessary conditions for a minimum we set their coefficients equal to zero. We obtain

$$H_u = 2Ru + B^T \lambda = 0. \quad (4.10)$$

So  $H_u = 0$  and our integral becomes

$$d\tilde{J} = H dt \Big|_T - \lambda^T dx \Big|_T - H dt \Big|_0 + \lambda^T dx \Big|_0 \quad (4.11a)$$

$$+ \int_0^T (2Qx + A^T \lambda + \chi_{[0, T-\tau]} C^T \lambda(t+\tau))^T \delta x \, dt \quad (4.11b)$$

$$+ \int_0^T \dot{\lambda}^T \delta x + (Ax + Cx(t-\tau) + Bu - \dot{x})^T \delta \lambda \, dt. \quad (4.11c)$$

Since  $\lambda$  and  $x$  are unconstrained and  $\delta x$ ,  $\delta \lambda$  are independent we can set their coefficients to zero to obtain

$$\dot{x} = H_\lambda = Ax + Cx(t-\tau) + Bu \quad (4.12a)$$

$$-\dot{\lambda} = H_x = 2Qx + A^T \lambda + \chi_{[0, T-\tau]} C^T \lambda(t+\tau). \quad (4.12b)$$

For our boundary conditions, we note that our endpoints  $0, T$  are fixed. Thus  $dt \Big|_0 = dt \Big|_T = 0$ . Furthermore,  $dx(0) = 0$  since we have fixed our initial condition. However,  $dx(T)$  is free, so we have

the boundary condition

$$\lambda(T) = 0. \quad (4.13)$$

The necessary conditions for our problem Eq. 4.1 are

$$\dot{x} = H_\lambda = Ax + Cx(t-\tau) + Bu \quad (4.14a)$$

$$-\dot{\lambda} = H_x = 2Qx + A^T\lambda + \chi_{[0, T-\tau]} C^T \lambda(t+\tau) \quad (4.14b)$$

$$H_u = 2Ru + B^T\lambda = 0 \quad (4.14c)$$

$$x(t) = \phi(t), \quad t \in [-\tau, 0] \quad (4.14d)$$

$$\lambda(T) = 0. \quad (4.14e)$$

## 4.2 Review of Existing Numerical Methods for Delayed Optimal Control Problems

Later in this work we demonstrate how to use GPOPS-II to solve the necessary conditions, Eq. 4.14, which we just derived. In this section, we discuss the pre-existing methods for solving delayed OCP. We rely on these methods to validate our results.

### 4.2.1 Method of Steps

We originally introduce the method of steps (MOS) in Chapter 2 as a way to reduce a DDE to a system of ODEs. MOS can also be used to rewrite a delayed optimal control problem as a system of control problems without the delay, enabling the use of a wider range of optimal control solvers. We seek to use MOS to rewrite an optimal control problem with constant delay in the state in order to solve it with GPOPS-II, which will be very useful in establishing a “true” solution later in this work.

Given a general optimal control problem with a state delay

$$\dot{x} = f(t, x(t), x(t-\tau), u(t)) \quad (4.15a)$$

$$J = \int_0^T L(t, x(t), u(t)) dt \quad (4.15b)$$

we can rewrite the cost as

$$\sum_{i=1}^N \int_0^\tau L(t, x_i(t), u_i(t)) dt \quad (4.16)$$

and our dynamics as the system

$$\dot{x}_1(t) = f(t, x_1(t), \phi(t), u_1(t)) \quad (4.17a)$$

$$\dot{x}_i = f(t, x_i(t), x_{i-1}(t), u_i(t)), \quad i = 2, \dots, N \quad (4.17b)$$

$$x_1(0) = \phi(0) \quad (4.17c)$$

$$x_i(0) = x_{i-1}(\tau), \quad i = 2, \dots, N. \quad (4.17d)$$

To illustrate, choose the specific example

$$\min \int_0^5 10(x(t)-2)^2 + u^2(t) dt \quad (4.18a)$$

subject to

$$\dot{x}(t) = -0.5x(t) + c x(t-\tau) + u(t) \quad (4.18b)$$

$$x(t) = 1, \quad -\tau \leq t < 0 \quad (4.18c)$$

$$x(0) = 1 \quad (4.18d)$$

with  $\tau = 1$  and  $c = +1.2$ . The MOS formulation of this problem is then equivalent to minimizing the cost

$$\sum_{i=1}^5 \int_0^1 10(x_i(t)-2)^2 + u_i^2(t) dt \quad (4.19a)$$

subject to the following dynamics on  $0 \leq t \leq 1$ :

$$\dot{x}_1(t) = -0.5x_1(t) + 1.2 + u_1(t) \quad (4.19b)$$

$$\dot{x}_i(t) = -0.5x_i(t) + 1.2x_{i-1}(t) + u_i(t), \quad i = 2, 3, 4, 5 \quad (4.19c)$$

$$x_1(0) = 1 \quad (4.19d)$$

$$x_i(0) = x_{i-1}(1), \quad i = 2, 3, 4, 5. \quad (4.19e)$$

We note that the MOS problem formulation in Eq. 4.19 is now a BVP, not just an initial value problem. We give the sample code required to solve this problem in GPOPS-II in the Appendix.

## 4.2.2 Control Parameterization

In addition to MOS we demonstrate in Chapter 3 that control parameterization can be a viable method for solving an optimal control problem accurately, though computational times often run long and the user would need to implement their own grid refinement scheme. The overall method for using control parameterization with a delayed optimal control problem is essentially identical

to that described in Chapter 3, the key difference being we need to introduce an integrator which is capable of handling delays.

We choose to implement control parameterization in MATLAB using the DDE integrator `dde23` and the nonlinear optimizer `fmincon`. We discuss the underlying Runge-Kutta method utilized in `dde23` in Chapter 2. We again choose to parameterize our control with  $N = 28$  subintervals and cubic splines. In order to pass our vector of control parameters  $u$  to the integrator `dde23`, we need to make our constant prehistory into a column vector:

```
function s = ddehist(t,u)
s = [1;0]
end
```

and pass  $u$  as an additional parameter argument to our derivative function:

```
function dydt = dde(t,y,Z,u)
```

where  $Z$  is our delay variable.

Then our function call to `dde23` is

```
sol = dde23(@dde, tau, @ddehist,[t0,tf],options,umin)
```

where `umin` is the vector of optimized parameters we obtain from `fmincon`. We solve several computational examples using this technique in Chapter 5 and validate our initial results with data obtained from the FORTRAN solver SOSD. We demonstrate our code solves relatively simple delayed state optimal control problems accurately, enabling us to utilize our code as a "true" solution to which we compare our results from GPOPS-II.

### 4.2.3 Direct Transcription

The advantage of control parameterization implemented using `dde23` and MATLAB is the user does not require any specialized software. This technique may be generalized to any DDE solver and nonlinear optimizer in another computing language. However, the user loses the benefit of powerful grid refinement techniques, sparse solvers, and other solver options present in specialized optimal control software. Additionally, computational times can run upwards of 20 minutes on a standard laptop. For more advanced applications, it is preferable to use a direct transcription code, which uses much finer grids than the control parameterization approach we describe in the previous section.

The basic process for solving a delayed optimal control problem using direct transcription is essentially identical to applying direct transcription to an optimal control problem without delay: the software first discretizes the optimal control problem on a given time grid, then passes the nonlinear system to an optimizer. If the tolerances are met, the process stops. If not, the grid is refined and

the optimization continues. A major difference occurs during the discretization: the software must now account for new dependencies among the delay terms. Furthermore, the structure and sparsity pattern of Jacobians is altered, which can impact the numerical solvers used on each mesh iteration.

Direct transcription methods have two key advantages in that they do not require the delayed necessary conditions, which feature an advance in the adjoint and can become quite complicated, and they are more global methods. Solving a delayed problem requires access to past values of either the state or the control. In a direct transcription method, the entire state and control vectors are available unlike in a step by step integrator.

### 4.3 Previously Existing Solvers for Delayed Optimal Control Problems

Direct transcription methods underly some of the most powerful commercially-available optimal control software packages. As noted in our introduction there are commercially-available software packages which solve optimal control problems with delays, each with its own advantages and drawbacks. For our purposes, we limit our discussion here to the two software packages we utilize in our work. We utilize the FORTRAN-based Sparse Optimization Suite, and later in this work we discuss the use of the Altair software Activate.

#### 4.3.1 SOS

There are several software tools available within the Sparse Optimization Suite (SOS); we utilize the package SOSD to validate the results from GPOPS-II and control parameterization. SOS is another direct transcription method. It utilizes three possible discretization methods: trapezoid, Hermite-Simpson, or a higher order Runge-Kutta method. In particular SOSD will utilize a second-order trapezoid discretization at the start of the computation, and then move to a fourth order Hermite-Simpson discretization on subsequent finer grids.

Suppose we have a delayed optimal control problem in the same format as that given in Eq. 4.1a - Eq. 4.1b . SOSD introduces a pseudovariable  $y$  for the delayed term  $x(t - \tau)$ . Then we have the dynamics

$$\dot{x} = f(x(t), u(t), y(t), t) \quad (4.20a)$$

$$0 = y(t) - x(t - \tau), \quad (4.20b)$$

a delayed differential algebraic equation [12].

After the software discretizes the problem, it is passed to a nonlinear optimizer. SOS is equipped with both sequential quadratic programming (SQP) or interior point (barrier) methods. Our data use the SQP method; in either case the optimizer requires both the Jacobian and Hessian of the problem. SOS utilizes a finite difference scheme to compute the necessary derivatives. The solver

then checks to see if the computed solution meets the user-specified error tolerances. If not, the solver refines the grid and begins the process again. This powerful and efficient grid refinement is a major advantage of using a direct transcription method on challenging optimal control problems, and why we seek to use GPOPS-II on delayed optimal control problems rather than relying on our own control parameterization method.

### 4.3.2 Activate

SOS is a powerful optimization software but requires the user to handle multiple FORTRAN files. For cases when a user may not want to script extensively, another option is Activate. Activate is a graphical modeling and simulation software produced by the company Altair. It is written in Open Matrix Language (OML), an interpreted language similar to MATLAB. Activate relies on the use of block diagrams to build models, similar to those used in Simulink. Activate also has the option for scripting in OML or generating code from the block diagram [19]. We will discuss Active in Section 5.3.

## 4.4 New Use of GPOPS-II on Delayed Optimal Control Problems

Having reviewed some of the existing methods of solving DDEs and delayed OCP, we now turn our attention to the software GPOPS-II. In this work, we consider several new applications of using GPOPS-II to solve problems with state and control delays. We establish two new problem formulations, which we refer to as GPOPS-II<sub>m</sub> and GPOPS-II<sub>ow</sub>. In addition to using GPOPS-II to solve delayed OCP, we demonstrate how to utilize GPOPS-II as an integrator on DDEs as well as a boundary value problem solver on the necessary conditions. We consider how the necessary conditions for different problem formulations affect their numerical convergence to the correct answer.

### 4.4.1 Formulating a Delayed State Optimal Control Problem in GPOPS-II

We begin our discussion by exploring how to implement an optimal control problem with state delay in GPOPS-II. The software GPOPS-II gives the user access to the entire state and control history of the computation [61]. Thus, we are able to formulate a delay problem in such a way as to allow GPOPS-II to attempt to solve the problem [11]. We note the code given below implements the dynamics and prehistory for the OCP given in Eq. 4.18 with  $\tau = 1$ .

We introduce the following code snippet into the “Continuous” file:

```
tau = input.auxdata.tau; %value of the delay
a = input.auxdata.a; % coefficient of state in dynamics
b = input.auxdata.b; %coefficient of control in dynamics
```

```

c = input.auxdata.c; %coefficient of delay term in dynamics
x = input.phase.state(:,1); %full state history
u = input.phase.control(:,1); %full control history
t = input.phase.time(:,1) % time grid of all collocation points
NN = iter(1,1): %iteration count
if NN < 4
dx = a*x + u; %initialize without delay term
else
ts = t-tau; %delayed time vector
tsn = ts<=0; %delayed times <= 0
tsp = ts>0; %delayed times > 0
xd1 = ones(size(tsn)); %prehistory for this example
xd2 = interp1(t,x,ts(tsp),'linear'); %linearly interpolate delay term
xd = [xd1;xd2]; %delayed state
dx = a*x + c*xd + b*u; %dynamics for this example
end
phaseout.dynamics = dx; %dynamics passed to GPOPS-II to solve

```

where the "if" statement allows GPOPS-II to initialize without the delay term, and xd2 linearly interpolates the vector of delayed values. We refer to this problem formulation, with an interpolation of the delay term, as GPOPS-II<sub>m</sub> [11].

We will demonstrate in Chapter 5 that GPOPS-II struggles to solve the GPOPS-II<sub>m</sub> problem formulation accurately for longer delays. We know from [60] that GPOPS-II exploits an assumed sparsity pattern in the first and second order derivatives of the problem. We hypothesize these presumed sparsity patterns ignore crucial derivative dependencies introduced by the delay term.

We note the user has the option to pass analytic derivatives to GPOPS-II; however the user needs to specify the location of each derivative and the associated variable. This structure means we cannot specify the appropriate derivative dependencies introduced by the delay term. We therefore explore additional methods of solving a delayed optimal control problem using GPOPS-II. For example, an iterative method we refer to as GPOPS-II<sub>ow</sub> is introduced later to investigate the effect of these derivative dependencies.

In the following section, we give the form of the collocation equations for GPOPS-II<sub>m</sub>. Then we give the necessary conditions for the delayed problem, on which we use the GPOPS-II<sub>m</sub> formulation. Later, we show that as the delay  $\tau$  approaches zero, the necessary conditions for delay problem, which we pass to GPOPS-II<sub>m</sub>, converge to the undelayed case, enabling GPOPS-II<sub>m</sub> to obtain an accurate solution to some optimal control problems for small delays.

#### 4.4.1.1 Collocation Equations for Delayed State Optimal Control Problems

Having explained how we implement GPOPS-II<sub>m</sub>, we now give the collocation equations for GPOPS-II<sub>m</sub>, which demonstrate the actual equations which we ask GPOPS-II to solve. We will identify which terms are not properly calculated by GPOPS-II<sub>m</sub>, introducing errors into the computation. We use the notation and conventions established in [60].

We consider a scalar OCP with state delay, so we have

$$\dot{x} = a(t)x(t) + c x(t - \tau) + b(t)u(t), \quad t \in (0, T] \quad (4.21a)$$

$$x = \phi(t), \quad t \in [-\tau, 0] \quad (4.21b)$$

$$J = \int_0^T x(t)^2 + u(t)^2 dt \quad (4.21c)$$

on the interval  $[0, T]$  where  $\tau$  is the value of the delay,  $u$  is our control,  $J$  is our cost, and  $\phi(t) = \phi = 1$  is our prehistory. We allow that the coefficients  $a(t)$  and  $b(t)$  may vary with time. GPOPS-II approximates the system on  $K$  mesh intervals,  $k = 1, \dots, K$ ,  $[T_k, T_{k+1}]$ .

Because GPOPS-II uses the Legendre Gauss Radau collocation points defined on  $[-1, 1]$ , each mesh interval  $[T_k, T_{k+1}]$  is mapped to  $[-1, 1]$  via  $\frac{t_0 - t_f}{2}$  where  $t_0$  and  $t_f$  depend upon the endpoints of each interval. The approximation of the state and control are defined in

$$x^{(k)}(y) \approx \sum_{j=1}^{N_k+1} x_j^{(k)} p_j^{(k)}(y) \quad (4.22)$$

and

$$u^{(k)}(y) \approx \sum_{j=1}^{N_k+1} u_j^{(k)} p_j^{(k)}(y) \quad (4.23)$$

respectively, where  $N_k$  is the order of the approximating polynomial on that mesh interval, and we have that our  $y$  are in  $[-1, 1]$ . Here,  $p_j^{(k)}$  are the Lagrange interpolating polynomials, defined by

$$p_j^{(k)}(y) = \prod_{l=1, l \neq j}^{N_k+1} \frac{y - y_l^{(k)}}{y_j^{(k)} - y_l^{(k)}}. \quad (4.24)$$

We seek to obtain the value of our delayed variable at a grid point  $y_j^{(k)}$ , the value of our state  $x$  at  $y_j^{(k)} - \tau$ . However,  $y_j^{(k)} - \tau$  may not always be a grid point. We use the built-in MATLAB function `interp1` to linearly interpolate our state from the time grid  $y_j^{(k)}$  to the delayed time grid  $y - \tau$ , for those  $y - \tau > 0$ . To obtain  $x(y_j^{(k)} - \tau)$ , MATLAB interpolates the function at neighboring grid points.

Let  $y_j^{(k)} - \tau \in [y_{j-s}^{(k)}, y_{j-s+1}^{(k)}]$ . Then our delayed variable is

$$x(y_j^{(k)} - \tau) = 1, \quad y_j^{(k)} - \tau \leq 0 \quad (4.25)$$

and for those  $y_j^{(k)} - \tau > 0$  we have

$$x(y_j^{(k)} - \tau) = x(y_{j-s}^{(k)}) + (y_j^{(k)} - y_{j-s}^{(k)}) \frac{x(y_{j-s+1}^{(k)}) - x(y_{j-s}^{(k)})}{y_{j-s+1}^{(k)} - y_{j-s}^{(k)}}, \quad (4.26)$$

where  $j = 1 \dots N$ . Then, to use the LGR nodes on  $[-1, 1]$ , the dynamics will be approximated as

$$\begin{aligned} \sum_{j=1}^{N_k+1} \frac{dp_j^{(k)}(y)}{dy} x_j^{(k)} &= \frac{t_f - t_0}{2} x_j^{(k)} a(y_j^{(k)}) \\ &+ \frac{t_f - t_0}{2} \left( c x_j^{(k)} \left( p_j^{(k)}(y_{j-s}^{(k)}) + \frac{y_j^{(k)} - y_{j-s}^{(k)}}{y_{j-s+1}^{(k)} - y_{j-s}^{(k)}} (p_j^{(k)}(y_{j-s+1}^{(k)}) - p_j^{(k)}(y_{j-s}^{(k)})) \right) + u_j^{(k)} b(y_j^{(k)}) \right). \end{aligned} \quad (4.27)$$

The matrix structure we obtain for these system dynamics is the same as that given in [60] with the addition of banded blocks to the matrix approximating the state. These banded entries are the result of our linear interpolation of the delay term. These delayed entries fill in some of the sparsity, resulting in a largely upper triangular structure. We note these entries will be neglected when GPOPS-II calculates the Jacobians, and therefore result in approximate Jacobians being used by IPOPT. Substituting the approximation into our cost, Eq. 4.21b, GPOPS-II uses the following quadrature rule from [60]

$$J \approx \sum_{k=1}^K \sum_{j=1}^{N_k+1} \frac{t_f - t_0}{2} w_j^{(k)} (p_j^{(k)})^2 (y_j^{(k)}) \left( (x_j^{(k)})^2 + (u_j^{(k)})^2 \right). \quad (4.28)$$

The diagram in (4.29) demonstrates the structure of the matrix for the dynamics given in Eq. 4.27. We assume fixed polynomial order  $P = 2$  and that the delay will be a grid point,  $y_{j+\tau}^{(k)}$ . We see that in rows corresponding to the dynamics at the collocation points, extra entries are introduced by the delay terms. These delay terms will not be taken into account by GPOPS-II when it passes the Jacobians to IPOPT, and in Chapter 5 we give conditions on how the size of these delay terms affects

the convergence of IPOPT to the correct solution.

$$\begin{bmatrix}
 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\
 \bullet & \bullet & \bullet & 0 & \dots & 0 & 0 & \bullet & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\
 \bullet & \bullet & \bullet & 0 & \dots & 0 & 0 & 0 & \bullet & 0 & \dots & 0 & 0 & 0 & 0 \\
 \bullet & \bullet & \bullet & 1 & \dots & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & \bullet & \bullet & \bullet & 0 & 0 & 0 & 0 & \dots & 0 & \bullet & 0 & 0 \\
 0 & 0 & 0 & \bullet & \bullet & \bullet & 0 & 0 & 0 & 0 & \dots & 0 & 0 & \bullet & 0 \\
 0 & 0 & 0 & \bullet & \bullet & \bullet & 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\
 \vdots & \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
 0 & 0 & 0 & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \bullet & 1 & 0 & 0 \\
 0 & 0 & 0 & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & 0 & \bullet & \bullet & \bullet \\
 0 & 0 & 0 & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & 0 & \bullet & \bullet & \bullet \\
 0 & 0 & 0 & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & 0 & \bullet & \bullet & \bullet
 \end{bmatrix} \quad (4.29)$$

#### 4.4.1.2 Case A: Boundary Value Problem for the Delayed Tracking Problem

As we note above, GPOPS-II<sub>m</sub> neglects the contributions of the delay terms when it calculates the Jacobians of the delayed OCP. We now give the necessary conditions for optimality for a tracking problem with state delay, for the form of the delay problem we pass to GPOPS-II<sub>m</sub>. We will refer to this as Case A in later sections.

Our interest in the necessary conditions are the following: first, given that GPOPS-II<sub>m</sub> performs well as an integrator on a simple delayed system without a control, we suppose GPOPS-II<sub>m</sub> may obtain a more accurate solution to the delayed optimal control problem by solving the necessary conditions. Second, later in this chapter we are able to demonstrate when the necessary conditions for OCP with state delay approach those of an OCP without a delay. This helps us to predict when GPOPS-II<sub>m</sub> will perform well on an OCP with state delay. We present our numerical results for utilizing GPOPS-II<sub>m</sub> in Chapter 5.

Suppose we have the system

$$\dot{x} = Ax + Cx(t-\tau) + Bu \quad (4.30a)$$

$$J = \int_0^T (x-r)^T Q(x-r) + u^T R u \, dt, \quad Q \geq 0. \quad (4.30b)$$

Here  $A, B, C$  may be time-varying with piecewise smooth entries and  $R > 0$ . Our Hamiltonian is

$$H(t, x, x(t-\tau), u, \lambda) = (x-r)^T Q(x-r) + u^T R u + \lambda^T (Ax + Cx(t-\tau) + Bu). \quad (4.31)$$

Then the necessary conditions, which we derived at the beginning of this chapter, give the boundary value problem

$$\dot{x} = Ax + Cx(t-\tau) + Bu, \quad t \in [0, T] \quad (4.32a)$$

$$\dot{\lambda} = -2Q(x-r) - A^T \lambda - \chi_{[0, T-\tau]} C^T \lambda(t+\tau) \quad (4.32b)$$

$$\lambda(T) = 0 \quad (4.32c)$$

$$0 = 2Ru + B^T \lambda \quad (4.32d)$$

$$x(t) = \phi(t), \quad t \in [-\tau, 0], \quad (4.32e)$$

where  $x(t) = \phi(t)$  is the prehistory for our delay term and  $\chi$  is the characteristic function. Then from Eq. 4.32d we have

$$u(t) = -\frac{1}{2} R^{-1} B^T \lambda. \quad (4.33)$$

Substituting Eq. 4.33 into our dynamics Eq. 4.32 gives

$$\dot{x} = Ax + Cx(t-\tau) - \frac{1}{2} B R^{-1} B^T \lambda \quad (4.34a)$$

$$\dot{\lambda} = -2Q(x-r) - A^T \lambda - \chi_{[0, T-\tau]} C^T \lambda(t+\tau) \quad (4.34b)$$

$$\lambda(T) = 0 \quad (4.34c)$$

$$x(t) = \phi(t), \quad t \in [-\tau, 0]. \quad (4.34d)$$

#### 4.4.2 An Iterative Formulation for the Delayed State Optimal Control Problem

As we mention at the beginning of this section, we suppose the GPOPS-II<sub>m</sub> formulation neglects contributions from the delay terms in its derivative calculations, causing difficulty for the solver. We therefore introduce a new problem formulation to test the effect of these delay terms. In Chapter 2 we explain how to use GPOPS-II to solve a DDE using the iterative method waveform relaxation. We consider we may be able to obtain similarly successful results for the delayed optimal control problem using an iterative method, which we call waveform optimization and will refer to as GPOPS-II<sub>ow</sub>.

In this GPOPS-II<sub>ow</sub> formulation, the delay term is treated as a known function. At the first iteration, we solve the delayed optimal control problem given in Eq. 4.30 with delay  $\tau = 0$ . We then take  $\tau \neq 0$  and pass in the initial state and time as parameters, and use the initial state and time to interpolate our delay term in the "continuous" file.

Thus our subsequent function calls are

```
[time1, state1, control1] = wavef(a, c, tau, time0, state0),
```

where  $a, c$  are coefficients,  $\tau$  is the delay, and  $\text{time0}$  and  $\text{state0}$  are the values of the time grid and state obtained from the previous iteration. We include our numerical results for this approach in

Chapter 5. Below we motivate the BVP for the waveform optimization problem and then demonstrate that as the delay  $\tau$  approaches zero, the GPOPS-IIow formulation does not approach the undelayed formulation, causing the numerical approximation to converge to the incorrect solution.

#### 4.4.2.1 Case B: Boundary Value Problem for Waveform Optimization Problem

Waveform optimization is an iterative method, and so we treat the delay term like a known function interpolated from past values. We refer to this formulation as Case B. Then, on iteration  $N$ , we have the system

$$\dot{x}_N = Ax_N + Cx_{N-1}(t - \tau) + Bu_N \quad (4.35a)$$

$$J_N = \int_0^T (x_N - r)^T Q(x_N - r) + u_N^T R u_N dt. \quad (4.35b)$$

Our Hamiltonian is

$$H_N(t, x_N, x_{N-1}(t - \tau), u_N, \lambda_N) = (x_N - r)^T Q(x_N - r) + u_N^T R u_N + \lambda_N^T (Ax_N + Cx_{N-1}(t - \tau) + Bu_N). \quad (4.36)$$

Then we have the boundary value problem

$$\dot{x}_N = Ax_N + Cx_{N-1}(t - \tau) + Bu_N \quad (4.37a)$$

$$\dot{\lambda}_N = -2Q(x_N - r) - A^T \lambda_N \quad (4.37b)$$

$$0 = 2u_N + B^T \lambda_N \quad (4.37c)$$

$$\lambda_N(T) = 0 \quad (4.37d)$$

$$x_{N-1} = \phi(t), \quad t \in [-\tau, 0], \quad N = 0 \quad (4.37e)$$

and we have

$$u_N = -\frac{1}{2}R^{-1}B^T \lambda_N. \quad (4.38)$$

Substituting Eq. 4.38 into our dynamics gives

$$\dot{x}_N = Ax_N + Cx_{N-1}(t - \tau) - \frac{1}{2}BR^{-1}B^T \lambda_N \quad (4.39a)$$

$$\dot{\lambda}_N = -2Q(x_N - r) - A^T \lambda_N \quad (4.39b)$$

$$\lambda_N(T) = 0 \quad (4.39c)$$

$$x_{N-1} = \phi(t), \quad t \in [-\tau, 0], \quad N = 0. \quad (4.39d)$$

### 4.4.3 Error Estimates Between Problem Formulations

Now that we have established the form of the delayed boundary value problem for both GPOPS-II<sub>m</sub> and GPOPS-II<sub>ow</sub>, we seek to obtain bounds on the differences between the boundary value problems for our two delayed problem formulations and the undelayed case. We show as the delay  $\tau$  approaches zero the delayed boundary value problem for GPOPS-II<sub>m</sub> converges to the undelayed problem, Case C, enabling us to obtain an optimal solution using the GPOPS-II<sub>m</sub> formulation. However as the delay increases the error term grows, and GPOPS-II<sub>m</sub> is no longer able to find an optimal solution. In the case of GPOPS-II<sub>ow</sub> we demonstrate that the boundary value problem for GPOPS-II<sub>ow</sub> does not converge to the undelayed case; this explains why we do not see a more accurate solution using GPOPS-II<sub>ow</sub> as we decrease the value of the delay.

#### 4.4.3.1 Convergence of Delayed Problem

We now attempt to estimate the differences between the states  $x$  for each problem formulation and the multipliers  $\lambda$  for each problem formulation. We first consider Case A, the GPOPS-II<sub>m</sub> formulation in Eq. 4.34, and Case C in Eq. 3.9, the undelayed problem formulation. Assume our solutions to Eq. 4.34 and Eq. 3.9 exist and are piecewise smooth. We can use a Taylor expansion to obtain a first order approximation of  $x(t - \tau)$  in Eq. 4.34a and Eq. 3.9a. For our dynamics, let  $x_A, \lambda_A$  be the  $x, \lambda$  from Case A and  $x_C, \lambda_C$  be the  $x, \lambda$  from Case C. Then we have

$$\dot{x}_A = Ax_A + Cx_A(t) - \mathcal{O}(\tau) - \frac{1}{2}BR^{-1}B^T\lambda_A \quad (4.40)$$

and

$$\dot{x}_C = Ax_C + Cx_C(t) - \frac{1}{2}BR^{-1}B^T\lambda_C, \quad (4.41)$$

where the  $\mathcal{O}(\tau)$  term in Eq. 4.40 depends on  $\|x'_A\|_\infty$ .

Similarly, taking a first order Taylor approximation around  $\lambda(t + \tau)$  gives us

$$\dot{\lambda}_A = -2Q(x_A - r) - A^T\lambda_A + \chi_{[0, T-\tau]}(C^T\lambda_A(t) + \mathcal{O}(\tau)), \quad (4.42)$$

where the  $\mathcal{O}(\tau)$  in Eq. 4.42 depends on  $\|\lambda'_A\|_\infty$ . Now, the characteristic function  $\chi_{[0, T-\tau]}$  in Eq. 4.42 is one almost everywhere except on the interval  $[T - \tau, T]$ . Thus we can take  $\chi_{[0, T-\tau]} = 1$  with error term  $\hat{\mathcal{O}}(\tau)$ , so we obtain

$$\dot{\lambda}_A = -2Q(x_A - r) - A^T\lambda_A - C^T\lambda(t) + \hat{\mathcal{O}}(\tau) + \mathcal{O}(\tau)^2. \quad (4.43)$$

Then our equation for  $\dot{x}_A$  is as in Eq. 4.40, and our equation for  $\dot{\lambda}_A$  is as in Eq. 4.43. Let  $\delta x_{AC}$  be

$x_A - x_C$  and  $\delta\lambda_{AC} = \lambda_A - \lambda_C$ . Then we have the system

$$\dot{\delta x}_{AC} = A\delta x_{AC} + C\delta x_{AC} - \frac{1}{2}BR^{-1}B^T\delta\lambda_{AC} + \mathcal{O}(\tau) \quad (4.44a)$$

$$\dot{\delta\lambda}_{AC} = -2Q\delta x_{AC} - A^T\delta\lambda_{AC} - C^T\delta\lambda_{AC} + \hat{\mathcal{O}}(\tau) + \mathcal{O}(\tau)^2 \quad (4.44b)$$

with the boundary conditions  $\delta x_{AC}(0) = 0$  and  $\delta\lambda_{AC}(T) = 0$ .

In, general this is a system of the form

$$Z' = PZ + \mathcal{O}(\tau) \quad (4.45)$$

with constant linear boundary conditions where  $Z = [\delta x_{AC}; \delta\lambda_{AC}]$ . For the case  $\tau = 0$  we know the boundary value problem has a unique solution. Then for values of  $\tau$  near 0 we will have a solution and solutions will be continuous in  $\tau$  [2]. Thus as the delay  $\tau$  in Case A approaches 0, our solutions from Case A will approach those of Case C.

#### 4.4.3.2 Convergence of Waveform Optimization Problem

We now consider Case B, waveform optimization, and Case C, the undelayed problem. By observation, we know the iteration in Case B converges; suppose Eq. 4.39a and Eq. 4.39b converge to

$$\dot{x}_B = Ax_B + Cx_B(t - \tau) - \frac{1}{2}BR^{-1}B^T\lambda_B \quad (4.46a)$$

$$\dot{\lambda}_B = -2Q(x_B - r) - A^T\lambda_B. \quad (4.46b)$$

Again, assume our solutions exist and are piecewise smooth. Then we can obtain first order approximation of  $x(t - \tau)$  in Eq. 4.39a. We have

$$\dot{x}_B = Ax_B + Cx_B(t) - \tilde{\mathcal{O}}(\tau) - \frac{1}{2}BR^{-1}B^T\lambda_B, \quad (4.47)$$

where the  $\tilde{\mathcal{O}}(\tau)$  in Eq. 4.47 depends on  $\|x'_B\|_\infty$ . Define  $x_A - x_B$  as  $\delta x_{AB}$  and let  $\lambda_A - \lambda_B$  be  $\delta\lambda_{AB}$ .

Then we have the system

$$\dot{\delta x}_{AB} = A\delta x_{AB} + C\delta x_{AB} - \frac{1}{2}BR^{-1}B^T\delta\lambda_{AB} + \mathcal{O}(\tau) \quad (4.48a)$$

$$\dot{\delta\lambda}_{AB} = -2Q\delta x_{AB} - A^T\delta\lambda_{AB} - C^T\lambda_A + \mathcal{O}(\tau) + \mathcal{O}(\tau)^2. \quad (4.48b)$$

We can see that unlike Eq. 4.44b, Eq. 4.48b contains an extra term in the multiplier  $\lambda$ . The equation in our multiplier Eq. 4.46b is missing a  $C^T\lambda_B$  term because in waveform optimization we treat the delay term as a known function. Thus as we decrease the delay in Eq. 4.39 it does not approach the

solution for the zero delay case as given in Eq. 3.9.

## CHAPTER

# 5

## NUMERICAL RESULTS

Here we present the numerical results of our extensive computational experimentation. As we explain in our discussion of GPOPS-II in Chapter 3, GPOPS-II adapts the number of mesh intervals and the order of the interpolating polynomial. In our case, we find that allowing GPOPS-II to vary the order of the polynomial does not substantially increase the accuracy of our results but does increase run time. Using the `hp-PattersonRao` mesh method, GPOPS-II assumes a degree of smoothness in the solution when it increases the order of the polynomial. Many of our states and controls have discontinuities at the value of the delay, and reduced smoothness at multiples of the delay, so those assumptions of smoothness do not hold. Therefore in many of the examples we discuss, we fix the order of the interpolating polynomial. At the end of the chapter, we present an analytical analysis based on our numerical results.

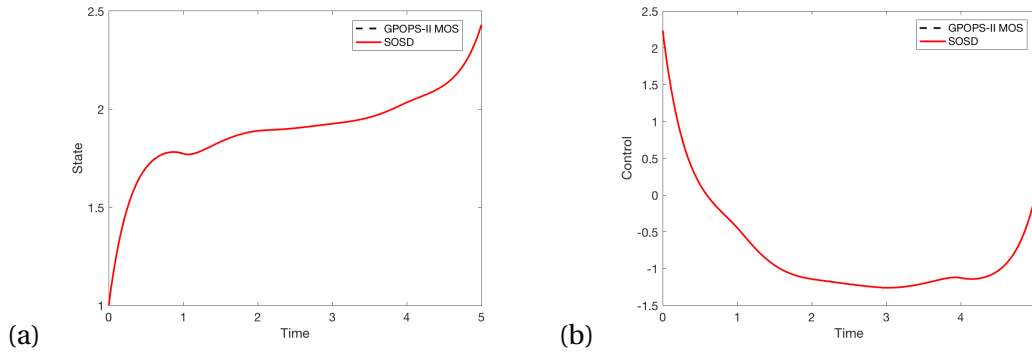
### 5.1 Use of GPOPS-II as a Delay Differential Equation Solver

Before using GPOPS-II to solve a delayed optimal control problem, we first seek to determine if GPOPS-II can accurately resolve delayed dynamics. We therefore use GPOPS-II as an integrator to solve several delayed differential equations without a control, as well as a boundary value problem solver on the necessary conditions and MOL formulations. We utilize several different approaches: the method of steps (MOS), GPOPS-II and waveform relaxation (GPOPS-IIw), GPOPS-II with a linear interpolation of the delay term (GPOPS-IIm), and the method of lines (MOL).

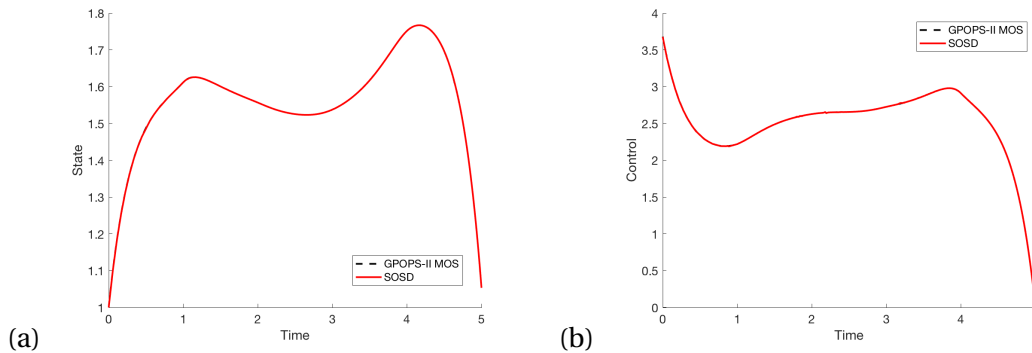
### 5.1.1 Using the MOS Formulation

Here we present our results for using GPOPS-II on the MOS example Eq. 4.19 given in Chapter 4. We allow three mesh iterations, fix our polynomial order at  $P = 3$ , and set our tolerances to  $1e-07$ . We include solution plots for the stable  $c = -1.2$  and unstable  $c = +1.2$ . We compare our MOS solutions to data from SOSD in Fig. 5.1 and Fig. 5.2.

We obtain  $J = 43.421409$  from MOS and  $J = 43.421408$  from SOSD, a difference of  $3.2242e-06\%$  for  $c = -1.2$ . For  $c = +1.2$  we obtain  $J = 8.0179012$  from MOS and  $J = 8.017899$  from SOSD, a difference of  $2.7439e-05\%$ . Our results demonstrate that using GPOPS-II with MOS can be a highly effective technique. We are able to use GPOPS-II with MOS to validate results from other methods later in this chapter.



**Figure 5.1** MOS (a) State and (b) control from Eq. 4.19 with  $c = +1.2$ ,  $\tau = 1$ .



**Figure 5.2** MOS (a) State and (b) control from Eq. 4.19 with  $c = -1.2$ ,  $\tau = 1$ .

### 5.1.2 Solving a DDE Using Waveform Relaxation

While MOS is a useful technique, it can be impractical for systems where the delay is short relative to the time interval. Therefore we now consider behavior of GPOPS-II as it solves a simple delay differential equation (DDE) using waveform relaxation, GPOPS-IIw [11]. Consider the problem

$$\dot{x} = ax + cx(t - \tau), \quad (5.1)$$

where  $a$  and  $c$  are constants and  $\tau$  is the delay. We call GPOPS-II recursively and compare our results to MATLAB's built in DDE solver, dde23, as well as to waveform relaxation using the MATLAB solver ode45.

#### 5.1.2.1 Example

We take  $a = -.5$ ,  $c = -1.2$ , and  $\tau = 1.0$  to obtain the problem

$$\dot{x} = -.5x - 1.2x(t - 1.0). \quad (5.2)$$

For both GPOPS-II and ode45 we take five iterations. We allow GPOPS-II one mesh iteration on the first function call and four mesh iterations during subsequent function calls. We set our tolerances to  $1e-06$  for the first iteration and  $1e-09$  for the remaining four. Further iterations do not improve the size of the residual and are therefore omitted. Fig. 5.4 shows that while GPOPS-II and ode45 are able to solve the problem to the same order of accuracy,  $1e-05$ , ode45 is slightly more accurate.

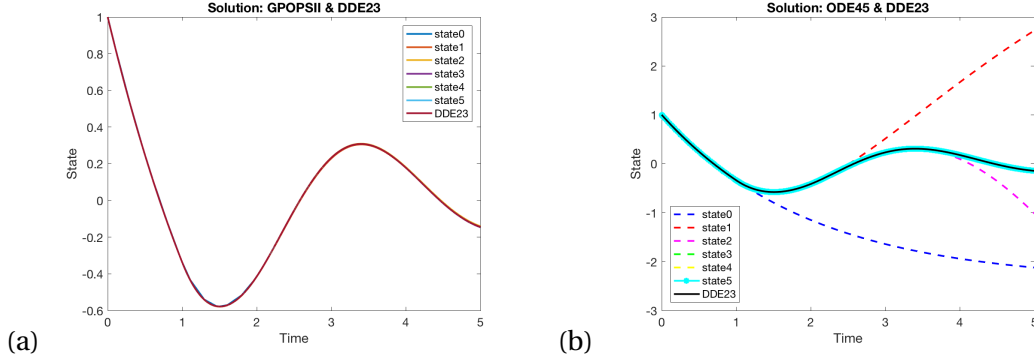
The residuals are nearly identical in shape, however GPOPS-II's residual in Fig. 5.4(a) has a great deal of oscillation on  $[0, 2]$ . Interestingly, Fig. 5.3 demonstrates that GPOPS-II converges almost immediately, while ode45 takes a full three iterations before it converges. Using GPOPS-IIw we are able to accurately solve a DDE, without needing to rewrite the DDE to eliminate the delay as in MOS.

#### 5.1.2.2 Comparison of GPOPS-IIw as a DDE Solver to Matlab DDE Solver

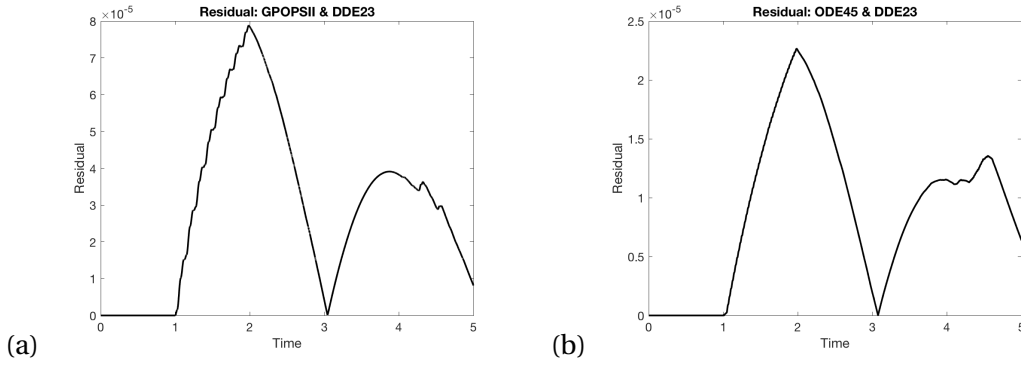
Both MOS and GPOPS-IIw are successful at solving a DDE, however we are also interested in using GPOPS-II directly as an integrator on a DDE. We now compare the ability of GPOPS-IIw to solve a basic delay differential equation (DDE) to that of the MATLAB software package dde23. We treat the solution from dde23 as the "true" solution to the DDE. We have four examples: one asymptotically stable Eq. 5.3a, one barely asymptotically stable Eq. 5.4a, one completely unstable Eq. 5.5a, and one slightly unstable Eq. 5.6a:

$$\dot{x} = -2x + x(t - 2) \quad (5.3a)$$

$$\lambda = -0.2731 \quad (5.3b)$$



**Figure 5.3** (a) Solutions obtained by GPOPS-II and dde23 and (b) solutions obtained by ode45 and dde23 to Eq. 5.2 for  $c = -1.2$  and  $\tau = 1.0$ .



**Figure 5.4** (a) Residuals from GPOPS-IIw and dde23 and (b) residuals from ode45 and dde23 for Eq. 5.2 with  $c = -1.2$  and  $\tau = 1.0$ .

$$\dot{x} = -3x + 2.8x(t - .3) \quad (5.4a)$$

$$\lambda = -0.1084 \quad (5.4b)$$

$$\dot{x} = -3x + 3.4x(t - .3) \quad (5.5a)$$

$$\lambda = 0.2011 \quad (5.5b)$$

$$\dot{x} = -3x + 3.2x(t - .3) \quad (5.6a)$$

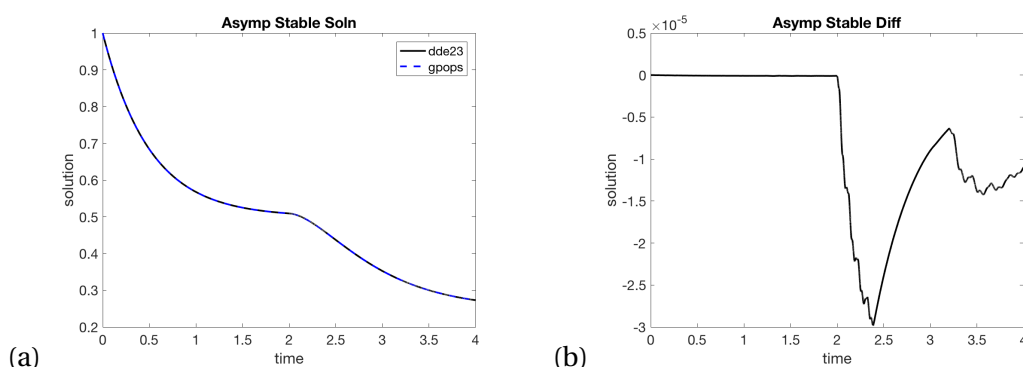
$$\lambda = 0.1029. \quad (5.6b)$$

To find the roots of our delay differential equations we use the root finding package developed for MATLAB by Zhen Wu and Wim Michiels [82]. We note the software gives both an approximate and corrected eigenvalue. We use the approximate eigenvalues. To compare GPOPS-II and dde23 we first solve Eq. 5.3a-Eq. 5.6a using the GPOPS-II approach we describe in Chapter 4. We then solve Eq. 5.3a-Eq. 5.6a in dde23 and output the solution onto the time grid points we obtain from

GPOPS-II. This enables us to plot the two solutions, as well as the difference between them, to assess how well GPOPS-II finds the solution to the DDE relative to dde23. In all cases, we set GPOPS-II to a maximum of three iterations. We set all our tolerances in both packages to  $10^{-7}$ . We initially solve the equations on a shorter interval,  $[0,4]$ , and later repeat our comparison on longer intervals.

### 5.1.2.2.1 Solutions on a Short Interval

Interestingly, the solutions of GPOPS-II and dde23 agree very well in all cases for the short interval  $[0,4]$ . Our solution plots from GPOPS-II in Fig. 5.5-Fig. 5.8 are visually indistinguishable from dde23. We expect that GPOPS-II would be able to solve the asymptotically stable equation Eq. 5.3a very well but thought the software may struggle when it came to the unstable equation Eq. 5.5a. However we can see in Fig. 5.7(b) and Fig. 5.8(b) that the two solutions differ on the order  $10^{-5}$ .



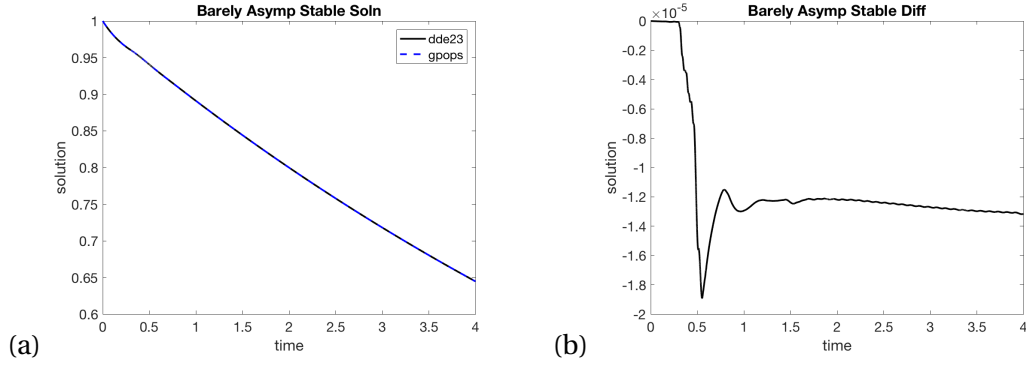
**Figure 5.5** (a) Asymptotically stable Eq. 5.3a solution in dde23 and GPOPS-II and (b) difference of solutions on  $[0,4]$  with tolerances  $10^{-7}$ .

### 5.1.2.2.2 Solutions on a Longer Interval

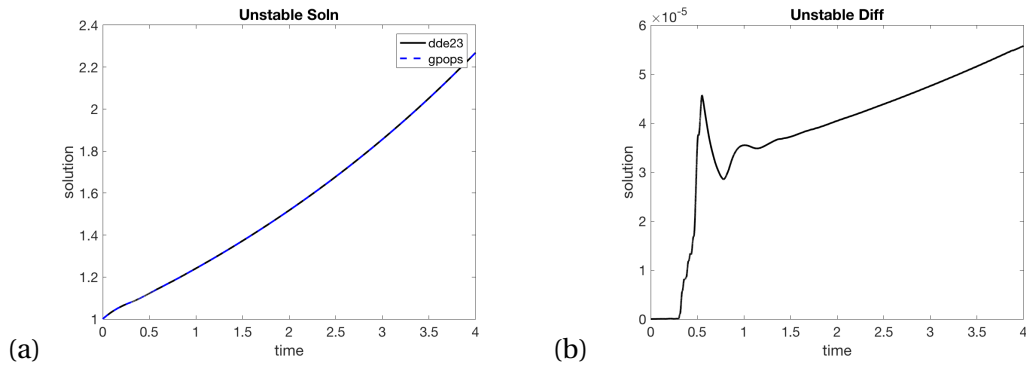
Next we examine the behavior of GPOPS-II relative to dde23 on longer intervals. We begin with  $[0, T] = [0, 20]$  and gradually increase the length of the interval to observe the change in behavior. The two software solutions agreed very well until we reach a final time of  $T = 23.95$ . Attempting to take a longer time interval is too computationally expensive for GPOPS-II to find a solution.

We can see from Fig. 5.9 and Fig. 5.10 that GPOPS-II is still able to approximate the solution very well. Surprisingly, as shown in Fig. 5.12, GPOPS-II even does well with the approximation of the slightly unstable example in Eq. 5.6a. However we can see in Fig. 5.11 that GPOPS-II is unable to successfully approximate the very unstable example Eq. 5.5a on a longer interval.

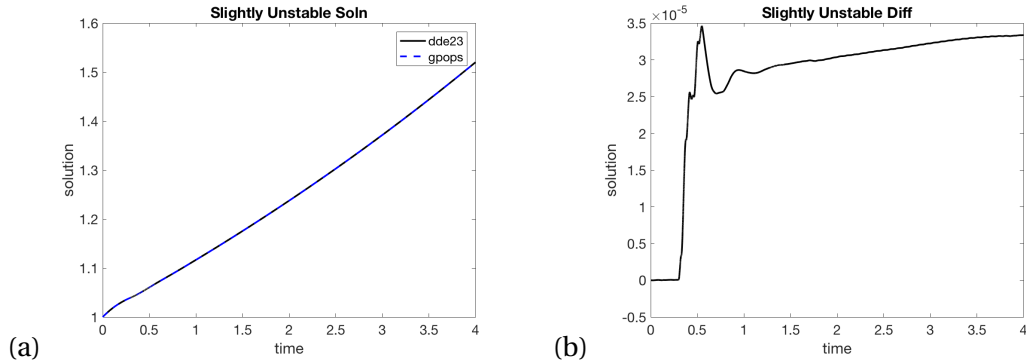
We now increase all of our tolerances to  $1e-08$  and re-run several examples to see if GPOPS-II's



**Figure 5.6** (a) Barely asymptotically stable Eq. 5.4a solution in dde23 and GPOPS-IIIm and (b) difference of solutions on  $[0,4]$  with tolerances  $10^{-7}$ .

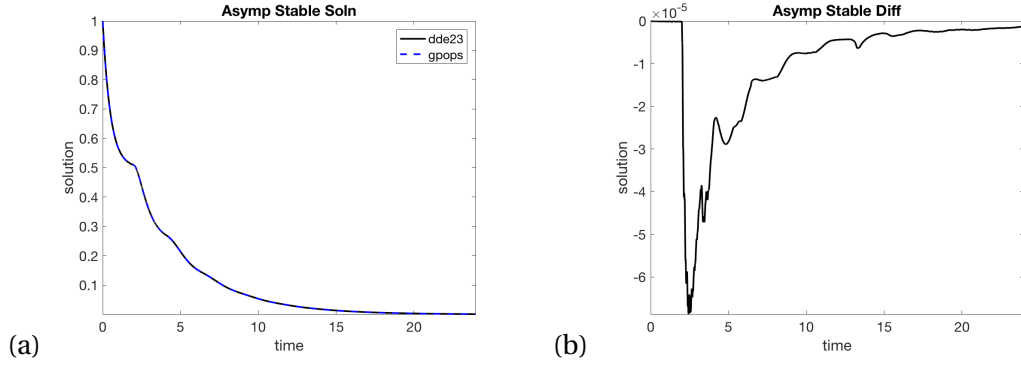


**Figure 5.7** (a) Unstable Eq. 5.5a solution in dde23 and GPOPS-IIIm and (b) difference of solutions on  $[0,4]$  with tolerances  $10^{-7}$ .

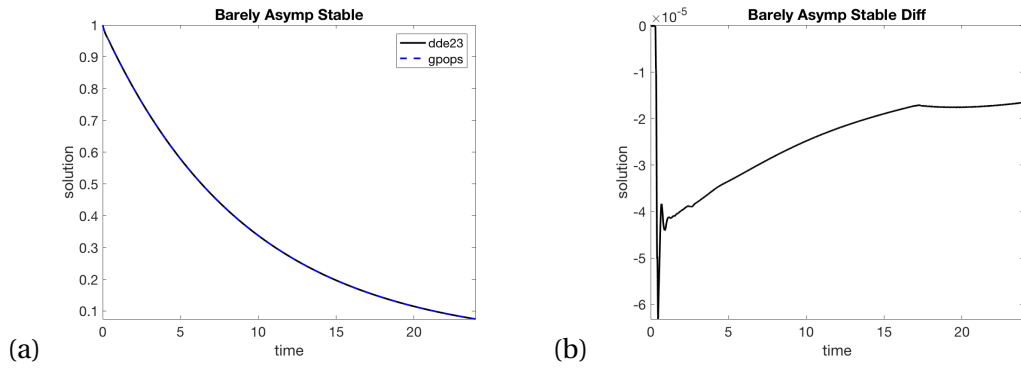


**Figure 5.8** (a) Slightly unstable Eq. 5.6a solution in dde23 and GPOPS-IIIm and (b) difference of solutions on  $[0,4]$  with tolerances  $10^{-7}$ .

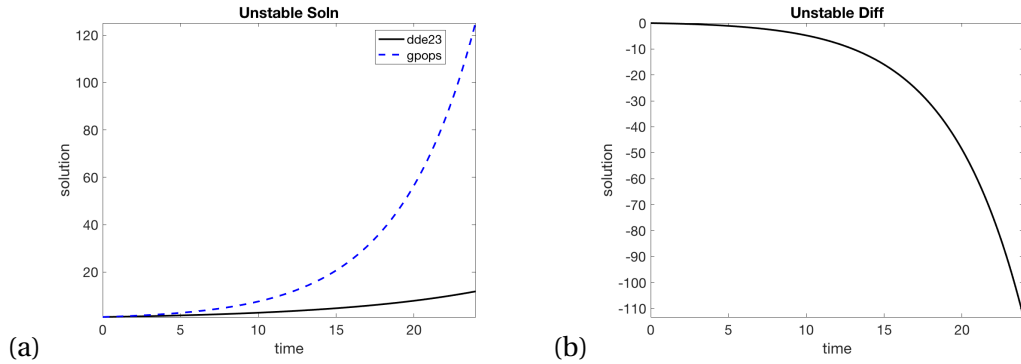
approximation improves. We begin with the asymptotically stable example in Eq. 5.3a. We can see from Fig. 5.13(b) that increasing our tolerance improves our approximation. Our error is now order



**Figure 5.9** (a) Asymptotically stable Eq. 5.3a solution in dde23 and GPOPS-II and (b) difference of solutions on  $[0, 23.95]$  with tolerances  $10^{-7}$ .

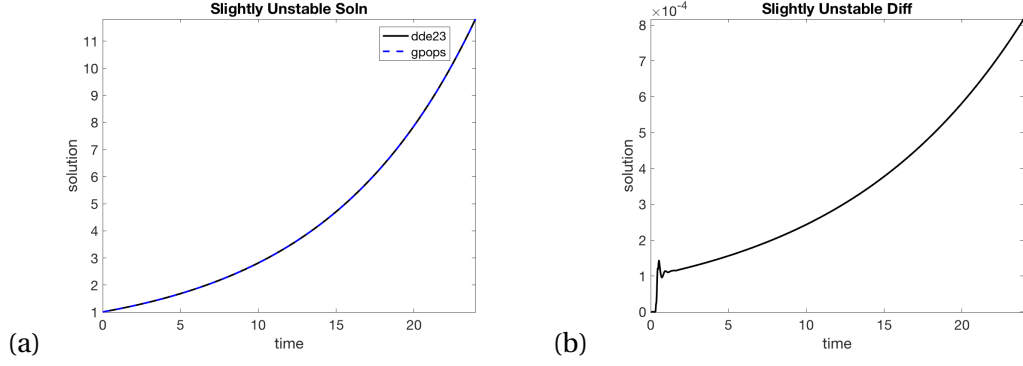


**Figure 5.10** (a) Barely asymptotically Eq. 5.4a solution in dde23 and GPOPS-II and (b) difference of solutions on  $[0, 23.95]$  with tolerances  $10^{-7}$ .

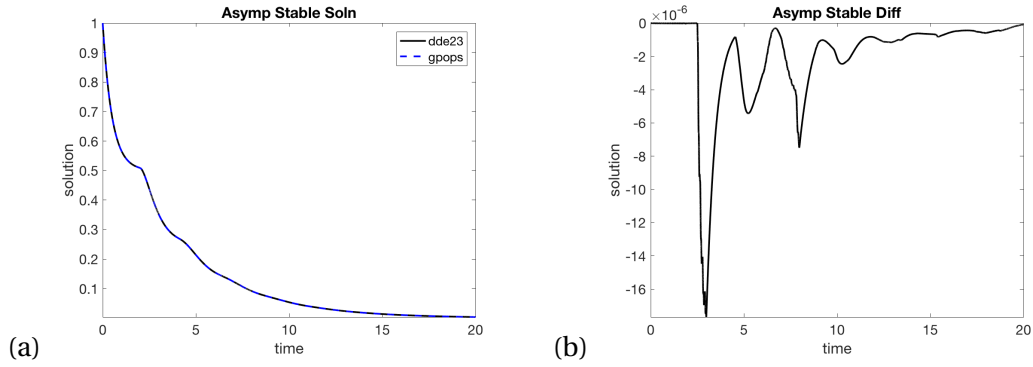


**Figure 5.11** (a) Unstable Eq. 5.5a solution in dde23 and GPOPS-II and (b) difference of solutions on  $[0, 23.95]$  with tolerances  $10^{-7}$ .

$1e-06$  as opposed to  $1e-05$ . We now re-run the unstable example in Eq. 5.5a to see if increasing our tolerance will improve the approximation. We can see from Fig. 5.14 that increasing our tolerance



**Figure 5.12** (a) Slightly unstable Eq. 5.6a solution in dde23 and GPOPS-IIIm and (b) difference of solutions on  $[0,23.95]$  with tolerances  $10^{-7}$ .



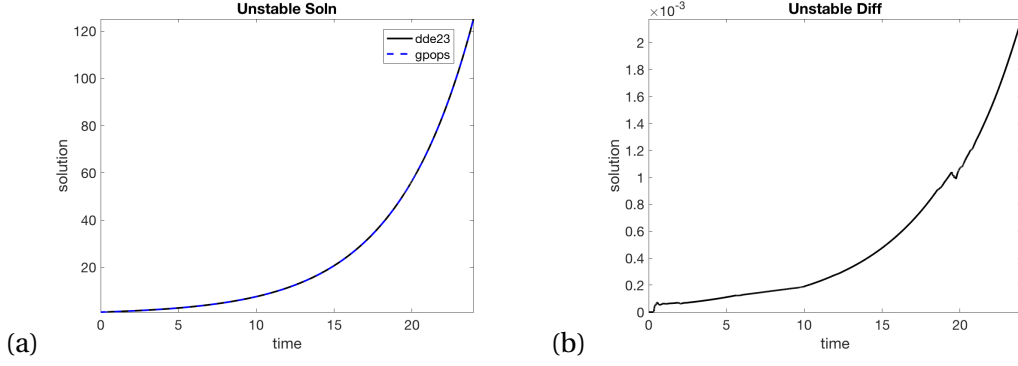
**Figure 5.13** (a) Asymptotically stable Eq. 5.4a solution from dde23 and GPOPS-IIIm and (b) difference of solutions on  $[0,20]$  for tolerance  $10^{-8}$ .

dramatically improves our approximation. Previously in Fig. 5.11 the GPOPS-IIIm solution began to diverge completely from the dde23 solution the error was much greater than one. Now our error is of the order  $1e-03$ . Our results indicate that on all but very unstable problems on longer time intervals, GPOPS-IIIm is an effective integrator of delayed dynamics.

### 5.1.3 Solving the Delayed Necessary Conditions with GPOPS-IIIm

Our previous results show GPOPS-IIIm can be a very effective integrator of delayed dynamics. We now examine the use of GPOPS-IIIm as a boundary value problem (BVP) solver on the necessary conditions. We use the necessary conditions in Eq. 4.34 to obtain the two-point boundary value problem with

$$a = -0.5, \quad c = -1.2, \quad b = 1, \quad \tau = 1 \quad (5.7)$$



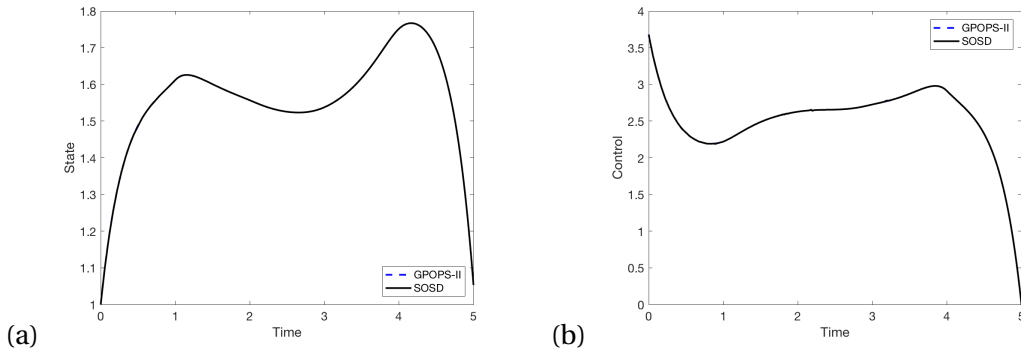
**Figure 5.14** (a) Unstable solution Eq. 5.5a from dde23 and GPOPS-II and (b) difference of solutions on  $[0, 23.95]$  for tolerance  $10^{-8}$ .

and cost

$$J = \int_0^5 10(x-2)^2 + u^2 dt. \quad (5.8)$$

We fix the order of the approximating polynomial at  $P = 3$ . GPOPS-II struggles to solve Eq. 5.7 for higher tolerances, therefore we set our mesh tolerance at  $1e-05$ , and keep our IPOPT tolerance at  $1e-07$ . We allow four mesh iterations and compare our results to the solver SOSD from the Sparse Optimization Suite (SOS) which we took to be "truth" for this experiment [13]. GPOPS-II obtains a cost of  $J = 43.4212447$ , which agrees very well with the cost  $J = 43.421408$  obtained by SOSD.

We plot our results in Fig. 5.15, which show the state and control obtained by solving the necessary conditions agree very well with those obtained by SOSD. These results demonstrate we are able to successfully solve this OCP with state delay in GPOPS-II using the necessary condition formulation, as well as demonstrating GPOPS-II can be an effective delay BVP solver in MATLAB.



**Figure 5.15** (a) State and (b) control for necessary conditions Eq. 4.34 with  $\tau = 1$ ,  $c = -1.2$ .

### 5.1.3.1 Nonlinear Example Using the Necessary Conditions

Given GPOPS-II is an effective delay BVP solver on linear dynamics we test if GPOPS-II performs equally well on nonlinear dynamics, including those with control constraints. We consider the modified Van der Pol Oscillator from [53]:

$$\min J = \frac{1}{2} \int_0^5 (x_1^2(t) + x_2^2(t) + u^2(t)) dt \quad (5.9)$$

subject to

$$\dot{x}_1 = x_2(t) \quad (5.10a)$$

$$\dot{x}_2 = -x_1(t) + (1 - x_1^2(t))x_2(t-1) + u(t) \quad (5.10b)$$

$$|u(t)| \leq 0.75 \quad (5.10c)$$

$$x_1(0) = 1, \quad (5.10d)$$

$$x_2(t) = 0, \quad t \in [-1, 0] \quad (5.10e)$$

$$x_1(5) = -1 \quad (5.10f)$$

$$x_2(5) = 0. \quad (5.10g)$$

The necessary conditions for this problem are

$$\dot{x}_1 = x_2 \quad (5.11a)$$

$$\dot{x}_2 = -x_1 + (1 - x_1^2)x_2(t-1) + u \quad (5.11b)$$

$$\dot{\lambda}_1 = -x_1 - (-\lambda_2 - 2\lambda_2 x_1 x_2(t-1)) \quad (5.11c)$$

$$\dot{\lambda}_2 = -x_2 - \lambda_1 - \chi_{[0,4]}(t) (\lambda_2(t+1)(1 - x_1^2(t+1))), \quad (5.11d)$$

where the Hamiltonian to be minimized is

$$\min_u H = \frac{1}{2} (x_1^2 + x_2^2 + u^2) + \lambda_1 x_2 + \lambda_2 (-x_1 + (1 - x_1^2)x_2(t-1)) + \lambda_2 u \quad (5.12)$$

subject to

$$|u| \leq .75. \quad (5.13)$$

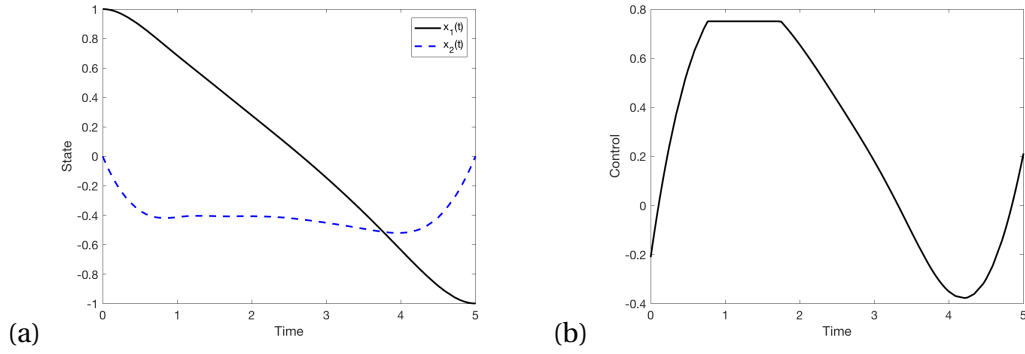
Then to minimize Eq. 5.12 over  $u$  we have

$$\min_u \frac{1}{2} u^2 + \lambda_2 u. \quad (5.14)$$

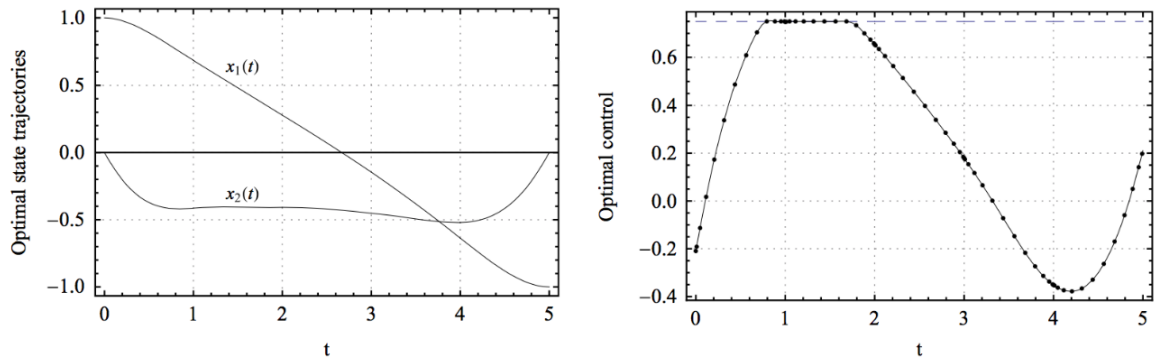
This implies

$$u = \begin{cases} .75 & \lambda_2 < -.75 \\ -\lambda_2 & |\lambda_2| \leq .75 \\ -.75 & \lambda_2 > .75. \end{cases} \quad (5.15)$$

We keep our IPOPT tolerance at  $1e-07$  and our mesh tolerance at  $1e-05$ . We take  $P = 3$ , allow three mesh iterations, and set the initial number of mesh intervals to  $N = 20$ . We obtain cost  $J = 1.9866728$  from GPOPS-II<sub>m</sub>, which agrees with the values presented in [53] out to two significant figures. Our results in Fig. 5.16 are nearly graphically identical to those given in Fig. 5.17 from [53]; we are able to use GPOPS-II<sub>m</sub> on a delayed BVP with nonlinear dynamics.



**Figure 5.16** (a) State and (b) control the modified Van der Pol oscillator with cost  $J = 1.9866728$ .



**Figure 5.17** Optimal state and control for modified Van der Pol Oscillator from [53].

#### 5.1.4 GPOPS-II and the Method of Lines

We see that GPOPS-II can be an effective delay BVP solver on the necessary conditions. For more complicated systems, the necessary condition formulation may not be practical to implement. We now consider using the method of lines (MOL) to obtain a boundary value problem from a delayed OCP.

Consider the system

$$\dot{x} = -.5x - 1.2x(t-1) + u \quad (5.16a)$$

$$J = \int_0^T 10(x-2)^2 + u^2 dt, \quad (5.16b)$$

where the delay is  $\tau = 1$  and  $T = 5$ . We can rewrite Eq. 5.16 as a system of PDEs with respect to time and a delayed variable,  $\theta$ . Then we have

$$U_t = U_\theta, \quad -1 \leq \theta \leq 0, \quad t > 0 \quad (5.17a)$$

$$U(0, \theta) = \phi(\theta), \quad -1 \leq \theta \leq 0 \quad (5.17b)$$

$$U_t = -0.5U(t, 0) - 1.2U(t, -1) + u(t), \quad t \geq 0 \quad (5.17c)$$

$$J_t = 10(U(t, 0) - 2)^2 + u(t)^2, \quad t \geq 0, \quad (5.17d)$$

where  $\phi(\theta)$  is our prehistory,  $\phi(\theta) = 1$ .

Now we can use the method of lines, where we use a centered difference to compute the partial derivatives in  $\theta$ , and step size  $h$  to rewrite Eq. 5.17 as a system of ordinary differential equations (ODEs). We have

$$U'_0(t) = 1/2h(-U_2(t) + 4U_1(t) - 3U_0(t)) + \tilde{\mathcal{O}}(h^3) \quad (5.18a)$$

$$U'_k(t) = 1/2h(U_{k+1}(t) - U_{k-1}(t)) + \mathcal{O}(h^3), \quad k = 1 \dots N \quad (5.18b)$$

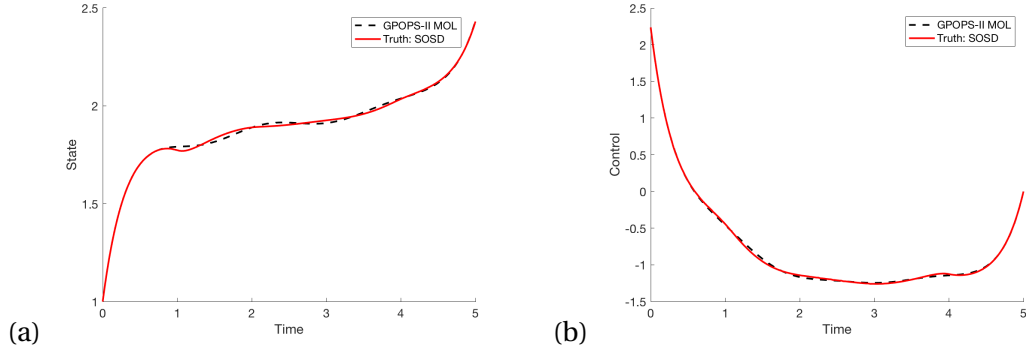
$$U'_N = -0.5U_N(t) - 1.2U_0 + u(t), \quad k = N \quad (5.18c)$$

$$J'_N = 10(U_N(t) - 2)^2 + u(t)^2, \quad k = N, \quad (5.18d)$$

where  $U'_0$  is as derived in Eq. 2.24 - Eq. 2.27.

We use the MOL approach given above to solve Eq. 5.16 with  $N = 5$  using GPOPS-II and plot our results in Fig. 5.18. For our purposes, we take the solution obtained by SOSD as "truth". We set our tolerances at  $1e-07$  and allow three mesh iterations. For Eq. 5.18, GPOPS-II obtains a cost of  $J = 43.416160$  while SOSD obtains cost  $J = 43.421408$ , a 0.0121% difference. We are able to use MOL with GPOPS-II to solve a delay problem accurately with a relatively coarse mesh.

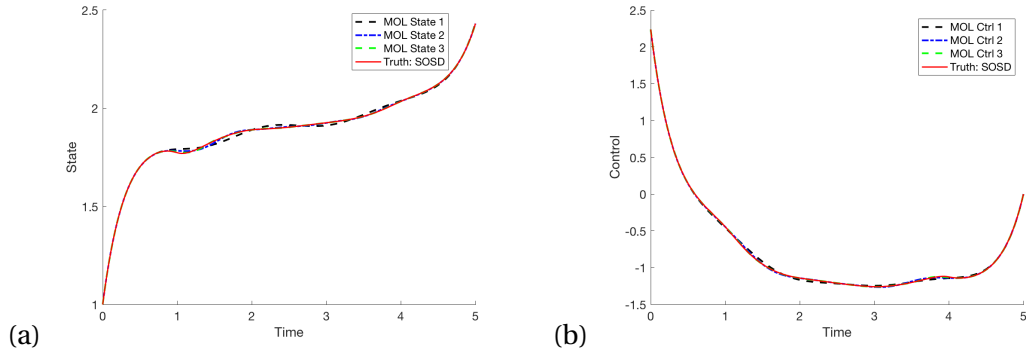
In Chapter 2 we show the error in our central difference approximation is  $\mathcal{O}(h^2)$  while the error in our one-sided Taylor approximation at the boundary is  $\mathcal{O}(h^3)$ . We now test the numerical rate of



**Figure 5.18** (a) State and (b) control obtained for Eq. 5.16 using GPOPS-II and MOL with  $\tau = 1$  and  $c = -1.2$ .

convergence with a mesh refinement study on the optimal control problem given in Eq. 5.16. The results in Table 5.1 demonstrate that refining the mesh does decrease the difference in our costs. For  $N = 9$  and above, the difference between MOL and SOSD is of order  $10^{-4}$ .

The plots in Fig. 5.19 indicate similarly promising results. We test three grid sizes:  $N = 5, 10, 20$  and plot the resulting solutions against those from SOSD. We can see that as we increase  $N$ , the states and controls from MOL in Fig. 5.19 approach those of SOSD. In fact, the state and control for  $N = 20$  are visually indistinguishable from SOSD. These results indicate that we may expect refining our grid will give a more accurate approximation of the control and trajectory. While using MOL on larger or more complicated problems may result in a very high dimensional problem, for simple examples like the problem in Eq. 5.18 using GPOPS-II and MOL can be a good choice.



**Figure 5.19** (a) State and (b) control obtained for Eq. 5.16 with grid size  $N = 5, 10, 20$  and  $\tau = 1$ ,  $c = -1.2$ .

**Table 5.1** Mesh refinement study using MOL and GPOPS-II vs SOSD for Eq. 5.16.

$N$	Cost $J$	$ J_{MOL} - J_{SOSD} $
5	43.4161601	0.0052
7	43.4225584	0.0012
9	43.4209837	$4.2430e-04$
10	43.4205858	$8.2220e-04$
15	43.4206727	$7.3530e-04$
20	43.4210928	$3.1520e-04$

## 5.2 Use of GPOPS-II to Solve Delayed State Optimal Control Problems

In the previous sections we utilize GPOPS-II to solve delay differential equations and delayed boundary value problems. We now shift our focus to the use of GPOPS-II directly on OCPs with delays in the state. We demonstrate when we are likely to obtain a reasonable answer from both GPOPS-II<sub>m</sub> and GPOPS-II<sub>ow</sub>.

### 5.2.1 Use of GPOPS-II with Linear Interpolation (GPOPS-II<sub>m</sub>)

We begin our discussion with GPOPS-II<sub>m</sub>. In order to assess the performance of GPOPS-II<sub>m</sub> we first consider the delayed dynamics

$$\dot{x} = -0.5x + cx(t - \tau) + u, \quad x(0) = 1 \quad (5.19)$$

subject to the quadratic cost given above in Eq. 5.8. We solve the optimal control problem using GPOPS-II<sub>m</sub> for several values of  $\tau$  and  $c$ . We then compare our results to those obtained by control parameterization and SOSD. We again parameterize the control with cubic splines and  $N = 28$  subintervals. We use the MATLAB solver dde23 to solve the dynamics and MATLAB's nonlinear optimizer fmincon to optimize our cost  $J$  over the control,  $u$ . In all cases we set our tolerances to  $10^{-7}$ . We allow GPOPS-II<sub>m</sub> to take three mesh iterations. Our initial value for  $u$  was  $u(0) = 6 * (1 : N + 1)^{-1}$ .

Initially, we begin with the stable case  $c = -1.2$  and consider a small delay of  $\tau = 0.05$  to obtain

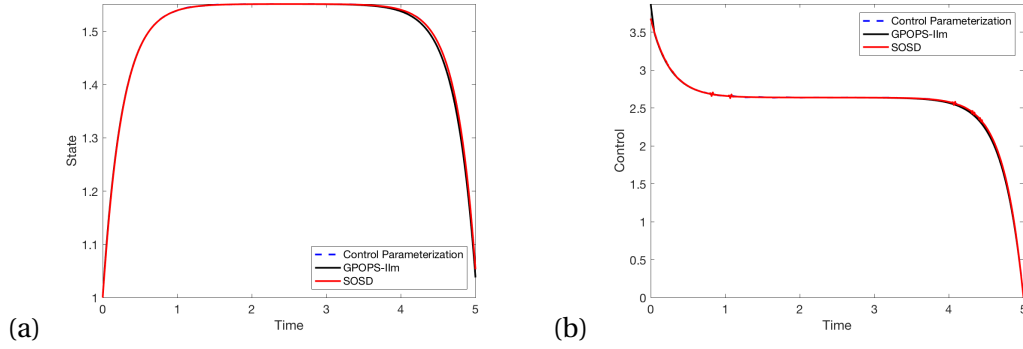
$$\dot{x} = -0.5x - 1.2x(t - 0.05) + u, \quad x(0) = 1. \quad (5.20)$$

We plot our results below in Fig. 5.20. We can see the results for the state agree very well. However, GPOPS-II<sub>m</sub> misses the slight spikes in the control in Fig. 5.20(b). We record the cost in Table 5.2 and can see that GPOPS-II<sub>m</sub> agrees with SOSD to two digits while our results for the cost using control parameterization agrees with SOSD to three digits, indicating our control parameterization code is producing a low order solution.

We now consider a longer delay of  $\tau = 1$  and keep  $c = -1.2$  to obtain the dynamics

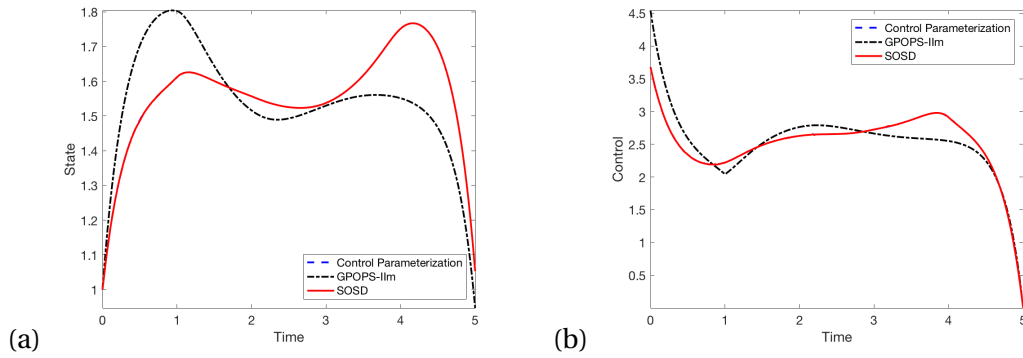
$$\dot{x} = -0.5x - 1.2x(t - 1) + u, \quad x(0) = 1. \quad (5.21)$$

We plot our results in Fig. 5.21. We can see the that difference between GPOPS-II<sub>m</sub> and the other two solvers is even more pronounced. The state in Fig. 5.21(a) appears to be a reflection of the states obtained from SOSD and control parameterization. GPOPS-II<sub>m</sub> also introduces a corner into the control in Fig. 5.21(b) approximately where the delay takes effect at  $t = 1$ . While our costs from SOSD and control parameterization in Table 5.2 agree relatively well the cost from GPOPS-II<sub>m</sub> is



**Figure 5.20** (a) State and (b) control obtained by GPOPS-IIIm, control parameterization, SOSD for  $c = -1.2$ ,  $\tau = 0.05$  with cost Eq. 5.8.

significantly higher.

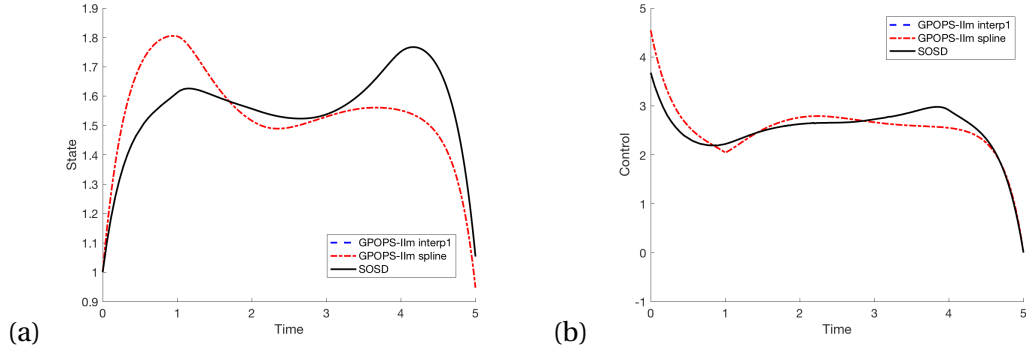


**Figure 5.21** (a) State and (b) control obtained by GPOPS-IIIm, control parameterization, SOSD for  $c = -1.2$ ,  $\tau = 1$  with cost Eq. 5.8.

We note in Chapter 4 that we utilize a linear interpolation of the delay term in GPOPS-IIIm. We test if using a higher order interpolation, the MATLAB `spline` function, improves our results for Eq. 5.21. We obtain cost  $J = 44.664865$  using `spline`, a 2.8233% difference with SOSD. We obtain  $J = 44.664151$  using `interp1`, a 2.8217% difference with SOSD. We can see from Fig. 5.22 there is no visual difference between the solutions produced using `spline` and those produced with `interp1`.

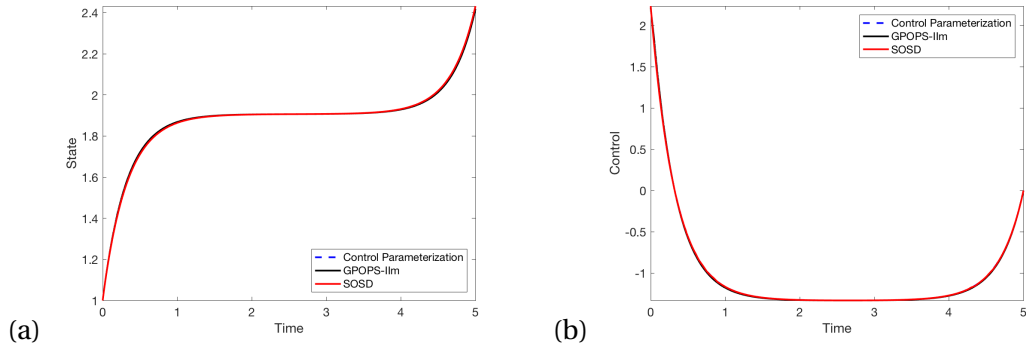
We consider the case where  $c = +1.2$ . Taking  $\tau = 0.05$  we have

$$\dot{x} = -0.5x + 1.2x(t - 0.05) + u, \quad x(0) = 1. \quad (5.22)$$



**Figure 5.22** (a) State and (b) controls from GPOPS-IIIm using linear interpolation and splines on the delay term, compared to SOSD for  $c = -1.2$ ,  $\tau = 1$  with cost Eq. 5.8.

We compare the performance of GPOPS-IIIm to both control parameterization and SOSD. We can see from Fig. 5.23 that GPOPS-IIIm again performs well for a small delay. Interestingly the small spike in the control present for the  $c = -1.2$  case in Eq. 5.20 is no longer present for the  $c = +1.2$  case. Our costs in Table 5.2 show that GPOPS-IIIm only agrees with SOSD to two digits while control parameterization agrees with SOSD to three digits. We note that our cost, 3.33, encourages our optimizers to keep the state close to 2. However taking  $c$  positive resulted in a final state value of approximately 2.4 in Fig. 5.23(a) as opposed to the final state value of 1.8 obtained using a negative  $c$  value in Fig. 5.20(a).

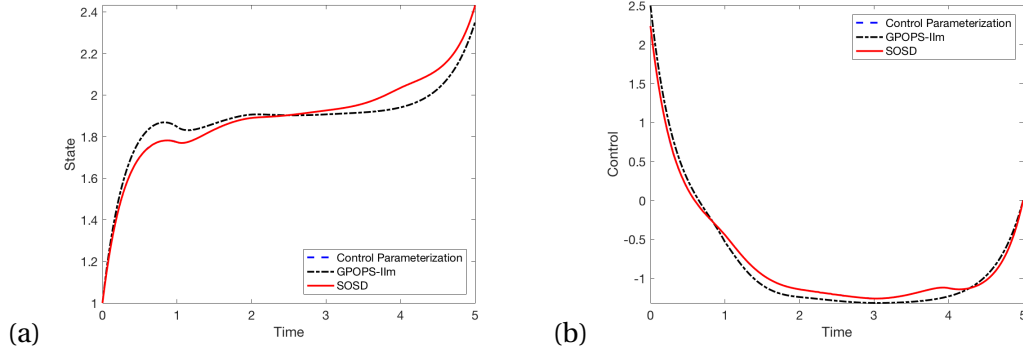


**Figure 5.23** (a) State and (b) control obtained by GPOPS-IIIm, control parameterization, SOSD for  $c = 1.2$ ,  $\tau = 0.05$  with cost Eq. 5.8.

We increase  $\tau$  to 1 and again take  $c = +1.2$  to obtain the dynamics

$$\dot{x} = -0.5x + 1.2x(t-1) + u, \quad x(0) = 1. \quad (5.23)$$

Fig. 5.24 shows that control parameterization and GPOPS-IIIm perform much better for Eq. 5.23 as opposed to the  $c = -1.2$  case in Eq. 5.21. As with the  $\tau = 0.5$  case, we no longer see a corner in the control from GPOPS-IIIm in Fig. 5.24(b) that was present in Fig. 5.21(b). Additionally the state from GPOPS-IIIm does not appear to be a reflection of the states obtained by control parameterization or SOSD. Our costs in Table 5.2 continued to agree relatively well, with GPOPS-IIIm returning the highest value of the cost at  $J = 8.2505467$  and SOSD returning the lowest value of the cost at  $J = 8.0178992$ .



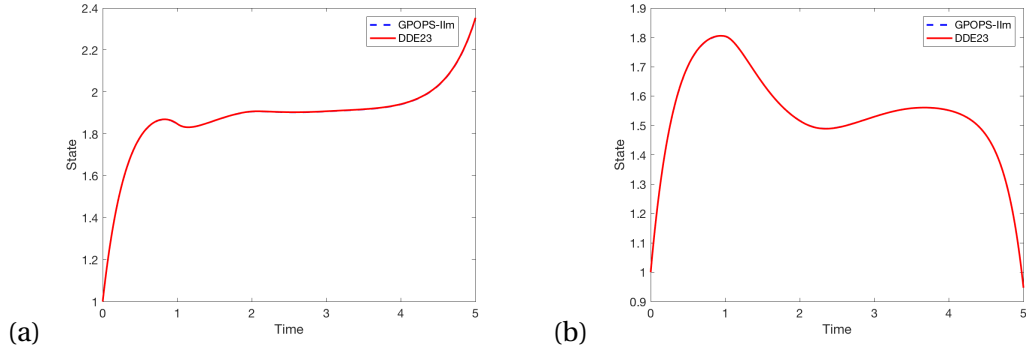
**Figure 5.24** (a) State and (b) control obtained by GPOPS-IIIm, control parameterization, SOSD for  $c = 1.2, \tau = 1$  with cost Eq. 5.8.

**Table 5.2** Values of the optimal cost,  $J$ , for each  $\tau$  and  $c = \pm 1.2$  obtained by GPOPS-IIIm, SOSD, and control parameterization.

$c$	$\tau$	$J$ (Ctrl Param)	$J$ (GPOPS-IIIm)	$J$ (SOSD)
-1.2	1.0	43.4214095	44.6641508	43.421408
-1.2	0.05	46.8766437	46.8795593	46.876541
1.2	1.0	8.0179026	8.2505467	8.0178992
1.2	0.05	9.4539446	9.4558176	9.4539374

Though GPOPS-IIIm does not produce the same trajectory as SOSD, the costs are very similar. We seek to establish whether the control from GPOPS-IIIm correctly integrates the dynamics to produce the state trajectories in Fig. 5.24(a) and Fig. 5.21(a) or if the results are due to numerical error. To test our control, we integrate the delayed dynamics in Eq. 5.23 using dde23 utilizing the control from GPOPS-IIIm. The results in Fig. 5.25 demonstrate that GPOPS-IIIm successfully integrates the dynamics and produces a suboptimal control.

Our results in this section show GPOPS-IIIm performs well on OCP with delays in the state for

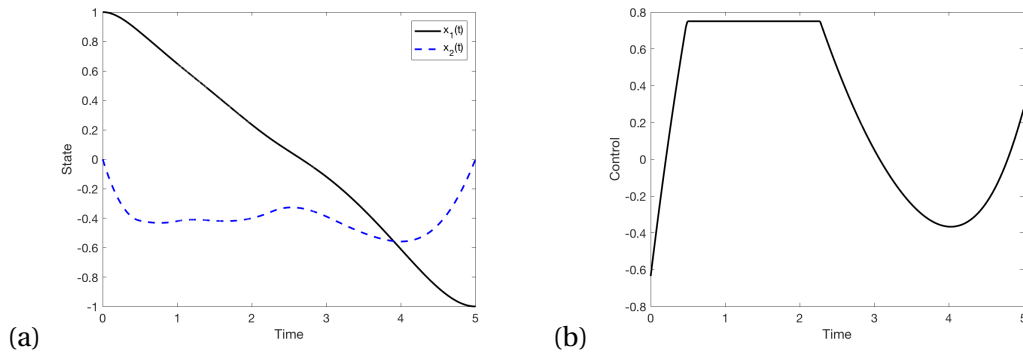


**Figure 5.25** (a) State obtained by GPOPS-IIIm and dde23 for  $c = 1.2$  and (b)  $c = -1.2$ ,  $\tau = 1$  with cost Eq. 5.8.

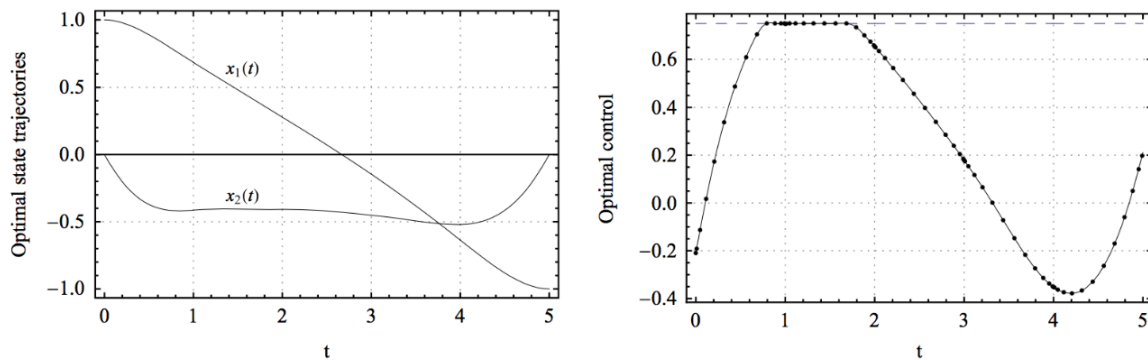
small delays or when the coefficients on the delay term and the state have the same sign. Our results for small  $\tau$  are consistent with our analysis of the necessary conditions for GPOPS-IIIm in Chapter 4. Later in this chapter we present a result based on the coefficient of the delay terms which help to explain the change in numerical results we observe for differing values of  $c$ .

### 5.2.1.1 Nonlinear Example

Given that GPOPS-II<sub>m</sub> successfully computes suboptimal controls for linear dynamics, we now test the performance of GPOPS-II<sub>m</sub> on a nonlinear example. Consider the modified Van der Pol oscillator given in Eq. 5.9 and Eq. 5.10 from [53], which we previously examined using the necessary conditions. We continue to use three mesh iterations, set  $P = 3$ , let our initial  $N = 20$ , and set our tolerances to  $1e-07$ . GPOPS-II<sub>m</sub> performs very well on our example. Our plots in Fig. 5.26 are very similar to those from [53] in Fig. 5.27, however their control starts at -2.0 while ours starts at -6.0, and our control remains at the maximum value of .75 for slightly longer. The authors found the cost for a range of parameter values, the largest being  $J = 1.987116$ . Our cost is slightly higher at  $J = 2.0237941$ . GPOPS-II<sub>m</sub> is able to compute a reasonable solution to highly nonlinear dynamics.



**Figure 5.26** (a) State and (b) control for modified Van der Pol oscillator using GPOPS-II<sub>m</sub> with cost  $J = 2.0237941$ .



**Figure 5.27** Optimal state and control for modified Van der Pol Oscillator from [53].

### 5.2.2 Use of GPOPS-II and Waveform Optimization (GPOPS-IIow)

Earlier in this chapter we see that GPOPS-II works well as an integrator on DDEs using waveform relaxation. We now consider if using GPOPS-II as an iterative method will work well on a delayed OCP. In particular, we test if using GPOPS-II iteratively will produce a better solution to the delayed state optimal control problem in Eq. 5.19 and Eq. 5.8. Waveform optimization is a method of calling GPOPS-II recursively to solve the problem

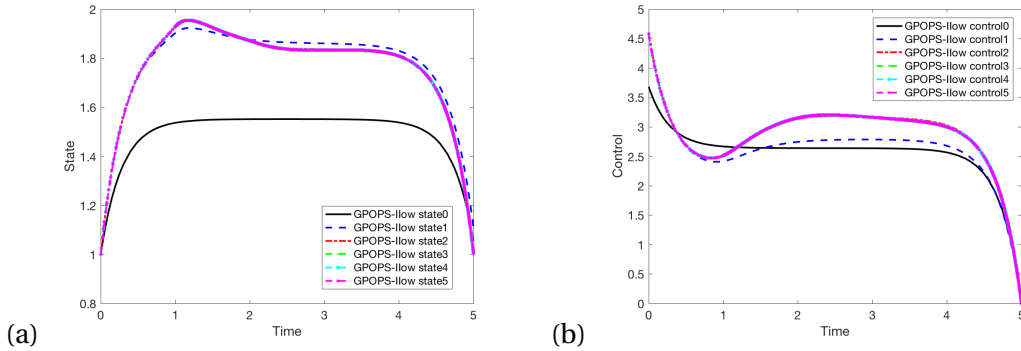
$$\dot{x}_n = a x_n + c x_{n-1}(t - \tau) + u_n, \quad x(0) = 1 \quad (5.24)$$

subject to the cost

$$J_n = \int_0^T 10(x_n - 2)^2 + u_n \, dt. \quad (5.25)$$

We refer to this method as GPOPS-IIow [11]. In all cases, we run GPOPS-IIow for five iterations. For our first iteration we set our tolerances to  $1e-06$  and allow one mesh iteration. For the four subsequent iterations we set our tolerances at  $1e-09$  and allow four mesh iterations. Allowing GPOPS-IIow more mesh iterations does not improve the convergence of the method but does significantly increase computational time. GPOPS-IIow struggles to solve the  $\tau = 0$  case in Eq. 5.24 on the first run. Subsequent function calls are simply refining the initial solution and GPOPS-IIow runs much more quickly. We experimented with calling GPOPS-IIow for more than five iterations, however this did not improve the approximation.

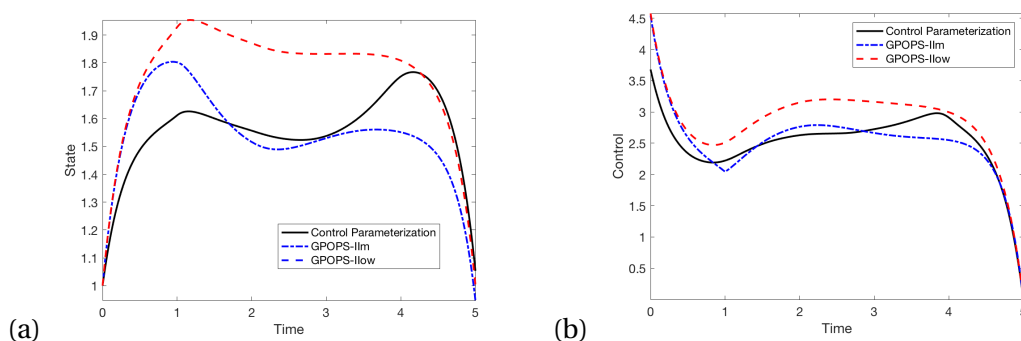
We begin with  $\tau = 1$  and  $c = -1.2$ . We can see from Fig. 5.28 that GPOPS-IIow does converge within five iterations. We now compare the results from GPOPS-IIow to those from control param-



**Figure 5.28** (a) States and (b) controls from GPOPS-IIow for Eq. 5.24 with  $\tau = 1$  and  $c = -1.2$ .

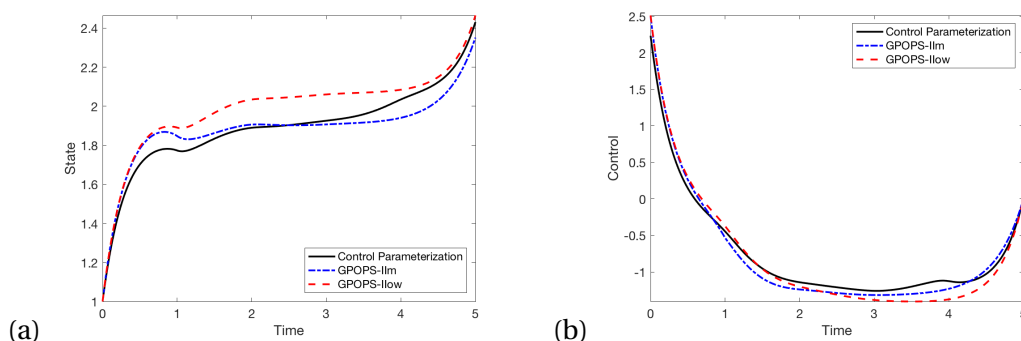
terization and GPOPS-II<sub>m</sub>, where we take control parameterization as our “true” solution. We again

use  $N = 28$  subintervals and cubic splines to parameterize our control. The states and controls in Fig. 5.29 show that while GPOPS-IIow produces a control relatively similar to GPOPS-IIIm and control parameterization, GPOPS-IIow produces a much larger state. This is reflected in our costs: control parameterization gives a cost of  $J = 43.4214094$  while GPOPS-IIIm gives a cost of  $J = 44.6641508$  and GPOPS-IIow produces the largest cost of  $J = 47.1541370$ . The states from both GPOPS-IIIm and GPOPS-IIow seem to have a maximum in the first half of the interval, while control parameterization has its maximum in the second half.



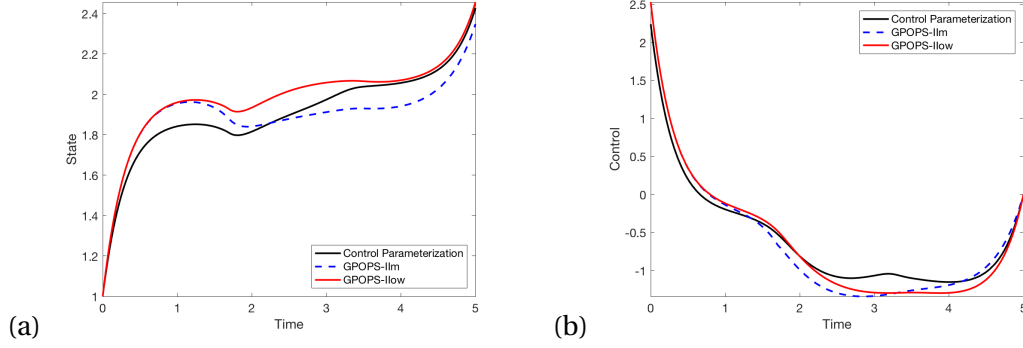
**Figure 5.29** (a) Comparison of states and (b) controls produced by GPOPS-IIow, GPOPS-IIIm, and control parameterization for Eq. 5.19 with  $\tau = 1$  and  $c = -1.2$ .

We consider a more unstable system with  $c = +1.2$  and keep the delay  $\tau = 1$ . GPOPS-IIow obtains a cost of  $J = 8.6946605$  as opposed to the control parameterization cost of  $J = 8.0179026$ . We again graph all three approaches in Fig. 5.30. Although GPOPS-IIow still produces a higher cost, it approximates the solutions to this system much more closely than the  $c = -1.2$  case.



**Figure 5.30** (a) Comparison of states and (b) controls produced by GPOPS-IIow, GPOPS-IIIm, and control parameterization for Eq. 5.19 with  $\tau = 1$  and  $c = +1.2$ .

Next, we keep  $c = +1.2$  and increase our delay to  $\tau = 1.7$ . GPOPS-II<sub>m</sub> gives a cost of  $J = 7.1701048$  while control parameterization gives a cost of  $J = 6.6920511$ . GPOPS-II<sub>ow</sub> gives a cost of  $J = 7.1537929$ . Interestingly the cost using GPOPS-II<sub>ow</sub> is lower than the cost from GPOPS-II<sub>m</sub>. Fig. 5.32 demonstrates that two solutions from GPOPS-II are nearly identical until the delay takes effect. The largest difference is in the state; Fig. 5.32(a) shows GPOPS-II<sub>ow</sub> consistently produces a higher value.

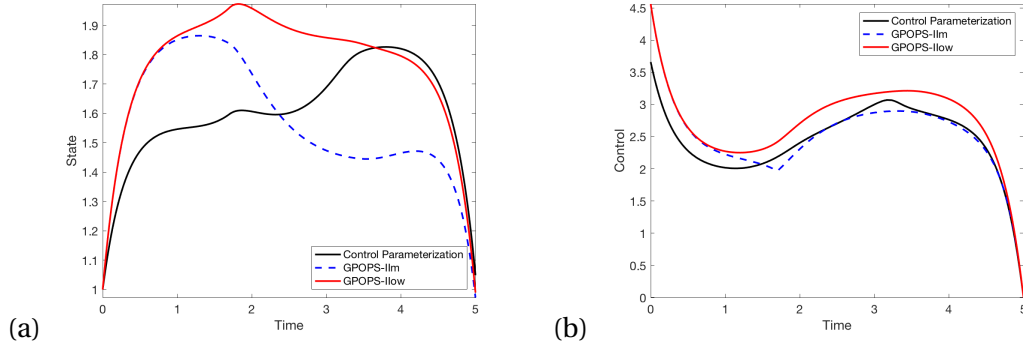


**Figure 5.31** (a) Comparison of states and (b) controls produced by GPOPS-II<sub>ow</sub>, GPOPS-II<sub>m</sub>, and control parameterization for Eq. 5.19 with  $\tau = 1.7$  and  $c = +1.2$ .

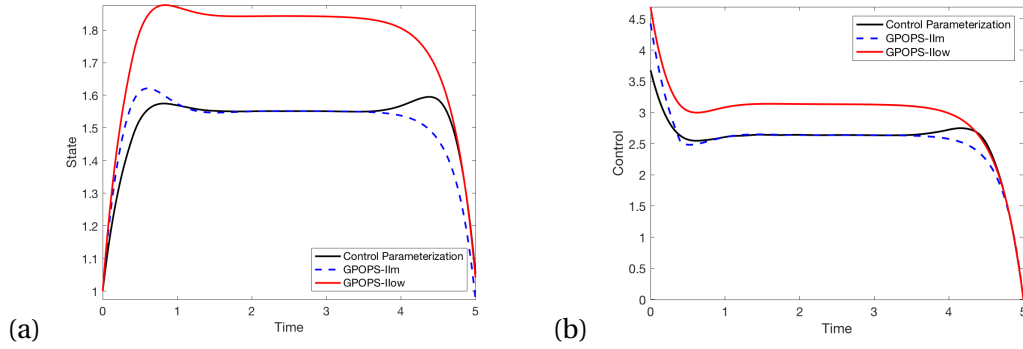
We now keep our delay at  $\tau = 1.7$  but change to  $c = -1.2$ . GPOPS-II<sub>m</sub> gives the cost  $J = 44.0635701$ , while using control parameterization gives the cost  $J = 40.6378870$ . GPOPS-II<sub>ow</sub> gives a cost of  $J = 43.7472571$ . While both methods using GPOPS-II produce a significantly higher cost than that produced by control parameterization, GPOPS-II<sub>ow</sub> produces a lower cost than GPOPS-II<sub>m</sub>; this behavior reflects the results from the  $c = +1.2$  case. It seems in systems with a longer delay, the waveform method may be a better choice. Fig. 5.32(a) in particular shows that neither method using GPOPS-II approximates the state very well, but the waveform method more closely approximates the value of the state toward the right end of the interval.

After observing the behavior of GPOPS-II<sub>ow</sub> with a longer delay, we consider the behavior of GPOPS-II<sub>ow</sub> with a small delay of  $\tau = 0.4$  and  $c = -1.2$ . Using GPOPS-II<sub>m</sub> we obtain the cost  $J = 9.1392437$ . Optimization by control parameterization gives a cost of  $J = 9.0871722$ . GPOPS-II<sub>ow</sub> gives a cost of  $J = 9.9720657$ , which is significantly higher than the cost obtained by control parameterization. Fig. 5.34 demonstrates that for a smaller delay, GPOPS-II<sub>m</sub> is a superior choice to waveform optimization.

We now use  $c = -1.2$  and keep  $\tau = 0.4$ . GPOPS-II<sub>m</sub> gives a cost of  $J = 45.9988038$ , while optimization via control parameterization obtains a cost of  $J = 45.7439663$ . Using GPOPS-II<sub>ow</sub>, we obtain the cost  $J = 50.0036677$ . Fig. 5.33 demonstrates that, again, changing the sign on the



**Figure 5.32** (a) Comparison of states and (b) controls produced by GPOPS-IIlow, GPOPS-IIIm, and control parameterization for Eq. 5.19 with  $\tau = 1.7$  and  $c = -1.2$ .

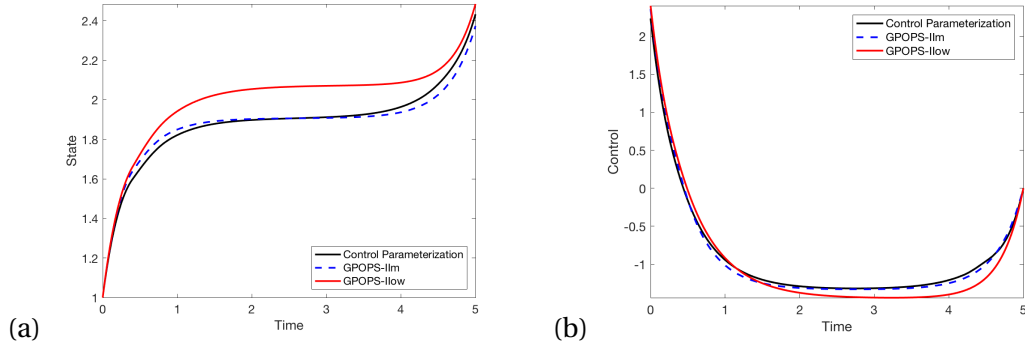


**Figure 5.33** (a) Comparison of states and (b) controls produced by GPOPS-IIlow, GPOPS-IIIm, and control parameterization for Eq. 5.19 with  $\tau = 0.4$  and  $c = -1.2$ .

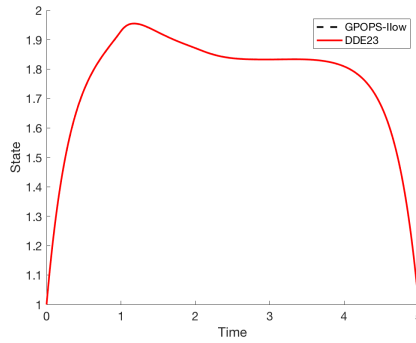
delay term results a significant decrease in performance by GPOPS-II. As with the  $c = +1.2$  case, the costs and Fig. 5.33 show the shorter delay results in a less accurate result from GPOPS-IIlow. We can see from Fig. 5.33(a) that GPOPS-IIlow has produces a much larger state and control than control parameterization, explaining the significantly increased cost.

We now test if GPOPS-IIlow is correctly integrating its delayed dynamics. To do so, we pass the control computed by GPOPS-IIlow to dde23, and use dde23 to integrate our delayed dynamics with  $c = -1.2$  and  $\tau = 1$ . We can see from Fig. 5.35 that the control computed by GPOPS-IIlow does in fact produce the correct state. We know that for problems where GPOPS-IIlow does converge, it will converge to a suboptimal solution.

Our numerical results reflect the work we did in Chapter 4: the error in Eq. 4.44, the difference in our necessary conditions, is  $\mathcal{O}(\tau)$  and so for shorter delays GPOPS-IIIm converges to the undelayed case. Thus for systems with small delays, GPOPS-IIIm is a useful problem formulation. GPOPS-IIlow, on the other hand, is not solving the same set of necessary conditions and so does not converge to



**Figure 5.34** (a) Comparison of states and (b) controls produced by GPOPS-IIlow, GPOPS-IIlm, and control parameterization for Eq. 5.19 with  $\tau = 0.4$  and  $c = +1.2$ .



**Figure 5.35** State from GPOPS-IIlow and state obtained by dde23 integrating the delayed dynamics with  $c = -1.2$ ,  $\tau = 1$ .

the undelayed necessary conditions as the delay decreases. This explains the poor performance of GPOPS-IIlow for shorter delays. On intervals with longer delays, however, GPOPS-IIlow may be used to obtain a suboptimal control. Though the cost is higher, the computational time is much lower than that of control parameterization.

### 5.3 Control Parameterization Using Activate

In addition to using GPOPS-II to solve delayed OCP in MATLAB, we worked with the software Activate. Our interest in Activate is motivated by its ability to use block diagrams rather than scripts to run, and the fact it is written in open source OML. A basic edition of the software is free to download, making it a very accessible tool for researchers to use. Users may obtain a free copy for personal use in [1]. We note that Activate runs in a Windows environment. In our work we utilize a virtual Windows environment.

There are two main methods of optimization using Activate: the user has the option to optimize

parameters by iterating simulation runs, or the user can utilize one of two optimization blocks within the model. We utilize the `BobyqaOpt` block, where `BobyqaOpt` stands for Bound Optimization BY Quadratic Optimization [19]. The `BobyqaOpt` algorithm is an iterative algorithm which seeks to minimize an objective function  $F(x)$ ,  $x \in \mathbb{R}^n$  subject to the constraints  $a_i \leq x_i \leq b_i$ . It relies on a quadratic approximation  $Q$  such that  $Q(y_j) = F(y_j)$ ,  $j = 1, \dots, n$  where the  $y_j$  are the interpolation points [65]. The user needs to specify the upper and lower bounds on the trust region, the cost, a set of parameters over which to optimize, and an initial value for the parameters [19].

We use `Activate` with control parameterization to solve the example

$$\dot{x} = -0.5x + c x(t - \tau) + u, \phi(t) = 1 \quad (5.26a)$$

$$J = \int_0^T 10(x - 2)^2 + u^2 dt, \quad (5.26b)$$

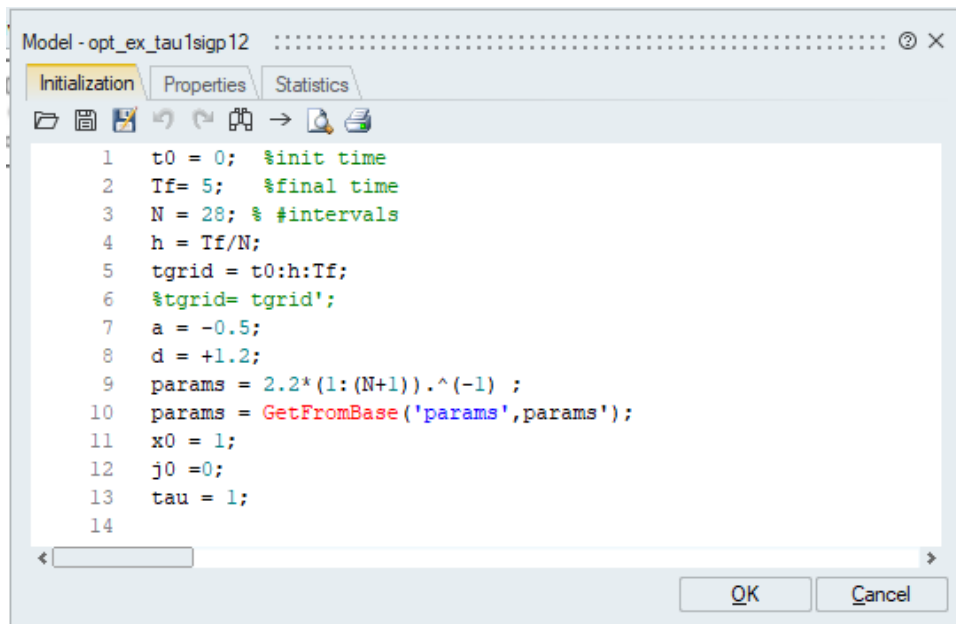
where  $\phi(t) = 1$  is our prehistory. `Activate` incorporates both fixed and variable delay systems using its `FixedDelay` and `VariableDelay` blocks. We use a `FixedDelay` block and define the delay  $\tau$  in the Initialization script. We calculate the cost using a `MathExpression` block and an integral block, which integrates Eq. 5.26b with initial condition  $J(0) = 0$  to account for the lower bound on the integral. The final value of the integral in Eq. 5.26b is then passed to the `BobyqaOpt` block to be optimized. The initial and final time are set as simulation parameters, which are fixed. We utilize the solver `LSODA` for stiff and nonstiff ODEs. The solver initially uses a nonstiff Adams method and switches to a stiff BDF (backwards differentiation formula) as needed [38].

We define an initial value of the parameters in our model initialization script, and pass this to the `BobyqaOpt` block using the "GetFromBase" argument. The parameter values are updated using the "To Base" argument, which enables us to use the new value of the parameter in our cost. To generate the control,  $u$ , we use the `SignalGenerator` block.

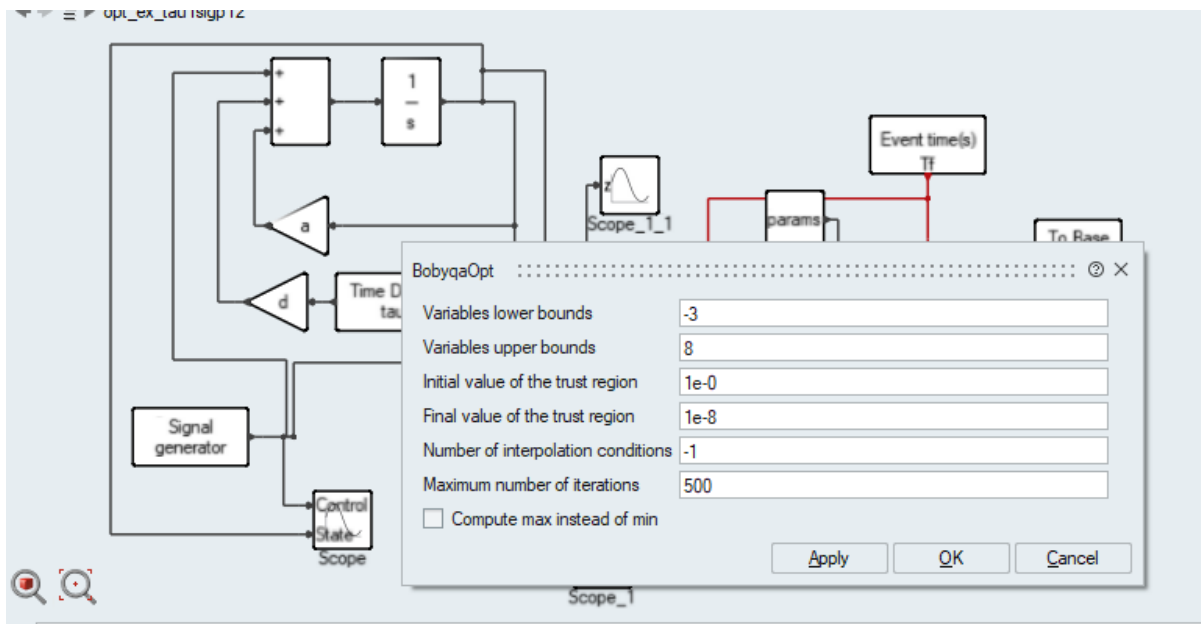
We define a time grid in the initialization script, and use natural splines in the `SignalGenerator` block to interpolate our parameters at the time grid. We then pass this signal to the cost in the `MathExpression` block and to the dynamics using the `Sum` block. The `BobyqaOpt` block automatically restarts the simulation until the specified error tolerances are met, at which point the final value of the cost is displayed in a `Display` block. We use the `Scope` block to plot our final state and control in Fig. 5.39 along with our other numerical results.

We consider the example in Eq. 5.26 with  $c = +1.2$  and  $\tau = 1$ . We define our initial conditions, time grid, and initial parameter value in the Initialization script in Fig. 5.37. We choose  $N = 28$  subintervals for comparison to our earlier results from using control parameterization in MATLAB. One challenge is determining an appropriate initial value for the control parameters. We first use our initial value from our control parameterization in MATLAB,  $u = 6 * (1 : N + 1)^{-1}$ , however this produces a suboptimal result. By observation of Fig. 5.30(b) we ascertain that a reasonable initial

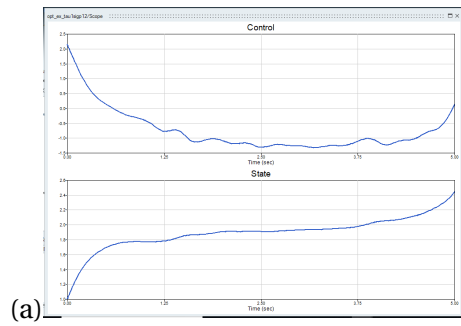




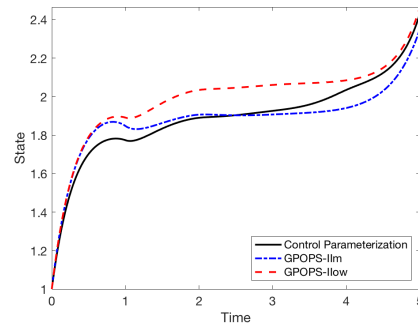
**Figure 5.37** Initialization script for Eq. 5.26 with  $c = +1.2$  and  $\tau = 1$ .



**Figure 5.38** BobbyqaOpt block parameters for Eq. 5.26 with  $c = +1.2$  and  $\tau = 1$ .



(a)



(b)

**Figure 5.39** (a) State and control obtained from Activate for Eq. 5.26 and (b) state obtained by control parameterization and GPOPS-II with  $c = +1.2$  and  $\tau = 1$ .

## 5.4 Analysis of Results

Our numerical results demonstrate that GPOPS-II<sub>m</sub> struggles to obtain an optimal solution to a delayed OCP for certain values of our delay or coefficients. We also see that GPOPS-II<sub>m</sub> obtains better results when solving the necessary conditions, a delayed BVP. When GPOPS-II is simply integrating the dynamics, it is no longer providing a cost or inequality constraints to IPOPT. We are able to demonstrate that in this case IPOPT becomes a damped quasi-Newton method, and for a sufficiently accurate initial value, the iteration will still converge.

We follow the notation and conventions established in [14]. Our NLP is of the form

$$\min_{x \in \mathbb{R}^n} f(x) \quad (5.27a)$$

$$\text{such that } c(x) = 0 \quad (5.27b)$$

$$x_L \leq x \leq x_U, \quad (5.27c)$$

where  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  is our objective function,  $c(x)$  are our equality constraints, and Eq. 5.27c are our inequality constraints. We consider the equivalent formulation

$$\min_{x \in \mathbb{R}^n} f(x) \quad (5.28a)$$

$$\text{such that } c(x) = 0 \quad (5.28b)$$

$$0 \leq x. \quad (5.28c)$$

Let  $x_k$  be the approximation to our NLP variables,  $\lambda_k$  the approximation to our equality constraint multipliers, and  $z_k$  the approximation to our inequality constraint multipliers. Then, given iterate  $(x_k, \lambda_k, z_k)$ , the inner loop to compute search directions  $(d_k^x, d_k^\lambda, d_k^z)$  is

$$\begin{pmatrix} W_k & A_k & -I \\ A_k^T & 0 & 0 \\ Z_k & 0 & X_k \end{pmatrix} \begin{pmatrix} d_k^x \\ d_k^\lambda \\ d_k^z \end{pmatrix} = - \begin{pmatrix} \nabla f + A_k \lambda_k - z_k \\ c(x_k) \\ X_k Z_k e - \mu_j e \end{pmatrix}. \quad (5.29)$$

Here,  $W_k$  is the Hessian  $\nabla_{xx}^2 \mathcal{L}(x_k, \lambda_k, z_k)$  of the Lagrangian and  $A_k := \nabla c(x_k)$ . Then the iterates are computed by

$$x_{k+1} = x_k + \alpha_k d_k^x \quad (5.30a)$$

$$\lambda_{k+1} = \lambda_k + \alpha_k d_k^\lambda \quad (5.30b)$$

$$z_{k+1} = z_k + \alpha_k^z d_k^z, \quad (5.30c)$$

where the  $\alpha_k$  are the step sizes. For details on step size and search direction choice, see [80]. When we use IPOPT to simulate the delayed dynamics or solve a delayed BVP, we no longer have the

inequality constraints or objective function and the state values are completely determined by the equality constraints  $c(x) = 0$  and Eq. 5.30.

Then the NLP with GPOPS-II becomes

$$\tilde{c}'(x_k) d_k^x = -c(x_k) \quad (5.31a)$$

$$x_{k+1} = x_k + \alpha_k d_k^x. \quad (5.31b)$$

Here,  $\tilde{c}'(x_k)$  is an approximation to the true gradient  $c'(x_k)$  where  $\tilde{c}'(x_k)$  is obtained by neglecting those entries which come from the delayed terms. We note that for  $x_k$  close to the solution of  $c(x) = 0$ ,  $c'(x_k)$  and  $\tilde{c}'(x_k)$  are nonsingular. We now establish a sufficient condition for the convergence of the damped quasi-Newton method given sufficiently accurate initial values for our variables.

We examine the change in  $\|c(x)\|^2$  over one iteration with search direction  $\hat{\delta}$  obtained from Eq. 5.31. For the moment we neglect the  $k$  subscript. First, note that  $c(x + \hat{\delta}) = c(x) + c'(x)\hat{\delta} + \mathcal{O}(\|\hat{\delta}\|^2)$ . Then we have

$$c(x + \hat{\delta})^T c(x + \hat{\delta}) - c(x)^T c(x) = \quad (5.32a)$$

$$(c(x) + c'(x)\hat{\delta} + \mathcal{O}(\|\hat{\delta}\|^2))^T (c(x) + c'(x)\hat{\delta} + \mathcal{O}(\|\hat{\delta}\|^2)) - c(x)^T c(x) = \quad (5.32b)$$

$$\hat{\delta}^T c'(x)^T c(x) + c(x)^T c'(x)\hat{\delta} + \mathcal{O}(\|\hat{\delta}\|^2). \quad (5.32c)$$

Dropping our higher order terms, and substituting  $\hat{\delta} = -\tilde{c}'(x)^{-1}c(x)$  back into Eq. 5.32 we have

$$-(\tilde{c}'(x)^{-1}c(x))^T c'(x)^T c(x) - c(x)^T c'(x)\tilde{c}'(x)^{-1}c(x) = \quad (5.33a)$$

$$-c(x)^T (\tilde{c}'(x)^{-T} c'(x)^T + c'(x)\tilde{c}'(x)^{-1}) c(x). \quad (5.33b)$$

We can write  $c'(x)$  as  $E + F$  where  $E = \tilde{c}'(x)$  contains entries for the undelayed terms in  $c'(x)$  and  $F$  contains entries corresponding to the delay terms in  $c'(x)$ . We note that  $F$  has nonzero entries corresponding to zero entries in  $E$ . Then Eq. 5.33b becomes

$$-c^T (E^{-T}(E + F)^T + (E + F)E^{-1}) c = \quad (5.34a)$$

$$-c^T ((I + FE^{-1})^T + (I + FE^{-1})) c. \quad (5.34b)$$

We want Eq. 5.34 to be negative along search direction  $\hat{\delta}$  provided it is negative along direction  $\delta$ . The condition that the numerical radius of  $FE^{-1} < 1$  is a sufficient condition to guarantee Eq. 5.34 < 0, a weaker requirement than  $\|FE^{-1}\| < 1$ . This condition on the coefficients of the delay is sufficient but not necessary, and the IPOPT iteration may converge when Eq. 5.34 < 0 does not hold.

The result we just presented is for solving a BVP with delay or carrying out an integration. Now

consider the case when we use GPOPS-IIIm on a delayed OCP. We have the problem

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & f(x) \\ \text{such that} \quad & c(x) = 0, \end{aligned} \tag{5.35a}$$

$$\tag{5.35b}$$

where  $x = [\bar{x} \ u]^T$ , where  $\bar{x}$  are the state variables, and we have no inequality constraints. The true search direction  $\delta$  is computed by

$$c'(x)\delta = -c(x). \tag{5.36}$$

However, because GPOPS-IIIm neglects the delay terms in the gradient, IPOPT is actually solving

$$\tilde{c}'(x)\hat{\delta} = -c(x), \tag{5.37}$$

where  $\tilde{c}'(x)$  is an approximation of the true gradient  $c'(x)$  obtained by neglecting the delay terms. We note that here

$$c(x) = [c(\bar{x}) \ c(u)] \tag{5.38}$$

and

$$c'(x) = [c_{\bar{x}} \ c_u]. \tag{5.39}$$

Since  $c_{\bar{x}}$  is invertible, we have  $[c_{\bar{x}} \ c_u]$  has full row rank.

We again wish to consider the change in  $\|c(x)\|^2$  over a single iteration, using the search direction  $\hat{\delta}$  from Eq. 5.37. First, note that  $c(x + \hat{\delta}) = c(x) + c'(x)\hat{\delta} + \mathcal{O}(\|\hat{\delta}\|^2)$  and  $c'(x) = \tilde{c}'(x) + F$  where  $\tilde{c}'$  is the Jacobian used by GPOPS-IIIm and  $F$  contains the derivatives of the delay terms. Then  $c'(x)\hat{\delta} = (\tilde{c}'(x) + F)\hat{\delta} = -c(x) + F\hat{\delta}$  and we have

$$c(x + \hat{\delta})^T c(x + \hat{\delta}) - c(x)^T c(x) = \tag{5.40a}$$

$$(F\hat{\delta} + \mathcal{O}(\|\hat{\delta}\|^2))^T (F\hat{\delta} + \mathcal{O}(\|\hat{\delta}\|^2)) - c^T(x)c(x) = \tag{5.40b}$$

$$\hat{\delta}^T F^T F \hat{\delta} + \mathcal{O}(\|\hat{\delta}\|^3) - c^T(x)c(x), \tag{5.40c}$$

where we drop our higher order terms.

Given  $c(x) \neq 0$ , for either  $\hat{\delta}$  or  $\|F\|^2$  sufficiently small relative to  $\|c(x)\|^2$  we have that Eq. 5.40c is negative and can expect our iteration to converge. An important factor is the relative size of the coefficients on our delayed and undelayed terms. For cases where the coefficient on the delayed term is large relative to the coefficients on the undelayed terms, we cannot guarantee the convergence of IPOPT.

### 5.4.1 Effect of Delay Coefficient on Solutions

Given our analytic results on the size of our delay terms from the previous section, we now analyze numerically how altering the coefficients on the delay and the state in a delayed OCP affects the convergence of GPOPS-II to the correct solution. We first consider GPOPS-II on the following problem

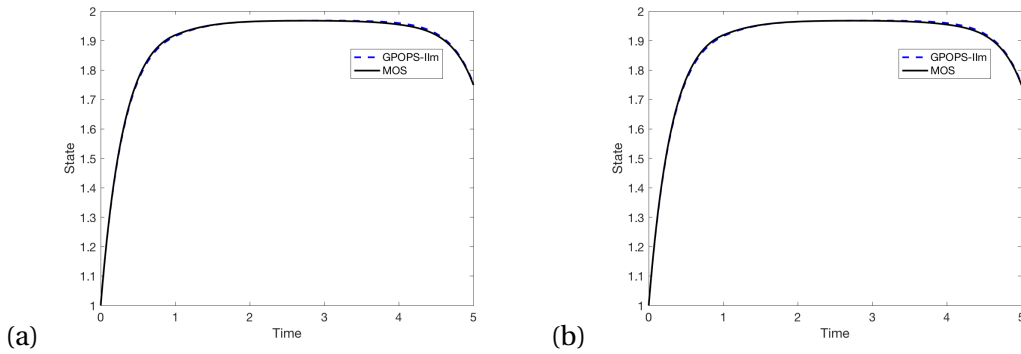
$$\dot{x} = -.5x + c x(t-1) + u \quad (5.41a)$$

$$J = \int_0^5 10(x-2)^2 + u^2 dt \quad (5.41b)$$

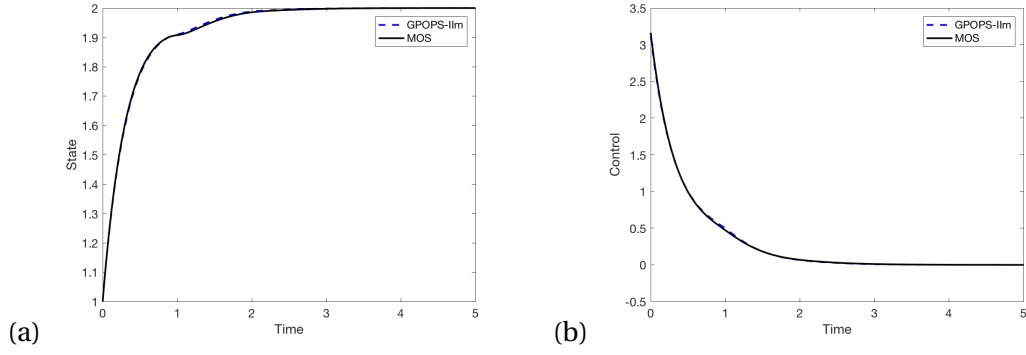
for several parameter values of  $c$  and observe how changing  $c$  affects the quality of our solution.

We compare our solutions from GPOPS-II to those from MOS. We implement MOS in GPOPS-II. Unless otherwise noted we use  $N = 4$ ,  $P = m = 3$ , and tolerances are set at  $1e-07$ . Costs for each value of  $c$  are displayed in Table 5.3 and run times using the MATLAB "tic" and "toc" functions are displayed in Table 5.4. Computations are conducted on a late 2013 15 inch MacBook Pro with a 2.6GHz Intel Core processor. In this section all timings should be considered approximate as machine processes frequently affect recorded computational times for the very small time scales which we present.

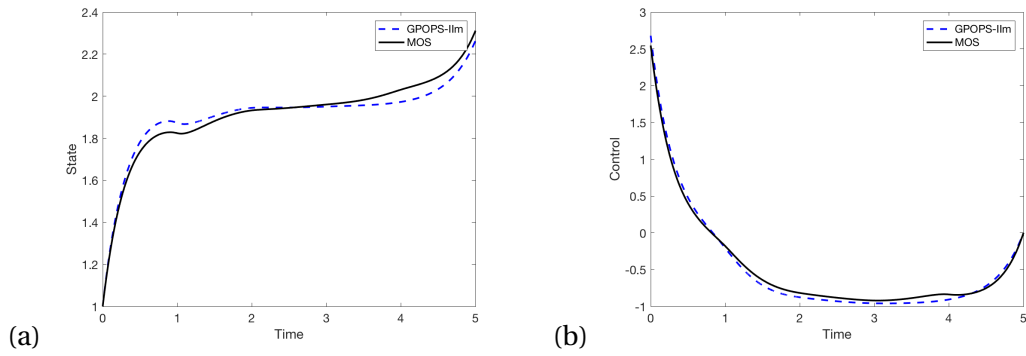
We can clearly see in Fig. 5.40-Fig. 5.44 that as the value of  $c$  is increases, the quality of the solution from GPOPS-II degrades. We also see consistently increasing computational times from GPOPS-II while the computational time for MOS remains relatively constant. The results in Table 5.4 reflect our analytic results; when the delayed state coefficient  $c$  becomes large relative to the undelayed state coefficient  $a$ , our solution from GPOPS-II degrades.



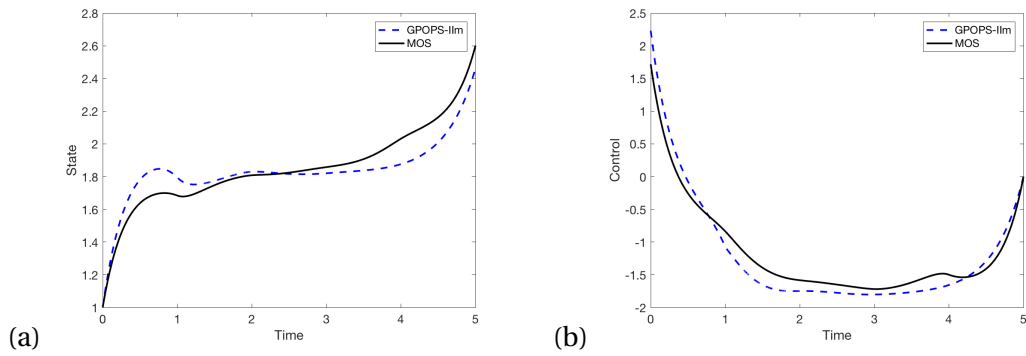
**Figure 5.40** (a) State and (b) control by GPOPS-II and MOS with  $c = .1$  for Eq. 5.41, cost  $J = 7.3484273$  from GPOPS-II, and  $J = 7.3476357$  from MOS.



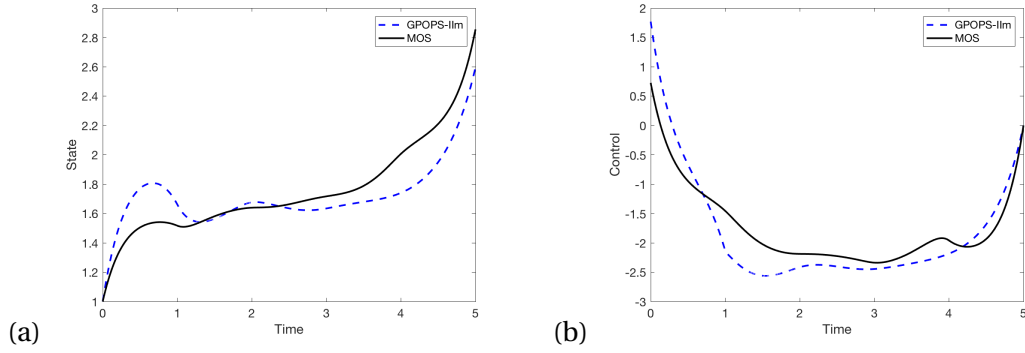
**Figure 5.41** (a) State and (b) control of Eq. 5.41 with  $c = .5$ , cost 3.7867743 from GPOPS-IIIm, and cost  $J = 3.7859437$  from MOS.



**Figure 5.42** (a) State and (b) control of Eq. 5.41 with  $c = 1.0$ , cost 5.6303104 from GPOPS-IIIm, and cost  $J = 5.5426332$  from MOS.



**Figure 5.43** (a) State and (b) control of Eq. 5.41 with  $c = 1.5$ , cost 13.9834065 from GPOPS-IIIm, and cost  $J = 13.3242859$  from MOS.



**Figure 5.44** (a) State and (b) control of Eq. 5.41 with  $c = 2.0$ , cost 27.4879834 from GPOPS-IIIm, and cost  $J = 25.4211318$  from MOS.

**Table 5.3** Costs from GPOPS-IIIm and MOS for different values of  $c$  in Eq. 5.41.

$c$	$J$ GPOPS-IIIm	$J$ MOS	$ \Delta J $	% Difference
-2.0	78.3348869	72.5385748	5.7963	1.9209
-1.5	56.7225871	54.3770261	2.3456	1.0556
-1.0	37.0650163	36.3281545	0.7369	0.5020
-.5	20.4889357	20.3895389	0.0994	0.1216
-.1	10.7545304	10.7529283	0.0016	0.0037
.1	7.3484273	7.3476357	0.0008	0.0108
.5	3.7867743	3.7859437	0.0008	0.0219
1.0	5.6303104	5.5426332	0.0877	1.5695
1.5	13.9834065	13.3242859	0.6591	4.8274
2.0	27.4879834	25.4211318	2.0669	7.8128

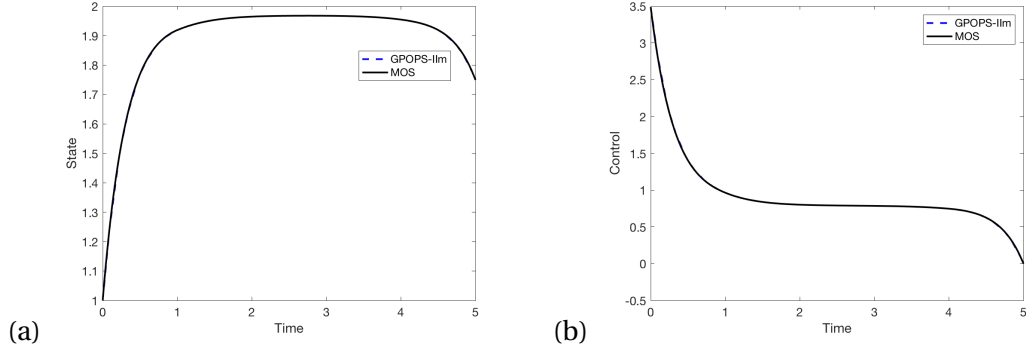
**Table 5.4** Timing, in seconds, for different values of  $c$  in Eq. 5.41.

$c$	T (s) GPOPS-IIIm	T (s) MOS	$ \Delta T $	% Difference
-2.0	2.0585	0.5836	1.4749	27.9115
-1.5	2.2258	0.5508	1.6750	30.1628
-1.0	1.0625	0.5979	0.4646	13.9906
-.5	0.7252	0.8281	0.1029	3.3123
-.1	2.4818	1.1224	1.3594	18.8586
.1	1.5604	1.1256	0.4348	32.3753
.5	2.0176	1.0721	0.9456	61.2118
1.0	3.6704	1.0764	2.5940	109.2947
1.5	5.1277	1.1164	4.0113	128.4829
2.0	8.0710	1.3813	6.6897	141.5465

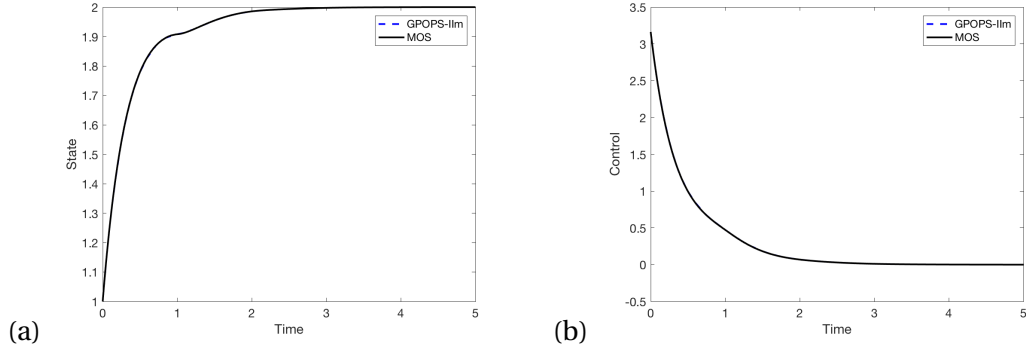
#### 5.4.1.1 Solving the Necessary Conditions

We now compare the timings and solutions of MOS versus solving the necessary conditions for Eq. 5.41 with differing values of  $c$ . For  $c = .1, .5$  we are able to keep our tolerances at  $1e-07$ . For  $c = 1$

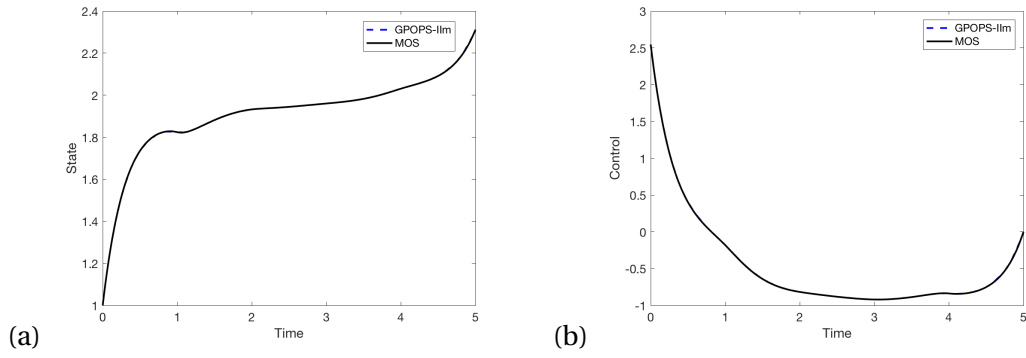
we need to decrease our mesh tolerance in the necessary conditions to  $1e-06$  in order to obtain an answer. We decrease our mesh tolerance again to  $1e-05$  for  $c = 1.5, c = -1, c = -0.5$ . The costs in Table 5.5 and graphs in Fig. 5.45-Fig. 5.49 show that using the necessary condition formulation of the problem greatly improves our results.



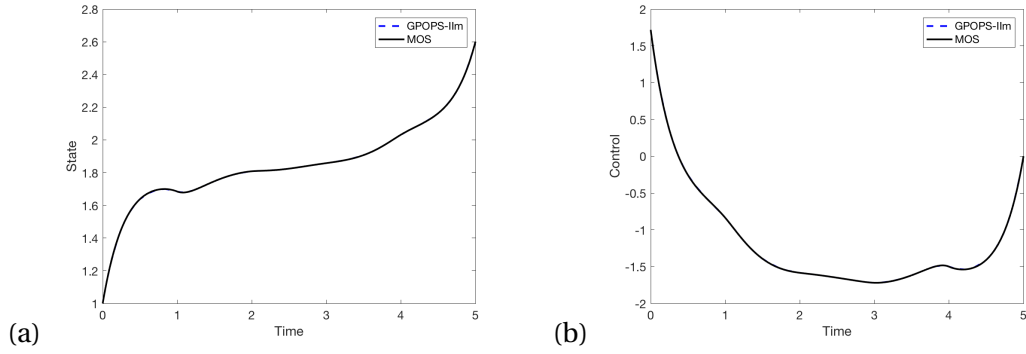
**Figure 5.45** (a) State and (b) control of Eq. 5.41 with  $c = .1, a = -.5$ , cost 7.3477053 from GPOPS-IIIm with necessary conditions and cost  $J = 7.3476357$  from MOS.



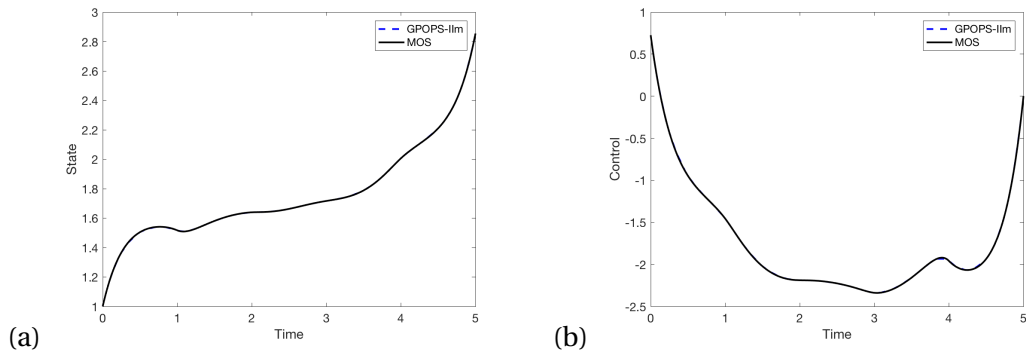
**Figure 5.46** (a) State and (b) control of Eq. 5.41 with  $c = .5, a = -.5$ , cost 3.7861046 from GPOPS-IIIm with necessary conditions and cost  $J = 3.7859437$  from MOS.



**Figure 5.47** (a) State and (b) control of Eq. 5.41 with  $c = 1, a = -.5$ , cost 5.5421608 from GPOPS-IIIm with necessary conditions and cost  $J = 5.5426332$  from MOS.



**Figure 5.48** (a) State and (b) control of Eq. 5.41 with  $c = 1.5, a = -.5$ , cost 13.3243674 from GPOPS-IIIm with necessary conditions and cost  $J = 13.3242859$  from MOS.



**Figure 5.49** (a) State and (b) control of Eq. 5.41 with  $c = 2.0, a = -.5$ , cost 25.4211284 from GPOPS-IIIm with necessary conditions and cost  $J = 25.4211318$  from MOS.

**Table 5.5** Costs from GPOPS-II<sub>m</sub> and MOS for different values of  $c$  in Eq. 5.41 using GPOPS-II<sub>m</sub> on the necessary conditions.

$c$	$J$ GPOPS-II <sub>m</sub> Nec. Conds.	$J$ MOS	$ \Delta J $	$\Delta J\%$
-2.0	72.5393963	72.5385748	0.0008	0.0003
-1.5	54.3770318	54.3770261	0.0000	0.0000
-1.0	36.3263025	36.3281545	0.0019	0.0013
-.1	10.7527379	10.7529283	0.0002	0.0004
.1	7.3477053	7.3476357	6.9600e-05	0.0002
.5	3.7861046	3.7859437	1.6090e-04	0.0011
1.0	5.5421608	5.5426332	4.7240e-04	0.0021
1.5	13.3243674	13.3242859	8.1500e-05	0.002
2.0	25.4211284	25.4211318	3.4000e-06	0.0000

**Table 5.6** Timing, in seconds, for different values of  $c$  in Eq. 5.41 using GPOPS-II<sub>m</sub> on the necessary conditions.

$c$	T (s) GPOPS-II <sub>m</sub> Nec. Conds.	T (s) MOS	$ \Delta T $	$\Delta T\%$
-2.0	2.1144	0.5683	1.4317	111.4901
-1.5	2.9189	0.5993	2.3196	131.8629
-1.0	6.6841	0.6002	6.0839	167.0414
-.1	1.6347	1.1584	0.4763	34.1055
.1	0.5463	1.1029	0.5566	67.4994
.5	1.7410	1.1370	0.6040	41.9736
1.0	0.6641	1.6937	1.0296	87.3357
1.5	1.1085e+03	1.6050	1.1069e+03	199.4217
2.0	1.1301e+03	1.1123	1.1290e+03	199.6067

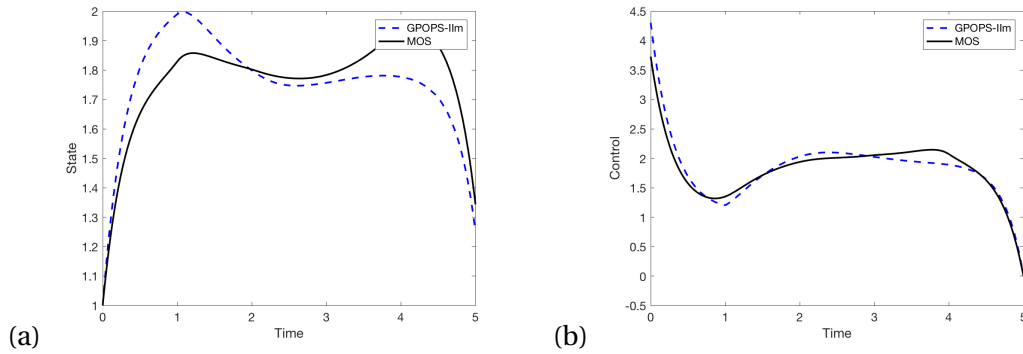
### 5.4.2 Changing the value on the state coefficient

We see that increasing the size of the delay term by increasing the coefficient  $c$  negatively impacts our results. Now we study the effect of changing the size of the undelayed state coefficient,  $a$ . We consider the example

$$\dot{x} = ax - 1.2x(t-1) + u \quad (5.42a)$$

$$J = \int_0^5 10(x-2)^2 + u^2 dt \quad (5.42b)$$

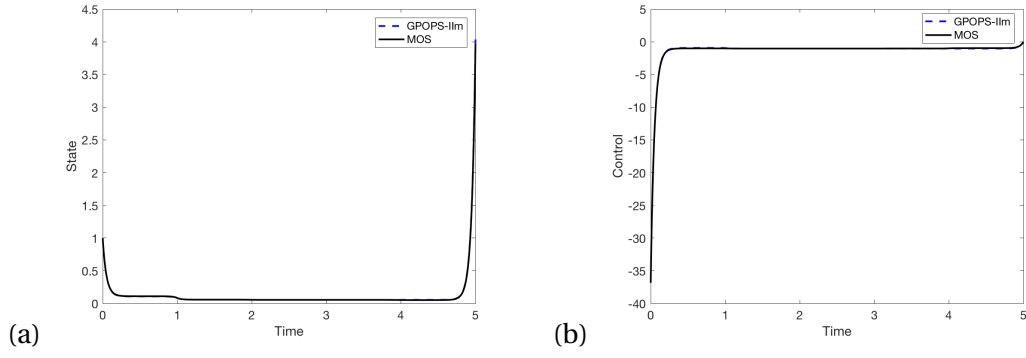
for several different values of  $a$ . We give costs and run times in Tables 5.7 and 5.8 respectively. We also include a small selection of plots. Our results in Table 5.7 as well as the plots in Fig. 5.50 - Fig. 5.53 demonstrate that GPOPS-IIIm produces better results for larger values of  $a$ , which is consistent with our analytic result from the previous section.



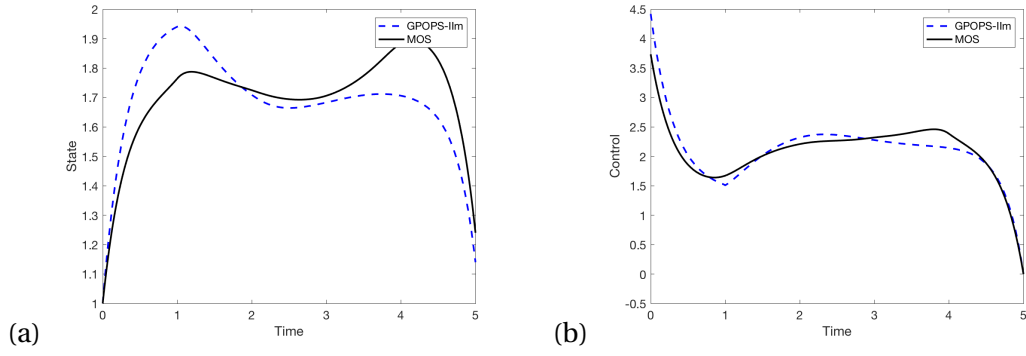
**Figure 5.50** (a) State and (b) control of Eq. 5.42 with  $a = 0.1$ ,  $c = -1.2$ , cost 22.2395200 from GPOPS-IIIm and cost  $J = 21.5224898$  from MOS.

#### 5.4.2.1 Necessary Conditions

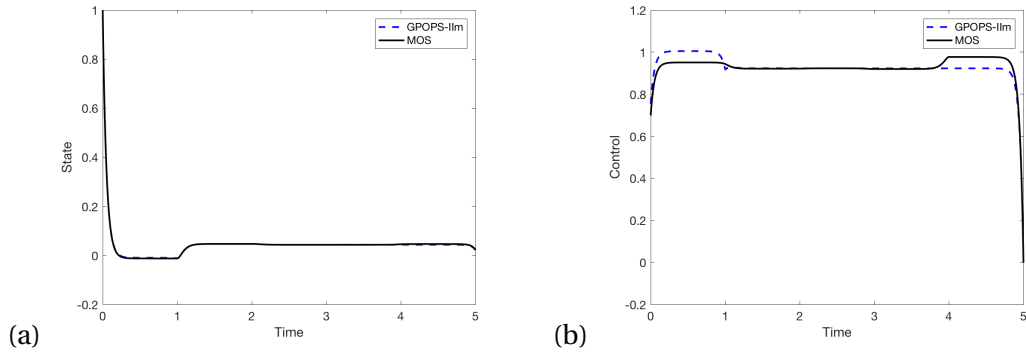
Our previous results are for a delayed OCP. Now we consider how varying the coefficient  $a$  Eq. 5.42 effects the solution of the necessary conditions for Eq. 5.42 with GPOPS-IIIm. We begin with mesh tolerance  $1e-05$ , IPOPT tolerance  $1e-06$  and one mesh iteration for  $a = -.5, -.1, .1, 0.5$ . We need to lower our mesh tolerance to  $1e-04$  to obtain an answer for  $a = 1.5, 1.0, -1.0, -1.5$ . We note that large values of  $a$  result in highly unstable dynamics, making the BVP difficult to solve. We can see from Table 5.10 that decreasing our mesh tolerance greatly increases our speed of execution. Both the values of our cost and the solution plots in Fig. 5.54 and Fig. 5.55 reflect our analytic result; we obtain far better results when  $a$  is larger than  $c$ .



**Figure 5.51** (a) State and (b) control of Eq. 5.42 with  $a = 20.0$ ,  $c = -1.2$ , cost 222.1349950 from GPOPS-IIIm and cost  $J = 222.1272862$  from MOS.



**Figure 5.52** (a) State and (b) control of Eq. 5.42 with  $a = -0.1$ ,  $c = -1.2$ , cost 29.2185129 from GPOPS-IIIm and cost  $J = 28.3062769$  from MOS.



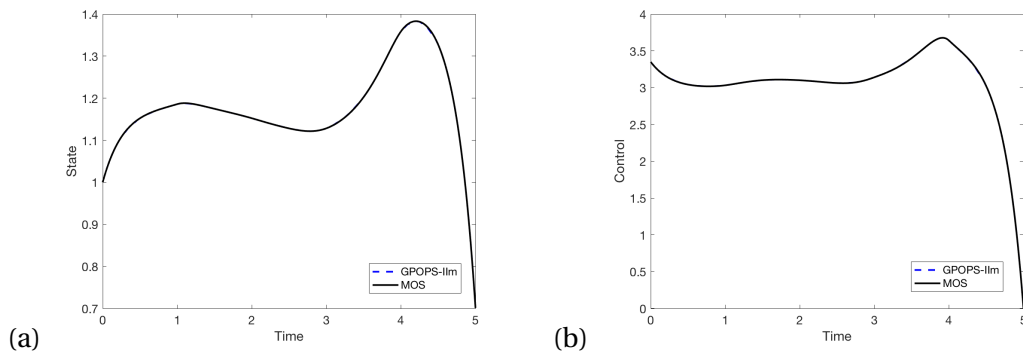
**Figure 5.53** (a) State and (b) control of Eq. 5.42 with  $a = -20.0$ ,  $c = -1.2$ , cost 196.2159127 from GPOPS-IIIm and cost  $J = 196.2104009$  from MOS.

**Table 5.7** Costs from GPOPS-IIIm and MOS for different values of  $a$  in Eq. 5.42 using GPOPS-IIIm.

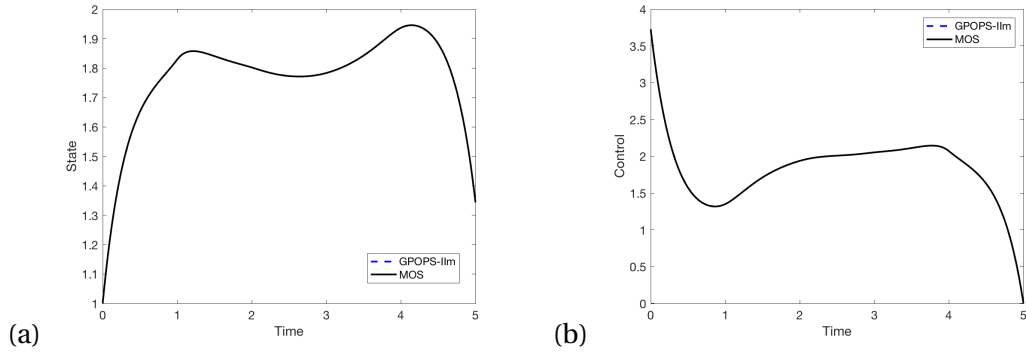
$a$	$J$ GPOPS-IIIm	$J$ MOS	$\Delta J$	$\Delta J\%$
-20	196.2159127	196.2104009	0.0055	0.0028
-5.0	159.4283099	159.0217559	0.4066	0.2553
-1.5	83.9586711	82.4496479	1.5090	1.8136
-.1	29.2185129	28.3062769	0.9122	3.1716
.1	22.2395200	21.5224898	0.7170	3.2770
.5	10.8533359	10.5200718	0.3333	3.1185
1.0	3.4373448	3.3986159	0.0387	1.1331
1.5	5.0562660	4.9662331	0.0900	1.7966
2.0	14.9002094	14.5105841	0.3896	2.6495
5.0	111.7284026	111.1001045	0.6283	0.5639
10.0	179.6720811	179.578356	0.0937	0.0522
20.0	222.1349950	222.1272862	0.0077	0.0035

**Table 5.8** Timing, in seconds, for different values of  $a$  in Eq. 5.42 using GPOPS-IIIm.

$a$	T (s) GPOPS-IIIm	T (s) MOS	$\Delta T$	$\Delta T\%$
-20	.9153	0.9151	.0002	0.0219
-5	1.2176	0.8239	.3937	38.5697
-.5	1.3842	0.6285	.7557	75.0932
-.1	3.1618	0.7938	2.3680	119.7290
.1	1.4243	0.7020	0.7223	67.9396
.5	.9969	.7979	0.1990	22.1752
1.0	1.1668	.6706	0.4962	54.0111
1.5	1.0224	.7673	0.2551	28.5076
2.0	1.0896	.7606	0.3290	35.5637
5.0	1.2759	.8230	0.4529	43.1559
10.0	1.0214	0.9437	0.0777	7.9080
20.0	.8577	1.1514	-0.2937	29.2370



**Figure 5.54** (a) State and (b) control using GPOPS-IIIm on the necessary conditions for Eq. 5.42 with  $a = -1.5$  and  $c = -1.2$  and cost  $J = 82.4499980$  from GPOPS-IIIm.



**Figure 5.55** (a) State and (b) control using GPOPS-IIIm on the necessary conditions for Eq. 5.42 with  $a = +0.1$  and  $c = -1.2$  and cost  $J = 21.5220325$  from GPOPS-IIIm.

**Table 5.9** Costs from GPOPS-IIIm and MOS for different values of  $a$  in Eq. 5.42 using GPOPS-IIIm on the necessary conditions.

$a$	$J$ GPOPS-IIIm Nec. Conds	$J$ MOS	$ \Delta J $	$\Delta J\%$
-1.5	82.4499980	82.4496479	3.5010e-04	4.2462e-04
-1.0	63.3406456	63.3403483	2.9730e-04	4.6937e-04
-0.5	43.4215062	43.4214094	9.6800e-05	2.2293e-04
-.1	28.3060628	28.3062769	2.1410e-04	7.5637e-04
.1	21.5220325	21.5224898	4.5730e-04	2.1248e-03
.5	10.5198523	10.5200718	2.1950e-04	2.0865e-03
1.0	3.3986579	3.3986159	4.2000e-05	1.2358e-03
1.5	4.9674099	4.9662331	1.1768e-03	2.3693e-02

**Table 5.10** Timing, in seconds, for different values of  $a$  in Eq. 5.42 using GPOPS-II on the necessary conditions.

$a$	T(s) GPOPS-IIIm Nec. Conds.	T (s) MOS	$ \Delta T $	$\Delta T\%$
-1.5	1.3868	0.7910	.59580	54.7158
-1.0	2.6896	1.4655	1.2241	58.9204
-.5	1.8231e+03	1.0034	1.82209e+03	199.7800
-.1	507.1222	0.9385	5.0618e+02	199.2611
.1	1.6796e+03	1.4613	1.6781e+03	199.6523
.5	1.3431e+03	0.9084	1.3422e+03	199.7296
1.0	1.6744	1.6233	0.0511	3.0991
1.5	0.6883	0.9414	0.2531	31.0609

## CHAPTER

# 6

## CONTROL DELAYS

In addition to using GPOPS-II<sub>m</sub> to solve optimal control problems (OCPs) with delays in the state, we consider OCPs with delays in the control. In this chapter we demonstrate that we are able to successfully solve an OCP with a constant delay in the control using GPOPS-II<sub>m</sub>, as well as using the necessary conditions to solve a BVP with a control delay.

We consider OCPs with control delays in the following form

$$J = \int_0^T 10(x(t)-r)^2 + u(t)^2 dt \quad (6.1a)$$

$$\dot{x} = ax(t) + bu(t) + du(t-s), \quad t \in (0, T] \quad (6.1b)$$

$$x(0) = x_0 \quad (6.1c)$$

$$u(t) = \psi(t) \quad t \in [-s, 0]. \quad (6.1d)$$

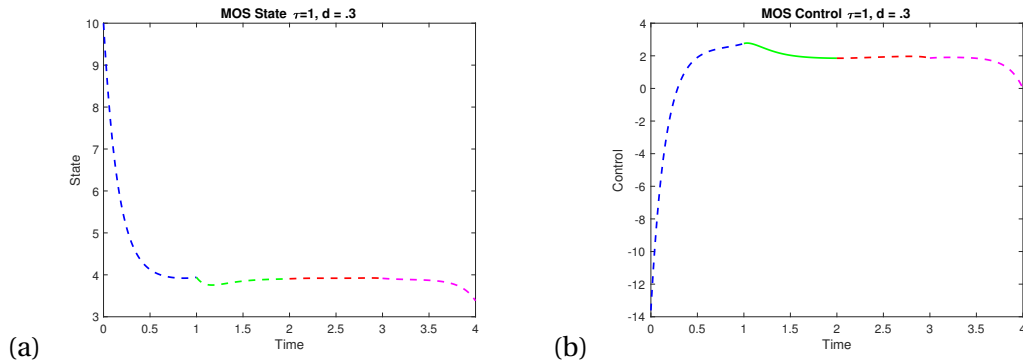
We consider the following approaches: the method of steps, solving the necessary conditions, and the GPOPS-II<sub>m</sub> approach originally described in Chapter 4. We are able to use the method of steps and data obtained from SOSD to validate our results from GPOPS-II.

## 6.1 Control Delays and MOS

We begin by solving Eq. 6.1 using MOS to give us data to which we can compare our results from GPOPS-II and SOSD. Initially we take

$$T = 4, x(0) = 10, r = 4, a = -1.14, b = 2, d = 0.3, s = 1, \text{ and } \psi = -2.3. \quad (6.2)$$

We first allow GPOPS-II to vary the order of the interpolating polynomial; this does not improve the quality of our approximation but does increase computational time. We therefore use GPOPS-II as an  $h$  method by fixing the order of the interpolating polynomial at  $P = 3$ . We set a mesh tolerance of  $1e-07$  and our tolerance for IPOPT at  $1e-06$ , and allow three mesh iterations with  $N = 4$  initial mesh intervals. We obtain a cost of  $J = 53.27062$ , as opposed to  $J = 53.27103$  obtained from SOSD.



**Figure 6.1** (a) State and (b) control obtained using MOS for Eq. 6.2 and cost  $J = 53.27062$ .

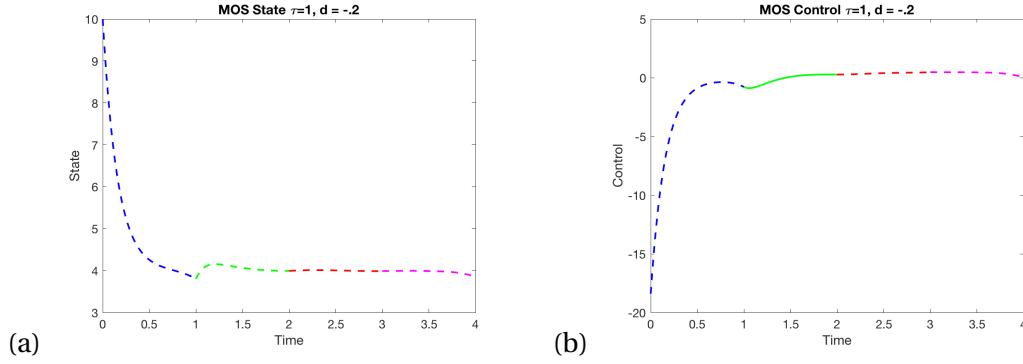
We now consider the example

$$T = 4, x(0) = 10, r = 4, a = -0.2, b = 2.0, d = -0.4, s = 1, \psi = -2.3 \quad (6.3)$$

which we solve using MOS. The solution plots in Fig. 6.2 demonstrate that we continue to see a corner in the state and control when the delay takes effect at time  $t = 1$ , indicating a place in our solution where we may expect GPOPS-II to have difficulty.

## 6.2 Solving the Necessary Conditions

In Chapter 4 we give the necessary conditions for a delayed state OCP. We then use GPOPS-II as a BVP solver to solve the necessary conditions and obtain an accurate solution to the delayed state OCP. In this section we demonstrate an identical approach may sometimes be used to obtain an



**Figure 6.2** (a) State and (b) control obtained using MOS for Eq. 6.3 and cost  $J = 56.18724$ .

accurate solution to an OCP with constant delay in the control.

Consider the following form of the problem given in Eq. 6.1:

$$\min J = \int_0^T 10(x(t)-4)^2 + u(t)^2 dt \quad (6.4a)$$

$$\text{subject to } \dot{x} = ax + bu + du(t-s) \quad (6.4b)$$

$$x(0) = 10 \quad (6.4c)$$

$$u(t) = \psi(t), -s \leq t \leq 0, \quad (6.4d)$$

where our  $u$  is unconstrained. Let  $v = u(t-s)$  where  $s$  is our constant delay. Define

$$H(t, x, u, v, \lambda) = f_0(t, x, u, v) + \lambda f(t, x, u, v). \quad (6.5)$$

Then we have the necessary conditions

$$\dot{x} = H_\lambda \text{ a.e. } t \in [0, T] \quad (6.6a)$$

$$\dot{\lambda} = -H_x \quad (6.6b)$$

$$\lambda(T) = 0 \quad (6.6c)$$

$$0 = H_u(t) + \chi_{[0, T-s]}(t)H_v(t+s) \quad (6.6d)$$

$$x(0) = 10, \quad (6.6e)$$

where *a.e.* stands for "almost everywhere."

For the scalar, linear case given in Eq. 6.4 we obtain the BVP

$$\dot{x} = ax(t) + bu(t) + du(t-s) \quad (6.7a)$$

$$\dot{\lambda} = -20(x(t) - r) - a\lambda(t) \quad (6.7b)$$

$$\lambda(T) = 0 \quad (6.7c)$$

$$0 = 2u(t) + b\lambda(t) + \chi_{[0, T-s]}(t)d\lambda(t+s) \quad (6.7d)$$

$$x(0) = 10. \quad (6.7e)$$

We can use the constraint in Eq. 6.7d to obtain the following expression for  $u(t)$

$$u(t) = -\frac{1}{2}(b\lambda(t) + \chi_{[0, T-s]}(t)d\lambda(t+s)) \quad (6.8)$$

and a shifted constraint to obtain an expression for the delay term  $u(t-s)$ :

$$u(t-s) = -\frac{1}{2}(b\lambda(t-s) + \chi_{[0, T-s]}(t-s)d\lambda(t)), \quad t \in (s, T]. \quad (6.9)$$

We note that  $\chi_{[s, T]}(t)$  on  $[s, T]$  is equivalent to  $\chi_{[0, T-s]}(t-s)$  on  $[0, T-s]$ . Then we have the dynamics

$$\dot{x} = ax(t) + b\left[-\frac{1}{2}(b\lambda(t) + \chi_{[s, T]}(t)d\lambda(t+s))\right] + d\psi(t), \quad t \in [0, s], \quad (6.10)$$

where  $u(t-s)$  is given by the prehistory  $\psi(t)$  for  $t \in [0, s]$  and

$$\begin{aligned} \dot{x} = ax(t) + b\left[-\frac{1}{2}(b\lambda(t) + \chi_{[0, T-s]}(t)d\lambda(t+s))\right] \\ + d\left[-\frac{1}{2}(b\lambda(t-s) + \chi_{[s, T]}(t)d\lambda(t))\right], \quad t \in (s, T] \end{aligned} \quad (6.11)$$

otherwise.

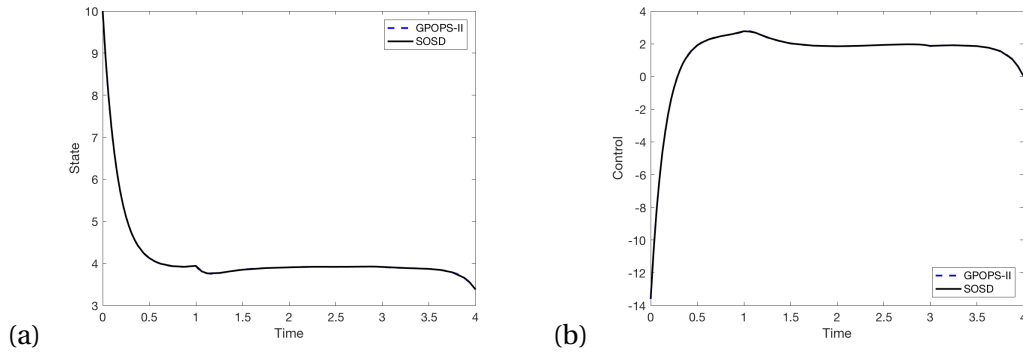
Having eliminated  $u$  from our equations we can now pass in our equations for  $x, \lambda$ , and  $J$  to GPOPS-IIIm as state variables where we add the dynamic constraint and boundary condition

$$\dot{J} = 10(x(t) - r)^2 + \left(-\frac{1}{2}(b\lambda + \chi_{[0, T-s]}(t)d\lambda(t+s))\right)^2, \quad J(0) = 0. \quad (6.12)$$

Our BVP is now given by Eq. 6.12, Eq. 6.11, Eq. 6.7b, and Eq. 6.7c. GPOPS-IIIm no longer has to optimize a control, and can instead integrate the dynamics to solve the BVP while satisfying the constraints. The result, which we demonstrate in the following section, is GPOPS-IIIm no longer relies on incorrect Jacobians obtained for the optimization problem and often finds the correct solution.

### 6.2.1 Numerical Results

We solve the necessary conditions for the example given in Eq. 6.2. We set our mesh tolerance at  $1e-07$  and our IPOPT tolerance at  $1e-06$ . For this problem, we find we need to set our initial number of mesh intervals to  $N = 20$  in order to obtain an initial grid fine enough to permit a solution. We allow three mesh iterations,  $m = 3$ , and fix the order of our polynomial at  $P = 3$ . For this problem we obtain a cost of  $J = 53.27083$  as opposed to  $J = 53.27062$  from MOS and  $J = 53.27103$  from SOSD. The plots in Fig. 6.3 demonstrate that not only do our costs agree very well, but that GPOPS-IIm obtains an accurate control and trajectory for Eq. 6.2.



**Figure 6.3** (a) State and (b) control obtained by GPOPS-IIm solving the necessary conditions for Eq. 6.2 with cost  $J = 53.27083$ .

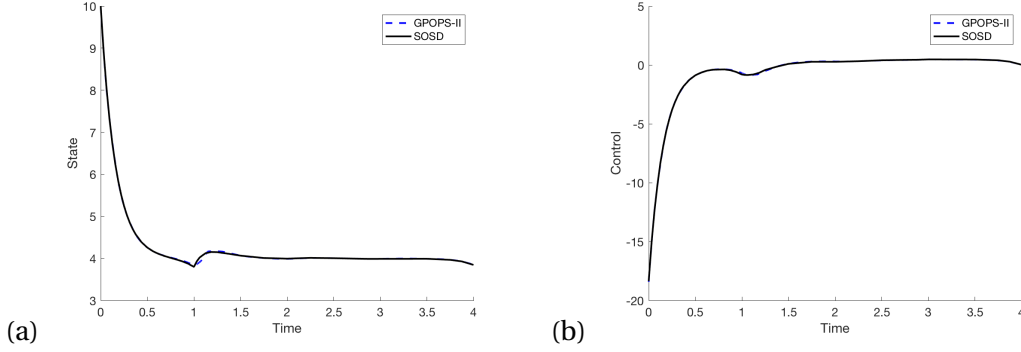
We then solve the necessary conditions for the example in Eq. 6.3. We again fix the order of our polynomial at  $P = 3$ , take  $N = 20$  and allow three mesh iterations. We need to lower our tolerances to  $1e-04$  and  $1e-06$  for our mesh and IPOPT respectively. This problem is exceptionally sensitive to the choice of initial value. The graphs in Fig. 6.4 are the solution for the following initial values

$$\begin{aligned} \text{tGuess} &= [0; 0.8125; 1; 1.5; 3; 4]; \\ \text{xGuess} &= [10, -16.3547, 0; 4, .1, 2; 0.38, 0.1, 4; 4.2, 0, 6; 4, 0, 8; 4, 0, 10]; \end{aligned}$$

where  $\text{xguess}$  is the array of column vectors  $[x, \lambda, J]$ .

The points in  $\text{tguess}$  are chosen to place two extra time grid points on either side of the discontinuity at  $t = s$ . The remaining time grid values are chosen to account for values toward the end of our interval. We set  $x(0)$  as our initial condition. To obtain  $\lambda(0) = -16.3547$ , we obtain an expression for  $\lambda$  in terms of  $u$ , and use the value of  $u(0)$  from SOSD. We set the initial value of the cost  $J = 0$ , and the final value of  $\lambda(T) = 0$ . The remaining state values are chosen via estimation of our graphical data from SOSD.

Adding additional points to the initial value for the control  $u$  greatly improves the quality of the solution. We can see from Fig. 6.4 that with a reasonable starting point informed by SOSD, the solutions from GPOPS-II are nearly identical to those produced by SOSD. Using GPOPS-II on the necessary conditions we obtain a cost of  $J = 56.027914$  while SOSD obtains a cost of  $J = 56.187740$ .



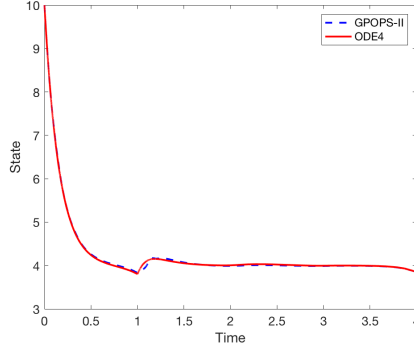
**Figure 6.4** (a) State and (b) control obtained by GPOPS-II solving the necessary conditions for Eq. 6.3 with cost  $J = 55.2559463$ .

In addition to the choice of initial value, another possible explanation for the sensitivity of the problem Eq. 6.3 may be found in the frequency domain. For the dynamics in Eq. 6.1b, we have the Laplace transform solution

$$X(\hat{t}) = (\hat{t} - a)^{-1} (b + e^{-\hat{t}s} d) U(\hat{t}), \quad (6.13)$$

where  $\hat{t}$  is our Laplace transform variable. In the first example, Eq. 6.2,  $b$  and  $d$  have the same sign, and we obtain a highly accurate solution from GPOPS-II. In the second example, Eq. 6.3,  $b$  and  $d$  have opposite signs: the coefficient in front of the control may become very small or even go to zero. The result is the control for the system will not be well-determined.

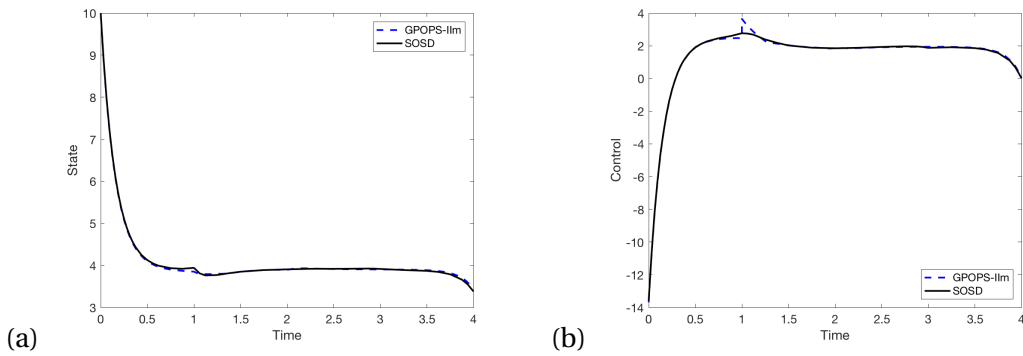
To verify that the control computed by GPOPS-II steers the dynamics along the correct trajectory we use the fixed grid MATLAB integrator ode4, a fourth order Runge-Kutta method. We pass the time grid and control from GPOPS-II to ode4 and integrate our dynamics with the values given in Eq. 6.3. The plot in Fig. 6.5 demonstrates that state obtained GPOPS-II and the state obtained by integrating the dynamics with ode4 are essentially identical. Our results show that for a sufficiently accurate choice of initial value in the state and control variables, we able to use GPOPS-II to solve an OCP with a delay in the control.



**Figure 6.5** State obtained by GPOPS-II and obtained by ode4 using the control computed by GPOPS-II<sub>m</sub> solving the necessary conditions for Eq. 6.3.

### 6.3 Control Delays and GPOPS-II<sub>m</sub>

We now use the GPOPS-II<sub>m</sub> method discussed earlier in Chapter 4 to solve the example in Eq. 6.1 and Eq. 6.2. We consider the problem as an  $h$  method with fixed polynomial order  $P = 3$  as well as allowing  $P$  to vary between 2 and 6. We keep the same tolerances utilized in the MOS example. For fixed order  $P = 3$  we obtain cost  $J = 53.3702471$ . When we allow the order to vary we obtain a slightly higher cost of  $J = 53.3764644$ . This is most likely due to the larger grid  $N = 83$  points obtained when we allow the polynomial order to vary as opposed to the grid  $N = 73$  points obtained when we fixed  $P = 3$ . The states and controls produced are essentially identical, therefore we will only be including the plots for fixed  $P = 3$ . We can see from Fig. 6.6 that while GPOPS-II<sub>m</sub> approximates the state from SOSD very well, it introduces a sharp discontinuity into the control at time  $t = 1$  when the delay in the control takes effect.



**Figure 6.6** (a) State and (b) control using GPOPS-II<sub>m</sub> for Eq. 6.2 with  $J = 53.3702471$ .

In order to improve our answer from GPOPS-II<sub>m</sub> we consider splitting the problem into two phases, with the break between the phases at time  $t = s$ . We implement this strategy for the problem given above in Eq. 6.1 and Eq. 6.2. We set our tolerances for the mesh and the NLP solver IPOPT at  $1e-07$  and  $1e-06$  respectively. We fix the order at  $P = 3$  and allow three mesh iterations. Splitting the problem into two phases does not improve the accuracy of the state or control but slightly lowers cost to  $J = 53.3369513$ .

In our section on using the necessary conditions to solve Eq. 6.2 we note that taking a sufficiently fine initial grid is key to obtaining a solution. We test if using GPOPS-II<sub>m</sub> and a finer initial grid will result in a more accurate answer. Increasing  $N$  has a negligible affect on our cost and on our solution trajectories, however our results in Table 6.1 demonstrate that increasing the number of initial mesh intervals  $N$  from 4 to 10 significantly decreases our run time. We note that increasing  $N$  significantly beyond 20 does not continue to improve our run time. All timings are conducted using the MATLAB `tic` and `toc` functions on a late 2013 15 inch MacBook Pro with a 2.6 GHz Intel Core i7 processor.

We now use GPOPS-II<sub>m</sub> to solve Eq. 6.3, and encounter the same difficulties as using the necessary conditions. We take  $N = 20$ ,  $P = 3$ , and allow three mesh iterations. We need to reduce our mesh tolerance to  $1e-04$  and our IPOPT tolerance to  $1e-06$ . We use the same technique of adding extra points to our initial values. For Eq. 6.3 we use

```
xGuess          = [x0;4; xf];
uGuess          = [-2.3; -0.3796; -0.8118] ;
guess.phase(1).time      = [t0;0.8125; t2];
guess.phase(1).state     = xGuess;
guess.phase(1).control   = uGuess;
guess.phase(1).integral  = 0;
guess.phase(2).time      = [t2;1.5;tf];
guess.phase(2).state     = [4;4; 4];
guess.phase(2).control   = [-0.8118; 0.0933 ;0.0211];
guess.phase(2).integral  = 0;
```

Given the sensitivity of Eq. 6.3 to the initial values in the control, we use data from SOSD to provide our initial value for the control. Using the two-phase formulation of Eq. 6.3 we obtain a cost of  $J = 55.8079917$ , lower than the cost from SOSD at  $J = 56.187740$ . We note that while it is possible the optimal cost is lower than that obtained by SOSD, it is more likely that GPOPS-II<sub>m</sub> is reducing the cost by violating the continuity constraint at  $t = 1$ .

We therefore choose to return to the single phase formulation of the problem. We take  $N = 20$ ,  $P = 3$ , and allow three mesh iterations with the initial values

```
tGuess = [0; 0.8125; 1; 1.5; 5];
```

```

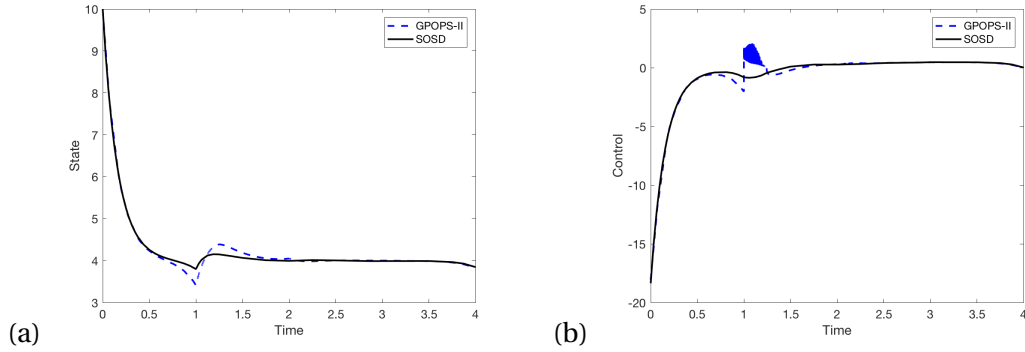
xGuess = [10; 4; 4; 4; 4];
uGuess = [7; -.3796; -0.8118; 0; 0];

```

where our initial values for the time and control are data from SOSD, and our state values are chosen to correspond with the reference trajectory  $r = 4$ . We keep our tolerances at  $1e-04$  and  $1e-06$  for the mesh and IPOPT respectively, and obtain cost  $J = 57.1385819$ . We plot the single-phase solution in Fig. 6.7. While GPOPS-II<sub>m</sub> captures the behavior of the state in Fig. 6.7(a), it introduces significant noise into the control in Fig. 6.7(b).

Due to the noise introduced in the control by IPOPT, our answers vary slightly each time we used GPOPS-II<sub>m</sub> to solve Eq. 6.3 with a different number of initial mesh intervals. All the variation we observe in our cost is within our numerical tolerances. We continue to utilize initial  $N = 20$  in order to decrease the computational time.

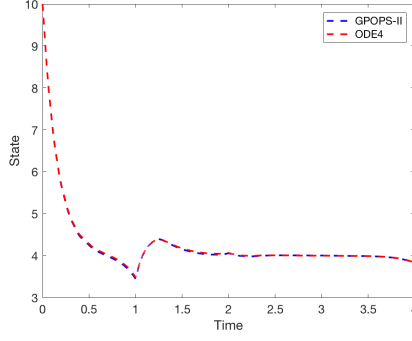
We again seek to gauge the quality of the control computed by GPOPS-II<sub>m</sub>. We utilize the fixed grid integrator `ode4` from MATLAB to test if the computed control  $u$  from GPOPS-II<sub>m</sub> will steer our dynamics along the correct trajectory. We can see from Fig. 6.8 that the control from GPOPS-II<sub>m</sub> produces the correct state, in spite of the noise.



**Figure 6.7** (a) State and (b) control for Eq. 6.3 using GPOPS-II<sub>m</sub> with cost  $J = 57.1385819$ .

**Table 6.1** Run times, in seconds, and cost for Eq. 6.2 using GPOPS-II<sub>m</sub> and the two phase problem formulation.

$N$	Run Time (s)	Cost
4	208.5892	53.3369513
10	4.2490	53.3392184
20	3.7683	53.3395318
30	4.9627	53.3398289



**Figure 6.8** Comparison of states from GPOPS-II and ode4 for Eq. 6.3.

### 6.3.1 Smoothing the Control

While Fig. 6.8 demonstrates that GPOPS-II is correctly integrating our dynamics, the amount of noise in the control is suboptimal. We seek to reduce the noise we see in the controls produced by GPOPS-II in Fig. 6.6 and Fig. 6.7.

#### 6.3.1.1 Adding Additional Dynamic Equation

In [12] the authors introduce pseudovariables for the delay terms to reduce the noise in their results. We examine a similar approach. When the delay takes effect at  $t = s$ , we see a sharp increase in our control  $u$ . This increase is associated with a large value of the first derivative,  $\dot{u}$ . We add a new control variable,  $z$ , and add the dynamic equation  $\dot{u} = z$ . We then add  $z$  to our cost quadratically,  $\epsilon \|z\|^2$ . Now  $u$  is implemented as a state variable,  $u(t-s)$  is a state delay, and  $z$  is our new control. By adding  $z$  to the cost, we encourage the optimizer to minimize the first derivative of  $u$ . Our problem formulation for this approach is:

$$\min_u J = \int_0^5 10(x(t)-r)^2 + u(t)^2 + \epsilon z^2 dt \quad (6.14a)$$

$$\dot{x} = ax(t) + bu(t) + du(t-s) \quad (6.14b)$$

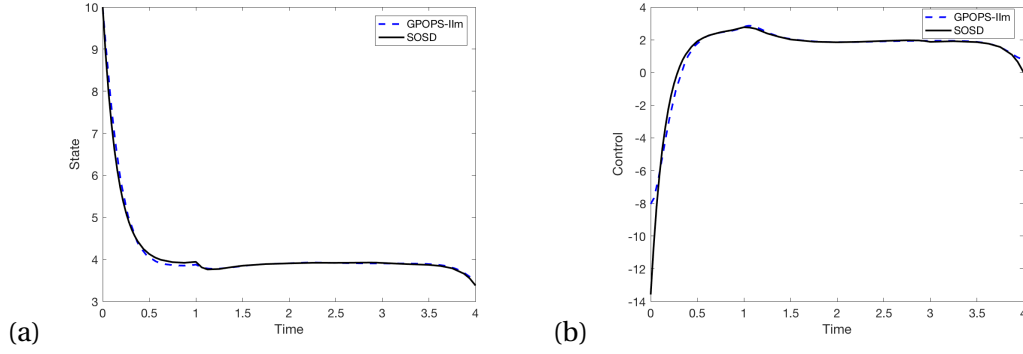
$$\dot{u} = z \quad (6.14c)$$

$$x(0) = x_0 \quad (6.14d)$$

$$u(t) = \psi(t), \quad t \in [-s, 0]. \quad (6.14e)$$

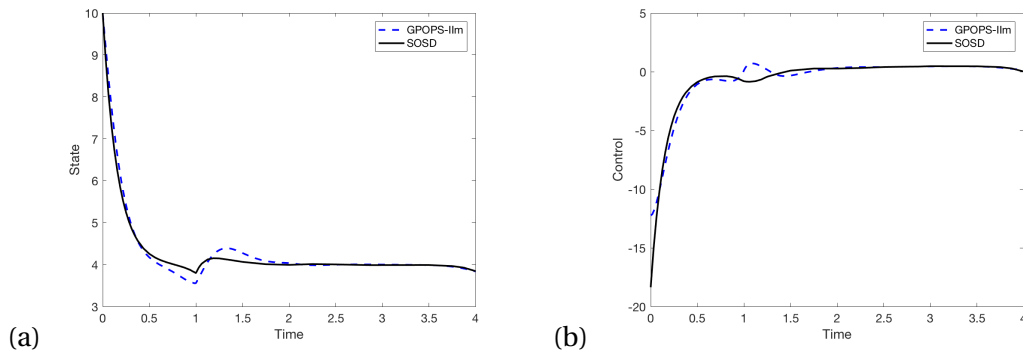
We first consider Eq. 6.2 using the problem formulation in Eq. 6.14, with  $\epsilon = .01$ . We keep our tolerances at  $1e-07$ , take three mesh iterations, set  $P = 3$ , and the initial number of mesh intervals  $N = 20$ . We can see from Fig. 6.9 our approach is highly effective, and has removed the discontinuity from  $u$  in Fig. 6.9(b). We obtain cost 56.824463 from GPOPS-II as compared to  $J = 53.27103$

from SOSD. We expect our cost from GPOPS-IIIm to increase due to the  $\epsilon \|z\|^2$  term in the integral. Subtracting  $\int_0^4 \epsilon \|z\|^2 dt = 2.423768$  from our GPOPS-IIIm cost we obtain a new cost  $J = 54.400695$  from GPOPS-IIIm, much closer to that from SOSD.



**Figure 6.9** (a) State and (b) control for Eq. 6.2 using GPOPS-IIIm and adding  $\dot{u} = z$  to the dynamics, cost  $J = 56.824463$ .

We now test the approach in Eq. 6.14 on the second example given in Eq. 6.3, again with  $\epsilon = .01$ . We note this problem is more sensitive than the previous example. In order to obtain an answer we need to reduce our mesh tolerance to  $1e-03$  and our IPOPT tolerance to  $1e-04$ . We take the initial number of mesh intervals  $N = 20$ , our polynomial order  $P = 3$ , and three mesh iteration. We obtain cost  $J = 61.665679$  from GPOPS-IIIm, higher than  $J = 56.187740$  from SOSD. The value of cost after we subtract the integral of  $\epsilon \|z\|^2$  is  $J = 58.529214$ , much closer to that of SOSD. The control Fig. 6.10(b) demonstrates that we are able to remove most of the noise from the solution.

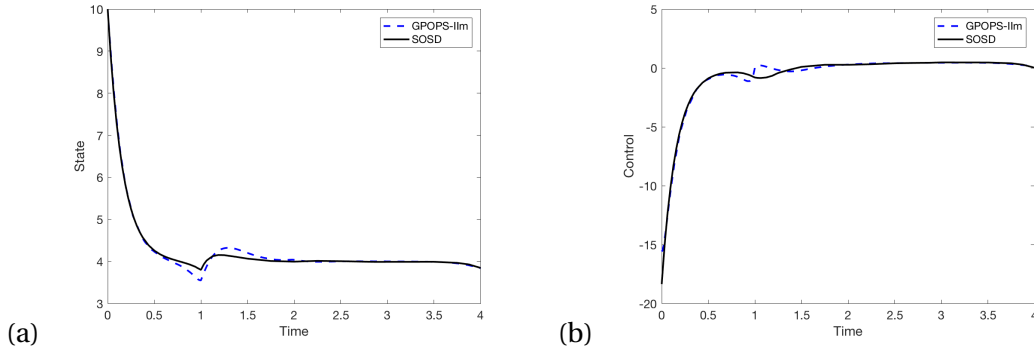


**Figure 6.10** (a) State and (b) control for Eq. 6.3 using GPOPS-IIIm and adding  $.01\|z\|^2$  to cost, with  $J = 61.665679$ .

While we see improved results for Eq. 6.3 using  $\epsilon = .01$  we consider we may be able to further improve our results using a variable definition. We can see in Fig. 6.10(b) that the control for this problem is extremely steep on  $[0, 1]$ , and thus will have a much larger first derivative on that interval. We therefore define epsilon as

$$\begin{cases} \epsilon = .001 & \text{if } t \in [0, 1] \\ \epsilon = .01 & \text{else} \end{cases} \quad (6.15)$$

in order to control the size of the derivative  $u' = z$  on  $[0, 1]$ . Using this definition, we obtain  $J = 57.5017110$  and  $\int_0^4 \epsilon \|z\|^2 dt = 0.707674$ . Subtracting the value of our integral from our cost  $J$ , we obtain the new cost of  $J = 56.794046$  from GPOPS-IIIm, which is much closer to  $J = 56.187740$  from SOSD. We can see from Fig. 6.11 that changing our definition of  $\epsilon$  results in a slightly more accurate trajectory from GPOPS-IIIm.



**Figure 6.11** (a) State and (b) control for Eq. 6.3 using GPOPS-IIIm and adding  $\epsilon \|z\|^2$  to cost,  $\epsilon$  as in Eq. 6.15, with  $J = 57.5017110$ .

## CHAPTER

# 7

# CONCLUSIONS AND FUTURE RESEARCH

In this chapter we discuss our contributions and future research questions. We give an overview of the new numerical techniques introduced in this thesis and the conclusions we may draw from our results. We discuss our analytic results and when we may expect GPOPS-II to perform well on certain types of delay problems. We then discuss opportunities for future work, including new methods for smoothing the control produced by GPOPS-II and methods for improving our results using Activate.

## 7.1 Conclusions and Contributions

In this thesis, we seek to address the lack of a user-friendly delayed OCP solver for MATLAB. We have examined the application of the MATLAB-based software GPOPS-II to delay problems. We explore the use of GPOPS-II as an integrator of delay differential equations, an optimizer on optimal control problems with delays, and as a delay boundary value problem solver on the necessary conditions for optimal control problems with delays. We establish two separate methods for formulating delayed OCP such that GPOPS-II will accept them. The first method, which we refer to as GPOPS-II<sub>m</sub>, allows the user to implement delay terms directly by interpolating the delay.

We demonstrate GPOPS-II<sub>m</sub> is effective at solving DDEs without a control. For OCPs with delays, GPOPS-II<sub>m</sub> performs well on problems with small delays, and produces suboptimal solutions for longer delays. We establish a result for the OCP with constant delay, which demonstrates that the nonlinear optimizer will converge when the delay term is small relative to the Jacobian of the system.

We demonstrate the effectiveness of GPOPS-II<sub>m</sub> as a delay BVP solver on the necessary conditions, introducing a new tool for MATLAB users. We show when GPOPS-II<sub>m</sub> is able to compute optimal solutions for the BVP, and present a result which gives a sufficient condition for convergence.

GPOPS-II<sub>m</sub> can be a highly effective tool for solving delay problems in MATLAB. However, the GPOPS-II<sub>m</sub> formulation is highly sensitive to the choice of initial value, particularly in the control. We are able to mitigate some of the sensitivity by the addition of extra points in the vector of initial values. When good data are available, we use these data to inform our choice of initial value and improve our results. For OCPs and BVPs with delays in the control, GPOPS-II<sub>m</sub> can produce a noisy control. We demonstrate that by adding the additional dynamic equation  $u' = z$  and adding  $\epsilon \|z\|^2$  to our cost,  $\epsilon < 1$ , we are able to almost completely smooth out the noise in the computed control.

Computational times can run long when using GPOPS-II<sub>m</sub> on a problem with a large delay or highly unstable dynamics. We therefore explore a second method to apply GPOPS-II to problems with delays in the state: the use of an iterative method. We first demonstrate the efficacy of using GPOPS-II and waveform relaxation on a DDE and then extend the approach to a delayed OCP, which we refer to as waveform optimization and abbreviate as GPOPS-II<sub>ow</sub>. GPOPS-II<sub>ow</sub> runs quickly even for larger delays and produces suboptimal solutions. We show, using the necessary conditions, that the amount of suboptimality does not decrease with the size of the delay. The utility of the GPOPS-II<sub>ow</sub> approach is therefore problem-dependent.

Many of the results we have just discussed appear in the paper "Examination of solving optimal control problems with delays using GPOPS-II", which we published with Dr. Stephen Campbell and Dr. John Betts [11]. The paper discusses our work with GPOPS-II<sub>ow</sub> on problems with state delays and our work with GPOPS-II<sub>m</sub> on problems with state and control delays. We present our result on the sufficient condition for convergence of the BVP.

In addition to the work presented in this thesis, we address the sensitivity of GPOPS-II to the choice of initial value in the paper "Initial guess sensitivity in computational optimal control problems" [10] which we published with Dr. Stephen Campbell and Dr. John Betts. We demonstrate that, for a nonlinear optimal control problem, the choice of initial value can change the extrema to which the software converges. For highly nonlinear problems, the same initial value can generate fluctuations in the computed extrema.

## 7.2 Future Research

In the preceding section we give an account of the contributions of this thesis. In this section, we discuss further areas of interest and possible applications of our results.

### 7.2.1 Smoothing the Control

As we discuss in Chapter 6 and in our Conclusions section, GPOPS-II can produce noisy controls when solving OCPs and BVPs with a delay in the control. We find one method of smoothing the control involves the introduction of an additional dynamic equation  $u' = z$  and adding  $\epsilon \|z\|^2$  to our cost. We find that the definition of  $\epsilon$  is key to obtaining a good result. Future research questions include how to determine the best choice of  $\epsilon$  to obtain the most accurate solution to the OCP.

We also note the importance of the choice of initial value for the control when obtaining a good solution. GPOPS-II allows the user to optimize over static parameters. We therefore consider making the prehistory for the control  $\psi$  in Eq. 6.4 a parameter in our optimization.

### 7.2.2 Identification of Delay

In addition to making the prehistory a parameter, we may consider applications of parameterizing the value of the delay. In some important problems, the value of the delay is a parameter in the model. Consider crystal growth problems: the time at which the crystallization process occurs may be not be known. See, for example, [37]. The user may introduce the value of the delay as a parameter in GPOPS-II, so the software can optimize over possible delay values.

### 7.2.3 Applications of Mesh Refinement for an OCP in Activate

In Section 5.3 we present results using the software Activate to solve an OCP with delay in the state. We are able to use the Time Delay and BobyqaOpt blocks to implement the delayed dynamics and solve the optimal control problem. However, when we compare our results using Activate to those from SOSD we see that the solution computed by Activate is suboptimal. As we discuss in Chapter 3, SOSD is a direct transcription code and has the advantage of powerful mesh refinement schemes.

The Activate routine involves repeated calls to the optimizer in the BobyqaOpt block, where the parameters are passed from the main simulation environment to the optimizer via the 'GetFromBase' argument [65]. See Fig. 5.36 and Fig. 5.37 for an example of a simulation diagram in Section 5.3. The BobyqaOpt block will restart the simulation until the specified error tolerances are met. We propose the introduction of a mesh refinement scheme which would refine the vector of parameters passed to the optimizer at the re-start of each simulation run. There are several options for mesh refinement, including an  $h$  method as employed by SOSD or a global  $p$  method as employed in a general pseudospectral method, which we discuss in Chapter 3. Currently, parameters in Activate are defined by the user. A better choice would be to select collocation points in the mesh refinement scheme according to a type of Gaussian quadrature rule.

## BIBLIOGRAPHY

- [1] *Altair Activate: Multi-Disciplinary system simulation*. 2018. URL: <https://www.altair.com/mbd2019/activate/>.
- [2] Antsaklis, P. J. & Michel, A. N. *Linear systems*. Springer Science & Business Media, 2006.
- [3] Bartoszewski, Z & Kwapisz, M. “On the convergence of waveform relaxation methods for differential-functional systems of equations”. *Journal of Mathematical Analysis and Applications* **235.2** (1999), pp. 478–496.
- [4] Bartoszewski, Z & Kwapisz, M. “On error estimates for waveform relaxation methods for delay-differential equations”. *SIAM Journal on Numerical Analysis* **38.2** (2000), pp. 639–659.
- [5] Becerra, V. M. “Solving complex optimal control problems at no cost with PSOPT”. *2010 IEEE International Symposium on Computer-Aided Control System Design*. IEEE. 2010, pp. 1391–1396.
- [6] Bellen, A. & Zennaro, M. *Numerical methods for delay differential equations*. Oxford University Press, 2013.
- [7] Bellman, R. “On the computational solution of differential-difference equations”. *Journal of Mathematical Analysis and Applications* **2.1** (1961), pp. 108–110.
- [8] Bellman, R. & Cooke, K. L. *Differential difference equations*. Academic Press, 1963.
- [9] Betts, J. T. *Practical methods for optimal control and estimation using nonlinear programming*. Vol. 19. SIAM, 2010.
- [10] Betts, J. T., Campbell, S. L. & Digirolamo, C. “Initial guess sensitivity in computational optimal control problems”. *Numerical Algebra, Control & Optimization* **10.1** (2020), pp. 39–41.
- [11] Betts, J. T., Campbell, S. L. & Digirolamo, C. M. “Examination of solving optimal control problems with delays using GPOPS-II”. *To appear in Numerical Algebra, Control, & Optimization* (2020).
- [12] Betts, J. T., Campbell, S. L. & Thompson, K. C. “Solving optimal control problems with control delays using direct transcription”. *Applied Numerical Mathematics* **108** (2016), pp. 185–203.
- [13] Betts, J. *Sparse optimization suite, SOS, user’s guide, release 2015.11*. 2016.
- [14] Biegler, L. T. & Zavala, V. M. “Large-scale nonlinear programming using IPOPT: An integrating framework for enterprise-wide dynamic optimization”. *Computers & Chemical Engineering* **33.3** (2009), pp. 575–582.
- [15] Bjørhus, M. “On dynamic iteration for delay differential equations”. *BIT Numerical Mathematics* **34.3** (1994), pp. 325–336.

- [16] Bonalli, R., Hérissé, B. & Trélat, E. "Solving optimal control problems for delayed control-affine systems with quadratic cost by numerical continuation". *2017 American Control Conference (ACC)*. IEEE. 2017, pp. 649–654.
- [17] Brandt-Pollmann, U., Winkler, R., Sager, S., Moslener, U. & Schlöder, J. P. "Numerical solution of optimal control problems with constant control delays". *Computational Economics* **31.2** (2008), pp. 181–206.
- [18] Bryson, A. E. "Optimal control-1950 to 1985". *IEEE Control Systems Magazine* **16.3** (1996), pp. 26–33.
- [19] Campbell, S. L. & Nikoukhah, R. *Modeling and simulation with compose and activate*. Springer, 2018.
- [20] Canuto, C., Hussaini, M. Y., Quarteroni, A. & Zang, T. A. *Spectral methods: Fundamentals in single domains*. Springer Science & Business Media, 2007.
- [21] Corwin, S., Sarafyan, D & Thompson, S. "DKLAG6: A code based on continuously imbedded sixth-order Runge-Kutta methods for the solution of state-dependent functional differential equations". *Applied Numerical Mathematics* **24.2-3** (1997), pp. 319–330.
- [22] Darby, C. L., Hager, W. W. & Rao, A. V. "An hp-adaptive pseudospectral method for solving optimal control problems". *Optimal Control Applications and Methods* **32.4** (2011), pp. 476–502.
- [23] Dell’Elce, L. & Scheeres, D. J. "Sensitivity of optimal control problems arising from their Hamiltonian structure". *The Journal of the Astronautical Sciences* (2018), pp. 1–13.
- [24] Elnagar, G., Kazemi, M. A. & Razzaghi, M. "The pseudospectral Legendre method for discretizing optimal control problems". *IEEE Transactions on Automatic Control* **40.10** (1995), pp. 1793–1796.
- [25] Erneux, T. *Applied delay differential equations*. Vol. 3. Springer Science & Business Media, 2009.
- [26] Fahroo, F & Ross, I. M. "Direct trajectory optimization by a Chebyshev pseudospectral method". *Journal of Guidance, Control, and Dynamics* **25.1** (2002), pp. 160–166.
- [27] Frankena, J. "Optimal control problems with delay, the maximum principle and necessary conditions". *Journal of Engineering Mathematics* **9.1** (1975), pp. 53–64.
- [28] Garg, D., Patterson, M., Hager, W., Rao, A., Benson, D. & Huntington, G. "An overview of three pseudospectral methods for the numerical solution of optimal control problems". AAS/AIAA Astrodynamics Specialist Conference. Pittsburgh, PA, 2009.

- [29] Garg, D., Patterson, M., Hager, W. W., Rao, A. V., Benson, D. A. & Huntington, G. T. "A unified framework for the numerical solution of optimal control problems using pseudospectral methods". *Automatica* **46.11** (2010), pp. 1843–1851.
- [30] Garg, D., Patterson, M. A., Francolin, C., Darby, C. L., Huntington, G. T., Hager, W. W. & Rao, A. V. "Direct trajectory optimization and costate estimation of finite-horizon and infinite-horizon optimal control problems using a Radau pseudospectral method". *Computational Optimization and Applications* **49.2** (2011), pp. 335–358.
- [31] Gill, P. E., Murray, W. & Saunders, M. A. *User's Guide for SNOPT Version 7: Software for large-scale nonlinear programming, Systems Optimization Laboratory (SOL)*. 2006.
- [32] Göllmann, L., Kern, D. & Maurer, H. "Optimal control problems with delays in state and control variables subject to mixed control–state constraints". *Optimal Control Applications and Methods* **30.4** (2009), pp. 341–365.
- [33] Göllmann, L. & Maurer, H. "Theory and applications of optimal control problems with multiple time-delays". *Journal of Industrial & Management Optimization* **10.2** (2014), p. 413.
- [34] Górecki, H., Fuska, S., Grabowski, P. & A., K. *Analysis and synthesis of time delay systems*. John Wiley & Sons Inc, 1989.
- [35] Guinn, T. "Reduction of delayed optimal control problems to nondelayed problems". *Journal of Optimization Theory and Applications* **18.3** (1976), pp. 371–377.
- [36] Hale, J. "History of delay equations". *Delay Differential Equations and Applications*. Springer, 2006, pp. 1–28.
- [37] Hillis, B. G., Losey, B. P., Weng, J., Ghaleb, N., Hou, F. & Martin, J. D. "From rate measurements to mechanistic data for condensed matter reactions: A case study using the crystallization of  $[\text{Zn}(\text{OH}_2)(6)][\text{ZnCl}_4]$ ". *Crystals* **7.1** (2017).
- [38] Hindmarsh, A. C. & Petzold, L. R. "LSODA, Ordinary Differential Equation Solver for Stiff or Non-Stiff System" (2005). URL: <http://www.nea.fr/abs/html/uscd1227.html>.
- [39] Izák, M., Görges, D. & Liu, S. "Optimal control of networked control systems with uncertain time-varying transmission delay". *IFAC Proceedings Volumes* **43.19** (2010), pp. 13–18.
- [40] Jahromi, A. F., Xie, W. F. & Bhat, R. B. "Ride control of passenger cars with semiactive suspension system using a linear quadratic regulator and hybrid optimization algorithm". *Proceedings of the International Conference on Aerospace, Mechanical, Automotive and Materials Engineering (ICAMAME'12)*. 2012, pp. 1083–1089.
- [41] Kalman, R. E. et al. "Contributions to the theory of optimal control". *Bol. Soc. Mat. Mexicana* **5.2** (1960), pp. 102–119.

- [42] Karbowski, A. "Shooting methods to solve optimal control problems with state and mixed control-state constraints". *International Conference on Automation*. Springer. 2016, pp. 189–205.
- [43] Karush, W. "Minima of functions of several variables with inequalities as side constraints". *M. Sc. Dissertation. Dept. of Mathematics, Univ. of Chicago* (1939).
- [44] Kharatishvili, G. L. "The maximum principle in the theory of optimal processes involving delay". *Doklady Akademii Nauk*. Vol. 136. 1. Russian Academy of Sciences. 1961, pp. 39–42.
- [45] Khellat, F. "Optimal control of linear time-delayed systems by linear Legendre multiwavelets". *Journal of Optimization Theory and Applications* **143**.1 (2009), pp. 107–121.
- [46] Kierzenka, J. & Shampine, L. F. "A BVP solver based on residual control and the MATLAB PSE". *ACM Transactions on Mathematical Software* **27**.3 (2001), pp. 299–316.
- [47] Krotov, V. & Kurzanski, A. "National achievements in control theory (The aerospace perspective)". *IFAC Proceedings Volumes* **37**.6 (2004), pp. 37–48.
- [48] Kye, W.-H., Choi, M., Rim, S., Kurdoglyan, M., Kim, C.-M. & Park, Y.-J. "Characteristics of a delayed system with time-dependent delay time". *Physical Review E* **69**.5 (2004), 055202(R).
- [49] Kyrychko, Y. & Hogan, S. "On the use of delay equations in engineering applications". *Journal of Vibration and Control* **16**.7-8 (2010), pp. 943–960.
- [50] Lelarsmee, E., Ruehli, A. E. & Sangiovanni-Vincentelli, A. L. "The waveform relaxation method for time-domain analysis of large scale integrated circuits". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **1**.3 (1982), pp. 131–145.
- [51] Lewis, F. L., Vrabie, D. & Syrmos, V. L. *Optimal control*. John Wiley & Sons, 2012.
- [52] Liu, F., Hager, W. W. & Rao, A. V. "Adaptive mesh refinement method for optimal control using nonsmoothness detection and mesh size reduction". *Journal of the Franklin Institute* **352**.10 (2015), pp. 4081–4106.
- [53] Maleki, M. & Hashim, I. "Adaptive pseudospectral methods for solving constrained linear and nonlinear time-delay optimal control problems". *Journal of the Franklin Institute* **351**.2 (2014), pp. 811–839.
- [54] Marzban, H. R. & Tabrizidooz, H. R. "Solution of nonlinear delay optimal control problems using a composite pseudospectral collocation method". *Commun. Pure Appl. Anal* **9**.13791389 (2010), p. 18.
- [55] Marzban, H. & Hoseini, S. "Solution of linear optimal control problems with time delay using a composite Chebyshev finite difference method". *Optimal Control Applications and Methods* **34**.3 (2013), pp. 253–274.

- [56] Michiels, W. & Niculescu, S.-I. *Stability, control, and computation for time-delay systems: an eigenvalue-based approach*. Vol. 27. SIAM, 2014.
- [57] Nocedal, J., Wächter, A. & Waltz, R. A. “Adaptive barrier update strategies for nonlinear interior methods”. *SIAM Journal on Optimization* **19.4** (2009), pp. 1674–1693.
- [58] Norkin, S. B. et al. *Introduction to the theory and application of differential equations with deviating arguments*. Vol. 105. Academic Press, 1973.
- [59] Patterson, M. A., Hager, W. W. & Rao, A. V. “A ph mesh refinement method for optimal control”. *Optimal Control Applications and Methods* **36.4** (2015), pp. 398–421.
- [60] Patterson, M. A. & Rao, A. “Exploiting sparsity in direct collocation pseudospectral methods for solving optimal control problems”. *Journal of Spacecraft and Rockets* **49.2** (2012), pp. 354–377.
- [61] Patterson, M. A. & Rao, A. V. “A general-purpose MATLAB software for solving multiple-phase optimal control problems version 2.3” (2016).
- [62] Peng, H., Wang, X., Zhang, S. & Chen, B. “An iterative symplectic pseudospectral method to solve nonlinear state-delayed optimal control problems”. *Communications in Nonlinear Science and Numerical Simulation* **48** (2017), pp. 95–114.
- [63] Pesch, H. J., Plail, M. & Munich, D. “The maximum principle of optimal control: a history of ingenious ideas and missed opportunities”. *Control and Cybernetics* **38.4A** (2009), pp. 973–995.
- [64] Pontryagin, L. S. *Mathematical theory of optimal processes*. Routledge, 2018.
- [65] Powell, M. J. “The BOBYQA algorithm for bound constrained optimization without derivatives”. *Technical Report DAMTP 2009/NA06* (2009), pp. 26–46.
- [66] Przemyslaw, B. & Shampine, L. “A 3 (2) pair of Runge–Kutta formulas”. *Appl Math Lett* **2.4** (1989), pp. 321–325.
- [67] Rao, A. V. “A survey of numerical methods for optimal control”. *Advances in the Astronautical Sciences* **135.1** (2009), pp. 497–528.
- [68] Schmidt, M. “An interior-point method for nonlinear optimization problems with locatable and separable nonsmoothness”. *EURO Journal on Computational Optimization* **3.4** (2015), pp. 309–348.
- [69] Sethi, S. P. “What is optimal control theory?” *Optimal Control Theory*. Springer, 2019, pp. 1–26.
- [70] Shampine, L. F. “Dissipative approximations to neutral DDEs”. *Applied Mathematics and Computation* **203.2** (2008), pp. 641–648.
- [71] Shampine, L. F. & Reichelt, M. W. “The MATLAB ODE suite”. *SIAM Journal on Scientific Computing* **18.1** (1997), pp. 1–22.

- [72] Shampine, L. F., Thompson, S & Kierzenka, J. "Solving delay differential equations with dde23". URL <http://www.runet.edu/~thompson/webddes/tutorial.pdf> (2000).
- [73] Shampine, L. "Solving ODEs and DDEs with residual control". *Applied Numerical Mathematics* **52.1** (2005), pp. 113–127.
- [74] Silva, C., Maurer, H & Torres, D. "Optimal control of a tuberculosis model with state and control delays." *Mathematical Biosciences and Engineering: MBE* **14.1** (2017), pp. 321–337.
- [75] Tian, B. & Zong, Q. "Optimal guidance for reentry vehicles based on indirect Legendre pseudospectral method". *Acta Astronautica* **68.7-8** (2011), pp. 1176–1184.
- [76] Tucker, A. W. & Kuhn, H. "Nonlinear programming". *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, Univ. of California Press. 1951, pp. 481–492.
- [77] Ulbrich, M., Ulbrich, S. & Vicente, L. N. "A globally convergent primal-dual interior-point filter method for nonlinear programming". *Mathematical Programming* **100.2** (2004), pp. 379–410.
- [78] Von Stryk, O. & Bulirsch, R. "Direct and indirect methods for trajectory optimization". *Annals of Operations Research* **37.1** (1992), pp. 357–373.
- [79] Wächter, A. & Biegler, L. T. "Line search filter methods for nonlinear programming: Motivation and global convergence". *SIAM Journal on Optimization* **16.1** (2005), pp. 1–31.
- [80] Wächter, A. & Biegler, L. T. "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming". *Mathematical Programming* **106.1** (2006), pp. 25–57.
- [81] Weinstein, M. J. & Rao, A. V. "Algorithm 984: ADiGator, a toolbox for the algorithmic differentiation of mathematical functions in MATLAB using source transformation via operator overloading". *ACM Transactions on Mathematical Software (TOMS)* **44.2** (2017), 21:1–21:25.
- [82] Wu, Z. & Michiels, W. "Reliably computing all characteristic roots of delay differential equations in a given right half plane using a spectral method". *Journal of Computational and Applied Mathematics* **236.9** (2012), pp. 2499–2514.
- [83] Zubik-Kowal, B. & Vandewalle, S. "Waveform relaxation for functional-differential equations". *SIAM Journal on Scientific Computing* **21.1** (1999), pp. 207–226.

## **APPENDIX**

## APPENDIX

# A

## RELEVANT CODE

### A.1 MATLAB Scripts

In this section we include example code for all the numerical techniques utilized in this thesis. We give examples of how to implement control parameterization in MATLAB as well as how to implement various problems in GPOPS-II. A problem in GPOPS-II requires three files to run: `main.m` for the problem setup and bounds, `continuous.m` for the dynamics, and `events.m` which contains the objective and phase constraints. We include all three GPOPS-II files for each problem. We note that in most of the codes included in this Appendix, the order of the interpolating polynomial is fixed to decrease computational time.

#### A.1.1 MOS Using GPOPS-II

We begin with the GPOPS-II code for implementing the method of steps (MOS) for the problem given in Eq. 4.19 with  $c = -1.2$  and  $\tau = 1$ .

```
%-----Begin Main File-----%  
function [answer] = mos_main(N,P,m)  
%this is the main file for using MOS in GPOPS-II  
%solves  $ax + cx(t-\tau) + bu$   
%subject to  $10(x-2)^2 + u^2$  using MOS
```

```

t0 = 0; %initial time
tf = 1; %final time
x0 = [1,2,3,4,4]; %initial state condition
xf = [2,3,4,4,4]; %final state value
c = -1.2; %delay state coefficient
a = -.5; % state coefficient
b = 1; %control coefficient
auxdata.a = a;
auxdata.b = b;
auxdata.c = c;
xmin = -200*ones(1,5); %state bounds
xmax = 200*ones(1,5);
umin = -100*ones(1,5); %control bounds
umax = 100*ones(1,5);

%-----Set up Problem Bounds -----%
bounds.phase.initialtime.lower = t0;
bounds.phase.initialtime.upper = t0;
bounds.phase.finaltime.lower = tf;
bounds.phase.finaltime.upper = tf;
bounds.phase.initialstate.lower = [1, -100*ones(1,4)];
bounds.phase.initialstate.upper = [1, 100*ones(1,4)];
bounds.phase.state.lower = xmin;
bounds.phase.state.upper = xmax;
bounds.phase.finalstate.lower = -100*ones(1,5);
bounds.phase.finalstate.upper = 100*ones(1,5);
bounds.phase.control.lower = umin;
bounds.phase.control.upper = umax;
bounds.phase.integral.lower = 0;
bounds.phase.integral.upper = 10000;
bounds.eventgroup(1).lower = [0,0,0,0]; %enforce continuity across eqns
bounds.eventgroup(1).upper = [0,0,0,0];

%-----Provide Guess of Solution-----%
guess.phase.time = [t0;tf];
guess.phase.state = [x0;xf];
guess.phase.control = [1,1,1,1,1;0,0,0,0,0];

```

```

guess.phase.integral = 0;

%---Provide Mesh Refinement Method and Initial Mesh---%
mesh.method = 'hp-PattersonRao';
mesh.tolerance = 1e-07;
mesh.maxiteration = m;
mesh.colpointsmax = P;
mesh.colpointsmin = P;
mesh.phase.colpoints = P*ones(1,N);
mesh.phase.fraction = ones(1,N)/N;

%-----Assemble Information into Problem Structure-----%
setup.name = 'mos_statedly';
setup.functions.continuous = @mos_cont;
setup.functions.endpoint = @mos_events;
setup.auxdata = auxdata;
setup.bounds = bounds;
setup.guess = guess;
setup.nlp.solver = 'ipopt';
setup.derivatives.supplier = 'sparseCD';
setup.derivatives.derivativelevel = 'second';
setup.derivatives.dependencies = 'full';
setup.nlp.ipoptoptions.tolerance = 1e-07;
setup.mesh = mesh;

%----Solve Problem Using GPOPS2----%
output = gpops2(setup);

solution = output.result.solution;
x = solution.phase.state;
u = solution.phase.control;
t = solution.phase.time;

cost = solution.phase.integral;
J_mos = sprintf('%0.7f',cost) %print cost

%plot solution on whole interval [0,5]

```

```

figure()
plot(t,x(:,1),'k-','linewidth',2)
hold on
plot(t+1,x(:,2),'k-','linewidth',2)
plot(t+2,x(:,3),'k-','linewidth',2)
plot(t+3,x(:,4),'k-','linewidth',2)
plot(t+4,x(:,5),'k-','linewidth',2)
xlabel('Time')
ylabel('State')
set(gca,'fontsize',14)

figure()
plot(t,u(:,1),'k-','linewidth',2)
hold on
plot(t+1,u(:,2),'k-','linewidth',2)
plot(t+2,u(:,3),'k-','linewidth',2)
plot(t+3,u(:,4),'k-','linewidth',2)
plot(t+4,u(:,5),'k-','linewidth',2)
xlabel('Time')
ylabel('Control')
set(gca,'fontsize',14)

[answer] = [t x u];

%-----%
%---Begin Continuous File---%
%-----%
function phaseout = mos_cont(input)
x = input.phase.state;
u = input.phase.control;
t = input.phase.time;
a = input.auxdata.a;
b = input.auxdata.b;
c = input.auxdata.c;
r = 2;

%dynamic equations

```

```

dx1 = a*x(:,1) + c + b*u(:,1);
dx2 = a*x(:,2) + c*x(:,1) + b*u(:,2);
dx3 = a*x(:,3) + c*x(:,2) + b*u(:,3);
dx4 = a*x(:,4) + c*x(:,3) + b*u(:,4);
dx5 = a*x(:,5) + c*x(:,4) + b*u(:,5);

dx = [dx1, dx2, dx3, dx4, dx5];

%cost equation
integrand = (10*(x(:,1)-r).^2 + u(:,1).^2) + ...
            (10*(x(:,2)-r).^2 + u(:,2).^2)+...
            (10*(x(:,3)-r).^2 + u(:,3).^2) + ...
            (10*(x(:,4)-r).^2 + u(:,4).^2) + ...
            (10*(x(:,5)-r).^2 + u(:,5).^2);

phaseout.dynamics = dx;
phaseout.integrand = integrand;

%-----%
%-----Begin Events File-----%
%-----%
function output = mos_events(input)
x0 = input.phase(1).initialstate;
xf = input.phase(1).finalstate;

%boundary conditions linking the MOS equations together
output.eventgroup(1).event = [x0(2)-xf(1), x0(3)-xf(2),...
                             x0(4)-xf(3), x0(5)-xf(4)];
output.objective = input.phase.integral;

```

### A.1.2 GPOPS-II and MOL

We give the code to solve the OCP with state delay in Eq. 5.16 using the method of lines.

```

%-----Begin Main File-----%
function [answer] = MOL_main(N,P,m,s)
%solve delay problem  $x' = ax + cx(t-\tau) + d \cdot u$ 
%uses Method of Lines w/ finite diff approximation

```

```

%.....Set Up Bounds For Problem.....%

t0                = 0;
tf                = 5;
x0                = [ones(1,s+1)];
xf                = [4*ones(1,s+1)];
tau               = 1;    % Delay
a                 = -0.5;
c                 = -1.2;
d                 = 1;
xmin              = [-3000*ones(1,s+1)];
xmax              = [2000*ones(1,s+1)];
auxdata.tau       = tau;
auxdata.a         = a;
auxdata.c         = c;
auxdata.d         = d;
auxdata.s         = s;
umin              = -100;
umax              = 100;
bounds.phase.initialtime.lower = t0;
bounds.phase.initialtime.upper = t0;
bounds.phase.finaltime.lower   = tf;
bounds.phase.finaltime.upper   = tf;
bounds.phase.initialstate.lower = x0;
bounds.phase.initialstate.upper = x0;
bounds.phase.state.lower       = [-30*ones(1,s+1)];
bounds.phase.state.upper       = [30*ones(1,s+1)];
bounds.phase.finalstate.lower  = [-30*ones(1,s+1)];
bounds.phase.finalstate.upper  = [30*ones(1,s+1)];
bounds.phase.control.lower     = umin;
bounds.phase.control.upper     = umax;
bounds.phase.integral.lower    = 0;
bounds.phase.integral.upper    = 10000;

%----- Provide Guess of Solution -----%
tGuess                = [t0;tf];

```

```

xGuess          = [x0;xf];
uGuess          = [1;0] ;
guess.phase.time = tGuess;
guess.phase.state = xGuess;
guess.phase.control = uGuess;
guess.phase.integral = 0;

%-----Provide Mesh Refinement Method and Initial Mesh -----%
mesh.method      = 'hp-PattersonRao';
mesh.tolerance   = 1e-8;
mesh.maxiteration = m;
mesh.colpointsmax = P;
mesh.colpointsmmin = P;
mesh.phase.colpoints = P*ones(1,N);
mesh.phase.fraction = ones(1,N)/N;

%----- Assemble Information into Problem Structure -----%
setup.name = 'MOL';
setup.functions.continuous = @MOL_continuous;
setup.functions.endpoint = @MOL_events;
setup.auxdata = auxdata;
setup.bounds = bounds;
setup.guess = guess;
setup.nlp.solver = 'ipopt';
setup.derivatives.supplier = 'sparseCD';
setup.derivatives.derivativelevel = 'second';
setup.derivatives.dependencies = 'full';
setup.mesh = mesh;

%----- Solve Problem Using GPOPS2-----%
output = gpops2(setup);
solution=output.result.solution ;
state=solution.phase.state;
u=solution.phase.control;
time=solution.phase.time;
xN = state(:,end);

```

```

cost = solution.phase.integral;
J_MOL = sprintf('%0.7f',cost)

[answer] = [time xN u];
%-----%
%---Begin Continuous File---%
%-----%
function phaseout = MOL_continuous(input)
tau = input.auxdata.tau; % delay
a = input.auxdata.a; % state coeff
c = input.auxdata.c; %delay state coeff
d = input.auxdata.d; % control coeff
s = input.auxdata.s;
x = input.phase.state;
u = input.phase.control;
t = input.phase.time;
% s = # of points in finite diff mesh
h = tau/s;
xdim = size(x);

%begin system for s mesh points
dx0 = (-3/(2*h))*x(:,1) + (2/h)*x(:,2) + (-1/(2*h))*x(:,3);
dxk = 1/(2*h)*(x(:,(3:end)) - x(:,(1:end-2)));
dxN = a*x(:,end)+ c*x(:,1) + d*u;

%transpose to obtain column vectors
dx = [dx0'; dxk'; dxN'];
dx = dx';
integrand = 10*(x(:,end) - 2).^2 + u.^2;
phaseout.dynamics = dx;
phaseout.integrand = integrand;
%-----%
%-----Begin Events File-----%
%-----%
function output = MOL_events(input)

output.objective = input.phase.integral;

```

### A.1.3 GPOPS-II on the State Delayed Necessary Conditions

We demonstrate how to use GPOPS-II as a delayed BVP solver on the necessary conditions with state delay given in Eq. 4.34.

```
%-----Begin Main File-----%
function [answer] = stateconds_main(N,P,m)
%solve  $x' = ax + cx(t-\tau) + bu$ 
% with cost integrand  $10(x-2)^2 + u^2$  using necessary conditions
t0 = 0;
tf = 5;
tau = 1;
a = -.5;
b = 1;
c = .5;
r = 2;
auxdata.tau = tau;
auxdata.a = a;
auxdata.b = b;
auxdata.c = c;
auxdata.r = r;

%----- Set up Problem Bounds -----%
bounds.phase.initialtime.lower = t0;
bounds.phase.initialtime.upper = t0;
bounds.phase.finaltime.lower = tf;
bounds.phase.finaltime.upper = tf;

%states are x, costates lambda, cost J
bounds.phase.initialstate.lower = [1,-30, 0];
bounds.phase.initialstate.upper = [1, 30, 30];
bounds.phase.state.lower = [-100, -100, 0];
bounds.phase.state.upper = [100, 100, 100];
bounds.phase.finalstate.lower = [-30, 0, 0];
bounds.phase.finalstate.upper = [100, 0, 100];

%-----Provide Guess of Solution-----%
tGuess = [t0;tf];
```

```

xGuess = [1, -7, 0; 2, 0, 50];
guess.phase.time = tGuess;
guess.phase.state = xGuess;

%----Provide Mesh Refinement Method and Initial Mesh--%
mesh.method = 'hp-PattersonRao';
mesh.tolerance = 1e-05;
mesh.maxiteration = m;
mesh.colpointsmax = P;
mesh.colpointsmin = P;
mesh.phase.colpoints = P*ones(1,N);
mesh.phase.fraction = ones(1,N)/N;

%-----Assemble Information into Problem Structure -----%
setup.name = 'State_Conds';
setup.functions.continuous = @stateconds_cont;
setup.functions.endpoint = @stateconds_events;
setup.auxdata = auxdata;
setup.bounds = bounds;
setup.guess = guess;
setup.nlp.solver = 'ipopt';
setup.derivatives.supplier = 'sparseCD';
setup.derivatives.derivativelevel = 'second';
setup.derivatives.dependencies = 'full';
setup.nlp.ipoptoptions.tolerance = 1e-6;
setup.mesh = mesh;

%-----Solve Problem Using GPOPS2 -----%
output = gpops2(setup);

solution=output.result.solution ;

x = solution.phase.state(:,1);
l = solution.phase.state(:,2);
%reconstruct control from lambda
u = -.5*l;
t = solution.phase.time;

```

```

cost = output.result.objective;
J = sprintf('%0.7f', cost)

[answer] = [t x u];

%-----%
%---Begin Continuous File---%
%-----%
function phaseout = stateconds_cont(input)
a = input.auxdata.a;
b = input.auxdata.b;
c = input.auxdata.c;
r = input.auxdata.r;
tau = input.auxdata.tau;

x = input.phase.state(:,1); %state
l = input.phase.state(:,2); %multiplier
j = input.phase.state(:,3); %cost

tt= input.phase.time;
iter = size(tt);
N = iter(1,1);

if N < 4 %initialize without delay
    dx = a*x + c*x + (-.5*l);
    dl = -a*x -c*l - 20*(x-2);
    dj = 10*(x-r).^2 + (-.5*l).^2;
else
    ttmtau = tt-tau;
    tsn = ttmtau(ttmtau<=0);
    tsp = ttmtau(ttmtau>0);
    xd1 = ones(size(tsn));
    xd2 = interp1(tt, x, tsp); %interpolate delay term
    xdel = [xd1;xd2];
    dx = a*x + c*xdel + -.5*l;

%account for characteristic function

```

```

    keep_tt = tt(tt <= 4);
    zero_tt = tt(tt > 4);

%implement advance in lambda:
    newtime = keep_tt + tau;
    lnew1 = interp1(tt, l, newtime);
    lnew2 = 0*zero_tt;
    dell = [lnew1; lnew2];

    dl = -a*l - c*dell - 20*(x-r);
    dj = 10*(x-r).^2 + (-.5*l).^2;
end

    phaseout.dynamics = [dx, dl, dj];
%-----%
%-----Begin Events File-----%
%-----%
function output = stateconds_events(input)
g = input.phase.finalstate(3); %minimize cost
output.objective = g;

```

#### A.1.4 GPOPS-II on Control Delayed Necessary Conditions

We present sample code of how to implement the necessary conditions with control delay in Eq. 6.7. As we demonstrate in Eq. 6.11 we have a delay as well as an advance in the multiplier  $\lambda$  when we solve for  $u(t-s)$  in terms of  $\lambda$ .

```

%-----Begin Main File-----%
function [answer] = ctrlneconds_main(N,P,m)

t0 = 0;
tf = 4;
x0 = 10;
r = 4;
a = -.2;
b = 2;
d = -.4;
s = 1; %control delay

```

```

auxdata.r = r;
auxdata.a = a;
auxdata.b = b;
auxdata.d = d;
auxdata.s = s;
bounds.phase.initialtime.lower = t0;
bounds.phase.initialtime.upper = t0;
bounds.phase.initialstate.lower = [x0, -300,0];
bounds.phase.initialstate.upper = [x0,300,0];
bounds.phase.state.lower = [-30,-300,-300];
bounds.phase.state.upper = [30,300,300];
bounds.phase.finaltime.lower = tf;
bounds.phase.finaltime.upper = tf;
bounds.phase.finalstate.lower = [-30, 0, 0];
bounds.phase.finalstate.upper = [30,0,100];

%-----Initial Guess-----%
guess.phase.time    = [0;0.8125;1;1.5; 3; 4];
guess.phase.state   = [10,-16.3547,0;4,.1,2; 0.38,0.1,4;...
                      4.2,0,6; 4,0,8 ; 4,0,10];

%-----Provide Mesh Refinement Method and Initial Mesh -----%

mesh.method          = 'hp-PattersonRao';
mesh.tolerance        = 1e-4 ;
mesh.maxiteration      = m;
mesh.colpointsmx      = P;
mesh.colpointsmn      = P;
mesh.phase.colpoints  = P*ones(1,N);
mesh.phase.fraction   = ones(1,N)/N;

%----- Assemble Information into Problem Structure -----%
setup.name = 'Control_BVP';
setup.functions.continuous = @ctrlneconds_cont;
setup.functions.endpoint = @ctrlneconds_events;
setup.auxdata = auxdata;
setup.bounds = bounds;

```

```

setup.guess = guess;
setup.nlp.solver = 'ipopt';
setup.nlp.ipoptoptions.tolerance = 1e-6;
setup.derivatives.supplier = 'sparseCD';
setup.derivatives.derivativelevel = 'second';
setup.derivatives.dependencies = 'full';
setup.mesh = mesh;

%----- Solve Problem Using GPOPS2 -----%
output = gpops2(setup);
solution=output.result.solution ;
state = solution.phase.state;
time = solution.phase.time;
x = state(:,1);
l = state(:,2);
cost = output.result.objective;
j_neconds = sprintf('%0.7f',cost)

%reconstruct control
%get lambda(t+s) on [0,tf -s], 0 else
keepst = time(time<= 3);
zerost = time(time>3);
newtime = keepst + s;
lnew1 = interp1(time,l,newtime);
lnew2 = 0*zerost;
dell = [lnew1;lnew2];
u = -0.5*(b*l + d*dell);

ttms = time-s;
tsn = ttms(ttms<=0); %delay times <= 0
tsp = ttms(ttms>0); %delay times > 0
ud1 = -2.3*ones(size(tsn));
ud2 = interp1(time,u, tsp);
udel = [ud1;ud2];

[answer] = [time x u udel];
%-----%

```

```

%---Begin Continuous File---%
%-----%
function phaseout = ctrlneconds_cont(input)
x = input.phase.state(:,1);
l = input.phase.state(:,2);
tt = input.phase.time;
a = input.auxdata.a;
r = input.auxdata.r;
b = input.auxdata.b;
d = input.auxdata.d;
s = input.auxdata.s;
iter = size(tt);
N = iter(1,1);

if N < 4
    dx = a*x + b*(-.5*b*l) + d*(-0.5*b*l);
    dl = -20*(x-r) -a*l;
    dj = 10*(x-r).^2 + (-.5*b*l).^2;
else
    %get lambda(t+s) on [0,tf -s], 0 else
    keep_tt = tt(tt<= 3);
    zero_tt = tt(tt>3);
    newtime = keep_tt + s;
    lnew1 = interp1(tt,l,newtime);
    lnew2 = 0*zero_tt;
    dell = [lnew1;lnew2];

    % get lambda(t) on [1,tf], 0 else
    keep1 = l(tt >1);

    %get lambda(t-s) on [0,tf]
    ttms = tt-s;
    tsn = ttms(ttms<=0); %delay times <=
    tsp = ttms(ttms>0); %delay times > 0
    lms = interp1(tt,l,tsp);
    lm_size= size(lms);

```

```

%construct u in terms of lambda
%still considered state variable by gpopsii
upos = -.5*(b*lms + d*keep1);
uneg = -2.3*ones(size(tsn));
usub = [uneg;upos];

dx = a*x + b*-.5*(b*1 + d*dell) + d*usub;
dl = -20*(x-r) - a*1;
dj = 10*(x-r).^2 + (-.5*(b*1 + d*dell)).^2; %add cost as dynamic eq
end
%no integral cost here
phaseout.dynamics = [dx, dl, dj];

%-----%
%-----Begin Events File-----%
%-----%
function output = ctrlneccconds_events(input)
g = input.phase.finalstate(3); %minimize cost variable
output.objective = g;

```

### A.1.5 GPOPS-II and Waveform Relaxation

We demonstrate how to solve Eq. 5.2 using GPOPS-II and waveform relaxation. We include the main driver file, as well as all three GPOPS-II files. The first set of files `waveform0`, `waveform0_continuous`, and `wave0Events` provide an initialization for the iteration.

```

%-----Driver File---%
%uses gpops-ii and waveform relaxation to solve  $x' = ax + c*x(t-\tau)$  on  $[0, 5]$ ,
%prehistory = 1
a = -0.5;
c = -1.2;
tau = 1;

[time0,state0] =waveform0(a,c,tau); %solves the optimal control problem
[time1,state1] =waveform1(a,c,tau,time0,state0);
[time2,state2] =waveform1(a,c,tau,time1,state1);
[time3,state3] =waveform1(a,c,tau,time2,state2);
[time4,state4] =waveform1(a,c,tau,time3,state3);

```

```

[time5,state5] =waveform1(a,c,tau,time4,state4);

%---Begin Main File-----%
%initial iteration of waveform relaxation
function [time0,state0] = waveform0(a,c,tau)

%----- Provide and Set Up All Bounds for Problem -----%

t0                                = 0;
tf                                = 5; %final time T
x0                                = 1; %IC
xf                                = 1;
xmin                              = -20;
xmax                              = 20;
auxdata.tau                       =tau; %delay
auxdata.a                         =a; %coeff on x
auxdata.c                         =c; %coeff on delay term
P                                  =4;
N                                  =10;
umin                              = -20;
umax                              =20;
bounds.phase.initialtime.lower    = t0;
bounds.phase.initialtime.upper    = t0;
bounds.phase.finaltime.lower      = tf;
bounds.phase.finaltime.upper      = tf;
bounds.phase.initialstate.lower   = 1;
bounds.phase.initialstate.upper   =1;
bounds.phase.finalstate.upper     =20;
bounds.phase.finalstate.lower     =-20;
bounds.phase.state.lower          =-20;
bounds.phase.state.upper          =20;

%----- Provide Guess of Solution -----%
tGuess                            = [t0;tf];
xGuess                            = [x0;xf];
guess.phase.time                  = tGuess;

```

```

guess.phase.state      = xGuess;

%-----Provide Mesh Refinement Method and Initial Mesh -----%
mesh.method            = 'hp-PattersonRao';
mesh.tolerance         = 1e-6;
mesh.maxiteration      = 1;
mesh.colpointsmx       = P;
mesh.colpointsmn       = P;
N                      = 10;
mesh.phase.colpoints   = P*ones(1,N);
mesh.phase.fraction    = ones(1,N)/N;

%----- Assemble Information into Problem Structure -----%
setup.name = 'wave0';
setup.functions.continuous = @waveform0_continuous;
setup.functions.endpoint = @waveform0_events;
setup.auxdata =auxdata;
setup.bounds = bounds;
setup.guess = guess;
setup.nlp.solver = 'ipopt';
setup.derivatives.supplier = 'sparseCD';
setup.derivatives.derivativelevel = 'second';
setup.derivatives.dependencies = 'full';
setup.method = 'RPM-Integration';
setup.mesh = mesh;

%----- Solve Problem Using GPOPS2 -----%
output = gpops2(setup);

solution=output.result.solution;

state0=solution.phase.state;
time0=solution.phase.time;

%-----Begin Continuous File-----%
function phaseout = waveform0_continuous(input)

```

```

tau=input.auxdata.tau; %delay
a=input.auxdata.a; %coeff on x
c=input.auxdata.c; %coeff on delay term
x= input.phase.state(:,1);
tt=input.phase.time(:,1);
iter=size(tt);
NN=iter(1,1);

%initialize without delay term
if NN < 4
dx = a*x;
else
ts=tt-tau;
tsn=find(ts<=0);
tsp=find(ts >0);
xd1=ones(size(tsn)); %prehist = 1
xd2= interp1(tt,x,ts(tsp),'linear'); %interpolate delay term
xd=[xd1;xd2];
    dx= d*xd+a*x;
end

phaseout.dynamics = dx;

%----Begin Events File---%
function output = waveform0_events(input)

output.objective = input.phase.integral;

%--Begin Main File---%
function [newtime,newstate] = waveform1(a,c,tau,oldtime,oldstate)
t0                                = 0;
tf                                = 5;
x0                                = 1;
xf                                = 1;
P=4; %polynomial order
N=10; %initial # of mesh intervals

```

```

xmin                                = -20;
xmax                                = 20;
auxdata.tau=tau;
auxdata.a=a;
auxdata.c=c;
auxdata.OLDtime=oldtime;
auxdata.OLDstate=oldstate;
umin                                = -20;
umax                                =20;
bounds.phase.initialtime.lower     = t0;
bounds.phase.initialtime.upper     = t0;
bounds.phase.finaltime.lower       = tf;
bounds.phase.finaltime.upper       = tf;
bounds.phase.initialstate.lower    = 1;
bounds.phase.initialstate.upper    =1;
bounds.phase.finalstate.upper      =20;
bounds.phase.finalstate.lower      =-20;
bounds.phase.state.lower            =-20;
bounds.phase.state.upper            =20;

%----- Provide Guess of Solution -----%
tGuess          = [t0;tf];
xGuess          = [x0;xf];
guess.phase.time = tGuess;
guess.phase.state = xGuess;

%-----Provide Mesh Refinement Method and Initial Mesh -----%
mesh.method      = 'hp-PattersonRao';
mesh.tolerance   = 1e-9;
mesh.maxiteration = 4;
mesh.colpointsmx = P;
mesh.colpointsmn = P;
N                = 10;
mesh.phase.colpoints = P*ones(1,N);
mesh.phase.fraction  = ones(1,N)/N;

```

```

%----- Assemble Information into Problem Structure -----%
setup.name                        = 'waveform';
setup.functions.continuous        = @waveform1_continuous;
setup.functions.endpoint          = @waveform_events;
setup.auxdata =auxdata;
setup.bounds                      = bounds;
setup.guess                       = guess;
setup.nlp.solver                  = 'ipopt';
setup.derivatives.supplier        = 'sparseCD';
setup.derivatives.derivativelevel = 'second';
setup.derivatives.dependencies    = 'full';
setup.method                      = 'RPM-Integration';
setup.mesh                       = mesh;

%----- Solve Problem Using GPOPS2 -----%
output = gpops2(setup);

solution=output.result.solution
cost = solution.phase.integral
J_wave = sprintf('%0.7f',cost)
newstate=solution.phase.state;
newtime=solution.phase.time;

%-----%
%---Begin Continuous File---%
%-----%
function phaseout = waveform1_continuous(input)

tau=input.auxdata.tau;
a=input.auxdata.a;
c=input.auxdata.c;
x= input.phase.state(:,1);
oldstate=input.auxdata.oldstate;
oldtime=input.auxdata.oldtime;

tt=input.phase.time(:,1);

```

```

iter=size(tt);
NN=iter(1,1);
xd=ones(size(tt));

if NN < 4
dx1 = a*x1;
else
    ts=tt-tau;
    tsn=find(ts<=0);
    tsp=find(ts >0);
    xd1=ones(size(tsn));
    xd2= interp1(oldtime,oldstate,ts(tsp),'linear');
    xd=[xd1;xd2];
    dx= c*xd+a*x;
end

integrand          = 10*(x - 2).^2 + u.^2;
phaseout.dynamics  = dx;
phaseout.integrand = integrand;

%-----%
%-----Begin Events File-----%
%-----%
function output = waveform_events(input)
g = input.phase.finalstate;
output.objective = g;

```

### A.1.6 GPOPS-II on State Delay Differential Equation

We demonstrate how to implement a differential equation with state delay in GPOPS-II. The code given below is for Eq. 5.5a. We note there is no control or cost.

```

%-----Begin Main File-----%
function soln = dde_main(N)
%solve DDE with state delay using GPOPS-II
% Set up initial data
t0 = 0;

```

```

tf = 4;
x0 = 1;
xf = 10;
tau = .3; %delay value
P = 4;
xmin = -5000;
xmax = 5000;
auxdata.tau = tau;

%.....Set Up Bounds For Problem.....%
%large bounds for x unconstrained
bounds.phase.initialtime.lower = t0;
bounds.phase.initialtime.upper = t0;
bounds.phase.finaltime.lower = tf;
bounds.phase.finaltime.upper = tf;
bounds.phase.initialstate.lower = x0;
bounds.phase.initialstate.upper = x0;
bounds.phase.state.lower = -5000 ;
bounds.phase.state.upper = 5000;
bounds.phase.finalstate.lower = -5000;
bounds.phase.finalstate.upper = 5000;

%----- Initial Guess -----%
tGuess = [t0;tf];
xGuess = [x0;xf];
guess.phase.time = tGuess;
guess.phase.state = xGuess;

%-----Provide Mesh Refinement Method and Initial Mesh -----%
mesh.method = 'hp-PattersonRao';
mesh.tolerance = 1e-9;
mesh.maxiteration = 4;
mesh.colpointsmax = 4; %fixed order polynomial
mesh.colpointsmin = 4;
mesh.phase.colpoints = P*ones(1,N);
mesh.phase.fraction = ones(1,N)/N;

```

```

%----- Assemble Information into Problem Structure ----%
setup.name = 'dde_main';
setup.functions.continuous = @dde_continuous;
setup.functions.endpoint = @dde_events;
setup.auxdata = auxdata;
setup.bounds = bounds;
setup.guess = guess;
setup.nlp.solver = 'ipopt';
setup.derivatives.supplier = 'sparseCD';
setup.derivatives.derivativelevel = 'second';
setup.derivatives.dependencies = 'full';
setup.method = 'RPM-integration';
setup.mesh = mesh;

%----- Solve Problem Using GPOPS2 -----%
output = gpops2(setup);

solution = output.result.solution;

state=solution.phase.state;
time=solution.phase.time;
[soln] = [time state]

%-----%
%---Begin Continuous File---%
%-----%
function phaseout = dde_continuous(input)
tau =input.auxdata.tau;
x = input.phase.state;
t = input.phase.time;
iter = size(t);
N = iter(1,1);
if N<4
dx = -3*x + 3.4*x;    %unstable
else
    tmtau = t-tau;
    tsn = tmtau(find(tmtau<0));

```

```

    tsp = tmtau(find(tmtau>=0));
    xd1 = ones(length(tsn),1);%prehistory is 1
    size(xd1)
    xd2 = interp1(t,x,tsp);
    delx = [xd1;xd2];
    dx = -3*x + 3.4*delx; %unstable
end

phaseout.dynamics = dx;

%-----%
%-----Begin Events File-----%
%-----%
function output = dde_events(input)
g = input.phase.finalstate;
output.objective = g;

```

### A.1.7 GPOPS-II on OCP with State Delay

We now give an example how to solve a delayed optimal control problem using GPOPS-II. We give the code for the dynamics in Eq. 5.19 with  $c = -1.2$  and the cost given in Eq. 3.33, a quadratic tracking problem.

```

function [answer] = gpopsiim_main(N,P,m)
% solves  $ax + cx(t-\tau) + bu$  on  $[0, T]$ 
% cost  $10(x-2)^2 + u^2$ 
% fixed polynomial order N

%.....Set Up Bounds For Problem.....%

t0 = 0;
tf = 5;
x0 = 1;
xf = 4;
tau = 1;      % Delay
a = -.5;
b = 1;
c = -1.2;

```

```

xmin = -3000; %large x bounds, no path constraints
xmax = 2000;
auxdata.tau = tau;
auxdata.a = a;
auxdata.b = b;
auxdata.c = c;

%leave deliberately large bounds, u unconstrained
umin = -100;
umax = 100;
bounds.phase.initialtime.lower = t0;
bounds.phase.initialtime.upper = t0;
bounds.phase.finaltime.lower = tf;
bounds.phase.finaltime.upper = tf;
bounds.phase.initialstate.lower = 1;
bounds.phase.initialstate.upper = 1;
bounds.phase.state.lower = -30;
bounds.phase.state.upper = 30;
bounds.phase.finalstate.lower = -30;
bounds.phase.finalstate.upper = 30;
bounds.phase.control.lower = umin;
bounds.phase.control.upper = umax;
bounds.phase.integral.lower = 0;
bounds.phase.integral.upper = 10000;

%-----Provide Guess of Solution-----%
tGuess = [t0;2.5;tf]; % extra points in initial guess to improve solution
xGuess = [x0;1.5;1];
uGuess = [-16;2.5;0];
guess.phase.time = tGuess;
guess.phase.state = xGuess;
guess.phase.control = uGuess;
guess.phase.integral = 0;

%----Provide Mesh Refinement Method and Initial Mesh--%
mesh.method = 'hp-PattersonRao';
mesh.tolerance = 1e-07;

```

```

mesh.maxiteration = m;
mesh.colpointsmx = P;
mesh.colpointsmn = P;
mesh.phase.colpoints = P*ones(1,N);
mesh.phase.fraction = ones(1,N)/N;

%----- Assemble Information into Problem Structure -----%

setup.name = 'GPOPS-IIIm';
setup.functions.continuous = @gpopsiim_continuous;
setup.functions.endpoint = @gen_events;
setup.auxdata = auxdata;
setup.bounds = bounds;
setup.guess = guess;
setup.nlp.solver = 'ipopt';
setup.derivatives.supplier = 'sparseCD';
setup.derivatives.derivativelevel = 'second';
setup.derivatives.dependencies = 'full';
setup.mesh = mesh;

%----- Solve Problem Using GPOPSII-----%
output = gpops2(setup);
solution = output.result.solution ;
state = solution.phase.state;
u = solution.phase.control;
time = solution.phase.time;
x = state(:,1);

%reconstruct delayed state
ttmtau = time-tau;
tsn = ttmtau(ttmtau<=0); %delay times <= 0
tsp = ttmtau(ttmtau>0); %delay times > 0
xd1 = ones(size(tsn)); %prehistory p(t) = 1
xd2 = interp1(time,x,tsp);
delx = [xd1;xd2];

cost = solution.phase.integral

```

```

J_gpopsiim = sprintf('\%0.7f', cost)

[answer] = [time x delx u ];

%-----%
%---Begin Continuous File---%
%-----%
function phaseout = gpopsiim_continuous(input)

tau = input.auxdata.tau;
a = input.auxdata.a;
b = input.auxdata.b;
c = input.auxdata.c;

x = input.phase.state(:,1);
u = input.phase.control;
tt = input.phase.time;

%identify when code initializes
iter= size(tt);
N = iter(1,1);

%ignore delay term when initializing
if N<4
dx = a*x + c*x + b*u;

else
    ttmtau = tt-tau;
    tsn = ttmtau(ttmtau<=0); %delay times <= 0
    tsp = ttmtau(ttmtau>0); %delay times > 0
    xd1 = ones(size(tsn)); %prehistory p(t) = 1
    xd2 = interp1(tt,x,tsp); %interpolate state at delay times
    delx = [xd1;xd2]; % assemble delayed state
    dx = a*x + c*delx + b*u;
end

integrand = 10*(x-2).^2 + u.^2;

```

```

phaseout.dynamics = dx;
phaseout.integrand = integrand;
%-----%
%-----Begin Events File-----%
%-----%
function output = gen_events(input)
output.objective = input.phase.integral;

```

### A.1.8 GPOPS-II on OCP with Control Delay

Below, we give the code to implement Eq. 6.3 using GPOPS-II.

```

%-----Begin Main File-----%
function [answer] = delayedctrl_main(N,P,m)
% solves  $ax + bu + du(t-s)$  on  $[0, T]$ 
% w/ cost  $10(x-r)^2 + u^2$ 
% ORDER OF POLY FIXED
%.....Set Up Bounds For Problem.....%

t0 = 0;
tf = 4;
x0 = 10;
xf = 4;
s = 1;      % Delay
a = -.2;
b = 2;
d = -.4;
r = 4;
xmin = -3000;
xmax = 2000;
auxdata.s = s;
auxdata.a = a;
auxdata.b = b;
auxdata.d = d;
auxdata.r = r;
umin = -100;
umax = 100;

```

```

bounds.phase.initialtime.lower = t0;
bounds.phase.initialtime.upper = t0;
bounds.phase.finaltime.lower = tf;
bounds.phase.finaltime.upper = tf;
bounds.phase.initialstate.lower = x0;
bounds.phase.initialstate.upper = x0;
bounds.phase.state.lower = -30;
bounds.phase.state.upper = 30;
bounds.phase.finalstate.lower = -30;
bounds.phase.finalstate.upper = 30;
bounds.phase.control.lower = umin;
bounds.phase.control.upper = umax;
bounds.phase.integral.lower = 0;
bounds.phase.integral.upper = 10000;

%-----Provide Guess of Solution-----%
xGuess = [x0;4; 4; 4, xf];
uGuess = [7; -0.3796; -0.8118; 0; 0] ;
guess.phase.time = [t0;0.8125; 1; 1.5; tf];
guess.phase.state = xGuess;
guess.phase.control = uGuess;
guess.phase.integral = 0;

%----Provide Mesh Refinement Method and Initial Mesh--%
mesh.method = 'hp-PattersonRao';
mesh.tolerance = 1e-04;
mesh.maxiteration = m;
mesh.colpointsmax = P;
mesh.colpointsmin = P;
mesh.phase.colpoints = P*ones(1,N);
mesh.phase.fraction = ones(1,N)/N;

%----- Assemble Information into Problem Structure-----%
setup.name = 'Control_Delay';
setup.functions.continuous = @delayedctrl_cont;
setup.functions.endpoint = @delayedctrl_events;
setup.auxdata = auxdata;

```

```

setup.bounds = bounds;
setup.guess = guess;
setup.nlp.solver = 'ipopt';
setup.derivatives.supplier = 'sparseCD';
setup.derivatives.derivativelevel = 'second';
setup.derivatives.dependencies = 'full';
setup.nlp.ipoptoptions.tolerance = 1e-4;
setup.nlp.ipoptoptions.maxiterations = 1000;
setup.mesh = mesh;

%-----Solve Problem Using GPOPS2 -----%
output = gpops2(setup);
solution=output.result.solution ;
x =solution.phase.state(:,1);
u=solution.phase.control(:,1);
t=solution.phase.time;
cost = solution.phase.integral;
J_gposiim = sprintf('%0.7f',cost)

[answer] = [t x u];

%-----%
%---Begin Continuous File---%
%-----%
function phaseout = delayedctrl_cont(input)
s = input.auxdata.s;
a  = input.auxdata.a;
b  = input.auxdata.b;
d  = input.auxdata.d;
r  = input.auxdata.r;
x  = input.phase.state(:,1);
u   = input.phase.control(:,1);
tt  = input.phase.time;
iter= size(tt);
N   = iter(1,1);

if N<4

```

```

    dx = a*x + b*u + d*u;
else
    ttms = tt-s;
    tsn = ttms(ttms<=0); %delay times <= 0
    tsp = ttms(ttms>0); %delay times > 0
    ud1 = -2.3*ones(size(tsn)); %prehistory
    ud2 = interp1(tt,u,tsp);
    delu = [ud1;ud2];
    dx = a*x + b*u + d*delu;
end
    integrand          = 10*(x-2).^2 + u.^2;
    phaseout.dynamics = dx;
phaseout.integrand = integrand;

%-----%
%-----Begin Events File-----%
%-----%
function output = delayedctrl_events(input)

output.objective = input.phase.integral;

```

### A.1.9 GPOPS-II on OCP with Control Delay, Extra Variable

We give the code to implement Eq. 6.3 with extra dynamic equation  $u' = z$ , adding  $\epsilon \|z\|^2$  to the cost. In order to obtain  $\int_0^T \epsilon \|z\|^2 dt$ , we create a new state variable  $zval$  and add the dynamic equation  $dzval = \epsilon \|z\|^2$ . We then subtract  $\int_0^T \epsilon \|z\|^2 dt$  from our computed cost in order to obtain the cost for Eq. 6.3 without  $z$  increasing the results.

```

function [answer] = delayedctrl_main(N,P,m)
% solves ax + bu + du(t-s) on [0, T]
% w/ cost 10(x-r)^2 + u^2
% ORDER OF POLY FIXED

%.....Set Up Bounds For Problem.....%
t0 = 0;
tf = 4;
x0 = 10;
z0 = -7;

```

```

zf = 0;
xf = 4;
tau = 1;          % Delay
a = -.2;
b = 2;
d = -.4;
r = 4;
xmin = -3000;
xmax = 2000;
auxdata.s = s;
auxdata.a = a;
auxdata.b = b;
auxdata.d = d;
auxdata.r = r;
zmin = -100;
zmax = 100;
bounds.phase.initialtime.lower = t0;
bounds.phase.initialtime.upper = t0;
bounds.phase.finaltime.lower = tf;
bounds.phase.finaltime.upper = tf;
bounds.phase.initialstate.lower = [x0,-200,0];
bounds.phase.initialstate.upper = [x0,200,0];
bounds.phase.state.lower = [-200,-200,-200];
bounds.phase.state.upper = [200,200,200];
bounds.phase.finalstate.lower = [-200,-200,-200];
bounds.phase.finalstate.upper = [200,200,200];
bounds.phase.control.lower = zmin;
bounds.phase.control.upper = zmax;
bounds.phase.integral.lower = 0;
bounds.phase.integral.upper = 10000;

%-----Provide Guess of Solution -----%
xGuess          = [x0,z0,0;4,0,.1; 4,0,.1; 4,0,.1; xf, zf, 4];
uGuess          = [7; -0.3796; -0.8118; 0; 0] ;
guess.phase.time = [t0;0.8125; 1; 1.5; tf];
guess.phase.state = xGuess;
guess.phase.control = uGuess;

```

```

guess.phase.integral = 0;

%---Provide Mesh Refinement Method and Initial Mesh---%
mesh.method = 'hp-PattersonRao';
mesh.tolerance = 1e-04;
mesh.maxiteration = m;
mesh.colpointsmax = P;
mesh.colpointsmin = P;
mesh.phase.colpoints = P*ones(1,N);
mesh.phase.fraction = ones(1,N)/N;

%----- Assemble Information into Problem Structure -----%
setup.name = 'Control_Delay';
setup.functions.continuous = @delayedctrl_cont;
setup.functions.endpoint = @delayedctrl_events;
setup.auxdata = auxdata;
setup.bounds = bounds;
setup.guess = guess;
setup.nlp.solver = 'ipopt';
setup.derivatives.supplier = 'sparseCD';
setup.derivatives.derivativelevel = 'second';
setup.derivatives.dependencies = 'full';
setup.nlp.ipoptoptions.tolerance = 1e-4;
setup.nlp.ipoptoptions.maxiterations = 1000;
setup.mesh = mesh;

%-----Solve Problem Using GPOPS2 -----%
output = gpops2(setup);
solution=output.result.solution ;
x =solution.phase.state(:,1);
u=solution.phase.state(:,2);
z = solution.phase.control;
t=solution.phase.time;
cost = solution.phase.integral;
J_gpopsiim = sprintf('%0.7f',cost)

%integral of \epsilon*||z||^2

```

```

zval = solution.phase.state(:,3);
endz = zval(end); %value of integral at T
z_cost = sprintf('%0.7f',endz)

%cost minus value of integral of \epsilon*||z||^2
diff = cost- endz;
diff1 = sprintf('%0.7f',diff)

[answer] = [t x u zval];

%-----%
%---Begin Continuous File---%
%-----%
function phaseout = delayedctrl_cont(input)
s = input.auxdata.s;
a  = input.auxdata.a;
b  = input.auxdata.b;
d  = input.auxdata.d;
r  = input.auxdata.r;
x  = input.phase.state(:,1);
z  = input.phase.control(:,1);
u  = input.phase.state(:,2);

%create new state variable
%gives value of integral of epsilon*z^2 on [0,T]
zval = input.phase.state(:,3);

epsilon= .01;
tt  = input.phase.time;
iter= size(tt);
N   = iter(1,1);

if N<4 %initialize without delay
    dx = a*x + b*u + d*u;
    du = z;
    dzval = epsilon*z.^2;
else

```

```

ttms = tt-s;
tsn = ttms(ttms<=0); %delay times <= 0
tsp = ttms(ttms>0); %delay times > 0
ud1 = -2.3*ones(size(tsn)); %prehistory
ud2 = interp1(tt,u,tsp);
delu = [ud1;ud2];
dx = a*x + b*u + d*delu;
du = z;

%variable epsilon
% .001 z^2 on [0,1]
% .01 z^2 on [1,4]
t1 = find(tt==1);
dzval = epsilon*z.^2;
dzval(1:t1) = .1*dzval(1:t1);
end

%variable epsilon
% .001 z^2 on [0,1]
% .01 z^2 on [1,4]
newz = epsilon*z.^2;
newz(1:t1) = .1*newz(1:t1);

integrand = 10*(x - r).^2 + u.^2 + newz;
phaseout.dynamics = [dx, du, dzval];
phaseout.integrand = integrand;

%-----%
%-----Begin Events File-----%
%-----%
function output = delayedctrl_events(input)

output.objective = input.phase.integral;

```

#### A.1.10 GPOPS-IIow State Delay

We given the code to solve Eq. 5.24 with cost Eq. 5.25 using GPOPS-IIow, waveform optimization.

```

%-----Driver File-----%
%use waveform optimization on  $x' = a x + c x(t-\tau) + u$  on [0 5]
a=-0.5;
c=-1.2; % or 1.2
tau=1; %delay or 1, 0.5, 0.25, 0.05
[time0,state0,control0] =wave0f(a,c,0); %solves the optimal control problem
[time1,state1,control1] =wavef(a,c,tau,time0,state0);
[time2,state2,control2] =wavef(a,c,tau,time1,state1);
[time3,state3,control3] =wavef(a,c,tau,time2,state2);
[time4,state4,control4] =wavef(a,c,tau,time3,state3);
[time5,state5,control5] =wavef(a,c,tau,time4,state4);

%-----wave0 Main File-----%
function [time0,state0,control0] = wave0f(a,c,tau)

%-----Provide and Set Up All Bounds for Problem -----%

t0 = 0;
tf = 5;
x0 = 1;
xf = 1;
xmin = -20;
xmax = 20;
%tau =1;
auxdata.tau=tau;
auxdata.a=a;
auxdata.c=c;
P=4;
N=10;
umin = -20;
umax =20;
bounds.phase.initialtime.lower = t0;
bounds.phase.initialtime.upper = t0;
bounds.phase.finaltime.lower = tf;
bounds.phase.finaltime.upper = tf;
bounds.phase.initialstate.lower = 1;
bounds.phase.initialstate.upper =1;

```

```

bounds.phase.finalstate.upper =20;
bounds.phase.finalstate.lower =-20;
bounds.phase.state.lower      =-20;
bounds.phase.state.upper      =20;
bounds.phase.control.lower    = umin;
bounds.phase.control.upper    =umax;
bounds.phase.integral.lower   = 0;
bounds.phase.integral.upper   = 100000;

%-----Provide Guess of Solution -----%
tGuess      = [t0;tf];
xGuess      = [x0;xf];
uGuess      = [1;1];
guess.phase.time      = tGuess;
guess.phase.state     = xGuess;
guess.phase.control   = uGuess;
guess.phase.integral  = 0;

%----Provide Mesh Refinement Method and Initial Mesh-----%
mesh.method      = 'hp-PattersonRao';
mesh.tolerance   = 1e-6;
mesh.maxiteration = 1;
mesh.colpointsmax = P;
mesh.colpointsmin = P;
N                = 10;
mesh.phase.colpoints = P*ones(1,N);
mesh.phase.fraction = ones(1,N)/N;

%----- Assemble Information into Problem Structure -----%
setup.name      = 'wave0';
setup.functions.continuous = @wave0Continuous;
setup.functions.endpoint   = @wave0Events;
setup.auxdata =auxdata;
setup.bounds      = bounds;
setup.guess       = guess;
setup.nlp.solver  = 'ipopt';
setup.derivatives.supplier = 'sparseCD';

```

```

setup.derivatives.derivativelevel = 'second';
setup.derivatives.dependencies    = 'full';
setup.method                      = 'RPM-Integration';
setup.mesh                        = mesh;

%----- Solve Problem Using GPOPS2 -----%
output = gpops2(setup);

solution=output.result.solution

state0=solution.phase.state;
control0=solution.phase.control;
time0=solution.phase.time;

%-----%
%---Begin Continuous File---%
%-----%
function phaseout = wave0Continuous(input)

tau=input.auxdata.tau;
a=input.auxdata.a;
c=input.auxdata.c;
x= input.phase.state(:,1);

u=input.phase.control(:,1);
tt=input.phase.time(:,1);
iter=size(tt);
NN=iter(1,1);

if NN < 4
dx = a*x +u; %initialize without delay
else
    ts=tt-tau;
    tsn=find(ts<=0);
    tsp=find(ts >0);
    xd1=ones(size(tsn));
    xd2= interp1(tt,x,ts(tsp),'linear'); %interpolate delay term

```

```

xd=[xd1;xd2];
    dx1= c*xd+a*x+ u;
end

integrand          =    10*(x-2).*(x-2) +  u.*u;

phaseout.dynamics  = dx;
phaseout.integrand = integrand;

%-----%
%-----Begin Events File-----%
%-----%
function output = wave0(input)

output.objective = input.phase.integral;

%---Begin Main 'wavef' File---%
function [newtime,newstate,newcontrol] = wavef(a,c,tau,oldtime,oldstate)

%-----%
t0          = 0;
tf          = 5;
x0          = 1;
xf          = 1;
xmin        = -20;
xmax        = 20;
auxdata.tau=tau;
auxdata.a=a;
auxdata.c=c;
auxdata.oldtime=oldtime;
auxdata.oldstate=oldstate;
P=4;
N=10;

umin        = -20;
umax        =20;
bounds.phase.initialtime.lower = t0;

```

```

bounds.phase.initialtime.upper = t0;
bounds.phase.finaltime.lower   = tf;
bounds.phase.finaltime.upper   = tf;
bounds.phase.initialstate.lower = 1;
bounds.phase.initialstate.upper = 1;
bounds.phase.finalstate.upper = 20;
bounds.phase.finalstate.lower = -20;
bounds.phase.state.lower       = -20;
bounds.phase.state.upper       = 20;
bounds.phase.control.lower     = umin;
bounds.phase.control.upper     = umax;
bounds.phase.integral.lower    = 0;
bounds.phase.integral.upper    = 100000;

```

```

%-----%
%----- Provide Guess of Solution -----%
%-----%

```

```

tGuess          = [t0;tf];
xGuess          = [x0;xf];
uGuess          = [1;1];
guess.phase.time = tGuess;
guess.phase.state = xGuess;
guess.phase.control = uGuess;
guess.phase.integral = 0;

```

```

%-----%
%----- Provide Mesh Refinement Method and Initial Mesh -----%
%-----%

```

```

mesh.method      = 'hp-PattersonRao';
mesh.tolerance   = 1e-9;
mesh.maxiteration = 4;
mesh.colpointsmax = P;
mesh.colpointsmin = P;
N                = 10;
mesh.phase.colpoints = P*ones(1,N);
mesh.phase.fraction = ones(1,N)/N;

```

```

%-----%
%----- Assemble Information into Problem Structure -----%
%-----%

setup.name                = 'wavef';
setup.functions.continuous = @wavefContinuous;
setup.functions.endpoint   = @wavefEvents;
setup.auxdata =auxdata;
setup.bounds               = bounds;
setup.guess                = guess;
setup.nlp.solver           = 'ipopt';
setup.derivatives.supplier = 'sparseCD';
setup.derivatives.derivativelevel = 'second';
setup.derivatives.dependencies = 'full';
setup.method               = 'RPM-Integration';
setup.mesh                 = mesh;

%-----%
%----- Solve Problem Using GPOPS2 -----%
%-----%

output = gpops2(setup);

solution=output.result.solution
cost = solution.phase.integral
J_gpopsiiow = sprintf('%0.7f',cost)
newstate=solution.phase.state;
newcontrol=solution.phase.control;
newtime=solution.phase.time;

%--Begin Continuous File---%
function phaseout = wavefContinuous(input)

tau=input.auxdata.tau;
a=input.auxdata.a;
c=input.auxdata.c;
x= input.phase.state(:,1);
oldstate=input.auxdata.oldstate;

```

```

oldtime=input.auxdata.oldtime;

u=input.phase.control(:,1);
tt=input.phase.time(:,1);
iter=size(tt);
NN=iter(1,1);
xd=ones(size(tt));
if NN < 4
dx1 = a*x +u;
else
    ts=tt-tau;
    tsn=find(tts<=0);
    tsp=find(ts >0);
    xd1=ones(size(tsn));
    xd2= interp1(oldtime,oldstate,ts(tsp),'linear');
    xd=[xd1;xd2];
    dx1= c*xd+a*x+ u;
end

integrand          =    10*(x-2).*(x-2) +  u.*u;

phaseout.dynamics  = dx;
phaseout.integrand = integrand;

%--Begin Events File---%
function output = wavefEvents(input)

output.objective = input.phase.integral;

```

### A.1.11 Control Parameterization and OCP with State Delay

We demonstrate how to implement an OCP with state delay with linear dynamics and quadratic cost in MATLAB. We use the delay differential equation solver `dde23` and the nonlinear optimizer `fmincon`, and solve the problem given by Eq. 5.21 and Eq. 5.8.

```

%-----Begin Driver File-----%
% Driver for Control parameterization code for delayed OCP
%solves  $dx = ax + cx(t-\tau) + bu$  on  $[0,T]$ 

```

```

% with quadratic cost  $dJ = 10(x-2)^2 + u^2$ 
t0 = 0;
tf = 5;
N = 28;
tau = 1; %value of delay
x0 = [1 0]; %delay IC
step = tf/N;
time1 = t0:step:tf;

[J1,umin] = ctrlparam_mincost2();

ctrlparam_cost = sprintf('%0.7f',J1)

options = ddeset('RelTol',1e-7,'AbsTol',1e-7);
sol = dde23(@ddex1,[tau],@ddex1hist,[t0,tf],options,umin);
t = answer(:,1);
umin = spline(time1,umin,t);

function s = ddex1hist(t,u)
s=[1;0];
end

function dydt = ddex1(t,y,Z,u)
N = 28;
t0 = 0;
tf = 5;
step = tf/N;
time = t0:step:tf;
ylag = Z(:,1);
dydt(1,:) = -0.5*y(1) -1.2*ylag(1) + spline(time,u,t);
dydt(2,:) = 10*(y(1)-2).^2 + spline(time,u,t).*spline(time,u,t);
end

%-----Additional File-----%
function [J1,umin] = ctrlparam_mincost2()
N = 28; % # mesh intervals for control
IC = 4*(1:(N+1)).^(-1); %initial condition  $x(0) = 0$ 

```

```

ub = 7*ones(N+1,1); %upper bound for control
lb = -2*ones(N+1,1); %lower bound for control

options=optimoptions('fmincon','MaxIter',9000, 'TolCon',1e-7,...
                    'TolFun',1e-7);
[umin,J1,exitflag] = fmincon(@cost,IC,[],[],[],[],lb,ub,[],options);
end

%delay function
%solve with dde23
function [J,t] = cost(u)
t0 = 0;
tf = 5;
x0 = [1 0];
options = ddeset('RelTol',1e-7,'AbsTol',1e-7);
sol = dde23(@ddex1, [1],@ddex1hist,[t0,tf],options,u);
time= sol.x;
vals= sol.y;
J = vals(2,end);
end

function s = ddex1hist(t,u)
s=[1;0];
end
%
function dydt = ddex1(t,y,Z,u)
N = 28;
t0 = 0;
tf = 5;
step = tf/N;
time = t0:step:tf;
ylag = Z(:,1);
dydt(1,:) = -0.5*y(1) - 1.2*ylag(1) + spline(time,u,t);
dydt(2,:) = 10*(y(1)-2).^2 + spline(time,u,t).*spline(time,u,t);
end

```

## INDEX

- Activate, 53, 89, 122
- BobyqaOpt, 89, 122
- BVP, 18, 26, 32, 48, 50, 57–60, 72, 110
- bvp4c, 24, 26
- Collocation, 55
  - Diagram, 57
  - Direct, 37
  - Indirect, 10, 33
- Control Parameterization, 42, 50, 84, 89
  - Activate
    - Numerical Results, 91
  - MATLAB Code, 177
  - Numerical Results, 78
- dde23, 24, 29, 50, 65–67, 69, 78, 88, 177
- DDEs, 2, 66
  - Characteristic Equation, 21
  - Continuous Dependence, 15
  - Existence Results, 13, 16
  - Stability, 21, 22
- Direct Shooting Method, 36
- Direct Transcription, 10, 34, 51
- fmincon, 42, 50, 78, 177
- GPOPS-II, 38, 63
  - Mesh Refinement, 40
- GPOPS-II<sub>m</sub>, 54, 60
  - Control Delay, 107
    - Numerical Results, 114
  - Smoothing the Control, 116, 122
  - Numerical Results, 67, 78
    - Nonlinear, 83
  - State Delay
    - MATLAB Code, 155
- GPOPS-II<sub>ow</sub>, 54, 58, 61
  - MATLAB Code, 170
- Numerical Results, 83
- Hamiltonian, 5, 31, 46, 58, 59, 73
- interp1, 80
- Interpolation
  - Lagrange, 39, 55
  - Linear, 55
  - Splines, 42, 78
- IPOPT, 35, 57
  - Convergence Analysis, 93
    - Numerical, 96
- LGR, 38, 55, 56
- LSODA, 89
- Mesh Refinement, 39, 122
- Method of Lines, 19
  - MATLAB Code, 137
  - Numerical Results, 74
- Method of Steps, 17, 26, 49, 96
  - MATLAB Code, 132
  - Numerical Results, 64, 108
- Necessary Conditions, 30
  - Control Delay, 109
    - MATLAB Code, 144
    - Numerical Results, 111
  - Delayed Optimal Control Problem, 7
  - GPOPS-II<sub>m</sub>, 48, 57
  - GPOPS-II<sub>ow</sub>, 59
  - State Delay, 46, 60, 61
    - MATLAB Code, 140
    - Numerical Results, 72, 96, 104
- Nonlinear Dynamics, 73
- ode4, 113, 116
- ode45, 29, 42, 65
- Pseudospectral Method, 38, 55

roots, 22

SNOPT, 41

SOS, 52

SOSD, 51, 52, 64, 72, 76, 78, 115

spline, 80

Waveform Optimization, 58, 59, 61

Waveform Relaxation, 28, 83

    MATLAB Code, 148

    Numerical Results, 64