ABSTRACT

TOTH, ALEXANDER RAYMOND. A Theoretical Analysis of Anderson Acceleration and Its Application in Multiphysics Simulation for Light-Water Reactors. (Under the direction of Carl Kelley.)

In this work, we are concerned with both contributing to the theoretical foundation for Anderson acceleration, a method for accelerating the convergence rate of Picard iteration, and evaluating its performance in the context of coupled multiphysics problems in nuclear reactor simulation. Anderson acceleration proceeds by maintaining a depth of previous iterate information in order to compute a new iterate as a linear combination of previous evaluations of the fixed-point map, where the linear combination coefficients are obtained by solving a linear least-squares problem. Prior to this work, theory for this method was fairly sparse, dealing mainly with showing its relation to quasi-Newton multisecant updating and, when applied to linear problems, GMRES iteration. The analysis presented in this work significantly expands upon the theory for this method. As this method is intended as an acceleration method for Picard iteration, our analysis concerns problems for which Picard iteration is convergent, namely when the fixed-point mapping is contractive. We present analysis which represent the first convergence results for limited-memory variations of Anderson acceleration and for nonlinear problems. Additionally, we present analysis for several variations on the standard Anderson acceleration method. In particular, we consider a variation which adjusts the memory utilization in order to maintain good conditioning of the least-squares problem, and we present local improvement results for the case in which the fixed-point map can only be evaluated approximately.

With respect to coupled multiphysics problems, we examine Anderson acceleration as an alternative to Picard iteration in the context of black-box code coupling in nuclear reactor simulation. Picard iteration comes with several drawbacks in this context, namely relatively slow convergence and poor robustness. To test the potential for Anderson acceleration to improve upon the weaknesses of Picard iteration, we first consider a one-dimensional model problem which recreates several phenomena observed in higher fidelity couplings. We then consider the Tiamat code coupling being developed as part of the Consortium for Advanced Simulation of LWRs (CASL). Tiamat couples the Bison fuel performance code with the MPACT neutronics and COBRA-TF thermal hydraulics codes to provide a tool for pellet-cladding interaction analysis. Prior to this work, this code utilized exclusively Picard iteration to couple these single-physics codes. We overview Tiamat and describe how Anderson acceleration has been integrated into this coupling by posing the coupled system as a fixed-point problem in terms of coupling parameters. We then examine the performance gains obtained from utilizing Anderson acceleration for this coupling by considering parameter studies at various problem sizes.

A Theoretical Analysis of Anderson Acceleration and Its Application in Multiphysics Simulation for Light-Water Reactors

by
Alexander Raymond Toth

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

2016

APPROVED BY:

Pierre Gremaud	Dmitriy Anistratov
Roger Pawlowski	Carl Kelley Chair of Advisory Committee

DEDICATION

To my parents for their constant support and encouragement, and to my dog Max, who has been by my side through the ups and down of this whole process.

BIOGRAPHY

Alexander Raymond Toth was born on October 14, 1988 to parents Joseph and Eileen in Elmhurst, IL in the west suburbs of Chicago. Raised along side older brother Mike in Villa Park, IL, he graduated from Willowbrook High School in 2007. For his undergraduate education, he attended the University of Notre Dame. During the summer of 2010, he participated in a Research Experience for Undergraduates (REU) program at North Carolina State University, under the direction of Dr. Ralph Smith. Upon graduating from Notre Dame in 2011 with a B.A. in mathematics, and minoring in Russian language and literature, he returned to Raleigh to continue his mathematics education by enrolling in the Ph.D. program in applied mathematics at North Carolina State University, studying under the direction of Dr. Tim Kelley. In the course of his graduate education, he regularly collaborated with researchers at Oak Ridge National Laboratory, and spent a significant portion of time stationed at the lab.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Tim Kelley, for providing me with this great opportunity and for all his help and guidance along the way. Next, I would like to thank my committee for helping me through this process. Finally, I would also like to thank all the math teachers I've had over the years for fostering my interest in mathematics at an early age, encouraging me to participate on the math team in high school, and challenging me and pushing my boundaries in this subject.

Next, I would like to thank all the folks with whom I regularly dealt and collaborated at Oak Ridge National Laboratory during my time spent there. In particular, thank you to Roger Pawlowski, Stuart Slattery, and Steven Hamilton for their research guidance and assistance in programming and computing matters, and thanks to Linda Weltman for her administrative support. And finally, thank you to all my office mates in the CASL intern office for making my working environment such an enjoyable one to be at, and for answering nuclear questions for the non-nuclear engineer in the room.

Lastly, I would like to make acknowledgement for my funding sources and use of computing resources. This work has been supported by the Consortium for Advanced Simulation of Light Water Reactors (www.casl.gov), an Energy Innovation Hub (http://www.energy.gov/hubs) for Modeling and Simulation of Nuclear Reactors under U.S. Department of Energy Contract No. DE-AC05-00OR22725, and by National Science Foundation Grants DMS-1406349, and SI2-SSE-1339844. This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction to Coupled Multiphysics Problems

1.1 Introduction

Many important problems of interest involve complex systems which feature several interdependent physical processes which interact in a variety of ways. Hence, accurately simulating the entire system involves not only solving these individual physical processes, but accounting for the manner in which they interact. An example on which we focus in this work the coupled physics within a reactor core. Some of the relevant processes involved in this system include heat generation from fission, coolant flow through the core, heat exchange between fuel and coolant, and changes in the chemical composition of the fuel. Each of these physical processes affects the others in some way. For instance, heat generation from fission affects properties of the fuel, and these fuel properties in turn determine that rate at which fission occurs. The interacting physical processes in a coupled system may cover a wide range of areas of expertise, and as a result, it is desirable to couple together software which has been specifically developed for accurately solving some set of physical processes in order to simulate the entire coupled system. In many cases, such software for solving single sets of physics has been developed over the course of many years and can greatly vary in functionality, with respect to what can be computed and accessed by the user. It is then of interest to develop flexible and robust methods in order to efficiently utilize existing software in order to simulate coupled multiphysics systems.

In this chapter, we precisely describe the problem at hand by first introducing some important terminology and notation, and describing a general formulation of multiphysics coupling as presented in [44]. We then overview some of the solution methods that have traditionally been utilized in solving coupled multiphysics problems, including the advantages and drawbacks of these methods.

1.1.1 Basic Definitions

In this section, we introduce several important definitions for describing multiphysics coupling:

- Coupled physics - When the solution of one set of physics depends on the solution of another set of physics, theses physics are said to be coupled.
- Degree of physics coupling - The level of influence one set of physics has on another set. Two sets of physics are said to be strongly coupled if a change in the solution of one set results in a large change in the solution of the other. Otherwise, if a change in the solution of one set of physics results in a negligible change in the solution of the other, the sets of physics are said to be weakly coupled.
- Directional coupling - The manner in which two sets of physics depend on each other. Two sets of physics are said to be two-way coupled of the solution of each set is dependent on the solution of the other. Conversely, the sets of physics are said to feature one-way or forward coupling if the solution of only one set is dependent on the other. In this case, one set of physics may be solved independently of the other, possibly simplifying the solution of the coupled system.
- State variables - The set of variables that a single-physics application is solving for.
- Residual (constraint) equation - The system of equations that a physics code solves to compute the solution state variables. These are often generated from conservation or balance laws.
- Response function - A quantity of interest which a code is used to compute. This may correspond to the state variables themselves, but more generally may include postprocessed values computed from state variables and other independent parameters.
- Transfer function - A function which maps data from one application code to an input for another code. This may simply perform an interpolation of state variables or a response function between meshes, or possibly a mapping from a volume source to a point source, or vice versa. For problems featuring meshes distributed in parallel these functions will involve parallel communication as well, so these functions may include a fair bit of complexity.

1.1.2 General Formulation

We now describe coupled multiphysics with mathematical rigor. We first consider a single-physics application. The residual equation corresponding to this application is given by

$$f(\dot{x}, x, \{p_l\}, t) = 0, \tag{1.1}$$

where

- $x \in \mathbb{R}^{N_x}$ is the vector of state variables,
- $\dot{x} \in \mathbb{R}^{N_x}$ is the time derivative of the state variables,
- $\{p_l\} = \{p_0, \ldots, p_{N_p-1}\}$ is the set of independent parameter sub-vectors,
- t is the time variable.

It is of interest to determine the state variables which solve this residual equation over some time interval of interest. In this work, we will be more concerned with solving steady state problems, for which (1.1) simplifies to

$$f(x, \{p_l\}) = 0. \tag{1.2}$$

This form may still be used to describe transient simulation, in which case this formulation represents the set of equations being solved for a given time step.

It is fairly straightforward to extend this formulation to coupled multiphysics problems. We now consider a set of N_f single-physics applications. Application i has its own residual equation which it solves, as described by (1.2). Some of the set of parameter sub-vectors for this application may depend on state variables or other data from the other applications, so we partition the parameter sub-vectors into $\{z_{i,j}\}$, the sub-vectors which depend on the solution to other applications, and $\{p_{i,k}\}$, the remaining sub-vectors whose values are independent of the other applications. We then write the residual equation corresponding to application i as

$$f_i(x_i, \{z_{i,j}\}, \{p_{i,k}\}) = 0, \quad \text{for} \quad i = 0, \ldots, N_f - 1. \tag{1.3}$$

As the coupling parameters depend on some collection of state variables and independent parameters from other applications, we can express them as follows

$$z_{i,j} = r_{i,j}(\{x_m\}, \{p_{m,n}\}). \tag{1.4}$$

- $\{x_m\} = \{x_0, \ldots, x_{N_f-1}\}$ is the set of state variables for all applications,

- $\{p_{m,n}\} = \{p_{0,0}, \ldots, p_{0,N_{p_0}-1}, \ldots, p_{i,0}, \ldots, p_{i,N_{p_i}-1}, \ldots\}$ is the set of all independent parameter sub-vectors for all applications,
- $r_{i,j}$ is a transfer function which maps state variable and parameter data from the other applications to compute a dependent coupling parameter vector. This notation may bury significant complexity, as these transfer functions may involve computation of response functions, parallel communication, volume averaging, interpolation, etc. In practice, the dependencies of the transfer functions should be fairly sparse, usually mapping data from only a single application to a parameter vector for another.

Making the substitution (1.4), the above residual equation becomes

$$f_i(x_i, \{r_{i,j}(\{x_m\}, \{p_{m,n}\})\}, \{p_{i,k}\}) = 0, \quad \text{for} \quad i = 0, \ldots, N_f - 1. \tag{1.5}$$

Associated with this multiphysics system are the response functions

$$g_i(\{x_m\}, \{r_{j,k}(\{x_m\}, \{p_{m,n}\})\}, \{p_{m,n}\}), \quad \text{for} \quad i = 0, \ldots, N_g - 1. \tag{1.6}$$

The problem at hand is then to determine a collection of state variables $\{x_m\}$ such that each residual equation (1.5) is simultaneously solved.

1.2 Standard Solution Methods

In this section, we overview standard solution methods which have frequently been used in solving coupled multiphysics problems. We note that the available solution methods may be limited by the capability of the physics codes being used. A code may have the capability to expose residual evaluations to the user, or it may only be able to solve for the state variables and return the solution or some post-processed response functions. Which data are exposed to the user will dictate which solution methods may be utilized.

1.2.1 Picard Iteration

Perhaps the simplest solution method for this class of problems is Picard iteration (also known as fixed-point iteration or successive substitution). We describe Picard iteration in a more general sense in Appendix A. This type of iteration proceeds by sequentially solving single-physics applications and transferring the updated state variables or response functions to the other applications, and iterating in this manner until a consistent solution or some measure of failure is attained.

The main advantage for this method is its simplicity of implementation and flexibility. This method requires the minimum that can be expected of a physics code: that it can accept coupling parameters as inputs and return response functions. The only data that needs to be exposed are whichever response functions are required to evaluate the transfer functions for each of the single-physics applications. The solution state variables for a given application might not even need to be accessible, so long as the required response functions are. Single-physics applications may be treated as black boxes with internal workings opaque to the user. Termination of this sort of iteration is then generally determined by small changes in various response functions (1.6) for the coupled system from iteration to iteration. Because of this method's flexibility in regard to leveraging existing software and its relative simplicity of implementation it has been very widely used in solving coupled multiphysics problems. In particular, it has been the standard method utilized in the Consortium for Advanced Simulation of Light-Water Reactors (CASL). It is utilized in the VERA core simulator [40], the main product of CASL, as well as Tiamat [45], the coupling on which we focus later in this work.

In order to implement a Picard iteration in attempt to solve a coupled multiphysics problem, one needs to define some order in which to solve the single-physics applications. The two most common types of Picard iteration in this context (so named after their similarities to the corresponding stationary iterative methods for linear systems) are:

- Block Jacobi - Data transfers between all applications are carried out at the same time, and each set of physics is independently solved.
- Block Gauss-Seidel - Single-physics applications are sequentially solved with updated solutions transferred to the other applications as soon as they are obtained.

That is, in block Jacobi there are alternating phases of solving each of the single-physics applications, and then computing updated coupling parameters. Conversely, for block Gauss-Seidel the applications are solved one at a time in a sequential order, passing data as it is obtained. These represent the only two types of orderings when the coupled system comprises of only two applications, but additional orderings are possible when coupling more applications. This will be examined further in Section 6.2.

As a more concrete example , we consider the following two-application multiphysics system

$$f_0(x_0, r_{0,0}(x_1)) = 0, \tag{1.7}$$

$$f_1(x_1, r_{1,0}(x_0)) = 0. \tag{1.8}$$

We seek solutions x_0^* and x_1^* such that both $f_0(x_0^*, r_{0,0}(x_1^*)) = 0$ and $f_1(x_1^*, r_{1,0}(x_0^*)) = 0$. We can represent a block Jacobi iteration scheme for this system as follows:

- Given x_0^0, x_1^0.
- For $n = 0, 1, \ldots$
  - Solve $f_0(x_0^{n+1}, r_{0,0}(x_1^n)) = 0$ for x_0^{n+1}.
  - Solve $f_1(x_1^{n+1}, r_{1,0}(x_0^n)) = 0$ for x_1^{n+1}.

Note that in this, each of the transfer functions , $r_{0,0}$ and $r_{1,0}$, is evaluated using the approximate solutions that are present at the beginning of the iteration. We similarly represent the block Gauss-Seidel iteration as follows:

- Given x_0^0, x_1^0.
- For $n = 0, 1, \ldots$
  - Solve $f_0(x_0^{n+1}, r_{0,0}(x_1^n)) = 0$ for x_0^{n+1}.
  - Solve $f_1(x_1^{n+1}, r_{1,0}(x_0^{n+1})) = 0$ for x_1^{n+1}.

In this, x_0^{n+1} is transferred to the other application as soon as it is obtained, and f_1 is solved given this updated value. Assuming that x_0^{n+1} is closer to the solution than x_0^n, the residual equation f_1 given these updated values should be closer to the actual problem we are looking to solve, so this should result in an improved approximation to the solution.

In general, a block Gauss-Seidel scheme is expected to converge in fewer iteration than a block Jacobi scheme, as it keeps the applications more tightly coupled in a sense. Additionally, because a block Gauss-Seidel scheme keeps individual physics components more tightly coupled, it will likely converge for several problems where a block Jacobi scheme does not, particularly if the sets of physics are very tightly coupled. However, as each of the single-physics solves for block Jacobi is independent, it is possible to execute each solve simultaneously, whereas single-physics solves need to be executed serially for block Gauss-Seidel due to the sequential nature of the solves. In a parallel computing environment, this simultaneous solution of single-physics systems may result in significantly lower time per block Jacobi iteration than that for a block Gauss-Seidel iteration. However, significant reduction in per iteration run-time requires careful load balancing so that each of the single-physics solves takes roughly the same time.

Utilizing Picard iteration as a solution method for coupled multiphysics problems features several noteworthy drawbacks. One of the primary disadvantages of this method is its relatively slow rate of convergence. When convergent, the rate of convergence for this method should be expected to be at best q-linear (see Appendix A). Another drawback of this method is potential poor robustness. It may simply diverge, or it may require a ad hoc chosen numerical damping for convergence, which will be the case when we consider the Tiamat code coupling in Chapter 2.

1.2.2 Jacobian-Free Newton-Krylov

An alternative solution method to Picard iteration which has been employed in coupled multiphysics problems is Jacobian-free Newton-Krylov (JFNK) [29, 31]. In contrast with Picard iteration, where the individual sets of physics are treated in a partitioned manner and solved indepently, JFNK achieves a tighter coupling by solving all the sets of physics simultaneously. Because of this more tightly coupled treatment, and several other advantages that Newton-like methods offer, JFNK has become widely utilized in solving coupled multiphysics problems. For instance, JFNK is a foundational tool for the MOOSE (Multiphysics Object Oriented Simulation Environment) framework [20]. In MOOSE, the Galerkin finite-element method is used for discretization and geometric representation [7] and JFNK is used to solve the resulting systems of equations. The Bison nuclear fuel performance code [23], which is utilized in the Tiamat coupling that we introduce in Chapter 2, is built on this framework.

A more formal description of Newton's method, JFNK, and convergence theory for these methods, is given in Appendix A. As the name JFNK suggests, this method is derived from Newton's method, which solves the equation $F(u) = 0$ by iterating

$$u_{k+1} = u_k + d_k, \tag{1.9}$$

where the Newton direction, d_k, solves the linear equation

$$F'(u_k)d_k = -F(u_k). \tag{1.10}$$

In the context of coupled multiphysics problems, the residual F in the above equation is a monolithic residual which is comprised of the residual equations for each of the single-physics systems

$$F(\{x_m\}, \{p_{m,n}\}) = \begin{pmatrix} f_0(x_0, \{r_{0,i}(\{x_m\}, \{p_{m,n}\})\}, \{p_{0,j}\}) \\ f_1(x_1, \{r_{1,i}(\{x_m\}, \{p_{m,n}\})\}, \{p_{1,j}\}) \\ \vdots \\ f_{N_f-1}(x_{N_f-1}, \{r_{N_f-1,i}(\{x_m\}, \{p_{m,n}\})\}, \{p_{N_f-1,j}\}) \end{pmatrix} = 0, \tag{1.11}$$

where f_i is the the residual equation for single-physics system i. The Jacobian matrix corre-

sponding to this residual has the form

$$F'(\{x_m\}) = \begin{pmatrix} \frac{\partial f_0}{\partial x_0} & \frac{\partial f_0}{\partial x_1} & \cdots & \frac{\partial f_0}{\partial x_{N_f-1}} \\ \frac{\partial f_1}{\partial x_0} & \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_{N_f-1}} \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_{N_f-1}}{\partial x_0} & \frac{\partial f_{N_f-1}}{\partial x_1} & \cdots & \frac{\partial f_{N_f-1}}{\partial x_{N_f-1}} \end{pmatrix}. \tag{1.12}$$

In this, the off-diagonal blocks, $\frac{\partial f_i}{\partial x_j}$ for $i \neq j$, will be zero if none of the transfer functions $\{r_{i,k}\}$ corresponding to the single-physics residual i depends on x_j, and generally non-zero otherwise. There are several drawbacks to forming and storing the full Jacobian matrix. First, for even moderately large problems, forming the Jacobian may be prohibitively expensive, either with respect to computation or storage. If the entire set of state variables $\{x_m\}$ contains N variables, the Jacobian has N^2 entries, which may be too large to store. Even if there are several zero off-diagonal blocks, the amount of computation and storage required for the full Jacobian may be excessive. Second, it may not be possible to compute the Jacobian at all. Not only does it require differentiating each single-physics residual with respect to its own state variables, but the state variables for other applications for which it has a non-zero dependence as well. Even if a given physics code can compute a Jacobian for its residual equation with respect to its own state variables, the off-diagonal blocks may be difficult to obtain.

To avoid forming a Jacobian matrix, (1.10) may be solved in a matrix-free manner by utilizing a Krylov method. For this, one simply requires a method of computing the action of the Jacobian matrix on a given vector, $F'(u_k)v$. As with forming the full Jacobian, computing a Jacobian-vector product analytically may be infeasible or impossible. As a result, some approximation to the Jacobian-vector product may be the only possible option in many cases. In JFNK, the Jacobian-vector product is approximated using a finite difference. This finite difference approximation is as follows

$$F'(u_k)v = \frac{F(u_k + \epsilon v) - F(u_k)}{\epsilon}, \tag{1.13}$$

where ϵ is a perturbation parameter. We see that each finite-difference approximation requires only the evaluation of the residual equation at a small perturbation to the current solution u_k. The individual residual equations may need to be scaled if they deal with quantities of vastly different magnitude, otherwise selection of the perturbation factor ϵ in the forward-difference Jacobian-vector product may be problematic.

Returning to the example system in Equations (1.7)-(1.8), we seek a solution $u^* = \begin{pmatrix} x_0^* \\ x_1^* \end{pmatrix}$

to the monolithic system

$$F\begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = \begin{pmatrix} f_0(x_0, r_{0,0}(x_1)) \\ f_1(x_1, r_{1,0}(x_0)) \end{pmatrix} = 0. \tag{1.14}$$

In order to apply JFNK to solve this system, we simply require the ability to evaluate the perturbed residual about the current iterates x_0^n and x_1^n

$$F\begin{pmatrix} x_0^n + \epsilon v_0 \\ x_1^n + \epsilon v_1 \end{pmatrix} = \begin{pmatrix} f_0(x_0^n + \epsilon v_0, r_{0,0}(x_1^n + \epsilon v_1)) \\ f_1(x_1^n + \epsilon v_1, r_{1,0}(x_0^n + \epsilon v_0)) \end{pmatrix} = 0. \tag{1.15}$$

JFNK is attractive for several reasons. First, Newton-like methods feature fast local convergence. Second, Newton-like methods feature several globalization methods, i.e. line searches [17] or trust-region methods [30]. As a result, even with a poor initial iterate, Newton's method will converge to a solution or fail to do so in a predictable manner. Lastly, this method ensures consistency upon convergence. A Picard iteration generally terminates upon small change in response functions. However, small changes in response functions may not necessarily imply convergence of the state variables, so it is possible for a Picard iteration to declare convergence prematurely. However, because JFNK deals with the actual residuals for the application codes, on solution of the composite residual, the state variables for each physics application will also be converged.

However, JFNK comes with its own disadvantages. First, this method is more restrictive in implementation than Picard. Each of the codes needs to provide access to the state variables for which it is solving, and be able to return its residual. Many codes work as black boxes, in which inputs are specified, and output response functions are returned. The internal workings of the codes are not exposed to the user, so access to state variables or a residual may not be provided, or a residual might not be computed at all. Second, while the nonlinear iteration is generally expected to converge in few iterations, it may require many iterations in the linear solve phase, and as one residual evaluation is required per linear iteration, this may be prohibitively costly. In this case, it is necessary either to have a good preconditioner or in some way reduce the cost of residual evaluations in the linear iterations.

1.2.3 Nonlinear Elimination

An additional technique that has been used for coupled multiphysics problems in various fields is nonlinear elimination [38, 67]. This is not a solution method in itself, but it can provide flexibility in the choice of solution method for a coupled problem. This approach proceeds by using the Implicit Function Theorem to express the state variables for given a single-physics system as a function of the other single-physics state variables on which its residual equation

(1.5) depends, and using this relation to eliminate that system from the coupled problem. In this way, a coupled problem can be reduced to solving for only a subset of the single-physics systems, and the solution to any eliminated system can be recovered from the solutions for those that are solved. This can provide flexibility if the systems that are eliminated are preventing from utilizing more advanced solution methods, e.g. enabling the use of Newton's method by nonlinearly eliminating a single-physics system for which we can not access a residual or compute Jacobian information.

As an example, again consider the two-component coupled system given by Equations (1.7) and (1.8). The residual equation $f_1(x_1, r_{1,0}(x_0)) = 0$ implicitly defines x_1 as a function of x_0, so we denote the solution of this equation given any x_0 by $x_1(x_0)$. We can then rewrite (1.7) as

$$f_0(x_0, r_{0,0}(x_1(x_0))) = 0. \tag{1.16}$$

This reduces the coupled problem to a single-physics problem which we solve for only x_0, and we can apply some nonlinear solution method to solve this reduced equation. For instance, one can apply the Picard iteration by letting x_0^{n+1} be the solution of $f_0(x_0, r_{0,0}(x_1(x_0^n))) = 0$. However, this does not differ from block Gauss-Seidel for the coupled system, as this will alternate between solving f_1 for x_1 internally and using this value to update x_0 by solving f_0. More usefully, one may be able to solve this reduced system by a Newton-like method. In [34] the authors analyze full Newton's method applied to a nonlinearly eliminated system. Even without analytic Jacobian information, this technique may enable the use of JFNK if it is possible to nonlinearly eliminate all systems for which one can not compute and directly access state variables and residuals. For example, suppose that we have access to the state variables x_0 and can compute the residual f_0, but this information is not accessible for the second system. This would preclude using JFNK as described in the previous section, as this requires the ability to compute and access the state variables and residual for both systems. However, in this case it will be possible to apply JFNK to the reduced system (1.16), as this only requires the ability to evaluate the reduced residual.

Chapter 2

Tiamat Overview

2.1 Introduction

We now restrict our focus to the specific case of coupled multiphysics problems in the context of nuclear reactor simulation. The particular coupling on which we focus in this work is called Tiamat [45]. Tiamat is one of several code couplings being developed by the Consortium for Advanced Simulation of LWRs (CASL). Tiamat is intended as a tool for pellet-cladding interaction (PCI) analysis. A nuclear fuel rod is comprised of a stack of UO_2 fuel pellets enclosed in a Zircaloy tube, referred to as the cladding. Initially there is a gap between the fuel pellets and the cladding, and this closes during operation as a result of thermal expansion and fission product swelling in the fuel pellets, and inward displacement of the cladding due to external coolant pressure. PCI refers to cladding failure due to strains caused by contact between fuel pellets and the surrounding cladding during operation, resulting in release of radioactive fission products into the coolant. Such behavior usually occurs as a result of rapid changes in the local power distribution during power maneuvers. PCI is a problem of significant interest to the nuclear industry, as reactor operating restrictions established to mitigate this issue may result in reduced power generation [10]. It is then desirable to improve upon modeling and simulation techniques for this problem, as this could result in improvements in fuel design and quantification of safety margins.

PCI failure is governed by the complex interplay of the thermal, mechanical, and chemical behavior within a fuel rod, so an integral fuel performance code is required to simulate these processes. Bison is a single-rod fuel performance code being developed to model this coupled thermal-mechanical-chemical behavior in order to assess safety margins and fuel rod design [23]. Bison can be used to calculate key figures of merit, such as maximum hoop stress, which indicate the potential for PCI failure within a given fuel rod. In order for Bison to accurately compute these quantities of interest, it needs to be provided high-fidelity representations of

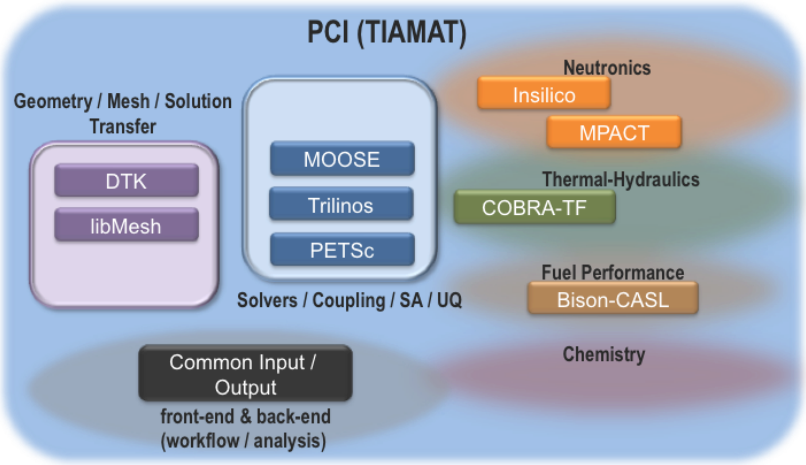


Figure 2.1: Codes utilized in the Tiamat code coupling

several conditions within the reactor, specifically the rate of heat generation from fission in the fuel, which is the main thermal source in the fuel, and the clad surface temperature, which acts as a thermal boundary condition. This is accomplished in Tiamat by coupling Bison with other CASL single-physics codes to provide the necessary feedback, specifically the MPACT neutronics [14] and COBRA-TF thermal hydraulics [50] codes. Bison in turn provides feedback to the other single-physics codes, as the fuel temperatures affects the fission heat generation rate, and the heat flux from the fuel to the coolant affects several important coolant properties. We are concerned with finding a solution to this fully-coupled problem as efficiently as possible.

In the remainder of this chapter, we will first overview the major codes participating in this coupling in Section 2.2. We will then describe precisely the problem at hand and how it has previously been solved by Picard iteration in Section 2.3, and lastly we will briefly present numerical results illustrating the behavior of Picard iteration for this problem in Section 2.3.3.

2.2 Participating Codes

The collection of codes being actively developed and utilized within the CASL project is known as the Virtual Environment for Reactor Applications [61]. Tiamat utilizes a rather large subset of the codes in the VERA suite. A schematic of the major participating codes is shown in Figure 2.1. In this we see the single-physics application codes as well as a variety of utilities for driving input/output, data transfers, and solvers. In this section we overview the major codes that are utilized in Tiamat.

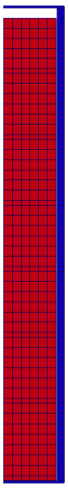


Figure 2.2: Bison 2-D axisymmetric finite element representation of a fuel rod; the radial dimension is scaled by a factor of 100

2.2.1 Bison

As has been stated, the Bison fuel performance code [23] is being developed to provide single-rod, fuel performance modeling capability, which we utilize in Tiamat in order to calculate figures of merit which indicate the potential for PCI failures in PWRs. Bison is built upon Idaho National Laboratory's MOOSE framework [20], so it uses the finite element method for geometric representation and JFNK to solve the resulting systems of partial differential equations. It includes 1D, 2D, and full 3D modeling capabilities. The fuel rod geometric representation utilized by Bison within Tiamat is a 2D R-Z axially-symmetric, smeared-pellet model. This is illustrated in Figure 2.2, where the red region is fuel and the blue region is cladding. The system of equations that Bison solves is as follows

$$\rho C_p \frac{\partial T}{\partial t} + \nabla \cdot (-k \nabla T) - q = 0, \tag{2.1}$$

$$\nabla \cdot \sigma + \rho f = 0. \tag{2.2}$$

These equations represent conservation of energy and momentum respectively. In Equation (2.1), T, ρ, C_p, k, and q are the temperature, density, specific heat, thermal conductivity, and volumetric energy generation due to fission respectively. This is a standard heat equation, with source given by fission energy production. In Equation (2.2), σ is the Cauchy stress tensor, and f is the body force per unit mass. This equation governs the displacement field, i.e. deformation of the fuel pellets and cladding due to mechanical forces. Bison utilizes various constitutive relationships, depending on internal material models, which relate the displacement to the Cauchy

stress tensor, through a strain tensor. The material models are strongly influenced by the time history, so the Bison application is inherently a transient code. The state variables that Bison solves for are the temperature distribution and the displacement field. The coupling between the temperature solution and the mechanical solution is non-linear due to the complex dependency of the material properties on temperature, stress, and strain. In addition to the above, Bison accounts for changing chemical composition of the fuel and clad by the following equation representing species conservation

$$\frac{\partial C}{\partial t} + \nabla \cdot \mathbf{J} + \lambda C - S = 0, \tag{2.3}$$

where C, λ, S, and $\mathbf{J}$ are the concentration, decay constant, source, and mass flux respectively for a given chemical species. This affects material properties, and causes swelling in the pellets due to production of fission product.

A core-simulator is traditionally uses a quasi-static model, with a series of steady-state calculations [45], but since the material models within Bison are strongly dependent on the time history this code must be run transient. Then, when coupling Bison with the other application codes, we solve Bison one time step at a time, while the other codes are solved in steady-state mode. We represent the residual equation resulting from discretizing Equations (2.1) and (2.2) for a given time step as follows

$$f_B(x_B, T_c, q) = 0. \tag{2.4}$$

In this, the vector of state variables x_B is comprised of fuel temperature and displacement unknowns. Note, we also identify the quantities T_c and q in this equation. These are coupling parameter sub-vectors. That is, they are the $z_{i,j}$ vectors defined in Section 1.1.2. These are quantities that depend on solutions to other application codes which affect the solution of (2.4). In this case, T_c is the cladding surface temperature and q is the fission heat generation rate. The dependence of Equation (2.1) on q is obvious, and T_c acts as the outer boundary condition in this equation. In (2.4), we suppress the independent parameter sub-vector notation, $\{p_{i,k}\}$, from Equation (1.3), which can include things like geometry specifications or other reactor conditions which are do not depend on the solution of any application codes. As Bison utilizes JFNK internally, it computes this residual directly, though access for the user is not easily provided.

2.2.2 COBRA-TF (CTF)

COBRA-TF (CTF) [50] is a thermal-hydraulic simulation code designed for LWR analysis. CTF is currently being developed and maintained by the Reactor Dynamics and Fuel Management Group at the Pennsylvania State University. CTF uses a two-fluid, three-field representation of

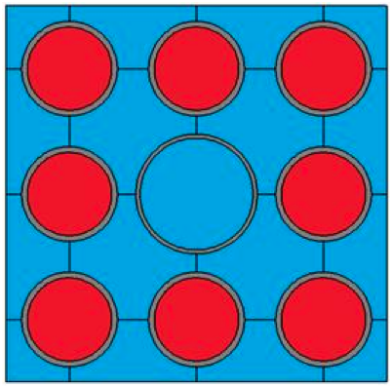


Figure 2.3: Subchannel representation utilized by CTF for a 3x3 array of fuel rods

the two-phase flow. The three fields considered are liquid film, liquid droplets and vapor. For each field, the equations and solved are as follows

$$\frac{\partial}{\partial t}(\alpha_k \rho_k) + \nabla \cdot (\alpha_k \rho_k \vec{v}_k) = L_k + M_k^T, \tag{2.5}$$

$$\frac{\partial}{\partial t}(\alpha_k \rho_k \vec{v}_k) + \nabla \cdot (\alpha_k \rho_k \vec{v}_k \vec{v}_k^*) = \alpha_k \rho_k \vec{g} - \alpha_k \nabla P + \nabla \cdot [\alpha_k (\tau_k + T_k)] + \vec{M}_k^L + \vec{M}_k^d + \vec{M}_k^T, \tag{2.6}$$

$$\frac{\partial}{\partial t}(\alpha_k \rho_k h_k) + \nabla \cdot (\alpha_k \rho_k h_k \vec{v}_k) = -\nabla \cdot [\alpha_k (\vec{Q}_k + \vec{q}_k^T)] + \Gamma_k h_k + q'''_{w,k} + \alpha_k \frac{\partial P}{\partial t}. \tag{2.7}$$

These equations represent conservation of mass, momentum, and energy respectively. In these equations, the subscript k denotes the field under consideration. Some important quantities to note in this equation are the void fraction α_k, the density ρ_k, the velocity field $\vec{v}_k$, the enthalpy h_k, the pressure P, and the volumetric wall heat transfer $q'''_{w,k}$. For discretization, CTF utilizes a simplified subchannel form of these equations. A subchannel refers to the gap between a collection of rods as illustrated in Figure 2.3. CTF defines control volumes over axial sections of subchannels, and enforces Equations (2.5), (2.6), and (2.7) over these control volumes. The method used by CTF to solve these equations is called the Semi-Implicit Method for Pressure-Linked Equations [43]. CTF additionally includes several internal models useful for reactor analysis, such as spacer grid models and built-in material properties.

As mentioned in the previous section, with the exception of Bison we utilize the application codes in steady-state mode, so we seek a steady state solution to Equations (2.5), (2.6), and (2.7). However, internally CTF always solves the time dependent form of these equations. To approximate a steady state solution CTF uses a pseudo-steady-state solver which marches forward in time for a particular flow/power state until it has been determined that the solution has become sufficiently close to steady state. CTF tracks five quantities to determine the steady

state convergence, which are as follows:

- The amount of energy stored in the fluid.
- The amount of energy stored in the solids.
- The amount of mass stored in the system.
- Global energy balance.
- Global mass balance.

These quantities are defined precisely in [50]. CTF declares steady-state convergence upon sufficiently small changes in these quantities relative to the time step.

We represent the discretized residual of the steady-state form of Equations (2.5), (2.6), and (2.7) as

$$f_C(x_C, q'') = 0. \tag{2.8}$$

In this, x_C represents the discretized form of the velocity field, density, enthalpy, etc. The coupling parameter vector q'' is the heat flux from the fuel to the coolant. As was previously stated, the CTF conservation equations are discretized by integrating over control volumes, and this converts the volumetric wall heat transfer $q'''_{w,k}$ to the total wall heat transfer $q_{w,k}$. This value can also be computed by integrating the heat flux from the fuel q'' over the cladding surface area which bounds the control volume, so the heat flux represents a source in (2.7). We note here that while CTF is governed by Equation (2.8), it does not include the capability to evaluate this residual. It only has the capability to solve for its state variables and return various response functions.

2.2.3 MPACT

The reactor core simulator MPACT [14] has been developed collaboratively by researchers at the University of Michigan and Oak Ridge National Laboratory to provide an advanced pin-resolved transport capability within VERA. This code solves the neutron transport equation

$$\begin{aligned}\Omega \cdot \nabla\psi(r,\Omega,E) + \Sigma_t(r,E)\psi(r,\Omega,E) = &\int_0^\infty dE' \int_{4\pi} d\Omega'\, \Sigma_s(r, \Omega\cdot\Omega', E' \to E)\psi(r,\Omega',E') \\ &+ \frac{\chi(r,E)}{4\pi k_{eff}} \int_0^\infty dE' \int_{4\pi} d\Omega'\, \nu\Sigma_f(r,E')\psi(r,\Omega',E'). \end{aligned} \tag{2.9}$$

In this equation, ψ is called the angular flux, and it is a measure of intensity of neutrons passing through a point in space with a given direction and energy. It is a function of position r, direction Ω, and energy E. Σ_t is referred to as the total cross section, Σ_s is the scattering cross section,

and Σ_f is the fission cross section. These quantities represent probability densities of a neutron undergoing each given type of interaction. Lastly, k_{eff} is called the dominant eigenvalue, and this represents the average number of neutrons born per fission event that go on to undergo a fission event. In 3-dimensional space, ψ is a 6-dimensional function (3 space, 2 angle, and energy), so due to the high dimensionality, several simplifications are generally made when solving this problem. First, rather than solving for the directionally dependent ψ, one typically solves for various angular moments of the angular flux. Second, energy dependence is generally treated through the multigroup approximation. In this, the energy variable is partitioned into energy groups given by the collection of intervals $\{[E_g, E_{g-1}]\}_{g=1}^{N_G}$, and (2.9) is integrated over each of these energy group. This results in the following

$$\begin{aligned}\Omega \cdot \nabla \psi_g(r,\Omega) + \Sigma_{t,g}(r)\psi_g(r,\Omega) = &\sum_{g'=1}^{N_G} \int_{4\pi} d\Omega' \, \Sigma_{s,g'\to g}(r, \Omega\cdot\Omega')\psi_{g'}(r,\Omega') \\ &+ \frac{\chi_g(r)}{4\pi k_{eff}} \sum_{g'=1}^{N_G} \int_{4\pi} d\Omega' \, \nu\Sigma_{f,g'}(r)\psi_{g'}(r,\Omega'), \quad g = 1,\ldots,N_G, \quad (2.10)\end{aligned}$$

where

$$\psi_g(r,\Omega) = \int_{E_g}^{E_{g-1}} dE \, \psi(r,\Omega,E), \quad (2.11)$$

$$\chi_g(r) = \int_{E_g}^{E_{g-1}} dE \, \chi(r,E), \quad (2.12)$$

$$\Sigma_{x,g} = \frac{\int_{E_g}^{E_{g-1}} dE \, \Sigma_x(r,E)\phi(r,E)}{\int_{E_g}^{E_{g-1}} dE \, \phi(r,E)}. \quad (2.13)$$

In (2.13), the subscript x represents any of the reaction types, and the weighting function ϕ, known as the scalar flux, is the zeroth angular moment of the angular flux. As the scalar flux is not known a priori, the group cross sections are typically computed using approximate weighting functions. Accurately solving this multigroup equation requires high-fidelity approximation of these group cross sections, and such approximation can be rather expensive.

There are many methods that have been developed to solve Equation (2.10). Some of these methods include discrete ordinates angular treatment with either finite difference or method of characteristics spatial treatment [51], Monte Carlo methods [36], expansion in spherical harmonics [51], and hybrid deterministic/Monte Carlo methods [64]. In MPACT, the workhorse method for solving this equation is a 2D/1D scheme in space accelerated with coarse-mesh finite difference (CMFD) [28]. The 2D/1D scheme decomposes the geometry as an axial stack of planes in the radial direction, as illustrated in Figure 2.4. The 2D problem is solved using

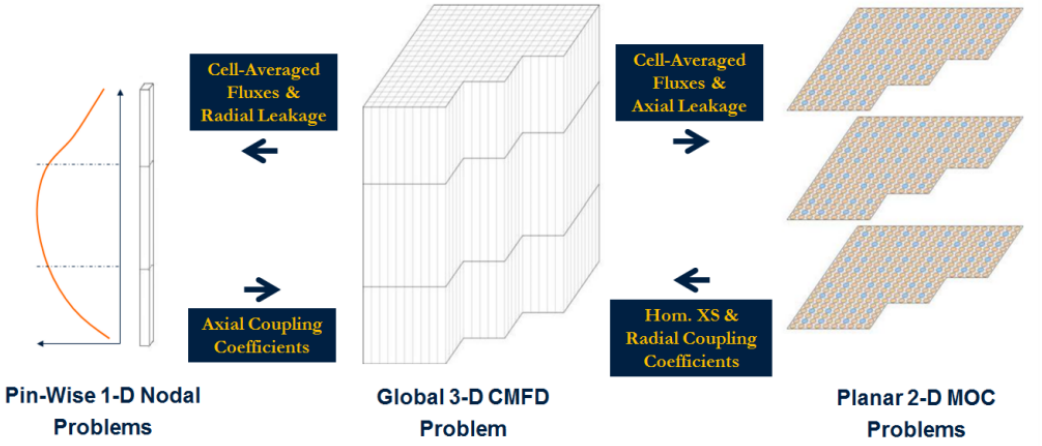


Figure 2.4: Schematic of the MPACT solution process, coupling 2D/1D treatment of the transport equation with 3D CMFD acceleration (from [57])

the method of characteristics, and the 1D problem is solved by a lower order approximation. For computing multigroup cross sections, MPACT utilizes either the subgroup method [15] or the embedded self-shielding method [65]. CMFD accelerates this 2D/1D scheme by globally rebalancing the flux with a diffusion-like equation on a coarse mesh using coefficients computed from a fine mesh approximate solution. The coarse mesh equation is formulated in such a way that its solution is consistent with the fine mesh solution.

As with the first two applications, we represent the discretized form of Equation (2.10) in the following residual form

$$f_M(x_M, T_f, T_w, \rho_w) = 0. \tag{2.14}$$

In this, the vector of state variables x_m consists of group scalar fluxes and the dominant eigenvalue. The dependencies T_f, T_w, and ρ_w represent the fuel temperature, coolant temperature, and coolant density respectively. These affect the solution of (2.10) through the dependence of the multigroup cross sections on these material properties. Like CTF, MPACT does not feature the capability to evaluate this residual given some input set of state variables x_M.

2.2.4 Data Transfer Kit (DTK)

The Data Transfer Kit (DTK) [56] is a software package being developed at Oak Ridge National Laboratory which provides parallel services for mesh and geometry searches and data transfers for multiphysics applications. It it useful for passing data between meshes or geometries which may not conform spatially and have arbitrary parallel distribution. It achieves good scaling properties for data transfers through use of a method called the rendezvous algorithm. In this,

an intermediate decomposition of the source mesh/geometry is utilized in order to localize and load balance search operations.

In Tiamat, DTK was used for determining the parallel MPI communication mappings and for moving all data between codes. Issues such as unit conversions, as each application code uses different units, and differing coordinate systems are handled within these data transfer objects. We describe the DTK data transfer objects which are utilized in Tiamat in more detail when describing the formulation of the coupled problem in Section 2.3.1

2.2.5 PIKE

PIKE is a package of Trilinos [27] which provides interfaces and various utilities for black-box code coupling. This framework is leveraged heavily throughout Tiamat. One of the main features utilized is the PIKE solver class, which provides an interface for solving couplings between black-box physics codes which interact through transfers of coupling parameter data, i.e. problems formulated as described in Section 1.1.2. Two implementations of PIKE solvers are provided in the package: a block Jacobi solver and a block Gauss-Seidel solver. These solvers are concrete implementations of the Picard iterations that were outlined in Section 1.2.1. These solvers work with abstract interfaces for model evaluator and data transfer objects which are also defined in PIKE. A PIKE model evaluator is wrapper class for single-physics application codes. The main functionality of this class is to solve the underlying application and return response functions. Additionally, it provides several other routines for step control for time-dependent application codes. Tiamat includes concrete model evaluator implementations for each of the three application codes. A PIKE data transfer is an abstraction of a transfer function. The purpose of this class is to provide an interface for mapping data computed by one application code to input arrays for other codes. In Tiamat, DTK is the primary driver in the implementations of the PIKE data transfer objects.

Some other important pieces from this package that are utilized throughout Tiamat are as follows

- Status tests - Status tests are used to check for convergence or failure of an iteration. Some of the status tests included with PIKE check for successful local convergence of each participating physics application, small changes in response functions of interest, or exceeding a maximum number of iterations.

- Solver observers - Solver observers are used to provide additional functionality to PIKE solvers. They give the ability to insert action at some point within the solution process, e.g. before or after a full solve or a single iteration. For instance, observers which are used in Tiamat carry out the pre-solve initialization phase which we describe in Section 2.3, and post-process output upon a successful solve.

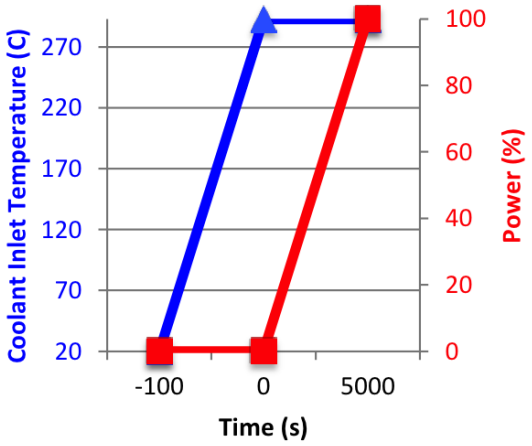


Figure 2.5: Variation of the inlet coolant temperature (in blue) and power level (in red) during ramp of Bison from cold zero-power (CZP) to hot full-power (HFP)

- Multiphysics distributor - The multiphysics distributor class provides several utilities for parallel task management. It can create and provide MPI sub-communicators for model evaluator and data transfers, and is useful for checking whether an application or transfer exists on a given MPI process. Additionally, the multiphysics distributor includes utilities for parallel output.

2.3 Tiamat Simulation Process

We are interested in utilizing Tiamat to carry out simulations at normal operating conditions, referred to as hot full-power (HFP). However, as we have mentioned the material models in Bison have a strong time-dependence, so an initialization phase needs to be followed in order to bring Bison from shut-down conditions, referred to as cold zero-power (CZP), up to HFP. A direct coupling between MPACT and CTF can simulate HFP directly, so this is used in order to obtain an estimate of HFP conditions, which is used to ramp Bison to this estimated state. There are several options for how this HFP estimation can be carried out. First, one or more iterations of stand-alone coupled MPACT/CTF may be carried out. Another option is to read in data from a restart file generated offline from a previous MPACT/CTF or Tiamat run. The Bison ramping process, which is illustrated visually in Figure 2.5, essentially simulates start-up of the reactor first from CZP to a state referred to as hot zero-power (HZP) in which the coolant is warmed to the operating inflow temperature but no power is being produced in the fuel rods, and then from HZP to the estimated HFP conditions. The Tiamat solution process is then given by the following .

1. Estimate the conditions at HFP in stand-alone MPACT/CTF as described above.

2. Model the transition from CZP to HZP in Bison. In this step, Bison simulates over a period of 100 seconds with the inlet coolant temperature linearly varied from 293K to 565K.

3. Model the transition from HZP to HFP in Bison. In this step, Bison simulates over a period of 48 hours with clad surface temperatures linear varied from 565K to the estimated value from CTF in Step 1, and the power distribution linearly varied from zero to the estimated value from MPACT in Step 1.

4. Model the reactor state at HFP for one or more time step.

The first three steps in this process represent a fixed-cost for any given Tiamat simulation. The last step, in which we solve the fully-coupled system given by the three single-physics codes for one or more time step, is where we focus our attention, as this is where improvement can be made through advancement in techniques for solving tightly coupled multiphysics problems. We first describe more precisely the problem being solved in this step, and the methods which have been used to solve it prior to this work.

2.3.1 Fully-Coupled Problem Formulation

In the fully-coupled solve phase, we seek solutions to each of the single-physics applications such that each system is simultaneously solved. That is, we seek single-physics solutions , x_B^*, x_C^*, and x_M^*, which solve the following monolithic residual equation

$$F\begin{pmatrix} x_B \\ x_C \\ x_M \end{pmatrix} = \begin{pmatrix} f_B(x_B, T_c, q) \\ f_C(x_C, q'') \\ f_M(x_M, T_f, T_w, \rho_w) \end{pmatrix} = 0, \tag{2.15}$$

where f_B, f_C, and f_M are the single-physics residual equations defined in Equations (2.4), (2.8), and (2.14). As we had previously noted, the auxiliary conditions for each of the single-physics residual equations comes from quantities computed by other physical systems, so it remains to explicitly define these coupling parameter sub-vectors. We do this using the concept of transfer functions introduced in Section 1.1.2. In the fully coupled solve phase, five data transfer objects are utilized, and they are as follows

- Bison to CTF: The heat flux from the fuel to the coolant computed by Bison is transferred to CTF. Computation of this quantity involves computing the outward normal derivative

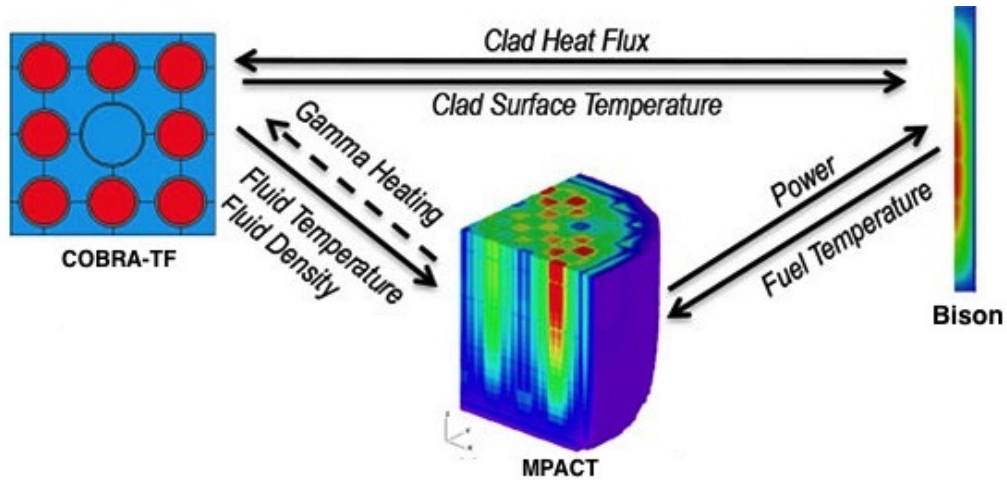


Figure 2.6: Data transfers utilized by Tiamat in the fully-coupled hot full-power (HFP) solve phase

of the temperature distribution. We denote this transfer function as

$$q'' = r_{C,B}(x_B). \tag{2.16}$$

- Bison to MPACT: The fuel temperature computed by Bison is transferred to MPACT. Denote this transfer function as
$$T_f = r_{M,B}(x_B). \tag{2.17}$$

- CTF to Bison: The clad surface temperature computed by CTF is transferred to Bison. Denote this transfer function as
$$T_c = r_{B,C}(x_C). \tag{2.18}$$

- CTF to MPACT: The coolant temperature and density computed by CTF is transferred to MPACT. Denote this transfer function as
$$\begin{pmatrix} T_w \\ \rho_w \end{pmatrix} = r_{M,C}(x_C) = \begin{pmatrix} r_{M,C,T}(x_C) \\ r_{M,C,\rho}(x_C) \end{pmatrix}. \tag{2.19}$$

- MPACT to Bison: The power distribution computed by MPACT is transferred to Bison. Denote this transfer function as
$$q = r_{B,M}(x_M). \tag{2.20}$$

For each of these data transfers, the general process for evaluating the transfer function is to first compute the desired quantity on the set of source processes, and then perform a parallel communication to pass the data to the correct target processes. In general, this parallel transfer can involve interpolation or some more complex method of moving data between meshes. However, in Tiamat this is simplified by performing all transfers on a coupling mesh, which corresponds to the coarsest application axial mesh in the simulation. In this case, this is the axial mesh used by both CTF and MPACT. The coupling mesh consists of the volumes given by subdividing each fuel rod at a given set of axial bounds. Transfers between MPACT and CTF are volume-to-volume, so these simply consist of copying data to the correct volume. For transfers from Bison to the other application, a post-processor first averages the values defined on the Bison finite element mesh over the coupling mesh cells, and transfers these averages. Transfers to Bison are volume-to-point, and this is accomplished by simply assigning the value at a given finite element node the averaged value for whichever volume in the coupling mesh contains it.

Now given the notation in Equations (2.16)–(2.20), we can represent the fully-coupled system (2.15) in the following form

$$F\begin{pmatrix} x_B \\ x_C \\ x_M \end{pmatrix} = \begin{pmatrix} f_B(x_B, r_{B,C}(x_C), r_{B,M}(x_M)) \\ f_C(x_C, r_{C,B}(x_B)) \\ f_M(x_M, r_{M,B}(x_B), r_{M,C,T}(x_C), r_{M,C,\rho}(x_C)) \end{pmatrix} = 0. \tag{2.21}$$

Then, at each time step in the fully-coupled HFP solve phase we seek solutions x_B^*, x_C^*, and x_M^* such that Equation (2.21) is satisfied.

2.3.2 Solution of Fully-Coupled Problem

Given the problem formulation from the previous section, we now describe the methods which have been previously utilized to solve this fully-coupled HFP problem. As was stated in Section 2.2.5, Tiamat utilizes PIKE solvers for solving the fully-coupled problem, and currently only two PIKE solver implementations are included in the package: a block Gauss-Seidel solver and a block Jacobi solver. Tiamat can utilize either of these solvers. The process that block Gauss-Seidel follows for this coupling is shown in Algorithm 1. In this, the order that we have chosen to solve the applications is MPACT, followed by Bison, and lastly CTF. Again, these solves must be executed sequentially. After the process for ramping Bison to HFP, each of the of the application codes will have an estimated solution to its system at HFP conditions. These estimated solutions serve as the initial iterates, x_B^0, x_C^0, and x_M^0, for the algorithm when solving for the first time step. In solving for more than one time step, the final solution from the previous time step is used as the initial iterate for the current step.

Algorithm 1 Block Gauss-Seidel Nonlinear Solve for Tiamat

Given x_B^0, x_C^0, and x_M^0.
for $k = 0, 1, \ldots$ until converged **do**
    Transfer Bison to MPACT, $T_f^k = r_{M,B}(x_B^k)$.
    Transfer CTF to MPACT, $T_w^k = r_{M,C,T}(x_C^k)$ and $\rho_w^k = r_{M,C,\rho}(x_C^k)$.
    Solve $f_M(x_M, T_f^k, T_w^k, \rho_w^k) = 0$ for x_M^{k+1}.
    Transfer MPACT to Bison, $q^{k+1} = r_{B,M}(x_M^{k+1})$.
    Transfer CTF to Bison, $T_c^k = r_{B,C}(x_C^k)$.
    Solve $f_B(x_B, T_c^k, q^{k+1}) = 0$ for x_B^{k+1}.
    Transfer Bison to CTF, $q''_{k+1} = r_{C,B}(x_B^{k+1})$.
    Solve $f_C(x_C, q''_{k+1}) = 0$ for x_C^{k+1}.
end for

Algorithm 2 Damped Block Gauss-Seidel Nonlinear Solve for Tiamat

Given x_B^0, x_C^0, and x_M^0.
for $k = 0, 1, \ldots$ until converged **do**
    Transfer Bison to MPACT, $T_f^k = r_{M,B}(x_B^k)$.
    Transfer CTF to MPACT, $T_w^k = r_{M,C,T}(x_C^k)$ and $\rho_w^k = r_{M,C,\rho}(x_C^k)$.
    Solve $f_M(x_M, T_f^k, T_w^k, \rho_w^k) = 0$ for x_M^{k+1}.
    Transfer MPACT to Bison, $q^{k+1} = r_{B,M}(x_M^{k+1})$.
    if $k > 0$ **then**
        Damp the transferred power, $q^{k+1} = (1 - \omega)q^k + \omega q^{k+1}$.
    end if
    Transfer CTF to Bison, $T_c^k = r_{B,C}(x_C^k)$.
    Solve $f_B(x_B, T_c^k, q^{k+1}) = 0$ for x_B^{k+1}.
    Transfer Bison to CTF, $q''_{k+1} = r_{C,B}(x_B^{k+1})$.
    Solve $f_C(x_C, q''_{k+1}) = 0$ for x_C^{k+1}.
end for

Algorithm 3 Damped Block Jacobi Nonlinear Solve for Tiamat

Given x_B^0, x_C^0, and x_M^0.
for $k = 0, 1, \ldots$ until converged **do**
    Transfer Bison to MPACT, $T_f^k = r_{M,B}(x_B^k)$.
    Transfer CTF to MPACT, $T_w^k = r_{M,C,T}(x_C^k)$ and $\rho_w^k = r_{M,C,\rho}(x_C^k)$.
    Transfer MPACT to Bison, $q^k = r_{B,M}(x_M^k)$.
    if $k > 0$ **then**
        Damp the transferred power, $q^k = (1-\omega)q^{k-1} + \omega q^k$.
    end if
    Transfer CTF to Bison, $T_c^k = r_{B,C}(x_C^k)$.
    Transfer Bison to CTF, $q_k'' = r_{C,B}(x_B^k)$.
    Solve $f_M(x_M, T_f^k, T_w^k, \rho_w^k) = 0$ for x_M^{k+1}.
    Solve $f_B(x_B, T_c^k, q^k) = 0$ for x_B^{k+1}.
    Solve $f_C(x_C, q_k'') = 0$ for x_C^{k+1}.
end for

At normal operating conditions, the procedure outlined in Algorithm 1 will fail to converge due to oscillation in the solution induced by certain error modes [24]. A standard method that has been utilized to remedy this issue is to introduce a numerical damping. Algorithm 2 shows the same block Gauss-Seidel solution process, now with a damping applied to the power update. In this, $\omega \in (0, 1]$ is a damping factor. While these damping factors are chosen ad hoc, it has been observed that factors in the range 0.3-0.6 generally perform fairly well [24, 39, 66].

The process that the block Jacobi solver follows is shown in Algorithm 3. This algorithm alternates between phases of performing all data transfers and then solving all application codes. As with block Gauss-Seidel, a damping on the power update is included, and this is generally required in order to obtain convergence of the iteration. It is important to note that the solves in this case need not be performed in sequential order, and this allows block Jacobi to more effectively utilize parallelism. Figure 2.7 shows the parallel distribution utilized in Tiamat. Each of the application codes exists in its own process space, and due to this design choice each of the applications can perform a solve simultaneously. Conversely, for block Gauss-Seidel there is significantly more processor idle time, as all processes associated with a given application are idle while another application is solving. Hence, in an ideal scenario in which each application requires the same solve time, the time required per block Jacobi iteration would be roughly one third of the time per block Gauss-Seidel iteration.

Convergence of Tiamat is judged at both a local and global level. Local convergence refers to convergence of the individual single-physics application codes, for which each of the applications has its own set of criteria. Global convergence refers to convergence of the whole coupled system. Global convergence is generally determined by checking for small changes in various response

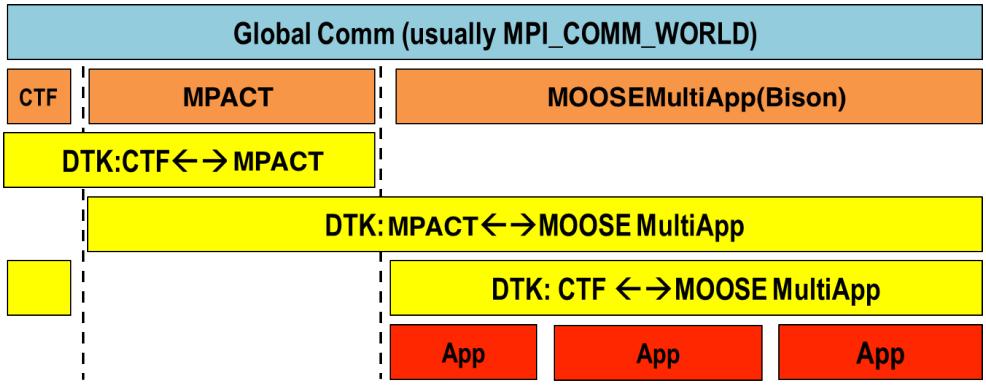


Figure 2.7: MPI communication layers in Tiamat

functions between coupled iterations. PIKE status tests are used in order to determine the global convergence of the coupled system. In order for a simulation to declare successful global convergence, each of the application codes must have successfully converged locally, and the following conditions for must be satisfied

- Bison: the absolute change in the maximum fuel temperature from iteration to iteration must be less than some user-defined tolerance ϵ_T.
- CTF: the absolute change in the maximum coolant temperature and maximum clad surface temperature from iteration to iteration must be less than ϵ_T.
- MPACT: the relative change (in the l_2 norm) of the power distribution is less than some user-defined tolerance ϵ_q between Picard iterations, and the absolute change in k_{eff} must be less than some tolerance ϵ_k.

The iteration fails if it takes more than a prescribed maximum iteration count.

2.3.3 Performance of Picard Iteration

We now consider some test problems which illustrate the behavior of Picard iteration when applied to this problem. For these tests, we simulate a single fuel rod, and solve for one time step at HFP. 12 processes are utilized: 8 allocated to MPACT, 3 to Bison, and 1 to CTF. Within MPACT, these tests utilize 8-group cross sections. This is a rather coarse energy mesh, but this is not of great concern as we are concerned more with convergence behavior than high accuracy in the solution. We simply note that higher fidelity cross sections will result in higher solve times for MPACT than what is observed in these results. The inputs for this problem are specified in greater detail in Appendix C.

Table 2.1: Comparison of block Gauss-Seidel vs block Jacobi for single rod Tiamat simulation at various power levels. Power damping factor $\omega = 0.5$ and max iteration count = 25

Power Level	Method	Iterations	Solve Time (s)	k_{eff}	$T_{f,max}$
25%	Gauss-Seidel	9	303	1.23708	483.55
	Jacobi	14	392	1.23708	483.54
50%	Gauss-Seidel	7	292	1.23105	694.09
	Jacobi	15	407	1.23106	693.92
75%	Gauss-Seidel	8	332	1.22493	931.20
	Jacobi	18	462	1.22497	930.18
100%	Gauss-Seidel	8	353	1.21857	1194.67
	Jacobi	DNC			
125%	Gauss-Seidel	12	445	1.21214	1505.16
	Jacobi	DNC			

First, in Table 2.1 are results from Tiamat simulations utilizing both block Gauss-Seidel and block Jacobi as the solution method at several power levels. The power levels are listed as a percentage of the rated power specified in the input. This determines the magnitude of the power compute by MPACT, which is then passed to Bison. The power appears linearly as a source in Bison's residual equation (2.4), so the power level affects the strength of the coupling between the applications. Increasing the power increases the strength of the coupling, and thus the difficulty of the problem. We see that with the exception of 125% power, each block Gauss-Seidel iteration converges with comparable iteration counts and run times. At 125% power there is a significant increase in these quantities. For block Jacobi, there is a more obvious upward trend throughout. Performance is comparable at the two lowest power levels, but as it is increased further the iteration counts rise sharply, and it begins to fail to converge. These results indicate that for both methods the performance may suffer as the strength of coupling between the sets of physics becomes greater.

The dominant eigenvalue k_{eff} and the maximum fuel temperature $T_{f,max}$ are listed in this table to indicate the level of agreement between these two solution methods. We see that the two methods agree well for the cases where they both converge.

Lastly, we observe in this table that the run times for block Jacobi are generally significantly higher than those for block Gauss-Seidel. This can be explained by the application timing breakdown presented in Table 2.2. This considers the 75% power case, as this is the highest power level at which both methods converge. In this table, we see the time required by each of the application solves and data transfers. We first note that each of the data transfers requires a negligible amount of the time, and the application solves dominate the cost. In particular, with this allocation of processes Bison solves take roughly 80–90% of the time per iteration for

Table 2.2: Breakdown of solve and transfer timings (in seconds) for Tiamat HFP solve phase at 75% power

	Phase	Num Calls	Total Time	Time/Call
Gauss-Seidel	Bison Solve	8	140.3	17.5
	CTF Solve	8	4.0	0.5
	MPACT Solve	8	24.9	3.1
	Bison to CTF	8	4.8e-3	6.0e-4
	Bison to MPACT	8	2.9e-3	3.6e-4
	CTF to Bison	8	5.6e-3	7.0e-4
	CTF to MPACT	8	9.7e-3	1.2e-3
	MPACT to Bison	8	2.9e-1	3.6e-2
Jacobi	Bison Solve	18	315.2	17.5
	CTF Solve	18	10.6	0.59
	MPACT Solve	18	66.6	3.7
	Bison to CTF	18	5.2e-2	2.8e-3
	Bison to MPACT	18	1.0e-2	5.4e-4
	CTF to Bison	18	5.9e-1	3.2e-2
	CTF to MPACT	18	2.1e+0	1.1e-1
	MPACT to Bison	18	6.9e-1	3.7e-2

Gauss-Seidel. Because of this poor balance, little reduction in per iteration run-time results from simultaneously solving the applications in block Jacobi. It may be possible to bring Bison and MPACT into better balance with a different processor allocation. However, currently CTF is only parallelized to run with one processor per fuel assembly, and this may make good balancing of all three application codes problematic.

Next, Figure 2.8 shows the dependence of the performance of block Gauss-Seidel on the damping factor ω. We note that block Jacobi behaves similarly. We observe that the performance of the method is very strongly dependent on this parameter. Fairly consistent performance is achieved over damping factors in range 0.4—0.6, and iteration counts rise rapidly away from this range. We also note that the performance depends noticeably on the power level. The left side of the curves remain static but on the right there is an upward trend. As a result, a damping factor that was suitable for a lower power level may be quite bad at higher powers. Additionally, the damping factor at which the method performs optimally is dependent on the power level. The optimum level shifts to the left as the power increased, and hence it difficult to say prior to simulation that one has chosen the best damping level for a given problem.

These results illustrate poor robustness of Picard iteration with respect to the power and damping level, and this could become problematic when simulating more tightly coupled systems. Because of theses numerical weakness, it is of interest to utilize some alternative method to Picard iteration for fully-coupled HFP solve. JFNK would be a good alternative due to its

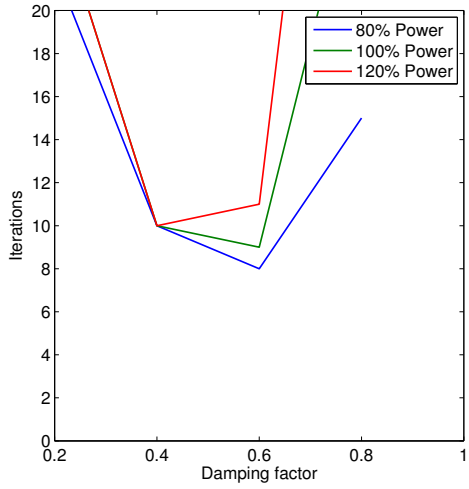


Figure 2.8: Block Gauss-Seidel iterations to convergence for Tiamat single-rod simulation, varying the damping factor and power level

fast local convergence and good robustness from globalization methods. However, these codes do not provide the functionality required to implement JFNK as described in Section 1.2.2. Neither CTF nor MPACT computes the residuals (2.8) and (2.14), and while Bison employs JFNK internally, access to its residual vector is not provided to the users. Because of the restrictiveness of the codes utilized in the coupling, we need a method which does not require significantly more information than Picard it implement. The method that we will consider for this purpose is called Anderson acceleration, and we introduce this concept in the following chapter.

Chapter 3

Analysis of Anderson Acceleration

In this chapter, we consider the algorithm proposed by Anderson in [2] which has come to be known as Anderson acceleration or mixing. The algorithm has also essentially been rediscovered and discussed under various names including Pulay mixing or Direct Inversion in the Iterative Subspace (DIIS) in [47,48] for electronic structures calculations, Nonlinear Krylov Acceleration in [9], and IQL-ILS [16] for fluid-structure interaction calculations. Anderson originally considered the algorithm in the context of solving a particular class of nonlinear integral equations, but it has subsequently gained popularity as a method to accelerate the convergence rates of Picard iterations. Given some mapping $G : \mathbb{R}^N \to \mathbb{R}^N$, the method to solve $G(u) = u$ proceeds as shown in Algorithm 4. In this, m is an algorithmic parameter that dictates the maximum depth for which previous iterate information is stored. We refer to the algorithm for any particular value of m as Anderson-m. Anderson-0 corresponds to standard Picard iteration. We need to store both u and one of $F(u)$ or $G(u)$ at each iterate, so the storage burden is a maximum of $2(m + 1)$ vectors. The additional cost in implementing this method as opposed to Picard iteration is dominated by the solution of the minimization problem. In the constrained form in Algorithm 4, the problem may be solved with a linear program or Lagrange multipliers. There are several equivalent ways to describe the algorithm with the minimization problem given in unconstrained form. If the minimization problem is solved in the l_2 norm, as is standard, we then need only solve a linear least-squares problem. In the form originally posed by Anderson, we determine $(\theta_1^{(k)}, \ldots, \theta_{m_k}^{(k)})$ which solve the problem

$$\min_{(\theta_1,\ldots,\theta_{m_k})} \left\| F(u_k) + \sum_{i=1}^{m_k} \theta_i^{(k)} (F(u_{k-i}) - F(u_k)) \right\|, \tag{3.3}$$

and then calculate

$$u_{k+1} = G(u_k) + \sum_{i=1}^{m_k} \theta_i^{(k)} (G(u_{k-i}) - G(u_k)). \tag{3.4}$$

Algorithm 4 Anderson acceleration with inexact function evaluations

Given initial iterate u_0 and storage depth parameter $m \in \mathbb{N}$.
Set $u_1 = G(u_0)$.
Set $\hat{F}_0 = G(u_0) - u_0$.
for k = 1,2,... **do**
    Set $m_k = \min\{m, k\}$.
    Set $F_k = G(u_k) - u_k$.
    Determine $\alpha^{(k)} = (\alpha_0^{(k)}, \ldots, \alpha_{m_k}^{(k)})$ which solves

$$\min_{\alpha=(\alpha_0,\ldots\alpha_{m_k})^T} \left\| \sum_{i=0}^{m_k} \alpha_i F_{k-m_k+i} \right\|, \tag{3.1}$$

    subject to the constraint $\sum_{i=0}^{m_k} \alpha_i = 1$.
    Set

$$u_{k+1} = \sum_{i=0}^{m_k} \alpha_i^{(k)} G(u_{k-m_k+i}). \tag{3.2}$$

end for

Here the coefficients $\{\alpha_i^{(k)}\}$ and $\{\theta_i^{(k)}\}$ are related by $\alpha_i^{(k)} = \theta_{m_k-i}^{(k)}$ for $0 \leq i < m_k$ and $\alpha_{m_k}^{(k)} = 1 - \sum_{i=1}^{m_k} \theta_i^{(k)}$. Note that this requires the computation of the entire least-squares coefficient matrix at each iteration. A third form which may be implemented more efficiently expresses the algorithm in terms of differences between successive iterates. For this, we determine $\gamma^{(k)} = (\gamma_1^{(k)}, \ldots, \gamma_{m_k}^{(k)})^T$ which solve the problem

$$\min_{\gamma} \|F(u_k) - \mathcal{F}_k \gamma\|, \tag{3.5}$$

where $\mathcal{F}_k = (\Delta F_{k-m_k+1}, \ldots, \Delta F_k)$ and $\Delta F_i = F(u_i) - F(u_{i-1})$. Then, we calculate

$$u_{k+1} = u_k + F(u_k) - (\mathcal{U}_k + \mathcal{F}_k)\gamma^{(k)}, \tag{3.6}$$

where $\mathcal{U}_k = (\Delta u_{k-m_k+1}, \ldots, \Delta u_k)$ and $\Delta u_i = u_i - u_{i-1}$. In this form $\{\alpha_i^{(k)}\}$ and $\{\gamma_i^{(k)}\}$ are related by $\alpha_0 = \gamma_0$, $\alpha_i = \gamma_i - \gamma_{i-1}$ for $1 \leq i \leq m_k - 1$, and $\alpha_{m_k} = 1 - \gamma_{m_k-1}$. To update $\mathcal{F}_k$ and $\mathcal{U}_k$ between iterations we append the new difference vectors at the end and drop the first columns if the storage limit has been reached. Solving the least-squares problem by taking QR factorization of $\mathcal{F}_k$ each iteration then results in a marginal cost of $O(m_k^2 N)$ over Picard iteration. However, as mentioned in [62], we can obtain the QR factorization of $\mathcal{F}_k$ from that of $\mathcal{F}_{k-1}$, and this reduces the cost of the factorization to $O(m_n N)$ operations. We describe the method by which the QR factorization may be updated from iteration to iteration in Section 4.2.3. If N is much larger than m, the savings in operations from updating the QR factors in this manner is not

substantial, especially if the evaluation of G is relatively expensive. However, there may be an appreciable difference in storage for problems of interest. Storing $\mathcal{F}_k$ and computing a QR factorization each iteration requires at least temporary storage of both $\mathcal{F}_k$ and the Q factor, both of which have the same size. Conversely, when updating the QR factorization directly, only the Q and R factors need to be stored, and for reasonably sized storage parameter m, the storage burden of the R factor is negligible. Hence, for problems where N is very large, this yields an appreciable difference.

3.1 Review of Literature

We begin with an overview of previous work related to Anderson acceleration. Anderson acceleration is one of several methods which has been proposed and studied for the purpose of accelerating the rate of convergence for slowly converging series. Many acceleration methods are sequence transformations, i.e. methods which utilize the iterates produced by a slowly converging sequence $\{x_k\}$ in order to construct a new, faster converging sequence $\{y_k\}$. This includes scalar acceleration methods such as Richardson extrapolation, the Aitken delta–squared process, and the Wynn epsilon method [8]. These methods can be applied to vector sequences in a component-wise manner, or one may utilize a vector extrapolation method such as reduced rank extrapolation, mimimal polynomial extrapolation, or modified minimal polynomial extrapolation [52]. Anderson acceleration differs from these methods in that it does not construct the original fixed-point iteration sequence. The acceleration from this method is derived from storing a history of previous iterates in order to compute a better approximation to the solution than the fixed-point iteration.

Again, Anderson acceleration was first proposed by Anderson in [2]. While he provides no rigorous analysis of the method, Anderson discusses several practical considerations. For instance, he claims that in practice, the method seems to perform best for small values of m, generally less than 10. Additionally, he describes how incorporate a mixing parameter into the algorithm. For this, the only difference in the above algorithm is that we replace (3.2) with

$$u_{k+1} = (1 - \beta_k) \sum_{i=0}^{m_k} \alpha_i u_{k-m_k+i} + \beta_k \sum_{i=0}^{m_k} \alpha_i G(u_{k-m_k+i}), \tag{3.7}$$

where the sequence of scalars $\{\beta_k\}$ are referred to as mixing parameters. This corresponds to a damping factor in the context of Picard iteration. The choice of an appropriate mixing parameter is often necessary for the iterates to converge or obtain an acceptable rate of convergence.

Subsequent analysis has primarily dealt with showing the equivalence between this method and other methods in some sense. First, in [19] the authors show that Anderson acceleration

may be viewed as a sort of quasi-Newton method. It is worth noting that the authors consider Anderson acceleration to solve $F(u) = 0$, not specifically as a fixed-point solver. To see this equivalence, consider (3.6). We rewrite this, now including mixing parameters, as:

$$u_{k+1} = u_k + \beta_k F(u_k) - (\mathcal{U}_k + \beta_k \mathcal{F}_k)\gamma^{(k)}. \tag{3.8}$$

Then, assuming $\mathcal{F}_k$ is full rank, $\gamma^{(k)}$ is obtained by solving the normal equations, which gives $\gamma^{(k)} = (\mathcal{F}_k^T \mathcal{F}_k)^{-1} \mathcal{F}_k^T F(u_k)$. Substituting this into (3.8), we obtain

$$u_{k+1} = u_k - G_k F(u_k), \tag{3.9}$$

where we have defined

$$G_k \equiv -\beta_k I + (\mathcal{U}_k + \beta_k \mathcal{F}_k)(\mathcal{F}_k^T \mathcal{F}_k)^{-1} \mathcal{F}_k^T. \tag{3.10}$$

In [19], it is claimed that this matrix G_k forms an approximate inverse Jacobian of $F(x)$ in the sense that the matrix minimizes $\|G_k + \beta_k I\|_F$ over all matrices which satisfy the inverse multisecant condition

$$G_k \mathcal{F}_k = \mathcal{U}_k. \tag{3.11}$$

This is referred to this as the Type-II method in [19]. Along these lines, they define the Anderson's family of methods according to

$$u_{k+1} = u_k + \beta_k F(u_k) - (\mathcal{U}_k + \beta_k \mathcal{F}_k) V_k^T F(u_k), \tag{3.12}$$

where $V_k \in \mathbb{R}^{n \times m}$ satisfies $V_k^T \mathcal{F}_k = I$. The choice $V_k^T = (\mathcal{F}_k^T \mathcal{F}_k)^{-1} \mathcal{F}_k^T$ gives the Type-II method above. Conversely, $V_k^T = (\mathcal{U}_k^T \mathcal{F}_k)^{-1} \mathcal{U}_k^T$, under the assumption that $\mathcal{U}_k^T \mathcal{F}_k$ is nonsingular, gives what is referred to in [19] as the Type-I method. For this choice of V_k^T we have

$$G_k \equiv -\beta_k I + (\mathcal{U}_k + \beta_k \mathcal{F}_k)(\mathcal{U}_k^T \mathcal{F}_k)^{-1} \mathcal{U}_k^T. \tag{3.13}$$

As in [62], from the Sherman-Morrison-Woodbury formula this corresponds to the approximate Jacobian

$$J_k = G_k^{-1} = -\frac{1}{\beta_k} I + \frac{1}{\beta_k} (\mathcal{U}_k + \beta_k \mathcal{F}_k)(\mathcal{U}_k^T \mathcal{U}_k)^{-1} \mathcal{U}_k^T, \tag{3.14}$$

which satisfies the direct multisecant condition $J_k \mathcal{U}_k = \mathcal{F}_k$. According to [19] this minimizes $\|J_k + \frac{1}{\beta_k} I\|_f$ among matrices satisfying the direct multisecant condition.

In [62], it is shown that when G is linear, i.e. $G(u) = Au + b$ with $A \in \mathbb{R}^{N \times N}$ and $b \in \mathbb{R}^N$, the Type-I method is "essentially equivalent" to the Arnoldi method applied to the equivalent problem $(I - A)x = b$ when both are applied to the same initial iterate. It is assumed that the Type-I iteration is untruncated, i.e. $m_k = k$ at each step. The methods are equivalent in the

sense that the iterates of one method may easily be obtained from those of the other according to the following: $u_k^{\text{Arnoldi}} = \sum_{i=0}^{k} \alpha_i^{(k)} u_i^{\text{Type-I}}$ and $u_{k+1}^{\text{Type-I}} = A u_k^{\text{Arnoldi}} + b$, where $\{u_k^{\text{Type-I}}\}$ and $\{x_k^{\text{Arnoldi}}\}$ are the sequences of Type-I and Arnoldi iterates respectively.

It is also shown in [62] that standard Anderson acceleration on the linear function $G(x) = Ax + b$ is "essentially equivalent" in the same sense to GMRES applied to the equivalent problem $(I - A)x = b$. The authors obtain this result assuming no mixing, i.e. $\beta_k = 1$ at each step. Let $\{u_k^{\text{AA}}\}$ and $\{u_k^{\text{GMRES}}\}$ be the sequences of Anderson acceleration and GMRES iterates respectively. Under the assumptions that $\|r_{k-1}^{\text{GMRES}}\| = \|b - (I - A)u_{k-1}^{\text{GMRES}}\| \neq 0$ and that $\|r_{j-1}^{\text{GMRES}}\| > \|r_j^{\text{GMRES}}\|$ for each j such that $0 < j < k$, the iterates of one method may easily obtained from those of the other as follows: $u_k^{\text{GMRES}} = \sum_{i=0}^{k} \alpha_i^{(k)} u_i^{\text{AA}}$ and $u_{k+1}^{\text{AA}} = A u_k^{\text{GMRES}} + b$. This equivalence between the methods is established as follows. Using the fact that $\sum_{i=0}^{k} \alpha_i^{(k)} = 1$, for untruncated Anderson acceleration with mixing parameters equal to one and with linear G, the Anderson iterates may be rewritten as follows

$$u_{k+1}^{\text{AA}} = A\bar{u}_{k+1} + b, \tag{3.15}$$

where we define

$$\bar{u}_{k+1} = \sum_{i=0}^{k} \alpha_i^{(k)} u_i^{\text{AA}}. \tag{3.16}$$

The equivalence of the methods in the given sense is established once it is shown that $u_k^{\text{GMRES}} = \bar{u}_{k+1}$. This is shown inductively in [62] by showing that if for $1 \leq j \leq k$, $\{u_1^{\text{AA}} - u_0, \ldots, u_j^{\text{AA}} - u_0\}$ is a basis for $\mathcal{K}_j = \text{span}\{r_0, (I - A)r_0, \ldots, (I - A)^{j-1} r_0\}$, then $\bar{u}_{j+1}$ and u_j^{GMRES} solve the same minimization problem. It is then shown for $1 \leq j \leq k$, that $\{u_1^{\text{AA}} - u_0, \ldots, u_j^{\text{AA}} - u_0\}$ is indeed a basis for $\mathcal{K}_j$. Hence, it follows that for $1 \leq j \leq k$, it holds that $u_j^{\text{GMRES}} = \sum_{i=0}^{j} \alpha_i^{(j)} u_i^{\text{AA}}$. The authors also claim that Anderson-m is equivalent in the same sense to a truncated variant of GMRES, and a variant of Anderson acceleration which is restarted after m steps is equivalent in the same sense to GMRES(m).

In [46], the behavior of untruncated Anderson acceleration for linear problems is further characterized. The above result is extended to show the equivalence to GMRES for general nonzero mixing parameters. This again relies on showing that under the non-stagnation assumption for GMRES it holds that $\text{span}\{u_1^{\text{AA}} - u_0, \ldots, u_k^{\text{AA}} - u_0\} = \mathcal{K}_k$. As a result, regardless of whatever nonzero mixing parameters are used, the result is that $\bar{u}_{k+1}^{\text{AA}} = u_k^{\text{GMRES}}$. Additionally, in [46] mixing parameters are calculated for the linear problem which are ideal in the sense that the residual is minimized at each step. However, it is claimed that iteration with these ideal mixing parameters does not essentially accelerate convergence when compared to Anderson acceleration with arbitrary nonzero mixing parameters due to the GMRES equivalence in each case.

In both [62] and [46] it is shown that if GMRES stalls at step k with $r_{k-1}^{\text{GMRES}} = r_k^{\text{GMRES}}$, then $u_k^{\text{AA}} = u_{k+1}^{\text{AA}}$. While GMRES continues and converges to the solution, Anderson acceleration stalls indefinitely at this point. In [62], it is also suggested that near-stagnation of GMRES, that is consecutive iterates which are nearly equal, could lead to ill-conditioning of the least-squares problem for the next step of Anderson acceleration. Because of this potential numerical weakness, untruncated Anderson acceleration should not in general be used as an alternative to GMRES for linear problems when the size of the problem is not prohibitive with regard to storage. In [62] poor conditioning of the least-squares problem is posed as another potential numerical weakness of this algorithm, so a method is proposed in which m_k is modified throughout the iteration in order to maintain acceptable conditioning of the least-squares problem.

3.2 Standard Convergence Analysis

With the exception of full GMRES, the results in the previous section show the relationship between Anderson acceleration and other methods for which there is not sufficient theory. As a result, these results do not provide satisfactory convergence theory, especially for nonlinear problems and the limited memory variations of Anderson acceleration. Therefore, we attempted to expand upon the convergence theory for this method, and we have proved several local convergence theorems for both linear and nonlinear problems. In all of the forthcoming results, with the exception of Section 3.4, the norm is not specified, so the results hold using any vector norm and corresponding induced matrix norm. Additionally, each result relies on the fact that $(\alpha_0^{(k)}, \ldots, \alpha_{m_k}^{(k)})$ is a solution to (3.1), so we have

$$\left\| \sum_{i=0}^{m_k} \alpha_i^{(k)} F(u_{k-m_k+i}) \right\| \leq \left\| \sum_{i=0}^{m_k} \alpha_i F(u_{k-m_k+i}) \right\|, \tag{3.17}$$

for any set of coefficients such that $\sum_{i=0}^{m_k} \alpha_i = 1$. In particular, we have

$$\left\| \sum_{i=0}^{m_k} \alpha_i^{(k)} F(u_{k-m_k+i}) \right\| \leq \|F(u_k)\|. \tag{3.18}$$

We note that several results from this section are published in [58].

3.2.1 Analysis for Linear Problems

In this section, we characterize the convergence for Anderson applied to solve $u = G(u)$, where G is linear, i.e. $G(u) \equiv Au + b$ with $A \in \mathbb{R}^{N \times N}$ and $b \in \mathbb{R}^N$. In this case, the corresponding

fixed-point residual is given by

$$F(u) = G(u) - u = b - (I - A)u.$$

Theorem 3.1. *If $\|A\| = c < 1$, then the Anderson iterates, for any m or $m = \infty$, $\{u_i\}$ converge r-linearly to the solution $u^* = (I - A)^{-1}b$ with r-factor c, and the residuals $\{F(u_i)\}$ converge q-linearly to zero with q-factor c.*

Proof. First, because $\|A\| < 1, I - A$ is nonsingular and thus u^* exists. We will first prove the convergence for the residuals, and the convergence of the iterates will follow from this. Because $\sum_{i=0}^{m_k} \alpha_i^{(k)} = 1$, we can rewrite the $k + 1$ residual as

$$\begin{aligned}
F(u_{k+1}) &= b - (I - A)u_{k+1} \\
&= \sum_{i=0}^{m_k} \alpha_i^{(k)} [b - (I - A)(Au_{k-m_k+i} + b)] \\
&= \sum_{i=0}^{m_k} \alpha_i^{(k)} [b - (I - A)b - A(I - A)u_{k-m_k+i}] \\
&= \sum_{i=0}^{m_k} \alpha_i^{(k)} A[b - (I - A)u_{k-m_k+i}] \\
&= A \sum_{i=0}^{m_k} \alpha_i^{(k)} F(u_{k-m_k+i}).
\end{aligned}$$

Then, by (3.18)

$$\|F(u_{k+1})\| \leq c \left\| \sum_{i=0}^{m_k} \alpha_i^{(k)} F(u_{k-m_k+i}) \right\| \leq c\|F(u_k)\|.$$

This proves the result for the residuals. Next, we define $e = u - u^*$, so we have $F(u) = -(I - A)e$. Then, because $\|A\| < 1$, we have $\|(I - A)^{-1}\| \leq \frac{1}{1-c}$. Thus, we have

$$\|e_k\| \leq \frac{1}{1-c}\|F(u_k)\| \leq \frac{1}{1-c}c^k\|F(u_0)\| \leq \frac{1+c}{1-c}c^k\|e_0\|,$$

which is in fact r-linear convergence with r-factor c. □

3.2.2 Analysis for Nonlinear Problems

We now turn our attention to solving the problem $u = G(u)$ where G may be nonlinear. Much of the following analysis relies in Assumption 3.1, which implies the standard assumptions for local convergence of Newton's methods.

Assumption 3.1.

1. *There is* $u^* \in \mathbb{R}^N$ *such that* $F(u^*) = G(u^*) - u^* = 0$.
2. G *is Lipschitz continuously differentiable in the ball* $\mathcal{B}_{\hat{\rho}}(u^*) = \{u : \|e\| \leq \hat{\rho}\}$ *for some* $\hat{\rho} > 0$.
3. *There is* $c \in (0,1)$ *such that for all* $u, v \in \mathcal{B}_{\hat{\rho}}(u^*)$, $\|G(u) - G(v)\| \leq c\|u - v\|$.

The second is implies F being Lipschitz continuously differentiable with the same Lipschitz constant. Additionally, the last of these assumptions implies that $\|G'(u)\| \leq c < 1$ for all $u \in \mathcal{B}_{\hat{\rho}}(u^*)$. In particular, $\|G'(u^*)\| < 1$, which implies that $F'(u^*) = G'(u^*) - I$ is nonsingular. In the following section, we will also use the following Lemma.

Lemma 3.1. *Assume that Assumption 3.1 hold and let* $\gamma > 0$ *denote the Lipschitz constant for* F' *in* $\mathcal{B}_{\hat{\rho}}(u^*)$. *Then for* $\rho \leq \hat{\rho}$ *sufficiently small and all* $u \in \mathcal{B}_{\rho}(u^*)$

$$\|F(u) - F'(u^*)e\| \leq \frac{\gamma}{2}\|e\|^2, \tag{3.19}$$

and

$$(1 - c)\|e\| \leq \|F(u) \leq (1 + c)\|e\|. \tag{3.20}$$

We now prove a local r-linear convergence result. The result holds for any general iteration of the form

$$u_{k+1} = \sum_{i=0}^{m_k} \alpha_i^{(k)} G(u_{k-m_k+i}), \tag{3.21}$$

for any $m \in \mathbb{N}$ with $m_k = \min\{m, k\}$ such that the coefficients satisfy the following assumption.

Assumption 3.2.

1. $\|\sum_{i=0}^{m_k} \alpha_i^{(k)} F(u_{k-m_k+i})\| \leq \|F(u_k)\|$ *holds.*
2. $\sum_{i=0}^{m_k} \alpha_i^{(k)} = 1$.
3. *There is some* M_α *such that* $\sum_{i=0}^{m_k} |\alpha_i^{(k)}| \leq M_\alpha$ *for all* $k \geq 0$.

The first two are trivially satisfied by Anderson acceleration, however there is no reason to assume that the third is satisfied in general. For the special case where $m = 1$ and the minimization problem is solved in the l_2 norm, we will see that this assumption is in fact satisfied. There are several ways the algorithm may be modified so that this last assumption is guaranteed to be satisfied for more general cases, such as:

- Restarting the iteration if the coefficient sum exceeds some tolerance.

- Imposing a bound constraint on the sum $\sum_{i=0}^{m_k} |\alpha_i|$ in the linear least squares problem and solving the problem with the method of [13].
- Solving the minimization problem in the l_1 or l_∞ norms, adding a bound constraint, and formulating the resulting problem as a linear program, for which there are many efficient solvers.

The first is the simplest, and based on our experience unlikely to significantly affect the iteration. In numerical experiments, we have not observed examples where the coefficient absolute value sum becomes unreasonably large, even in cases where the minimization problem becomes highly ill-conditioned.

Theorem 3.2. *Let Assumption 3.1 hold, and let $c < \hat{c} < 1$. Then if u_0 is sufficiently close to u^* the iterates defined by* (3.21) *given Assumption 3.2 converge r-linearly to u^* with r-factor no greater than $\hat{c}$. In fact,*

$$\|F(u_k)\| \leq \hat{c}^k \|F(u_0)\|, \tag{3.22}$$

and

$$\|e_k\| \leq \frac{1+c}{1-c} \hat{c}^k \|e_0\|. \tag{3.23}$$

Proof. We let $u_0 \in \mathcal{B}_{\hat{\rho}}(u^*)$ and assume that ρ is small enough so that the conclusions of Lemma 3.1 hold. We will prove (3.22). (3.23) will follow from (3.22) and Lemma 3.1. Recall that we let γ be the Lipschitz constant for F'. We let ρ be small enough so that $\rho < 2(1-c)/\gamma$ and

$$\frac{\frac{c}{\hat{c}} + \left(\frac{M_\alpha \gamma \rho}{2(1-c)}\right) \rho \hat{c}^{-m-1}}{1 - \frac{\gamma \rho}{2(1-c)}} \leq 1. \tag{3.24}$$

Then, reduce $\|e_0\|$ further so that

$$\frac{M_\alpha (c + \gamma\rho/2)}{1-c} \hat{c}^{-m} \|F(u_0)\| \leq \frac{M_\alpha (1+c)(c+\gamma\rho/2)}{1-c} \hat{c}^{-m} \|e_0\| \leq \rho. \tag{3.25}$$

We will proceed by induction on K. Assume that for all k such that $0 \leq k \leq K$ we have

$$\|F(u_k)\| \leq \hat{c}^k \|F(u_0)\|. \tag{3.26}$$

Clearly, this holds for $K = 0$. Equations (3.25) and (3.26) imply that $\|e_k\| \leq \rho$ for $1 \leq k \leq K$. Hence, by (3.19)

$$F(u_k) = F'(u^*) e_k + \Delta_k,$$

where

$$\|\Delta_k\| \le \frac{\gamma}{2}\|e_k\|^2. \tag{3.27}$$

This gives

$$G(u_k) = u^* + G'(u^*)e_k + \Delta_k. \tag{3.28}$$

Then, the next iterate is given by

$$\begin{aligned} u_{K+1} &= \sum_{i=0}^{m_K} \alpha_i^{(K)}(u^* + G'(u^*)e_{K-m_K+i} + \Delta_{K-m_K+i}) \\ &= u^* + \sum_{i=0}^{m_K} \alpha_i^{(K)} G'(u^*)e_{K-m_K+i} + \bar{\Delta}_K, \end{aligned} \tag{3.29}$$

because $\sum_{i=0}^{m_K} \alpha_i^{(K)} = 1$. Here

$$\bar{\Delta}_K = \sum_{i=0}^{m_K} \alpha_i^{(K)} \Delta_{K-m_K+i}.$$

We then need to bound $\bar{\Delta}_K$. (3.27) and (3.28) imply that

$$\|\bar{\Delta}_K\| \le \sum_{i=0}^{m_K} |\alpha_i^{(K)}|\gamma\|e_{K-m_K+i}\|^2/2. \tag{3.30}$$

Next, Lemma 3.1, the induction hypothesis, and the fact that

$$K - m_K + i = K - \min\{m, K\} + i \ge K - m$$

imply that

$$\begin{aligned} \|e_{K-m_K+i}\| &\le \frac{1}{1-c}\|F(u_{K-m_K+i})\| \\ &\le \frac{1}{1-c}\hat{c}^{K-m_K+i}\|F(u_0)\| \\ &\le \frac{1}{1-c}\hat{c}^{K-m}\|F(u_0)\| \le \frac{1}{1-c}\hat{c}^{-m}\|F(u_0)\|. \end{aligned} \tag{3.31}$$

Then, because $\sum_{i=0}^{m_K} |\alpha_i^{(K)}| \le M_\alpha$ and $\|e\| \le \rho$ for the previous iterates, we have

$$\begin{aligned} \|\bar{\Delta}_K\| \le \frac{\gamma}{2(1-c)}\|F(u_0)\| \sum_{i=0}^{m_K} |\alpha_i^{(K)}| \|e_{K-m_K+i}\| &\le \frac{M_\alpha\gamma\rho}{2(1-c)}\hat{c}^{K-m}\|F(u_0)\| \\ &\le \frac{M_\alpha\gamma\rho}{2(1-c)}\hat{c}^{-m}\|F(u_0)\|. \end{aligned} \tag{3.32}$$

Next, we write (3.29) as

$$e_{K+1} = \sum_{i=0}^{m_K} \alpha_i^{(K)} G'(u^*) e_{K-m_K+i} + \bar{\Delta}_K. \tag{3.33}$$

From (3.31), we have

$$\| \sum_{i=0}^{m_K} \alpha_i^{(K)} G'(u^*) e_{K-m_K+i} \| \leq \frac{M_\alpha c}{1-c} \hat{c}^{-m} \|F(u_0)\|. \tag{3.34}$$

Then, combining (3.33) with (3.32) and (3.34) gives

$$\|e_{K+1}\| \leq \|F(u_0)\| \left(\frac{M_\alpha (c + \gamma\rho/2)}{1-c} \right) \hat{c}^{-m} \leq \rho.$$

Since $\|e_{K+1}\| \leq \rho \leq \hat{\rho}$, we may apply (3.28) with $k = K+1$ to obtain

$$F(u_{K+1}) = (G'(u^*) - I) e_{K+1} + \Delta_{K+1},$$

where

$$\|\Delta_{K+1}\| \leq \frac{\gamma}{2} \|e_{K+1}\|^2 \leq \frac{\gamma\rho}{2(1-c)} \|F(u_{K+1})\|. \tag{3.35}$$

Then, from (3.33) and the fact that $G'(u^*)$ and $G'(u^*) - I$ commute, we have

$$\begin{aligned}
F(u_{K+1}) &= G'(u^*) \sum_{i=0}^{m_K} \alpha_i^{(K)} (G'(u^*) - I) e_{K-m_K+i} + (G'(u^*) - I)\bar{\Delta}_K + \Delta_{K+1} \\
&= G'(u^*) \sum_{i=0}^{m_K} (\alpha_i^{(K)} F(u_{K-m_K+i}) - \alpha_i^{(K)} \Delta_{K-m_K+i}) + (G'(u^*) - I)\bar{\Delta}_K + \Delta_{K+1} \\
&= G'(u^*) \sum_{i=0}^{m_K} \alpha_i^{(K)} F(u_{K-m_K+i}) - \bar{\Delta}_K + \Delta_{K+1}.
\end{aligned}$$

We assumed that $\rho < 2(1-c)/\gamma$, so $\frac{\rho\gamma}{2(1-c)} < 1$, and thus

$$\|F(u_{K+1}) - \Delta_{K+1}\| \geq |\|F(u_{K+1})\| - \|\Delta_{K+1}\|| \geq \left(1 - \frac{\gamma\rho}{2(1-c)}\right) \|F(u_{K+1})\|.$$

Therefore, by Assumption 3.2, the induction hypothesis, and (3.32) we have

$$\begin{aligned}\left(1-\frac{\gamma\rho}{2(1-c)}\right)\|F(u_{K+1})\| &\leq c\|\sum_{i=0}^{m_K}\alpha_i^{(K)}F(u_{K-m_K+i})\|+\|\bar{\Delta}_K\| \\ &\leq c\|F(u_K)\|+\frac{M_\alpha\gamma\rho}{2(1-c)}\hat{c}^{K-m}\|F(u_0)\| \\ &\leq \left(\frac{c}{\hat{c}}+\frac{M_\alpha\gamma\rho}{2(1-c)}\hat{c}^{-m-1}\right)\hat{c}^{K+1}\|F(u_0)\|.\end{aligned}$$

Hence, because we assumed (3.24), we have

$$\|F(u_{K+1})\| \leq \frac{\left(\frac{c}{\hat{c}}+\frac{M_\alpha\gamma\rho}{2(1-c)}\hat{c}^{-m-1}\right)}{\left(1-\frac{\gamma\rho}{2(1-c)}\right)}\hat{c}^{K+1}\|F(u_0)\| \leq \hat{c}^{K+1}\|F(u_0)\|.$$

This completes the induction. □

3.2.3 Numerical Tests

To illustrate some of the ideas presented in this section, we consider as an example solving the Chandrasekhar H-equation [11]. In continuous form, we seek a function $H \in C[0,1]$ which satisfies

$$H(\mu) = G(H) \equiv \left(1-\frac{\omega}{2}\int_0^1 \frac{\mu}{\mu+\nu}H(\nu)\,\mathrm{d}\nu\right)^{-1}, \tag{3.36}$$

where $\omega \in [0,1]$ is a parameter. From [58], the solution $H^*(\mu) \geq 1$ satisfies

$$\|H^*\|_\infty \leq \min\left(3, \frac{1}{\sqrt{1-\omega}}\right). \tag{3.37}$$

Additionally, for $\epsilon > 0$ sufficiently small and $u, v \in \mathcal{B}_\epsilon(H^*)$

$$\|G(u)-G(v)\| \leq \frac{(1+\epsilon)^2\|H^*\|_\infty^2\omega}{2}\|u-v\|, \tag{3.38}$$

for any L^p norm. This inequality carries over the the discrete problem, and in particular, G is locally contractive for $\omega = 0.5$, which is one of the test cases we will consider. It is additionally known that for $\omega < 1$,

$$\rho(G'(H*)) \leq 1-\sqrt{1-\omega} < 1. \tag{3.39}$$

Hence $\|G'(H^*)\| < 1$ for some choice of matrix norm, and thus $G(H)$ is a local contraction in some neighborhood of H^* with respect to that norm. This implies that fixed-point iteration will be locally convergent, and the theory from the previous applies section with this choice norm.

Table 3.1: H-equation iteration statistics for Newton-GMRES and fixed point iteration

	Newton-GMRES			Fixed Point		
ω	0.5	0.99	1.0	0.5	0.99	1.0
Fs	12	18	49	11	75	23970

Discretization of (3.36) by composite midpoint rule with N nodes gives

$$G(u)_i = \left(1 - \frac{\omega}{2N} \sum_{j=1}^{N} \frac{\mu_i u_j}{\mu_i + \mu_j}\right)^{-1}, \ 1 \leq i \leq N, \tag{3.40}$$

where $\mu_i = (i-1/2)/N, 1 \leq i \leq N$. We compare Anderson acceleration with fixed point iteration and Newton-GMRES for this problem. We we consider an $N = 500$ point discretization with $\omega = 0.5, 0.99$, and 1.0. $F'(u^*)$ is singular for $\omega = 1.0$, but both fixed point iteration and Newton-GMRES will converge for our choice of initial iterate. For tests with Anderson, we solve the optimization problem with respect to the l_1, l_2, and l_∞ norm, and vary the storage depth over $m = 1, \ldots, 6$. In each case, we use the initial iterate $u_0 = (1, \ldots, 1)^T$, which is a good initial iterate for $\omega = 0.5$, and a marginal one for the other two. We tabulate the number of function evaluations as a metric of cost. This ignores the cost of the optimization problem within Anderson and the orthogonalization cost within Newton-GMRES, which are non-negligible at this problem size, but should be essentially negligible for larger problems where G is expensive to evaluate. We terminate the iteration on the relative residual reduction $\|F(u_k)\|/\|F(u_0)\| \leq 10^{-8}$.

Results for solving these tests with Newton-GMRES and fixed point iteration as show in Table 3.1. This table tabulates the number of iterations needed for convergence over each value of ω. In both cases the number of function evaluations needed for convergence increases rather sharply as ω increases to 1.0, the case where $F'(u^*)$ is singular. At this value of ω, Newton's method is linearly convergent, and fixed point iteration converges sublinearly.

In Tables 3.2, 3.3, and 3.4, we present results from solving the equations for each value of ω using Anderson acceleration. In these tables, we tabulate the number of function evaluations needed for convergence, the maximum condition number of the coefficient matrix over the history of each iteration, which we call κ_{max}, and the maximum value of $\sum_{i=0}^{m_k} |\alpha_i^{(k)}|$ over the history of each iteration, which we refer to as S_{max}. We note that as m increases, κ_{max} becomes very large, but S_{max} remains reasonably small. This suggests that the assumption in the coefficient sums in Assumption 3.2 is reasonable, at least for reasonably small values of m. In each case, Anderson is competitive with Newton-GMRES in number of function evaluations, but is generally significantly better. With respect to the choice of optimization norm, none of

Table 3.2: H-equation Anderson statistics, $\omega = 0.5$

	l_1 Optimization			l_2 Optimization			l_∞ Optimization		
m	Fs	κ_{max}	S_{max}	Fs	κ_{max}	S_{max}	Fs	κ_{max}	S_{max}
1	7	1.00e+00	1.4	7	1.00e+00	1.4	7	1.00e+00	1.5
2	6	1.40e+03	1.4	6	2.90e+03	1.4	6	2.21e+04	1.4
3	6	7.75e+05	1.4	6	6.19e+05	1.4	6	5.91e+05	1.4
4	7	1.19e+09	1.4	6	9.63e+08	1.4	6	9.61e+08	1.4
5	7	1.90e+13	1.4	6	2.46e+10	1.4	6	2.48e+10	1.4
6	7	2.60e+14	1.4	6	2.46e+10	1.4	6	2.48e+10	1.4

Table 3.3: H-equation Anderson statistics, $\omega = 0.99$

	l_1 Optimization			l_2 Optimization			l_∞ Optimization		
m	Fs	κ_{max}	S_{max}	Fs	κ_{max}	S_{max}	Fs	κ_{max}	S_{max}
1	11	1.00e+00	5.2	11	1.00e+00	4.0	10	1.00e+00	10.8
2	10	1.19e+04	5.2	10	9.81e+03	5.4	10	4.34e+02	5.9
3	10	6.55e+05	5.2	10	2.17e+06	5.4	11	1.13e+06	5.9
4	12	6.92e+09	5.2	11	6.39e+08	5.4	11	8.33e+08	5.9
5	12	1.46e+10	5.2	12	1.64e+11	5.4	12	3.66e+10	5.9
6	14	4.55e+10	7.5	12	1.49e+12	5.4	12	1.05e+11	5.9

the considered norms performed significantly better than the others. Then, as a result of the simplicity and lower cost of solving a linear least-squares problem compared to a linear program, this does not present a compelling reason to further pursue either the l_1 or l_∞ norm for the optimization.

3.3 Preconditioning

The previous section provides local convergence results for this method with mixing parameter equal to one provided that the function G is contractive near the solution. However, a different value of mixing parameter can often result in faster convergence, or might be necessary to obtain convergence at all if G is not contractive near the solution. Even then, choosing an appropriate mixing parameter may not be sufficient if G is not contractive, in which case we must precondition the problem to obtain a solution. To implement preconditioning, we will recall the formulation of Anderson acceleration (3.6), which we can rewrite as

$$u_{k+1} = \sum_{i=0}^{m_k} \alpha_i^{(k)} [u_{k-m_k+i} + F(u_{k-m_k+i})]. \tag{3.41}$$

Table 3.4: H -equation Anderson statistics, $\omega = 1.0$

	l_1 Optimization			l_2 Optimization			l_∞ Optimization		
m	Fs	κ_{max}	S_{max}	Fs	κ_{max}	S_{max}	Fs	κ_{max}	S_{max}
1	21	1.00e+00	3.0	21	1.00e+00	3.0	19	1.00e+00	4.8
2	18	8.99e+04	43.0	16	2.90e+03	14.3	33	4.48e+04	25.3
3	25	8.43e+07	26.3	17	2.99e+06	23.4	50	3.05e+07	182.0
4	22	2.10e+08	12.7	21	6.25e+08	6.6	34	4.92e+08	39.2
5	21	1.30e+09	21.9	27	1.06e+10	14.8	33	1.45e+09	105.8
6	40	3.77e+11	47.2	35	1.44e+11	180.5	33	2.89e+10	18.6

We note that this formulation does not rely on G, and can be applied to solve the equation $F(u) = 0$. Now, we can apply the Anderson acceleration algorithm to the function $F_\beta(u) = \beta F(u)$ rather that $F(u)$. This preconditions F by a constant multiple of the identity, and so long as $\beta \neq 0$, F and F_β have the same solutions. In case, the algorithm gives

$$u_{k+1} = \sum_{i=0}^{m_k} \alpha_i^{(k)} [u_{k-m_k+i} + \beta F(u_{k-m_k+i})]. \tag{3.42}$$

We note that if $F(u) = G(u) - u$, this is the same as (3.7) if β_k is kept at a constant value β. Thus, we can view choosing a mixing parameter different from one as a particular kind of preconditioning. We can employ more general left preconditioning by applying the algorithm to the function $\bar{F}(u) = \beta M(u) F(u)$, where $M(u) \in \mathbb{R}^{N \times N}$ is some preconditioner. In order to not change the solution set, M needs to be nonsingular at all relevant u. Then, applying Anderson to $\bar{F}$, we have

$$u_{k+1} = \sum_{i=0}^{m_k} \alpha_i^{(k)} [u_{k-m_k+i} + \beta M(u_{k-m_k+i}) F(u_{k-m_k+i})], \tag{3.43}$$

where the minimization problem for the coefficients is now in terms of preconditioned residuals

$$\min_{(\alpha_0,\ldots,\alpha_{m_k})} \left\| \sum_{i=0}^{m_k} \alpha_i M(u_{k-m_k+i}) F(u_{k-m_k+i}) \right\|, \quad \text{such that} \quad \sum_{i=0}^{m_k} \alpha_i = 1.$$

Note that this is what results from applying our original formulation of Anderson acceleration to find a fixed point of $\bar{G}(u) = u + \beta M(u) F(u)$. Then, the local convergence results from the previous section will apply if we have chosen mixing parameter and preconditioner such that the function $\bar{G}(u)$ is contractive in a neighborhood of the solution u^*. This will be the case if

$\|\bar{G}'(u^*)\| < 1$. We have $\bar{G}'(u) = I + \beta M(u)F'(u) + \beta T(u)$, where $T(u) \in \mathbb{R}^{N\times N}$ with (i,j) entry

$$T_{ij} = \sum_{k=1}^{N} \frac{\partial}{\partial u_j}(M_{ik}(u))F_k(u).$$

Each $F_k(u^*) = 0$, so $T(u^*) = 0$, and thus $\hat{G}'(u^*) = I + \beta M(u^*)F'(u^*)$. This provides some insight as to how to go about selecting the mixing parameter and preconditioner. We would like

$$\|I - [-\beta M(u^*)]F'(u^*)\| < 1,$$

so we must choose the mixing parameter and preconditioner such that $-\beta M(u^*)$ is an approximate inverse of $F'(u^*)$. We may simply let $M(u)$ to be an approximate inverse of $-F'(u)$ letting $\beta = 1$, or let $M(u)$ to be an approximate inverse of $F'(u)$ and choose $\beta = -1$. As an extreme, we may select $M(u) = F'(u)^{-1}$. In this case each iteration will require essentially the same number of operations as Newton's method while carrying an additional storage burden.

We furthermore note that in the case where we consider mixing parameters, but no other preconditioning, to guarantee local convergence, we need β such that $G_\beta(u) = u + \beta F(u)$ is contractive near u^*. As above, this will be the case if $\|G'_\beta(u^*)\| = \|I + \beta F'(u^*)\| < 1$, which may be achievable depending on the clustering of the eigenvalues of $F'(u^*)$.

3.4 Adjusting Storage Depth for Conditioning

In this section, we consider a variation of the Anderson acceleration algorithm in which the storage depth is adjusted in order to maintain good conditioning of the least-squares problem. This alteration to the method was suggested in [62] purely as a heuristic consideration in order to ensure sufficient accuracy in the solution of the least squares problem. This variation proceeds as shown in Algorithm 5. In this, $\mathcal{F}_k$ and $\mathcal{U}_k$ contain the m_k most recent differences between consecutive residuals and iterates respectively, and m_k simply acts as a counter for the number of vectors in each matrix. The management of these matrices is similar to the standard algorithm in that new vectors are appended and the oldest discarded if the storage threshold is exceeded, but now we also require that $\kappa(\mathcal{F}_k) \leq \tau$, for some tolerance τ, and discard the oldest vectors until this is satisfied. We note that the matrices will always contain at least one vector each (so long as $\Delta F_k \neq 0$ at any point in the iteration), because $\kappa(\mathcal{F}_k) = 1$ in the case where the current storage depth is reduced to one. Hence, we see that if the tolerance τ is set fairly small, the algorithm will likely reduce to Anderson-1.

While this variation was originally presented as a practical consideration without analysis, we have shown that this change allows us to show convergence of the method without utilizing the coefficient sum bound in Assumption 3.2. For the following result, we additionally illustrate

Algorithm 5 Anderson acceleration with adjusted storage depth

Given initial iterate u_0, mixing parameter β, storage depth parameter m, and condition number bound τ
Set $u_1 = u_0 + \beta F_0$
Initialize $m_k = 0, \mathcal{F}_0 = [], \mathcal{U}_0 = []$
for k = 1,2,... **do**
    Set $\mathcal{F}_k = [\mathcal{F}_{k-1}, \Delta F_k], \mathcal{U}_k = [\mathcal{U}_{k-1}, \Delta u_k], m_k = m_k + 1$
    if $m_k > m$ **then**
        Set $\mathcal{F}_k = [\mathcal{F}_k(2 : m_k, :)], \mathcal{U}_k = [\mathcal{U}_k(2 : m_k, :)], m_k = m_k - 1$
    end if
    while $\kappa(\mathcal{F}_k) > \tau$ **do**
        Set $\mathcal{F}_k = [\mathcal{F}_k(2 : m_k, :)], \mathcal{U}_k = [\mathcal{U}_k(2 : m_k, :)], m_k = m_k - 1$
    end while
    Determine $\gamma^{(k)}$ which solves

$$\min_{\gamma=(\gamma_0,\dots\gamma_{m_k-1})^T} \|F_k - \mathcal{F}_k\gamma\| \tag{3.44}$$

    Set

$$u_{k+1} = u_k + \beta F_k - (\mathcal{U}_k + \beta\mathcal{F}_k)\gamma^{(k)} \tag{3.45}$$

end for

how arbitrary mixing parameters affect the analysis. In this case, we require the following assumption, which is a slight variation of Assumption 3.1.

Assumption 3.3.

- *There exists* u^* *such that* $F(u^*) = G(u^*) - u^* = 0$
- *There is* $\hat{\rho} > 0$ *such that* $F'(u)$ *is Lipschitz continuous with constant* γ *in* $\mathcal{B}_{\hat{\rho}}(u^*) = \{u : \|u - u^*\| \leq \hat{\rho}\}$
- *For all* $u, v \in \mathcal{B}_{\hat{\rho}}(u^*)$*, there is* $c \in (0, 1)$ *such that* $\|G_\beta(u) - G_\beta(v)\| \leq c\|u - v\|$

A minor difference for this case is that the earlier analysis permitted use of arbitrary norm for solving the least-squares problem and tracking convergence. In this analysis, we assume that the norm is induced by an inner product. This is necessary for the use of orthogonal projectors in the proof of the theorem. As the l_2 norm is generally the preferred norm, this should not be a controversial assumption.

Before stating the theorem, we will first establish some consequences from the above set of assumptions. First note that the last point implies

$$c \geq \|G'_\beta(u)\| = \|(1 - \beta)I + \beta G'(u)\| = \|I + \beta F'(u)\|,$$

for $u \in \mathcal{B}_{\hat{\rho}}(u^*)$. By the Banach Lemma, this implies that $F'_\beta(u) = -(I - G'_\beta(u))$ is nonsingular. We also have $F'_\beta(u) = \beta F'(u)$, which implies that $F'(u)$ is likewise nonsingular as $F'(u)^{-1} = (\frac{1}{\beta}F'_\beta(u))^{-1} = \beta F'_\beta(u)^{-1}$. From this, we have the following bounds for each $u \in \mathcal{B}_{\hat{\rho}}(u^*)$

$$\|F'(u)\| = \|\frac{1}{\beta}F'_\beta(u)\| = \frac{1}{|\beta|}\|I - G'_\beta(u)\| \leq \frac{1+c}{|\beta|}, \tag{3.46}$$

and

$$\|F'(u)^{-1}\| = |\beta|\|F'_\beta(u)^{-1}\| \leq |\beta|\frac{1}{1 - \|G'_\beta(u)\|} \leq \frac{|\beta|}{1-c}. \tag{3.47}$$

Next, consider $u, v \in \mathcal{B}_{\hat{\rho}}(u^*)$. In this case, we have

$$\begin{aligned}\|F(u) - F(v)\| = \frac{1}{|\beta|}\|F_\beta(u) - F_\beta(v)\| = \frac{1}{|\beta|}\|G_\beta(u) - G_\beta(v) - (u-v)\| \\ \leq \frac{1}{|\beta|}(\|G_\beta(u) - G_\beta(v)\| + \|u - v\|) \leq \frac{1+c}{|\beta|}\|u - v\|,\end{aligned} \tag{3.48}$$

and

$$\begin{aligned}\|u - v\| = \|G_\beta(u) - \beta F(u) - (G_\beta(v) - \beta F(v))\| =\leq \|G_\beta(u) - G_\beta(v)\| + |\beta|\|F(u) - F(v)\| \\ \leq c\|u - v\| + |\beta|\|F(u) - F(v)\| \Rightarrow \|u - v\| \leq \frac{|\beta|}{1-c}\|F(u) - F(v)\|.\end{aligned} \tag{3.49}$$

In particular, letting $v = u^*$ in the above inequalities results in the following (as $F(u^*) = 0$)

$$\frac{1-c}{|\beta|}\|e\| \leq \|F(u)\| \leq \frac{1+c}{|\beta|}\|e\|. \tag{3.50}$$

From all of the above, we show the following theorem:

Theorem 3.3. *Suppose that Assumption 3.3 holds, and let $\hat{c} \in (c, 1)$. Then, there is $\rho \leq \hat{\rho}$ such that for $\|e_0\|$ sufficiently small, the iteration described by Algorithm 5 remains in $\mathcal{B}_\rho(u^*)$, the fixed-point residuals converge q-linearly to zero with q-factor at most $\hat{c}$, i.e.*

$$\|F_{k+1}\| \leq \hat{c}\|F_k\|, \tag{3.51}$$

and the errors converge r-linearly to zero and satisfy the bound

$$\|e_{k+1}\| \leq \hat{c}^{k+1}\frac{1+c}{1-c}\|e_0\|. \tag{3.52}$$

Moreover,

$$\limsup \frac{\|F_{k+1}\|}{\|F_k\|} \leq c. \tag{3.53}$$

Proof. First, we let ρ be small enough such that it holds that

$$\left(1+\frac{3\gamma\rho|\beta|}{2(1-c)}\right)\left(\frac{\gamma\rho|\beta|}{2(1-c)}+c+\frac{2\gamma\tau\rho m|\beta|}{1-c}\right)\leq\hat{c}. \tag{3.54}$$

We note that the left hand side above approaches c as ρ decreases to zero, and is monotone increasing for $\rho\geq 0$, so this inequality is necessarily attainable for small enough ρ, as $\hat{c}>c$. We then let $\|e_0\|\leq\rho$ small enough so that

$$\|e_0\|\leq\frac{\rho(1-c)^2}{|\beta|(1+c)\left(\frac{\gamma\rho}{2}+\frac{c(1+c)}{|\beta|}+\frac{2\gamma\rho\tau(1+c)}{1-c}\right)}. \tag{3.55}$$

Now, we will prove (3.51) and (3.52) by induction. The induction hypothesis holds for $k=0$, because the first iterate is a step of fixed-point iteration with the fixed-point map G_β, that is $u_1=G_\beta(u_0)$, and we assume G_β to be contractive with constant c. Thus

$$\|e_1\|=\|G_\beta(u_0)-G_\beta(u^*)\|\leq c\|e_0\|\leq\hat{c}\|e_0\|,$$

and

$$\begin{aligned}\|F_1\|=\|\frac{1}{\beta}F_\beta(u_1)\|=\frac{1}{|\beta|}\|G_\beta(u_1)-G_\beta(u_0)\|\leq\frac{c}{|\beta|}\|u_1-u_0\|\\ =\frac{c}{|\beta|}\|G_\beta(u_0)-u_0\|=\frac{c}{|\beta|}\|F_\beta(u_1)\|=\frac{c}{|\beta|}\|\beta F_0\|=c\|F_0\|\leq\hat{c}\|F_0\|.\end{aligned}$$

We then assume that the induction hypothesis holds for each $0\leq i\leq k$ for some $k\geq 0$. That is, we assume that $u_i\in\mathcal{B}_\rho(u^*)$, and (3.51) and (3.52) hold for $0\leq i\leq k$ for $k\geq 0$. We will first express the new iterate (3.45) in a different form. We note that because $\gamma^{(k)}$ solves (3.44), $\mathcal{F}_k\gamma^{(k)}=P_kF_k$, where P_k is the orthogonal projector onto the column space of $\mathcal{F}_k$, and similarly $F_k-\mathcal{F}_k\gamma^{(k)}=(I-P_k)F_k$, where $I-P_k$ is the projector onto the orthogonal complement. The projector P_k has rank at most m_k, and $I-P_k$ likewise has rank at least $N-m_k$. With this, we can write

$$u_{k+1}=u_k-\mathcal{U}_k\gamma^{(k)}+\beta(F_k-\mathcal{F}_k\gamma^{(k)})=u_k-\mathcal{U}_k\gamma^{(k)}+\beta(I-P_k)F_k. \tag{3.56}$$

Now, for $k-m_k+1\leq i\leq k$ we have

$$\Delta F_i=\int_0^1 F'(u_{i-1}+t\Delta u_i)\Delta u_i\,\mathrm{d}t=F'(u_k)\Delta u_i+\int_0^1[F'(u_{i-1}+t\Delta u_i)-F'(u_k)]\Delta u_i\,\mathrm{d}t. \tag{3.57}$$

From this, we get

$$\Delta u_i = F'(u_k)^{-1}\Delta F_i - F'(u_k)^{-1}E_{k,i}, \tag{3.58}$$

where we define

$$E_{k,i} \equiv \int_0^1 [F'(u_{i-1} + t\Delta u_i) - F'(u_k)]\Delta u_i \, \mathrm{d}t.$$

Given the Assumption 3.3, and from the assumption that $\|e_i\| \leq \rho$ for each previous iterate, we bound each of these error terms as follows

$$\begin{aligned}
\|E_{k,i}\| &\leq \int_0^1 \gamma \|u_{i-1} + t\Delta u_i - u_k\| \, \mathrm{d}t \|\Delta u_i\| \\
&= \int_0^1 \gamma \|te_i + (1-t)e_{i-1} - e_k\| \, \mathrm{d}t \|\Delta u_i\| \\
&\leq \int_0^1 \gamma (t\|e_i\| + (1-t)\|e_{i-1}\| + \|e_k\|) \, \mathrm{d}t \|\Delta u_i\| \\
&= \gamma \left(\frac{\|e_i\|}{2} + \frac{\|e_{i-1}\|}{2} + \|e_k\| \right) \|\Delta u_i\| \\
&\leq 2\gamma\rho\|\Delta u_i\|.
\end{aligned} \tag{3.59}$$

From (3.58), we can write

$$\mathcal{U}_k = F'(u_k)^{-1}\mathcal{F}_k - F'(u_k)^{-1}E_k, \tag{3.60}$$

where $E_k = (E_{k,k-m_k+1}, \ldots, E_{k,k})$. Substituting this into (3.56), and recalling that $\mathcal{F}_k\gamma^{(k)} = P_kF_k$, this gives

$$\begin{aligned}
u_{k+1} &= u_k - [F'(u_k)^{-1}\mathcal{F}_k - F'(u_k)^{-1}E_k]\gamma^{(k)} + \beta(I - P_k)F_k \\
&= u_k - F'(u_k)^{-1}P_kF_k + \beta(I - P_k)F_k + F'(u_k)^{-1}E_k\gamma^{(k)} \\
&= u_k - F'(u_k)^{-1}F_k + F'(u_k)^{-1}[(I + \beta F'(u_k))(I - P_k)F_k + E_k\gamma^{(k)}].
\end{aligned} \tag{3.61}$$

We see that the direction differs from a Newton direction by the orthogonal projection term and the higher order expansion terms in E_k. This results in the following bound:

$$\begin{aligned}
\|e_{k+1}\| &\leq \|F'(u_k)^{-1}\| \left(\|F'(u_k) - F_k\| + \|I + \beta F'(u_k)\| \|F_k\| + \|E_k\gamma^{(k)}\| \right) \\
&\leq \frac{|\beta|}{1-c} \left(\frac{\gamma}{2}\|e_k\|^2 + c\|F_k\| + \|E_k\gamma^{(k)}\| \right).
\end{aligned} \tag{3.62}$$

Now, because the algorithm adjusts the storage depth to ensure good conditioning of $\mathcal{F}_k = [\Delta F_{k-m_k+1}, \ldots, \Delta F_k]$, we know that $\mathcal{F}_k$ has full column rank and the least squares problem (3.44) has the solution $\gamma^{(k)} = (\mathcal{F}_k^T\mathcal{F}_k)^{-1}\mathcal{F}_k^TF_k$. The expression $(\mathcal{F}_k^T\mathcal{F}_k)^{-1}\mathcal{F}_k^T$ represents the

pseudoinverse of $\mathcal{F}_k$, so from the bound on $\kappa(\mathcal{F}_k)$, we have

$$\begin{aligned}\tau &\geq \kappa(\mathcal{F}_k)\\ &= \|\mathcal{F}_k\|\|(\mathcal{F}_k^T\mathcal{F}_k)^{-1}\mathcal{F}_k^T\|\\ &\geq \|\mathcal{F}_k v_i\|\|(\mathcal{F}_k^T\mathcal{F}_k)^{-1}\mathcal{F}_k^T\|\\ &= \|\Delta F_{k-m_k+i}\|\|(\mathcal{F}_k^T\mathcal{F}_k)^{-1}\mathcal{F}_k^T\|,\end{aligned}$$

where v_i is the i^{th} canonical vector. Thus, for $1 \leq i \leq m_k$ we have

$$\|(\mathcal{F}_k^T\mathcal{F}_k)^{-1}\mathcal{F}_k^T\| \leq \|\Delta F_{k-m_k+i}\|^{-1}\tau. \tag{3.63}$$

Then, using this and (3.59), the following holds

$$\begin{aligned}\|E_k\gamma^{(k)}\| &\leq \|E_k\|\|(\mathcal{F}_k^T\mathcal{F}_k)^{-1}\mathcal{F}_k^T\|\|F_k\|\\ &\leq \sum_{i=1}^{m_k}(\|E_{k,k-m_k+i}\|\|(\mathcal{F}_k^T\mathcal{F}_k)^{-1}\mathcal{F}_k^T\|)\|F_k\|\\ &\leq \sum_{i=1}^{m_k}(\|E_{k,k-m_k+i}\|\|\Delta F_{k-m_k+i}\|^{-1}\tau)\|F_k\|\\ &\leq \sum_{i=1}^{m_k}\left(2\gamma\rho\frac{\|\Delta u_{k-m_k+i}\|}{\|\Delta F_{k-m_k+i}\|}\tau\right)\|F_k\|\\ &= 2\gamma\rho\tau\|F_k\|\sum_{i=1}^{m_k}\frac{\|\Delta u_{k-m_k+i}\|}{\|\Delta F_{k-m_k+i}\|}.\end{aligned}$$

Thus, we have

$$\|E_k\gamma^{(k)}\| \leq \frac{2\gamma\rho\tau|\beta|m}{1-c}\|F_k\|.$$

Substituting this into (3.62) and using the inductive hypothesis gives

$$\begin{aligned}\|e_{k+1}\| &\leq \frac{|\beta|}{1-c}\left(\frac{\gamma}{2}\|e_k\|^2 + c\|F_k\| + \frac{2\gamma\rho\tau|\beta|m}{1-c}\|F_k\|\right)\\ &\leq \frac{|\beta|}{1-c}\left(\frac{\gamma\rho}{2} + \frac{c(1+c)}{|\beta|} + \frac{2\gamma\rho\tau m(1+c)}{1-c}\right)\|e_k\|\\ &\leq \frac{|\beta|(1+c)}{(1-c)^2}\left(\frac{\gamma\rho}{2} + \frac{c(1+c)}{|\beta|} + \frac{2\gamma\rho\tau m(1+c)}{1-c}\right)\|e_0\|.\end{aligned}$$

Then, by the assumption (3.55), we assume that $\|e_0\|$ is small enough so that $\|e_{k+1}\| \leq \rho$, so $u_{k+1} \in \mathcal{B}_\rho(u^*)$. Then, we can use the fundamental theorem of calculus and the expression for

the new iterate (3.61) to rewrite the new residual as

$$\begin{aligned} F_{k+1} &= \int_0^1 F'(u^* + te_{k+1})e_{k+1}\,\mathrm{d}t \\ &= \int_0^1 F'(u^* + te_{k+1})F'(u_k)^{-1}\,\mathrm{d}t \left(F'(u_k)e_k - F_k + (I + \beta F'(u_k))(I - P_k)F_k + E_k\gamma^{(k)}\right). \end{aligned}$$

Hence

$$\|F_{k+1}\| \le \int_0^1 \|F'(u^* + te_{k+1})F'(u_k)^{-1}\|\,\mathrm{d}t \left(\|F'(u_k)e_k - F_k\| + \|I + \beta F'(u_k)\|\|F_k\| + \|E_k\gamma^{(k)}\|\right). \quad (3.64)$$

For any $t \in [0, 1]$, we can write

$$F'(u^*+te_{k+1})F'(u_k)^{-1} = I-(I-F'(u^*+te_{k+1})F'(u_k)^{-1}) = I-(F'(u_k)-F'(u^*+te_{k+1}))F'(u_k)^{-1}.$$

Hence, by Lipschitz continuity of F' in $\mathcal{B}_\rho(u^*)$, we have

$$\begin{aligned} \int_0^1 \|F'(u^* + te_{k+1})F'(u_k)^{-1}\|\,\mathrm{d}t &\le \int_0^1 1 + \|F'(u_k) - F'(u^* + te_{k+1})\|\|F'(u_k)^{-1}\|\,\mathrm{d}t \\ &\le \int_0^1 1 + \gamma\|e_k - te_{k+1}\|\|F'(u_k)^{-1}\|\,\mathrm{d}t \\ &\le 1 + \frac{\gamma|\beta|}{1-c}\left(\|e_k\| + \frac{\|e_{k+1}\|}{2}\right) \\ &\le 1 + \frac{3\gamma\rho|\beta|}{2(1-c)}. \end{aligned} \quad (3.65)$$

Now, Assumption 3.3, (3.49) and (3.63) imply the following bounds on the remaining parts of (3.64)

$$\|F'(u_k)e_k - F_k\| \le \frac{\gamma\|e_k\|^2}{2} \le \frac{\gamma|\beta|\|e_k\|}{2(1-c)}\|F_k\|, \quad (3.66)$$

$$\|I + \beta F'(u_k)\|\|F_k\| \le c\|F_k \quad (3.67)$$

$$\begin{aligned} \|E_k\gamma^{(k)}\| &\le \sum_{i=1}^{m_k} \|E_{k,k-m_k+i}\|\|(\mathcal{F}_k^T\mathcal{F}_k)^{-1}\mathcal{F}_k^T\|\|F_k\| \\ &\le \left(\gamma\tau\sum_{i=1}^{m_k}\left(\frac{\|e_{k+m_k+i-1}\| + \|e_{k-m_k+i}\|}{2} + \|e_k\|\right)\frac{\|\Delta u_{k-m_k+i}\|}{\|\Delta F_{k-m_k+i}\|}\right)\|F_k\| \\ &\le \left(\frac{\gamma\tau|\beta|}{1-c}\sum_{i=1}^{m_k}\left(\frac{\|e_{k+m_k+i-1}\| + \|e_{k-m_k+i}\|}{2} + \|e_k\|\right)\right)\|F_k\|. \end{aligned} \quad (3.68)$$

Substituting these into (3.64) gives

$$\frac{\|F_{k+1}\|}{\|F_k\|} \leq \left(1 + \frac{\gamma|\beta|}{1-c}\left(\|e_k\| + \frac{\|e_{k+1}\|}{2}\right)\right) \left(\frac{\gamma|\beta|\|e_k\|}{2(1-c)} + c + \frac{\gamma\tau|\beta|}{1-c}\sum_{i=1}^{m_k}\left(\frac{\|e_{k+m_k+i-1}\| + \|e_{k-m_k+i}\|}{2} + \|e_k\|\right)\right). \quad (3.69)$$

From this, we get

$$\frac{\|F_{k+1}\|}{\|F_k\|} \leq \left(1 + \frac{3\gamma\rho|\beta|}{2(1-c)}\right)\left(\frac{\gamma\rho|\beta|}{2(1-c)} + c + \frac{2\gamma\tau\rho m|\beta|}{1-c}\right) \leq \hat{c}, \quad (3.70)$$

where the final inequality is simply the assumption (3.54). Hence, we have $\|F_{k+1}\| \leq \hat{c}\|F_k\|$, and the bound (3.59) follows directly from the q-linear convergence in the residuals. We have

$$\|e_{k+1}\| \leq \frac{|\beta|}{1-c}\|F_{k+1}\| \leq \hat{c}^{k+1}\frac{|\beta|}{1-c}\|F_0\| \leq \hat{c}^{k+1}\frac{1+c}{1-c}\|e_0\|. \quad (3.71)$$

Hence, we see that $\|e_k\|$ converges to zero r-linearly. This fact and (3.69) imply the last claim. Because the errors go to zero, as $k \to \infty$ we have

$$1 + \frac{\gamma|\beta|}{1-c}\left(\|e_k\| + \frac{\|e_{k+1}\|}{2}\right) \to 1,$$

and

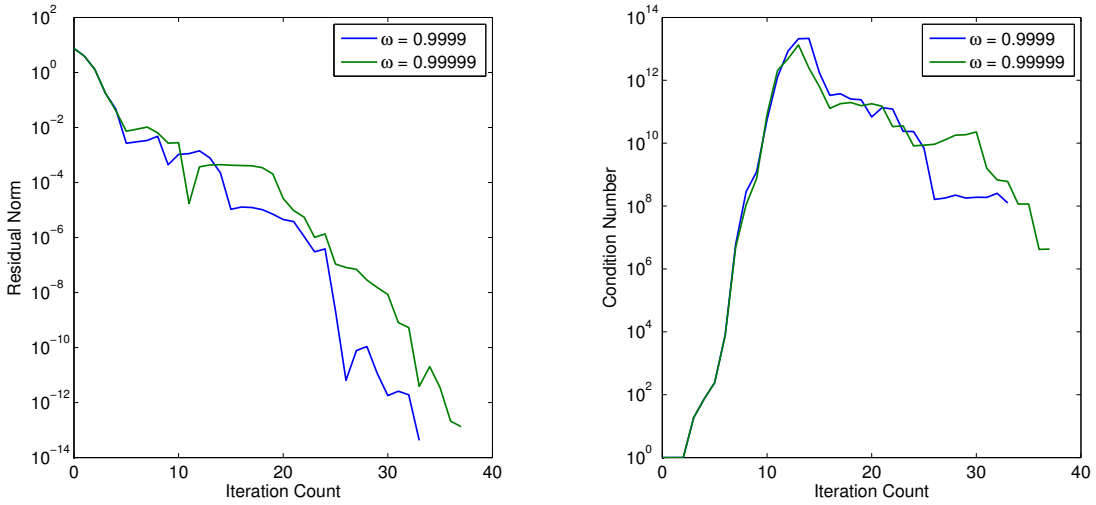
$$\frac{\gamma|\beta|\|e_k\|}{2(1-c)} + c + \frac{\gamma\tau|\beta|}{1-c}\sum_{i=1}^{m_k}\left(\frac{\|e_{k+m_k+i-1}\| + \|e_{k-m_k+i}\|}{2} + \|e_k\|\right) \to c.$$

Hence, from (3.69) it in fact follows that

$$\limsup \frac{\|F_{k+1}\|}{\|F_k\|} \leq c.$$

□

We will note here that the condition on the initial error imposed by equations (3.54) and (3.55) may require the initial error to be very small in order to display the monotonic decrease in the residual predicted by this theorem, especially in cases where the fixed-point map is weakly contractive (i.e. c is very near to 1) or a moderately large condition number bound τ is chosen. However, without such a good initial iterate the r-linear local convergence results from Section 3.2 should still hold, and the results from this section should apply once the iteration has reached a point where each of the stored iterates is within $\mathcal{B}_\rho(u^*)$. That is, without a sufficiently good initial iterate we may expect some non-monotonic reduction in the fixed-point residual



(a) Residual norms

(b) Coefficient matrix condition number, $\kappa(\mathcal{F}_k)$

Figure 3.1: Solving H-equation with Anderson-10 for various ω

early in the iteration, but at some point the residual norms should display strict monotonic decrease as predicted by this theorem.

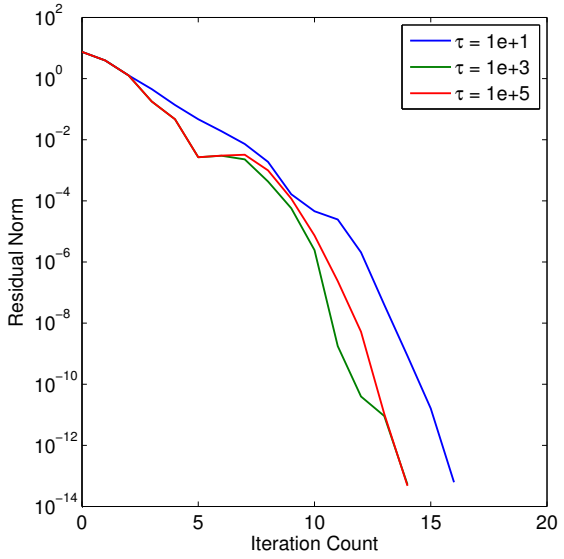
3.4.1 Numerical Tests

As a numerical example of this storage adjusted algorithm, we again consider the Chandrasekhar H-equation introduced in Section 3.2.3. We consider tests with $N = 400, m = 10$, and both $\omega = 0.9999$ and $\omega = 0.99999$. For both of these values of ω the spectral radius of $G'(u^*)$ is very near unity. First, in Figure 3.1, we see plots of the residual norm and coefficient matrix condition number resulting from solving this equation with the two values of ω by Anderson-10 without adjusting the storage depth for conditioning. In the residual plot, we observe that the convergence is non-monotonic, as the previous theory predicts. In fact, in both cases there are points where the residual norm increases by over an order of magnitude from iteration to iteration. Additionally, we observe that the condition number of the coefficient matrix rapidly grows very large in both cases. We note that utilizing the storage adjusted algorithm with a condition number bound $\tau \geq 10^{14}$ would produce the same iteration, at least for the portion of the iteration shown here. This does not violate the theory from this section, however, because allowing such a large condition number bound on such weakly contractive fixed-point maps would require an incredibly good initial iterate in order to satisfy the conditions of the theorem.

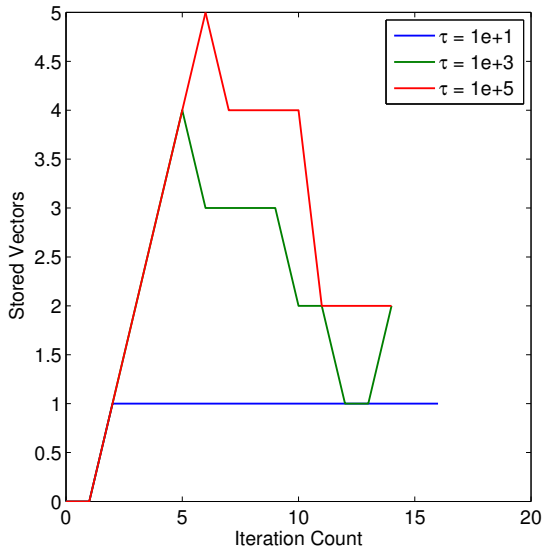
In Figure 3.2, we see the residual histories and storage utilization from solving the same problems with coefficient matrix condition bounds $\tau = 10, 10^3$, and 10^5. In each case, we set

$m = 10$, so the storage depth will be 10 at a maximum. However, in each case the condition number bound prevents the maximum depth from being attained. For both problems, the bound $\tau = 10$ results in taking Anderson-1 steps each iteration, resulting in monotonically decreasing residual histories. $\tau = 10^3$ and $\tau = 10^5$ allow more of the allocated storage to be utilized, but the storage depth utilized never rises above 5. While the residual histories for these larger bounds display some non-monotonicity, they are noticeably smoother than those observed in Figure 3.1, and the rate of convergence is similar to that with $\tau = 10$. In every case, the number of iterations to convergence when adjusting for conditioning is roughly half of that for the unbounded case. It seems that $m = 10$ is a poor choice for this problem, and adjusting to maintain a reasonable condition number can act as a safeguard against such a poor choice of storage depth. However, it may be possible that setting too small a bound may restrict the number of stored vectors from expanding when it would be advantageous, so this may also sacrifice performance.

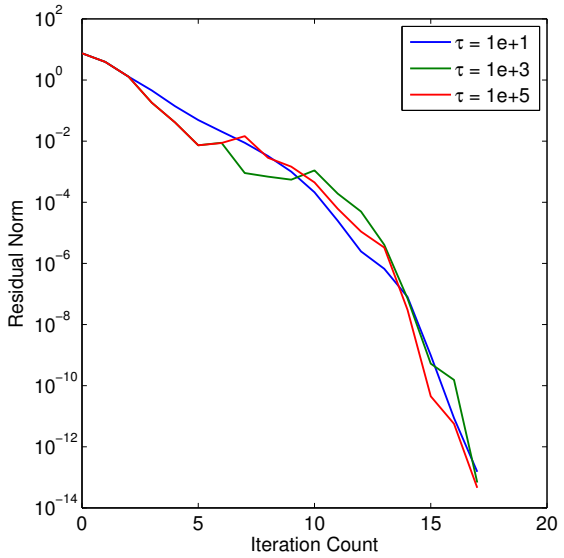
The non-monotonic convergence observed in Figure 3.2 for the larger bounds would seem to indicate that the initial iterate used for these tests was not good enough to satisfy the theory we have derived. To verify that q-linear convergence is indeed obtained with better initial data, the same tests were run with improved initial iterates obtained by can simply performing some number of Picard iterations, and using the result from that as the initial iterate for Anderson acceleration. In Figures 3.3 and 3.4 we see Anderson acceleration residual histories for which the start of the acceleration is delayed by a number of Picard iterations chosen to reduce the initial residual (and thus roughly the initial error) by a factor $\delta = 10^2, 10^4$, and 10^6. For $\omega = 0.9999$, this meant delaying the start of the Anderson acceleration until iterations 22, 150, and 347 respectively. For $\omega = 0.99999$, the start of the Anderson acceleration is delayed until iterations 22, 215, and 784 respectively to achieve the desired reduction. For $\omega = 0.9999$, each initial iterate seems sufficiently good for $\tau = 10$. However, we need the initial iterate improved by a factor of 10^4 before monotonic convergence for $\tau = 10^3$, and 10^6 for $\tau = 10^5$. For $\omega = 0.99999$, non-monotonic convergence is observed for each choice of τ with only a factor of 10^2 improvement in the initial iterate. With a factor of 10^4 improvement in the initial residual, we obtain monotonic convergence for $\tau = 10$ and $\tau = 10^3$. In all cases, with initial iterate improved by a factor of 10^6, we observe very smooth q-linear convergence in the residual plots, so it seems that this initial iterate is sufficiently good for even the largest condition number bound we consider. While this is interesting for observing theory, this approach is not practical as the cost of Picard iterations required to obtain a good enough initial iterate to satisfy the theory was significantly higher than simply using Anderson acceleration from the beginning and accepting non-monotonic convergence.



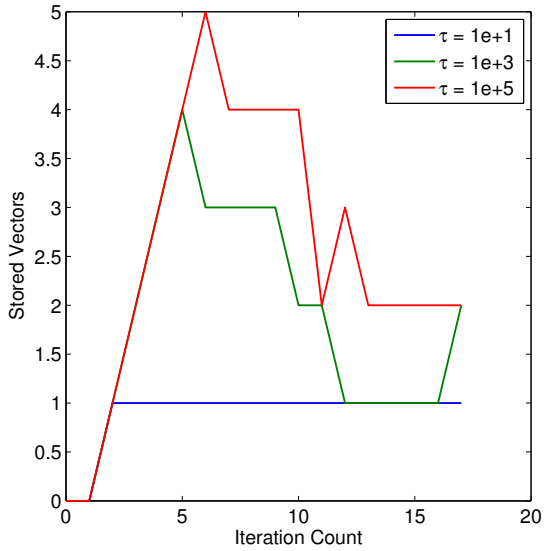
(a) Residual histories with $\omega = 0.9999$



(b) Storage utilization with $\omega = 0.9999$

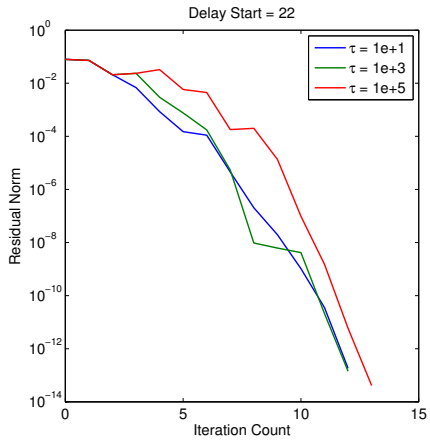


(c) Residual histories with $\omega = 0.99999$

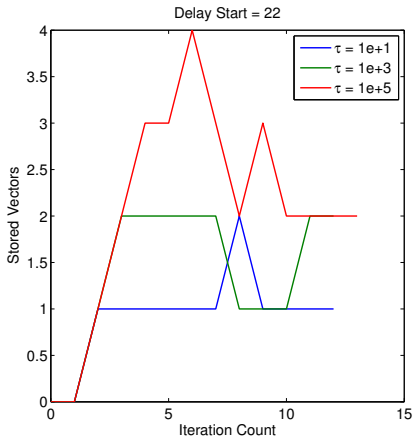


(d) Storage utilization with $\omega = 0.99999$

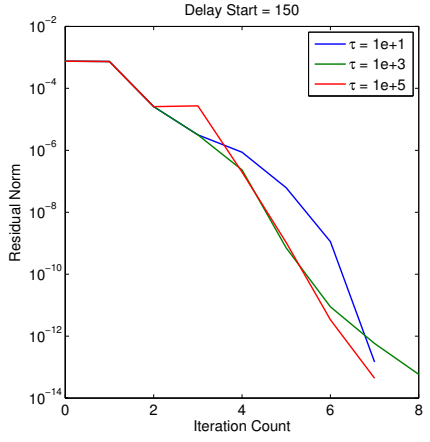
Figure 3.2: Solving H-equation by Algorithm 5 with $m = 10$ for various ω and condition number bound τ



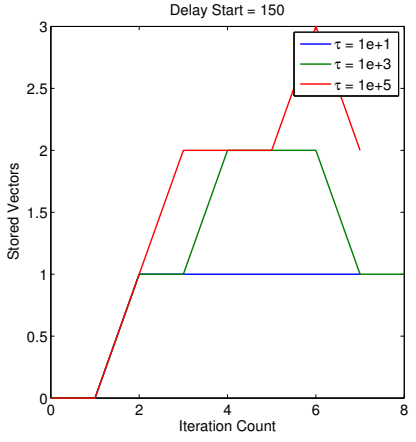
(a) Residual histories, $\delta = 10^2$



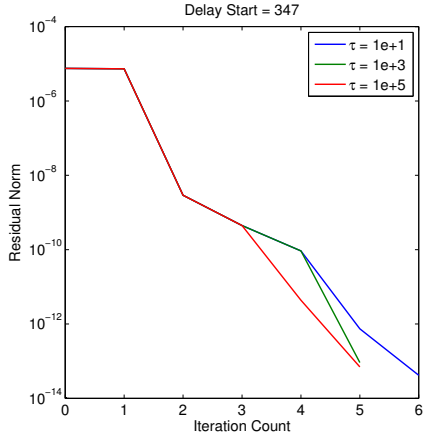
(b) Storage utilization, $\delta = 10^2$



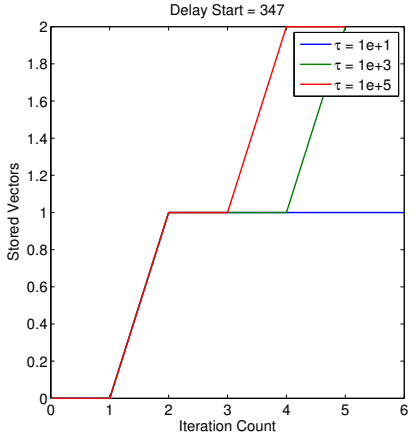
(c) Residual histories, $\delta = 10^4$



(d) Storage utilization, $\delta = 10^4$

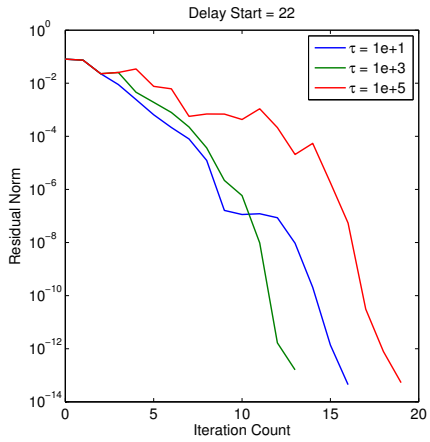


(e) Residual histories, $\delta = 10^6$

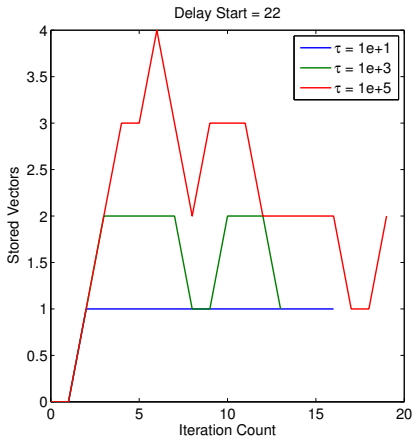


(f) Storage utilization, $\delta = 10^6$

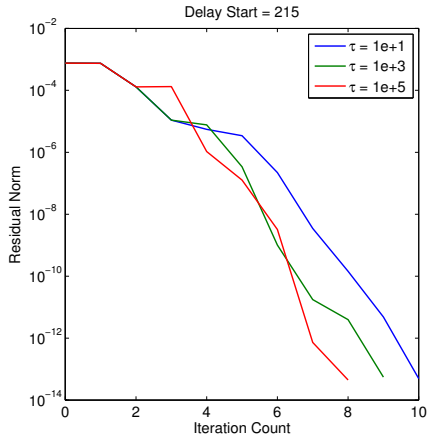
Figure 3.3: Solving H-equation with $\omega = 0.9999$ and initial residual norm reduced by a factor δ from the base case



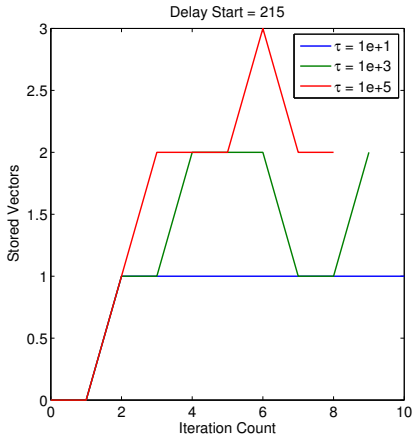
(a) Residual histories, $\delta = 10^2$



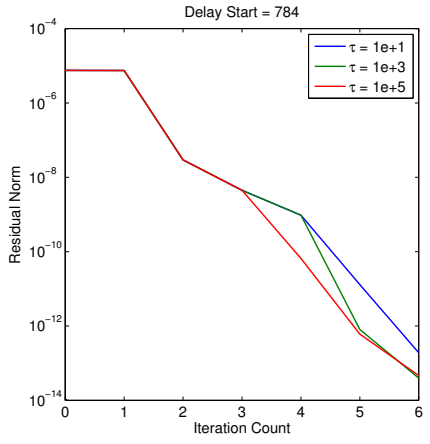
(b) Storage utilization, $\delta = 10^2$



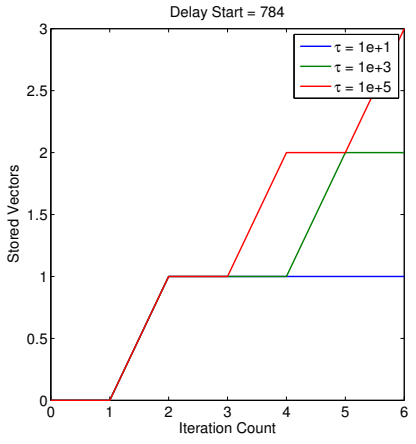
(c) Residual histories, $\delta = 10^4$



(d) Storage utilization, $\delta = 10^4$



(e) Residual histories, $\delta = 10^6$



(f) Storage utilization, $\delta = 10^6$

Figure 3.4: Solving H-equation with $\omega = 0.99999$ and initial residual norm reduced by a factor δ from the base case

3.5 Deterministic Errors in the Function Evaluation

We now consider the Anderson acceleration algorithm with inaccuracies in the function evaluation. That is, we attempt to solve $G(u) = u$, but we only have the capability to evaluate $\hat{G}(u) = G(u) + \epsilon(u)$. This gives the same error in the fixed-point residual $\hat{F}(u) = \hat{G}(u) - u = F(u) + \epsilon(u)$, where $F(u) = G(u) - u$ is the true residual. This ϵ term could represent things like rounding error, an inaccurate matrix-vector product for linear problems, or more pertinent to our work, the effect of internal tolerances from single-physics solves in multiphysics coupling problems or possibly some sort of Monte Carlo simulation involved in the function evaluation. The Anderson algorithm with errors in the function evaluation proceeds as follows.

Algorithm 6 Anderson acceleration with inexact function evaluations

1: Given initial iterate u_0 and storage depth parameter m.
2: Set $u_1 = \hat{G}(u_0)$.
3: Set $\hat{F}_0 = \hat{G}(u_0) - u_0$.
4: **for** k = 1,2,... **do**
5: Set $m_k = \min\{m, k\}$.
6: Set $\hat{F}_k = \hat{G}(u_k) - u_k$.
7: Determine $\alpha^{(k)} = (\alpha_0^{(k)}, \ldots, \alpha_{m_k}^{(k)})$ which solves

$$\min_{\alpha=(\alpha_0,\ldots\alpha_{m_k})^T} \left\| \sum_{i=0}^{m_k} \alpha_i \hat{F}_{k-m_k+i} \right\|. \tag{3.72}$$

8: Set

$$u_{k+1} = \sum_{i=0}^{m_k} \alpha_i^{(k)} \hat{G}_{k-m_k+i}. \tag{3.73}$$

9: **end for**

The only difference from the error-free case is that the optimization problem is solved over linear combinations of the perturbed residuals, and the new iterate is a linear combination of perturbed function evaluations. We will first consider the case where the errors are deterministic and bounded is norm, i.e. $\|\epsilon(u)\| \leq \epsilon_0$ for some ϵ_0. We will then show how to adapt these results to the case where there the function errors are stochastic.

3.5.1 Linear Problems

We first let G be a linear function, i.e. $G(u) = Au + b$ where $A \in \mathbb{R}^{N\times N}$ and $b \in \mathbb{R}^N$. As a result of the equivalence of Anderson acceleration to GMRES iteration for linear problems, any

analysis for GMRES with inaccuracies in the matrix-vector product should provide some insight into the behavior of Anderson acceleration in this context. The effect of inaccurate matrix-vector products in the context of Krylov methods has been studied previously [53–55]. The results in these papers for GMRES suggest that errors in early matrix-vector products may accumulate and lead to significant loss of accuracy in the solution returned by the solver, and this effect may be particularly problematic if the problem requires many iterations. In this section, we adapt results from the previous sections to the case where only an inaccurate function evaluation may be obtained, and show that in the context we consider the effect of inaccurate function evaluations may not be as pessimistic as the result for GMRES might suggest, so long as the fixed-point map is not very weakly contractive..

Before presenting our analysis for Anderson, for the sake of comparison we will establish some results for Picard iteration with the inaccurate function $\hat{G}$.

Theorem 3.4. *Suppose that $G(u) = Au + b$, where $\|A\| = c < 1$. Then, the Picard iteration $u_{k+1} = \hat{G}(u_k)$ satisfies the following bounds for $k \geq 0$*

$$\|e_k\| \leq c^k \|e_0\| + \frac{1 - c^k}{1 - c}\epsilon_0, \tag{3.74}$$

and

$$\|F_k\| \leq c^k \|F_0\| + (1 + c)\frac{1 - c^k}{1 - c}\epsilon_0. \tag{3.75}$$

Proof. We will prove these bounds by induction. These bounds trivially hold for $k = 0$, as they simply state $\|e_0\| \leq \|e_0\|$ and $\|F_0\| \leq \|F_0\|$. We then suppose that the bounds hold for some $k \geq 0$. We will begin by relating e_{k+1} to e_k and F_{k+1} to F_k. First, noting that the solution u^* satisfies $u^* = G(u^*) = Au^* + b$, we have

$$e_{k+1} = u_{k+1} - u^* = \hat{G}(u_k) - G(u^*) = (Au_k + b + \epsilon(u_k)) - (Au^* + b) = Ae_k + \epsilon(u_k). \tag{3.76}$$

Then, recalling that the true residual is given by $F(u) = (A - I)u + b$

$$\begin{aligned} F_{k+1} = (A - I)u_{k+1} + b = (A - I)(Au_k + b + \epsilon(u_k)) + b = A(A - I)u_k + Ab + (A - I)\epsilon(u_k) \\ = A[(A - I)u_k + b] + (A - I)\epsilon(u_k) = AF_k + (A - I)\epsilon(u_k). \end{aligned} \tag{3.77}$$

Then, it follows that

$$\begin{aligned}
\|e_{k+1}\| &= \|Ae_k + \epsilon(u_k)\| \\
&\leq c\|e_k\| + \epsilon_0 \\
&\leq c\left(c^k\|e_0\| + \frac{1-c^k}{1-c}\epsilon_0\right) + \epsilon_0 \\
&= c^{k+1}\|e_0\| + \left(\frac{c(1-c^k)}{1-c} + 1\right)\epsilon_0 \\
&= c^{k+1}\|e_0\| + \frac{1-c^{k+1}}{1-c}\epsilon_0,
\end{aligned}$$

and similarly

$$\begin{aligned}
\|F_{k+1}\| &= \|AF_k + (A-I)\epsilon(u_k)\| \\
&\leq c\|F_k\| + (1+c)\epsilon_0 \\
&\leq c\left(c^k\|F_0\| + (1+c)\frac{1-c^k}{1-c}\epsilon_0\right) + (1+c)\epsilon_0 \\
&= c^{k+1} + (1+c)\left(\frac{c(1-c^k)}{1-c} + 1\right)\epsilon_0 \\
&= c^{k+1}\|F_0\| + (1+c)\frac{1-c^{k+1}}{1-c}\epsilon_0.
\end{aligned}$$

Hence, we see that the bounds hold for $k+1$, so this completes the induction. □

From this result it holds that $\limsup_{k\to\infty}\|e_k\| \leq \frac{\epsilon_0}{1-c}$ and $\limsup_{k\to\infty}\|F_k\| \leq \frac{(1+c)\epsilon_0}{1-c}$. Thus, asymptotically the iteration errors and residuals are proportional to the error in the function evaluation, and unless c is very near one, the error in the function evaluation is not significantly amplified. Ideally, we would like a similar result to hold for Anderson acceleration, and this is in fact the case, though we must now utilize Assumption 3.2. In our original analysis of the error-free case, this assumption was required in analysis for nonlinear problems, though no such assumption was necessary for linear problems.

Theorem 3.5. *Suppose that $G(u) = Au + b$, where $\|A\| = c < 1$, and Assumption 3.2 holds. Then the iteration produced by Algorithm 6 satisfies the following bounds for $k \geq 0$*

$$\|e_k\| \leq c^k\frac{1+c}{1-c}\|e_0\| + \left(1-c^k\right)\frac{M_\alpha + c}{(1-c)^2}\epsilon_0, \tag{3.78}$$

and

$$\|F_k\| \leq c^k\|F_0\| + \left(1-c^k\right)\frac{M_\alpha + c}{1-c}\epsilon_0. \tag{3.79}$$

Proof. We again proceed by induction. We will first show (3.79), and then (3.78) will follow directly from this. Again, the inequalities hold trivially for $k = 0$ as they say $\|e_0\| \leq \frac{1+c}{1-c}\|e_0\|$ and $\|F_0\| \leq \|F_0\|$, and $\frac{1+c}{1-c} > 1$ because $c < 1$.

Now suppose the inequalities hold for some $k \geq 0$. We will begin by expressing F_{k+1} in an alternate form.

$$\begin{aligned}
F_{k+1} &= (A-I)u_{k+1} + b \\
&= (A-I)\left(\sum_{i=0}^{m_k} \alpha_i^{(k)}(Au_{k-m_k+i} + b + \epsilon(u_{k-m_k+i}))\right) + b \\
&= A\left(\sum_{i=0}^{m_k} \alpha_i^{(k)}(A-I)u_{k-m_k+i}\right) + Ab + (A-I)\left(\sum_{i=0}^{m_k} \alpha_i^{(k)}\epsilon(u_{k-m_k+i})\right) \\
&= A\sum_{i=0}^{m_k} \alpha_i^{(k)}((A-I)u_{k-m_k+i} + b + \epsilon(u_{k-m_k+i})) - \sum_{i=0}^{m_k} \alpha_i^{(k)}\epsilon(u_{k-m_k+i}) \\
&= A\sum_{i=0}^{m_k} \alpha_i^{(k)}\hat{F}_{k-m_k+i} - \sum_{i=0}^{m_k} \alpha_i^{(k)}\epsilon(u_{k-m_k+i}).
\end{aligned}$$

From this, and recalling that $\{\alpha_i^{(k)}\}$ solves (3.72) and thus $\left\|\sum_{i=0}^{m_k} \alpha_i^{(k)}\hat{F}_{k-m_k+i}\right\| \leq \|\hat{F}_k\|$, it holds that

$$\|F_{k+1}\| \leq c\|\hat{F}_k\| + M_\alpha\epsilon_0 \leq c\|F_k\| + (M_\alpha + c)\epsilon_0. \tag{3.80}$$

Then, from the induction hypothesis, we have

$$\begin{aligned}
\|F_{k+1}\| &\leq c\|F_k\| + (M_\alpha + c)\epsilon_0 \\
&\leq c\left(c^k\|F_0\| + \left(1 - c^k\right)\frac{M_\alpha + c}{1-c}\epsilon_0\right) + (M_\alpha + c)\epsilon_0 \\
&= c^{k+1}\|F_0\| + (M_\alpha + c)\left(\frac{c(1-c^k)}{1-c} + 1\right)\epsilon_0 \\
&= c^{k+1}\|F_0\| + (1 - c^{k+1})\frac{M_\alpha + c}{1-c}\epsilon_0.
\end{aligned}$$

Therefore, (3.79) holds for $k + 1$. Then, to prove the bound on the iteration errors, we recall that

$$(1-c)\|e\| \leq \|F(u)\| \leq (1+c)\|e\|.$$

(3.78) then directly follows from this and (3.79), as

$$\|e_{k+1}\| \leq \frac{\|F_{k+1}\|}{1-c} \leq c^{k+1}\frac{\|F_0\|}{1-c} + \left(1 - c^{k+1}\right)\frac{M_\alpha + c}{(1-c)^2}\epsilon_0 \leq c^{k+1}\frac{1+c}{1-c}\|e_0\| + \left(1 - c^{k+1}\right)\frac{M_\alpha + c}{(1-c)^2}\epsilon_0.$$

Hence, (3.78) also holds for $k+1$, so this completes the induction. □

Now we have $\limsup_{k\to\infty} \|e_k\| \leq \frac{(M_\alpha+c)\epsilon_0}{(1-c)^2}$ and $\limsup_{k\to\infty} \|F_k\| \leq \frac{(M_\alpha+c)\epsilon_0}{1-c}$. $M_\alpha \geq 1$, so it seems that the errors in the function evaluation may be amplified a bit more than for Picard iteration. However, for a reasonably small value of m, we have observed that value of M_α generally remains fairly small. We also see an extra factor of $\frac{1}{1-c}$, which can be large if the function is weakly contractive. Thus, so long as the contractive constant c is not very close to one, the effect of an inaccurate function evaluation should not be significantly worse for Anderson acceleration than for Picard iteration for linear problems.

3.5.2 Nonlinear Problems

We now consider the case where $G(u)$ is a nonlinear function. Again, we will first consider the effect that inaccuracies in the evaluation of G have on Picard iteration. We obtain a very similar result to that for linear problems. In this case however, we assume that G is a contraction in a neighborhood of the solution, and we must impose a bound on the size of the function errors in order to ensure that the iteration remains in this neighborhood.

Theorem 3.6. *Suppose that $G(u)$ is contractive with constant c in the neighborhood $\mathcal{B}_{\hat{\rho}}(u^*)$. Then, given $u_0 \in \mathcal{B}_{\hat{\rho}}(u^*)$ and ϵ_0 sufficiently small, the iterates produced by Picard iteration remain in $\mathcal{B}_{\hat{\rho}}(u^*)$ and satisfy the following bounds for $k \geq 0$*

$$\|e_k\| \leq c^k \|e_0\| + \frac{1-c^k}{1-c}\epsilon_0, \tag{3.81}$$

and

$$\|F_k\| \leq c^k \|F_0\| + (1+c)\frac{1-c^k}{1-c}\epsilon_0. \tag{3.82}$$

Proof. First, we let ϵ_0 be small enough such that

$$\epsilon_0 \leq (1-c)\hat{\rho}. \tag{3.83}$$

This condition guarantees that if $u \in \mathcal{B}_{\hat{\rho}}(u^*)$ then $\hat{G}(u) \in \mathcal{B}_{\hat{\rho}}(u^*)$. To see this, suppose that $\|u-u^*\| \leq \hat{\rho}$. Then

$$\|\hat{G}(u) - u^*\| = \|(G(u)+\epsilon(u)) - G(u^*)\| \leq c\|u-u^*\| + \epsilon_0 \leq c\hat{\rho} + (1-c)\hat{\rho} = \hat{\rho}.$$

Hence, if ϵ_0 satisfies (3.83) and $u_0 \in \mathcal{B}_{\hat{\rho}}(u^*)$ then each successive Picard iterate will remain in this neighborhood.

Now, we can prove the bounds (3.81) and (3.82) by induction, but the proof is essentially identical to the proof for the linear case, instead using the contractivity of G to show $\|e_{k+1}\| \leq$

$c\|e_k\| + \epsilon_0$ and $\|F_{k+1}\| \leq c\|F_k\| + (1+c)\epsilon_0$, so this will be omitted here. □

Hence, we obtain nearly the same result for linear and nonlinear problems for Picard iteration. The only difference is that nonlinear result only holds locally and requires a sufficiently small bound on the errors. The next result shows that we also obtain very similar results for Anderson for linear and nonlinear problems. As with Picard, the significant difference is that the results now hold locally and require a bound on the acceptable size of ϵ_0.

Theorem 3.7. *Suppose that $G(u)$ is contractive with constant c and $F'(u)$ is Lipschitz continuous with constant γ in the neighborhood $\mathcal{B}_{\hat{\rho}}(u^*)$, and let $\hat{c} \in (c, 1)$. Then, if Assumption 3.2 holds and given $\rho, \|e_0\|$, and ϵ_0 sufficiently small, the iteration produced by Algorithm 6 satisfies the following for $k \geq 0$: the iterates remains in the neighborhood $\mathcal{B}_\rho(u^*)$ (i.e. $\|e_k\| \leq \rho$), and the following bounds hold*

$$\|e_k\| \leq \hat{c}^k \frac{1+c}{1-c}\|e_0\| + \left(1 + \frac{\gamma\rho}{2(1-c)}\right)(M_\alpha + c)\frac{1-\hat{c}^k}{(1-c)(1-\hat{c})}\epsilon_0, \tag{3.84}$$

$$\|F_k\| \leq \hat{c}^k \|F_0\| + \left(1 + \frac{\gamma\rho}{2(1-c)}\right)(M_\alpha + c)\frac{1-\hat{c}^k}{1-\hat{c}}\epsilon_0. \tag{3.85}$$

Proof. We first let $\rho \leq \hat{\rho}$ and reduce ρ if necessary so that the following holds

$$\left(1 + \frac{\gamma\rho}{2(1-c)}\right)\left(c + \frac{\gamma\rho M_\alpha}{2(1-c)}\hat{c}^{-m}\right) \leq \hat{c}. \tag{3.86}$$

This is necessarily achievable for some $\rho > 0$, because the left hand side is a continuous function of ρ which goes to c at $\rho = 0$. Then, let $\|e_0\|$ and ϵ_0 satisfy

$$\|e_0\| \leq \min\left\{\rho, \left(1 + \frac{\gamma\rho}{2(1-c)}\right)\frac{1-c}{1+c}\rho\right\}, \tag{3.87}$$

and

$$\epsilon_0 \leq \frac{(1-c)(1-\hat{c})}{(1 + \frac{\gamma\rho}{2(1-c)})(M_\alpha + c)}\rho. \tag{3.88}$$

We again proceed by induction. As before, the case for $k = 0$ is trivial. We assume $\|e_0\| \leq \min\{\rho, (1 + \frac{\gamma\rho}{2(1-c)})\frac{1-c}{1+c}\rho\} \leq \rho$ so $u_0 \in \mathcal{B}_\rho(u^*)$, and (3.84) and (3.85) state $\|e_0\| \leq \frac{1+c}{1-c}\|e_0\|$ and $\|F_0\| \leq \|F_0\|$ which again hold trivially.

Next, let $k \geq 0$ and suppose that $\|e_j\| \leq \rho$ and the bounds (3.84) and (3.85) hold for each $0 \leq j \leq k$. We will first show that $\|e_{k+1}\| \leq \rho$. We begin by noting that we can write the previous residuals as follows

$$F_{k-m_k+i} = F'(u^*)e_{k-m_k+i} + \Delta_{k-m_k+i}, \quad 0 \leq i \leq m_k, \tag{3.89}$$

where

$$\|\Delta_{k-m_k+i}\| \leq \frac{\gamma}{2}\|e_{k-m_k+i}\|^2. \tag{3.90}$$

Alternately, because $F'(u^*)$ is invertible due to the Banach Lemma, we can write this as

$$\begin{aligned} e_{k-m_k+i} &= F'(u^*)^{-1}(F_{k-m_k+i} - \Delta_{k-m_k+i}) \\ &= F'(u^*)^{-1}(\hat{F}_{k-m_k+i} - \Delta_{k-m_k+i} - \epsilon(u_{k-m_k+i})). \end{aligned} \tag{3.91}$$

We rewrite u_{k+1} as

$$u_{k+1} = \sum_{i=0}^{m_k} \alpha_i^{(k)} \hat{G}(u_{k-m_k+i}) = \sum_{i=0}^{m_k} \alpha_i^{(k)}[u_{k-m_k+i} + \hat{F}(u_{k-m_k+i})]. \tag{3.92}$$

Because $\sum_{i=0}^{m_k} \alpha_i^{(k)} = 1$, this implies

$$e_{k+1} = \sum_{i=0}^{m_k} \alpha_i^{(k)}[e_{k-m_k+i} + \hat{F}(u_{k-m_k+i})]. \tag{3.93}$$

Substituting (3.91) into this, and defining

$$\bar{\Delta}_k = \sum_{i=0}^{m_k} \alpha_i^{(k)} \Delta_{k-m_k+i}, \text{ and } E_k = \sum_{i=0}^{m_k} \alpha_i^{(k)} \epsilon(u_{k-m_k+i}),$$

results in the following

$$\begin{aligned} e_{k+1} &= \sum_{i=0}^{m_k} \alpha_i^{(k)}[F'(u^*)^{-1}(\hat{F}_{k-m_k+i} - \Delta_{k-m_k+i} - \epsilon(u_{k-m_k+i})) + \hat{F}(u_{k-m_k+i})] \\ &= F'(u^*)^{-1} \sum_{i=0}^{m_k} \alpha_i^{(k)}[(I + F'(u^*))\hat{F}_{k-m_k+i} - \Delta_{k-m_k+i} - \epsilon(u_{k-m_k+i}))] \\ &= F'(u^*)^{-1} \left(G'(u^*) \sum_{i=0}^{m_k} \alpha_i^{(k)} \hat{F}_{k-m_k+i} - \bar{\Delta}_k - E_k \right). \end{aligned} \tag{3.94}$$

Applying norms and recalling that $\{\alpha_i^{(k)}\}$ solve (3.72) and thus $\| \sum_{i=0}^{m_k} \alpha_i^{(k)} \hat{F}_{k-m_k+i}\| \leq \|\hat{F}_k\|$ gives

$$\|e_{k+1}\| \leq \frac{1}{1-c} \left(c\|\hat{F}_k\| + \|\bar{\Delta}_k\| + \|E_k\| \right). \tag{3.95}$$

We then bound the quantities $\|\bar{\Delta}_k\|$ and $\|E_k\|$. $\|E_k\|$ is straightforward, as

$$\|E_k\| \leq \sum_{i=0}^{m_k} |\alpha_i^{(k)}| \epsilon_0 \leq M_\alpha \epsilon_0. \tag{3.96}$$

Now, using (3.90) and the assumption that $\|e_j\| \leq \rho$ for each $0 \leq j \leq k$ gives

$$\|\bar{\Delta}_k\| \leq \frac{\gamma}{2} \sum_{i=0}^{m_k} |\alpha_i^{(k)}| \|e_{k-m_k+i}\|^2 \leq \frac{\gamma\rho}{2} \sum_{i=0}^{m_k} |\alpha_i^{(k)}| \|e_{k-m_k+i}\|. \tag{3.97}$$

Using the induction hypothesis, for $0 \leq i \leq m_k$ we have the following bound:

$$\begin{aligned} \|e_{k-m_k+i}\| &\leq \frac{\|F_{k-m_k+i}\|}{1-c} \\ &\leq \frac{1}{1-c}\left(\hat{c}^{k-m_k+i}\|F_0\| + \left(1+\frac{\gamma\rho}{2(1-c)}\right)\frac{(M_\alpha+c)(1-\hat{c}^{k-m_k+i})}{1-\hat{c}}\epsilon_0\right) \\ &\leq \frac{1}{1-c}\left(\hat{c}^{k-m}\|F_0\| + \left(1+\frac{\gamma\rho}{2(1-c)}\right)\frac{(M_\alpha+c)(1-\hat{c}^{k})}{1-\hat{c}}\epsilon_0\right). \end{aligned}$$

In this we used the fact that for any i such that $0 \leq i \leq m_k$, it holds that $k-m \leq k-m_k+i \leq k$. Hence $\hat{c}^{k-m_k+i} \leq \hat{c}^{k-m}$ and $1-\hat{c}^{k-m_k+i} \leq 1-\hat{c}^k$. Then, from this and Assumption 3.2 it follows that

$$\|\bar{\Delta}_k\| \leq \frac{\gamma\rho M_\alpha}{2(1-c)}\left(\hat{c}^{k-m}\|F_0\| + \left(1+\frac{\gamma\rho}{2(1-c)}\right)\frac{(M_\alpha+c)(1-\hat{c}^{k})}{1-\hat{c}}\epsilon_0\right). \tag{3.98}$$

Now, combining (3.96) and (3.98), and using the induction hypothesis for $\|F_k\|$ results in the following

$$\begin{aligned} c\|\hat{F}_k\| + \|\bar{\Delta}_k\| + \|E_k\| \leq{}& c\left(\hat{c}^k\|F_0\| + \left(1+\frac{\gamma\rho}{2(1-c)}\right)(M_\alpha+c)\frac{1-\hat{c}^k}{1-\hat{c}}\epsilon_0\right) \\ &+ \frac{\gamma\rho M_\alpha}{2(1-c)}\left(\hat{c}^{k-m}\|F_0\| + \left(1+\frac{\gamma\rho}{2(1-c)}\right)\frac{(M_\alpha+c)(1-\hat{c}^{k})}{1-\hat{c}}\epsilon_0\right) \\ &+ (M_\alpha+c)\epsilon_0. \end{aligned} \tag{3.99}$$

Collecting terms, this gives

$$\begin{aligned} c\|\hat{F}_k\| + \|\bar{\Delta}_k\| + \|E_k\| \leq{}& \left(c + \frac{\gamma\rho M_\alpha}{2(1-c)}\hat{c}^{-m}\right)\hat{c}^k\|F_0\| \\ &+ \left[\left(1+\frac{\gamma\rho}{2(1-c)}\right)\left(c+\frac{\gamma\rho M_\alpha}{2(1-c)}\right)(1-\hat{c}^k) + 1 - \hat{c}\right]\frac{M_\alpha+c}{1-\hat{c}}\epsilon_0. \end{aligned} \tag{3.100}$$

We now note that $1 \leq \hat{c}^{-m}$, so (3.86) also implies that $\left(1+\frac{\gamma\rho}{2(1-c)}\right)\left(c+\frac{\gamma\rho M_\alpha}{2(1-c)}\right) \leq \hat{c}$. Using

(3.86), (3.100) then becomes

$$\begin{aligned} c\|\hat{F}_k\| + \|\bar{\Delta}_k\| + \|E_k\| &\leq \frac{\hat{c}}{1+\frac{\gamma\rho}{2(1-c)}}\hat{c}^k\|F_0\| + \left(\hat{c}(1-\hat{c}^k)+1-\hat{c}\right)\frac{M_\alpha + c}{1-\hat{c}}\epsilon_0 \\ &= \frac{\hat{c}^{k+1}}{1+\frac{\gamma\rho}{2(1-c)}}\|F_0\| + (1-\hat{c}^{k+1})\frac{M_\alpha+c}{1-\hat{c}}\epsilon_0. \end{aligned} \tag{3.101}$$

Now, substituting this into (3.95) and using the bounds (3.87) and (3.88) gives

$$\begin{aligned} \|e_{k+1}\| &\leq \frac{1}{1-c}\left(\frac{\hat{c}^{k+1}}{1+\frac{\gamma\rho}{2(1-c)}}\|F_0\| + (1-\hat{c}^{k+1})\frac{M_\alpha+c}{1-\hat{c}}\epsilon_0\right) \\ &\leq \frac{(1+c)\hat{c}^{k+1}}{(1-c)(1+\frac{\gamma\rho}{2(1-c)})}\|e_0\| + (1-\hat{c}^{k+1})\frac{M_\alpha+c}{(1-c)(1-\hat{c})}\epsilon_0 \\ &\leq \hat{c}^{k+1}\rho + (1-\hat{c}^{k+1})\rho = \rho. \end{aligned} \tag{3.102}$$

Hence, $\|e_{k+1}\| \leq \rho$, and thus u_{k+1} is in the neighborhood $\mathcal{B}_\rho(u^*)$. It then remains to show the bounds (3.84) and (3.85) hold for $k+1$. We can now use the fundamental theorem of calculus and the expression (3.94) to give the following

$$\begin{aligned} F_{k+1} &= \int_0^1 F'(u^* + te_{k+1})e_{k+1}\,\mathrm{d}t \\ &= \int_0^1 F'(u^*+te_{k+1})F'(u^*)^{-1}\,\mathrm{d}t\left(G'(u^*)\sum_{i=0}^{m_k}\alpha_i^{(k)}\hat{F}_{k-m_k+i} - \bar{\Delta}_k - E_k]\right). \end{aligned}$$

Applying norms, this gives

$$\|F_{k+1}\| \leq \int_0^1 \|F'(u^*+te_{k+1})F'(u^*)^{-1}\|\,\mathrm{d}t\left(c\|\hat{F}_k\| + \|\bar{\Delta}_k\| + \|E_k\|\right). \tag{3.103}$$

Because $\|e_{k+1}\| \leq \rho$, for any $t \in [0,1]$ $u^* + te_{k+1} \in \mathcal{B}_\rho(u^*)$. Then, by Lipschitz continuity of F' in this neighborhood we have the following

$$\begin{aligned} \|F'(u^*+te_{k+1})F'(u^*)^{-1}\| &= \|I - [F'(u^*) - F'(u^*+te_{k+1})]F'(u^*)^{-1}\| \\ &\leq 1 + \|F'(u^*) - F'(u^*+te_{k+1})\|\|F'(u^*)^{-1}\| \\ &\leq 1 + \frac{\gamma t\|e_{k+1}\|}{1-c}. \end{aligned}$$

Therefore,

$$\int_0^1 \|F'(u^*+te_{k+1})F'(u^*)^{-1}\|\,\mathrm{d}t \leq 1 + \frac{\gamma\|e_{k+1}\|}{2(1-c)} \leq 1 + \frac{\gamma\rho}{2(1-c)}. \tag{3.104}$$

This, along with equations (3.103) and (3.101) give

$$\begin{aligned}
\|F_{k+1}\| &\leq \left(1+\frac{\gamma\rho}{2(1-c)}\right)\left(c\|F_k\|+\|\bar{\Delta}_k\|+\|E_k\|\right) \\
&\leq \left(1+\frac{\gamma\rho}{2(1-c)}\right)\left(\frac{\hat{c}^{k+1}}{1+\frac{\gamma\rho}{2(1-c)}}\|F_0\|+(1-\hat{c}^{k+1})\frac{M_\alpha+c}{1-\hat{c}}\epsilon_0\right) \\
&= \hat{c}^{k+1}\|F_0\|+\left(1+\frac{\gamma\rho}{2(1-c)}\right)(M_\alpha+c)\frac{1-\hat{c}^{k+1}}{1-\hat{c}}\epsilon_0. \qquad (3.105)
\end{aligned}$$

Thus (3.85) holds for $k+1$, and (3.84) follows directly from this as

$$\begin{aligned}
\|e_{k+1}\| &\leq \frac{\|F_{k+1}\|}{1-c} \\
&\leq \hat{c}^{k+1}\frac{\|F_0\|}{1-c}+\left(1+\frac{\gamma\rho}{2(1-c)}\right)(M_\alpha+c)\frac{1-\hat{c}^{k+1}}{(1-c)(1-\hat{c})}\epsilon_0 \\
&\leq \hat{c}^{k+1}\frac{1+c}{1-c}\|e_0\|+\left(1+\frac{\gamma\rho}{2(1-c)}\right)(M_\alpha+c)\frac{1-\hat{c}^{k+1}}{(1-c)(1-\hat{c})}\epsilon_0. \qquad (3.106)
\end{aligned}$$

This completes the induction. □

This shows that

$$\limsup_{k\to\infty}\|e_k\| \leq \left(1+\frac{\gamma\rho}{2(1-c)}\right)\frac{(M_\alpha+c)}{(1-c)(1-\hat{c})}\epsilon_0,$$

and

$$\limsup_{k\to\infty}\|F_k\| \leq \left(1+\frac{\gamma\rho}{2(1-c)}\right)\frac{(M_\alpha+c)}{(1-\hat{c})}\epsilon_0.$$

Hence, asymptotically the the evaluation errors may be amplified by an additional factor of $(1+\frac{\gamma\rho}{2(1-c)})(\frac{1-c}{1-\hat{c}})$ when compared with the result for linear problems, but the results can be made arbitrarily close by letting $\hat{c}$ be sufficiently close to c and ρ sufficiently small. We note that the conditions (3.86) and (3.88) may require $\|e_0\|$ to be significantly smaller than $\hat{\rho}$, which is the radius for which the Picard results hold. This seems to suggest that it may make sense to perform several Picard iterations if the initial iterate is not known to be relatively good. We also note that the bound on ϵ_0 is smaller in this case than it is for Picard, and it may be very small if the contractive constant is near one or if M_α is fairly large. We will return to this issue when considering the effect of varying single-physics application tolerances for Tiamat single-rod tests in Section 6.4.1.

3.6 Stochastic Errors in the Function Evaluation

The results in the previous section can be modified fairly simply to provide results for the case where the errors in the function evaluation are probabilistic in nature. For this, we consider a problem setup as in [63]. In this, the evaluation of F is assumed to involve a Monte Carlo simulation with N_{MC} trials, and the error in the Monte Carlo residual $\hat{F}(u, N_{MC})$ is assumed to satisfy Assumption 3.4. The authors of this paper consider the effect that errors of this nature in the residual evaluation have on JFNK. It is observed that the inaccuracies are problematic for the Krylov solves for the Newton step, and they formulate an algorithm in which the number of trials in the residual evaluation are increased as the iteration progresses which performs well. Results in this section suggestion suggest that Anderson might be a more resilient alternative to JFNK in this context, and does not require the number of Monte Carlo trials to increase throughout the iteration. The results in this section simply extend the results from the previous section, and show that these bounds can be made to hold for any finite number of iterations with arbitrary probability given enough Monte Carlo trials. We will present the adapted results for linear problems, and the results for nonlinear problems may be reproduced analogously.

Assumption 3.4. *There is a function c_F and an open set $\mathcal{B}'$ which contains $\mathcal{B}_{\hat{\rho}}(u^*)$ such that for all $u \in \mathcal{B}'$ and $\delta > 0$*

$$Prob\left(\|F(u) - \hat{F}(u, N_{MC})\| > \frac{c_F(\delta)}{\sqrt{N_{MC}}}\right) < \delta. \tag{3.107}$$

In keeping with the conventions from the previous section, we will define the function evaluation error $\epsilon(u, N_{MC}) = \hat{F}(u, N_{MC}) - F(u)$ and the inexact fixed-point map $\hat{G}(u, N_{MC}) = G(u) + \epsilon(u, N_{MC})$. Then, given (sufficiently small in the case of nonlinear problems) $\epsilon_0 > 0$, the results from the previous section will hold for a finite number K iterations if $\|\epsilon(u_j, N_{MC})\| \leq \epsilon_0$ for each $0 \leq j \leq K-1$. As the errors have no upper bounds, but rather small variances, this will not hold in general. However, we can choose N_{MC} such that this will in fact be the case with any arbitrary probability in $(0, 1)$.

Theorem 3.8. *Suppose that $G(u) = Au + b$, where $\|A\| = c < 1$, and let a positive integer $K, \omega \in (0, 1)$, and $\epsilon_0 > 0$ be given. Then, there exists N_{MC} such that with probability at least $(1 - \omega)$ the Picard iteration $u_{k+1} = \hat{G}(u_k, N_{MC})$ satisfies the following bounds for $0 \leq k \leq K$:*

$$\|e_k\| \leq c^k \|e_0\| + \frac{1 - c^k}{1 - c}\epsilon_0, \tag{3.108}$$

and

$$\|F_k\| \leq c^k \|F_0\| + (1 + c)\frac{1 - c^k}{1 - c}\epsilon_0. \tag{3.109}$$

Proof. As was already noted, this follows immediately from Theorem 3.4 if $\|\epsilon(u_k, N_{MC})\| \leq \epsilon_0$ for $0 \leq k \leq K-1$. We first define

$$\delta = 1 - (1-\omega)^{1/K}. \tag{3.110}$$

Then, if we let

$$N_{MC} \geq \left(\frac{c_F(\delta)}{\epsilon_0} \right)^2, \tag{3.111}$$

it will hold with probability at least $1-\delta$ that for any $0 \leq k \leq K-1$

$$\|\epsilon(u_k, N_{MC})\| \leq \epsilon_0. \tag{3.112}$$

Each function evaluation is independent, so with probability at least $(1-\delta)^K = 1-\omega$, that $\|\epsilon(u_k, N_{MC})\| \leq \epsilon_0$ for each $0 \leq k \leq K-1$. Hence, from Theorem 3.4 it follows that the desired bounds hold with probability at least $1-\omega$ □

We can analogously use Theorem 3.5 to give the following result for Anderson acceleration on linear problems.

Theorem 3.9. *Suppose that $G(u) = Au + b$, where $\|A\| = c < 1$, and Assumption 3.2 holds, and let a positive integer $K, \omega \in (0,1)$, and $\epsilon_0 > 0$ be given. Then there exists N_{MC} such that with probability at least $(1-\omega)$ the iteration produced by Algorithm 6 satisfies the following bounds for $0 \leq k \leq K$:*

$$\|e_k\| \leq c^k \frac{1+c}{1-c} \|e_0\| + \left(1 - c^k\right) \frac{M_\alpha + c}{(1-c)^2} \epsilon_0, \tag{3.113}$$

and

$$\|F_k\| \leq c^k \|F_0\| + \left(1 - c^k\right) \frac{M_\alpha + c}{1-c} \epsilon_0. \tag{3.114}$$

The proof of this theorem is essentially identical to that for the previous theorem, and in fact given the same integer K, failure probability ω, and error bound ϵ_0, the required number of Monte Carlo trails is the same. In a similar manner, we can show that given large enough N_{MC}, the results from Theorem 3.7 hold for some finite number of iterations with arbitrary probability as well.

Chapter 4

Trilinos Anderson Acceleration Implementation

4.1 Introduction

As part of this work, we have added an Anderson acceleration solver to the Trilinos package of nonlinear solver software NOX. Trilinos is a package of numerical analysis software developed by Sandia National Laboratories. We describe Trilinos, as well as its packages which we reference in this chapter, in greater detail in Appendix B. NOX provides interfaces for various objects (solvers, vector types, status tests) as abstract base classes, and concrete implementations are created as derived subclasses. In the case of NOX solvers, the base class is the `NOX::Solver::Generic` class. This class provides the interface for subroutines which initialize the solver, compute a new iterate, solve the problem to completion, and provide access to data from the solver. Classes which inherit from this base class include the `LineSearchBased` class which implements various nonlinear solver methods based on line search globalization [29], and the `TrustRegionBased` class which implements various trust region based methods [30]. We have implemented `NOX::Solver::AndersonAcceleration` as a new class which derives from the generic solver base class. In this section, we will describe the concrete implementation of several of the inherited member functions for this Anderson acceleration solver class, as well as several of the unit tests we have created to ensure proper functionality.

4.2 Solver Description

This solver attempts to solve the preconditioned problem

$$M(u)F(u) = 0, \tag{4.1}$$

where M is a preconditioner. Preconditioning is optional, and if not enabled the preconditioner is taken to be the identity. The new iterate computed by Anderson acceleration is given by

$$u_{k+1} = u_k + \beta M(u_k)F_k - (\mathcal{U}_k + \beta\mathcal{F}_k)\gamma^{(k)}, \tag{4.2}$$

where the columns of $\mathcal{U}_k$ are differences between consecutive iterates

$$\mathcal{U}_k = [u_{k-m_k+1} - u_{k-m_k}, \ldots, u_k - u_{k-1}],$$

the columns of $\mathcal{F}_k$ are differences between consecutive preconditioned residuals

$$\mathcal{F}_k = [M(u_{k-m_k+1})F_{k-m_k+1} - M(u_{k-m_k})F_{k-m_k}, \ldots, M(u_k)F_k - M(u_{k-1})F_{k-1}],$$

and $\gamma^{(k)}$ solves

$$\min_{\gamma} \|M(u_k)F_k - \mathcal{F}_k\gamma\|. \tag{4.3}$$

In this implementation, rather than storing the matrix $\mathcal{F}_k$ itself we store and update its QR factorization from iteration to iteration. We store $\mathcal{U}_k$ and the Q factor as objects of type `std::vector` which contain pointers to `NOX::Abstract::Vector` types, and these are referred to as `xMat` and `qMat` respectively. The R factor is stored as a dense matrix called `rMat`. For MPI implementations, the entire R factor is replicated across each participating process, but as it contains at most m^2 entries and in most cases m is kept relatively small, this should generally not be a significant storage burden. We overview the process of updating the QR factorization from iteration to iteration in detail in Section 4.2.3.

The solver requires the following inputs during construction:

1. A `NOX::Abstract::Group` object. This class provides the interface to the necessary problem information, i.e. the solution vector, residual, Jacobian, and preconditioner. At minimum, the solver requires the ability to evaluate the residual at the current solution vector. The group provides access to the solution and residual vectors as `NOX::Abstract::Vector`'s. This abstract interface includes routines for various vector operations including scaling, linear products, inner products, and norms. Access to individual vector entries, however, is not provided. If the solver is created with preconditioning enabled, the group implementation must provide a routine to apply the preconditioner to a vector. Jacobian information is only required if preconditioning is enabled, and the preconditioner requires the Jacobian in order to be applied.

2. A `NOX::StatusTest::Generic` object. This object dictates the stopping criteria for the solver. Status tests included with NOX include tests for absolute and relative residual

norm, maximum iteration counts, and a combination test for combining various other status tests. Additionally, user defined status tests may be supplied.

3. A `Teuchos::ParameterList`. The input parameter list dictates particular attributes of the solver to be created. The valid parameters which may be specified for this solver type are described in detail in the following section.

4.2.1 Solver Options

The constructor for this solver class uses an initialization list to copy data from the input parameters and allocate storage for several class members, and then calls the member function `init`. The main purpose of this member function is to set up the valid parameters for the solver, parse and validate the parameter list parameter list supplied to the constructor for the appropriate solver parameters, and set defaults for any unspecified parameters. The following parameters are supported by the solver.

- "Storage Depth" - An integer that determines the maximum storage depth stored by the solver. This is the parameter m for Anderson-m. We set a default value of 2, and throw an exception if the value passed in is less than 0. If the input value is 0, then the solver performs Picard iteration.

- "Mixing Parameter" - A double that gives the mixing parameter. The default value is set to 1.0 (no mixing), and an exception is thrown if the input value is outside the interval $[-1, 0) \cup (0, 1]$.

- "Adjust Matrix for Condition Number" - A Boolean that determines whether the storage depth will be adjusted to maintain good conditioning of the condition number of the least-squares coefficient matrix. The default value is set to false.

- "Condition Number Drop Tolerance" - A double that gives the condition number bound if adjusting the storage depth. This has no effect if the parameter "Adjust Matrix for Condition Number" is not set to true. The default value is set to $10e12$.

- "Acceleration Start Iteration" - . The default value is set to 1, which means Anderson acceleration begins right away. Although we require the storage depth to be at least 1, setting this parameter larger than the iteration count limit will reduce the solver to an Anderson-0 (Picard) solver.

- "Disable Storage Depth Check for Unit Testing" - An exception is thrown if the input storage depth parameter is greater than the problem size, as this necessarily results in a singular least squares problem after enough iterations. If this option is set to true, this

check is disabled. This is included for the purpose of unit testing, and is not intended to be set to true by users.

- A sublist "Preconditioning" which contains the entries:
  - "Precondition" - A Boolean that determines whether or not preconditioning will be used. The default value is set to false.
  - "Recompute Jacobian" - A Boolean that determines whether the Jacobian should be recomputed before applying the preconditioner each iteration. If the preconditioner is computed from the Jacobian and the group object does not internally update the Jacobian prior to computing the preconditioner, this needs to be set to true to update the preconditioner. Otherwise the preconditioner is computed from the initial Jacobian. The default value is false, and this option has no effect unless the "Precondition" parameter is set to true.

We create a parameter list which contains these parameter names with the given default values, and then call the parameter list member function `validateParametersAndSetDefaults`. This function compares the input parameter list agains the valid parameters list and throws an exception if either a parameter is specified which does not match one of the valid parameters, or if a parameter with a valid name has the incorrect value. This acts as a safeguard against potential input errors for the input parameter list.

4.2.2 Step Implementation

The bulk of the work done by the solver is performed in the member function `step`. This function implements the computation of a new iterate.

When in the acceleration stage of the iteration, we begin by computing the QR factors of the matrix $\mathcal{F}_k$. This is performed by calling two subroutines which update the QR factorization of a matrix resulting from appending a vector in the right or deleting a column from the left. The process for performing this update will be described in greater depth in the next section. Then, given the appropriate QR factors, we then solve the linear least-squares problem $\min_\gamma \|M(u_k)F_k - \mathcal{F}_k\gamma\|$. As we have the QR factorization of $\mathcal{F}_k$, this is simply performed by solving

$$R\gamma = Q^T M(u_k)F_k, \tag{4.4}$$

which proceeds as shown in the following code fragment.

```
for (int ii = 0; ii<nStore; ii++)
  RHS(ii,0) = precF->innerProduct( *(qMat[ii]) );
```

```
//Back-solve for gamma
for (int ii = nStore-1; ii>=0; ii--){
  gamma(ii,0) = RHS(ii,0);
  for (int jj = ii+1; jj<nStore; jj++){
    gamma(ii,0) -= rMat(ii,jj)*gamma(jj,0);
  }
  gamma(ii,0) /= rMat(ii,ii);
}
```

As the name suggests, in the above `precF` represents the product of the preconditioner and the residual.

Once the least-squares problem is solved, the computation of the new iterate is fairly straightforward. We first compute the "Anderson direction"

$$d = \beta M(u_k)F_k - (\mathcal{U}_k + \beta\mathcal{F}_k)\gamma^{(k)}. \tag{4.5}$$

We then use a line search to determine a step size λ and compute the new iterates as $u_{k+1} = u_k + \lambda d$. This is performed in the following code fragment.

```
if (nStore > 0)
  Rgamma.multiply(Teuchos::NO_TRANS,Teuchos::NO_TRANS,mixParam,rMat,gamma,0.0);
tempVec->update(mixParam,*precF);
for (int ii=0; ii<nStore; ii++)
  tempVec->update(-gamma(ii,0), *(xMat[ii]), -Rgamma(ii,0),*(qMat[ii]),1.0);
bool ok = lineSearchPtr->compute(*solnPtr, stepSize, *tempVec, *this);
```

The Anderson direction might not be a descent direction, so the use of a backtracking or polynomial line search may or may not be advisable. However, the line search may be useful for controlling the step size in order to maintain given bounds on the solution. This utility is offered through the "Safeguarded Step" line search option included in NOX.

The `step` function then finishes by evaluating the residual at the new iterate and checking for convergence or failure according to the status test. The member function `solve` carries out the iteration by simply calling this `step` routine until the status test for the solver has determined that the iteration has either succeeded or failed.

4.2.3 QR Management Routines

As was noted previously, our preferred method for solving the linear least squares problem (3.6) is storing the QR factorization of $\mathcal{F}_k$ rather than the matrix itself, and updating the factorization directly from iterate to iterate. There are several reasons why this method is preferable to

storing and fully factorizing $\mathcal{F}_k$ each iteration. First, because of the limited operations provided by the NOX vector interface, the only available QR factorization methods are those based on Gram-Schmidt orthogonalization. Householder QR would be more numerically robust [22], but requires access to individual vector entries which the NOX vector interface does not provide. Second, while the cost should in general be minimal compared to the function evaluation, updating the QR factors directly generally requires less computation, and communication in the case of distributed vectors. For full factorization by modified Gram-Schmidt the major computational cost is up to $\frac{m(m-1)}{2}$ inner product computations (or $m(m-1)$ if orthogonalizing twice for improved robustness). For MPI implementations inner products also require communication between processes, and we will see that we require as few or generally fewer inner products by updating the QR factorization directly. Lastly, this method results in minimal storage utilization. When storing and factoring $\mathcal{F}_k$, both the matrix and the Q factor (which have the same size) need to be stored temporarily. Of course, $\mathcal{F}_k$ could be overwritten by its Q factor and recovered by multiplying QR, but this introduces additional computation and is clearly not ideal.

In this section, we describe our implementation of the process for updating the QR factorization over successive iterations. As stated previously, the information from the previous iterates is stored in the "matrices" `xMat`, `qMat`, and `rMat`, where `xMat` and `qMat` are vectors of pointers to `NOX::Abstract::Vector` objects and `rMat` is a dense matrix. Each time `step` is called, we begin with `xMat`, `qMat`, and `rMat` from the previous iteration. The first thing done is to update these to the correct values for the current iteration. At the start of the acceleration, each of these is empty. We then allocate storage for these, and `xMat` is assigned the most recent iterate difference vector, `rMat` is assigned the norm of the most recent residual difference vector, and `qMat` is assigned the normalized residual difference vector. If the acceleration has already started, we perform this update in three stages. First, if the storage depth limit is already saturated we compute the updates resulting from discarding information from the oldest iterate. Next, we append the new iterate difference vector to `xMat` and compute the effect of appending the new residual difference on `qMat` and `rMat`. Lastly, if the storage depth is adjusted to maintain conditioning, the oldest iterates are dropped from storage until the condition bound is satisfied. The first two stages of the update could be performed in either order. We first perform the deletion for storage purposes. If the storage depth is already saturated and the new residual difference is appended first, then `xMat` and `qMat` will at least temporarily contain one more vector than the storage depth parameter dictates. Conversely, by performing the deletion first, the maximum storage depth is never exceeded.

The first two steps described above proceed in the code as shown in the following fragment. The variable `nStore` denotes the number of previous iterates currently in storage, and `storeParam` is the storage depth parameter. Additionally, `qrAdd` and `qrDelete` represent sub-

routines which compute the updated the QR factors of the coefficient matrix resulting from appending a vector on the right and deleting a vector from the left respectively.

```
if (nStore < storeParam){
  xMat.push_back(solnPtr->getX().clone(NOX::ShapeCopy));
  nStore++;
}
else{
  for (int ii = 0; ii<nStore-1; ii++)
    *(xMat[ii]) = *(xMat[ii+1]);
  qrDelete();
}
*(xMat[nStore-1]) = solnPtr->getX();
(xMat[nStore-1])->update(-1.0,oldSolnPtr->getX(),1.0);
oldPrecF->update(1.0, *precF, -1.0);
qrAdd(*oldPrecF);
```

In this, if the storage is not saturated, this simply allocates storage and writes in the new iterate difference vector to `xMat`, and calls `qrAdd` to compute the effect of appending the new residual difference vector on the QR factors. However, if the storage is already saturated, the columns of `xMat` are shuffled, overwriting the oldest vector, and `qrDelete` is called to update the QR factors by column deletion. Then the new difference vectors are appended.

Next, if the option to adjust the storage depth for conditioning is enabled, we potentially discard more of the stored iteration history. As the Q factor is has orthonormal columns, the least-squares coefficient matrix has the same condition number as `rMat` in the l_2 norm, so we adjust storage so that the condition number of `rMat` satisfies a given bound. For condition number estimates the LAPACK routine DGECON is used. This estimates the inverse of the condition number of a matrix, though it only does so with the l_1 or l_∞ norm. We choose to bound the condition number of `rMat` in the l_1 norm, and as the l_1 and l_2 norms are equivalent, this imposes a bound on the condition number of the coefficient matrix. The code fragment which performs this storage depth adjustment is shown below:

```
if (adjustForConditionNumber) {
  while ( (1.0/invCondNum > dropTolerance) && (nStore > 1)  ) {
    for (int ii = 0; ii<nStore-1; ii++)
      *(xMat[ii]) = *(xMat[ii+1]);
    xMat.pop_back();
    qrDelete();
    --nStore;
```

```
      lapack.GECON(normType,nStore,rMat.values(),nStore,rMat.normOne(),
        &invCondNum,&WORK[0],&IWORK[0],&info);
      if (utilsPtr->isPrintType(Utils::Details))
        utilsPtr->out() << "    Adjusted R condition number estimate ("
          << nStore << ") = " << 1.0/invCondNum << std::endl;
    }
  }
```

In this, `dropTolerance` is the specified condition number bound. This fragment repeatedly drops information from the oldest iterate and computes the condition number of the updated R factor until this condition number bound is satisfied. As before, we discard the oldest iterate information by overwriting it in `xMat` and calling `qrDelete` to update the QR factorization. As the condition number of a single vector is one, we should always retain at least one vector, and the inclusion of the while statement condition `nStore > 1` imposes this directly, as well as safeguards against an input dropTolerance below one. Then, all that remains is to describe the `qrAdd` and `qrDelete` subroutines.

qrAdd

We first consider the subroutine which updates the new QR factorization after appending a column. In this subroutine, we begin with the QR factors of the matrix $A \in \mathbb{R}^{N \times M}$ and a vector $b \in \mathbb{R}^N$, and we compute the QR factors of the appended matrix $A' = [A\ b]$. Given the factorization $A = QR$, we can write this as:

$$A' = \begin{bmatrix} QR & b \end{bmatrix} = \begin{bmatrix} Q & b \end{bmatrix} \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix}. \tag{4.6}$$

Then, as the columns of Q are already assumed to be orthonormal, to obtain the QR factors of A' we simply need to orthogonalize b against the columns of Q. This is accomplished via the Gram-Schmidt process [22]. In this, we sequentially subtract off of b its component in the direction of each of the columns of Q. Letting q_i be column i of Q, this proceeds as follows.

- For $i = 1, 2, \ldots, M$.
  - Set $r_{i,M+1} = q_i^T b$.
  - Set $b = b - r_{i,M+1} q_i$.
- Set $r_{M+1,M+1} = \|b\|$.
- Set $q_{M+1} = \frac{b}{r_{M+1,M+1}}$.

The updated factorization is then given by $A' = Q'R'$, where

$$Q' = \left[\begin{array}{cc} Q & q_{M+1} \end{array} \right], \tag{4.7}$$

and

$$R' = \left[\begin{array}{c|c} & r_{1,M+1} \\ R & \vdots \\ & r_{M.M+1} \\ \hline 0 & r_{M+1,M+1} \end{array} \right]. \tag{4.8}$$

In an initial implementation, the `qrAdd` subroutine was implemented as described above. However, it was seen to result in poor performance for some problems where the storage depth for Anderson is modestly large and the least-squares coefficient matrix becomes highly ill-conditioned. An example of this is shown in Figure 4.1a. This shows the iteration history for Anderson-10 for the Chandrasekhar H-equation (Section 3.2.3) with $\omega = 0.9999$. In this, "Loss of Orthogonality" is a measure of the distance of the Q factor from orthogonality which we define as:

$$\text{Loss of Orthogonality } = \|I - Q^T Q\|.$$

This is a problem for which the iteration should be convergent. Initially, the residual norms decrease as expected, but as the condition number of the least-squares matrix increases rapidly the loss of orthogonality in the Q factor becomes rather large. As a result, the iteration begins to diverge. This loss of orthogonality is consistent with a result in [6], which says that the loss of orthogonality in the Q factor for the modified Gram-Schmidt algorithm due to roundoff error is proportional to the condition number of the matrix being factored times machine epsilon. This problem may be rectified by performing another round of orthogonalization. That is, first factor $A = QR$, then compute the factorization $Q = \bar{Q}\bar{R}$. As Q should be somewhat near orthogonal, it should likely have a condition number somewhat near one. As a result, we expect the loss of orthogonality for $\bar{Q}$ to be on the order of machine epsilon. We then have an improved QR factorization of A by letting $A = \bar{Q}(\bar{R}R)$. In our case, this simply means that orthogonalizing the vector being appended against the columns of the previous Q factor twice should be sufficient to maintain orthogonality of the Q factor to machine precision. This agrees with a result from [42], which states that orthogonalizing twice in a Gram-Schmidt process will maintain sufficient orthogonality. In Figure 4.1b we see that this process of twice orthognalizing the newly appended column against the previous Q factor maintains orthogonality of the Q factor on the order of machine precision despite a highly ill-conditioned coefficient matrix, and the iteration converges as expected. The second round of orthogonalization doubles the computational cost of this subroutine, but the number of operations is still only O(MN), so

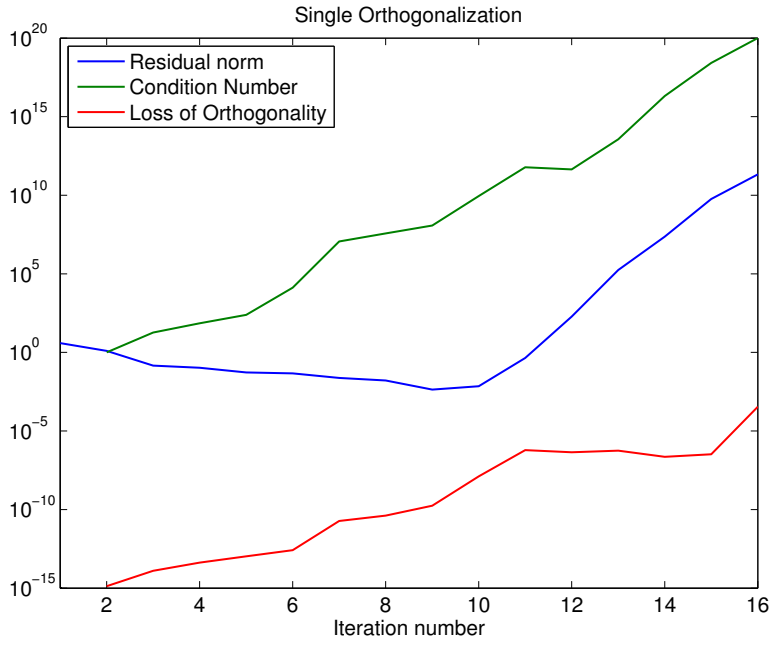
it seems that the additional robustness justifies the increased cost. This change results in the following code fragment, which is the subroutine as included in the solver:

```
void NOX::Solver::AndersonAcceleration::qrAdd(NOX::Abstract::Vector& newCol)
{
  // Increase size of QR factors.
  int N = qMat.size();
  qMat.push_back(solnPtr->getX().clone(NOX::ShapeCopy));
  rMat.reshape(N+1,N+1);
  // Update QR factors
  // Orthogonalize against previous columns once
  for (int ii = 0; ii<N; ii++){
    rMat(ii,N) = qMat[ii]->innerProduct(newCol);
    newCol.update(-rMat(ii,N),*(qMat[ii]),1.0);
  }
  // Reorthogonalize
  for (int ii = 0; ii<N; ii++){
    double Alpha = qMat[ii]->innerProduct(newCol);
    newCol.update(-Alpha,*(qMat[ii]),1.0);
    rMat(ii,N) += Alpha;
  }
  rMat(N,N) = newCol.norm();
  TEUCHOS_TEST_FOR_EXCEPTION((rMat(N,N) < 1.0e-16),std::logic_error,
    "Error - R factor is singular to machine precision!");
  *(qMat[N]) = newCol.scale(1.0/rMat(N,N));
}
```

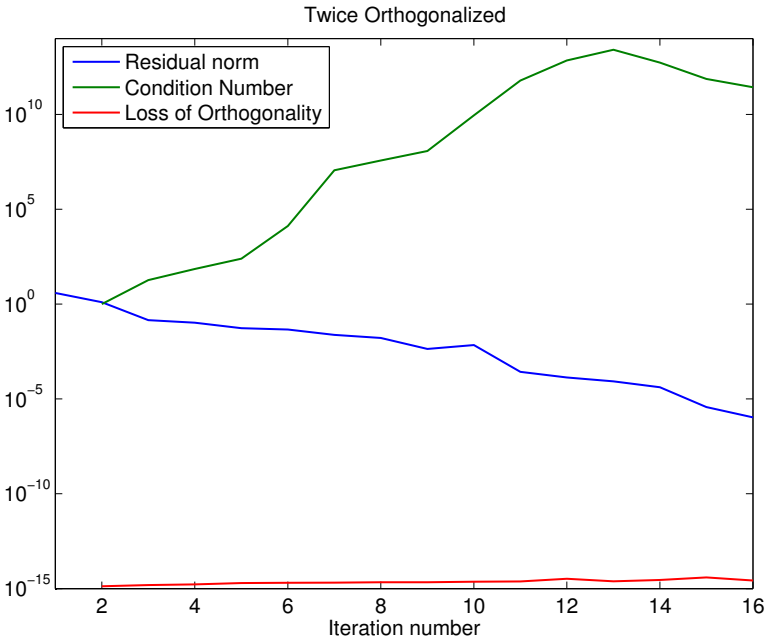
This requires at most only $2(m-1)$ inner product computations, which is less than or equal to the number required for full modified Gram-Schmidt with single orthogonalization except for $m = 2$ or 3, in which case this requires one additional inner product. This always requires at most as many inner products as modified Gram-Schmidt twice.

qrDelete

We next consider the subroutine which updates the QR factorization when the first column is deleted. In this subroutine, we begin with the QR factors for the matrix $A \in \mathbb{R}^{N \times M}$ where we write $A = [a_1\ a_2\ \ldots\ a_M]$, and compute the QR factors of $A' = [a_2\ \ldots\ a_M]$. Noting that we



(a) Single orthogonalization



(b) Twice orthogonalized

Figure 4.1: Convergence behavior and loss of orthogonality in the QR factorization of the least-squares coefficient matrix for H-equation test problem, $m = 10, \omega = 0.9999$

can write $a_i = Qr_i$, where r_i is column i of R, we can write:

$$A' = \begin{bmatrix} a_2 & \dots & a_M \end{bmatrix} = \begin{bmatrix} Qr_2 & \dots & Qr_M \end{bmatrix} = Q \begin{bmatrix} r_2 & \dots & r_M \end{bmatrix} \tag{4.9}$$

Denote entry j of r_i as $r_{i,j}$. As R is upper triangular, $r_{i,j} = 0$ for $j \geq i$. Hence, the matrix $[r_2 \ \dots \ r_M]$ is Upper Hessenberg, and it can be made upper triangular by the application of $M-1$ Givens rotations [22]. Letting $G_1, \dots, G_{M-1}$ be a sequence of rotations which upper triangularizes $[r_2 \ \dots \ r_M]$, we can write:

$$A' = (QG_1^T \dots G_{M-1}^T) \left(G_{M-1} \dots G_1 \begin{bmatrix} r_2 & \dots & r_M \end{bmatrix} \right) = Q_1' \begin{bmatrix} R' \\ 0 \end{bmatrix} \tag{4.10}$$

In this, R' is upper triangular, and as Q is assumed to have orthonormal columns and G_i is orthogonal for $1 \leq i \leq M-1$, the matrix Q_1' has orthonormal columns. Then, writing $Q_1' = [Q' \ q']$, where q' is the last column of Q_1', we have

$$A' = \begin{bmatrix} Q' & q' \end{bmatrix} \begin{bmatrix} R' \\ 0 \end{bmatrix} = Q'R' \tag{4.11}$$

Unlike the Gram-Schmidt process from the previous section, Givens rotations are known to have favorable roundoff properties [22], so it is not expected that special care need be taken to maintain orthogonality in the Q factor for this subroutine. Translating the above into code, the subroutine as included in the solver is shown in the following code fragment.

```
void NOX::Solver::AndersonAcceleration::qrDelete()
{
  int N = qMat.size();
  for (int ii = 0; ii<N-1; ii++){
    double temp = sqrt(rMat(ii,ii+1)*rMat(ii,ii+1)
      + rMat(ii+1,ii+1)*rMat(ii+1,ii+1));
    double c = rMat(ii,ii+1)/temp;
    double s = rMat(ii+1,ii+1)/temp;
    rMat(ii,ii+1) = temp;
    rMat(ii+1,ii+1) = 0;
    for (int jj = ii+2; jj<N; jj++){
      temp = c*rMat(ii,jj) + s*rMat(ii+1,jj);
      rMat(ii+1,jj) = -s*rMat(ii,jj) + c*rMat(ii+1,jj);
      rMat(ii,jj) = temp;
    }
```

```
    *tempVec = *(qMat[ii]);
    tempVec->update(s, *(qMat[ii+1]), c);
    qMat[ii+1]->update(-s, *(qMat[ii]), c);
    *(qMat[ii]) = *tempVec;
  }
  // Shrink the factors.
  qMat.pop_back();
  for (int ii=0; ii<N; ii++){
    for (int jj = 0; jj<N-1; jj++)
      rMat(ii,jj) = rMat(ii,jj+1);
  }
  rMat.reshape(N-1,N-1);
}
```

As the R factor is replicated across each process for MPI implementations, all operations in this routine are local, so this introduces no communication.

4.2.4 Solver Creation

NOX employs factory classes to create specific instances of its derived classes. Given a parameter list specifying details about the object to be created, these factory classes create the requested derived object and returns a reference to its base class. For NOX solvers, the factory class is `NOX::Solver::Factory`. This class has a single member function, `buildSolver`, which which returns a returns a pointer to a `NOX::Solver::Generic` object. Which derived class is created is determined by the parameter list passed in when the function is called. The parameter list is expected to have an entry named "Nonlinear Solver," which prior to the addition of the Anderson acceleration solver accepted values of "Line Search Based," "Trust Region Based," "Inexact Trust Region Based," "Tensor Based," or "Pseudo-Transient." Getting the factory to create the Anderson acceleration solver simply involved changing this function to return a new Anderson acceleration solver object if the "Nonlinear Solver" parameter is set to "Anderson Accelerated Fixed-Point." An example of using a factory object to create an Anderson acceleration solver is shown in the following code fragment:

```
// Create nox parameter list
Teuchos::RCP<Teuchos::ParameterList> nl_params =
  Teuchos::rcp(new Teuchos::ParameterList);
nl_params->set("Nonlinear Solver", "Anderson Accelerated Fixed-Point");
nl_params->sublist("Anderson Parameters").set("Storage Depth", 2);
nl_params->sublist("Anderson Parameters").set("Mixing Parameter", 1.0);
```

```
nl_params->sublist("Anderson Parameters").
  set("Acceleration Start Iteration", 1);
nl_params->sublist("Anderson Parameters").sublist("Preconditioning").
  set("Precondition", false);
nl_params->sublist("Printing").sublist("Output Information").
  set("Details",true);
nl_params->sublist("Printing").sublist("Output Information").
  set("Outer Iteration",true);

// Line search parameters
nl_params->sublist("Line Search").set("Method", "Full Step");

// Create the solver
Teuchos::RCP<NOX::Solver::Generic> solver =
  NOX::Solver::buildSolver(nox_group, combo, nl_params);
```

In this, we first create the parameter list for the nonlinear solver parameters, and we set the parameter "Nonlinear Solver" to "Anderson Accelerated Fixed-Point," which indicates that the Anderson acceleration solver is the derived class that will be created. We then set values for the "Anderson Parameters" sublist which controls the behavior of the solver through the options specified in Section 4.2.1, the "Printing" sublist which controls the output level, and the "Line Search" sublist which determines the details of line search object which will be internally created in the solver. We finally create the solver object by calling the nonmember function `buildSolver`, which itself creates a solver factory and calls the factory's `buildSolver` function. In this function call, the argument `nox_group` is the group object which contains information about the iterate value and function evaluation, and `combo` is the status test which is used to determine success or failure of the iteration.

4.3 Unit Tests

Included with the Anderson acceleration implementation, we have written several unit tests to test various aspects of the solver for both serial and parallel execution. In this section we will overview the performance of the solver for these test problems.

4.3.1 Rosenbrock Test

The first set of unit tests concern solving $F(u) = 0$, where F is the following Rosenbrock function

$$F\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} 10(u_2 - u_1^2) \\ 1 - u_1 \end{pmatrix}. \tag{4.12}$$

The analytic solution to this problem can be easily verified to be $(1, 1)^T$. There are three tests associated with this problem, and in each case we use as the initial iterate $u_0 = (-1.2, 1)^T$. The tests are as follows:

- In the first test, we simply solve the problem by Anderson-2. Since we have an analytic solution for this problem, we check this against the computed solution to ensure that the correct values are computed.
- In the second test, we attempt to solve the equation with Anderson-3 and enable the option to adjust the storage depth to maintain good conditioning of the least-squares coefficient matrix. Note that attempting to append a third vector in the least-squares coefficient matrix will always result in a singular R factor, since the vectors are only 2-dimensional. Hence, the condition number of the R factor after this appending will be infinite, and the storage depth will be adjusted down to 2. This test ensures the proper functionality of the option to adjusting storage depth for conditioning.
- In the last test, we solve the problem by Anderson-2 while utilizing the "Safeguarded Step" line search included in NOX. This tests the functionality of the line search option in the Anderson acceleration solver.

For each of the tests, successfully passing requires the computed solution to match the analytic solution and the solver to converge in the expected number of iterations.

We note here that the function $G(u) = u + F(u)$, which is the fixed-point problem that Anderson is solving, is not contractive near the solution. In fact, at the solution the Jacobian has spectral radius $\rho(G'(u^*)) \approx 18.49$. The convergence of the solver for each of the tests relies on the fact that we use Anderson-2 rather than Anderson-1 in each case. Expressing the Anderson iteration in the form given in Equation (3.61), when the storage depth has reached 2 the projector term $(I - P_k)$, which is in the portion of the expression where contractivity is important, becomes zero, and this is not the case for Anderson-1. All that remains is $u_k - F'(u_k)^{-1}F_k$, which is a Newton iteration, and the higher order expansion term $F'(u_k)^{-1}E_k\gamma^{(k)}$. For an Anderson-2 step, we will have $\|E_k\| = O(\sum_{i=0}^{2}\sum_{j=0}^{i}\|e_{k-i}\|\|e_{k-j}\|)$, so the convergence depends on both the current step and the two prior. This behavior is illustrated in Figure 4.2, which shows the residual history from the first unit test. We first observe an increase in the residual for iterations

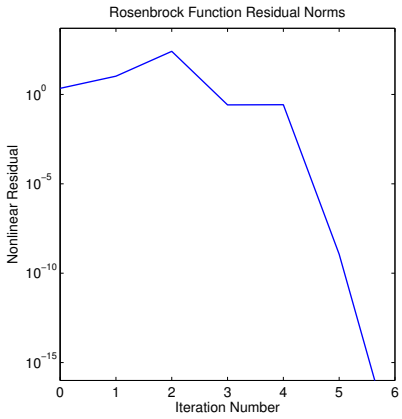


Figure 4.2: Solving the Rosenbrock function by Anderson-2

1 and 2, which correspond to a Picard iteration and and Anderson-1 iteration. The iteration then stagnates for an iteration, and then begins to converge super-linearly as the high error terms from the beginning of the iteration are discarded from memory.

4.3.2 Chandrasekhar H-equation Test

For the next set of unit test, we consider the Chandrasekhar H-equation from Section 3.2.3. There are two unit tests associated with this problem. In the first test, we consider solving the H-equation with $\omega = 0.999$ using Anderson-10. In this test, we solve the problem once, then we call the reset member function with the same initial iterate as the first solve and resolve the same problem. To pass the test, the solver must converge in the proper number of iterations, and compute the same solution with the first solve and the restarted solve. This ensures that the reset member function is functioning properly to prepare the solver for consecutive solves.

We can also use this first unit test to ensure that the solver is implemented in such a way that it properly handles the distributed memory case with MPI communication. In Table 4.1, we see results from solving this problem while varying the number of MPI processes utilized. The vectors are distributed so that there are approximately the same number of entries on each process. In each case there is no observable difference in the residual norm values for the first 10 iterations, and after that point, there is some minor deviation, though the values generally remain very close. When this divergence in residual norms occurs, the condition number of the least-squares coefficient matrix is very large, on the order of 10^{12}. This is large enough so that differences in rounding error could contribute to minor differences in the solution to the least-square problem, which could lead to the minor deviations we observe. In this case, the differences in rounding error likely come from the computation of inner products used in

Table 4.1: Solving H-equation with $\omega = 0.999$ and $N = 400$ by Anderson-10, varying the number of MPI processes utilized

	Number of MPI Processes			
Iteration	1	2	4	8
0	7.48e+0	7.48e+0	7.48e+0	7.48e+0
1	3.87e+0	3.87e+0	3.87e+0	3.87e+0
2	1.26e+0	1.26e+0	1.26e+0	1.26e+0
3	1.43e-1	1.43e-1	1.43e-1	1.43e-1
4	1.05e-1	1.05e-1	1.05e-1	1.05e-1
5	5.35e-2	5.35e-2	5.35e-2	5.35e-2
6	4.60e-2	4.60e-2	4.60e-2	4.60e-2
7	2.39e-2	2.39e-2	2.39e-2	2.39e-2
8	1.63e-2	1.63e-2	1.63e-2	1.63e-2
9	4.30e-3	4.30e-3	4.30e-3	4.30e-3
10	6.88e-3	6.88e-3	6.88e-3	6.88e-3
11	2.67e-4	2.66e-4	2.68e-4	2.66e-4
12	1.35e-4	1.38e-4	1.35e-4	1.36e-4
13	6.66e-5	6.66e-5	6.81e-5	6.82e-5
14	2.11e-5	2.17e-5	2.22e-5	2.29e-5
15	3.15e-6	3.39e-6	3.27e-6	3.35e-6
16	6.38e-7	6.24e-7	6.51e-7	6.45e-7
17	2.43e-7	2.71e-7	2.65e-7	2.40e-7
18	1.10e-7	1.05e-7	1.13e-7	8.14e-8

computing the R factor, and in computing the quantity $Q_k^T F_k$ when solving $Q_k R_k \gamma = F_k$. Since changing the number of processes affects the parallel distribution of the vectors, the order of addition in computing inner products will differ for each choice of MPI process count, and this will lead to slightly different accumulation of rounding error.

In the second unit test, we consider solving the H-equation with $\omega = 0.99$ using Anderson-5. In this, we simply solve the problem with the start of the Anderson acceleration delayed until the fifth iteration. That is, it performs Picard iteration for 5 steps and then beginning performing Anderson acceleration at this point. This test is meant to ensure the proper functionality of the "Acceleration Start Iteration" solver option. The only quantity that is checked in this test is that the iteration converges in the expected number of iterations. The residual norm history produced from this unit test is shown in Figure 4.3.

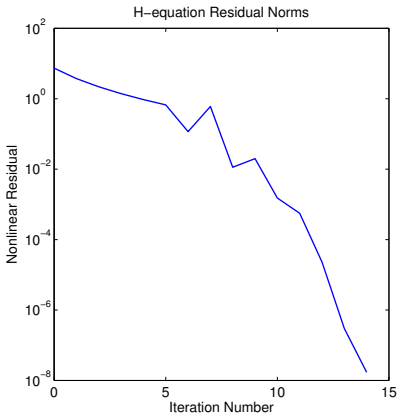


Figure 4.3: Solving H-equation with $\omega = 0.99$ by Anderson-5 with acceleration delayed until iteration 5

4.3.3 1DFEM Test

In the final unit test, we consider a finite element discretization of the following equation describing nonlinear heat conduction

$$-\frac{d^2T}{dx^2} + kT^2 = 0, \quad 0 \le x \le 1, \tag{4.13}$$

$$T(0) = 1, \quad T'(1) = 0.$$

To discretize, we first consider this equation in the weak form

$$\int_0^1 \frac{dT}{dx}\frac{dv}{dx}\,\mathrm{d}x + \frac{dT}{dx}v|_{x=0} + k\int_0^1 T^2 v\,\mathrm{d}x = 0, \tag{4.14}$$

where v is a test function. We seek the solution $\vec{T} = (T_0 \ \dots \ T_N)^T$ at the finite element nodal points $\vec{x} = (x_0 \ \dots \ x_N)^T$. To form the residual, we first impose the condition $T_0 - 1 = 0$. The other components of the residual come from evaluating the weak formulation with the test functions

$$v_i(x) = \begin{cases} \frac{x - x_{i-1}}{x_i - x_{i-1}} & x \in [x_{i-1}, x_i], \\ \frac{x_{i+1} - x}{x_{i+1} - x_i} & x \in [x_i, x_{i+1}], \\ 0 & \text{otherwise}, \end{cases} \tag{4.15}$$

for $i = 1, \dots, N$. This results in a residual equation of the form

$$F(\vec{T}) = \begin{pmatrix} T_0 - 1 \\ A\vec{T} + kB\vec{T}^2 \end{pmatrix} = 0, \tag{4.16}$$

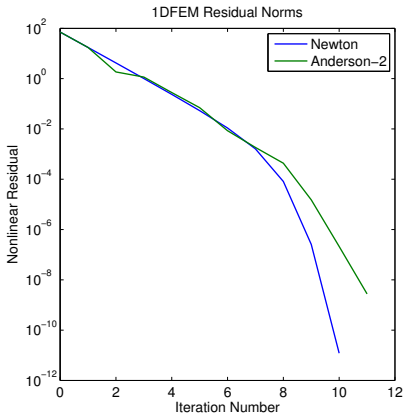


Figure 4.4: Solving the nonlinear heat conduction equation by Newton's method and Anderson-2

where $A, B \in \mathbb{R}^{N\times(N+1)}$ and $\vec{T}^2 = (T_0^2 \ \dots \ T_N^2)^T$.

This test is important since this is the only unit test in which we need to utilize the option to precondition the problem in order for Anderson to converge. As described in Section 3.3, we need to choose the mixing parameter β and preconditioner $M(u)$ such that $\|I + \beta M(u)F'(u)\| < 1$. We have an analytic Jacobian, so for the preconditioner we let $M(u) = F'(u)^{-1}$, which we compute with the Trilinos package ML (see Section B.2.8), and $\beta = -1$. This means that Anderson will be applied to solve the fixed-point problem

$$G(u) = u - F'(u)^{-1}F(u). \tag{4.17}$$

Note that this is simply applying Anderson acceleration to a Newton iteration.

In this test, we simply solve the above nonlinear equation (4.16) by Anderson-2. We consider $k = 1000$, and an initial iterate of $\vec{T}_0 = (1 \ \dots \ 1)^T$. To pass the test, the solver must simply successfully converge in the expected number of iterations. In Figure 4.4, we see a residual plot which compares the results from this test to the same problem solved by Newton's method. We observe that Anderson acceleration performs slightly worse than Newton's method. It appears that the Anderson acceleration residual norm is converging to zero superlinearly, though the order of convergence is likely less than 2, which is the q-order for Newton's method.

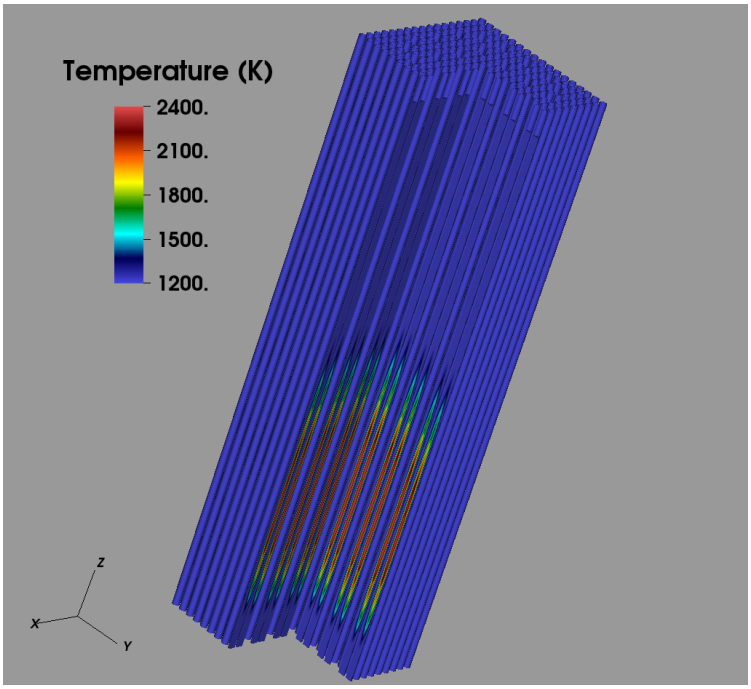
Chapter 5

1D Coupled Model Problem

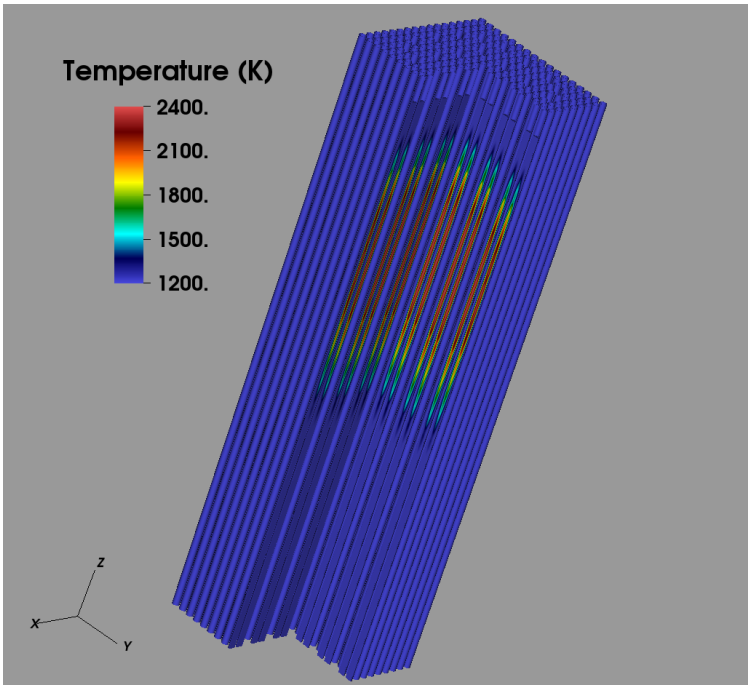
5.1 Introduction

We now return to coupled multiphysics problems in reactor simulation. In this chapter, which expands upon work first presented in [60], we consider a simplified model problem that we have developed in order to assess potential performance gains from Anderson acceleration prior to integrating it into the production level Tiamat code coupling. This problem models several of the interdependent physical processes present in the Tiamat coupling. In particular, we are concerned with the interdependence of the distribution of neutrons throughout the core and the transfer of heat between the fuel and coolant regions.

In order to give any meaningful insight for the production level code coupling, this model problem must recreate several key characteristics that have been observed with regard to the behavior of Picard iteration for solving coupled problems in reactor physics. Beyond Tiamat, Picard iteration has been very widely utilized for solving coupled multiphysics problems in nuclear reactor physics [24, 25, 32, 39, 40, 66]. In most cases, some sort of numerical damping scheme is required in order for the Picard iterations to be convergent. It has been observed that at high enough power, oscillatory behavior arises in the solution as the iteration progresses. It seems that this oscillatory behavior contributes to the poor convergence behavior of Picard iteration which we noted for Tiamat in Section 2.3.3. This oscillatory behavior is illustrated in Figure 5.1. This figure shows the temperature profile for consecutive iterations of a single assembly calculation which couples Insilico for neutronics and AMP for fuel performance with integrated subchannel flow [24]. As the iteration proceeds, there is an oscillatory shift between the lower temperature peaking on the top and the upper temperature peaking at the bottom, and the iteration fails to converge. This oscillation is equally observed in couplings between various codes for 3×3 "mini-assembly" calculation in [25] as well as for larger single assembly to full core problems in [24,39]. It seems that this behavior is inherent to the physical processes



(a) Lower temperature peaking



(b) Upper temperature peaking

Figure 5.1: Oscillatory temperature shift in Insilico/AMP coupling

considered, and does not depend strongly on the codes being coupled or the number of fuel rods. The oscillatory behavior seems primarily to be an axial phenomenon, so we attempt to utilize a one-dimensional model in order to recreate and analyze this behavior. We then use this model to evaluate the potential of Anderson acceleration as an alternative solution method to Picard iteration. While Anderson acceleration has become widely utilized to accelerate Picard iterations in other contexts, particularly SCF iterations for electronic structures calculations [26,35,47–49], its effectiveness in the context of coupled multiphysics problems in LWRs has been largely untested to this point.

In this chapter, we first describe the physical models that we considered and the discretized problem in Section 5.2. We then describe how Picard iteration and Anderson acceleration are used to solve this problem in Section 5.3. Lastly, in Section 5.4 we verify that this model problem recreates the expected behavior for Picard iteration and then compare the performance of Picard iteration with Anderson acceleration.

5.2 Physical Models and Discretization

In this simplified model problem we simulate the interdependence between the neutron distribution and the temperature distributions within the fuel and coolant regions in a single fuel rod of height L. We denote the cross sectional area at a given axial height as $A(z)$. This area comprises of a circular fuel region with radius R_f inscribed within a square coolant region. We attempt to capture the behavior of the physical system by a one-dimensional model, so we consider equations which describe the axial behavior of the neutron distribution and fuel and coolant temperatures, which we treat as homogenous at a given axial height.

For reactor analysis, the distribution of neutrons is governed by the Boltzmann transport equation (Equation (2.9)), and codes for this purpose solve some approximation of this equation. As the purpose of this simplified study concerns convergence behavior rather than simulation accuracy, we consider the fairly low-accuracy one-group diffusion equation

$$-\nabla \cdot D(\vec{r},T)\nabla\phi(\vec{r}) + (\Sigma_t(\vec{r},T) - \Sigma_s(\vec{r},T))\phi = \frac{1}{k}\nu\Sigma_f(\vec{r},T)\phi(\vec{r}), \tag{5.1}$$

where ϕ is the neutron scalar flux, Σ are the material cross sections, D is the diffusion coefficient, ν is the mean number of neutrons per fission, and k is the dominant eigenvalue. The cross section dependence on $\vec{r}$ and T indicates that these quantities depend on both the material at position r and its temperature. At the radial boundaries, we assume reflective boundary conditions. To obtain a one-dimensional equation, we integrate (5.1) over the radial area $A(z)$. Using the divergence theorem on the first term, the radial derivative terms vanish due to the reflective

conditions at the radial boundaries. This leaves

$$-\int_{A(z)} \frac{\partial}{\partial z} D(\vec{r}) \frac{\partial \phi(\vec{r})}{\partial z} \, \mathrm{d}A + \int_{A(z)} (\Sigma_t(\vec{r}) - \Sigma_s(\vec{r}))\phi(\vec{r}) \, \mathrm{d}A = \frac{1}{k} \int_{A(z)} \nu\Sigma_f(\vec{r})\phi(\vec{r}) \, \mathrm{d}A. \tag{5.2}$$

We then define the radially homogenized cross sections

$$\bar{\Sigma}(z) = \frac{\int_{A(z)} \Sigma(\vec{r})\phi(\vec{r}) \, \mathrm{d}A}{\int_{A(z)} \phi(\vec{r}) \, \mathrm{d}A}, \tag{5.3}$$

radially homogenized diffusion coefficient

$$\bar{D}(z) = \frac{\int_{A(z)} \phi(\vec{r}) \, \mathrm{d}A}{\int_{A(z)} D(\vec{r})^{-1}\phi(\vec{r}) \, \mathrm{d}A} = \frac{1}{3\bar{\Sigma}_t(z)}, \tag{5.4}$$

and the radially integrated scalar flux

$$\bar{\phi}(z) = \int_{A(z)} \phi(\vec{r}) \, \mathrm{d}A. \tag{5.5}$$

Substituting the radially homogenized cross sections and integrated scalar flux, and replacing $D(\vec{r})$ with the radially homogenized quantity, (5.2) reduces to

$$-\frac{d}{dz}\bar{D}\frac{d\bar{\phi}}{dz} + (\bar{\Sigma}_t - \bar{\Sigma}_s)\bar{\phi} = \frac{1}{k}\nu\bar{\Sigma}_f\bar{\phi}\,. \tag{5.6}$$

The neutron distribution affects the other physical systems primarily through heat generated from fission. The linear heat generation rate is given by

$$q'(z) = \int_{A(z)} E_f\Sigma_f(\vec{r})\phi(\vec{r}) \, \mathrm{d}A = E_f\bar{\Sigma}_f(z)\bar{\phi}(z)\,, \tag{5.7}$$

where E_f is the energy released per fission. As Equation (5.6) represents an eigenvalue problem, the eigenfunction has no explicit magnitude, and we choose to set the average linear power to a prescribed value P^*

$$\frac{1}{L}\int_0^L E_f\bar{\Sigma}_f\bar{\phi} \, \mathrm{d}z = P^*\,. \tag{5.8}$$

We will suppose that there is a vacuum at the upper and lower boundaries, so we impose the following Marshak boundary conditions

$$\bar{\phi}(0) - 2\bar{D}(0)\frac{d\bar{\phi}}{dz}(0) = 0, \tag{5.9}$$

$$\bar{\phi}(L) + 2\bar{D}(L)\frac{d\bar{\phi}}{dz}(L) = 0. \tag{5.10}$$

To discretize this system, we employ a finite difference approximation which results in the following system of equations

$$\phi_0 \left(\frac{1}{2} + \frac{D_1 + D_0}{2(z_1 - z_0)} + \frac{z_1 - z_0}{2}(\Sigma_{t,0} - \Sigma_{s,0}) \right) - \phi_1 \frac{D_1 + D_0}{2(z_1 - z_0)} = \frac{1}{k_{eff}} \phi_0 \frac{z_1 - z_0}{2} \nu\Sigma_{f,0}, \quad (5.11)$$

$$- \phi_{i-1} \frac{D_i + D_{i-1}}{2(z_i - z_{i-1})} + \phi_i \left(\frac{D_i + D_{i-1}}{2(z_i - z_{i-1})} + \frac{D_{i+1} + D_i}{2(z_{i+1} - z_i)} + \frac{z_{i+1} - z_{i-1}}{2}(\Sigma_{t,i} - \Sigma_{s,i}) \right)$$
$$- \phi_{i+1} \frac{D_{i+1} + D_i}{2(z_{i+1} - z_i)} = \frac{1}{k_{eff}} \phi_I \frac{z_{i+1} - z_{i-1}}{2} \nu\Sigma_{f,i}, \quad 1 \le i \le N-1, \quad (5.12)$$

$$- \phi_{N-1} \frac{D_N + D_{N-1}}{2(z_N - z_{N-1})} + \phi_N \left(\frac{1}{2} + \frac{D_N + D_{N-1}}{2(z_N - z_{N-1})} + \frac{z_N - z_{N-1}}{2}(\Sigma_{t,N} - \Sigma_{s,N}) \right) =$$
$$\frac{1}{k_{eff}} \phi_N \frac{z_N - z_{N-1}}{2} \nu\Sigma_{f,N}. \quad (5.13)$$

Combining (5.11), (5.12), and (5.13), we can write the system for $\phi = (\phi_0, \phi_1, \ldots, \phi_N)^T$ in the following form

$$A\phi = \frac{1}{k_{eff}} B\phi \quad (5.14)$$

where A is tridiagonal and B is diagonal. In implementation, we solve this problem as the generalized eigenproblem $B\phi = k_{eff} A\phi$ using the Generalized Davidson eigensolver included in the Trilinos package Anasazi (see Appendix B.2.6).

The solution eigenvectors returned by the Anasazi solvers will have norm one, so a scaling will be required to enforce the power normalization condition (5.8). To do this, we need to approximate the integral $\frac{1}{L}\int_0^L E_f \Sigma_f \tilde{\phi}\, \mathrm{d}z$, where $\tilde{\phi}$ is the unit norm solution returned by the solver. We simply approximate this value using composite trapezoidal rule, which gives

$$I = \frac{1}{L}\int_0^L E_f \Sigma_f \tilde{\phi}\, \mathrm{d}z \approx \frac{1}{2L} \sum_{j=1}^{N} (z_j - z_{j-1})(E_f \Sigma_{f,j-1} \tilde{\phi}_{j-1} + E_f \Sigma_{f,j} \tilde{\phi}_j).$$

Given this integral, computing $\phi = P^* \tilde{\phi}/I$ scales the eigenvector to satisfy the desired normalization. Then, because the cross sections are homogenized over radial directions, we have the linear heat generation rate $q' = E_f \Sigma_f \phi$.

The last component to this set of physics is computation of cross sections. We compute pin cell homogenized cross section data using a linear fit on reference data generated by the Scale module XSProc [1]. We assume isotropic scattering, so the diffusion coefficient is computed by

Table 5.1: 1D model problem cross sections at various fuel temperatures with constant coolant temperature 565K

Fuel Temp (K)	Σ_t	Σ_s	Σ_f	$\nu\Sigma_f$
500	0.655322	0.632804	0.0115249	0.0283528
1000	0.654535	0.631904	0.0114019	0.0280547
1500	0.653949	0.631236	0.0113002	0.0278078

Table 5.2: 1D model problem cross sections at various fuel and coolant temperatures

Fuel Temp (K)	Coolant Temp (K)	Σ_t	Σ_s	$\nu\Sigma_f$
565	565	0.655302	0.632765	0.0283063
1565	565	0.653976	0.631252	0.0277754
565	605	0.61046	0.589171	0.0265561

$D = \frac{1}{3\Sigma_t}$. The reference data for these computations are given in Tables 5.1 and 5.2. To compute cross sections in the case with constant coolant properties, we perform a piecewise linear interpolation using the three reference temperatures. For temperatures beyond the bounds of the reference temperatures, we extrapolate by simply extending the linear fit past the reference temperatures. This does not seem unreasonable, as the cross sections seem to have a rather linear dependence on fuel temperature in this range. Next, in the case with variable coolant temperature cross sections are computed by the linear fit

$$\Sigma(T_f, T_w) = \Sigma(565, 565) + \frac{\Sigma(1565, 565) - \Sigma(565, 565)}{1000}(T_f - 565) + \frac{\Sigma(565, 605) - \Sigma(565, 565)}{40}(T_w - 565). \quad (5.15)$$

In Table 5.2, Σ_f is not provided, but in Table 5.1 we see that ν is approximately constant and equal to 2.46, so we let $\Sigma_f = \nu\Sigma_f/2.46$.

Next, we let the fuel temperature be governed by a simple relation derived from Newton's Law of Cooling [5], which states

$$q'' = h(T_f - T_w), \quad (5.16)$$

where q'' is the heat flux and h is the heat transfer coefficient. To relate q'' and q' we consider the differential length of the rod between axial heights z and $z + dz$. This corresponds to a heat transfer surface area of $2\pi R_f dz$. In this model, we assume all the power generated in the fuel is deposited radially into the coolant, so the power transferred through this area is given by $q'dz$. Hence, the heat flux is $q'' = \frac{q'}{2\pi R_f}$. Making this substitution gives the relation

$$2\pi R_f h(T_f - T_w) = q'. \quad (5.17)$$

Discretization of this equation is trivial, and we simply evaluate the above expression at each axial height.

The last set of physics that we consider is coolant flow. We will consider two cases for the coolant temperature. In the first, we let the fluid temperature be axially constant. We assume a known incoming coolant temperature, and let this be the coolant temperature over the height of the reactor. That is, given inflow temperature T_{IN}, we let $T_{w,i} = T_{IN}$ for each node i. In this case, there is a two-way coupling between the fuel temperature and neutronics. The fuel temperature determines the cross sections, which affects the solution of (5.6). The neutronics in turn determines the power distribution, which affects the fuel temperature through (5.17).

In the second case, we integrate a simple flow model to determine the coolant temperature. In this model, we assume that flow is only in the axial direction with a constant mass flow rate $\dot{m}$. We again consider a differential length dz about z. We let the change in temperature from z to $z + dz$, dT_w, be governed by the simplified steady-flow thermal energy equation [5], which states

$$\dot{m} c_p(T_w) dT_w = q, \tag{5.18}$$

where q is the power generation in this interval. The power transferred to the coolant is again $q = q' dz$, so we let the axial profile of the coolant temperature be governed by the differential equation

$$\dot{m} c_p(T_w) \frac{dT_w}{dz} = q'. \tag{5.19}$$

Given the inflow temperature T_{IN}, this can be solved to give the axial coolant temperature profile. We let specific heat values be given by a piecewise linear interpolation using computed data at reference temperatures and pressures given in [18]. Integrating this flow model now results in a three way coupling between the neutron distribution, fuel properties, and coolant properties. The coolant and fuel temperatures both depend on neutronics through heat generation from fission. There is also a relationship between the coolant and fuel temperature imposed by (5.17). The neutron distribution depends on both the fuel and fluid temperatures through the cross section temperature dependence.

We derive the discretized form of this equation by integrating over $[z_{j-1}, z_j]$ and approximating by midpoint rule, which gives the approximation

$$\dot{m} c_p(T_{w,j-1/2}) \frac{dT_{w,j-1/2}}{dz} = q'_{j-1/2},$$

where the subscript $j-1/2$ indicates the values are evaluated at $z_{j-1/2} = \frac{1}{2}(z_{j-1} + z_j)$. We only

wish to involve quantities evaluated at mesh nodes, so we introduce the following approximations

$$\frac{dT_{w,j-1/2}}{dz} = \frac{T_{w,j} - T_{w,j-1}}{z_j - z_{j-1}},$$

$$T_{w,j-1/2} = \frac{T_{w,j} + T_{w,j-1}}{2},$$

$$q'_{j-1/2} = \frac{q'_j + q'_{j-1}}{2}.$$

This gives the discretized equation

$$\dot{m}c_p\left(\frac{T_{w,j} + T_{w,j-1}}{2}\right)\frac{T_{w,j} - T_{w,j-1}}{z_j - z_{j-1}} = \frac{q'_j + q'_{j-1}}{2}. \tag{5.20}$$

This, combined with the boundary condition $T_{w,0} = T_{IN}$ defines the system of equations to solve for the coolant temperature. It is nonlinear, and in implementation this will be solved by JFNK. The residual for this system is defined by

$$F(T)_j = \begin{cases} T_{w,0} - T_{IN}, & j = 0, \\ \dot{m}c_p([T_{w,j} + T_{w,j-1}]/2)(T_{w,j} - T_{w,j-1}) - (z_j - z_{j-1})(q'_j + q'_{j-1})/2, & 1 \le j \le N. \end{cases}$$

This system is simple to differentiate analytically, and its Jacobian is given by

$$F'(T)_{i,j} = \begin{cases} 1, & i = j = 0, \\ \dot{m}(-c_p(\frac{T_{w,i}+T_{w,i-1}}{2}) + c'_p(\frac{T_{w,i}+T_{w,i-1}}{2})[T_{w,j} - T_{w,j-1}]/2), & 1 \le i \le N, j = i-1, \\ \dot{m}(c_p(\frac{T_{w,i}+T_{w,i-1}}{2}) + c'_p(\frac{T_{w,i}+T_{w,i-1}}{2})[T_{w,j} - T_{w,j-1}]/2), & 1 \le i \le N, j = i, \\ 0, & \text{otherwise.} \end{cases}$$

As specific heat evaluations will be computed via a table lookup, we prefer not to involve its derivatives. For this reason, we implement a Jacobian-free method for solving this problem. To precondition the linear JFNK solves, we approximate the Jacobian by disregarding the terms involving specific heat temperature derivatives. For a realistic problem, the coolant temperature should only increase approximately 30 degrees over the height of the reactor, so the difference between neighboring coolant temperatures should be fairly small given a moderately fine mesh and these terms should be somewhat negligible. The preconditioner is then given by the inverse

of the matrix defined by

$$M(T_w)_{i,j} = \begin{cases} 1, & i = j = 0, \\ -\dot{m}c_p([T_{w,i} + T_{w,i-1}]/2), & 1 \le i \le N, j = i-1, \\ \dot{m}c_p([T_{w,i} + T_{w,i-1}]/2), & 1 \le i \le N, j = i, \\ 0, & \text{otherwise.} \end{cases}$$

5.3 Coupling Algorithms

We seek $\bar{\phi}, k, T_f$, and T_w such that Equations (5.6), (5.8), (5.17), and (5.19) are simultaneously satisfied. In this section, we describe how Picard iteration and Anderson acceleration can be utilized to solve this problem.

5.3.1 Picard Iteration

First, we will consider the two-way coupling with constant coolant properties listed in Table 5.1. Again, Picard iteration proceeds by cycling between solution of individual physical systems in some order. In this one-dimensional study, we will only consider a block Gauss-Seidel type fixed-point map. For the two-way coupling we alternate between solving the neturonics system and solving the fuel temperature system. This scheme implemented as follows.

- Given T_f^n, compute the cross sections $\Sigma(T_f^n)$.
- Solve the k-eigenvalue problem

$$-\frac{d}{dz}D(T_f^n)\frac{d\phi^{n+1/2}}{dz} + \left[\Sigma_t(T_f^n) - \Sigma_t(T_f^n)\right]\phi^{n+1/2} = \frac{1}{k_{eff}}\nu\Sigma_f(T_f^n)\phi^{n+1/2}.$$

- Apply the power normalization and compute linear power

$$\phi^{n+1} = \frac{P^*\phi^{n+1/2}}{\frac{1}{L}\int_0^L E_f\Sigma_f(T_f^n)\phi^{n+1/2}\,\mathrm{d}z},$$

$$q'_{n+1} = E_f\Sigma_f(T_f^n)\phi^{n+1}.$$

- Compute the updated temperature

$$T_f^{n+1} = T_w + \frac{q'_{n+1}}{2\pi Rh}.$$

As was previously stated, at high enough power levels this sort of iteration suffers poor convergence or possibly divergence due to oscillatory error modes which arise in the temperature and flux distributions. This has generally been addressed by utilizing a damping on the either the temperature or power update, and we choose to damp the temperature. That is, we replace the temperature update with the two-step process

$$\begin{aligned} T_f^{n+1/2} &= T_w + \frac{q'_{n+1}}{2\pi Rh}, \\ T_f^{n+1} &= (1-\omega)T_f^n + \omega T_f^{n+1/2}. \end{aligned}$$

We refer to ω as the damping parameter, and it has been noted that for this sort of iteration, the damping parameter which results in the fastest rate of convergence generally falls between 0.3 and 0.6.

Integrating the flow model results in only a slight change to this process. Following the example of the high fidelity coupling, we solve for the fuel and coolant temperatures as a tightly coupled system. Given a power distribution, we independently solve (5.19) for the coolant temperature, then use this in evaluating (5.17). When applying damping, we damp both temperatures using the same damping parameter. This iteration then proceeds as follows.

- Given T_f^n and T_w^n, compute the cross sections $\Sigma(T_f^n, T_w^n)$.
- Solve the k-eigenvalue problem

$$-\frac{d}{dz}D(T_f^n, T_w^n)\frac{d\phi^{n+1/2}}{dz} + \left[\Sigma_t(T_f^n, T_w^n) - \Sigma_t(T_f^n, T_w^n)\right]\phi^{n+1/2} = \frac{1}{k_{eff}}\nu\Sigma_f(T_f^n, T_w^n)\phi^{n+1/2}.$$

- Apply the power normalization and compute linear power

$$\phi^{n+1} = \frac{P^*\phi^{n+1/2}}{\frac{1}{L}\int_0^L E_f\Sigma_f(T_f^n)\phi^{n+1/2}\,\mathrm{d}z},$$

$$q'_{n+1} = E_f\Sigma_f(T_f^n)\phi^{n+1}.$$

- Solve for the coolant and fuel temperatures

$$\dot{m}c_p(T_w^{n+1/2})\frac{dT_w^{n+1/2}}{dz} = q'_{n+1},$$

$$T_f^{n+1/2} = T_w^{n+1/2} + \frac{q'_{n+1}}{2\pi Rh}.$$

- Apply the temperature damping

$$\begin{aligned} T_w^{n+1} &= (1-\omega)T_w^n + \omega T_w^{n+1/2}, \\ T_f^{n+1} &= (1-\omega)T_f^n + \omega T_f^{n+1/2}. \end{aligned}$$

In the iteration, we attempt to converge each of the temperatures, the scalar flux, and the eigenvalue.

5.3.2 Anderson Acceleration

We wish to use Anderson acceleration to improve the convergence rates, and potentially robustness, of iterations like those described in the previous section, which may be very slow for realistic problems of interest, like Tiamat, and require ad hoc damping factors. To utilize Anderson acceleration to solve this model problem, we need only to define the fixed-point map to provide to the solver. The maps will be defined from the Picard iterations, but there is some flexibility in how to implement this. We can choose to iterate on some subset or all of the state variables. We will choose to expose the temperature vectors to the Anderson solver and embed the neutronics application inside the fixed-point map evaluation. First, we will define $G(T_f)$ to be the fixed-point map for the two-way coupling. This function is evaluated as follows.

- Given T_f, compute the cross sections $\Sigma(T_f)$.
- Solve the k-eigenvalue problem

$$-\frac{d}{dz}D(T_f)\frac{d\tilde{\phi}}{dz} + \left[\Sigma_t(T_f) - \Sigma_t(T_f)\right]\tilde{\phi} = \frac{1}{k_{eff}}\nu\Sigma_f(T_f)\tilde{\phi}.$$

- Apply the power normalization and compute linear power

$$\phi = \frac{P^*\tilde{\phi}}{\frac{1}{L}\int_0^L E_f\Sigma_f(T_f)\phi\,\mathrm{d}z},$$

$$q' = E_f\Sigma_f(T_f)\phi.$$

- Compute the updated temperature

$$G(T_f) = T_w + \frac{q'}{2\pi Rh}.$$

Note that we can then represent the Picard iteration as $T_f^{n+1} = (1-\omega)T_f^n + \omega G(T_f^n)$. For the three-way coupling, the fixed-point map will be a function of both temperatures. We will call

this function $H\begin{pmatrix} T_f \\ T_w \end{pmatrix}$, and it is evaluated as follows.

- Given T_f, T_w, compute the cross sections $\Sigma(T_f, T_w)$.
- Solve the k-eigenvalue problem

$$-\frac{d}{dz}D(T_f)\frac{d\tilde{\phi}}{dz} + \left[\Sigma_t(T_f) - \Sigma_t(T_f)\right]\tilde{\phi} = \frac{1}{k_{eff}}\nu\Sigma_f(T_f)\tilde{\phi}.$$

- Apply the power normalization and compute linear power

$$\phi = \frac{P^*\tilde{\phi}}{\frac{1}{L}\int_0^L E_f\Sigma_f(T_f)\phi\,\mathrm{d}z},$$

$$q' = E_f\Sigma_f(T_f)\phi.$$

- Solve for $\tilde{T}_w$ and $\tilde{T}_f$ by

$$\dot{m}c_p\frac{d\tilde{T}_w}{dz} = q',$$

$$\tilde{T}_f = \tilde{T}_w + \frac{q'}{2\pi Rh}.$$

- Define H

$$H\begin{pmatrix} T_f \\ T_w \end{pmatrix} = \begin{pmatrix} \tilde{T}_f \\ \tilde{T}_w \end{pmatrix}.$$

Again, we can represent the Picard iteration as

$$\begin{pmatrix} T_f^{n+1} \\ T_w^{n+1} \end{pmatrix} = (1-\omega)\begin{pmatrix} T_f^n \\ T_w^n \end{pmatrix} + \omega H\begin{pmatrix} T_f^n \\ T_w^n \end{pmatrix}.$$

5.4 Numerical Results

We now present numerical results for solving this 1D model problem with Picard iteration and Anderson acceleration. For these tests, we select physical parameters to be approximately realistic. We let the dimensions of the fuel rod be given by $L = 360\,\mathrm{cm}$ and $R_f = 0.5\,\mathrm{cm}$. We set the pressure of the system to $15.5\,\mathrm{MPa}$, set a 100% power baseline at $P^* = 200\,\frac{\mathrm{W}}{\mathrm{cm}}$, let the energy released per fission be $E_f = 191.4\,\mathrm{MeV}$, and let the heat transfer coefficient be $h = 0.2\,\frac{\mathrm{W}}{\mathrm{m}^2\,\mathrm{K}}$. For the flow model, we assume an incoming coolant temperature of $T_{IN} = 565\,\mathrm{K}$

and a mass flow rate of $\dot{m} = 0.3\,\frac{\text{kg}}{\text{s}}$. We discretize each of the systems on the same evenly spaced mesh consisting of $N = 201$ axial nodes. In general, each of the systems need not be solved on the same mesh, but differing meshes only increases the complexity of data transfers between physical systems and should not significantly affect the convergence behavior of the coupled system. We solve the generalized eigenproblem resulting from discretization of (5.6) by the Generalized Davidson solver in the Trilinos package Anasazi (see Section B.2.6), and we solve the nonlinear system from (5.19) using a NOX JFNK solver. For the Anderson iterations, we utilize the NOX Anderson acceleration solver described in the previous chapter. To terminate each iteration, we require each of $\bar{\phi}, k, T_f$, and T_w to be sufficiently converged by requiring

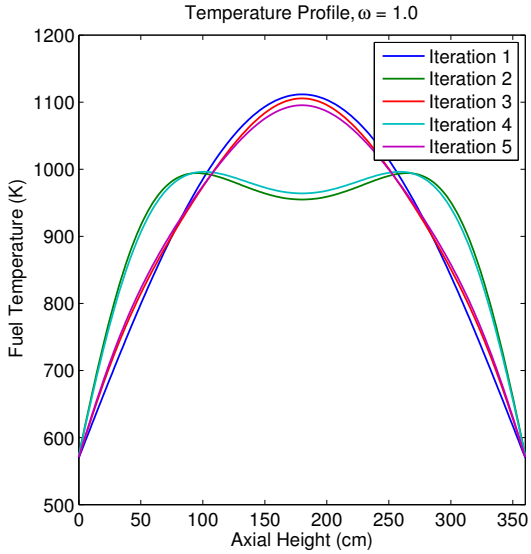
$$\begin{array}{ll} \frac{\|\bar{\phi}_{j+1}-\bar{\phi}_j\|}{\|\bar{\phi}_0\|} < \tau, & \frac{|k_{j+1}-k_j|}{k_0} < \tau, \\ \frac{\|T_{f,j+1}-T_{f,j}\|}{\|T_{f,0}\|} < \tau, & \frac{\|T_{w,j+1}-T_{w,j}\|}{\|T_{w,0}\|} < \tau, \end{array} \tag{5.21}$$

where τ is some tolerance. For Anderson, $\bar{\phi}$ and k are the values computed internally in the evaluation of the fixed-point map. In the following tests, we let $\tau = 10^{-4}$, and we begin with initial fuel and coolant profiles identically equal to T_{IN}.

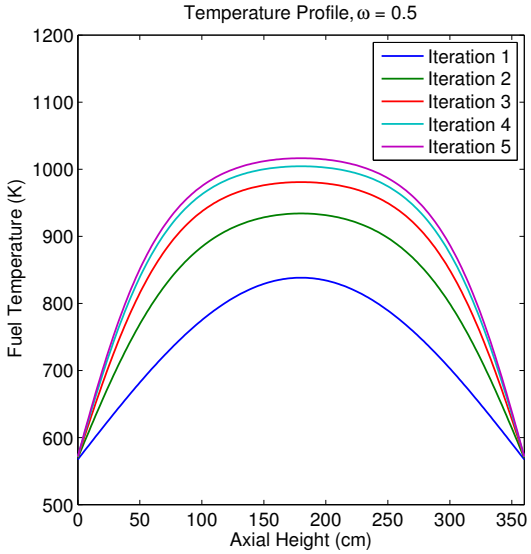
5.4.1 Picard Results

We first implemented the Picard couplings to verify that the one dimensional model in fact recreates the oscillatory behavior observed in the high-fidelity Insilico/AMP coupling. In the case where coolant properties are held constant, the problem is symmetric about the center height, and we expect the iterations oscillate between a center peaked distribution and a bimodal distribution. In Figures 5.2a and 5.2b, we see fuel temperature profiles resulting from the one dimensional model with constant coolant properties without and with damping. We see that without sufficient damping we in fact recreate this oscillatory behavior. When the flow model is integrated, the rise in the coolant temperature as it flows through the reactor introduces asymmetry into the problem. As a result, we expect oscillation between lower-peaked and upper-peaked distributions, as was displayed in Figure 5.1. Figure 5.2c shows the temperature behavior for the one dimensional model without temperature damping, and we again see that this problem retains sufficient physics to recreate the oscillatory behavior. Figure 5.2d shows a similar effect from damping as what is observed in the two way coupling.

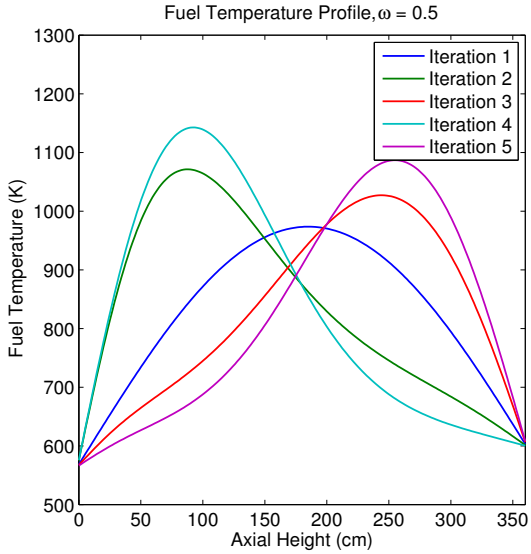
In Figure 5.3, we see the effect varying the damping parameter has on the number of Picard iterations to convergence. The dependence of the necessary and ideal damping parameters on power are very similar to what is observed in the high fidelity couplings. The three-way coupling generally requires more damping, but this increased sensitivity to the level of damping should not too surprising. In Table 5.2, we see that the cross sections have a significantly higher dependence on the coolant temperature than the fuel temperature, and the coolant temperature



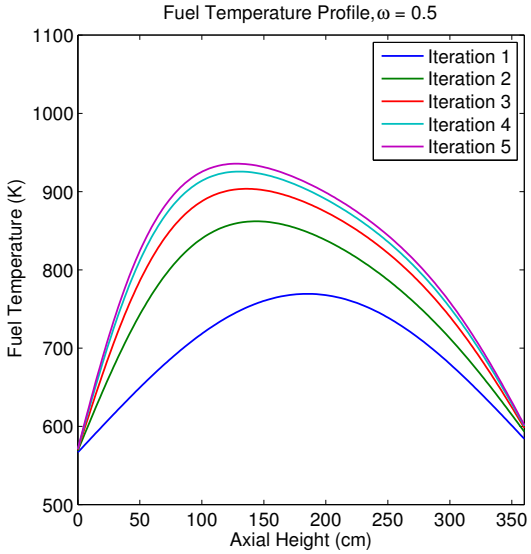
(a) Two-way coupling, no damping



(b) Two-way coupling, with damping

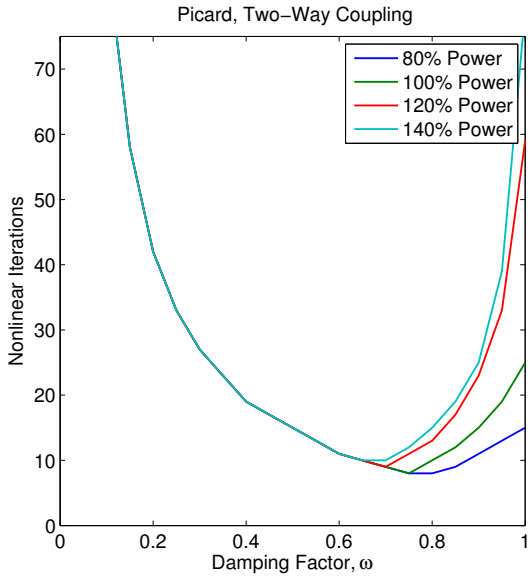


(c) Three-way coupling, no damping

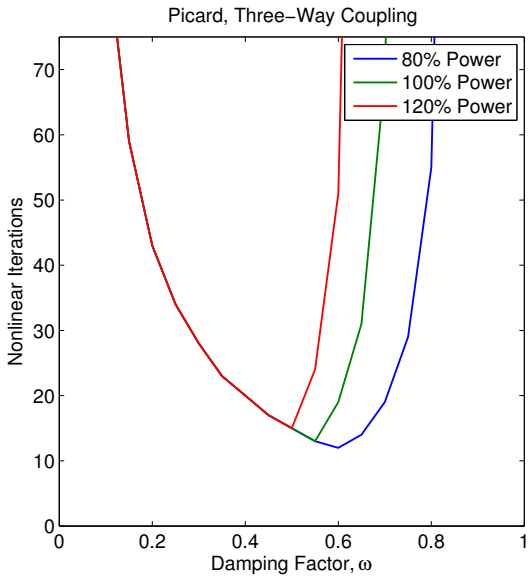


(d) Three-way coupling, with damping

Figure 5.2: Fuel temperature behavior for the model problem, without and with damping



(a) Two-way coupling



(b) Three-way coupling

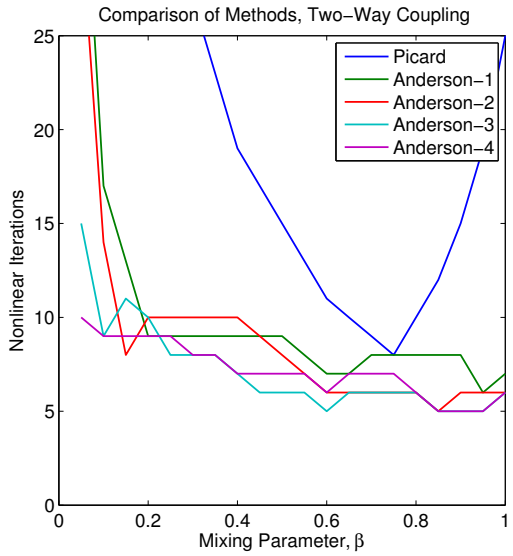
Figure 5.3: Picard iterations to convergence, varying damping factor

depends strongly on the scalar flux through (5.19). Hence the coolant temperature and neutron distribution are ratherly tightly coupled in these models. In general, we seem to recreate the convergence behavior of the Insilico/AMP coupling, namely oscillatory behavior in the solution vectors and strong dependence of iteration counts on both the damping parameter and power level. This suggests that this model should be a good surrogate for higher fidelity couplings, and our results for Anderson acceleration for this problem should provide some insight on its behavior for such high fidelity coupling.

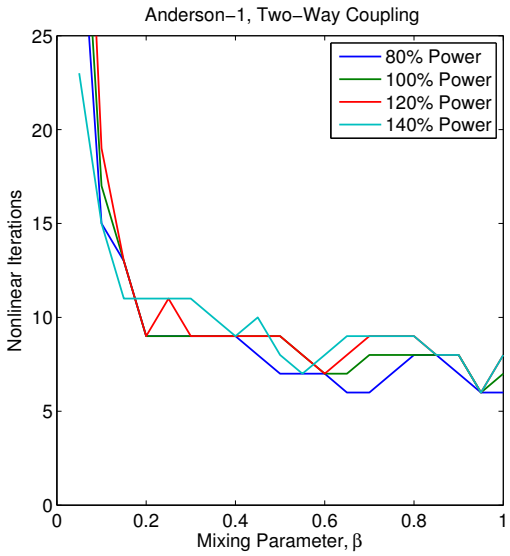
5.4.2 Anderson Results

In Figures 5.4 and 5.5 illustrate the convergence behavior of Anderson acceleration for the two-way coupling and the three-way coupling while varying the power levels, mixing parameter and storage depth. In most cases, the Anderson iterations do as well or better than the optimally damped Picard iterations. In addition to at worst a modest improvement in terms of number of iterations to convergence, we observe a significant improvement with respect to robustness. Each of the Anderson iterations converges regardless of the mixing parameter, and in all cases the number of iterations to convergence is generally insensitive to the choice of mixing parameter. As expected, the iterations counts worsen a bit as the mixing parameter approaches zero, but away from zero, for the most part there is no obvious relation between the iteration counts and the mixing parameter. While the Picard iterations counts generally increase rapidly for damping parameters away from the optimal value, the Anderson iterations do not have such an observable dependence on the mixing parameter. As a result, the mixing parameter does not need to be tuned to obtain acceptable performance.

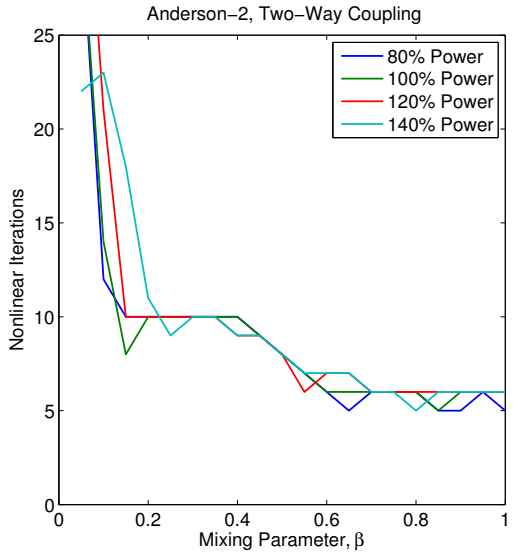
Next, we have noted that the optimal damping parameter for the Picard iterations depends rather strongly on the power level. In addition to this, the number of iterations at the optimal damping level increases with the power. Conversely, the number of Anderson iterations to convergence for a given mixing parameter does not depend strongly on the power level. In terms of the choice of storage depth, the results agree well with past experience. In most cases, Anderson-2 converges faster than Anderson-1, and Anderson-3 is faster than Anderson-2, but less so. Anderson-3 seems moderately more stable with respect to the mixing parameter than Anderson-2, and improvements seem to stagnate beyond this point. Then, ignoring memory requirements, Anderson-3 or Anderson-4 seems optimal for this problem. Lastly, we noted that the three-way coupling generally required significantly more damping to obtain convergence when compared to the two-way coupling as a result of the strong dependence of the cross sections on the coolant temperature. The Anderson iterations generally handle this stronger coupling better, as in most cases additional damping is not necessary and the optimal number of iterations are fairly comparable between the two mappings.



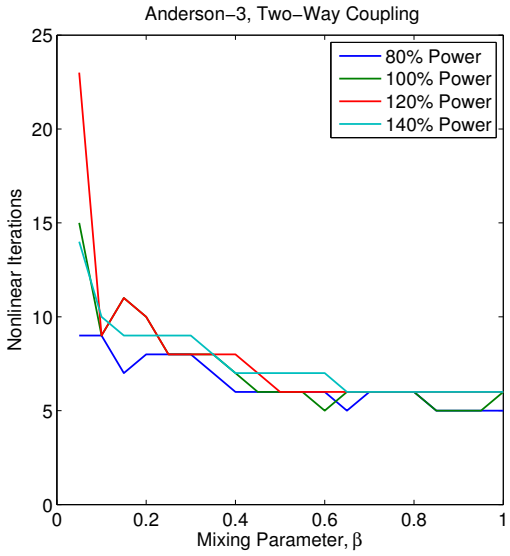
(a) Two-way coupling comparison, 100% power



(b) Two-way coupling, Anderson-1 iterations

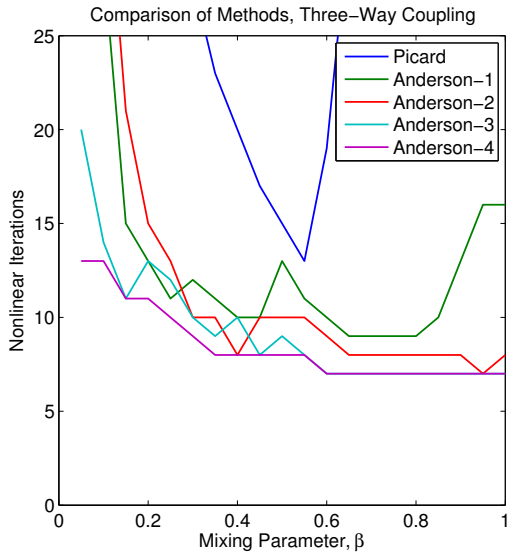


(c) Two-way coupling, Anderson-2 iterations

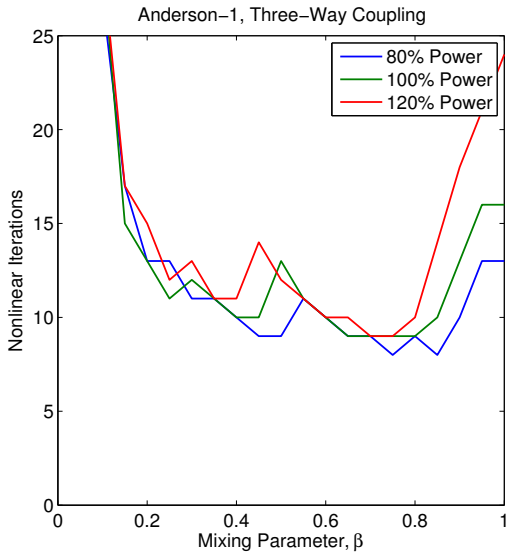


(d) Two-way coupling, Anderson-3 iterations

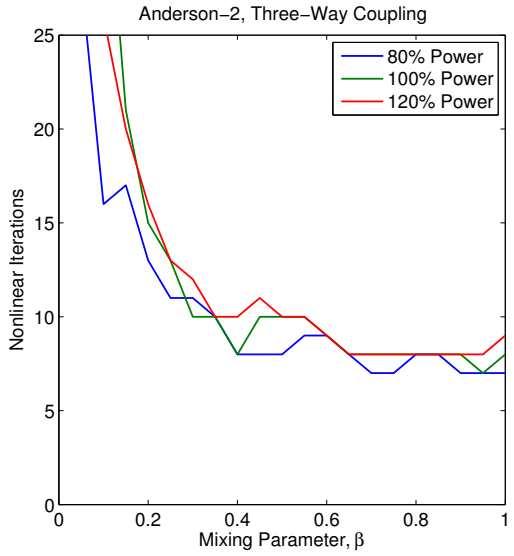
Figure 5.4: Nonlinear iterations to convergence for two-way coupling



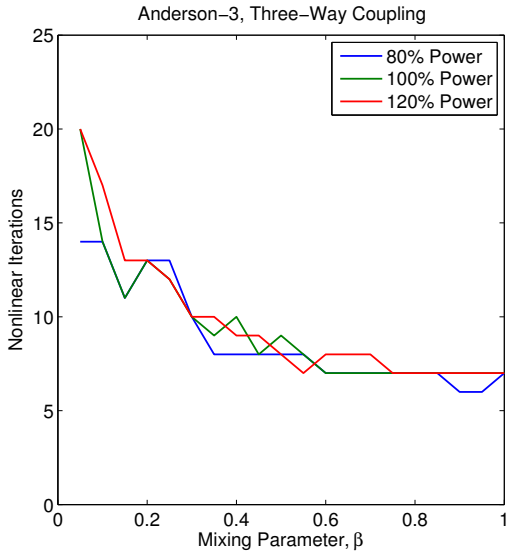
(a) Three-way coupling comparison, 100% power



(b) Three-way coupling, Anderson-1 iterations



(c) Three-way coupling, Anderson-2 iterations



(d) Three-way coupling, Anderson-3 iterations

Figure 5.5: Nonlinear iterations to convergence for three-way coupling

Chapter 6

Anderson Acceleration for Tiamat

6.1 Introduction

Given the promising results for Anderson acceleration on the model problem in the previous chapter, we now now return to Tiamat and examine the performance of Anderson acceleration on this production-level code coupling. Again, we are concerned with the solving the fully-coupled problem at HFP conditions. In order to apply Anderson acceleration to solve this problem, we simply need to define the vector of unknowns u and the fixed-point map $G(u)$ to which we apply the method. In this chapter, we first outline how we formulate the fixed-point problem in Section 6.2. We then consider some technical details about how we integrate Anderson acceleration into the Tiamat code in Section 6.3, and lastly, we consider numerical tests which illustrate the performance of Anderson acceleration in comparison to other methods in Section 6.4. Initial results from this work which utilize Insilico rather than MPACT for the neutronics application can be found in [59].

6.2 Definition of Fixed-Point Maps

Because we implement Anderson acceleration in order to improve the rate of convergence for Picard iteration, the fixed-point map to which we apply Anderson acceleration should in some way be derived from the Picard iteration. When performing Picard iteration, the fact that we are solving a fixed-point problem need not be explicitly stated. As described in Algorithms 1 and 3, the Picard iterations simply describe a process to map the current solutions to the new solutions through a sequence of solves and data transfers. While there is an underlying fixed-point problem $u = G(u)$ that this attempting to solve, this is treated implicitly and the vector of unknowns u and the fixed-point evaluation $G(u)$ need not be explicitly formed. Conversely, in order to utilize Anderson acceleration to solve this problem we need to explicitly define u

and $G(u)$, as we need to compute the fixed-point residual $F(u) = G(u) - u$ in order to form and solve the least-squares problem in the Anderson acceleration algorithm. Note that u and $G(u)$ should be defined in such a way that we can express the Picard iteration in the form $u_{k+1} = G(u_k)$.

The notation utilized in Algorithms 1 and 3 lends itself naturally to formulating the problem as a fixed-point problem in terms of the state variables x_B, x_C, and x_M. One could simply define

$$G \begin{pmatrix} x_B^n \\ x_C^n \\ x_M^n \end{pmatrix} = \begin{pmatrix} x_B^{n+1} \\ x_C^{n+1} \\ x_M^{n+1} \end{pmatrix}, \tag{6.1}$$

where x_B^{n+1}, x_C^{n+1}, and x_M^{n+1} are the updated solutions resulting from performing one Picard iteration given x_B^n, x_C^n, and x_M^n as inputs. This formulation is sufficient for Picard iteration, as this simply requires the ability to evaluate the operator G, though G and the state variables need not be explicitly accessible. This definition of the fixed-point problem will not work in the case of Tiamat for the purpose of implementing Anderson acceleration. Applying Anderson acceleration with this fixed-point map would require explicit access to the state variables for each of the applications in order to compute the fixed-point residual $F(u) = G(u) - u$, and this will not be possible with the applications we are utilizing, as neither CTF nor Bison provides simple access to their state variables.

As a result of this, we must formulate the fixed-point problem in a different form in order to apply Anderson acceleration. Rather than the single-physics state variables, we can instead attempt to pose the iteration as a fixed-point problem in terms of the coupling parameter vectors that are computed by the data transfer functions. That is, instead of mapping x_B, x_C, and x_M from their current values to an updated approximate solution, we pose the Picard iteration as a method for mapping T_f, T_w, ρ_w, T_c, q and q'' to updated values through a series of solves and data transfers. In this sense, we attempt to converge the solutions to the single-physics application codes to the fully-coupled solution by converging the coupling parameters on which the single-physics solutions depend. It is important to note that this coupling parameter data being passed between codes must be necessarily accessible in order to implement the data transfers.

In a sense, this method of formulating the fully-coupled problem in terms of coupling parameter vectors can be viewed as a form of nonlinear elimination. When we write the the fully-coupled problem in the form of the residual equation (2.21), we have eliminated the transfer functions from the coupled system in the sense that they are embedded within the single-physics systems. However, we can instead treat these transfer functions as additional sets of constraints,

and express the fully-coupled system in the following form

$$F\begin{pmatrix} x_B \\ x_C \\ x_M \\ T_f \\ T_w \\ \rho_w \\ T_c \\ q \\ q'' \end{pmatrix} = \begin{pmatrix} f_B(x_B, T_c, q) \\ f_C(x_C, q'') \\ f_M(x_M, T_f, T_w, \rho_w) \\ r_{M,B}(x_B) - T_f \\ r_{M,C,T}(x_C) - T_w \\ r_{M,C,\rho}(x_C) - \rho_w \\ r_{B,C}(x_C) - T_c \\ r_{B,M}(x_M) - q \\ r_{C,B}(x_B) - q'' \end{pmatrix} = 0. \tag{6.2}$$

Now, rather than eliminating the transfer functions from the system, we can employ a non-linear elimination scheme and instead eliminate the single-physics residuals from the system. The single-physics residual equation $f_B(x_B, T_c, q) = 0$, implicitly defines the solution x_B as a function of the clad surface temperature T_c and the power q. We then denote the solution to this equation given T_c and q by $x_B(T_c, q)$. We can similarly utilize $f_C(x_C, q'') = 0$ and $f_M(x_M, T_f, T_w, \rho_w) = 0$ to express x_C as a function of the heat flux q'' and x_M as a function of the fuel temperature T_f, coolant temperature T_w, and coolant density ρ_w, denoting these by $x_C(q'')$ and $x_M(T_f, T_w, \rho_w)$ respectively. Then, eliminating these from the coupled system, (6.2) reduces to

$$F\begin{pmatrix} T_f \\ T_w \\ \rho_w \\ T_c \\ q \\ q'' \end{pmatrix} = \begin{pmatrix} r_{M,B}(x_B(T_c, q)) - T_f \\ r_{M,C,T}(x_C(q'')) - T_w \\ r_{M,C,\rho}(x_C(q'')) - \rho_w \\ r_{B,C}(x_C(q'')) - T_c \\ r_{B,M}(x_M(T_f, T_w, \rho_w)) - q \\ r_{C,B}(x_B(T_c, q)) - q'' \end{pmatrix} = 0. \tag{6.3}$$

6.2.1 Block Gauss-Seidel Map

We now explicitly define the fixed-point problems that we utilize Anderson acceleration to solve. We first consider the block Gauss-Seidel scheme. As in Algorithm 1, we perform the single-physics solves in the order MPACT, Bison, then CTF. We express the block Gauss-Seidel fixed-point map by the following

1. Given T_f, T_w, ρ_w, T_c
2. Solve $f_M(x_M, T_f, T_w, \rho_w) = 0$ for x_M
3. Transfer MPACT to Bison, $q = r_{B,M}(x_M)$

4. Solve $f_B(x_B, T_c, q) = 0$ for x_B
5. Transfer Bison to MPACT, $\hat{T}_f = r_{M,B}(x_B)$
6. Transfer Bison to CTF, $q'' = r_{C,B}(x_B)$
7. Solve $f_C(x_C, q'') = 0$ for x_C
8. Transfer CTF to MPACT, $\hat{T}_w = r_{M,C,T}(x_C)$ and $\hat{\rho}_w = r_{M,C,\rho}(x_C)$
9. Transfer CTF to Bison, $\hat{T}_c = r_{B,C}(x_C)$

Then define

$$
\begin{aligned}
G_{GS}\begin{pmatrix} T_f \\ T_w \\ \rho_w \\ T_c \end{pmatrix} &= \begin{pmatrix} \hat{T}_f \\ \hat{T}_w \\ \hat{\rho}_w \\ \hat{T}_c \end{pmatrix} \\
&= \begin{pmatrix} r_{M,B}(x_B(T_c, r_{B,M}(x_M(T_f, T_w, \rho_w)))) \\ r_{M,C,T}(x_C(r_{C,B}(x_B(T_c, r_{B,M}(x_M(T_f, T_w, \rho_w)))))) \\ r_{M,C,\rho}(x_C(r_{C,B}(x_B(T_c, r_{B,M}(x_M(T_f, T_w, \rho_w)))))) \\ r_{B,C}(x_C(r_{C,B}(x_B(T_c, r_{B,M}(x_M(T_f, T_w, \rho_w)))))) \end{pmatrix}.
\end{aligned} \tag{6.4}
$$

Given the initial iterates $T_f^0 = r_{M,B}(x_B^0), T_w^0 = r_{M,C,T}(x_C^0), \rho_w^0 = r_{M,C,\rho}(x_C^0)$, and $T_c^0 = r_{B,C}(x_C^0)$, Picard iteration with this fixed-point map will produce the same sequence as that given by Algorithm 1. Also note that we have eliminated q and q'' from the quantities that we solve for. These quantities are computed internally in the evaluation of the fixed-point map, so they do not need to be passed in as inputs. Since q is computed internally in this fixed-point map, we do not consider a power damping when applying Anderson acceleration, and opt instead to utilize the Anderson acceleration mixing parameter. This also acts as a damping factor, but it is applied to each of the components of the vector of unknowns u, so in this case it acts as a uniform damping on all the temperature and density updates. We lastly note that there are multiple possible block Gauss-Seidel fixed-point maps, depending on the order in which one chooses to solve the applications. This ordering was chosen in order to as closely as possible mirror the approach from the previous chapter, in which the fixed-point map was defined in terms of temperatures alone.

6.2.2 Block Jacobi Map

We proceed in a similar manner to define the block Jacobi fixed-point map. Again, for the block Jacobi iteration we alternate between phases of solving all applications and then performing all

data transfers, so we let the block Jacobi map be defined by the following process

1. Given $T_f, T_w, \rho_w, T_c, q, q''$
2. Solve $f_M(x_M, T_f, T_w, \rho_w) = 0$ for x_M
3. Solve $f_B(x_B, T_c, q) = 0$ for x_B
4. Solve $f_C(x_C, q'') = 0$ for x_C
5. Transfer MPACT to Bison, $\hat{q} = r_{B,M}(x_M)$
6. Transfer Bison to MPACT, $\hat{T}_f = r_{M,B}(x_B)$
7. Transfer Bison to CTF, $\hat{q}'' = r_{C,B}(x_B)$
8. Transfer CTF to MPACT, $\hat{T}_w = r_{M,C,T}(x_C)$ and $\hat{\rho}_w = r_{M,C,\rho}(x_C)$
9. Transfer CTF to Bison, $\hat{T}_c = r_{B,C}(x_C)$

We then define

$$G_{JAC}\begin{pmatrix} T_f \\ T_w \\ \rho_w \\ T_c \\ q \\ q'' \end{pmatrix} = \begin{pmatrix} \hat{T}_f \\ \hat{T}_w \\ \hat{\rho}_w \\ \hat{T}_c \\ (1-\omega)q + \omega\hat{q} \\ \hat{q}'' \end{pmatrix} = \begin{pmatrix} r_{M,B}(x_B(T_c, q)) \\ r_{M,C,T}(x_C(q'')) \\ r_{M,C,\rho}(x_C(q'')) \\ r_{B,C}(x_C(q'')) \\ (1-\omega)q + \omega\, r_{B,M}(x_M(T_f, T_w, \rho_w)) \\ r_{C,B}(x_B(T_c, q)) \end{pmatrix}. \quad (6.5)$$

In this case q and q'' are present in the vector of unknowns. Since each of the applications is simultaneously solved at the beginning of the evaluation of the fixed-point map, each of the coupling parameter vectors is required as input. As the power is now included in the vector of unknowns, we allow for a damping on the power update.

6.2.3 Intermediate Map

As was mentioned in Section 1.2.1, the two types of fixed-point maps defined above represent the essentially the only possible orderings of single-physics solves when considering two applications. However, when considering more than two applications, there are more possible orderings in which to solve the application codes. For instance, one could partition the applications into groups and loop through solving those groups in some order, with applications within a group solved simultaneously. This would result in a per-iteration run time somewhere between block Jacobi and block Gauss-Seidel. This sort of intermediate fixed-point map could perform well

if the strength of the coupling between some of the applications is relatively weak. When two applications are simultaneously solved, the applications are treated as more weakly coupled, as the feedback between the two is delayed until the following iteration. Imposing a weaker coupling betweens sets of physics that are themselves weakly coupled might possibly not cause a dramatic increase in iterations over a block Gauss-Seidel scheme while offering improved utilization of parallel resources. One such intermediate map that we briefly consider is given by the following process

1. Given $T_f, T_w, \rho_w, T_c, q, q''$
2. Solve $f_M(x_M, T_f, T_w, \rho_w) = 0$ for x_M
3. Solve $f_B(x_B, T_c, q) = 0$ for x_B
4. Transfer MPACT to Bison, $\hat{q} = r_{B,M}(x_M)$
5. Transfer Bison to CTF, $q'' = r_{C,B}(x_B)$
6. Transfer Bison to MPACT, $\hat{T}_f = r_{M,B}(x_B)$
7. Solve $f_C(x_C, q'') = 0$ for x_C
8. Transfer CTF to MPACT, $\hat{T}_w = r_{M,C,T}(x_C)$ and $\hat{\rho}_w = r_{M,C,\rho}(x_C)$
9. Transfer CTF to Bison, $\hat{T}_c = r_{B,C}(x_C)$

We then define

$$G_{INT}\begin{pmatrix} T_f \\ T_w \\ \rho_w \\ T_c \\ q \end{pmatrix} = \begin{pmatrix} \hat{T}_f \\ \hat{T}_w \\ \hat{\rho}_w \\ \hat{T}_c \\ (1-\omega)q + \omega\hat{q} \end{pmatrix} = \begin{pmatrix} r_{M,B}(x_B(T_c, q)) \\ r_{M,C,T}(x_C(r_{C,B}(x_B(T_c, q)))) \\ r_{M,C,\rho}(x_C(r_{C,B}(x_B(T_c, q)))) \\ r_{B,C}(x_C(r_{C,B}(x_B(T_c, q)))) \\ (1-\omega)q + \omega\, r_{B,M}(x_M(T_f, T_w, \rho_w)) \end{pmatrix}. \quad (6.6)$$

In this map, we alternate between simultaneously solving Bison and MPACT and then solving CTF. While the solutions for Bison and MPACT should be rather strongly coupled, the stronger coupling that this map imposes between both of these applications and CTF could result in a lesser increase in iterations over block Gauss-Seidel than what is observed for block Jacobi. Like the block Jacobi map, this includes the power in the vector of unknowns, so we allow for a damping on the power update computed in this map.

We lastly note here the connection between the fixed-point maps we have defined in this section and the residual equation (6.3) that these iterations are intended to solve. First off, the

fixed-point residual for the block Jacobi map is given by

$$F_{JAC}\begin{pmatrix} T_f \\ T_w \\ \rho_w \\ T_c \\ q \\ q'' \end{pmatrix} = \begin{pmatrix} r_{M,B}(x_B(T_c,q)) - T_f \\ r_{M,C,T}(x_C(q'')) - T_w \\ r_{M,C,\rho}(x_C(q'')) - \rho_w \\ r_{B,C}(x_C(q'')) - T_c \\ \omega(r_{B,M}(x_M(T_f,T_w,\rho_w)) - q) \\ r_{C,B}(x_B(T_c,q)) - q'' \end{pmatrix}. \tag{6.7}$$

This is precisely the residual given in (6.3) (with the power component scaled by the damping factor).

Next, the fixed-point residual for the intermediate map is given by

$$F_{INT}\begin{pmatrix} T_f \\ T_w \\ \rho_w \\ T_c \\ q \end{pmatrix} = \begin{pmatrix} r_{M,B}(x_B(T_c,q)) - T_f \\ r_{M,C,T}(x_C(r_{C,B}(x_B(T_c,q)))) - T_w \\ r_{M,C,\rho}(x_C(r_{C,B}(x_B(T_c,q)))) - \rho_w \\ r_{B,C}(x_C(r_{C,B}(x_B(T_c,q)))) - T_c \\ \omega(r_{B,M}(x_M(T_f,T_w,\rho_w)) - q) \end{pmatrix}. \tag{6.8}$$

Note that the final component of the block Jacobi residual explicitly defines q'' as a function of the other unknowns as $q'' = r_{C,B}(x_B(T_c,q))$. Using this equation to eliminate q'' from the block Jacobi residual results in this intermediate map residual.

Lastly, the fixed-point residual for the block Gauss-Seidel map is given by

$$F_{GS}\begin{pmatrix} T_f \\ T_w \\ \rho_w \\ T_c \end{pmatrix} = \begin{pmatrix} r_{M,B}(x_B(T_c, r_{B,M}(x_M(T_f,T_w,\rho_w)))) - T_f \\ r_{M,C,T}(x_C(r_{C,B}(x_B(T_c, r_{B,M}(x_M(T_f,T_w,\rho_w)))))) - T_w \\ r_{M,C,\rho}(x_C(r_{C,B}(x_B(T_c, r_{B,M}(x_M(T_f,T_w,\rho_w)))))) - \rho_w \\ r_{B,C}(x_C(r_{C,B}(x_B(T_c, r_{B,M}(x_M(T_f,T_w,\rho_w)))))) - T_c \end{pmatrix}. \tag{6.9}$$

Like above, the last component of the intermediate map residual explicitly defines q as a function of the other unknowns as $q = r_{B,M}(x_M(T_f,T_w,\rho_w))$, and using this to eliminate q from the intermediate map residual gives precisely the residual for the block Gauss-Seidel map. Hence, the block Jacobi map is directly solving the residual equation given in Equation (6.3), while the block Gauss-Seidel and intermediate maps are solving nonlinearly eliminated forms this residual equation.

6.2.4 Scaling of Unknown Fields

As each of the components of the fixed-point maps (6.4), (6.5), and (6.6) represents a different physical quantity, it is possible that the relative scaling of these fields may be an issue with regard to the least-squares problem in Anderson acceleration. If the values for one field are significantly larger in magnitude than the others, then even small changes in this field will have a more significant weight the least-squares problem than large changes in the other fields. Similarly, if a field has entries of relatively small magnitude, the weight of this field in the least-squares problem will be negligible.

To address this issue, we apply Anderson acceleration instead to a scaled fixed-point problem. For this, we first introduce the scaled variables $v = Mu$, where M is a diagonal scaling matrix. We then apply Anderson to the scaled fixed-point problem given by

$$v = MG(M^{-1}v) \equiv H(v), \tag{6.10}$$

where G is one of the three fixed-point maps defined above, and u is the vector of unknowns corresponding to that map. We note that we can express the fixed-point residual corresponding to the scaled problem as

$$H(v) - v = MG(M^{-1}v) - v = M(G(M^{-1}v) - M^{-1}v) = M(G(u) - u). \tag{6.11}$$

That is, the scaled fixed-point residual is simply a left preconditioning of the unscaled residual.

It then remains to describe how the scaling matrix M is defined for each of the fixed-point maps we consider. Beginning with the simplest case, for the block Gauss-Seidel map we define the scaling matrix M_{GS} by

$$M_{GS} = \begin{pmatrix} \text{diag}(T_f^0)^{-1} & & & \\ & \text{diag}(T_w^0)^{-1} & & \\ & & \text{diag}(\rho_w^0)^{-1} & \\ & & & \text{diag}(T_c^0)^{-1} \end{pmatrix}, \tag{6.12}$$

where T_f^0, T_w^0, ρ_w^0, and T_c^0 are the fuel temperature, coolant temperature, coolant density, and clad temperature from the initial iterate, and diag indicates a diagonal matrix with the given vector along the diagonal. This choice of scaling gives as the initial iterate for the scaled problem a vector of ones. The scaled fixed-point residual will measure changes in each entry relative to its initial value. We note that this choice of scaling should not pose a threat of zero division, as both the temperature and density are given on an absolute scale, and should never be very near zero under normal conditions. Scaling for this map should likely not be of great concern, as the majority of unknowns we solve for are temperatures, which should all be on roughly the same

order of magnitude. However, with the chosen units the density values will be several orders of magnitude smaller than the temperatures, so this scaling will give the density field a more appreciable weight in the Anderson acceleration least-squares problem.

With the other fixed-point maps, scaling needs to be handled slightly differently, as employing a similar scheme as what is used for block Gauss-Seidel could could lead to zero division. While the power is on an absolute scale, local power values of zero are possible, and will in fact be the case in non-fuel pins or cladding regions. Zero division would similarly possible for the heat flux, and in fact negative heat flux values can be obtained. As the heat flux represents energy being deposited into the coolant, a negative value would represent energy flowing from the coolant into a rod, and this is possible in non-fuel rods. We address this issue by scaling these fields by average values rather then the entry-wise values. For the intermediate map, only the power is introduced as an unknown field in addition to those considered in the block Gauss-Seidel map. Then, employing the same scaling that is used for the block Gauss-Seidel map for the temperature and density fields, we define the scaling matrix M_{INT} for this map by

$$M_{INT} = \begin{pmatrix} M_{GS} & \\ & M_q \end{pmatrix}. \tag{6.13}$$

In this, we define

$$M_q = \begin{pmatrix} \frac{1}{\bar{q}_1^0} I & & \\ & \ddots & \\ & & \frac{1}{\bar{q}_{N_r}^0} I \end{pmatrix}, \tag{6.14}$$

where N_r is the number of rods and $\bar{q}_i^0$ is the initial average power in rod i. We define the initial average power in rod i as

$$\bar{q}_i^0 = \frac{1}{N_{q_i}} \sum_{j \in R_i} |q_j^0|, \tag{6.15}$$

where N_{q_i} is the number of power unknowns contained in rod i, R_i is the set of indices corresponding to power unknowns located in rod i, and q^0 is the initial power distribution. If $\bar{q}_i^0 = 0$ for some rod (e.g. a non-fuel rod), we disable scaling for this rod by setting this value to 1. Essentially, this scales the power components of the fixed-point residual by their initial rod-wise initial average value.

We lastly consider the scaling matrix for the block Jacobi map. This appends heat flux to the vector of unknowns for the intermediate map. Then employing the same scaling strategy for the temperature, density, and power fields as for the intermediate map, we define the block

Jacobi scaling matrix M_{JAC} by

$$M_{JAC} = \begin{pmatrix} M_{INT} & \\ & \frac{1}{\bar{q}_0''} I \end{pmatrix}, \tag{6.16}$$

where the initial average heat flux $\bar{q}_0''$ is defined as

$$\bar{q}_0'' = \frac{1}{N_{q''}} \sum_{i=0}^{N_{q''}} |q_{0,i}''|. \tag{6.17}$$

In this, $N_{q''}$ is the number of heat flux unknowns and q_0'' is the initial heat flux distribution. This scales each heat flux component of the fixed-point residual by the initial global average heat flux.

6.3 Implementation Details

In this section, we overview some technical details regarding the integration of Anderson acceleration into the Tiamat code coupling. In order to integrate Anderson acceleration into this coupling, we utilize the NOX solver that we described in Chapter 4, and we now overview how this is accomplished.

6.3.1 NOX Solver Creation

As described in Section 4.2, creating a NOX Anderson acceleration solver requires a NOX group, a status test, and a parameter list passed in during construction. The parameter list describes several options for the solver, like the storage depth and the mixing parameter, and the status test describes the termination criteria. The group contains most important information for describing an iteration, like the approximate solution, the residual vector, and derivative information. In order for the group to compute the residual and any other supported information, it must be provided some model evaluator object that defines how this information is computed. For this purpose, we utilize NOX's support for the Trilinos package Thyra (see Appendix B.2.4) for implementations of both the group and model evaluator classes. The model evaluator class we create inherits from the `Thyra::StateFuncModelEvaluatorBase` class, which is a basic implementation of a Thyra model evaluator which supports computing a residual given some input vector. The crux of this class is the definition of the inherited member function `evalModelImpl`, which is what defines the residual evaluation. In our case, this routine, which computes the fixed-point residual given some input set of coupling parameter vectors, proceeds essentially as follows.

1. Extract coupling parameter vectors from input vector.
2. Convert from scaled variables to unscaled variables by applying the inverse of the scaling matrix.
3. Write input coupling parameter vectors to the appropriate DTK target containers.
4. Run through sequence of solves and transfers.
5. Extract new coupling parameter data from the same DTK target containers.
6. Apply scaling to the variables.
7. Compute fixed-point residual (possibly applying block damping) and write result to the output vector.

This model evaluator class supports each of the fixed-point map types that were defined in the previous section. We simply specify on construction of this object whether we are considering block Gauss-Seidel, block Jacobi, or the intermediate map, and this determines within the model evaluator which coupling parameter vectors comprise the vector of unknowns u, and in what order the solves and data transfers should be performed. This class also features functions for creating the appropriate initial iterate and scaling vector for the given fixed-point map type given the current solutions to each of the single-physics application codes. With this model evaluator created, it is used to create a `NOX::Thyra::Group` object, which is then used to create the Anderson acceleration solver. Beneath the Thyra objects, Tpetra (see Appendix B.2.3) is utilized for the underlying distributed vectors.

6.3.2 Interfacing With PIKE

As was previously stated, Tiamat uses PIKE solvers for the fully-coupled solve, so in order to integrate Anderson acceleration with minimal change to the Tiamat code, this required creating a wrapper class which implements the PIKE solver interface while internally utilizing a NOX solver for the routine to compute a new iterate. The type of the internal solver is the NOX solver base class, `NOX::Solver::Generic`, so the underlying NOX solver need not be the Anderson acceleration solver. This is useful in Section 6.4.1, when we compare the performance of Anderson acceleration and Picard iteration with JFNK. This wrapper class allows Tiamat to easily interface with the NOX solver while utilizing the previously developed PIKE objects (status tests, solver observers, etc.) without any change. The wrapper class that has been developed is very minimal. This class contains a pointer to the underlying NOX solver and most of the work is deferred to this solver. The main functionality provided by this class is accessor functions for the underlying NOX solver and an implementation of the inherited

member function `stepImplementation`, which is the function which computes the next iterate. In this case, this simply calls the step routine in the NOX solver. This function computes the new iterate with, then evaluates the fixed-point residual at this new iterate. The exception to this is on the first iteration. For this step, we simply evaluate the fixed-point residual for the initial iterate. Otherwise, a call to the NOX solver step would advance the internal models twice, as this would evaluate the initial residual, compute the first NOX iterate, and then compute the residual at this new iterate (each residual evaluation involves solving the application codes). Additionally, while we are using NOX to solve the problem, PIKE rather than NOX is used to determine convergence of the system. The only status test we set in the NOX solver enforces a maximum number of iterations.

6.3.3 Setting NOX Initial Iterate

After the NOX solver is created, and has been wrapped in the PIKE solver, Tiamat solves the problem by simply calling the `solve` routine in the PIKE solver, which repeatedly calls the `step` function until the iteration has converged or failed to converge. The initialization phase for ramping Bison to HFP, which was described in Section 2.3, is performed by a PIKE solver observer, which inserts this action before the solver begins to solve the problem. The initial iterate we desire for the NOX Anderson acceleration solver is not available until after this initialization phase, as it is computed from the estimated HFP solutions for the single-physics applications that result from the initialization. Recall, however, that an initial iterate is required for the construction of the NOX group, and the group is required to create the NOX solver. The NOX solver must be set in the PIKE solver before `solve` is called, so this raises an issue as to how to set the initial iterate in the NOX solver.

The method by which we address this is to introduce another solver observer which sets the correct initial iterate by using the NOX solver's `reset` member function. When constructing the NOX solver, we simply pass in a dummy vector to the group which has the correct size. Then when solving the problem, this observer inserts an action before each call to the `step` function. It first checks which iteration the PIKE solver is at. If it is determined that the PIKE solver is in the first iteration, this observer uses the model evaluator described in Section 6.3.1 to compute the correct initial iterate, and the `reset` function in the NOX solver is called to set this as the initial iterate. This observer can also be used for resetting the NOX solver for use in repeated solves if solving for more than one time step.

6.4 Numerical Results

We now consider numerical tests of various problem size in order to demonstrate the behavior of Anderson acceleration when applied to solve the fully-coupled HFP problem in Tiamat, and these results are compared with other methods.

6.4.1 Single Fuel Rod Tests

We first consider the simulation of a single fuel rod at HFP conditions for a single time step. This is the same problem we considered when illustrating the behavior of Picard iteration in Section 2.3.3, so the input specifications for this problem are given in Appendix C.3.

Sensitivity to Power Level

We first compare Anderson acceleration against Picard iteration with respect to some of the issues that we noted as weaknesses for Picard, namely poor robustness with regard to variation in parameters like the power and damping levels. First, in Table 6.1, we see results from solving the fully-coupled problem at various power levels with both Picard and Anderson. We now include results for Picard iteration with the intermediate fixed-point map. As was noted previously, the Picard iterations display some sensitivity to the power level. With the block Gauss-Seidel map, there is only a slight rise in iteration counts when going from 100% power to 125% power. However the other two fixed-point maps display a significant dependence on the power level. The falloff in performance as the power level is increased for the intermediate map is actually even more dramatic than what is observed for the Jacobi map. In both cases, the Picard iterations fail to converge at 100% power and above.

The Anderson acceleration results in this table seem to indicate that this method is somewhat more robust with regard to power variation than Picard. We note that with the Gauss-Seidel map, both Anderson and Picard seem somewhat insensitive to changes in the power level. In both cases, the performance is fairly consistent over all but the 125% power case, for which there is a minor increase in iterations to convergence, and the rise for Anderson is slightly less than that for Picard. Also note that the in each case, the Anderson iteration converges in as few or fewer iterations than Picard iteration. Anderson with the other fixed-point maps does not display quite this level of robustness with regard to variation in the power level, but it does seem to provide a significant improvement over Picard with these maps. While there is an upward trend in the Anderson iteration counts, it is significantly less than the increase for Picard iteration, and the Anderson iterations actually converge at each power considered. We lastly note that even for the low powers, at which Picard performs fairly well, the Anderson

Table 6.1: Comparison of Picard and Anderson-2 with each of the fixed-point maps (Gauss-Seidel, intermediate, and Jacobi) for single-rod Tiamat simulation at various power levels. Damping factor = 0.5 and max iterations = 25

Power Level	Method	Iterations	Solve Time (s)	k_{eff}	$T_{f,max}$
25%	Picard (GS)	9	303	1.23708	483.55
	Picard (INT)	10	328	1.23709	483.57
	Picard (JAC)	14	392	1.23708	483.54
	Anderson-2 (GS)	7	267	1.23710	483.54
	Anderson-2 (INT)	9	290	1.23710	483.55
	Anderson-2 (JAC)	10	280	1.23708	483.54
50%	Picard (GS)	7	292	1.23105	694.09
	Picard (INT)	17	486	1.23106	693.97
	Picard (JAC)	15	407	1.23106	693.92
	Anderson-2 (GS)	7	290	1.23102	694.07
	Anderson-2 (INT)	10	346	1.23109	694.00
	Anderson-2 (JAC)	10	312	1.23104	694.12
75%	Picard (GS)	8	332	1.22493	931.20
	Picard (INT)	24	634	1.22500	929.81
	Picard (JAC)	18	462	1.22497	930.18
	Anderson-2 (GS)	8	324	1.22495	930.22
	Anderson-2 (INT)	12	389	1.22498	929.83
	Anderson-2 (JAC)	10	331	1.22496	930.89
100%	Picard (GS)	8	353	1.21857	1194.67
	Picard (INT)	DNC			
	Picard (JAC)	DNC			
	Anderson-2 (GS)	7	337	1.21858	1194.06
	Anderson-2 (INT)	14	466	1.21859	1193.20
	Anderson-2 (JAC)	19	486	1.21855	1195.16
125%	Picard (GS)	12	445	1.21214	1505.16
	Picard (INT)	DNC			
	Picard (JAC)	DNC			
	Anderson-2 (GS)	10	420	1.21218	1504.21
	Anderson-2 (INT)	12	400	1.21216	1505.35
	Anderson-2 (JAC)	16	459	1.21214	1504.62

acceleration provides a noticeable improvement over Picard iteration for the intermediate and Jacobi maps.

Sensitivity to Damping Level

We now consider the effect that damping has on the performance of Anderson acceleration for this problem. First, consider Figure 6.1. In this figure we see iteration counts for Picard and Anderson with the Gauss-Seidel map using various damping parameters at 100% power. As expected, the Picard curve displays a clear optimal damping level, and performance falls off rapidly away from this optimal level. Conversely, for Anderson we observe a significant improvement in robustness with respect to variation in damping. First of all, Anderson converges in each case considered, while Picard fails if the damping factor is too large. As the damping factor is made very small, the iteration counts begin to rise, but otherwise the performance displayed by Anderson acceleration is rather consistent across each damping level considered. It is also interesting to note that over this wide range at which Anderson performs well, the iteration counts are generally comparable to, and occasionally better than, the count for Picard at its optimal damping level. We also note that this represents a best case scenario for Picard. With the other fixed-point maps, Anderson provides a much more significant improvement over optimally damped Picard.

Next, we refer to Figure 6.2, which shows Anderson-2 iteration counts at various power and damping levels for each of the fixed-point maps. We had previously noted in Section 2.3.3 that the optimal damping level for Picard is dependent on the power level. In this figure, we note that this is not the case for Anderson acceleration. Each curve displays behavior similar to what we observe in Figure 6.1, in that the solve times can rise a bit for very small damping factors, but the performance is generally very consistent over a wide range of damping levels. We note that there is a slight upward trend in the iteration counts as the power is increased, though we expect the iterations to solution for Picard at the optimal damping level to similarly increase. Whereas for Picard power variation shifts the shape of the iteration count curve and the optimal damping level (as seen in Figure 2.8), for Anderson the shape of the curve is essentially unchanged. Hence, it should be easier to obtain good performance when utilizing Anderson acceleration than Picard iteration, as Picard iteration requires knowledge of a good damping parameter for a given problem, while Anderson displays more consistent good performance.

Anderson Acceleration Storage Depth Parameter

We now consider the effect of varying the storage depth parameter for Anderson acceleration. First, we refer again to Figure 6.1, which shows iteration count results for solving the problem by Picard, Anderson-1, Anderson-2, and Anderson-3 with the Gauss-Seidel map. In this, we

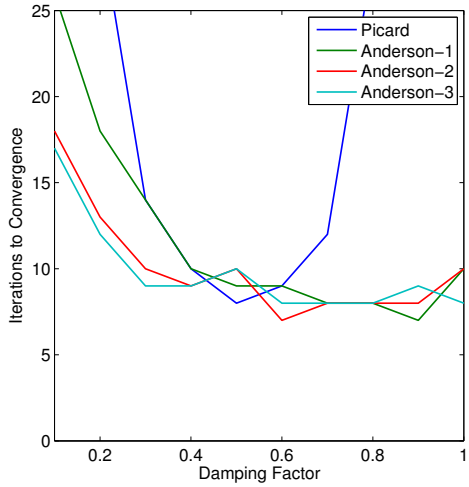
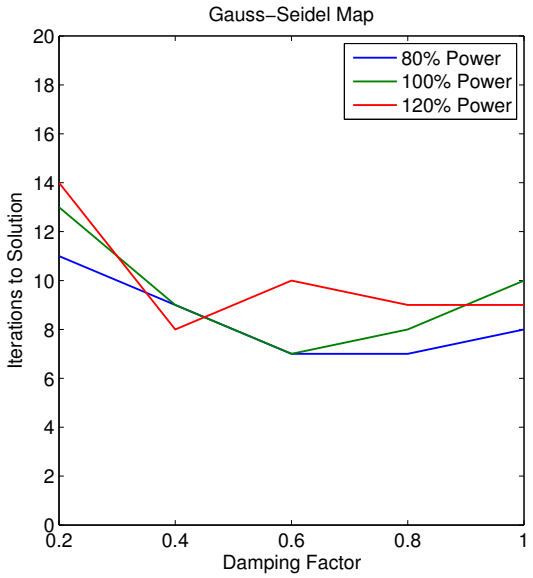


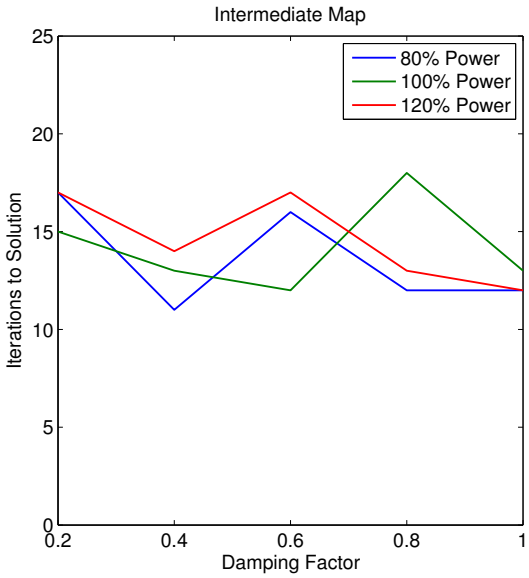
Figure 6.1: Comparison of Picard and Anderson with the Gauss-Seidel map for Tiamat single-rod tests at 100% power, varying the damping level and Anderson storage depth parameter

observe that the performance for each choice of storage depth parameter is rather similar. Anderson-2 and Anderson-3 only differ by more than a single iteration at damping factor 1.0, where Anderson-3 converges 2 iterations faster. Anderson-1 requires several more iterations than the other two storage depth parameters for damping factors smaller than 0.4, but over the range 0.4–1.0, the performance of all three methods is very similar and consistently comparable to Picard at its optimal level. The improved performance for smaller damping factors may give reason to choose Anderson-2 over Anderson-1, especially since the increase in storage and computational costs for Anderson-2 over Anderson-1 is not a significant concern. However, these results do not display a compelling reason to consider a storage depth parameter greater than 2.

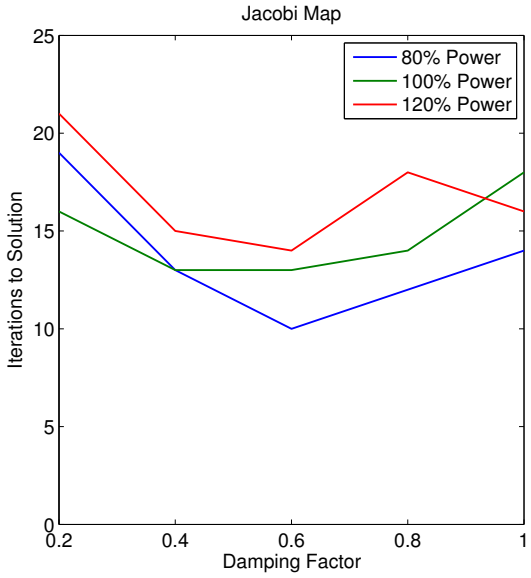
Next, Figure 6.3 displays fixed-point residual history plots comparing Picard and Anderson with various storage depth parameters. In this, we see that for each fixed-point map, Anderson provides a noticeable improvement over Picard with respect to the rate of convergence of the fixed-point residual for each considered storage depth parameter. This is especially apparent for the intermediate and Jacobi fixed-point maps, where we generally note residual norms for Anderson several orders of magnitude smaller than Picard at the point where the Anderson iterations terminate. However, even for the Gauss-Seidel map, each Anderson iteration reports a final fixed-point residual norm smaller than that for Picard by about an order of magnitude or more, even though the iterations all converge in a similar number of iterations. With respect to the choice of storage depth parameter for Anderson, these figures show similar results to what was noted above. That is, there is no obvious trend which suggests an advantage for one choice of storage depth parameter over the others. Anderson-1 converges in the fewest iterations



(a) Block Gauss-Seidel map

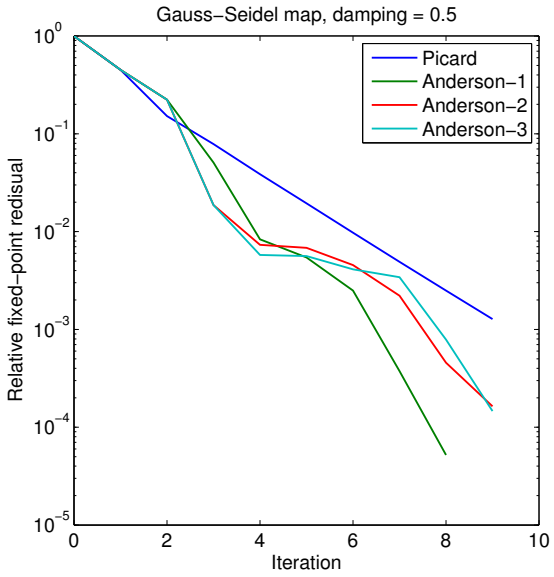


(b) Intermediate map

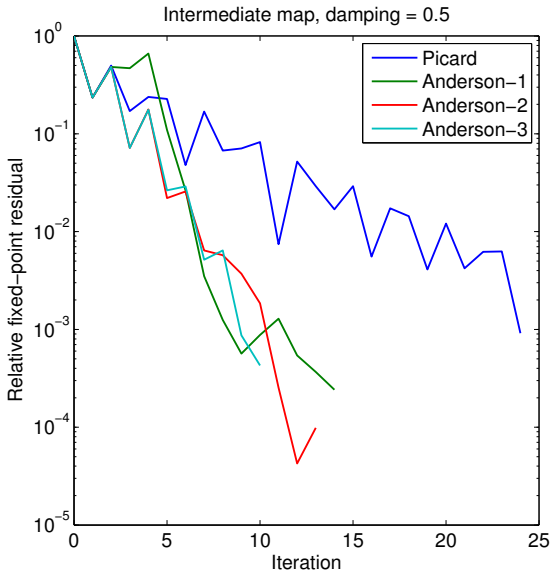


(c) Block Jacobi map

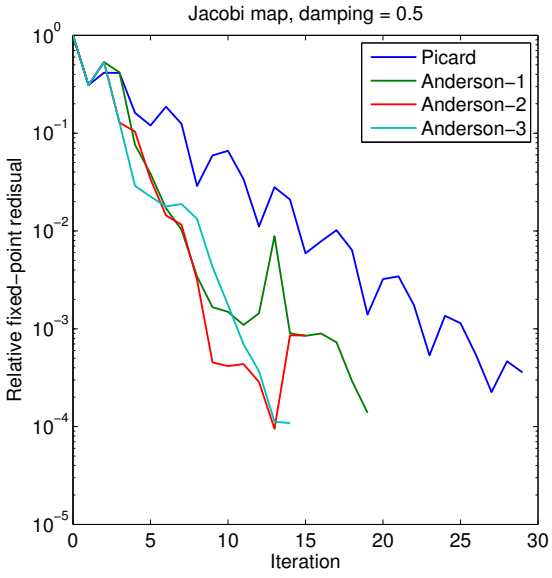
Figure 6.2: Anderson-2 iteration counts for single-rod Tiamat tests at several power levels, varying the damping factor



(a) Block Gauss-Seidel map



(b) Intermediate map



(c) Block Jacobi map

Figure 6.3: Relative fixed-point residual histories from Tiamat single-rod tests at 100% power for Picard iteration and Anderson acceleration with various storage depth parameters

for the Gauss-Seidel map, but referring to Figure 6.1, we see that is a matter of the choice of damping factor. For the other fixed-point maps, Anderson-1 is actually the slowest, and in general there is not a significant difference in the number of iterations to convergence for each storage depth parameters. Overall, the apparent rate of convergence for each choice of storage depth parameter is very similar, so due to the weaker performance of Anderson-1 for smaller damping factors which was noted above, we consider Anderson-2 from this point forward.

Comparison of Fixed-Point Maps

From Table 6.1 and Figure 6.2 we can compare the performance of the three considered fixed-point maps. As expected, we generally observe that the Gauss-Seidel map converges in the fewest iterations, followed by the intermediate map, and then the Jacobi map. Again, this is expected because Gauss-Seidel treats the applications in the most tightly coupled manner by transferring updated information as soon as it is obtained, while Jacobi treats them in the most weakly coupled manner by solving every application simultaneously. While Figure 6.2 shows that iteration counts generally seem somewhat lower for the intermediate map than the Jacobi map, they are fairly comparable and significantly higher than what we observe for the Gauss-Seidel map. It then seems that the strength of coupling between Bison and MPACT is rather strong, as the weaker coupling that the intermediate map imposes between these applications due to their simultaneous solution results in a significant rise from Gauss-Seidel in iterations to convergence.

With respect to timings, in Table 6.1 we observe that Anderson with the Gauss-Seidel map again generally does best. This problem, however, is not ideal for gauging the timing performance of the intermediate or Jacobi maps. Recall that in Table 2.2, which broke down application timings for Picard simulations for this problem, we observed that the Bison solve dominated the time per iteration. The advantage of the intermediate and Jacobi maps derives from the fact that the simultaneous solution of applications results in a lower time per iteration than Gauss-Seidel. This advantage disappears if the applications do not require roughly the same solve time. In this case the per-iteration time for each of the three maps is essentially equal to the Bison solve time. It is possible that the intermediate or Jacobi map may outperform Gauss-Seidel given these iteration counts, but this would require better balance in the application solve times. Still, as we observe the iteration counts for the intermediate map are generally comparable to the Jacobi map, it seems that this map is unlikely to be best in many cases. If solve times are poorly balanced, the additional iterations required will make it worse than Gauss-Seidel. Conversely, if solve times are well balanced, its iteration count will be similar to Jacobi, but with significantly higher time per iteration. As a result of this, we do not consider the intermediate map from this point forward.

Agreement Between Anderson and Picard Solutions

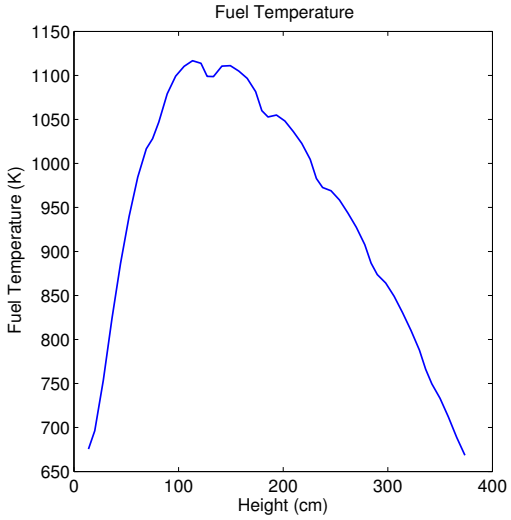
We now consider the level of accuracy of the solutions produced by Anderson acceleration. There has been work performed benchmarking the solutions computed by Tiamat against results from the VERA core simulator, which consists of a stand-alone coupling between MPACT and CTF [12] . In this, only the block Gauss-Seidel Picard iteration is considered, so we are satisfied if the solutions we obtain from the Anderson acceleration simulations are sufficiently close to the block Gauss-Seidel Picard solutions. Beyond the benchmarking that has been performed for Tiamat with Picard iteration, Picard and Anderson are solving the same fixed-point problem, so we expect that they should produce approximately the same solution.

We first refer again to Table 6.1. In this table, we list the dominant eigenvalue k_{eff} and the maximum fuel temperature $T_{f,max}$ resulting from solving the fully-coupled problem at various power levels with both Picard and Anderson-2. Within a given power level, the eigenvalues agree fairly closely, with no two eigenvalues differing by more than 10 pcm (pcm is a unit referring to the fifth decimal place in the eigenvalue). We also note that the difference in the maximum fuel temperature for any two simulations within a power level is less than 2 Kelvin.

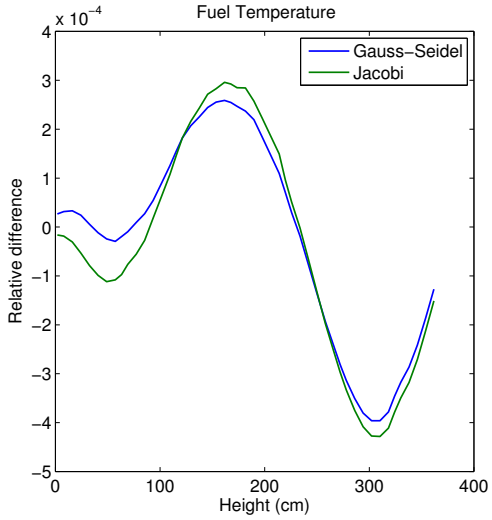
To explore the level of agreement further, we consider Figures 6.4, 6.5, 6.6, and 6.7. These figures display the solution fuel temperature, clad temperature, fission rate, and heat flux computed by block Gauss-Seidel Picard iteration, which we take as reference solutions, and additionally show the relative differences in these quantities computed by Anderson acceleration with both the Gauss-Seidel and Jacobi fixed-point maps. We observe that there is generally good agreement between the Anderson solutions and the Picard solution. Generally the last convergence criteria to be satisfied in simulations with the currently utilized termination criteria is the power tolerance, and in for these test we utilize a power convergence tolerance of $\epsilon_q = 10^{-4}$. The relative differences observed in these figures are essentially on this order of magnitude, so it seems that the relative differences we observe can likely be attributed to the global convergence criteria. We note that within each figure, the Anderson acceleration relative difference curves have generally the same shape. This could indicate that there is some minor bias introduced by utilizing Anderson acceleration, or this could indicate that the Anderson solutions are actually of higher accuracy than the Picard solution.

Sensitivity to Single-Physics Solve Tolerances

We now consider the effect that varying the individual single-physics solve tolerances has on the behavior of both Picard iteration and Anderson acceleration. Each of the application codes only solves its set of physics approximately and has its own set of tolerances which are used to determine a successful solve. As a result, when solving an single set of physics, we obtain some approximate solution $\hat{x}_i$ rather than the exact solution x_i. Hence, we can more accurately

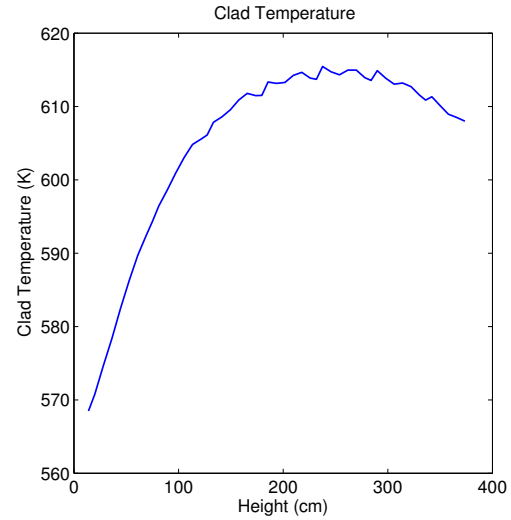


(a) Average fuel temperature computed by Picard iteration

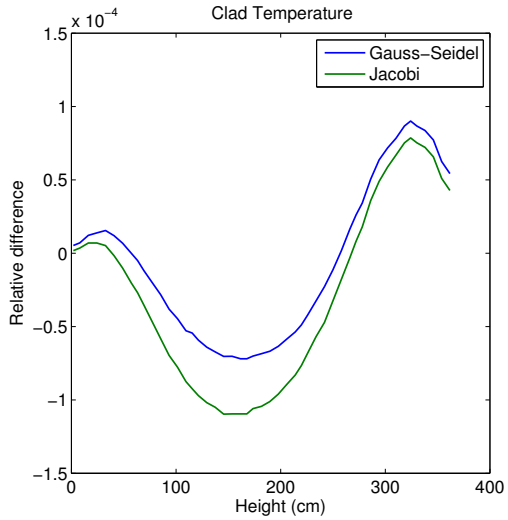


(b) Relative difference between Anderson acceleration and Picard solutions

Figure 6.4: Average fuel temperature computed by Picard iteration with Gauss-Seidel map, and relative difference between this curve and Anderson solutions

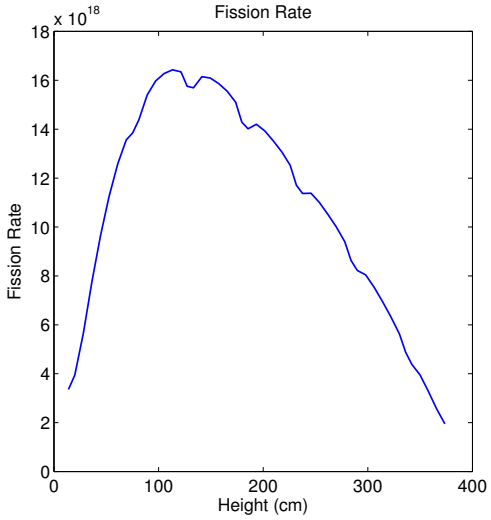


(a) Average clad temperature computed by Picard iteration

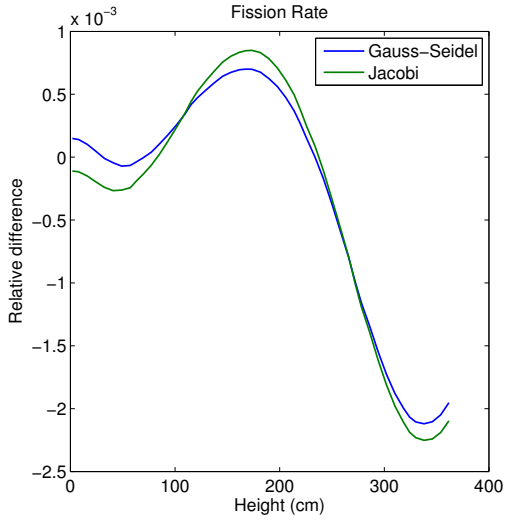


(b) Relative difference between Anderson acceleration and Picard solutions

Figure 6.5: Average clad temperature computed by Picard iteration with Gauss-Seidel map, and relative difference between this curve and Anderson solutions

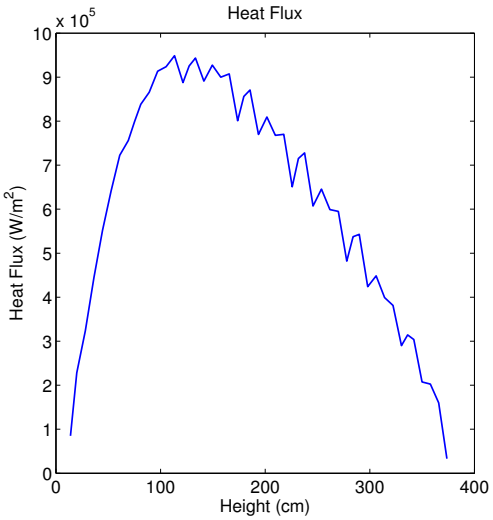


(a) Average fission rate computed by Picard iteration

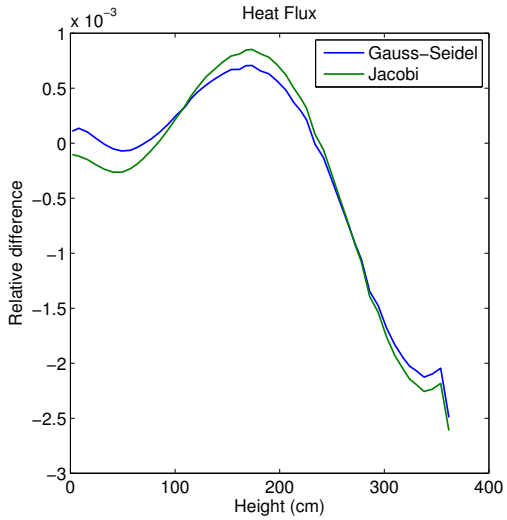


(b) Relative difference between Anderson acceleration and Picard solutions

Figure 6.6: Average fission rate computed by Picard iteration with Gauss-Seidel map, and relative difference between this curve and Anderson solutions



(a) Average heat flux computed by Picard iteration



(b) Relative difference between Anderson acceleration and Picard solutions

Figure 6.7: Average heat flux computed by Picard iteration with Gauss-Seidel map, and relative difference between this curve and Anderson solutions

represent the fixed-point maps to which we apply Anderson acceleration as

$$\hat{G}_{GS}\begin{pmatrix} T_f \\ T_w \\ \rho_w \\ T_c \end{pmatrix} = \begin{pmatrix} r_{M,B}(\hat{x}_B(T_c, r_{B,M}(\hat{x}_M(T_f, T_w, \rho_w)))) \\ r_{M,C,T}(\hat{x}_C(r_{C,B}(\hat{x}_B(T_c, r_{B,M}(x_M(T_f, T_w, \rho_w)))))) \\ r_{M,C,\rho}(\hat{x}_C(r_{C,B}(\hat{x}_B(T_c, r_{B,M}(\hat{x}_M(T_f, T_w, \rho_w)))))) \\ r_{B,C}(\hat{x}_C(r_{C,B}(\hat{x}_B(T_c, r_{B,M}(\hat{x}_M(T_f, T_w, \rho_w)))))) \end{pmatrix}, \tag{6.18}$$

and

$$\hat{G}_{JAC}\begin{pmatrix} T_f \\ T_w \\ \rho_w \\ T_c \\ q \\ q'' \end{pmatrix} = \begin{pmatrix} r_{M,B}(\hat{x}_B(T_c, q)) \\ r_{M,C,T}(\hat{x}_C(q'')) \\ r_{M,C,\rho}(\hat{x}_C(q'')) \\ r_{B,C}(\hat{x}_C(q'')) \\ (1-\omega)q + \omega\, r_{B,M}(\hat{x}_M(T_f, T_w, \rho_w)) \\ r_{C,B}(\hat{x}_B(T_c, q)) \end{pmatrix}. \tag{6.19}$$

In this, we have simply replaced exact solves with approximate solves. It is then of interest to determine how significantly the level of accuracy with which we solve the application codes affects the performance of the solution methods. Supposing that we can in some way bound the deviation between the evaluation of the fixed-point map with approximate and exact application solves, then the results from Section 3.5 provide some insight. We note that this will be the case for the Jacobi map if the application solve tolerances guarantee that the approximate solutions are sufficiently close to to the exact solutions, i.e. $\|\hat{x}_B(T_c, q) - x_B(T_c, q)\| < \epsilon_B$, $\|\hat{x}_C(q'') - x_C(q'')\| < \epsilon_C$, and $\|\hat{x}_M(T_f, T_w, \rho_w) - x_M(T_f, T_w, \rho_w)\| < \epsilon_M$, for some $\epsilon_B, \epsilon_C, \epsilon_M > 0$, and that the transfer functions are Lipschitz continuous (there is $L_{i,j}$ such that $\|r_{i,j}(x_j) - r_{i,j}(x'_j)\| \leq L_{i,j}\|x_j - x'_j\|$). With these assumptions, and letting $G(u)$ denote error-free evaluation of the fixed-point map, we have

$$\begin{aligned} \|\hat{G}_{JAC}(u) - G_{JAC}(u)\| &\leq (L_{M,B} + L_{C,B})\|\hat{x}_B(T_c, q) - x_B(T_c, q)\| \\ &\quad + (L_{B,C} + L_{M,C,T} + L_{M,C,\rho})\|\hat{x}_C(q'') - x_C(q'')\| \\ &\quad + \omega L_{B,M}\|\hat{x}_M(T_f, T_w, \rho_w) - x_M(T_f, T_w, \rho_w)\| \\ &\leq (L_{M,B} + L_{C,B})\epsilon_B + (L_{B,C} + L_{M,C,T} + L_{M,C,\rho})\epsilon_C \\ &\quad + \omega L_{B,M}\epsilon_M. \end{aligned}$$

Hence, we have a uniform bound on the difference between the approximate and exact evaluations, and this bound is proportional to the quality of the solution returned by each application. For the Gauss-Seidel map, we need to additionally assume that the exact Bison solution is Lipschitz continuous with respect to the power (there is L_B such that $\|x_B(T_c, q_1) - x_B(T_c, q_2) \leq L_B\|q_1 - q_2\|$) and that the exact CTF solution is Lipschitz continuous with respect to the heat

flux (there is L_C such that $\|x_C(q_1'') - x_C(q_2'')\| \leq L_C\|q_1'' - q_2''\|$). With this, in addition to what was assumed for the Jacobi map, we have

$$\begin{aligned}\|\hat{G}_{GS}(u) - G_{GS}(u)\| \leq L_{M,B}\|\hat{x}_B(T_c, r_{B,M}(\hat{x}_M(T_f, T_w, \rho_w))) - x_B(T_c, r_{B,M}(x_M(T_f, T_w, \rho_w)))\| \\ + (L_{B,C} + L_{M,C,T} + L_{M,C,\rho})\|\hat{x}_C(r_{C,B}(\hat{x}_B(T_c, r_{B,M}(\hat{x}_M(T_f, T_w, \rho_w))))) \\ - x_C(r_{C,B}(x_B(T_c, r_{B,M}(x_M(T_f, T_w, \rho_w)))))\|.\end{aligned}$$

Note the differences within these norms are between approximate and exact solutions to different problems. Then, to bound these quantities, we compare the approximate solution against the exact solution for the problem it is actually solving, and then bound the difference between the exact solutions to the different problems. In this manner, we bound the difference between the Bison solutions as

$$\begin{aligned}&\|\hat{x}_B(T_c, r_{B,M}(\hat{x}_M(T_f, T_w, \rho_w))) - x_B(T_c, r_{B,M}(x_M(T_f, T_w, \rho_w)))\| \\ &\quad \leq \|\hat{x}_B(T_c, r_{B,M}(\hat{x}_M(T_f, T_w, \rho_w))) - x_B(T_c, r_{B,M}(\hat{x}_M(T_f, T_w, \rho_w)))\| \\ &\quad + \|x_B(T_c, r_{B,M}(\hat{x}_M(T_f, T_w, \rho_w))) - x_B(T_c, r_{B,M}(x_M(T_f, T_w, \rho_w)))\| \\ &\quad \leq \epsilon_B + L_B L_{B,M}\|\hat{x}_M(T_f, T_w, \rho_w) - x_M(T_f, T_w, \rho_w)\| \leq \epsilon_B + L_B L_{B,M}\epsilon_M.\end{aligned}$$

With this, we similarly bound the difference between the CTF solutions as

$$\begin{aligned}&\|\hat{x}_C(r_{C,B}(\hat{x}_B(T_c, r_{B,M}(\hat{x}_M(T_f, T_w, \rho_w))))) - x_C(r_{C,B}(x_B(T_c, r_{B,M}(x_M(T_f, T_w, \rho_w)))))\| \\ &\quad \leq \|\hat{x}_C(r_{C,B}(\hat{x}_B(T_c, r_{B,M}(\hat{x}_M(T_f, T_w, \rho_w))))) - x_C(r_{C,B}(\hat{x}_B(T_c, r_{B,M}(\hat{x}_M(T_f, T_w, \rho_w)))))\| \\ &\quad + \|x_C(r_{C,B}(\hat{x}_B(T_c, r_{B,M}(\hat{x}_M(T_f, T_w, \rho_w))))) - x_C(r_{C,B}(x_B(T_c, r_{B,M}(x_M(T_f, T_w, \rho_w)))))\| \\ &\quad\quad \leq \epsilon_C + L_C L_{C,B}\|\hat{x}_B(T_c, r_{B,M}(\hat{x}_M(T_f, T_w, \rho_w))) - x_B(T_c, r_{B,M}(x_M(T_f, T_w, \rho_w)))\| \\ &\quad\quad\quad \leq \epsilon_C + L_C L_{C,B}(\epsilon_B + L_B L_{B,M}\epsilon_M).\end{aligned}$$

With the above, we have the following bound for the error in the evaluation of the Gauss-Seidel map

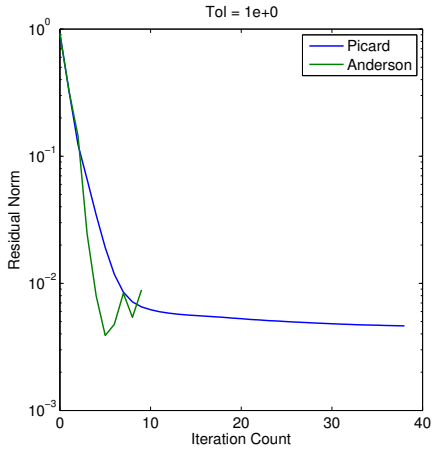
$$\begin{aligned}\|\hat{G}_{GS}(u) - G_{GS}(u)\| \leq (L_{B,C} + L_{M,C,T} + L_{M,C,\rho})\epsilon_C \\ + [L_{M,B} + (L_{B,C} + L_{M,C,T} + L_{M,C,\rho})L_C L_{C,B}](\epsilon_B + L_B L_{B,M}\epsilon_M).\end{aligned}$$

Again, this gives a bound on the fixed-point map evaluation error proportional to the error in each of the single physics solves. This bound has potential to be significantly larger than what was obtained for the Jacobi map, at least with respect to the Bison and MPACT errors, which makes sense due to propagation of error between sequential single-physics solves. Then,

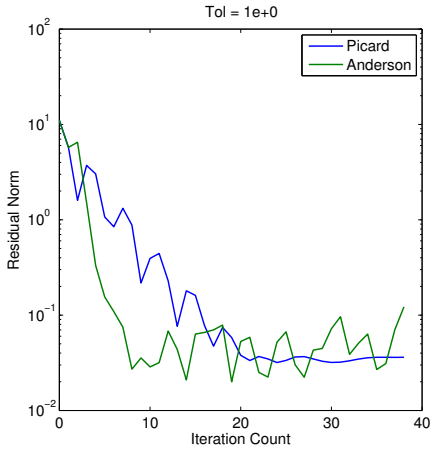
supposing that the above assumptions hold, for both Picard and Anderson the theory from Section 3.5 predicts r-linear convergence in the error and residual up to some stagnation point which is proportional to the size of the error in the fixed-point map evaluation.

To test this, we first consider sensitivity to CTF solve tolerances. We outlined the tolerances that CTF uses to determine whether it has become sufficiently steady state in Section 2.2.2. Again, it measures the change in 5 quantities from time step to time step, and declares convergence when they are sufficiently small. Since CTF internally solves a time dependent problem, the error in the CTF solution represents a deviation from steady-state. Rather than considering each of these tolerances, we choose a single tolerance to vary. For this, we choose the global energy balance tolerance. In order to test only this tolerance, we set the values for the other tolerances very large so that only the specified tolerance determines the number of time steps taken per CTF solve. We note here that when CTF models only the coolant region, as in the fully-coupled Tiamat solve phase, the solid energy storage tolerance is not utilized. This is important as CTF does model the solid region in the HFP estimation phase, so this allows us to tune this solid energy storage tolerance so that Bison is ramped to the same HFP estimate conditions, regardless of how loose or tight we set the global energy balance tolerance. Results from varying the CTF global energy balance tolerance for both the Gauss-Seidel and Jacobi maps are given in Figure 6.8. Figures 6.8a and 6.8b show results from utilizing a very loose 1% tolerance. During the coupled solve, this resulted in CTF declaring convergence after only one time step in each CTF solve. We see that both Picard and Anderson approach a stagnation point, and this occurs in both maps. Only Anderson with the Gauss-Seidel map declares convergence, but it does so at a point far from the actual solution. Note that in each case, the residuals for Picard and Anderson stagnate about approximately the same point. While the theory from Section 3.5 suggested that Anderson may amplify the error in the function evaluation more than Picard, especially if the fixed-point map is weakly contractive, we do not observe this here. The residuals for Anderson simply appear more jagged about the same stagnation point as Picard. In the remaining figures, we reduce the tolerances to 0.01% and 0.0001%. With the relatively loose 0.01% tolerance, Anderson and Picard perform comparably using the Gauss-Seidel map, and Anderson performs noticeably better with the tighter 0.0001% tolerance. In both cases, there is a reduction in iterations to convergence with the tighter tolerance. With the Jacobi map, Anderson significantly outperforms Picard at both 0.01% and 0.0001%, and there is marginal improvement for both Picard and Anderson when the tolerance is reduced from 0.01% to 0.0001%.

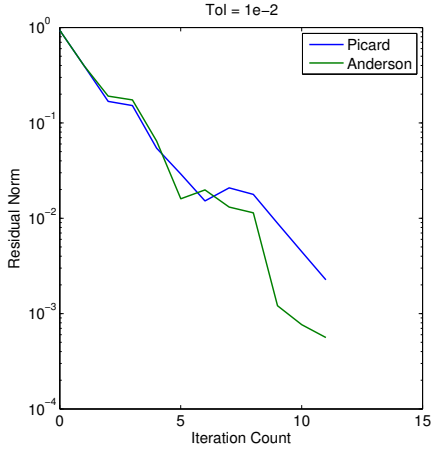
Next, we consider the convergence criteria for MPACT. MPACT determines convergence base upon the relative change in the group scalar flux and absolute change in the dominant eigenvalue between coarse-mesh finite-difference iterations. For these tests, we choose to vary only the tolerance for the change in the scalar flux. Figure 6.9 shows results from solving the



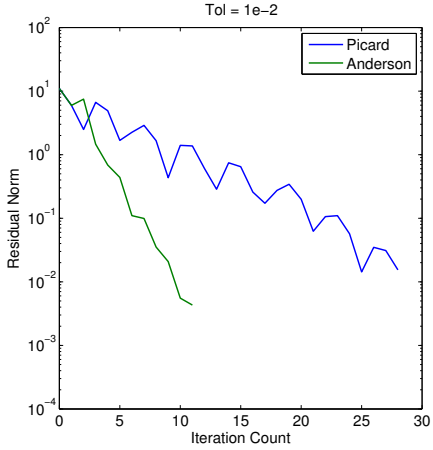
(a) Gauss-Seidel map, tolerance = 1%



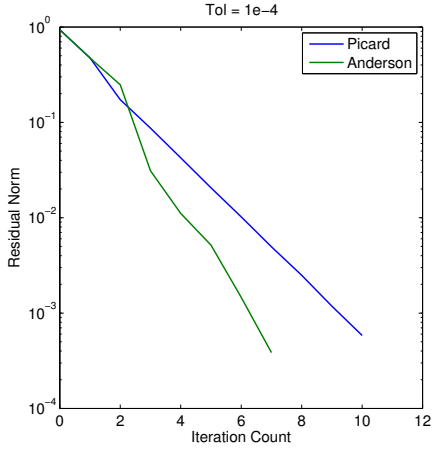
(b) Jacobi map, tolerance = 1%



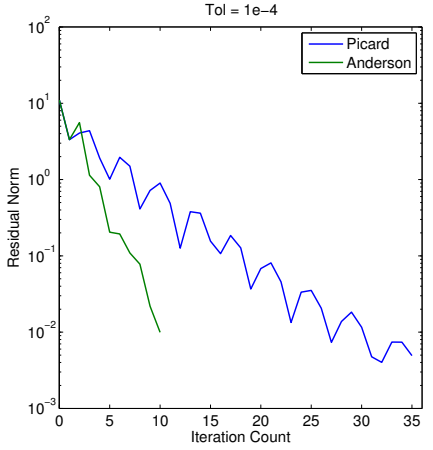
(c) Gauss-Seidel map, tolerance = 0.01%



(d) Jacobi map, tolerance = 0.01%

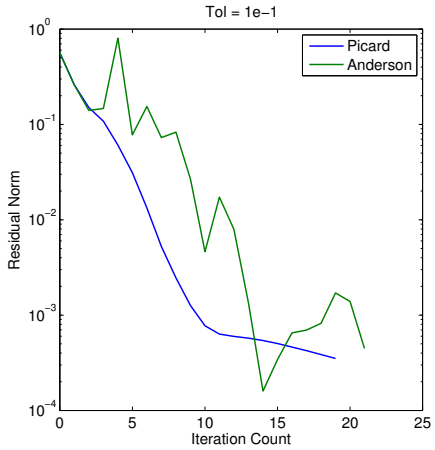


(e) Gauss-Seidel map, tolerance = 0.0001%

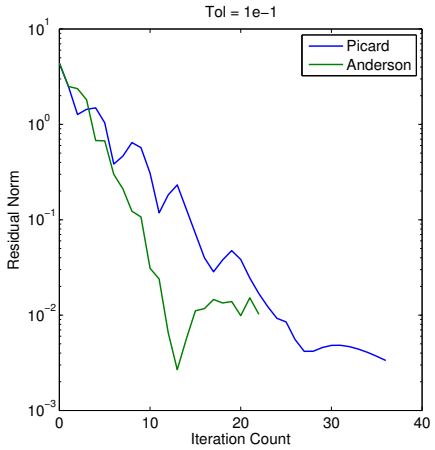


(f) Jacobi map, tolerance = 0.0001%

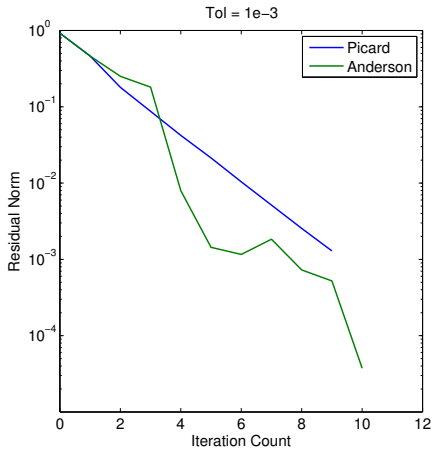
Figure 6.8: Varying CTF global energy balance tolerance in Tiamat single-rod tests



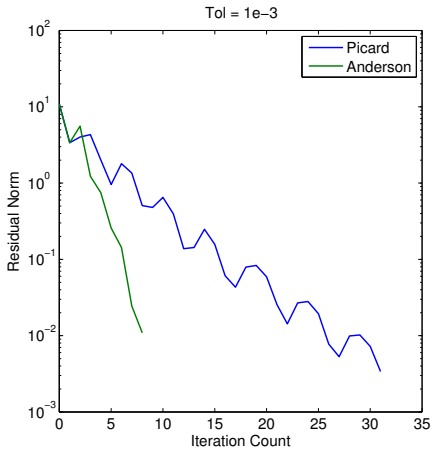
(a) Gauss-Seidel map, tolerance = 1.0e-1



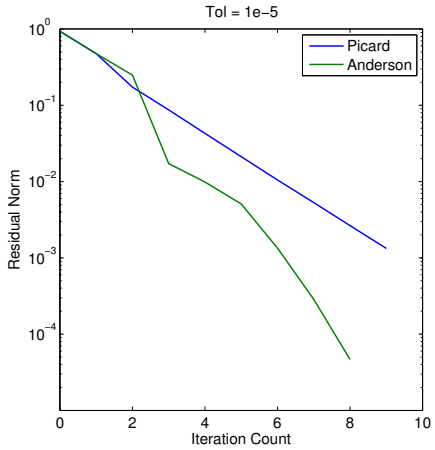
(b) Jacobi map, tolerance = 1.0e-1



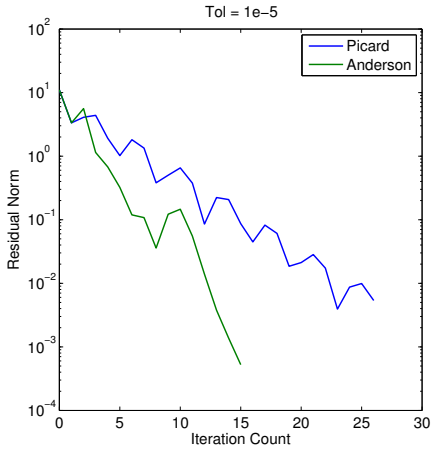
(c) Gauss-Seidel map, tolerance = 1.0e-3



(d) Jacobi map, tolerance = 1.0e-3



(e) Gauss-Seidel map, tolerance = 1.0e-5

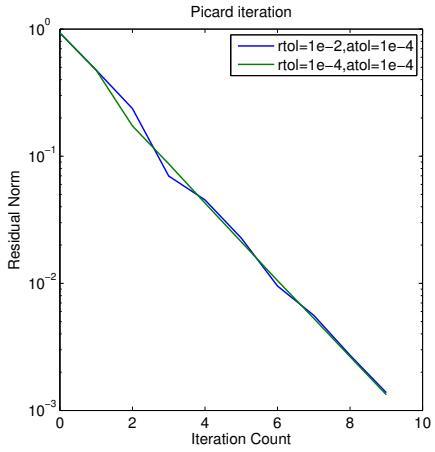


(f) Jacobi map, tolerance = 1.0e-5

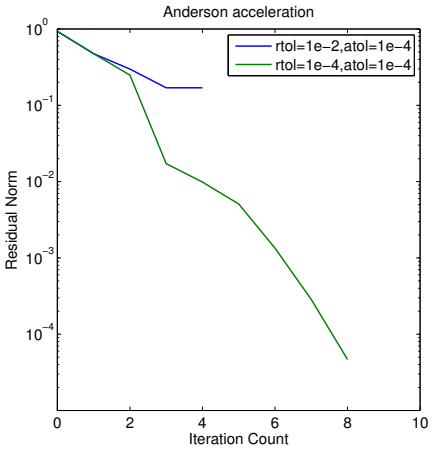
Figure 6.9: Varying MPACT scalar flux tolerance in Tiamat single-rod tests

coupled problem with both Picard and Anderson at various scalar flux convergence tolerances. The results are fairly similar to what was observed when varying the CTF convergence criteria. In Figures 6.9a and 6.9b we utilize a fairly loose tolerance, and we observe somewhat good convergence up until a stagnation point is reached. As before, this stagnation point occurs at roughly the same residual norm for both Picard and Anderson, so it does not seem that Anderson significantly amplifies the error in the function evaluation. At an intermediate tolerance of $\tau =$ 1e-3, we see that Anderson and Picard perform comparably with the Gauss-Seidel map, and Anderson performs significantly better than Picard with the Jacobi map. With the tightest tolerance of $\tau = 1e-5$, which is the default tolerance for MPACT, Anderson with the Gauss-Seidel map performs noticeably better than Picard. There is very little observable difference when decreasing the tolerance below this point. One thing to note is that for the Jacobi map, Anderson converges in fewer iterations with $\tau =$ 1e-3 than $\tau =$ 1e-5. However, this is due to false convergence being declared in the looser tolerance case, as the final maximum fuel temperature differs from the expected solution by approximately 3 degrees and the final eigenvalue differs by approximately 5pcm.

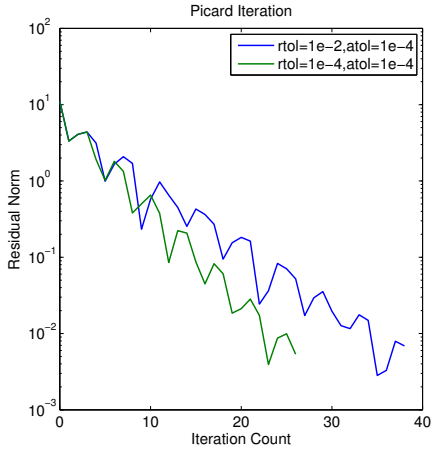
Lastly, as Bison internally utilizes JFNK, it declares convergence based on the residual norm. It utilizes an absolute/relative tolerance in which the absolute and relative tolerances τ_a and τ_r are specified, and the problem is said to be converged when $\|f_B(x_B^n)\| \leq \tau_a + \tau_r \|f_B(x_B^0)\|$, where f_B is Bison's residual. Figure 6.10 shows results from solving the coupled system with variation in both the absolute and relative tolerances. The values used in the default Bison input template are $\tau_r =$ 1e-4 and $\tau_a =$ 1e-10. First note that decreasing the relative tolerance below this value did not result in a noticeable difference in any case considered. At these base values, we never obtain a residual norm less than 1e-5, so setting the absolute tolerance to any value less than this would have no effect on the iteration. In fact, with a given relative tolerance we do not observe significant difference in any case considered when an absolute tolerance less than 1e-1 is used. Figures 6.10a and 6.10b show the effect of varying the relative tolerance in the Gauss-Seidel map. We see that Picard seems to be be rather insensitive to this change. Conversely, with a looser relative tolerance, Anderson seems to stagnate briefly before declaring convergence of the coupled system far from the solution. The remaining figures depict results from using the Jacobi map. In Figures 6.10c and 6.10d we consider variation of the relative tolerance. As with the Gauss-Seidel map, the Anderson iterations display greater sensitivity to this change. For both maps, an incorrect solution is obtained using Anderson with the looser relative tolerance. For Picard, there is a more noticeable difference from the change in the relative tolerance than for the Gauss-Seidel map, but it still converges to the expected solution. Lastly, Figures 6.10e and 6.10f show the effect of varying the absolute tolerance. As with the relative tolerance, Picard is rather insensitive to variation in this parameter. Conversely, with



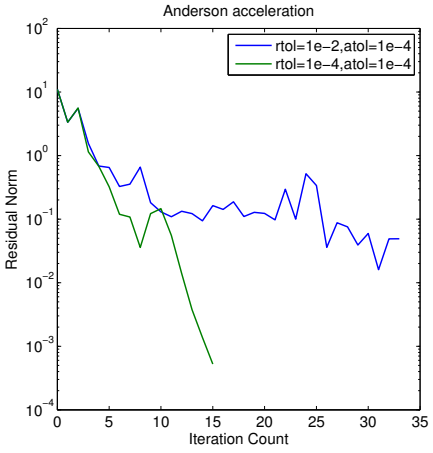
(a) Picard for Gauss-Seidel map, varying τ_r



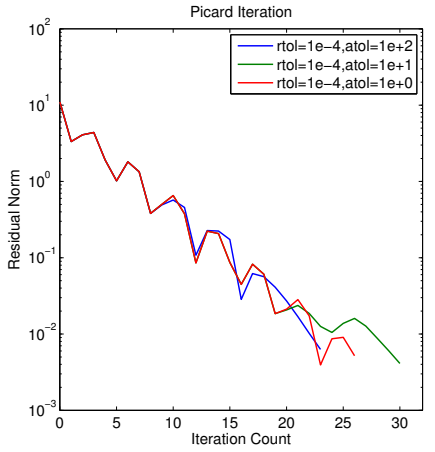
(b) Anderson for Gauss-Seidel map, varying τ_r



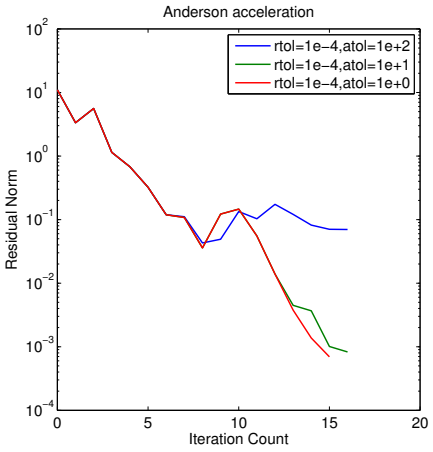
(c) Picard for Jacobi map, varying τ_r



(d) Anderson for Jacobi map, varying τ_r



(e) Picard for Jacobi map, varying τ_a



(f) Anderson for Jacobi map, varying τ_a

Figure 6.10: Varying Bison JFNK tolerances in Tiamat single-rod tests

the largest absolute tolerance, Anderson stagnates at an incorrect solution. As the tolerance is decreased it rapidly begins to perform better than Picard.

Variation of Field Scaling

We now examine the effect of varying the relative weights of the fields in our choice of the scaling matrices M_{GS} and M_{JAC}. The scaling matrices were chosen in order to bring the entries in the vector of unknowns u to roughly the same order of magnitude, but it may still be the case that some quantities are under or overrepresented in the Anderson acceleration least-squares problem. To test this, we introduce additional scaling factors into our definition of the scaling matrices. With this addition, we now define the Gauss-Seidel scaling matrix as

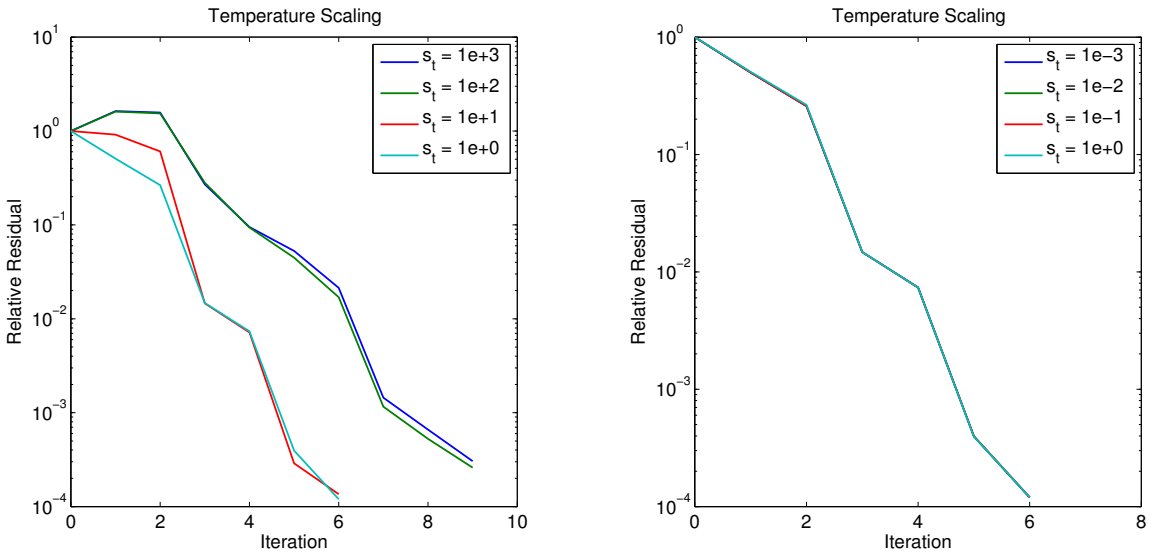
$$M_{GS} = \begin{pmatrix} [s_t \operatorname{diag}(T_f^0)]^{-1} & & & \\ & [s_t \operatorname{diag}(T_w^0)]^{-1} & & \\ & & [s_d \operatorname{diag}(\rho_w^0)]^{-1} & \\ & & & [s_t \operatorname{diag}(T_c^0)]^{-1} \end{pmatrix}, \tag{6.20}$$

and the Jacobi scaling matrix as

$$M_{JAC} = \begin{pmatrix} M_{GS} & & \\ & \frac{1}{s_p} M_q & \\ & & \frac{1}{s_h \, \bar{q}''} I \end{pmatrix}, \tag{6.21}$$

where s_t is the scaling factor for the temperatures, s_d is the density scaling factor, s_p is the power scaling factor, and s_h is the heat flux scaling factor. Note that these factors appear in the inverse, so decreasing a scaling factor will increase the weight of the corresponding field, and vice versa.

We first consider tests with the block Gauss-Seidel map. For this, we set the density scaling factor at one, and vary only the temperature scaling factor, since only the relative weight between the temperature and density fields is important. Relative residual plots from solving the coupled problem with Anderson acceleration for various temperature scaling factors are shown in Figure 6.11. Again, scaling factors larger than one will give the temperature fields less weight in the Anderson least-squares problem, so changes in density have a larger effect. Conversely scaling factors smaller than one weigh temperature changes more heavily in the least-squares problem. In Figure 6.11a we see results with scaling factors larger than one. We see that as the scaling factor is made larger, and relatively more weight is given to density changes in the least-squares problem, convergence slows, and the number of iterations required for convergence increases by 50%. The very small change in the residual plots from s_t =1e+2 to s_t =1e+3 would seem to indicate that in both cases, density changes are dominating the



(a) Temperature scaling factors $s_t \geq 1.0$

(b) Temperature scaling factors $s_t \leq 1.0$

Figure 6.11: Varying temperature scaling for Anderson-2 single-rod Tiamat tests with the Gauss-Seidel map

Anderson least-squares problem, and giving density more weight by increasing s_t further would not significantly affect the iteration. Figure 6.11b shows results with scaling factor less than one, and the curves in this plot are not visibly distinguishable. This would seem to indicate that at the base case of $s_t = 1.0$, temperature changes dominate the residual, as giving these more weight does not visibly affect the convergence behavior. This however does not seem to be a problem, as weighing density more heavily slowed convergence, as we saw previously. It then seems unlikely that any choice of parameter will result in faster convergence than what is obtained from our original scaling with $s_t = 1.0$.

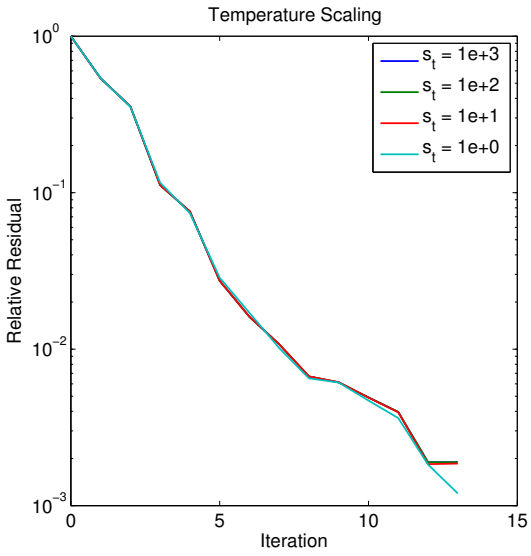
We next consider the block Jacobi scheme. We now have more parameters to vary, so for these tests we choose a single factor to vary over the values $10^{-3}, 10^{-2}, 10^{-1}, 10^{+0}$, $10^{+1}, 10^{+2}, 10^{+3}$, and keep the other factors frozen at 1.0. Figures 6.12 and 6.13 present results from following this approach with each of the four scaling factors. Figures 6.12a and 6.12b show relative residual plots from varying the temperature scaling factor. We see that for scaling factors $s_t \geq 1.0$ the convergence behavior does not change significantly, but giving the temperature changes more weight in the residual by letting $s_t < 1.0$ has an effect on the iteration. This would indicate that at $s_t = 1.0$, the temperature changes are dominated by the rest of the residual. Despite this, giving the temperatures additional weight in the residual does not result in a noticeable gain in performance. In fact, each factor $s_t < 1.0$ required an additional iteration to achieve

convergence. Figures 6.12c and 6.12d show results from varying the density scaling, and the results are largely similar to what is observed for the temperature scaling. There is little visible difference in the iteration for any scaling factor $s_d \geq 1.0$ and performance is fairly consistent as s_d is decreased below 1.0. At s_d =1e-3, density changes begin to dominate the residual and the convergence worsens. In any case, the base choice of $s_d = 1.0$ again converges in the fewest iterations. In Figures 6.13a and 6.13b, the residual plots with power scaling varied, we now observe changes in behavior for scaling factors both larger and smaller than 1.0. This indicates that at $s_p = 1.0$, the power changes are a significant, but not dominating, component of the residuals. As the scaling factor is increased above 1.0, the rate of convergence becomes consistently slower, and as the scaling factor is decreased below 1.0, the residual curves become less smooth but the overall convergence rate remains roughly the same as for $s_p = 1.0$. Again, there is no obvious trend that would suggest that there is an advantage to be gained by tuning this parameter. Lastly, Figures 6.13c and 6.13d show results from varying the heat flux scaling. Like the power scaling, there is an observable change in behavior for factors larger and smaller than 1.0. As we have observed throughout, there is no compelling trend that would suggest that any choice of scaling parameter different from $s_h = 1.0$ would result in obvious improvement. Lastly note, that Figures 6.13d is a bit unique in the increasingly large spike in the residual at iteration 1. This is due to the ramp to hot full-power prior to the coupled solve. The first Bison solve in the fully-coupled solve is for a small time step using the same coupling parameters as the final step in the HFP estimation, so the difference in the heat flux resulting from these two solves is negligible. When updated power and clad temperatures are provided to Bison at the next coupled iteration, an appreciable change in the heat flux is obtained, and the increased weight given to the heat flux in the residual is reflected by this spike.

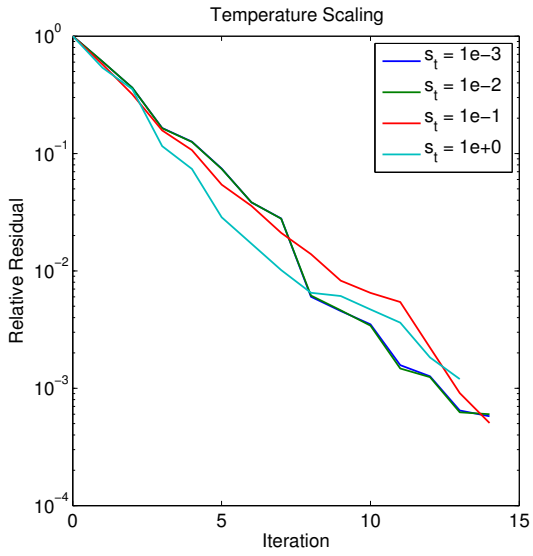
For both the Gauss-Seidel and Jacobi maps, we see that varying the scaling of the fields can significantly impact the convergence behavior. However in each case, our original choice of scaling converged in the fewest iterations of the cases considered. With the Jacobi map, we have several parameters we can vary, and it may be the case that we could obtain improved performance with changes to multiple scaling factors. However, this approach does not seem practical, as a better choice of scaling factors may be problem dependent.

Comparison with JFNK

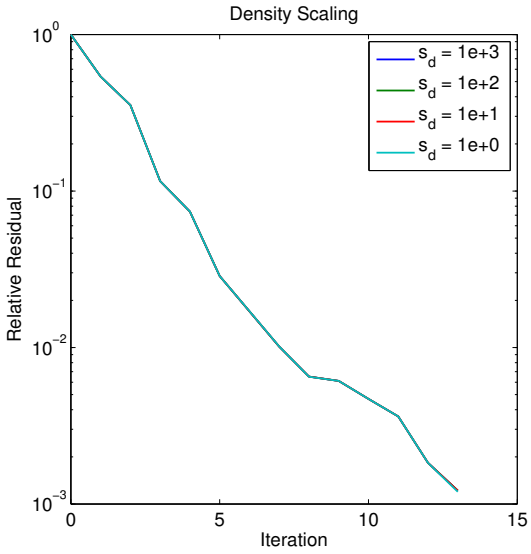
Expressed in the forms (6.4) and (6.5), each of the fixed-point maps corresponds to a fixed-point residual which we can both compute and access. In fact, with respect to implementation, NOX expects the function evaluation to be provided in residual form, so these fixed-point residuals are what is actually computed in the model evaluator. Hence, we can additionally compare the results obtained using Anderson acceleration against results from solving these fixed-point



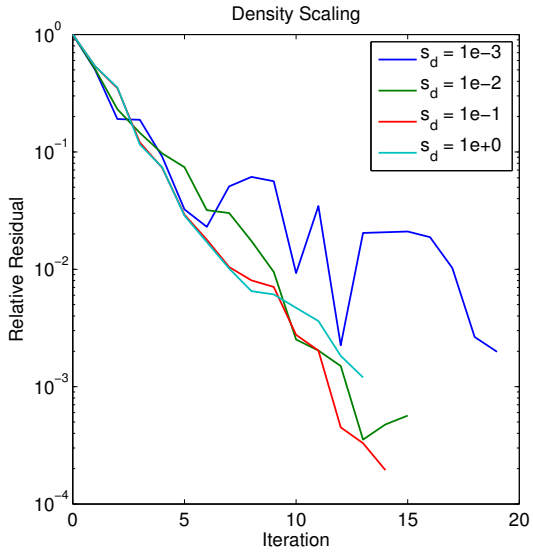
(a) Temperature scaling factors $s_t \geq 1.0$



(b) Temperature scaling factors $s_t \leq 1.0$

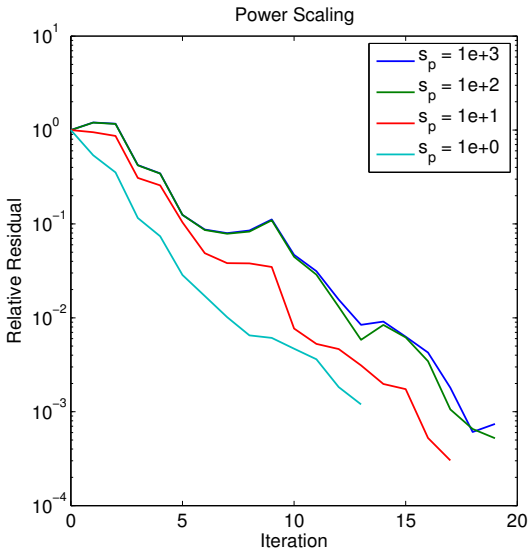


(c) Density scaling factors $s_d \geq 1.0$

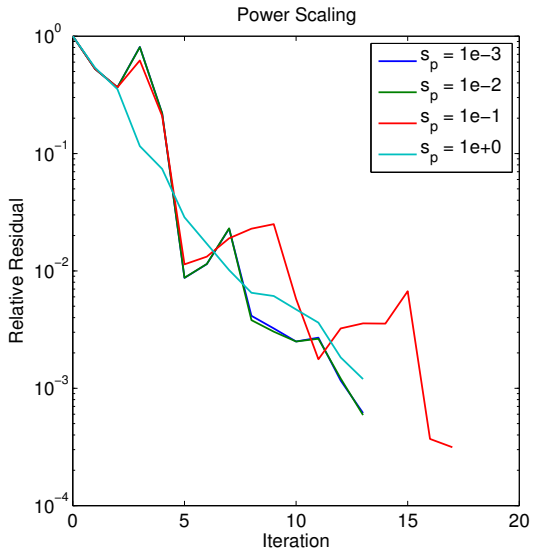


(d) Density scaling factors $s_d \leq 1.0$

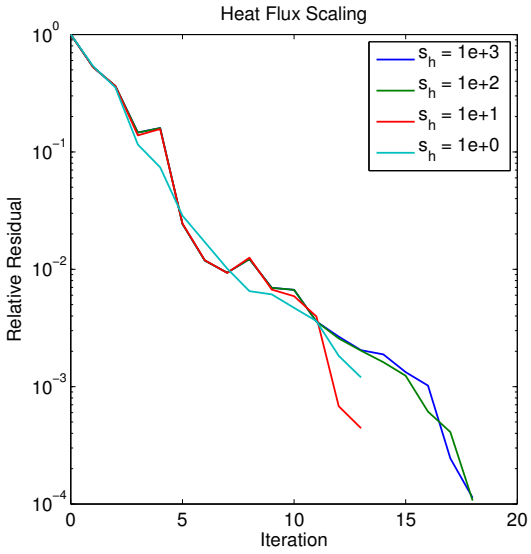
Figure 6.12: Varying temperature and density scaling for Anderson-2 single-rod Tiamat tests with the Jacobi map



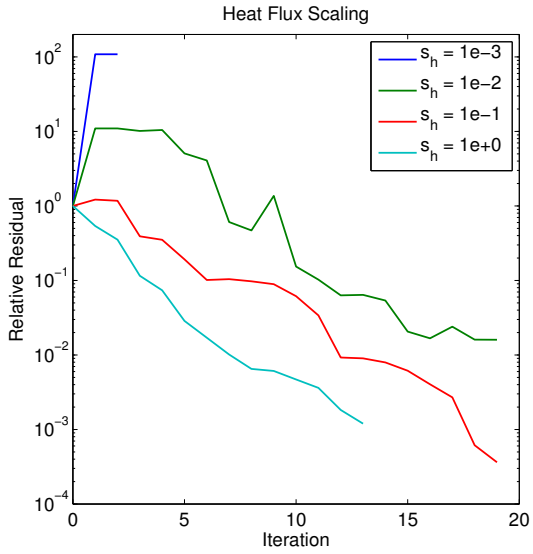
(a) Power scaling factors $s_p \geq 1.0$



(b) Power scaling factors $s_p \geq 1.0$

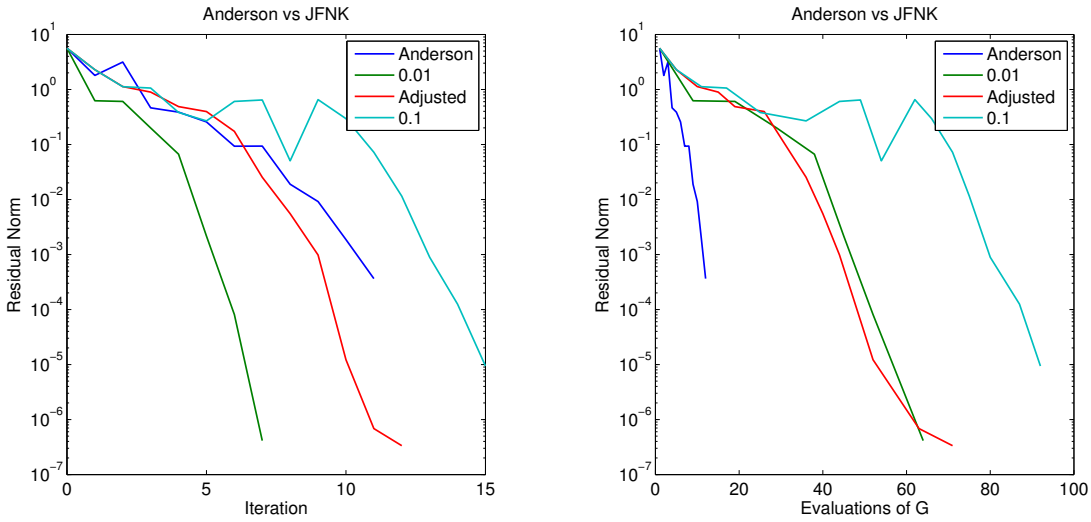


(c) Heat flux scaling factors $s_h \geq 1.0$



(d) Heat flux scaling factors $s_h \leq 1.0$

Figure 6.13: Varying power and heat flux scaling for Anderson-2 single-rod Tiamat tests with the Jacobi map



(a) Residual norm vs iteration

(b) Residual norm vs evaluations of G

Figure 6.14: Comparison of Anderson-2 and JFNK with block Jacobi map for Tiamat single-rod tests. JFNK uses a constant forcing term of 0.1 or 0.01, or an adjustable forcing term with initial value 0.1

residual equations using JFNK. There is minimal marginal work required in order to attempt to solve these residual systems by JFNK rather than Anderson acceleration. The only significant code alteration that must be made relates to creation of the forward difference Jacobian-vector operator and providing information about the linear solve strategy. The forward difference operator included in NOX has several options for computing the perturbation δ in the forward difference approximation $F'(u)v \approx \frac{F(u+\delta v)-F(u)}{\delta}$. We utilize the "KSP NOX 2001" option, which computes

$$\delta = \lambda \left(\frac{10^{-12}}{\lambda} + \frac{|u^T v|}{\|v\|^2} \right) \frac{u^T v}{|u^T v|}, \tag{6.22}$$

where λ is a user defined parameter. For the linear solve, we utilize the "Pseudo Block GMRES" iterative linear solver from the Trilinos package Belos (see Section B.2.5), which is a standard GMRES implementation included in this package. "Pseudo Block" refers with the strategy for solving with multiple right hand sides, but we only utilize this to solve equations with single right hand sides.

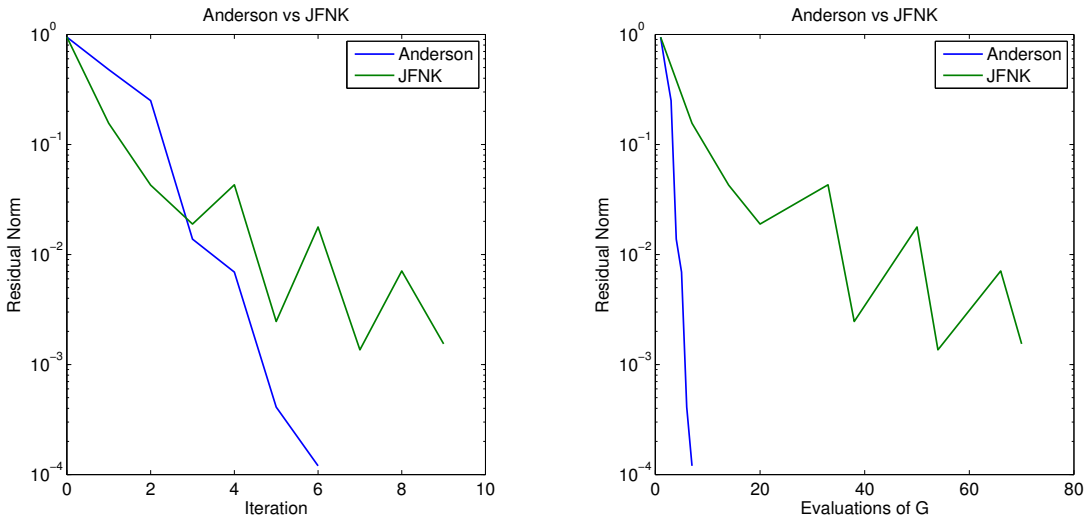
We first consider the block Jacobi fixed-point map. Results comparing the performance of Anderson acceleration and JFNK with this map are shown in Figure 6.14. For these tests, JFNK does not utilize a line search. In order to minimize the combination of the finite difference approximation error and the error in the function evaluation, the parameter λ in the Jacobian-vector product approximation should be chosen to be roughly the square root of the error in

the function evaluation [29]. With the standard application solve tolerances that have been used to this point, this requires a very large perturbation, which results in large error in the Jacobian-vector product approximation and a poor Newton direction. Because of this, for these tests we reduced several application solve tolerances, and we set the perturbation parameter $\lambda = 10^{-4}$. We reduced the MPACT scalar flux tolerance to 10^{-8}, and for CTF we set fluid/solid energy storage and mass storage tolerances to 5×10^{-6} and the global energy/mass balance tolerances to 10^{-4}. For the computation of the forcing term in the linear solve, we consider three strategies: a constant forcing term of 0.1, a constant forcing term of 0.01, and an adjustable forcing term with initial value 0.1. The adjustable forcing term is computed by

$$\eta_k = \gamma \left(\frac{\|F_k\|}{\|F_{k-1}\|} \right)^{\alpha}, \tag{6.23}$$

where it is enforced that $\max\{\gamma\eta_{k-1}^{\alpha}, \eta_{\min}\} \leq \eta_k \leq \eta_{\max}$, and we let $\gamma = 0.9, \alpha = 1.5, \eta_{\min} = 10^{-4}$, and $\eta_{\max} = 0.9$. In Figure 6.14, we observe that with respect to iterations to convergence, Anderson and JFNK are somewhat comparable. With forcing term 0.1, the iteration struggles initially, but otherwise each JFNK iteration displays fast local convergence as expected. With forcing term 0.01, JFNK converges several iterations before all the other methods. Anderson and JFNK with adjustable forcing term converge in the same number of coupled iterations, and JFNK with forcing term 0.1 requires several more iterations than the other methods. We note that in each JFNK iteration reduced the residual norm several orders of magnitude lower than the level that was required for Anderson to be declared converged. This may be due to the manner of determining convergence of the coupled system. The iteration terminates on small changes in various response functions from iteration to iteration, and as JFNK takes larger steps toward the solution, this could result in larger changes in these responses until the iteration is very close to the solution.

With respect to evaluations of the fixed-point map, Anderson is a clear winner over each JFNK iteration. This is due to the level of work required in the computation of the Newton step. Each inner GMRES iteration requires an evaluation of the fixed-point map. The forcing term 0.1 required between 2 and 6 iterations for each GMRES solve. Similarly, the adjustable forcing term generally required between 2 and 6 iterations each GMRES solve, with one iteration for which 9 linear iterations were required. The forcing term 0.01 required between 5 and 10 iterations per GMRES solve. Additionally, during iteration, Belos utilizes a cheaper approximation to the linear residual for measuring convergence, and when that has been declared successful the actual linear residual is computed to verify convergence, which expends an additional evaluation of the fixed-point map. Hence, each nonlinear JFNK iteration will require at minimum three evaluations of the fixed-point map, but generally several more. Conversely, Anderson simply requires a single evaluation of the fixed-point map per iteration. As a result,



(a) Residual norm vs iteration

(b) Residual norm vs evaluations of G

Figure 6.15: Comparison of Anderson-2 and JFNK with block Gauss-Seidel map for Tiamat single-rod tests. JFNK uses a constant forcing term of 0.01

even if Anderson were to require several iterations more than JFNK, it will still likely require many fewer evaluations of the fixed-point map. Additionally, as the evaluation of the fixed-point map is the dominant computational cost, run times for Anderson will be significantly lower as well.

We now note that the block Jacobi map seems to be the best case scenario for JFNK. In Figure 6.15, we see a comparison of Anderson and JFNK with forcing term 0.01 for the Gauss-Seidel map. In this, we see that JFNK performs much worse than Anderson with this map, as well as worse than JFNK with the Jacobi map. As we noted previously, the error in the evaluation of this map may be larger than in the Jacobi map due to accumulation of error between successive application solves, and this larger error in fact seems to be problematic. With the perturbation parameter $\lambda = 10^{-4}$ which was used previously, JFNK performs very poorly, and in fact frequently fails by computing negative fuel temperatures. For the results shown here, we choose $\lambda = 10^{-3}$. This may still be fairly small given the size of the evaluation error, but increasing it much further will result in a very poor approximation to the Jacobian-vector product. In any case, it seems that the combination of a fairly large perturbation and large evaluation error results in the computation of a very poor Newton step near the coupled solution. The iteration repeatedly takes a good step toward the solution, and then proceeds to compute a Newton step that does not seem to be a descent direction. This is evidenced by the repeated failure of the line search which we utilize for these results. We utilize a backtracking

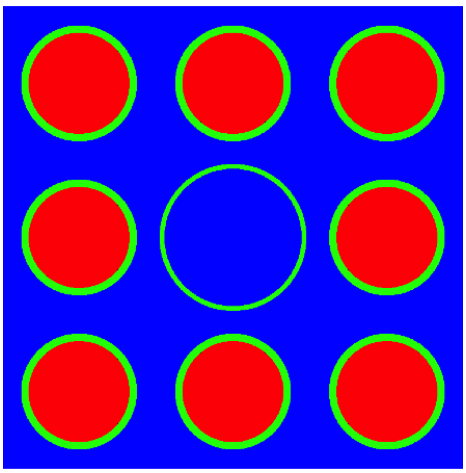


Figure 6.16: 3x3 mini-assembly layout, with 8 UO_2 fuel rods (in red) and a central guide tube

line search with a reduction factor 0.5 and a maximum of 7 step reductions. This gives a minimum step size of $7.8125e-3$, which is the step that is taken if the residual fails to achieve sufficient decrease. We note that a line search should be used with caution for this problem, as the the termination is based on changes in responses computed during the evaluation of the fixed-point map between successive nonlinear iterations rather than the residual norm, so a small computed step size may result in premature termination while the residual is still large. In any case, we again see that Anderson acceleration performs significantly better than JFNK in this context. The level of error in the evaluation of the fixed-point map can make it difficult for JFNK to perform well, and when it does performs well, too much work is required in the computation of the Newton step to be competitive with Anderson.

6.4.2 3x3 Mini-Assembly Tests

We next consider a slightly larger problem, consisting of a full 3D simulation of a 3x3 array containing 8 fuel rods around a central guide tube, illustrated by Figure 6.16. The input specifications for this problem are described in Section C.2. Some changes worth noting between the input for these test and those in the previous section are that in this case the global power convergence tolerance is increased from $\epsilon_q = 10^{-4}$ for the previous tests to $\epsilon_q = 10^{-3}$ for this test, and these tests perform a single additional subcycle of stand-alone coupled MPACT/CTF prior to ramping Bison to HFP. This subcycling improves the HFP estimate that is used during the ramp phase. As a result of these differences, these tests are run with an improved initial iterate for the fully-coupled solve and a looser convergence tolerance, so we expect that in general these will converge in few coupled iterations. These tests are run with 19 processors: 9 for both MPACT and Bison and 1 for CTF.

Table 6.2: Comparison of Picard and Anderson-2 with each of the Gauss-Seidel and Jacobi fixed-point maps for 3x3 Tiamat simulation at various power levels. Damping factor = 0.5 and max iteration count = 25

Power Level	Method	Iterations	Solve Time (s)	k_{eff}	$T_{f,max}$
25%	Picard (GS)	4	577	1.17014	483.13
	Picard (JAC)	7	678	1.17014	482.83
	Anderson-2 (GS)	5	650	1.17014	482.83
	Anderson-2 (JAC)	6	609	1.17014	482.85
50%	Picard (GS)	5	655	1.16716	681.98
	Picard (JAC)	7	735	1.16716	681.70
	Anderson-2 (GS)	5	650	1.16716	681.72
	Anderson-2 (JAC)	6	685	1.16715	681.69
75%	Picard (GS)	5	691	1.16431	890.90
	Picard (JAC)	6	720	1.16430	890.43
	Anderson-2 (GS)	5	681	1.16431	890.65
	Anderson-2 (JAC)	6	722	1.16431	890.42
100%	Picard (GS)	5	729	1.16137	1106.81
	Picard (JAC)	9	859	1.16137	1106.90
	Anderson-2 (GS)	5	736	1.16137	1106.72
	Anderson-2 (JAC)	8	790	1.16137	1107.02
125%	Picard (GS)	5	719	1.15855	1131.28
	Picard (JAC)	16	1168	1.15855	1131.37
	Anderson-2 (GS)	6	786	1.15855	1131.29
	Anderson-2 (JAC)	7	738	1.15854	1131.31

Table 6.3: Average application solve time(s) for Tiamat 3x3 tests at 100% power

Method	Bison	CTF	MPACT
Picard (Gauss-Seidel)	44.6	3.9	8.3
Picard (Jacobi)	44.3	3.1	8.9
Anderson-2 (Gauss-Seidel)	47.8	2.9	8.1
Anderson-2 (Jacobi)	42.5	3.6	9.2

Sensitivity to Power Variation

Table 6.2 presents results from solving the fully-coupled HFP problem by Picard iteration and Anderson acceleration with the Gauss-Seidel and Jacobi fixed-point maps at various power levels. As expected, we observe that the iteration counts are in general lower in this table due to the combination of improved initial iterate and looser power convergence tolerance. Picard with the block Gauss-Seidel scheme actually generally performs the best for this problem, though the performance of Anderson acceleration with this map is general comparable. For problems like this in which Picard iteration converges very quickly, say 5 iterations or less, it will be very difficult for Anderson to improve upon this, as the iterations are the same for the first two steps. This table displays some similar behavior to what was observed in Table 6.1. We note that, like in the single-rod tests, methods utilizing the block Gauss-Seldel map displayed generally good robustness to the increased strength of coupling between the applications introduced by increasing the power level. For both Picard and Anderson, there is only an increase by a single coupled iteration from the 25% power case to the 125% power case. Anderson with the Jacobi map performs similarly, as the difference in coupled iterations between the best cases, 25–75% power, and the worst case, 100% power, is only 2 iterations, and there is even an improvement as the power level is increased from 100% to 125%. As in the previous section, Picard with the Jacobi map performs more poorly as the power level is increased, and the iterations to convergence more than doubles when going from the lowest power considered to the highest.

Lastly, with respect to timings, we note that when utilizing the Gauss-Seidel map the solve times are generally lower than those obtained when utilizing the Jacobi map, despite the iteration counts for the Jacobi map generally not being significantly higher. This is again explained by the a poor balance in the application solve times, which is given in Table 6.3. We again observe that Bison solves are dominant cost in the per-iteration time for the Gauss-Seidel map, so again, the improvement in time per iteration for the block Jacobi scheme over the block Gauss-Seidel scheme is fairly small. This stresses the importance of good balance in the application solve times for block Jacobi, as we observe several cases in which block Jacobi only requires in 1 iteration more than block Gauss-Seidel but still has a longer solve time.

Agreement Between Picard and Anderson Solutions

In Table 6.2, we also list the dominant eigenvalue k_{eff} and the maximum fuel temperature $T_{f,max}$ as a measure of the level of agreement between the solutions computed by each of the methods. We note that these results indicate very good agreement between each solution method and choice of fixed-point map. Within a given power level, the eigenvalues computed by all the methods differ by at most 1 pcm. Similarly, the maximum fuel temperatures for each method differ by at most a half degree Kelvin. Even though these tests are being converged

Table 6.4: Iterations to convergence for Tiamat 3x3 tests at various damping level, 100% power

	Damping factor				
Method	0.2	0.4	0.6	0.8	1.0
Picard (Gauss-Seidel)	13	6	4	13	DNC
Picard (Jacobi)	13	11	14	DNC	DNC
Anderson-2 (Gauss-Seidel)	8	6	5	5	6
Anderson-2 (Jacobi)	10	9	7	6	6

to a looser power tolerance, these results actually seem to display better agreement with each other than what was observed in the single-rod tests.

Sensitivity to Damping Level

Table 6.4 shows the number of iterations required to achieve convergence for Picard and Anderson with both the Gauss-Seidel and Jacobi maps at 100% power, with various choices of damping factor. Again, the damping factor is a power damping in all cases except Anderson with the Gauss-Seidel map, where the mixing parameter is utilized. These results are fairly similar to what we observe for the single-rod tests. For Picard iteration, both block Gauss-Seidel and block Jacobi perform fairly well for damping factors in the range 0.4–0.6. As the damping factor moves away from this range, iteration counts begin to rise rapidly. As expected, letting the damping factor be too large leads to convergence failure. This table also seems to indicate that the Jacobi map is more sensitive to the damping factor than Picard, as its performance degrades more rapidly as the damping factor is increased.

Also similar to the single-rod tests, we observe that the Anderson iterations display more robustness with respect to variation in damping than Picard. For the smallest damping factor, iteration counts rise, but not dramatically, and otherwise the performance is fairly steady over a wide range of damping parameters. Over this range Anderson is performing about on par with (as in the case of the Gauss-Seidel map) or significantly better than (like the Jacobi map) Picard iteration performing at its optimal level. At its best, Picard with the Gauss-Seidel map converges one iteration faster than Anderson, but again for problems in which Picard converges so quickly, we can not generally expect Anderson to do better, since the iterations are the same for the first and second steps. For the Jacobi map, Anderson acceleration can reduce the iteration count from the best Picard case by approximately 40%, and this level of performance is observed over the range of damping factors 0.6–1.0. The improvement is lessened a bit as the damping factor is decreased below this level. Still, every case considered for Anderson converged faster than the best case for Picard.

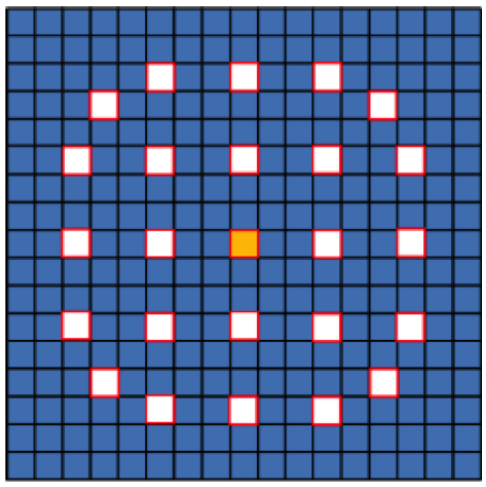


Figure 6.17: 17x17 assembly lattice with 264 UO_2 fuel rods (in blue), 24 guide tubes (in white), and 1 instrument tube (in orange)

6.4.3 17x17 Assembly Tests

We lastly consider tests consisting of a full 3D simulation of a single 17x17 fuel assembly in order to verify that the encouraging observations for Anderson acceleration from the previous two sections carry over to a problem of more significant size. This assembly contains 264 UO_2 fuel rods, 24 guide tubes, and a single central instrument tube. The layout of the assembly is illustrated in Figure 6.17, and problem inputs are specified in Appendix C.1. This test corresponds to the CASL progression problem P6a [21]. Unlike the first two problems, which are unit-test sized problems, this problem is considered large enough to be of significant interest. The following simulations were performed utilizing 64 processors: 32 allocated to MPACT, 31 to Bison, and 1 to CTF. Note that we are only able to allocate a single processor to CTF for this problem, as this code is currently only parallelized to the assembly-level [33].

Comparison of Anderson and Picard

In Table 6.5, we see results from solving the fully-coupled problem at HFP by Picard iteration and Anderson-2 with both the block Gauss-Seidel and block Jacobi fixed-point maps. We note that for both fixed-point maps, Anderson provides a modest improvement over Picard iteration in terms of iteration counts. For both maps, Anderson reduces the iterations to convergence by 25–30% from Picard, and based on results from the previous two sections, we expect that the damping factor of 0.5 that is utilized for these iterations should be near-optimal for the Picard iterations. Additionally, as expected, for both Picard and Anderson the Jacobi map requires significantly more coupled iterations to converge than Gauss-Seidel. In both cases, the Jacobi

Table 6.5: 17x17 assembly Tiamat test results, damping factor 0.5

	Iterations	Time(s)	k_{eff}	$T_{f,max}$
Picard (GS)	8	3935	1.16522	1134.76
Anderson-2 (GS)	6	3055	1.16522	1134.75
Picard (JAC)	17	3515	1.16522	1134.78
Anderson-2 (JAC)	12	2762	1.16522	1134.78

Table 6.6: Average application solve time(s) for Tiamat single-assembly tests

	Bison	CTF	MPACT	Time/Iteration
Picard (GS)	145	131	180	492
Anderson-2 (GS)	155	126	193	509
Picard (JAC)	148	127	172	207
Anderson-2 (JAC)	149	137	190	230

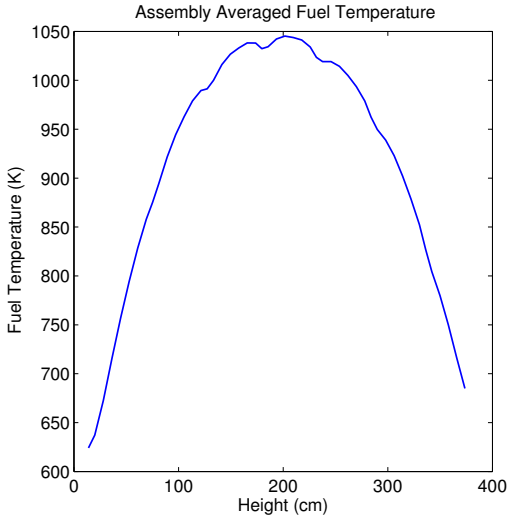
iterations require roughly twice as many iterations to converge as the Gauss-Seidel iterations.

With respect to timings, we observe that the Anderson iterations also converge in the least wall time. In fact, Anderson with the Jacobi map actually performs the best, despite requiring twice as many coupled iterations as Anderson with the Gauss-Seidel map. This can be explained by the application solve time breakdown presented in Table 6.6. In this, we note that MPACT generally requires slightly more time than the other applications, but overall the average solve times for the three applications are very well balanced. As a result of this, the block Jacobi schemes feature very efficient parallel utilization. Given the timings reported in this table, a block Jacobi iteration will require approximately 40%, on average, of the time required for a block Gauss-Seidel iteration. Because of this significant decrease in time per iteration, block Jacobi is able to require appreciably more iterations and still converge in the least time.

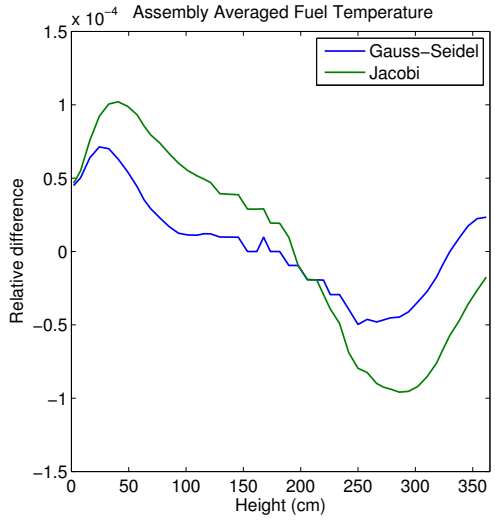
Agreement Between Anderson and Picard Solutions

As in the previous two sections, we wish to verify that the solutions resulting from Anderson acceleration agree well with the Picard solutions. To check this, we refer again to Table 6.5. In this table, we see that each Picard and Anderson iteration resulted in a dominant eigenvalue that agrees to at least 5 decimal places, and the maximum fuel temperature resulting from each of the simulations also agrees very well, differing by at most three-hundredths of a degree Celsius.

To further test the level of agreement, we consider Figures 6.18, 6.19, 6.20, and 6.21, which compare assembly averaged fuel temperature, clad temperature, fission rate, and heat flux resulting from Anderson acceleration with both the Gauss-Seidel and Jacobi fixed-point maps against the Gauss-Seidel Picard solutions, which we consider a reference solution. These figures

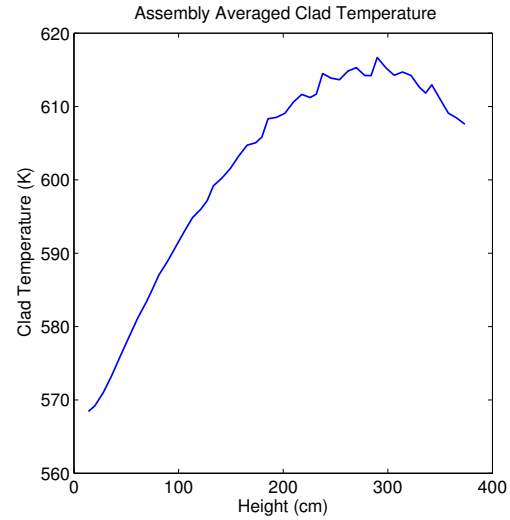


(a) Assembly averaged fuel temperature computed by Picard iteration

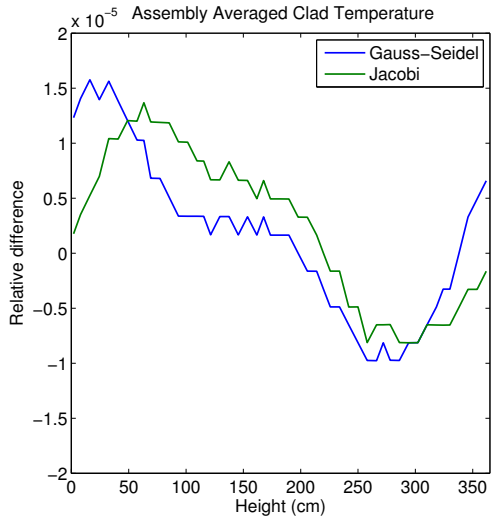


(b) Relative difference between Anderson acceleration and Picard solutions

Figure 6.18: Assembly averaged fuel temperature computed by Picard iteration with Gauss-Seidel map, and relative difference between this curve and Anderson solutions

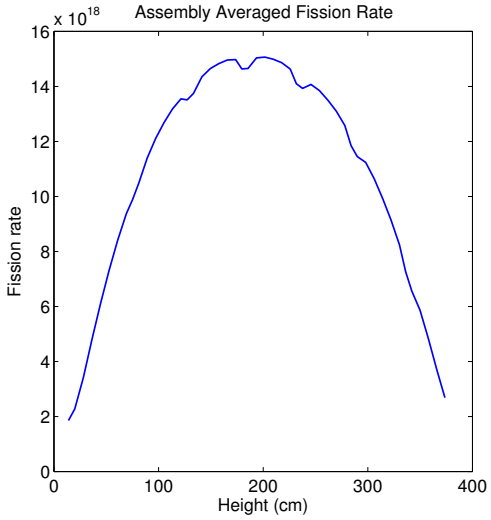


(a) Assembly averaged clad temperature computed by Picard iteration

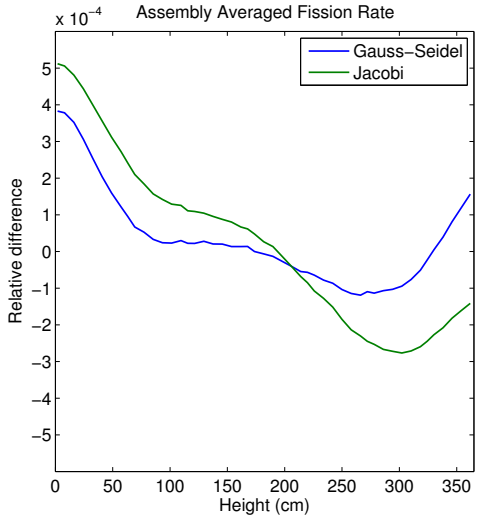


(b) Relative difference between Anderson acceleration and Picard solutions

Figure 6.19: Assembly averaged clad temperature computed by Picard iteration with Gauss-Seidel map, and relative difference between this curve and Anderson solutions

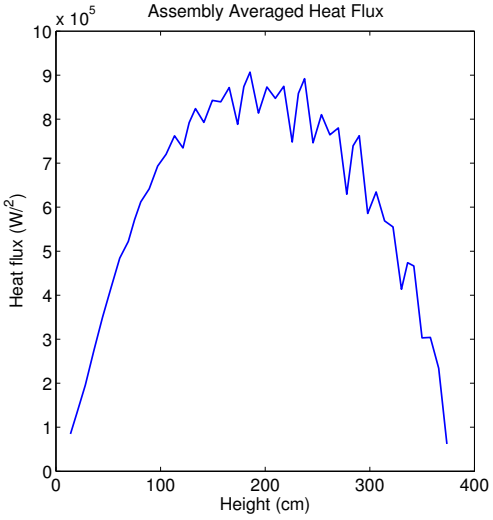


(a) Assembly averaged fission rate computed by Picard iteration

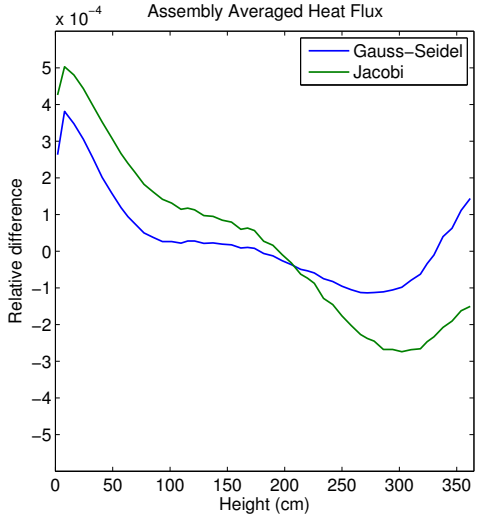


(b) Relative difference between Anderson acceleration and Picard solutions

Figure 6.20: Assembly averaged fission rate computed by Picard iteration with Gauss-Seidel map, and relative difference between this curve and Anderson solutions



(a) Assembly averaged heat flux computed by Picard iteration



(b) Relative difference between Anderson acceleration and Picard solutions

Figure 6.21: Assembly averaged heat flux computed by Picard iteration with Gauss-Seidel map, and relative difference between this curve and Anderson solutions

Table 6.7: Anderson-2 with Gauss-Seidel map with varying mixing parameter

Mixing Parameter	Iteration Count	Time(s)	k_{eff}	$T_{f,max}$
0.25	11	4995	1.16522	1134.79
0.5	6	3055	1.16522	1134.75
0.75	8	4025	1.16522	1134.78
1.0	7	3870	1.16522	1134.79

further indicate that the solutions obtained by the Anderson iterations agree very well with the Picard solution. The differences between these solutions seem explainable by the global convergence tolerances, as these tests utilize a power convergence tolerance (which is usually the final criteria to be satisfied) of $\epsilon_q = 1.0^{-4}$, and the relative differences in these figures are on this order of magnitude. Similar to the single-rod tests, in each figure both of the relative difference curves for the Anderson solutions have roughly the same shape. As this is observed in both the single-rod tests and these tests, this may may lend support to the Anderson solutions having higher accuracy than the Picard solution.

Sensitivity of Anderson to the Mixing Parameter

In Figure 6.7, we present results from Anderson-2 tests using the Gauss-Seidel map at various mixing parameters. In this, we note similar results to what was observed in the previous sections, in that the performance is generally fairly insensitive to the choice of mixing parameter. For the smallest choice of mixing parameter, there is a modest increase in the iteration count. However, for mixing parameters between 0.5–1.0 the performance of Anderson acceleration is rather consistent. Note that for the worst case in this range, mixing parameter 0.75, the number of iterations to convergence is the same as the number required for Gauss-Seidel Picard iteration in Table 6.5, and we assume that this represents Picard at or near its best. Hence, Anderson is again seen to be robust with respect to numerical damping, and it performs over a wide range of damping levels comparably to or better than Picard at its best.

Varying Cross Section Libraries

For all tests prior to this point, we utilized an 8-group test cross section library for the multi-group approximation for MPACT described in Section 2.2.3. Again, this refers to the discretization of the energy variable for the transport equation. The 8-group library is a rather coarse energy structure. In order to examine the performance of Anderson acceleration with more accurate solutions, we lastly consider tests which utilize higher fidelity cross sections. In Table 6.8, we present results comparing Picard and Anderson-2 with both the Gauss-Seidel and Jacobi maps using the high-fidelity 47-group cross section library. This table is analogous to Table 6.5,

only substituting the 47-group library for the 8-group library. We first note that changing the cross section library results in a significant change in the solution, as the eigenvalue changes by about 50 pcm, and the maximum fuel temperature increases by approximately 30 degrees Celsius. This however does not significantly affect the convergence behavior, though, as Anderson with the Gauss-Seidel map is the only case in which the iteration count changes, and increases by only a single iteration. Despite the increase for Anderson, it still converges faster than Picard, and again based on results from the previous sections, this should be a near-optimal damping factor for Picard.

We lastly note the rather significant increase in run times when utilizing the higher fidelity cross section library. Despite converging approximately in the same number of iterations in all cases, the 47-group cross section tests take 2–3 times as long to converge as the 8-group tests for the Gauss-Seidel map, and 4–5 times as long for the Jacobi map. This is explained by Table 6.9, which breaks down the average solve times for each application. When switching to the 47-group cross sections, the MPACT solves take approximately 5 times as long. As a result, with 47-group cross sections the MPACT solve takes, on average, roughly 75% of the time per iteration Gauss-Seidel iteration. Hence, there is fairly minor improvement in per-iteration run time from solving the applications simultaneously in the Jacobi map, while this still leads to a large increase in iterations to convergence. As noted above, in order for the block Jacobi scheme to be effective, the application solve times need to be well balanced, and in this case this would require a significant increase in the number of processors allocated to MPACT. However, balancing the application solve times for this coupling in general will be difficult, and this is actually due to CTF. This is because both Bison and MPACT parallelize well, but CTF is very coarsely parallelized. Ideally, Bison would be run with at least one processor allocated per rod, and MPACT can be run on thousands of processors [32], so the average solve time for these applications could be reduced significantly. However, this will leave CTF as a bottleneck, since it can only utilize a single processor per fuel assembly. As a result of this likely difficulty in load balancing, Gauss-Seidel should generally be the best option for most Tiamat simulations.

Table 6.8: 17x17 assembly Tiamat test results with 47-group cross section libraries and damping factor 0.5

	Iteration Count	Time(s)	k_{eff}	$T_{f,max}$
Picard (GS)	8	10166	1.16468	1161.27
Anderson-2 (GS)	7	9384	1.16468	1161.23
Picard (JAC)	17	16609	1.16468	1161.24
Anderson-2 (JAC)	12	12637	1.16468	1161.23

Table 6.9: Average application solve times for single-assembly Tiamat tests with 47-group cross section libraries

	Bison	CTF	MPACT	Time/Iteration
Picard (GS)	147	139	961	1271
Anderson-2 (GS)	166	118	1042	1341
Picard (JAC)	150	129	948	977
Anderson-2 (JAC)	153	135	1022	1053

Chapter 7

Conclusion

In this work, we were concerned with several aspects of the acceleration method for fixed-point iteration Anderson acceleration. First, we sought to expand the theoretical understanding of this method, as its theoretical foundation was rather sparse prior to this work. Focusing particularly on contractive fixed-point problems, we have proved several new convergence results which significantly expanded upon the theory for this method. Additionally, we have developed an implementation of Anderson acceleration that is compatible with distributed-memory vector types using MPI parallelism, which has been included in the Trilinos nonlinear solver package NOX. Lastly, we looked to evaluate this method in the context of coupled multiphysics problems in nuclear reactor simulation, specifically focusing on problems for which Picard iteration has been the primary solution method due to software restrictions. While useful for its simplicity and flexibility, Picard iteration features several drawbacks, namely relatively slow convergence and poor robustness, and we sought to evaluate the potential of Anderson acceleration to improve upon these issues. For this purpose, we integrated the NOX Anderson acceleration implementation into the Tiamat code coupling, which couples the Bison fuel performance, CTF thermal hydraulics, and MPACT neutronics codes in order to provide a tool for pellet-cladding interaction analysis, and performed an in-depth comparison between the performance of Picard iteration and Anderson acceleration for solving this fully-coupled problem. We will now overview the major conclusions that we have drawn in there areas, and additionally discuss some limitations of this work and areas for further development.

7.1 Anderson Acceleration Theory

Prior to this work, the bulk of the analysis for Anderson acceleration was in regard to showing its relation to other methods (quasi-Newton methods and multi-secant updating in [19], GMRES in [62]). Again, these results are interesting to note, but with the exception of GMRES with

full storage utilization, they provide little in the way of convergence theory. For our analysis, we sought to show more fundamental convergence results, especially for the limited-memory variant of this method and for nonlinear fixed-point problems. As this method is intended to accelerate fixed-point iteration, we focused on problems for which fixed-point iteration is convergent (locally for nonlinear fixed-point maps). We considered the case where the fixed-point map is contractive in a neighborhood of the solution. That is, for linear problems we assume that $G(u) = Au + b$ with $\|A\| = c < 1$, and for nonlinear problems we assume that there exist $\hat{\rho} > 0$ and $c \in [0, 1)$ such that for $u, v \in \mathcal{B}_{\hat{\rho}}(u^*)$, $\|G(u) - G(v)\| \leq c\|u - v\|$. Some of the significant results which we have shown include:

- For the linear fixed-point problem $u = G(u) \equiv Au + b$, the Anderson-m iterates $\{u_k\}$ with any storage depth parameter m converge to the solution u^* r-linearly with rate of convergence no worse than the contractive constant c, and the fixed-point residuals $\{F(u_k)\}$ converge to 0 q-linearly also with rate no worse than c.

- For nonlinear fixed-point problems, for any storage-depth parameter m the Anderson-m iterates $\{u_k\}$ converge to the solution u^* and the fixed-point residuals $\{F(u_k)\}$ converge to 0, both r-linearly with rate of convergence $\hat{c}$ which can be made arbitrarily close to c given good enough initial iterate. This result requires the assumption that there exists some constant $M_\alpha \geq 1$ such that $\sum_{i=0}^{m_k} |\alpha_i^{(k)}| \leq M_\alpha$ for all $k \geq 1$.

- For a variation of the Anderson acceleration algorithm which adjusts the storage depth to maintain good conditioning of the least-squares problem, the iterates $\{u_k\}$ converge r-linearly to the solution u^* and the fixed-point residuals $\{F(u_k)\}$ converge to 0 q-linearly, both with rate of convergence $\hat{c}$ which can be made arbitrarily close to c given good enough initial iterate. Unlike the previous result, this requires no assumption on the size of the linear combination coefficients. Additionally, as the convergence in the residuals is q-linear, the convergence for this method will be more readily apparent and the rate of convergence can be more easily estimated.

- When applying Anderson acceleration with a fixed-point map which can only be evaluated approximately (i.e. we evaluate $\hat{G}(u) = G(u) + \epsilon(u)$ where $\epsilon(u)$ is some error term), rather than convergence we obtain local improvement results. The bounds which are obtained ((3.84) and (3.85)) predict linear reduction in the error and fixed-point residual up to some stagnation point, and the stagnation point is proportional to the size of the error in the fixed-point map evaluation.

The first three results listed here essentially state that Anderson acceleration is in a sense is no worse than Picard iteration. Under these conditions, for both linear and nonlinear fixed-point

problems Picard iteration converges q-linearly in both the residual and error with rate of convergence c. For linear problems, we see that the rate of convergence is the same for both Anderson and Picard, and for nonlinear problems, the rate of convergence for Anderson can be made arbitrarily close to that of Picard iteration. However, in practice it is generally observed that Anderson acceleration provides an appreciable improvement in the rate of convergence over Picard. The reason that these results do not say that Anderson will definitively converge more rapidly than Picard is the fact that for each result we assume no improvement from the optimization step in Anderson acceleration. In each proof, we obtain the quantity $\| \sum_{i=0}^{m_k} \alpha_i^{(k)} F(u_{k-m_k+i})\|$, where the coefficients $\{\alpha_i^{(k)}\}$ are chosen to minimize $\|\sum_{i=0}^{m_k} \alpha_i F(u_{k-m_k+i})\|$ such that $\sum_{i=0}^{m_k} \alpha_i = 1$. We make the bound $\| \sum_{i=0}^{m_k} \alpha_i^{(k)} F(u_{k-m_k+i})\| \leq \|F(u_k)\|$, but it is possible that this bound significantly neglects the optimality of the linear combination coefficients. If we could, say, guarantee that $\| \sum_{i=0}^{m_k} \alpha_i^{(k)} F(u_{k-m_k+i})\| \leq \eta \|F(u_k)\|$ for some $\eta \in (0,1)$ for all $k \geq 1$, then we could definitively say that Anderson will converge more rapidly than fixed-point iteration. Going forward, it would be interesting to identify conditions, or possibly a class of problems, for which it could be said that Anderson acceleration will converge more rapidly than Picard iteration.

The final result described above shows that we also obtain a similar result for both Picard and Anderson in the context of an inaccurate fixed-point map evaluation. For both, we expect stagnation in the iteration error and fixed-point residual at some level proportional to the error in the function evaluation. Our result however suggests that Anderson may react more poorly than Picard if the size of the error in the fixed-point map evaluation is large, as the bound on the allowable size of the evaluation error for Anderson may be significantly smaller than that for Picard. Additionally, Anderson may behave more poorly than Picard if the error-free fixed-point map is weakly contractive. This is due to the fact that for Anderson the bound on the iteration error is derived from a bound on the exact fixed-point residual, whereas for Picard the iteration error bound is computed directly. If the fixed-point map is weakly contractive, the fixed-point residual may poorly reflect the size of the iteration error, and thus deriving a bound from the residual does little good.

While the results described above significantly expand upon the theoretical foundation for Anderson acceleration, there several other issues not addressed in these results for which practice could benefit from further theory development. First, these results provide little insight with regard to selection of the storage depth parameter m. The rate of convergence which we show does not depend in any way on this parameter. The only location in which this parameter comes into play in any of our analysis is in the definition of "good enough" for the initial iterate, and that we might expect the coefficient bound M_α to be larger for larger storage depth parameters. Ideally, there would be some theoretical justification for selecting a storage depth parameter, rather than simply determining a good parameter for a problem experimentally. Along these lines, in a similar manner to adjusting for conditioning there may be better methods

for dynamically selecting the storage depth at a given iteration in order to obtain the maximum residual reduction. One additional area where there is room for further theory development is in globalization of this method. The NOX implementation of Anderson acceleration includes an option for line searches, and the delayed start option seems like it can provide some measure of globalization by providing Anderson acceleration an improved initial iterate. However, the use of these options is currently not supported or guided by theory. Theoretical developments in these areas could potentially lead to improvements in the NOX Anderson acceleration solver, resulting in improved robustness and performance.

7.2 Coupled Multiphysics Problems

The remainder of this work concerned the solution of coupled multiphysics problems, specifically in the context of LWR simulation. In particular, we were concerned with problems for which we assume no more functionality from the application codes for solving individual sets of physics than the ability to solve their individual problems and returning some response functions. As Newton-like methods require at the very least the ability to compute and access a residual (to apply JFNK), Picard has to this point been the primary method for solving this sort of problem. In this study we explored the potential of Anderson acceleration to improve upon some of the drawbacks of Picard iteration. We tested this specifically using the Tiamat code coupling. Again, this couples together the Bison fuel performance, CTF thermal hydraulics, and MPACT neutronics codes. Bison solves its set of physics by JFNK, so it internally computes a residual, but the other applications are black boxes which can only accept coupling data, solve their sets of physics, and return some response functions. As a result, a straightforward application of a Newton-like method is not possible, so this coupling had been previously implemented using Picard iteration.

To integrate Anderson acceleration into Tiamat, we had to explicitly define the vector of unknowns u and the fixed-point map $G(u)$ for the fixed-point problem $u = G(u)$ which we apply Anderson acceleration to solve. These quantities are implicit in Picard iteration, but the these must be explicitly defined for Anderson acceleration in order to form and solve the least-squares problem and compute a new iterate. This involved reformulating the fully-coupled problem in the form of a fixed-point problem in terms of coupling data being passed between application codes. The problem needed to be formulated in this manner as the state variables for each of the application codes are not accessible, but the coupling data must necessarily be accessible in order to implement the Picard coupling. This formulation allowed us to implement Anderson acceleration while utilizing much of the existing infrastructure with minimal change. As the coupling data comprising the vector of unknowns and the fixed-point map represent various physical quantities, we applied Anderson to a scaled fixed-point problem to give the

various fields more equal weight in the least-squares problem solved in the Anderson acceleration algorithm. This scaling essentially represents a left preconditioning of the fixed-point residual.

To compare the performance of Anderson acceleration and Picard iteration, we considered various parameter studies. These studies include a comparison the two with respect to some of the issues which were noted to be problematic for Picard, namely robustness with respect to variation in damping and power levels. In general, we noted that Anderson provides a significant improvement in robustness. With Picard, the performance of the method is strongly dependent on selection of the damping level, but with Anderson this does not seem to be the case. Whereas Picard displays an optimal level of damping with performance degrading away from this optimal level, Anderson gave consistently good performance. At worst, Anderson performed comparably to optimally-damped Picard iteration, but it often provided a significant improvement. This sort of behavior was consistently displayed as the power level was varied. As we noted previously, the optimal damping level for Picard shifts depending on the power level. Thus, when solving a problem by Picard with some given damping factor, it can not be known without additional simulations whether the performance could be improved by changing the damping level. Conversely, since the performance of Anderson acceleration was consistently insensitive to the damping level, we can more comfortably assume that Anderson acceleration is performing near its optimal level.

Another aspect of this work was a comparison of ways to reformulate the fully-coupled problem in Tiamat as different fixed-point problems. We formulated the fully-coupled problem in the form of three fixed-point maps: a block Gauss-Seidel map which solves the applications sequentially, a block Jacobi map which solves all applications simultaneously, and an intermediate map which alternates between simultaneously solving Bison and MPACT and solving CTF. The advantage of the second two maps is a lower time per evaluation of the fixed-point map due to improved utilization of parallel resources. We observe that iteration counts for the intermediate map were generally fairly close to the Jacobi map while taking more time per iteration, so it seems that this formulation will rarely perform best. For Picard iteration, the increase in iteration counts for the Jacobi map over Gauss-Seidel makes it uncompetitive despite the reduced time per iteration, but for Anderson, the iterations to convergence for these maps can be made close enough so that Jacobi may give the best run time. However, for a block Jacobi scheme to be effective, care must be taken so that the solve times for the application codes are nearly the same. Balancing the solve times for the application codes in this specific coupling will be problematic due to the coarse parallelism of CTF. Bison and MPACT can be run with hundreds to thousands of processors, while CTF can currently only utilize a single processor per fuel assembly, and it will thus be a bottleneck. As a result of this, unless CTF is further developed to accommodate finer grained parallelism, the block Gauss-Seidel scheme will generally be the best option for Tiamat simulations.

We also considered a comparison of Anderson acceleration and JFNK for solving the fully-coupled problem in Tiamat. Anderson acceleration requires the formation and computation of a fixed-point residual, and one can attempt to solve this fixed-point residual equation by JFNK. We observe that JFNK performs significantly worse than Anderson for solving this fixed-point problem. The error introduced by the approximate solves in the evaluation of the fixed-point map can be problematic for JFNK, and when JFNK does perform well, too much work is required each nonlinear iteration to be competitive with Anderson acceleration. Each fixed-point map evaluation requires a full solve of each application code, and JFNK requires an evaluation of the fixed-point map each linear and nonlinear iteration. In the tests we considered, this resulted in more fixed-point map evaluations for 1 or 2 JFNK iterations than Anderson requires to fully solve the coupled problem. We will note however that in cases in which the application codes can compute and return residuals, or possibly even derivative information, Newton-like methods may very well be the better option. For instance, the study in [24] considers a coupling between neutronics and thermal hydraulics codes which can compute residuals, and by utilizing an approximate residual evaluation in linear solves, very good performance is obtained for JFNK. However, in our case where the residual for JFNK is a fixed-point residual involving internal application code solves, JFNK is unlikely to be competitive with Anderson.

Going forward, the techniques which have been utilized in this work could abstracted into a more general framework for integrating Anderson acceleration into code couplings which have used Picard iteration as the primary solution method. Returning to the general problem formulation described in Section 1.1.2, and suppressing the independent parameters $\{p_{m,n}\}$ from notation, consider the problem given by N_f coupled sets of physics. Suppose that the functionality of the codes corresponding to these single-physics systems is limited to solving their individual sets of physics and returning at least the responses necessary to evaluate the transfer functions. As we do not assume the ability to access state variables or compute single-physics residuals or derivative information, it is not possible to implement Newton's method in the manner described in Section 1.2.2. Picard iteration can be implemented by imposing some sequence of single-physics solves and transfers of updated coupling data. While the single-physics state variables may or may not be accessible, the data being passed between codes must be accessible in order to implement the Picard coupling. Then, as we did for the Tiamat coupling, we note that the residual equation $f_i(x_i, \{z_{i,j}\}) = 0$ implicitly defines the solution x_i as a function of the coupling parameters $\{z_{i,j}\}$. We then denote the solution to residual equation i given the coupling parameters $\{z_{i,j}\}$ as $x_i(\{z_{i,j}\})$. We thus reformulate the system

using the transfer functions to define the constraints, and write the fully-coupled system as:

$$F(\{z_{m,n}\}) = \begin{pmatrix} r_{0,0}(\{x_i(\{z_{i,j}\})\}) - z_{0,0} \\ \vdots \\ r_{0,N_{z_0}-1}(\{x_i(\{z_{i,j}\})\}) - z_{0,N_{z_0}-1} \\ \vdots \\ r_{k,0}(\{x_i(\{z_{i,j}\})\}) - z_{k,0} \\ \vdots \\ r_{k,N_{z_k}-1}(\{x_i(\{z_{i,j}\})\}) - z_{k,N_{z_k}-1} \\ \vdots \end{pmatrix} = 0, \tag{7.1}$$

where $\{z_{m,n}\}$ is the collection of all coupling parameter vectors. As this residual is evaluated by first solving of each set of physics given some input set of coupling parameter vectors and then evaluating transfer functions given these solutions, this corresponds to the fixed-point residual for a block Jacobi scheme. Other fixed-point maps can be obtained by employing a nonlinear elimination scheme on this system. The promising results obtained for the Tiamat coupling suggest that it may be worthwhile to investigate integrating Anderson acceleration in this manner into other code couplings which use primarily Picard iteration, as it may result in a significant improvement in terms of performance and robustness with respect to parameter variation.

REFERENCES

[1] SCALE: A comprehensive modeling and simulation suite for nuclear safety analysis and design. ORNL/TM-2005/39, Version 6.1, Oak Ridge National Laboratory, Oak Ridge, TN, 2011.

[2] D.G. Anderson. Iterative procedures for nonlinear integral equations. *Journal of the ACM*, 12(4):547–560, 1965.

[3] R.A. Bartlett. Teuchos::RCP beginner's guide: An introduction to the Trilinos smart reference-counted pointer class for (almost) automatic dynamic memory management in C++. Technical Report SAND2004-3268, Sandia National Laboratories, 2010.

[4] R.A Bartlett. TriBITS developers guide and reference. Technical Report L3:PHI.INF.P8.03, Consortium for Advanced Simulation of LWRs, 2014.

[5] T. Bergman, A. Lavine, F. Incropera, and D. Dewitt. *Fundamentals of Heat and Mass Transfer*. Wiley, 7th edition, 2011.

[6] A. Björk and C.C. Paige. Loss and recapture of orthogonality in the modified Gram-Schmidt algorithm. *SIAM Journal on Matrix Analysis and Applications*, 13(1):176–190, 1992.

[7] S.C. Brenner and L.R. Scott. *The Mathematical Theory of Finite Element Methods*. Springer, New York, 3rd edition, 2008.

[8] C. Brezinski. Convergence acceleration during the 20th century. *Journal of Computational and Applied Mathematics*, 122:1–21, 2000.

[9] M.T. Calef, E.D. Fichtl, J.S. Warsa, M. Berndt, and N.N. Carlson. Nonlinear Krylov acceleration applied to a discrete ordinates formulation of the k-eigenvalue problem. *Journal of Computational Physics*, 238:188–209, April 2013.

[10] N. Capps, B.D. Wirth, R. Montgomery, D. Sunderland, and M. Pytel. Evaluation of missing pellet surface geometry on cladding stress distribution and magnitude. Technical Report L3:FMC.FUELS.P11.01, Consortium for Advanced Simulation of LWRs, 2015.

[11] S. Chandrasekhar. *Radiative Transfer*. Dover Publications, New York, 1960.

[12] K.T. Clarno, R.P. Pawlowski, R.O. Montgomery, T.M. Evans, and B..S. Collins. High fidelity modeling of pellet-clad interaction using the CASL Virtual Environment for Reactor Applications. In *Joint International Conference on Mathematics and Computation, Supercomputing in Nuclear Applications and the Monte Carlo Method*, pages 1–15, 2015.

[13] T. Coleman and Y. Li. A reflective Newton method for minimizing a quadratic function subject to bounds on some of the variables. *SIAM Journal on Optimization*, 6(4):1040–1058, 1996.

[14] B. Collins, T. Downar, et al. MPACT theory manual. Technical Report CASL-U-2015-0078-000, Consortium for Advanced Simulation of LWRs, 2015.

[15] D.E. Cullen. Application of the probability table method to multigroup calculations of neutron transport. *Nuclear Science and Engineering*, 55(4):387–400, 1974.

[16] J. Degroote, K.-J. Bathe, and J. Vierendeels. Performance of a new partitioned procedure versus a monolithic procedure in fluid-structure interaction. *Computers & Structures*, 87:793–801, June 2009.

[17] S.C. Eisenstat and H.F. Walker. Globally convergent inexact Newton methods. *SIAM Journal on Optimization*, 4(2):393–422, 1994.

[18] M.M. El-Wakil. *Nuclear Heat Transport*. American Nuclear Society, La Grange Park, IL, 1993.

[19] H.-R. Fang and Y. Saad. Two classes of multisecant methods for nonlinear acceleration. *Numerical Linear Algebra with Applications*, 16(3):197–221, 2009.

[20] D. Gaston, C. Newman, and G. Hansen. MOOSE: A parallel computational framework for coupled systems of nonlinear equations. *Nuclear Engineering and Design*, 239(10):1768–1778, October 2009.

[21] A.T. Godfrey. VERA core physics benchmark progression problem specifications. Technical Report CASL-U-2012-0131-004, Revision 4, Consortium for Advanced Simulation of LWRs, 2014.

[22] G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, 3rd edition, 1996.

[23] J.D. Hales, S.R. Novascone, G. Pastore, D.M. Perez, B.W. Spencer, and R.L. Williamson. BISON theory manual. Technical Report October, Idaho National Laboratory, 2013.

[24] S. Hamilton, M. Berrill, K. Clarno, R. Pawlowski, A. Toth, C.T. Kelley, T. Evans, and B. Philip. An assessment of coupling algorithms for nuclear reactor core physics simulations. *Journal of Computational Physics*, 311:241–257, 2016.

[25] S. Hamilton et al. Multiphysics simulations for LWR analysis. In *International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering*, Sun Valley, ID, 2013.

[26] R.J. Harrison. Krylov subspace accelerated inexact Newton method for linear and nonlinear equations. *Journal of Computational Chemistry*, 25(3):328–334, 2003.

[27] M. Heroux et al. An Overview of Trilinos. Technical Report SAND2003-2927, Sandia National Laboratories, 2003.

[28] B.W. Kelley and E.W Larsen. 2D/1D approximations to the 3D neutron transport equation. I: Theory. In *International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering*, 2013.

[29] C.T. Kelley. *Iterative Methods for Linear and Nonlinear Equations*. Society for Industrial and Applied Mathematics, Philadelphia, PA, January 1995. no. 16 in Frontiers in Applied Mathematics.

[30] C.T. Kelley. *Iterative Methods for Optimization*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1999. no. 18 in Frontiers in Applied Mathematics.

[31] D.A. Knoll and D.E. Keyes. Jacobian-free Newton-Krylov methods: A survey of approaches and applications. *Journal of Computational Physics*, 193(2):357–397, January 2004.

[32] B. Kochunas, D. Jabaay, B. Collins, and T. Downar. Demonstration of neutronics coupled to thermal-hydraulics for a full-core problem using COBRA-TF / MPACT. Technical Report L3:RTM.P7.05, Consortium for Advanced Simulation of LWRs, 2014.

[33] V. Kucukboyaci, Y. Sung, and B. Salko. COBRA-TF parallelization and application to PWR reactor core subchannel DNB analysis. In *Joint International Conference on Mathematics and Computation, Supercomputing in Nuclear Applications and the Monte Carlo Method*, pages 1–18, 2015.

[34] P.J. Lanzkron, D.J. Rose, and J.T. Wilkes. An analysis of approximate nonlinear elimination. *SIAM Journal on Scientific Computing*, 17(2):538–559, 1996.

[35] L. Lin and C. Yang. Elliptic preconditioner for accelerating the self consistent field iteration in Kohn-Sham density functional theory. *Siam Journal on Scientific Computing*, 35(5):277–298, June 2013.

[36] I Lux and L Koblinger. *Monte Carlo Particle Transport Methods: Neutron and Photon Calculations*. CRC Press, 1991.

[37] K. Martin and B. Hoffman. *Mastering CMake*. Kitware, Inc., 6th edition, 2013.

[38] K. Mayaram and D.O. Pederson. Coupling algorithms for mixed-level circuit and device simulation. *IEEE Transactions on Computer Aided Design*, 11(8):1003–1012, 1992.

[39] L. Monti and T. Schulenberg. Coupled ERANOS / TRACE system for HPLWR 3 pass core analyses. In *International Conference on Mathematics, Computational Methods and Reactor Physics*, pages 1–14, Saratoga Springs, NY, 2009.

[40] S. Palmtag. Demonstration of neutronics coupled to thermal-hydraulics for a full-core problem using VERA. Technical Report L2:AMA.P7.02, Consortium for Advanced Simulation of LWRs, 2013.

[41] S. Palmtag and A. Godfrey. VERA common input user manual. Technical Report CASL-U-2014-0014-002, Consortium for Advanced Simulation of LWRs, 2015.

[42] B.N. Parlett. *The Symmetric Eigenvalue Problem*. Society for Industrial and Applied Mathematics, January 1998.

[43] S.V. Patankar. *Numerical Heat Transfer and Fluid Flow*. Hemisphere Publishing, 1980.

[44] R. Pawlowski, R.A. Bartlett, N. Belcourt, R. Hooper, and R. Schmidt. A theory manual for multi-physics code coupling in LIME version 1.0. Technical Report SAND2011-2195, Sandia National Laboratories, 2011.

[45] R. Pawlowski, K. Clarno, and R. Montgomery. Demonstrate integrated VERA-CS for the PCI challenge problem. Technical Report L1:CASL.P9.03, Consortium for Advanced Simulation of LWRs, 2014.

[46] F. Potra and H. Engler. A characterization of the behavior of the Anderson acceleration on linear problems. *Linear Algebra and its Applications*, 438(3):1002–1011, February 2013.

[47] P. Pulay. Convergence acceleration of iterative sequences. The case of SCF iteration. *Chemical Physics Letters*, 73(2):393–398, 1980.

[48] P. Pulay. Improved SCF convergence acceleration. *Journal of Computational Chemistry*, 3(4):556–560, 1982.

[49] T. Rohwedder and R. Schneider. An analysis for the DIIS acceleration method used in quantum chemistry calculations. *Journal of Mathematical Chemistry*, 49(9):1889–1914, August 2011.

[50] R.K. Salko and M.N. Avramova. CTF theory manual. Technical Report CASL-U-2015-0054-000, Consortium for Advanced Simulation of LWRs, 2015.

[51] R. Sanchez and N. McCormick. A review of neutron transport approximations. *Nuclear Science and Engineering*, 80:481–535, 1982.

[52] A. Sidi, W.F. Ford, and D.A. Smith. Acceleration of convergence of vector sequences. *SIAM Journal on Numerical Analysis*, 23(1):178–196, 1986.

[53] V. Simoncini and D.B. Szyld. Flexible inner-outer Krylov subspace methods. *SIAM Journal on Numerical Analysis*, 40(6):2219–2239, 2003.

[54] V. Simoncini and D.B. Szyld. Theory of inexact Krylov subspace methods and applications to scientific computing. *SIAM Journal on Scientific Computing*, 25(2):454–477, 2003.

[55] V. Simoncini and D.B. Szyld. Recent computational developments in Krylov subspace methods for linear systems. *Numerical Linear Algebra with Applications*, 14:1–59, 2007.

[56] S.R. Slattery, P.P.H. Wilson, and R.P. Pawlowski. The Data Transfer Kit: A geometric rendezvous-based tool for multiphysics data transfers. In *International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering*, pages 1–11, 2013.

[57] S.G. Stimpson. *An azimuthal, Fourier moment-based axial SN solver for the 2D/1D scheme*. PhD thesis, University of Michigan, 2015.

[58] A. Toth and C.T. Kelley. Convergence analysis for Anderson acceleration. *SIAM Journal on Numerical Analysis*, 53(2):805–819, 2015.

[59] A. Toth, C.T. Kelley, and R. Pawlowski. Demonstrate Anderson acceleration for coupled neutronics and thermal hydraulics in Tiamat. Technical Report L3:RTM.SUP.P10.01, Consortium for Advanced Simulation of LWRs, 2015.

[60] A. Toth, C.T. Kelley, S. Slattery, S. Hamilton, K. Clarno, and R. Pawlowski. Analysis of Anderson acceleration on a simplified neutronics/thermal hydraulics system. In *Joint International Conference on Mathematics and Computation, Supercomputing in Nuclear Applications and the Monte Carlo Method*, pages 1–12, 2015.

[61] J.A. Turner. Virtual environment for reactor applications (VERA). Technical Report L2:VRI.P7.01, Consortium for Advanced Simulation of LWRs, 2013.

[62] H.F. Walker and P. Ni. Anderson acceleration for fixed-point iterations. *SIAM Journal on Numerical Analysis*, 49(4):1715–1735, 2011.

[63] J. Willert, X. Chen, and C.T. Kelley. Newton's method for Monte Carlo-based residuals. *SIAM Journal on Numerical Analysis*, 53(4):1738–1757, 2015.

[64] J. Willert, C.T. Kelley, D.A. Knoll, and H. Park. Hybrid deterministic/Monte Carlo neutronics. *Siam Journal on Scientific Computing*, 35(5):62–83, 2013.

[65] M.L. Williams and K.S. Kim. The embedded self-shielding method. In *PHYSOR 2012*, Knoxville, TN, 2012.

[66] J. Yan, B. Kochunas, M. Hursin, T. Downar, Z. Karoutas, and E. Baglietto. Coupled computational fluid dynamics and MOC neutronic simulations of Westinghouse PWR fuel assemblies with grid spacers. In *14th International Topical Meeting on Nuclear Reactor Thermalhydraulics*, Toronto, Ontario, 2011.

[67] D.P. Young, W.P. Huffman, R.G. Melvin, C.L. Hilmes, and F.T. Johnson. Nonlinear elimination in aerodynamic analysis and design optimization. *Lecture Notes in Computational Science A*, pages 17–43, 2003.

APPENDICES

Appendix A

Iterative Methods for Linear and Nonlinear Equations

Anderson acceleration, and several of the other methods that we consider in this work (e.g. fixed-point iteration and JFNK) are considered iterative methods. In this sort of method, a hopefully improving approximation to the solution is generated by a sequence of iterates $\{u_n\}$ given some initial iterate u_0. In this appendix, we provide a more in-depth description of several of the methods which were referenced in this work, and overview some theory related to these methods.

A.1 Linear Equations

We first consider iterative methods for solving linear equations. By linear equations, we refer to problems of the form

$$Au = b,$$

where we seek $u \in \mathbb{R}^n$ given $A \in \mathbb{R}^{n \times n}$ and $b \in \mathbb{R}^n$. One sort of method for solving this problem is a direct method. For instance, one may decompose A using an LU or Cholesky (for symmetric positive-definite matrices) factorization and directly compute the solution by forward/back substitution. In general, the dominant cost for a direct method is the matrix factorization. If the matrix A is dense, the computational cost of either an LU or Cholesky factorization is $O(n^3)$, though this can be less if the matrix is sparse or banded. For many realistic problems the computational cost of direct methods is too large for such methods to be useful. As a result, iterative methods are the only practical option for many problems, in particular when n is very large.

A.1.1 Preliminaries

Before describing some commonly used iterative methods for linear equations, we first need to establish some conventions and definitions.

Definition A.1. *Given some vector norm on* $\mathbb{R}^n$ $\|\cdot\|$*, we define the induced matrix norm of* $A \in \mathbb{R}^{n\times n}$ *by the following*

$$\|A\| = \max_{x\neq 0} \frac{\|Ax\|}{\|x\|} = \max_{\|x\|=1} \|Ax\|. \tag{A.1}$$

Induced matrix norms have the attractive property that $\|Ax\| \leq \|A\|\|x\|$ for all $x \in \mathbb{R}^n$. An important concept related to matrix norms is that of the condition number.

Definition A.2. *The condition number of the matrix* $A \in \mathbb{R}^{n\times n}$ *is defined as* $\kappa(A) = \|A\|\|A^{-1}\|$*. If* A *is singular, its condition number is infinite.*

The condition number of a matrix is useful for relating the residual $r = b - Au$ to the error $e = u - u^*$, where $u^* = A^{-1}b$ is the exact solution. Termination of iterative methods are generally based on the size of the residual, as the error can not be computed without already knowing the solution, so it is important to know how well the residual reflects the error. A common termination criterion is to require some level of relative reduction in the residual, i.e. $\frac{\|r_k\|}{\|r_0\|} < \tau$ where $\tau \in (0, 1)$ is a given tolerance. This quantity is related to the relative error reduction by the following Lemma.

Lemma A.1. *Let* $u_0, b \in \mathbb{R}^n$ *and nonsingular* $A \in \mathbb{R}^{n\times n}$ *be given, and let* $u^* = A^{-1}b$*. Then, the following bound holds:*

$$\frac{\|e\|}{\|e_0\|} \leq \kappa(A)\frac{\|r\|}{\|r_0\|}. \tag{A.2}$$

Proof. To prove this, note that for any u it holds that

$$r = b - Au = A(u^* - u) = -Ae.$$

Thus, we have both $r = -Ae$ and $e = -A^{-1}r$. From these, we obtain $\|e\| \leq \|A^{-1}\|\|r\|$, and $\|r\| \leq \|A\|\|e\|$, which we can rewrite as $\|e\|^{-1} \leq \|A\|\|r\|^{-1}$ (so long as $u \neq u^*$). These inequalities give

$$\frac{\|e\|}{\|e_0\|} = \|e\|\|e_0\|^{-1} \leq (\|A^{-1}\|\|r\|)(\|A\|\|r_0\|^{-1}) = \kappa(A)\frac{\|r\|}{\|r_0\|}.$$

□

One last concept that we need to introduce is eigenvalues and eigenvectors.

Definition A.3. *We say that λ is an eigenvalue of matrix A if there exists a vector $x \neq 0$ such that*

$$Ax = \lambda x,$$

and x is referred to as an eigenvector. We refer to a corresponding eigenvalue–eigenvector pair as an eigenpair.

The matrix $A \in \mathbb{R}^{n\times n}$ has n (not necessarily distinct or real) eigenvalues. They are given by the roots of the characteristic polynomial of A, $p(z) = \det(A - zI)$. We now introduce one final definition related to the eigenvalues of a matrix.

Definition A.4. *Define $\sigma(A)$ to be the set of all eigenvalues of matrix A, which is referred to as the spectrum of A. We then define the spectral radius of A as $\rho(A) = \max_{\lambda\in\sigma(A)} |\lambda|$.*

A.1.2 Stationary Iterative Methods

Among the simplest of iterative methods for solving linear equations are stationary iterative methods. These typically proceed by splitting the matrix into some convenient form $A = B - C$ with B nonsingular, and formulating the problem as solving

$$Bu = Cu + b. \tag{A.3}$$

This is then attempted to be solved by iterating

$$u_{k+1} = B^{-1}Cu_k + B^{-1}b = Mu_k + d, \tag{A.4}$$

where $M = B^{-1}C$ and $d = B^{-1}b$. In this manner, the root finding problem is reformulated as a fixed problem, which is being solved by fixed-point iteration. That is, letting $G(u) = Mu+d$, this iteration can be represented in the form $u_{k+1} = G(u_k)$. We now present a theorem describing the behavior of fixed-point iteration for linear problems.

Theorem A.1. *If $\rho(M) < 1$, then $I - M$ is nonsingular, and the iteration* (A.4) *converges to the solution $(I - M)^{-1}d$*

Proof. We begin by noting that given some $\epsilon > 0$, there exists a matrix norm such that $\|M\| \leq \rho(M) + \epsilon$ (Theorem 1.3.1 from [29]). Letting $\epsilon = \frac{1-\rho(M)}{2}$, this implies that there is some matrix norm such that $\|M\| < 1$. We now show that $\sum_{i=0}^{\infty} M^i = (I - M)^{-1}$. First, denote the partial sum $S_j = \sum_{i=0}^{j} M^i$. Now, given $j, k > 0$ with $j > k$, and using the matrix norm indicated above, we have

$$\|S_j - S_k\| = \|\sum_{i=k+1}^{j} M^i\| \leq \sum_{i=k+1}^{j} \|M\|^i = \|M\|^{k+1}\frac{1 - \|M\|^{j-k}}{1 - \|M\|} \leq \frac{\|M\|^{k+1}}{1 - \|M\|}. \tag{A.5}$$

The quantity on the right goes to 0 as $j, k \to \infty$, so $\{S_j\}$ is a Cauchy sequence in $\mathbb{R}^{n\times n}$. Since $\mathbb{R}^{n\times n}$ is complete, this sequence therefore has some limit $S \in \mathbb{R}^{n\times n}$. Now, noting that $I + MS_j = S_{j+1}$ and letting $j \to \infty$, this becomes $I + MS = S$, or alternatively $(I - M)S = I$. Hence $S = \sum_{i=0}^{\infty} M^i = (I - M)^{-1}$. Next, we note that we can write (A.4) as

$$u_{k+1} = M^{k+1}u_0 + \left(\sum_{i=0}^{k} M^i\right) d. \tag{A.6}$$

Because $\|M\| < 1$, the first term on the right goes to 0 as $k \to \infty$, and by what was shown above, $\left(\sum_{i=0}^{k} M^i\right) d \to (I - M)^{-1}d$ as $k \to \infty$. Hence, $\{u_k\} \to (I - M)^{-1}d$. □

Note that $I-M = I-B^{-1}C = B^{-1}(B-C) = B^{-1}A$. Then, because $I-M$ is nonsingular, A is nonsingular as well and we have $(I-M)^{-1} = A^{-1}B$. Hence, $(I-M)^{-1}d = A^{-1}BB^{-1}b = A^{-1}b$, which is the correct solution of the linear system.

Some frequently used methods of this type include the Jacobi, Gauss-Seidel, and Successive Over-Relaxation (SOR) methods. For these methods, the matrix A is decomposed as $A = D - L - U$, where D is the diagonal component and $-L$ and $-U$ are the strict lower and upper triangular components respectively of A. For Jacobi, the splitting is chosen to be $B = D$ and $C = L + U$, giving the following iteration

$$u_{k+1} = D^{-1}(L + U)u_k + D^{-1}b. \tag{A.7}$$

As D is diagonal, the application of its inverse is very cheap. For Gauss-Seidel, the splitting is $B = D - L$ and $C = U$, giving

$$u_{k+1} = (D - L)^{-1}Uu_k + (D - L)^{-1}b. \tag{A.8}$$

Note that the application of the inverse of $D - L$ simply involves a triangular solve. Lastly, SOR requires some over-relaxation factor $\omega > 1$. The splitting used for this method is given by $B = D-\omega L$ and $C = (1-\omega)D+\omega U$, which is actually a splitting for the equation $(\omega A)u = \omega b$. This gives the iteration

$$u_{k+1} = (D - \omega L)^{-1}[(1 - \omega)D + \omega U]u_k + \omega(D - \omega L)^{-1}b. \tag{A.9}$$

Note that letting $\omega = 1$ simply gives the Gauss-Seidel method. Again, the convergence behavior of these methods is determined by the spectral radius of the iteration matrix.

A.1.3 Krylov Subspace Methods

Krylov methods are another class of iterative methods for linear equations. This type of method proceeds by choosing the k^{th} iterate to minimize some measure of the error over the shifted k^{th} Krylov subspace $u_0 + \mathcal{K}_k$, where

$$\mathcal{K}_k = \text{span}\{r_0, Ar_0, \ldots, A^{k-1}r_0\}$$

and $r_0 = b - Au_0$. Frequently utilized Krylov methods include the conjugate gradient (CG), generalized minimal residual (GMRES), CGNR, and CGNE methods. CG only applies to symmetric positive-definite matrices (i.e. $A^T = A$ and $u^T Au > 0$ for all $u \neq 0$), and it minimizes $\|u - u^*\|_A$ over the shifted Krylov subspace, where we define the A–norm of some vector x to be $\|x\|_A = \sqrt{x^T Ax}$. CGNR and CGNE enable the use of CG for non-spd matrices by applying it to a different problem. CGNR proceeds by applying CG to the normal equations $A^T Au = A^T b$, which results in minimizing $\|b - Au\|_2^2$. CGNE applies CG to solve $AA^T y = b$ for y and recovers the solution by computing $u = A^T y$. This method minimizes $\|u^* - u\|_2^2$ each iteration. Note that both of these methods require some method for applying A^T to a vector. Both also have an additional disadvantage in that $\kappa(A^T A) = \kappa(AA^T) = \kappa(A)^2$. GMRES also works for non-spd matrices, and this method chooses the k^{th} iterate to minimize the residual $\|b - Au\|_2$ over the space $u_0 + \mathcal{K}_k$. An advantage of this sort of method is that it only requires the ability to apply the action of A (and A^T for CGNR and CGNE) on a vector, and explicit representation of the matrix is not required. The analysis for each of these methods is fairly similar, so due to its "essential equivalence" to Anderson acceleration shown in [62], we will consider GMRES from this point forward. Additionally, we will take $\|\cdot\|$ to refer to the l_2 norm. Before characterizing the convergence behavior of this method, we require an additional definition.

Definition A.5. *We call polynomial $p_k(z)$ a k^{th} degree residual polynomial if p_k is of degree k and $p_k(0) = 1$. We denote the set of all k^{th} degree residual polynomials by $\mathcal{P}_k$.*

We now overview some results which describe the behavior of this method.

Theorem A.2. *Let A be nonsingular and u_k be the k^{th} iteration of GMRES. Then for any residual polynomial $p_k(z) \in \mathcal{P}_k$, we have*

$$\frac{\|r_k\|}{\|r_0\|} \leq \|p_k(A)\|.$$

Proof. First note that we can express any $u \in u_0 + \mathcal{K}_k$ in the form

$$u = u_0 + \sum_{i=0}^{k-1} \gamma_i A^i r_0.$$

Thus we have

$$r = b - Au = b - Au_0 - \sum_{i=1}^{k} \gamma_{i-1} A^i r_0 = r_0 - \sum_{i=1}^{k} \gamma_{i-1} A^i r_0 = p_k(A) r_0,$$

for some residual polynomial $p_k(z)$. Hence, each residual polynomial in $\mathcal{P}_k$ has a one-to-one correspondence to some element in the shifted Krylov subspace $u_0 + \mathcal{K}_k$. Now, let the residual polynomial $p_k(z)$ be given, and r be the residual corresponding to this polynomial as defined above. Due to the minimization property of the the k^{th} GMRES residual, we have $\|r_k\| \leq \|r\|$, and we thus have

$$\|r_k\| \leq \|r\| = \|p_k(A) r_0\| \leq \|p_k(A)\| \|r_0\|.$$

□

From this result, we can show that the GMRES iteration will converge to the solution in a finite number of iterations.

Theorem A.3. *Let A be nonsingular. Then, the GMRES iteration will converge to the solution $A^{-1}b$ in at most n iterations.*

Proof. Consider the characteristic polynomial of A, $\bar{p}(z) = \det(A - zI)$. The characteristic polynomial of an $n \times n$ matrix is of degree n. Since A is nonsingular, we have $\bar{p}(0) \neq 0$, so we can then define $p_n(z) = \frac{\bar{p}(z)}{\bar{p}(0)}$, which is in $\mathcal{P}_n$. The matrix A is a root of its characteristic equation, so we have $p_k(A) = 0$. Then, from what was shown above we have $\|r_n\| = 0$, and thus u_n is the exact solution. □

Often, the problem size n is too large for this number of iterations to be practical. In general, GMRES is utilized as an iterative method which terminates when some specified error tolerance is satisfied. In order to further characterize the behavior of this method, Theorem A.2 may be used to derived additional results. We lastly present one final result for specifically diagonalizable matrices.

Theorem A.4. *Let A be nonsingular and diagonalizable with eigen-decomposition $A = Q\Lambda Q^{-1}$, and let u_k be the k^{th} GMRES iterate. Then, for any $p_k \in \mathcal{P}_k$*

$$\frac{\|r_k\|}{\|r_0\|} \leq \kappa(Q) \max_{\lambda \in \sigma(A)} |p_k(\lambda)|.$$

If A is normal, then it can be diagonalized by a unitary transformation, in which case $\kappa(Q) = 1$. This result shows that the convergence behavior of GMRES (at least for diagonalizable matrices) can be characterized in terms of the eigenvalues of the matrix A.

We conclude discussion of GMRES by considering some practical aspects of this method. GMRES is generally implemented by maintaing an orthonormal basis V_k for for the Krylov subspace $\mathcal{K}_k$ using an Arnoldi process. Given the orthonormal basis V_k, any $z \in \mathcal{K}_k$ can be represented as $z = V_k y$ for some $y \in \mathbb{R}^k$. In this way, the GMRES minimization problem can be formulated as solving

$$\min_{y \in \mathbb{R}^k} \|r_0 - AV_k y\|.$$

The Arnoldi process produces a sequence of orthonormal bases $\{V_k\}$ such that

$$AV_k = V_{k+1} H_k,$$

where $H_k \in \mathbb{R}^{(k+1)\times k}$ is upper Hessenberg. Because V_{k+1} has orthonormal columns, for any $v \in \mathbb{R}^{k+1}$ we have $\|V_{k+1} v\| = \|v\|$. Hence, the GMRES least-squares problem can be further reformulated as solving

$$\min_{y \in \mathbb{R}^k} \|\beta e_1 - H_k y\|,$$

where $\beta = \|r_0\|$ and $e_1 = (1, 0, \ldots, 0)^T \in \mathbb{R}^{k+1}$. This problem can be solved efficiently using Givens rotations. As an orthonormal basis for the Krylov subspace $\mathcal{K}_k$ is maintained throughout the iteration, storage may be an issue if a large number of iterations is required or if the problem size is very large. To address this, GMRES is frequently used in a way that the iteration is restarted after some given number of iterations. This method, referred to as GMRES(m), proceeds by performing m iterations of GMRES, and the purging the memory and restarting with u_m as a new initial iterate.

A.2 Nonlinear Equations

We now turn our attention to iterative methods for nonlinear equations. In this section, we are concerned with solving two types of problems. First, we consider fixed-point problems. For this type of problem, we seek a solution to the equation $u = G(u)$, where $u \in \mathbb{R}^n$ and $G : \mathbb{R}^n \to \mathbb{R}^n$. We refer to G as a fixed-point map, and a solution of this equation as a fixed-point of G. Next, we consider root finding problems. In these, we seek a solution to the equation $F(u) = 0$, where $u \in \mathbb{R}^n$ and $F : \mathbb{R}^n \to \mathbb{R}^n$. We refer to F as the residual for this equation. There is an obvious relation between these types of problems, and it is often possible to solve a nonlinear equation in either form. For instance, a fixed-point problem can be posed as a root finding problem by defining $F(u) = G(u) - u$, and in this case we call F the fixed-point residual. Similarly, a root finding problem can be formulated as a fixed-point problem by defining $G(u) = u + F(u)$, or possibly as $G(u) = u + M(u)F(u)$, where $M(u)$ is some nonsingular preconditioner matrix.

A.2.1 Preliminaries

As for linear problems, we begin by establishing some definitions and conventions before describing some commonly used methods for solving nonlinear equations. First, much of the forthcoming analysis for Newton's method is based on derivatives of the function F, so we need the following definition.

Definition A.6. *Let $F : \mathbb{R}^n \to \mathbb{R}^n$ be differentiable at $u \in \mathbb{R}^n$. We then define the Jacobian of $F(u)$ as the matrix $F'(u) \in \mathbb{R}^{n\times n}$ with (i, j) component given by*

$$F'(u)_{ij} = \frac{\partial F(u)_i}{\partial u_j}.$$

Next, we require the following definition.

Definition A.7. *Let $\Omega \subset \mathbb{R}^n$ and $F : \Omega \to \mathbb{R}^m$. We call F Lipschitz continuous on Ω with Lipschitz constant $\gamma \geq 0$ if*

$$\|F(u) - F(v)\| \leq \gamma\|u - v\|,$$

for all $u, v \in \Omega$. We additionally refer to F as a contraction map if it it Lipschitz continuous with Lipschitz constant $\gamma \in [0, 1)$.

The concept of a contraction mapping is very important for the analysis of fixed-point iteration in Section A.2.2, and we also leveraged it heavily in our analysis of Anderson acceleration. Lipschitz continuity also utilized in the following set of assumptions, which we refer to as the *standard assumptions*. This set of assumptions is important for convergence analysis for Newton's method in Section A.2.3.

> **Assumption A.1.** *Given a function F on the set Ω, we say F satisfies the standard assumptions if:*

1. *$F(u) = 0$ has a solution $u^* \in \Omega$.*
2. *$F'(u)$ is Lipschitz continuous with Lipschitz constant γ on Ω.*
3. *$F'(u^*)$ is nonsingular.*

We required a similar set of assumptions for analysis of Anderson acceleration for solving nonlinear fixed-point problems. Next, much of our analysis requires the following theorem, which is a multivariate formulation of the Fundamental Theorem of Calculus.

Theorem A.5. *Let F be differentiable in an open, convex set $\Omega \subset \mathbb{R}^n$. Then, for any $u, v \in \Omega$*

$$F(u) - F(v) = \int_0^1 F'(v + t(u - v))(u - v)\,\mathrm{d}t.$$

In particular, if the solution $u^* \in \Omega$ this implies $F(u) = \int_0^1 F'(u^* + te)e \, \mathrm{d}t$. Lastly, to characterize the rate at which a sequence converges to its limit, we introduce the following terminology.

Definition A.8. *Consider the sequence $\{u_k\} \subset \mathbb{R}^n$ with limit $u^* \in \mathbb{R}^n$. We say that*

- *$u_k \to u^*$ q-linearly with q-factor $\sigma \in (0, 1)$ if*

$$\|u_{k+1} - u^*\| \leq \sigma \|u_k - u^*\|$$

  for sufficiently large k.

- *$u_k \to u^*$ q-superlinearly if*

$$\lim_{k \to \infty} \frac{\|u_{k+1} - u^*\|}{\|u_k - u^*\|} = 0$$

- *$u_k \to u^*$ q-superlinearly with q-order $\alpha > 1$ if there is $K > 0$ such that*

$$\|u_{k+1} - u^*\| \leq K \|u_k - u^*\|^\alpha$$

- *$u_k \to u^*$ q-quadratically if there is $K > 0$ such that*

$$\|u_{k+1} - u^*\| \leq K \|u_k - u^*\|^2$$

Note that q-quadratic convergence is a particular case of q-superlinear convergence. Using these definitions, we define the following, weaker sense of convergence.

Definition A.9. *Consider the sequence $\{u_k\} \subset \mathbb{R}^n$ with limit $u^* \in \mathbb{R}^n$. We say that $u_k \to u^*$ r-linearly if there is a sequence $\{\xi_n\} \subset \mathbb{R}^n$ such that $\xi_k \to 0$ q-linearly and $\|u_k - u^*\| \leq \xi_k$. Likewise, we say $u_k \to u^*$ r-superlinearly or r-quadratically if $\|u_k - u^*\| \leq \xi_k$ where $\xi_k \to 0$ q-superlinearly or q-quadratically respectively. Lastly, $u_k \to u^*$ r-superlinearly with r-order α if $\xi_k \to 0$ q-superlinearly with q-order α.*

Whereas q-type convergence implies monotonic decrease in the quantity $\|u_k - u^*\|$ each iteration, r-type convergence simply means an improving worst case bound, though u_{k+1} is not necessarily a better approximation to u^* than u_k. Hence, we see that this is a weaker sense of convergence, as a q-type convergent series is convergent in the corresponding r-type, though the converse is not true. In both cases, assuming a comparable cost per iteration, a superlinearly convergent method would generally be considered superior to a linearly convergent method. However, in many cases the per iteration cost of a linearly convergent method is low enough in comparison to the superlinearly convergent method that the linearly convergent method converges in less time despite a higher iteration count.

A.2.2 Fixed-Point Iteration

We first consider fixed-point iteration for solving the fixed-point problem $u = G(u)$, where $u \in \mathbb{R}^n$ and $G : \mathbb{R}^n \to \mathbb{R}^n$. This method is given by the following

$$u_{k+1} = G(u_k),$$

where u_0 is a given initial iterate. We frequently also refer to this method as Picard iteration, but it may also be known as nonlinear Richardson iteration or successive substitution. The following theorem provides sufficient conditions for the existence of a unique fixed-point in some set, and characterizes the convergence behavior of fixed-point iteration toward this solution.

Theorem A.6. *Let Ω be a closed subset of $\mathbb{R}^n$ and G be a contraction mapping on Ω with constant $c \in [0, 1)$ such that $G(u) \in \Omega$ for all $u \in \Omega$. Then, G has a unique fixed-point $u^* \in \Omega$ and the fixed-point iteration with any $u_0 \in \Omega$ satisfies the following bounds*

$$\|u_{k+1} - u^*\| \leq c\|u_k\|,$$

and

$$\|F(u_{k+1})\| \leq c\|F(u)\|,$$

where we define $F(u) = G(u) - u$. These bounds state that both $\{u_k\} \to u^*$ an $\{F(u_k)\} \to 0$ q-linearly with q-factor c.

Proof. We first note that since $u_0 \in \Omega$ and $G(u) \in \Omega$ for all $u \in \Omega$, we have that $\{u_k\} \subset \Omega$. We will begin by showing that the sequence $\{u_k\}$ is Cauchy in Ω. We first show that the quantity $\|u_i - u_0\|$ remains bounded for any $i \geq 1$. We note that for any $i \geq 0$ we have

$$\|u_{i+1} - u_i\| = \|G(u_i) - G(u_{i-1})\| \leq c\|u_i - u_{i-1}\| \leq \cdots \leq c^i\|u_1 - u_0\|.$$

Then, for any $i \geq 1$ we have

$$\|u_i - u_0\| = \|\sum_{j=0}^{i-1} u_{j+1} - u_j\| \leq \sum_{j=0}^{i-1} \|u_{j+1} - u_j\| \leq \sum_{j=0}^{i-1} c^j\|u_1 - u_0\| \leq \frac{\|u_1 - u_0\|}{1 - c}.$$

Then, let $i, j > 0$ with $i > j$. We then have

$$\|u_i - u_j\| = \|G(u_{i-i}) - G(u_{j-1})\| \leq c\|u_{i-1} - u_{j-1}\| \leq \cdots \leq c^j\|u_{i-j} - u_0\| \leq c^j \frac{\|u_1 - u_0\|}{1 - c}.$$

The quantity on the right approaches 0 as $i, j \to \infty$, so $\{u_k\}$ is Cauchy in Ω. A closed subset of a complete space is also complete, so this implies that $\{u_k\}$ has a limit in $u^* \in \Omega$. Since G

is continuous this implies that u^* is a fixed-point of G. To show that this fixed-point is unique, suppose that G has two fixed-points $u^*, v^* \in \Omega$. If $u^* \neq v^*$ this would imply

$$\|u^* - v^*\| = \|G(u^*) - G(v^*)\| \leq c\|u^* - v^*\| < \|u^* - v^*\|,$$

which is a contradiction, so it follows that the fixed-point u^* is the unique fixed-point in Ω. We conclude by showing the bounds on the error $u_k - u^*$ and the residual $F(u_k)$. First, we have

$$\|u_{k+1} - u^*\| = \|G(u_k) - G(u^*)\| \leq c\|u_k - u^*\|.$$

Similarly, we have

$$\begin{aligned}\|F(u_{k+1})\| = \|G(u_{k+1}) - u_{k+1}\| = \|G(u_{k+1}) - G(u_k)\| \\ \leq c\|u_{k+1} - u_k\| = c\|G(u_k) - u_k\| = c\|F(u_k)\|.\end{aligned}$$

□

In many cases, this rate of convergence may be prohibitively slow, especially if the fixed-point map G is expensive to evaluate. It is for this reason that we are interested in Anderson acceleration as an alternative to this solution method.

A.2.3 Newton's Method

We next consider Newton's method for solving the root finding problem $F(u) = 0$ with $u \in \mathbb{R}^n$ and $F : \mathbb{R}^n \to \mathbb{R}^n$. This method is derived from approximating the function $F(u)$ by the following linear model about the current iterate u_c

$$L(u) = F(u_c) + F'(u_c)(u - u_c),$$

and choosing the next iterate u_+ to be the root of this linear model. That is, u_+ satisfies

$$0 = F(u_c) + F'(u_c)(u_+ - u_c),$$

and solving this for u_+ gives the Newton iteration

$$u_+ = u_c - F'(u_c)^{-1}F(u_c). \tag{A.10}$$

Before describing the convergence behavior of this method, we require the following (Lemma 4.3.1 from [29]).

Lemma A.2. *Suppose that the standard assumptions hold. Then, there exists $\delta > 0$ such that for all $u \in \mathcal{B}_\delta(u^*)$ it holds that*

$$\|F'(u)\| \leq 2\|F'(u^*)\|, \tag{A.11}$$

$$\|F'(u)^{-1}\| \leq 2\|F'(u^*)^{-1}\| \tag{A.12}$$

$$\|F'(u^*)^{-1}\|^{-1}\|e\|/2 \leq \|F(u)\| \leq 2\|F'(u^*)\|\|e\|. \tag{A.13}$$

We now present a theorem which describes the local convergence behavior of Newton's method. In this result, we allow for the case in which there is error in the evaluation of the function and its derivative. That is, Newton's method is applied using the residual $\hat{F}(u_c) = F(u_c) + \epsilon(u_c)$ and Jacobian $\hat{F}'(u_c) = F'(u_c) + \Delta(u_c)$. In this case, the Newton iteration is described by

$$u_+ = u_c - (F'(u_c) + \Delta(u_c))^{-1}(F(u_c) + \epsilon(u_c)). \tag{A.14}$$

Analysis for the standard method is given by letting $\Delta(u_c) = 0$ and $\epsilon(u_c) = 0$. Local behavior of Newton's method is described by the following.

Theorem A.7. *Let the standard assumptions hold. Then there exist $K, \delta, \delta_1 > 0$ such that if $u_c \in \mathcal{B}_\delta(u^*)$ and $\|\Delta(u_c)\| \leq \delta_1$ then u_+ as given by* (A.14) *is defined and satisfies*

$$\|e_+\| \leq K(\|e_c\|^2 + \|\Delta(u_c)\|\|e_c\| + \|\epsilon(u_c)\|). \tag{A.15}$$

Proof. We first let δ be small enough such that the results in A.2 hold. Then, note that we can rewrite (A.14) as

$$\begin{aligned} u_+ = u_c - F'(u_c)^{-1}F(u_c) \\ + [F'(u_c)^{-1} - (F'(u_c) + \Delta(u_c))^{-1}]F(u_c) + (F'(u_c) + \Delta(u_c))^{-1}\epsilon(u_c). \end{aligned}$$

Applying norms, this implies

$$\begin{aligned} \|e_+\| \leq \|e_c - F'(u_c)^{-1}F(u_c)\| \\ + \|F'(u_c)^{-1} - (F'(u_c) + \Delta(u_c))^{-1}\|\|F(u_c)\| + \|(F'(u_c) + \Delta(u_c))^{-1}\|\|\epsilon(u_c)\|. \end{aligned} \tag{A.16}$$

We now need to bound each of the terms on the right hand side. First, we have

$$e_c - F'(u_c)^{-1}F(u_c) = F'(u_c)^{-1}\int_0^1 [F'(u_c) - F'(u^* + te_c)]e_c \, \mathrm{d}t.$$

Then, by (A.12) and the Lipschitz continuity of γ

$$\|e_c - F'(u_c)^{-1}F(u_c)\| \leq \gamma\|F'(u^*)^{-1}\|\|e_c\|^2. \tag{A.17}$$

We next note that Lemma A.2 implies that $F'(u_c)$ is nonsingular, so we can write

$$(F'(u_c) + \Delta(u_c))^{-1} = [F'(u_c)(I + F'(u_c)^{-1}\Delta(u_c))]^{-1}.$$

Then, letting $\delta_1 = \|F'(u^*)^{-1}\|^{-1}/4$, by (A.12) we have

$$\|\Delta(u_c)\| \leq \|F'(u^*)^{-1}\|^{-1}/4 \leq \|F'(u_c)^{-1}\|^{-1}/2,$$

and thus

$$\|F'(u_c)^{-1}\Delta(u_c)\| \leq 1/2.$$

By the Banach Lemma, $I + F'(u_c)^{-1}\Delta(u_c)$ is nonsingular, so u_+ is well defined, and

$$\|(I + F'(u_c)^{-1}\Delta(u_c))^{-1}\| \leq \frac{1}{1 - 1/2} = 2.$$

From this it follows that

$$(F'(u_c) + \Delta(u_c))^{-1} = (I + F'(u_c)^{-1}\Delta(u_c))^{-1}F'(u_c)^{-1},$$

and thus by (A.12)

$$\|(F'(u_c) + \Delta(u_c))^{-1}\| \leq \|(I + F'(u_c)^{-1}\Delta(u_c))^{-1}\|\|F'(u_c)^{-1}\| \leq 4\|F'(u^*)^{-1}\|. \tag{A.18}$$

Next, we can write

$$\begin{aligned} F'(u_c)^{-1} - (F'(u_c) + \Delta(u_c))^{-1} &= [F'(u_c)^{-1}(F'(u_c) + \Delta(u_c)) - I](F'(u_c) + \Delta(u_c))^{-1} \\ &= F'(u_c)^{-1}\Delta(u_c)(F'(u_c) + \Delta(u_c))^{-1}. \end{aligned}$$

By (A.12) and (A.18), we then have

$$\begin{aligned} \|F'(u_c)^{-1} - (F'(u_c) + \Delta(u_c))^{-1}\| &\leq \|F'(u_c)^{-1}\|\|\Delta(u_c)\|\|(F'(u_c) + \Delta(u_c))^{-1}\| \\ &(2\|F'(u^*)^{-1}\|)\|\Delta(u_c)\|(4\|F'(u^*)^{-1}\|) = 8\|F'(u^*)^{-1}\|^2\|\Delta(u_c)\|. \end{aligned} \tag{A.19}$$

Now, combining (A.16), (A.17), (A.18), and (A.19) with (A.11) and (A.13)

$$\|e_+\| \leq \gamma \|F'(u^*)^{-1}\| \|e_c\|^2 + 16\|F'(u^*)\| \|F'(u^*)^{-1}\|^2 \|\Delta(u_c)\| \|e_c\| + 4\|F'(u^*)^{-1}\| \|\epsilon(u_c)\| \quad \text{(A.20)}$$

Finally, (A.15) follows from setting

$$K = \max\{\gamma \|F'(u^*)^{-1}\|, 16\|F'(u^*)\| \|F'(u^*)^{-1}\|^2, 4\|F'(u^*)^{-1}\|\}.$$

□

This result shows that inaccuracies in the Jacobian evaluation only affect the iteration with respect to the rate of convergence. Conversely, the effect of inaccuracies in the function evaluation are only felt when the residual is on the order to the size of the error in the evaluation. This means that the iteration will appear to converge as normal up to some stagnation point proportional to the size of the error in the function evaluation, and after this point the iteration will stagnate.

For many problems, forming and storing a Jacobian matrix is either impractical or impossible. In order to avoid this, we formulate the Newton iteration as $u_+ = u_c + d$, where d solves the linear system

$$F'(u_c)d = -F(u_c).$$

This may be solved by a Krylov method like GMRES, which only requires some representation of the Jacobian–vector product $F'(u_c)v$. It is typical to find a Newton direction d which satisfies

$$\|F'(u_c)d + F(u_c)\| \leq \eta_c \|F(u_c)\|.$$

In this, η_c is referred to as the forcing term. The rate of convergence of the iteration may be varied by selecting the forcing term in various ways. For instance, letting $\eta_c = \eta \in (0, 1)$ will result in linear convergence, the rate of which will be determined by the choice of η. Letting $\eta_c = O(\|F(u_c)\|)$ will retain the q-quadratic convergence that is expected for the standard Newton's method.

In cases where an analytic Jacobian–vector product is not available, it may be approximated by the following finite–difference approximation

$$F'(u_c)v \approx \frac{F(u_c + hv) - F(u_c)}{h}, \quad \text{(A.21)}$$

where h is some perturbation parameter. Applying Newton's method with the Newton direction being computed by a Krylov method using a finite difference Jacobian–vector product gives a

method known as Jacobian–free Newton–Krylov (JFNK). The obvious advantage of this method is that no analytic derivative information needs to be computed. This only requires the ability to evaluate the residual F. However, this method does carry some drawbacks and challenges. First, assuming that $F(u_c)$ has already been computed, each finite–difference Jacobian–vector product requires a residual evaluation. This means that each inner Krylov iteration in the linear solve requires a residual evaluation. This may be problematic if the residual is computationally expensive and many Krylov iterations are needed in the linear solve. Hence, good preconditioning for the linear solve may be necessary. Next, appropriate selection of the perturbation parameter h may be difficult for some problems. The difference approximation (A.21) is an $O(h)$ approximation to $F'(u_c)v$, so it would seem that it is best to set h as small as possible. However, in actuality there is always some level of error in the residual evaluation, at the very least roundoff error. Hence, the approximation is actually computed using $F(u) + \epsilon(u)$, where ϵ represents the error in the residual evaluation, and this gives the approximation

$$F'(u_c)v \approx \frac{[F(u_c + hv) + \epsilon(u_c + hv)] - [F(u_c) - \epsilon(u_c)]}{h}.$$

Assuming some bound on the size of the error $\|\epsilon(u)\| \leq \epsilon_0$, this gives an $O(h + \epsilon_0/h)$ approximation to $F'(u_c)v$. The perturbation parameter h should be chosen to minimize the quantity in the O–term, and this is done when $h \approx \sqrt{\epsilon_0}$. Hence, if the ϵ term represents double precision roundoff error ($\epsilon_0 \approx 10^{-15}$), then a choice of $h = 10^{-7}$ is reasonable. If there is some additional level of uncertainty in the evaluation of F, then more care must be taken in the selection of h.

We lastly consider globalization of this method. The theory presented above is local, so it requires the iteration to be sufficiently close to a solution to be applicable. By globalization methods, we refer to methods for improving the likelihood of converging given a poor starting point. With a globalization method, the method will converge to a solution or fail to do so in a small number of ways. There are several methods for globalizing Newton's method, such as trust region methods [30] and line searches, which we describe here. In Algorithm 7, we describe a backtracking line search for Newton's method. This is likely the simplest form of a line search. There are other line search methods involving polynomial interpolation which may give better performance. In this algorithm, the condition $\|F(u_t)\| \leq (1 - \alpha\lambda)\|F(u)\|$ is referred to as sufficient decrease of $\|F\|$. A typical value for the parameter α is 10^{-4}. Also note that this algorithm allows flexibility in the reduction of the step size λ with the selection of $\sigma \in [\sigma_0, \sigma_1]$, where $0 < \sigma_0 < \sigma_1 < 1$. This will enforce

$$\sigma_0\lambda_{old} \leq \lambda_{new} \leq \sigma_1\lambda_{old}.$$

The lower bound safeguards against too large a step reduction, which may result in stagnation

Algorithm 7 Netwon's method with backtracking line search

Given initial iterate u.
while not converged **do**
    Find d such that $\|F'(u)d + F(u)\| \leq \eta\|F(u)\|$. If d can not be computed, terminate with failure.
    Set $\lambda = 1$.
    while $\lambda > \lambda_{min}$ **do**
        $u_t = u + \lambda d$.
        if $\|F(u_t)\| \leq (1 - \alpha\lambda)\|F(u)\|$ **then**
            Set $u = u_t$.
            break
        else
            Choose $\sigma \in [\sigma_0, \sigma_1]$.
            Set $\lambda = \sigma\lambda$.
        end if
    end while
end while

of the method. Typical values of these parameters are $\sigma_0 = 0.1$ and $\sigma_1 = 0.5$. Some obvious ways that the iteration produced by this algorithm may fail include singluar $F'(u_k)$ for some k, in which case the computation of the direction d will fail, the sequence converges to a local minimum of $\|F\|$ which is not a root, or the sequence becomes unbounded.

We conclude this section by presenting a theorem describing the behavior of Algorithm 7. We first need to introduce the following assumption.

Assumption A.2. *There exist $r, \gamma, m_f > 0$ such that F is defined, F' is Lipschitz continuous with constant γ, and $\|F'(u)^{-1}\| \leq m_f$ on the set*

$$\Omega(\{u_k\}, r) = \cup_{k=0}^{\infty}\{u|\|u - u_k\| \leq r\},$$

where $\{u_k\}$ is the sequence produced by Algorithm 7.

With this, we have the following theorem.

Theorem A.8. *Let $u_0 \in \mathbb{R}^n$ and $\alpha \in (0, 1)$ be given. Suppose that $\{u_k\}$is computed by Algorithm 7 with forcing terms $\{\eta_k\}$, the series $\{u_k\}$ remains bounded, and Assumption A.2 holds. Then $\{u_k\}$ converges to a solution u^* of $F(u) = 0$ at which the standard assumptions hold. Furthermore, for k sufficiently large, full steps are taken and the convergence behavior in this final phase is given by the local convergence theory described above.*

A.2.4 Quasi-Newton Methods

We conclude this appendix by considering quasi-Newton methods for solving the root finding problem $F(u) = 0$ for $u \in \mathbb{R}^n$ and $F : \mathbb{R}^n \to \mathbb{R}^n$. Quasi-Newton methods differ from Newton's method by maintaing both an approximation to the solution and an approximation to the Jacobian. That is, given the current approximation to the solution $u_c \in \mathbb{R}^n$ and the current approximation to the Jacobian $B_c \in \mathbb{R}^{n\times n}$, the quasi-Newton iteration takes the form

$$u_+ = u_c - B_c^{-1}F(u_c). \tag{A.22}$$

The general procedure for a quasi-Newton method is as follows:

- Compute the quasi-Newton direction $d = -B_c^{-1}F(u_c)$.
- Compute $u_+ = u_c + \lambda d$ using a line search.
- Compute the updated Jacobian approximation B_+ using B_c, u_c, and u_+.

There are many quasi-Newton methods, such as BFGS, DFP, PSB, and SR1 [30]. The method that we will consider in this section is Broyden's method. This computes the approximate Jacobian update by

$$B_+ = B_c + \frac{(y - B_c s)s^T}{s^T s}, \tag{A.23}$$

where $y = F(u_+) - F(u_c)$ and $s = u_+ - u_c$. Note that B_+ satisfies the equation

$$B_+ s = y.$$

We call this a secant equation, so Broyden's method is said to be a secant update method. Recall that Anderson acceleration can be written in the form of a quasi-Newton method which satisfies multiple secant conditions.

A significant advantage of this type of iteration over Newton-iterative methods in that only one residual evaluation is required each iteration. There is no inner iteration when computing the Newton direction. Furthermore, Broyden's method can be implemented in very computationally and memory efficient way. This is due to the fact that the updated approximate Jacobian is a rank one update of the previous approximate Jacobian. As shown in [30], this fact can be used with the Sherman-Morrison-Woodbury formula to recursively reduce the approximate Jacobian inverse to a product of rank one updates to the identity, resulting in the following:

$$B_i^{-1} = \prod_{j=0}^{i-1} \left(I + \frac{s_{j+1}s_j^T}{\|s_j\|^2} \right) B_0^{-1}.$$

Algorithm 8 Broyden's method

Given initial iterate u and approximate Jacobian B_0.
Set $s_0 = -B_0^{-1}F(u_0)$
Set $u = u + s_0$
Set $k = 0$.
while not converged **do**
    Set $z = -B_0^{-1}F(u)$.
    for $j = 0, \ldots, k-1$ **do**
        Set $z = z + s_{j+1}s_j^T z/\|s_j\|^2$.
    end for
    Set $s_{k+1} = z/(1 - s_k^T z/\|s_k\|^2)$.
    Set $u = u + s_{k+1}$.
    Set $k = k + 1$.
end while

This allows for the approximate Jacobian inverse to be applied cheaply. Using this formulation of the approximate inverse, the algorithm for Broyden's method can be formulated as shown in Algorithm 8. Note that this requires the storage of the set of step vectors $\{s_j\}$, and this set grows by one vector each iteration. For some problems, the storage of a vector per iteration may be too great a memory burden. To address this, this method can be augmented with a restart in a similar manner to GMRES.

We lastly overview some convergence theory for Broyden's method. Mirroring the definition $e = u - u^*$, we define the error in the Jacobian as $E = B - F'(u^*)$. Analysis for quasi-Newton methods is significantly different from that which we presented for the standard Newton's method, and relies on the following condition, which is referred to as the Dennis–Moré condition.

$$\lim_{k\to\infty} \frac{\|E_k s_k\|}{\|s_k\|} = 0. \tag{A.24}$$

This is used in the following theorem, which says that if the Broyden iteration converges to a solution and the Dennis–Moré condition holds, then the convergence is q-superlinear.

Theorem A.9. *Let the standard assumptions hold, let the sequence of nonsingular matrices $\{B_n\} \subset \mathbb{R}_{n\times n}$ and initial iterate $u_0 \in \mathbb{R}^n$ be given, and let the sequence $\{u_k\}$ be defined by* (A.22)*. Assume that $u_k \neq u^*$ for any k. Then, $u_k \to u^*$ q-superlinearly if and only if $u_k \to u^*$ and the Dennis–Moré condition* (A.24) *holds.*

Hence, Broyden's method can be analyzed by first establishing conditions for which the method will be convergent, and then verifying whether the Dennis–Moré condition holds, in which case the convergence will be q-superlinear. We first present the following theorem which claims that Broyden's method is locally q-linearly convergent.

Theorem A.10. *Let the standard assumptions hold, and $r \in (0,1)$ be given. Then there exist δ and δ_B such that if $\|e_0\| \leq \delta$ and $\|E_0\| \leq \delta_B$, then the Broyden sequences $\{u_k\}$ and $\{B_k\}$ are defined and $u_k \to u^*$ q-linearly with q-factor at most r.*

Note that unlike the analysis for Newton's method, this additionally requires a sufficiently good initial approximation to the Jacobian. The proof of this theorem relies on a concept referred to in [29] as "bounded deterioration" of the Jacobian approximation, which allows for the error in the Jacobian approximation $\|E_k\|$ to be bounded throughout the iteration. The final theorem we present for Broyden's method claims that the Dennis-Moré condition in fact holds, so the iteration locally converges q-superlinearly.

Theorem A.11. *Let the standard assumptions hold. Then there exist δ and δ_B such that if $\|e_0\| \leq \delta$ and $\|E_0\| \leq \delta_B$, then the Dennis–Moré condition holds for the Broyden iteration, and $u_k \to u^*$ q-superlinearly.*

Appendix B

Trilinos

B.1 Trilinos Overview

We now overview the numerical analysis software package Trilinos [27]. Trilinos is a collection of software developed at Sandia National Laboratories intended for use in solving complex, large-scale problems in computational science and engineering. Trilinos is divided into packages which cover a wide range of functionality. There are Trilinos packages focused on data structures, linear solvers, nonlinear solvers, preconditioners, automatic differentiation, and a variety of other applications. In this appendix, we describe the purpose of the Trilinos packages utilized in the work, and then describe how the Tribal Build, Integrate, and Test System (TriBITS) [4], which builds upon the CMake [37] makefile generation software package, is used to configure and build this software.

B.2 Relevant Packages

B.2.1 Teuchos

The package Teuchos contains a collection of common tools used in most other Trilinos packages. Teuchos is a core Trilinos package, and it is required to be enabled in order to utilize many other Trilinos packages. The functionality of this package is divided among five subpackages: Core, ParameterList, Comm, Numerics, and Remainder.

As the name suggests, the Teuchos Core subpackages contains many core features of Teuchos. This package contains a variety of tools dealing with areas like outputting, exception handling, and unit testing. A particularly widely used component of this package is its memory management tools, specifically its various array types and the `Teuchos::RCP` class [3]. This reference-counting smart pointer class is used for dynamic memory allocation and automatically deallocates storage when the reference count to an object goes to zero, which helps prevent

memory leaks. Additionally, this class features support for the detection and resolving of circular references.

The main component of the ParameterList subpackage is the `Teuchos::ParameterList` class. This class represents a hierarchical database which stores entries as a key–value pair. The key specifies the name of the parameter as a string, and the value is templated so it can hold any data type. This class is used in several other Trilinos packages as part of a factory design pattern. In this, the user creates a parameter list which describe various aspects of an object to be created, and this is passed to a factory class which handles creation of the object. This subpackage also contains tools for command line parsing and constructing parameter lists from XML input files.

The Comm subpackage contains several utilities for parallel communication, including an abstract communicator interface which provides routines for a subset of MPI functions. Implementations of this interface for single-processor programs and multi-processor programs utilizing MPI are included. This package also contains tools dealing with performance monitoring, including timers and flop counters.

The Numerics subpackage contains BLAS and LAPACK wrapper classes as well several data structures utilizing these.

B.2.2 Epetra

Epetra is a package which implements distributed-memory double precision vector and sparse matrix types, including routines for linear algebra with these distributed objects, and various associated utility classes. Commonly used matrix and vector types include the `Epetra_Vector`, `Epetra_MultiVector`, and `Epetra_CrsMatrix`. Construction of these types requires an `Epetra_Map`, which describes the distribution of the entries across all processes including local indices of the entries within a process and global indices across all processes. Required for the construction of an `Epetra_Map` is an `Epetra_Comm` object. For objects on a single process, this should be an `Epetra_SerialComm`, and for multi-processor distributed objects this should be an `Epetra_MpiComm`, which is a wrapper for a raw MPI communicator. Redistribution of data between objects with different maps is accomplished through the `Epetra_Import` and `Epetra_Export` classes. Many of the linear and nonlinear solver packages in Trilinos support Epetra data types.

B.2.3 Tpetra

Tpetra is a package very similar to Epetra, in that it implements distributed vectors and sparse matrix types, as well as linear algebra routines for these objects. Many objects in Tpetra have direct analogs in Epetra: `Tpetra::Map` to `Epetra_Map`, `Tpetra::MultiVector` to `Epetra_`-

`MultiVector`, `Tpetra::CrsMatrix` to `Epetra_CrsMatrix`, etc. A significant difference between these packages is that the majority of Tpetra classes include several template parameters. Potential template parameters for Tpetra objects include `Scalar`, `LocalOrdinal`, `GlobalOrdinal`, and `Node` types.

The `Scalar` template determines the type of data being stored by the vector and matrix types. While an `Epetra_MultiVector` may only contain double precision data, a `Tpetra::MultiVector` may be created to contain data of any scalar type. The `LocalOrdinal` and `GlobalOrdinal` templates determine the ordinal types used for local and global indexing of entries. Whereas local and global indices in Epetra are of the type `int`, templating on the local and global index types allows Tpetra objects to store larger objects than would be allowable with global indices of the type `int`. Additionally, the separation of the indexing types allows for local indices to be smaller ordinal types than the global indices, hence requiring less memory. The `Node` template defines the Kokkos node type for the object. Tpetra is said to be "hybrid parallel," meaning it utilizes distributed-memory parallelism through MPI as well as shared-memory parallelism within an MPI process. The `Node` template determines what sort of shared-memory parallelism is employed. Kokkos is a separate package of Trilinos which implements computational kernels for on-node shared-memory parallel operations. This currently includes support for programming models including OpenMP, Pthreads, and Cuda. In general, the default for the `Node` template provides an intelligent value, so this template may be ignored by the user in many cases.

B.2.4 Thyra

The Thyra package contains a set of interfaces for abstract numerical algorithms. A major component of this package is interface for abstract linear operator/vector operations. This includes interfaces for vector spaces, vectors, and linear operators. The vector space objects include routines for creating members from the vector space and define an inner product on the space. Vector and multi-vector classes include routines for scaling and adding together vectors from compatible vector spaces, and computing vector norms. The operator classes define an interface for applying the operator to a vector/multi-vector from the domain vector space and returning a vector/multi-vector from the range vector space. Another useful capability of this package is the nonlinear model evaluator interfaces. This includes interfaces for defining what inputs a nonlinear function will accept and what outputs it supports, including the residual, response functions, and sensitivity information. Additionally, the package includes adaptors which create concrete implementations of Thyra ojbects using Epetra and Tpetra.

B.2.5 Belos

Belos is a package for solving sparse, large-scale systems of linear equations. It includes a variety of iterative Krylov subspace solvers. Belos is extensible and interoperable in the sense that it treats operators and vectors as opaque objects. As Krylov methods only require an expression for a matrix-vector product rather than the actual matrix itself, Belos solvers are implemented by utilizing traits classes which define vector operations (addition, scalar multiplication, inner products) and operator application for given vector and operator types. Support for Epetra and Tpetra objects is included in this package. Some of the linear solvers in this package include:

- Single vector and block generalized minimal residual method (GMRES).
- Single vector and block conjugate gradient method (CG).
- Flexible GMRES.
- Biconjugate gradient stabilized method (BiCGStab).

B.2.6 Anasazi

Anasazi is an analogous package to Belos for the solution of sparse, large-scale eigenvalue problems. Like Belos, this package allows for use of arbitrary compatible vector and operator types through use of abstract interfaces for defining elementary vector/operator operations. Anasazi includes support for Epetra and Tpetra objects, as well as Thyra. Some of the eigensolvers included in this package are:

- Block Krylov-Schur.
- Block Davidson.
- Generalized Davidson.
- Riemannian Trust-Region.

B.2.7 Ifpack2

Ifpack2 is a package for incomplete factorizations and domain decomposition, specifically for use with Tpetra objects. This package includes two incomplete factorizations, which it refers to as ILUT and RILU(k). Each of these factorizations performs work within a single MPI process, so all work is local, but the quality of the factorization worsens as more processes are utilized. With respect to domain decomposition, this package includes and additive Schwarz operator. In addition to this, Ifpack2 includes implementations of operators corresponding to the stationary

iterative methods Jacobi iteration, Gauss-Seidel iteration, successive over-relaxation (SOR), and symmetric SOR (SSOR). The objects in this package can be used for several purposes, such as preconditioning a linear system or as a smoother in a multigrid method.

B.2.8 ML

ML is a package for multigrid solvers and preconditioners. It is intended for use in solving large sparse linear systems resulting from discretization of elliptic PDEs. Multigrid methods accelerate solution a linear system of equations on a fine grid using a sequence of corrections computed on a hierarchy of coarser grids. A multigrid methods generally consist of operations for restriction (mapping from a finer grid to a coarser grid), prolongation (mapping from a coarser grid to a finer grid), smoothing (reduction of high frequency error on a given grid), and direct solution at the coarsest grid in some order. ML provides a general framework for creating multigrid solvers and preconditioners, and includes implementations of several commonly used multigrid methods. Several other Trilinos packages may be utilized as smoothers and coarse grid solvers.

B.2.9 NOX

NOX is a package of nonlinear solver software. It includes a variety of Newton-based methods and trust region algorithms. Newton-based methods are implemented using the `NOX::Solver::LineSearchBased` class. Additionally, JFNK is supported through supplying a matrix-free operator to the linear solver for the Newton direction. Support for globalization methods is implemented through a variety of line search options, including backtracking, polynomial, or user defined line searches.

NOX solvers work with abstract interfaces, so users must either provide a concrete implementation of the interfaces or use one provided with the package. NOX includes support for Epetra and Thyra data types. Hence, Epetra objects may be used directly with NOX. Conversely, NOX does not include direct support for Tpetra data types. However, Thyra includes adapters for Tpetra, so Tpetra objects may be used with NOX by first wrapping them in the appropriate Thyra objects.

NOX also contains a subpackage LOCA, a software library for continuation and bifurcation analysis, which we do not consider in depth here.

B.2.10 PIKE

The PIKE package provides a set of interfaces and utilities for black-box code coupling. This includes interfaces for solvers, single-physics model evaluators, and data transfers, as well as

support for parallel memory management. We describe this package in greater detail in Section 2.2.5.

B.3 Configuring and Building Trilinos

Trilinos uses the TriBITS framework [4] in order to configure, build, and test code. This software is build on top of the CMake makefile generation software [37]. TriBITS is specialized for large distributed, possibly multi-repository, CMake projects. Each package of Trilinos has packages on which it depends, or which depend on it, and given a set of desired packages TriBITS automatically handles these dependencies and determines the full set of packages that will be built. It is also used to specify other configure time options, like the build type, supported third party libraries, compiler options, etc. When configuring Trilinos, the user must specify a list of options as shown in the example configure script below:

```
#!/bin/bash

EXTRA_ARGS=$@

rm -rf CMakeCache.txt
rm -rf CMakeFiles

##-------------------------------------------##

cmake \
    -D CMAKE_INSTALL_PREFIX:PATH=$PWD \
    -D CMAKE_BUILD_TYPE:STRING=DEBUG \
    -D CMAKE_VERBOSE_MAKEFILE:BOOL=OFF \
    -D TPL_ENABLE_MPI:BOOL=ON \
    -D TPL_ENABLE_Boost:BOOL=OFF \
    -D Trilinos_ENABLE_DEBUG:BOOL=ON \
    -D Trilinos_ENABLE_ALL_PACKAGES:BOOL=OFF \
    -D Trilinos_ENABLE_ALL_FORWARD_DEP_PACKAGES:BOOL=ON \
    -D Trilinos_ENABLE_ALL_OPTIONAL_PACKAGES:BOOL=ON \
    -D Trilinos_ENABLE_NOX:BOOL=ON \
    -D Trilinos_ENABLE_STK:BOOL=OFF \
    -D Trilinos_ENABLE_TESTS:BOOL=OFF \
    -D NOX_ENABLE_TESTS:BOOL=ON \
```

```
-D Trilinos_ASSERT_MISSING_PACKAGES:BOOL=OFF \
$EXTRA_ARGS \
$HOME/Trilinos/trilinos
```

In this script, we first invoke the `cmake` command, which calls the `cmake` executable to configure the source tree specified in the last line of the script. In this case we assume that the Trilinos source tree is located in the directory `$HOME/Trilinos/trilinos`. All the lines between invoking `cmake` and specifying the source tree define configuration options, including `$EXTRA_ARGS` which appends any additional options specified on the command line. These options specify which Trilinos packages and third-party libraries code should be built for, as well as specifying various other options. We first indicate to configure for a debug build, which will enable array bounds checking and other runtime checks. The line `-D TPL_ENABLE_MPI:BOOL=ON` tells CMake that we support building code which depends on MPI, while `-D TPL_ENABLE_Boost:BOOL=OFF` indicates that any code that depends on the third-party library Boost should not be build. Similarly, `-D Trilinos_ENABLE_NOX:BOOL=ON` says that the Trilinos package NOX should be enabled, which will also enable all packages on which this package has a necessary dependency, and `-D NOX_ENABLE_TESTS:BOOL=ON` tells CMake that we would like the unit tests for the NOX package to be built. When CMake has successfully completed configuration, the code can be built by invoking the `make` command, and any enabled unit tests can be run with the `ctest` command.

Appendix C

Tiamat Input Files

Tiamat simulations utilize the VERA common input format [41] to define problem specifications. The VERA common input format is a plain-text ASCII file which describes reactor state conditions, geometric specifications, and input parameters for physics codes. The file is divided into several blocks, each one describing some component of the simulation. The input blocks used in Tiamat simulations include:

- `[CASEID]` - This block defines a name for the simulation.
- `[STATE]` - This block describes several parameters relating to operating conditions (power level, pressure, inlet temperature, etc). Multiple instances of this block can be used to describe changing operating conditions during a time dependent simulation.
- `[CORE]` - This block provides the geometric/physical description of the reactor core, including the layout of fuel assemblies.
- `[ASSEMBLY]` - This block provides the geometric/physical description of the fuel assemblies in the core, including the layout of fuel rods and guide tubes. This layout is typically specified using octant symmetry.
- `[EDITS]` - This block defines the axial bounds for the cells which comprise the coupling mesh.
- `[BISON]` - This block defines input parameters for the Bison application code.
- `[COBRATF]` - This block defines input parameters for the CTF application code.
- `[MPACT]` - This block defines input parameters for the MPACT application code.

- `[COUPLING]` - This block defines parameters related to coupling of physics codes, specifically relaxation factors, global convergence tolerances, and the maximum number of coupled iterations.
- `[TIAMAT]` - This block defines parameters specific to the Tiamat coupling.

This input format also supports the blocks `[CONTROL]`, `[DETECTOR]`, and `[INSERT]`, which describe the geometry and location of control rods, detectors, and burnable poisons, but these are not used in any of the tests considered here. More detailed descriptions of the fields contained within each block are given in [41].

Each of the application codes in Tiamat utilizes different input formats, so prior to a Tiamat simulation, several pre-processing steps are required to generate the input for each of the applications. First, a pre-processor `react2xml.pl` converts the VERA input file into a machine-readable XML file. MPACT reads directly from this XML file, but Bison and CTF both have their own input formats. To generate these input files, two more pre-processors are used. The pre-processor `xml2ctf` generates a CTF input file using data specified in the XML file. Similarly, the pre-processor `xml2moose` generates a Bison input files from the XML file, though this additionally requires template input files for fuel rods and guide tubes.

C.1 17x17 Assembly Test Inputs

The VERA common input file for CASL Progression Problem 6a is given by the following.

```
[CASEID]
  title 'CASL Problem 6a'

[STATE]
  power  100.0         ! %
  tinlet 559.0 F       ! F
  boron  1300          ! ppmB
  pressure 2250        ! psia

  tfuel 900.0 K        ! K - 600K  Not used with T/H feedback! set to 900K with feedback
  modden 0.743         ! g/cc      Not used with T/H feedback!

  feedback on
  sym full

[CORE]
  size 1                ! 1x1 single-assembly
  rated 17.67   0.6824  ! MW, Mlbs/hr
  apitch 21.5
  height 406.328

  core_shape
```

```
    1

  assm_map
    A1

  lower_plate ss 5.0 0.5   ! mat, thickness, vol frac
  upper_plate ss 7.6 0.5   ! mat, thickness, vol frac
  lower_ref   mod 26.0 1.0
  upper_ref   mod 25.0 1.0


  bc_rad reflecting

  mat he     0.000176
  mat inc    8.19
  mat ss     8.0
  mat zirc   6.56 zirc4
  mat aic   10.20
  mat pyrex  2.23
  mat b4c    6.56

[ASSEMBLY]
  title "Westinghouse 17x17"
  npin 17
  ppitch 1.260

  fuel U31 10.257 95.0 / 3.1

!=== material label, key_name, density (lib_name defaults to key_name)
  mat he   0.000176
  mat inc  8.19
  mat ss   8.0
  mat zirc 6.56 zirc4

  cell 1     0.4096 0.418 0.475 / U31 he zirc
  cell 100          0.561 0.602 / mod    zirc       ! guide tube
  cell 200          0.561 0.602 / mod    zirc       ! instrument tube
  cell 7            0.418 0.475 / mod    mod        ! empty location
  cell 8            0.418 0.475 /     he zirc       ! plenum
  cell 9                  0.475 /        zirc       ! pincap

  lattice FUEL1
     200
       1 1
       1 1 1
     100 1 1 100
       1 1 1   1 1
       1 1 1   1 1 100
     100 1 1 100 1   1 1
       1 1 1   1 1   1 1 1
       1 1 1   1 1   1 1 1 1
```

```
lattice LGAP1
   200
     7 7
     7 7 7
   100 7 7 100
     7 7 7   7 7
     7 7 7   7 7 100
   100 7 7 100 7   7 7
     7 7 7   7 7   7 7 7
     7 7 7   7 7   7 7 7 7

lattice PLEN1
   200
     8 8
     8 8 8
   100 8 8 100
     8 8 8   8 8
     8 8 8   8 8 100
   100 8 8 100 8   8 8
     8 8 8   8 8   8 8 8
     8 8 8   8 8   8 8 8 8

lattice PCAP1
   200
     9 9
     9 9 9
   100 9 9 100
     9 9 9   9 9
     9 9 9   9 9 100
   100 9 9 100 9   9 9
     9 9 9   9 9   9 9 9
     9 9 9   9 9   9 9 9 9

axial A1    6.050
    LGAP1  10.281
    PCAP1  11.951
    FUEL1 377.711
    PLEN1 393.711
    PCAP1 395.381
    LGAP1 397.501

grid END inc  1017 3.866
grid MID zirc 875  3.810

grid_axial
    END  13.884
    MID  75.2
    MID 127.4
    MID 179.6
    MID 231.8
    MID 284.0
    MID 336.2
```

```
      END 388.2

  lower_nozzle  ss 6.05  6250.0  ! mat, height, mass (g)
  upper_nozzle  ss 8.827 6250.0  ! mat, height, mass (g)

[EDITS]

!  3in intervals in active fuel
  axial_edit_bounds
          11.951
          15.817
          24.028
          32.239
          40.45
          48.662
          56.873
          65.084
          73.295
          77.105
          85.17
          93.235
          101.3
          109.365
          117.43
          125.495
          129.305
          137.37
          145.435
          153.5
          161.565
          169.63
          177.695
          181.505
          189.57
          197.635
          205.7
          213.765
          221.83
          229.895
          233.705
          241.77
          249.835
          257.9
          265.965
          274.03
          282.095
          285.905
          293.97
          302.035
          310.1
          318.165
          326.23
```

```
          334.295
          338.105
          346.0262
          353.9474
          361.8686
          369.7898
          377.711

[COBRATF]
  nc      1             ! conduction option - radial conduction
  irfc    2             ! friction factor correlation default=2
  dhfrac  0.00          ! fraction of power deposited directly into coolant
  hgap    5678.3        ! gap conductance
  epso    0.001
  oitmax  5
  iitmax  40
  gridloss END 0.9070   ! spacer grid loss coefficient
  gridloss MID 0.9065   ! spacer grid loss coefficient
  dtmin   0.000001
  dtmax   0.1
  tend    0.1
  rtwfp   1000.0
  maxits  100000
  parallel 0
  courant 0.8
  global_energy_balance    0.0001 !%
  global_mass_balance      0.0001 !%
  fluid_energy_storage     0.005  !%
  solid_energy_storage     0.005  !%
  mass_storage             0.005  !%

[COUPLING]
  epsk          2.0  ! pcm
  epsp          1.0e-4
  eps_temp      1.0  ! C
  rlx_power     0.5
  rlx_tfuel     1.0
  rlx_den       1.0
  maxiter       20
  read_restart  restart.out

[TIAMAT]
  solver      gauss-seidel
  conserve_power_in_tiamat_transfer true
  num_subcycle_iterations_before_tiamat_ramping 0

[BISON]
!  globalparams_energy_per_fission = 3.2e11
  executioner_dt = 1.0
  executioner_end_time = 6.0e7
  executioner_num_steps = 1
  executioner_timestepper_time_dt = 1.0e3  1.0e5
```

```
  executioner_timestepper_time_t = 0      1.0e5
  mesh_clad_bot_gap_height = 1.52e-3
  fuel_pin_input_file_template = base.bison.i.template
  non_fuel_pin_input_file_template = guide_tube.moose.i.template
  ramping_time = -100.0 0.0 1.7e5
  ramping_factor = 0.0 0.0 1.0
  ramp_data_transfers = true
  mesh_type = smeared_pellet
  output_average_axial_values = true

[MPACT]
  vis_edits         none
  ray_spacing       0.05
!quad_set
  quad_type         CHEBYSHEV-GAUSS
  polars_octant     4
  azimuthals_octant 16
!iteration_control
  flux_tolerance    1e-4
  num_inners        3
  k_tolerance       1e-4
  up_scatter        2
  num_outers        100
  scattering        LTCP0
!cmfd
  cmfd              cmfd
  cmfd_solver       mgnode
  k_shift           1.5
  cmfd_num_outers   20
!2D1D
  split_TL          true
  TL_treatment      lflat
  nodal_method      nem
!  under_relax       1.0
!TH
  coupling_method   ctf_external
!parallel
  num_space         32
  num_angle         1
  num_energy        1
  num_threads       1
!xs_library
!  xs_filename       mpact47g_70s_v4.0_11032014.fmt
  xs_filename       mpact8g_70s_v4.0m0_02232015.fmt
  xs_type           ORNL
  xs_shielder       t
  subgroup_set      4
!mesh

  mesh fuel  3 1 1 / 8 8 8 8 8 8
  mesh gtube 3 1   / 8 8 8 8 8
```

C.2 Changes for 3x3 Mini-Assembly Tests

The input file for 3x3 test problems is largely the same as that for problem P6a, so we simply describe the differences from the file in C.1. The fields specified in the `[STATE]`, `[EDITS]`, `[COBRATF]`, and `[BISON]` blocks are identical for both problems. The differences between the files are as follows.

- `[CORE]` block:
  - The rated power and flow rate are scaled down to a 3x3 array, represented by the field `rated 0.53545 0.02125 ! MW, Mlbs/hr`.
- `[ASSEMBLY]` block:
  - The field `npin` is reduced to 3.
  - The `lattice` fields are reduced to a 3x3 array, given as follows:

```
lattice FUEL1
   200
     1 1

lattice LGAP1
   200
     7 7

lattice PLEN1
   200
     8 8

lattice PCAP1
   200
     9 9
```

  - The spacer grid and nozzle masses are scaled down to a 3x3 array, given by the following:

```
grid END inc  31.67 3.866
grid MID zirc 27.25 3.810

lower_nozzle  ss 6.05  194.64  ! mat, height, mass (g)
upper_nozzle  ss 8.827 194.64  ! mat, height, mass (g)
```

- `[MPACT]` block:
  - The field `ray_spacing` is changed to 0.15.
  - The fields `polars_octant` and `azimuthals_octant` are both set to 2.
  - The tolerance fields `flux_tolerance` and `k_tolerance` are reduced to 1.0e-5.
  - The `num_space` field is set to 9.
- `[COUPLING]` block:
  - The field `epsk` is increased from 2.0 to 5.0.
  - The field `epsp` is increased from 1.0e-4 to 1.0e-3.
  - The field `read_restart` is omitted.
- `[TIAMAT]` block:
  - The `num_subcycle_iterations_before_tiamat_ramping` field is changed to 1.

C.3 Changes for Single-Rod Tests

Again, the input file for single-rod tests is largely the same as that for problem P6a, so we simply describe the differences from the file in C.1. The fields specified in the `[EDITS]`, `[COBRATF]`, `[BISON]`, and `[TIAMAT]` blocks are identical for both problems. The differences between the files are as follows.

- `[STATE]` block:
  - The `boron` field is changed from 2250 to 1300.
- `[CORE]` block:
  - The rated power and flow rate are scaled to a single rod, represented by the field `rated 0.0670 0.0024 ! MW, Mlbs/hr`.
- `[ASSEMBLY]` block:
  - The field `npin` is reduced to 1.
  - The `lattice` fields are reduced to a single rod, given as follows:

```
lattice FUEL1
   1
```

```
lattice LGAP1
   7

lattice PLEN1
   8

lattice PCAP1
   9
```

  - The spacer grid and nozzle masses are scaled down to a single rod, given by the following:

```
grid END inc   4.67 3.866
grid MID zirc  3.25 3.810

lower_nozzle  ss 6.05   25.0  ! mat, height, mass (g)
upper_nozzle  ss 8.827  25.0  ! mat, height, mass (g)
```

- `[MPACT]` block:
  - The field `ray_spacing` is changed to 0.1.
  - The fields `polars_octant` and `azimuthals_octant` are both set to 2.
  - The tolerance fields `flux_tolerance` and `k_tolerance` are reduced to 1.0e-5.
  - The `num_space` field is set to 8.
- `[COUPLING]` block:
  - The field `epsk` is increased from 2.0 to 5.0.
  - The field `eps_temp` is decreased from 1.0 to 0.5.
  - The field `read_restart` is omitted.