SANDIA REPORT

SAND2005-6864
Unlimited Release
Printed November 2005

Robust Large-scale Parallel Nonlinear Solvers for Simulations

Brett W. Bader, Roger P. Pawlowski, and Tamara G. Kolda

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

SAND2005-6864
Unlimited Release
Printed November 2005

Robust Large-scale Parallel Nonlinear Solvers for Simulations

Brett W. Bader and Roger P. Pawlowski
Applied Computational Methods, Dept. 1416
Sandia National Laboratories
P.O. Box 5800, MS-0316
Albuquerque, NM 87185-0316

Tamara Kolda
Computational Sciences and Math, Dept. 8962
Sandia National Laboratories
P.O. Box 969, MS-9159
Livermore, CA 94551-9159

Abstract

This report documents research to develop robust and efficient solution techniques for solving large-scale systems of nonlinear equations. The most widely used method for solving systems of nonlinear equations is Newton's method. While much research has been devoted to augmenting Newton-based solvers (usually with globalization techniques), little has been devoted to exploring the application of different models. Our research has been directed at evaluating techniques using different models than Newton's method: a lower order model, Broyden's method, and a higher order model, the tensor method. We have developed large-scale versions of each of these models and have demonstrated their use in important applications at Sandia.

Broyden's method replaces the Jacobian with an approximation, allowing codes that cannot evaluate a Jacobian or have an inaccurate Jacobian to converge to a solution. Limited-memory methods, which have been successful in optimization, allow us to extend this approach to large-scale problems. We compare the robustness and efficiency of Newton's method, modified Newton's method, Jacobian-free Newton-Krylov method, and our limited-memory Broyden method. Comparisons are carried out for large-scale applications of fluid flow simulations and electronic circuit simulations. Results show that, in cases where the Jacobian was inaccurate or could not be computed, Broyden's method converged in some cases where Newton's method failed to converge. We identify conditions where Broyden's method can be more efficient than Newton's method.

We also present modifications to a large-scale tensor method, originally proposed by Bouaricha, for greater efficiency, better robustness, and wider applicability. Tensor methods are an alternative to Newton-based methods and are based on computing a step based on a local quadratic model rather than a linear model. The advantage of Bouaricha's method is that it can use any existing linear solver, which makes it simple to write and easily portable. However, the method usually takes twice as long to solve as Newton-GMRES on general problems because it solves two linear systems at each iteration. In this paper, we discuss modifications to Bouaricha's method for a practical implementation, including a special globalization technique and other modifications for greater efficiency. We present numerical results showing computational advantages over Newton-GMRES on some realistic problems.

We further discuss a new approach for dealing with singular (or ill-conditioned) matrices. In particular, we modify an algorithm for identifying a turning point so that an increasingly ill-conditioned Jacobian does not prevent convergence.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This report describes two classes of methods (and their extensions) for solving the nonlinear equations problem

$$\text{given } F : \mathbb{R}^n \to \mathbb{R}^n, \text{ find } x_* \in \mathbb{R}^n \text{ such that } F(x_*) = 0, \tag{1.1}$$

where it is assumed that $F(x)$ is at least once continuously differentiable. Large-scale systems of nonlinear equations defined by (1.1) arise in the simulation of many physical phenomena at Sandia, including systems produced by finite-difference or finite-element discretizations of boundary value problems for ordinary and partial differential equations. Typical examples include reacting flows, compressible flows, solid mechanics, device simulation, and circuit simulation.

Newton-based strategies are the most widely used methods for solving systems of nonlinear equations. While much research has been devoted to improving Newton-based solvers (usually with globalization techniques) [19, 38], little has been devoted to exploring the applicability of different underlying models, such as Broyden's method [57] and tensor methods [13]. Our research focuses on these alternate methods and investigates their performance on challenging problems. We present modifications to the basic algorithms that result in better performance, increased robustness, and greater ease-of-use.

Newton's method for solving (1.1) is based on creating a linear local model $M_N(x_k+d)$ of the function $F(x)$ around the current iterate $x_k \in \mathbb{R}^n$:

$$M_N(x_k+d) = F(x_k) + J(x_k)d, \tag{1.2}$$

where $d \in \mathbb{R}^n$ is the step and $J(x_k) \in \mathbb{R}^{n\times n}$ is the current Jacobian matrix, which is defined as the derivative of the residual equations with respect to the unknowns, i.e.,

$$J_{ij} = \frac{\partial F_i}{\partial x_j}.$$

A root of the local model (1.2) provides the Newton step

$$d_N = -J(x_k)^{-1}F(x_k),$$

which is used to reach the next trial point. Thus, Newton's method is defined when J_k is nonsingular, and consists of updating the current point with the Newton step,

$$x_{k+1} = x_k + d_N. \tag{1.3}$$

Because the sequence of iterates based upon (1.3) is not guaranteed to converge, a technique for modifying the step to ensure global convergence is used frequently. The two most common choices of globalization schemes are a line search procedure (which scales d_N by a fractional value) or a trust region algorithm (which chooses an optimal direction to the extent that the local model is trusted); see, e.g., [19, 45].

Methods that use direct linear solvers are impractical on large-scale problems because of the high linear algebra costs and large memory requirements. Thus, most practical approaches for solving large problems involve using an iterative method to approximately solve a local linear model using the Jacobian and then using the "inexact" step produced by the approximate solution. These methods are called "inexact" Newton methods [18]. Successively better approximations at each iteration preserve the rapid convergence behavior of Newton's method when nearing the solution. The computational savings reflected in this less expensive linear solve is usually partially offset with more outer iterations, but the overall savings is significant on large-scale problems.

Overall, Newton-based algorithms have many appealing properties such as robustness, scale invariance, and, most importantly, quadratic convergence. While Newton's method is usually the de facto standard, it is not always the best choice. This report discusses two alternate strategies.

1.1 Broyden's Method for Nonlinear Problems

In some cases, the derivative information required for a Newton-based solve is either unavailable or cannot be calculated efficiently. In these cases, alternate strategies can be more economical in terms of both implementation and execution costs.

Broyden's method [13] is an alternative to Newton's method that does not require an accurate Jacobian matrix. Broyden's method for solving (1.1) is based on creating an approximate linear local model $M_B(x_k + d)$ of the function $F(x)$ around the current iterate $x_k \in \mathbb{R}^n$:

$$M_B(x_k + d) = F(x_k) + B_k d,$$

where $B_k \in \mathbb{R}^{n \times n}$ is an approximation to the Jacobian composed of a sequence of secant approximations. Each secant approximation adds a rank-1 matrix to the Jacobian approximation so that the local model interpolates the function value at past iterates. In this way, approximate derivative information in the direction of the previous steps is included in the local model.

Since the Jacobian is approximated using information generated in previous iterates, the application need only supply B_0, the initial approximation to the Jacobian. This matrix can be as simple as the identity matrix, thus removing any requirement of a Jacobian evaluation. The closer B_0 approximates the Jacobian, the more efficient Broyden's method becomes. Due to the simple nature of the approximation matrix, Broyden's method requires very little computation time to perform a step, which makes it suitable for solving some large-scale problems.

The primary goal of our research with Broyden's method was to evaluate its performance and determine cases where this technology should be employed. We compare a limited-memory version of Broyden's method, based on [38], to various Newton-based strategies on a limited set of test problems.

1.2 Tensor Methods for Nonlinear Problems

In some cases, the Jacobian may be available, but the problem may be so nonlinear or difficult to solve with standard techniques that extra information is needed to converge. In these cases, including second-order information in the local model can produce a better step, which is the premise of tensor methods.

Tensor methods [57] solve (1.1) by including one more term from the Taylor series expansion in the local model. More precisely, tensor methods base each iteration on a simplified quadratic model of $F(x)$ such that the quadratic term is a low-rank secant approximation that augments the standard linear model. The local tensor model has the generic form

$$M_T(x_k + d) = F_k + J_k d + \tfrac{1}{2} T_k dd, \tag{1.4}$$

where $T_k \in \mathbb{R}^{n\times n\times n}$ is a low-rank tensor, which includes second-order information about $F(x)$ and is where these methods get their name. For this research we only consider a rank-1 approximation term T_k. Computational evidence in [57] suggests that a higher-rank approximation actually adds little to the computational performance of the direct method.

Tensor methods are aptly suited to target problems where the Jacobian at the root is singular or, at least, very ill-conditioned [3, 4]. Newton-based methods do not handle singular problems well because they converge linearly to the solution and, in some cases, with poor accuracy [14, 15, 16, 47]. On the other hand, tensor methods are superlinearly convergent on singular problems under mild conditions [23].

The primary goal of the tensor method research was to implement and refine large-scale tensor method algorithms to solve large problems of interest at Sandia. Various tensor strategies were developed and evaluated against Newton-based solvers.

1.3 Tensor Methods for Bifurcation Tracking

A final aspect of this research dealt with turning point identification algorithms. Because tensor methods have a faster rate of convergence than Newton's method on singular problems [23], our research focused on applying this technology to the bifurcation tracking algorithms in LOCA, particularly the turning point identification algorithm. LOCA uses a special technique called a bordering algorithm for solving this nonlinear problem by solving four intermediate subproblems involving the Jacobian matrix [55]. A linear combination of these four subproblem solutions provides the Newton step for the overall problem. However, because the algorithm is driving the Jacobian to a singularity, the four subproblems are increasingly difficult to solve and less accurate, which makes the resultant step less accurate and eventually unusable due to floating point round-off errors.

We investigated applying tensor methods to the bordered algorithm for turning point identification. Our research showed that a straightforward applications of tensor methods to the overall nonlinear problem did not appreciably improve the performance of the turning point algorithm. Nonetheless, we were able to improve the stability and accuracy of the turning point algorithm with a novel approach to solving ill-conditioned linear systems using an idea from tensor methods.

1.4 Organization and Notation

The organization of this report is broken into three main chapters, following the three distinct topics in our research. Each chapter is intended to be a stand-alone paper. Chapter 2 presents our research on Broyden's method. Chapter 3 discusses our research on tensor methods. Chapter 4 discusses the tensor method techniques applied to bifurcation identification algorithms. Each chapter contains its own summary and discussion, and Chapter 5 provides a broad summary of the overall project.

Throughout this report, the subscript k refers to the iteration number of the nonlinear iteration sequence. We denote the Jacobian $F'(x)$ by $J(x)$ and frequently abbreviate $J(x_k)$ as J_k and $F(x_k)$ as F_k. In keeping with the notation of Taylor series approximations for the value $F(x_k + d)$, we denote local models of a function about a point x_k as $M(x_k + d)$ and have included a subscript N, B, or T on M to specify the Newton, Broyden, or tensor model, respectively. In addition, the respective steps that solve these local models are denoted d_N, d_B, and d_T.

Chapter 2

Broyden Methods

In this chapter, we describe and evaluate Broyden's method for the solution of nonlinear equations, an alternative to Newton-based methods that does not require Jacobian information (but will make efficient use of any approximation to the Jacobian that is provided). This chapter presents Broyden's method and compares the algorithm with Newton's method. Examples from application codes used at Sandia are shown. The results demonstrate situations where Broyden's method is preferable to Newton's method.

2.1 Theory

The most prevalent strategy for solving nonlinear equations is to apply Newton's method to iteratively solve a local linear model until convergence [19]. The local linear model is defined as:

$$M(x_k + d) = F_k + J_k\, d, \tag{2.1}$$

where k is the iterate number, x_k is the current solution iterate (the approximation to the solution at iteration k), $J_k \in \boldsymbol{R}^{n\times n}$ is the Jacobian matrix evaluated at x_k, and $F_k = F(x_k)$ is the residual evaluated at x_k. The Jacobian matrix is defined as the derivative of the residual equations with respect to the unknowns, i.e.,

$$J_{ij} = \frac{\partial F_i}{\partial x_j}. \tag{2.2}$$

The next iterate is defined as

$$x_{k+1} = x_k + d_k,$$

where d_k is the solution to 2.1.

Some form of globalization may be needed to guarantee convergence. If a line search is used for globalization, then

$$x_{k+1} = x_k + \lambda_k d_k,$$

where λ_k is the line search parameter. The iterations progress until the approximate solution satisfies some convergence criteria. If a line search is used for globalization, the algorithm is as follows:

Algorithm GN [20]: Globalized Newton Method

```
LET $x_0$ BE GIVEN.
FOR $k = 0, 1, \ldots$ (UNTIL CONVERGENCE) DO:
    SOLVE: $J_k d_k = -F_k$ FOR $d_k$
    COMPUTE: $\lambda_k$ VIA A LINE SEARCH
    SET $x_{k+1} = x_k + \lambda_k d_k$.
```

If there is no line search, then $\lambda_k = 1$ for all k.

The iteration sequence above can be rewritten as follows:

$$x_{k+1} = x_k - \lambda_k J_k^{-1} F_k \tag{2.3}$$

Each nonlinear iteration requires solving a linear system. Difficulties arise not only in inverting or preconditioning the Jacobian, but also in computing the Jacobian. In fact, it is not always possible to compute the Jacobian directly because of the type of discretization scheme or because of discontinuities in derivative evaluations (e.g., due to table look-ups for material properties, third-party library functions, etc.). For example, the compressible flow code Premo [1] uses a 2nd order accurate finite volume scheme with Roe flux limiting that makes evaluating the Jacobian extremely complex and error-prone. Instead, colored finite difference techniques [25] and automatic differentiation [30] are used to evaluate the Jacobian. Finite differencing can be slow, especially if the connectivity graph is not supplied by the user. Additionally, the Jacobian may be inexact due to residuals that involve discontinuous derivatives (i.e., table look-ups) or difficult terms being ignored. Automatic differentiation is a promising technology for evaluating the Jacobian, but no universal tool has been developed. In some cases, Jacobians for FORTRAN and C code can be generated using source transformation software such as ADIFOR [7] and ADIC [8]. C++, on the other hand, requires an invasive templating of the scalar type in all residual fill algorithms. This can be difficult to use especially when programming languages are mixed due to third party libraries.

Instead of using a linear model as in 2.3, we propose to use an alternative model introduced by Broyden [13]. The iteration sequence in 2.3 is then replaced by:

$$x_{k+1} = x_k - \lambda_k B_k^{-1} F_k \tag{2.4}$$

where $B_k \approx J(x_k)$ is an approximation to the Jacobian based on a least-change secant update. Our implementation follows the limited-memory Broyden algorithm in Kelley [37, §8,3,2].

Recall that the *search direction* is denoted by $d_k = -B_k^{-1}F_k$. Let s_k denote the kth step; i.e.,

$$s_k = x_{k+1} - x_k = -\lambda_k B_k^{-1} F_k = \lambda_k d_k. \tag{2.5}$$

If no line search is used (so that $\lambda_k = 1$ for all k), then $s_k = d_k$.

It is interesting to note that the use of a line search can be problematic since d_k is not guaranteed to be a descent direction. Kelley [38] recommends that if a line search with Broyden's method fails, one should look to better preconditioning strategies or move to a Newton-Krylov scheme. In our computations, if the Broyden method fails during a line search, we recompute the Jacobian estimate and attempt another solve.

The Broyden update is a *secant update*, i.e., it obeys the secant condition:

$$B_{k+1}s_k = y_k, \tag{2.6}$$

where $y_k = F_{k+1} - F_k$. The Broyden update represents the least change from the previous matrix, B_k, that satisfies the secant condition. It is defined as

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)s_k^T}{\|s_k\|_2^2}. \tag{2.7}$$

A more convenient expression for B_k can be derived by observing that

$$y_k - B_k s_k = (F_{k+1} - F_k) + \lambda_k F_k = F_{k+1} - (1-\lambda_k)F_k, \tag{2.8}$$

and so we can rewrite B_k as

$$B_{k+1} = B_k + \frac{(F_{k+1} - (1-\lambda_k)F_k)\, s_k^T}{\|s_k\|_2^2}. \tag{2.9}$$

This recursive formula means that B_k can be formed implicitly, as we discuss in more detail in the sections that follow. Note that the updates are based on the residual evaluations (i.e., values of F), and so there is no need to compute a Jacobian.

2.1.1 The Choice of B_0 for Broyden's Method

The following is the analogue of Lemma 7.3.1 in [37]; here it is extended to the case with a line search parameter.

Lemma 2.1.1 *Let $\{x_k, B_k\}$ be the Broyden sequence generated by (F, x_0, B_0), and let $\{z_k, C_k\}$ be the Broyden sequence generated by $(B_0^{-1}F, x_0, I)$. Then*

$$x_k = z_k \quad \textit{and} \quad B_k = B_0 C_k. \tag{2.10}$$

(We assume the step lengths, λ_k, are the same for each sequence.)

This is a proof by induction. The case for $k = 0$ is trivial. Assume the claim holds for k. Observe that,

$$\begin{aligned} x_{k+1} &= x_k - \lambda_k B_k^{-1} F_k \\ &= z_k - \lambda_k (B_0 C_k)^{-1} F_k \\ &= z_k - \lambda_k C_k^{-1} (B_0^{-1} F_k) \\ &= z_{k+1}, \end{aligned}$$

and

$$\begin{aligned} B_{k+1} &= B_k + \frac{(F_{k+1} - (1-\lambda_k) F_k)\, s_k^T}{\|s_k\|_2^2} \\ &= B_0 C_k + B_0 \frac{\left(B_0^{-1} F_{k+1} - (1-\lambda_k) B_0^{-1} F_k\right) s_k^T}{\|s_k\|_2^2} \\ &= B_0 C_{k+1}. \end{aligned}$$

Q.E.D.

An important consequence of this lemma is that we can assume that $B_0 = I$ for all the analysis that follows.

2.1.2 An Implicit Representation of the Broyden matrix

The Broyden matrix can be stored implicitly using only $2k$ vectors plus B_0; see, e.g., Kelley [37] whose derivation is as follows.

We can express the Broyden update as a function of the previous iterate's Broyden matrix and a pair of vectors:

$$B_{k+1} = B_k + u_k v_k^T, \tag{2.11}$$

where

$$u_k = \frac{F_{k+1} - (1-\lambda_k) F_k}{\|s_k\|_2} \quad \text{and} \quad v_k = \frac{s_k}{\|s_k\|_2}. \tag{2.12}$$

We can now rearrange equation 2.11 into a more usable form that can be quite efficient. The Sherman-Woodbury-Morrison formula [27] says

$$B_{k+1} = \left(I - \frac{(B_k^{-1} u_k) v_k^T}{1 + v_k B_k^{-1} u_k} \right) B_k^{-1}. \tag{2.13}$$

We define

$$w_k = \frac{B_k^{-1} u_k}{1 + v_k B_k^{-1} u_k} = \frac{B_k^{-1} (F_{k+1} - (1-\lambda_k) F_k)}{\|s_k\|_2 + v_k^T B_k^{-1} (F_{k+1} - (1-\lambda_k) F_k)}. \tag{2.14}$$

Then

$$B_{k+1}^{-1} = (I - w_k v_k^T) B_k^{-1} = \prod_{j=0}^{k} (I - w_j v_j^T). \tag{2.15}$$

Thus, the inverse of the Broyden matrix can be stored using only $2k$ vectors plus the initial matrix inverse B_0^{-1}.

2.1.3 A More Efficient Implicit Representation of the Broyden Matrix

Kelley [37] observed that the Broyden matrix can be stored using only $k+1$ vectors. His derivation is as follows.

Define

$$z_k = B_k^{-1}(F_{k+1} - (1-\lambda_k)F_k) \quad \text{and} \quad \alpha_k = \|s_k\|_2 + v_k^T z_{k+1}, \tag{2.16}$$

so that

$$w_k = \frac{z_k}{\alpha_k}. \tag{2.17}$$

Then

$$\begin{aligned}
d_{k+1} &= -B_{k+1}^{-1} F_{k+1} \\
&= -\left(I - w_k v_k^T\right) B_k^{-1} F_{k+1} \\
&= -\left(I - w_k v_k^T\right) \left(B_k^{-1} F_{k+1} - (1-\lambda_k) B_k^{-1} F_k + (1-\lambda_k) B_k^{-1} F_k\right) \\
&= -\left(I - \frac{z_k}{\alpha_k} v_k^T\right) \left(z_k + (1-\lambda_k^{-1}) s_k\right) \\
&= -\left(1 - \frac{v_k^T z_k}{\alpha_k} - \left(\frac{\lambda_k^{-1} - 1}{\alpha_k}\right) v_k^T s_k\right) z_k - (\lambda_k^{-1} - 1) s_k \\
&= -\left(1 - \frac{v_k^T z_k + (\lambda_k^{-1} - 1)\|s_k\|_2}{\alpha_k}\right) z_k - (\lambda_k^{-1} - 1) s_k \\
&= -\left(1 - \frac{v_k^T z_k + (\lambda_k^{-1} - 1)\|s_k\|_2 + \lambda_k^{-1}\|s_k\|_2}{\alpha_k} + \frac{\lambda_k^{-1}\|s_k\|_2}{\alpha_k}\right) z_k - (\lambda_k^{-1} - 1) s_k \\
&= -\lambda_k^{-1}\|s_k\|_2 \frac{z_k}{\alpha_k} - (\lambda_k^{-1} - 1) s_k \\
&= -\lambda_k^{-1}\|s_k\|_2 w_k - (\lambda_k^{-1} - 1) s_k.
\end{aligned}$$

Solving for w_k, we have

$$\begin{aligned}
w_k &= \frac{-\lambda_k}{\|s_k\|_2}\left(d_{k+1} + (1-\lambda_k^{-1})s_k\right) \\
&= -\lambda_k \frac{d_{k+1}}{\|s_k\|_2} + (1-\lambda_k)\frac{s_k}{\|s_k\|_2} \\
&= \frac{-\lambda_k}{\lambda_{k+1}}\frac{s_{k+1}}{\|s_k\|_2} + (1-\lambda_k)\frac{s_k}{\|s_k\|_2}.
\end{aligned}$$

Substituting w_k in the expression for d_{k+1} yields

$$\begin{aligned}
d_{k+1} &= -B_{k+1}^{-1}F_{k+1} \\
&= -(I - w_k v_k^T)B_k^{-1}F_{k+1} \\
&= -\left(I + \lambda_k \frac{d_{k+1}s_k^T}{\|s_k\|_2^2} - (1-\lambda_k)\frac{s_k s_k^T}{\|s_k\|_2^2}\right)B_k^{-1}F_{k+1} \\
&= -B_k^{-1}F_{k+1} - \lambda_k \frac{s_k^T B_k^{-1}F_{k+1}}{\|s_k\|_2^2} d_{k+1} + (1-\lambda_k)\frac{s_k^T B_k^{-1}F_{k+1}}{\|s_k\|_2^2} s_k.
\end{aligned}$$

Solving for d_{k+1}, we get

$$\begin{aligned}
d_{k+1} &= \frac{-B_k^{-1}F_{k+1} + (1-\lambda_k)\frac{s_k^T B_k^{-1}F_{k+1}}{\|s_k\|_2^2} s_k}{1 + \lambda_k \frac{s_k^T B_k^{-1}F_{k+1}}{\|s_k\|_2^2}} \\
&= -\frac{\|s_k\|_2^2 B_k^{-1}F_{k+1} + (\lambda_k - 1)s_k^T B_k^{-1}F_{k+1}s_k}{\|s_k\|_2^2 + \lambda_k s_k^T B_k^{-1}F_{k+1}} \\
&= \frac{\|s_k\|_2^2\, p_k + (\lambda_k - 1)s_k^T p_k\, s_k}{\|s_k\|_2^2 - \lambda_k s_k^T p_k}
\end{aligned}$$

where $p_k = -B_k F_{k+1}$. The value of p_k is calculated recursively as follows. Let

$$p_k^0 = -F_{k+1} \quad \text{and} \quad p_k^{j+1} = (I - w_j v_j^T)p_k^j \text{ for } j = 0,\ldots,k-1. \tag{2.18}$$

Then $p_k = p_k^k = B_k^{-1}F_{k+1}$. The formula for p_k^{j+1} can be rewritten as

$$p_k^{j+1} = p_k^j + \frac{\lambda_j}{\lambda_{j+1}}\,\frac{s_j^T p_k^j}{\|s_j\|_2^2}\, s_{j+1} + (\lambda_j - 1)\,\frac{s_j^T p_k^j}{\|s_j\|_2^2}\, s_j, \tag{2.19}$$

for $j = 0,\ldots,k-1$.

The critical aspect in Broyden's method is that, as the iterations in k progress, we build up a set of rank-1 updates, and in combination with the initial Broyden matrix inverse,

B_0^{-1}, we get an estimate of the action of the current Jacobian without having to evaluate it directly at each iteration. We are in effect implicitly storing the inverse of the Jacobian at each iteration. This fact allows the Broyden method to be very efficient since we can reuse the factorization of B_0 for direct linear solves or we can reuse the preconditioner in the case of iterative linear solves. Additionally, the Broyden method makes no assumptions on how to solve the inverse of the Broyden matrix. This can be critical in certain applications. We will comment more on this in section 2.3.1.1.

2.2 Implementation Issues

During the assessment of Broyden's method, minor issues were found to have a critical impact on performance. The implementation had to be adjusted in order to address both robustness and efficiency issues. The key modification details are described in this section.

The first issue is the choice of the initial guess for the Broyden matrix, B_0. One choice is to simply use the identity matrix as the initial guess. This worked well for simple test problems with a small number of unknowns, but did not perform very well when scaled to larger problems. Instead, we found that an estimate of the Jacobian (hopefully inexpensive to compute and invert) worked best.

A second issue was found during the efficiency studies. Broyden's method was observed to take many more iterations than Newton's method to achieve the same reduction in the residual norm, $\|F\|$. This behavior is to be expected because Newton's method is updating derivative information at each iteration. In order to make Broyden competitive with Newton's method, we added a restart procedure.

There exist many choices for handling restarts. We chose to use the convergence rate as the trigger for initiating a restart of the broyden algorithm. The convergence rate, α_k, is defined as:

$$\alpha_k = \frac{\|F_k\|}{\|F_{k-1}\|} \tag{2.20}$$

where k is the current iterate. At the end of an iteration, if the convergence rate α_k was larger than the requested value, the method was restarted. A restart consisted of erasing the update vectors and recomputing B_0 using the current solution vector. Recall that B_0 is the approximate Jacobian supplied by the user. It may not require an update if it does not depend on the solution vector. Based on our results, restating when the convergence rate was greater than or equal to 0.1 had a dramatic improvement on solution times while requiring few additional evaluations of B_0. This result is based on using $B_0 = J_k$ and will be discussed in more detail in the following sections.

The augmentation of Broyden's method with a restart capability makes this method look very similar to the modified Newton method [37]. Modified Newton reuses the same Jacobian, J_0 at each iterate:

$$x_{k+1} = x_k - \lambda_k J_0^{-1} F_k \tag{2.21}$$

The difference between modified Newton and Broyden's method is that Broyden's method stores rank-1 updates during the iteration sequence and does not require that J_0 be exact. In the results section, we include results for the modified Newton method in order to isolate the effects of the rank-1 updates.

The use of Broyden's method required modifications to the convergence criteria used in some of the test codes. For example, in the reacting flow code MPSalsa, convergence was evaluated based solely on a weighted root mean square (WRMS) norm [11]:

$$||\delta x_k||_{wrms} < \text{tolerance} \tag{2.22}$$

where

$$||\delta x_k||_{wrms} \equiv C \sqrt{\frac{1}{N} \sum_{i=1}^{N} \left(\frac{(x_k)_i - (x_{k-1})_i}{RTOL|(x_{k-1})_i| + ATOL_i} \right)^2} \tag{2.23}$$

This is a dangerous test when used by itself since convergence is only determined based on the change between iterates, $(x_k)_i - (x_{k-1})_i$. In our initial tests, Broyden's method was observed to trigger premature convergence while the far from the actual solution. This is because the updates generated by Broyden's method was observed in general to be much smaller than solution changes between iterates for Newton's method. These small changes in x result in an artificially small WRMS norm that triggers convergence too soon. To remedy this situation, an additional convergence test was added that forced the norm of the residual, $||F_k||$, to be less than a specified tolerance. This points to the fact that one can not blindly replace Newton's method with Broyden's method. The algorithms, while very similar, generate different behavior in the order of magnitude of the solution update.

2.3 Results

Broyden's method was tested on various applications of interest to Sandia, including reacting flow and circuit simulation. In this section, we report on the performance for these applications.

The first section, 2.3.1, is an extensive analysis of the Broyden algorithm with comparisons to Newton, modified Newton, and Jacobian-Free Newton-Krylov methods for a fluid flow simulation of a counterflow jet reactor [53]. It demonstrates the cases where Broyden's method can have a large impact on robustness and performance. Following the CJR results are two sections that give a broad analysis of the performance of Broyden's method on some benchmark problems in fluid flow and electrical circuit simulation.

2.3.1 Counterflow Jet Reactor

Counterflow jet reactors (CJRs) have a wide variety of industrial applications including diffusion flame analysis for combustion [58], nanoparticle synthesis of semiconductors [56],

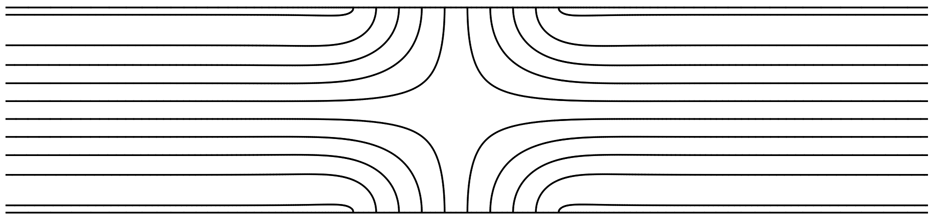

Figure 2.1. Streamlines for the flow pattern in a CJR. Reynolds number is 10.

and blood flow analysis [32]. Efficient and robust simulation of a CJR can significantly reduce difficulties in interpreting experimental observations [44]. The CJR problem is ideal for use in testing the Broyden algorithm because the nonlinearity is controlled by a single parameter, the Reynolds number (*Re*), and the Jacobian is available analytically. By controlling the nonlinearity through the Reynolds number, the problem difficulty can be controlled. By having an exact Jacobian, we can perform comparisons and isolate the errors that come from inexact or incomplete Jacobians. Finally, we use this problem since the parameter space and solution modes have been mapped out in detail [44].

The CJR consists of two co-axially aligned counterflowing jets of fluid. The jets collide and form a stagnation flow pattern. Streamlines for a typical CJR are depicted in Figure 2.1. The jets are aligned vertically, injecting fluid from the top and bottom of the domain. The stagnation flow forms in the center of the reactor and the fluid exits horizontally, confined between exit walls. The domain is two-dimensional using cartesian coordinates. There are 10,800 elements, 11,041 nodes, and 33,123 unknowns. The simulations were run on 16 1.0 GHz Pentium processors of a Linux cluster.

The fluid flow in the CJR model is described by the Navier-Stokes equations. We assume the flow is a laminar, isothermal, incompressible, Newtonian fluid. The governing transport PDEs used in our experiments are given below in dimensionless residual form.

$$\text{Conservation of mass:} \qquad \frac{\partial \rho}{\partial t} + \nabla \cdot \mathbf{u} = 0 \tag{2.24}$$

$$\text{Momentum transport:} \qquad \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} + \nabla P - \frac{1}{Re} \nabla^2 \mathbf{u} = 0 \tag{2.25}$$

The unknowns in dimensionless form are: $\mathbf{u}$, the fluid velocity vector, and P, the hydrodynamic pressure. The dimensionless Reynolds number is Re, defined as $Re = \frac{\rho D u_o}{\mu}$, where ρ is the fluid density, D is the spacing between the inlet jets, u_o is the inlet jet velocity, and μ is the fluid viscosity. The material properties are assumed constant since the system is isothermal.

To obtain an algebraic system of equations $F(x) = 0$, a stabilized-Galerkin finite element formulation is used to discretize equations 2.24 and 2.25. The stabilized method allows equal order interpolation of velocity and pressure and also provides stabilization of the convection operators to limit oscillations due to high grid Reynolds number effects. This formulation is described in [61] and follows the work of [35] and [63].

The equations are discretized and evaluated using the reacting flow code MPSalsa [60]. This code was developed by Sandia to simulate two- and three-dimensional reacting flows at low Mach numbers (incompressible flow regime). MPSalsa solves both steady-state and transient problems using a fully coupled Newton's method. The nonlinear solvers (Newton's method, modified Newton, Broyden's method, and Jacobian-free Newton-Krylov) were supplied by the NOX nonlinear solver package [41]. NOX is part of the Trilinos project [21, 34], a generic solver framework designed to meet the numerical analysis needs of Sandia applications.

Results concerning the CJR are divided into two sections based on the particular issue that the study addresses. Section 2.3.1.1 discusses the robustness of Broyden's method when applied to problems with inaccurate Jacobians. Section 2.3.1.2 discusses efficiency and performance issues where Broyden's method is compared to Newton and modified Newton techniques.

2.3.1.1 Robustness Under Inaccurate Jacobians

In this section, we compare the performance of Broyden's method to Newton's method to demonstrate how a lack of information in the Jacobian affects robustness. In chapter 2.1, we discussed some of the difficulties encountered in evaluating the Jacobians required by Newton's method. If the Jacobian is not accurate, Newton's method may have a slow convergence rate or fail to converge at all. There are two options: switch to a Jacobian-free Newton-Krylov (JFNK) approach [40] or use our Broyden implementation.

We now describe the JFNK approach. The Krylov-based methods only require matrix-vector products to perform a linear solve. If an explicit evaluation of the Jacobian is either inaccurate or unavailable, the matrix-vector products can be estimated by directional derivatives:

$$Jv = \frac{F(x + \eta v) - F(x)}{\eta} \tag{2.26}$$

where η is a scalar perturbation factor. Equation 2.26 forms the basis of the JFNK algorithm. This allows JFNK to accurately solve the Newton system without directly assembling a Jacobian.

There are drawbacks in this approach. The perturbation factor, η, in equation 2.26 is a scalar value and for problems where the solution vector has a large range of magnitudes, this value may not sufficiently perturb all the unknowns to get accurate sensitivities. This causes difficulties in obtaining the specified convergence tolerance and thus results in less accurate or non-convergent predictions. For more information on choosing perturbation parameters, we refer readers to [38]. Another difficulty is that a preconditioner is typically required for efficient linear solves using iterative solution techniques. This requires an estimate of individual Jacobian entries that may not be feasible. Finally, JFNK methods require that the residual equations be re-evaluated at each inner iteration of the GMRES linear solve to evaluate $F(x+\eta v)$. This can be prohibitively expensive depending on both the application's residual evaluation and the preconditioner efficiency. Despite these disadvantages, JFNK is very robust in solving nonlinear equations and is used quite often.

The Broyden method is an alternative strategy that is less restrictive than JFNK but not as efficient. Similar to JFNK methods, Broyden updates incorporate Jacobian information as the nonlinear iterations progress, and do not require explicit Jacobians. The unique aspect that separates Broyden's method from JFNK is that it makes no assumptions on how to solve the linear systems. JFNK restricts the linear solve to be an iterative Krylov method. In some problems, such as circuit modeling, we have observed run time efficiencies increased by an order of magnitude by switching from iterative linear solvers to direct solvers. This is due to the small size of specific problems, on the order of thousands of unknowns or less. While Xyce did not require JFNK or Broyden since it provides analytic Jacobians, other small-scale codes where the Jacobian is unavailable or inexact codes could benefit from the use of Broyden over JFNK.

The results shown here are for a CJR model with a Reynolds number of 10. This means that the advection term ($\mathbf{u}\cdot\nabla\mathbf{u}$) is roughly ten times larger than the diffusive term ($\mu\nabla^2\mathbf{u}$) in equation 2.25. To simulate a code with missing Jacobian information, we completely remove all derivatives in the Jacobian that correspond to the advection term. The residual equations are exact and evaluate all terms, but the Jacobian dependencies for the advection term have been dropped. Solves were performed using Newton's method, two variations on Broyden's method, and the JFNK method.

Figure 2.2 shows the L-2 norm of the residual as a function of the nonlinear iterate. Convergence was based on the $\|F\| \leq 10^{-6}$ and WRMS Norm ≤ 1.0. As the iteration sequence progresses, the missing information in the Jacobian causes Newton's method to diverge and fail. Broyden's method with restarts also fails. This occurs because the restarts throw away the accumulated dependency information on the convection term that is being aggregated only through the rank-1 updates. Therefore, the rank-1 updates are critical in obtaining convergence since they are the only way to incorporate missing Jacobian information. In the figure, the restart occurs after iteration 12 because the convergence rate is above the required value of 1.0. By disabling the restart capability, eventually the Broyden method accrues enough information on the missing Jacobian term to attain convergence. This demonstrates that Broyden's methods can be more robust than Newton's method when inaccurate Jacobians are present. The JFNK method also achieves convergence since the directional derivative used in the GMRES algorithm is computed using finite differencing of the residual equations. It too is able to capture the missing Jacobian information.

There are some important points to note in comparing the Broyden's method to JFNK

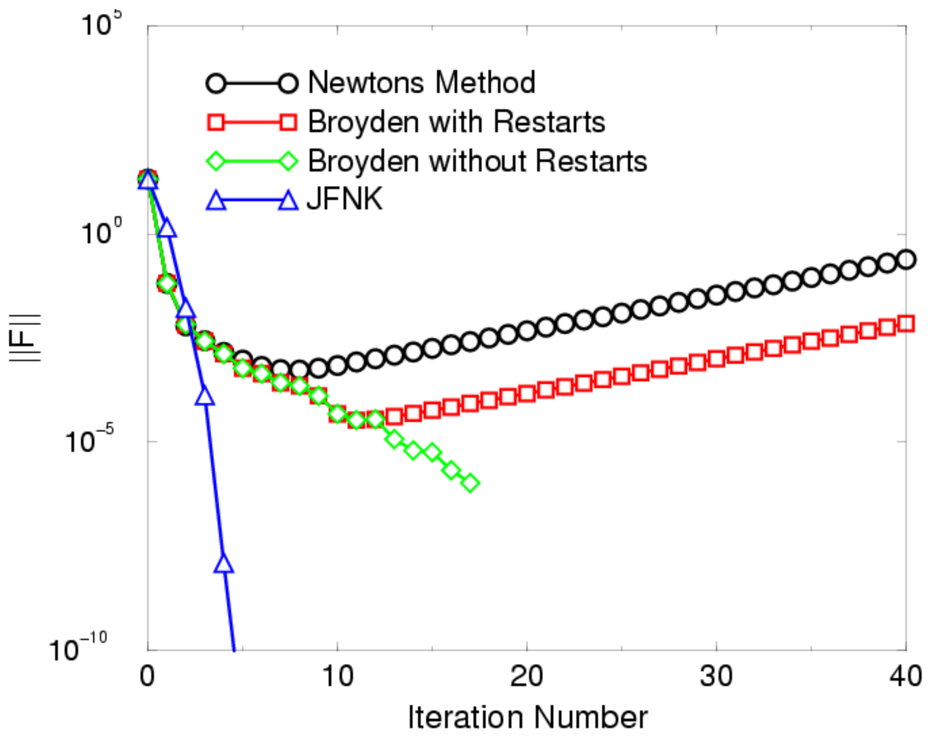

Figure 2.2. Plot of the residual norm as a function of nonlinear iteration number.

using Figure 2.2. First, we observe that JFNK has over-solved this problem by one nonlinear iteration. This is the result of the convergence criteria in the application. In particular this is caused by the WRMS norm stopping criteria. The change in the solution vector between iterates is still large, even though the tolerance for $||F||$ was met. This results in one extra nonlinear iteration in the JFNK algorithm. The second point to observe is that the convergence for JFNK is superlinear while Broyden's method exhibits linear convergence. Broyden's method takes about three times as many nonlinear iterations to converge as does JFNK.

Figure 2.2 does not give enough information to determine which method, Broyden or JFNK, should be used for a given application. The reason is that the total number of nonlinear iterations does not directly correlate to the efficiency of the code since the linear solves are much more expensive in JFNK. In table 2.1 we show the performance characteristics for the two methods that converged, Broyden without restarts and JFNK. NS is the total number of nonlinear iterations, F is number of residual (function) evaluations, J is the number of Jacobian evaluations, LSI is the number of linear solve iterations (i.e., the number of GMRES iterations), F Time is the total time taken to evaluate the residuals, J Time is the total time to evaluate the Jacobians, LS Time is the total time spent in the linear solve routines, and Total Time is the total run time. All times are reported in seconds. The

Method	NS	F	J	LSI	F Time	J Time	LS Time	Total Time
Broyden	17	18	1	1586	0.9	0.1	37.7	40.0
JFNK	5	497	5	471	23.8	0.5	32.3	36.8

Table 2.1. A comparison of robust methods under inaccurate Jacobians.

total run times are comparable to each other, with a slight edge going to JFNK. Assuming an iterative linear solver (as used in this study), Broyden will take many more nonlinear steps, resulting in a number of GMRES iterations proportional to the number of nonlinear steps. This factor is what slows down Broyden's method. This is balanced in JFNK by the number of residual evaluations per nonlinear step. Since JFNK uses directional derivatives in the matrix-vector multiply, the equations will be evaluated at least as many times as the number of GMRES iterations are required in the linear solve. The choice of methods is determined by how efficient the residual fills per nonlinear solve are compared to the extra number of nonlinear steps required by Broyden's method. Costly residuals favor Broyden while a large number of nonlinear steps favor JFNK.

Another issue is the degree of nonlinearity in a problem. The Reynolds number used in this study is relatively small. For larger Reynolds numbers, the difficulty in solving the nonlinear system may require a globalization or homotopy technique similar to those used in Newton's method. We did not pursue the use of globalization techniques here, but globalizations used in Newton-based methods can be trivially extended to the Broyden method.

2.3.1.2 Efficiency Comparisons

In many applications, Newton's method can be inefficient. This usually occurs when the equations are only mildly nonlinear; i.e., the Jacobian values do not vary widely from iteration to iteration. In such a situation, it may be more efficient to reuse the Jacobian in successive iterations rather than re-evaluating it at each iteration. The Jacobian need only be re-evaluated when the convergence rate (equation 2.20) stalls. This procedure is known as a modified Newton method [37]. The Broyden method performs the same operations as modified Newton, but has an advantage in that it additionally stores rank-1 updates during the iteration sequence to approximate the current Jacobian inverse as opposed to the original Jacobian inverse. This can give a better approximation to the Jacobian than just blindly reusing the original Jacobian. The updates may increase the convergence rate and, thus, boost efficiency.

For these demonstrations, the CJR test problem will be used again, but in both steady-

Run	Method	Max Rate	S/F	NS	F	J	F Time	J Time	Prec Time	LS Time	Total Time
1	Newton	NA	S	5	6	5	0.3	0.6	2.3	11.7	16.0
2	Mod. Newton	0.1	S	7	8	3	0.4	0.3	1.4	15.8	19.1
3	Broyden	0.1	S	7	8	3	0.4	0.3	1.4	15.9	19.6
4	Mod. Newton	1.0	S	14	15	2	0.7	0.2	0.9	31.7	34.9
5	Broyden	1.0	S	21	22	4	1.1	0.4	1.8	53.1	57.9

Table 2.2. Steady-state performance characteristics.

state and transient modes. The runs will use the full analytic Jacobian including all inter-equation dependencies. We will compare Newton's method, Broyden's method and the modified Newton method. For the Broyden and modified Newton runs, we performed two sets of simulations using different values of the convergence rate, α_k (section 2.2). The values used are 0.1 and 1.0. The smaller the value, the more often restarts occur, making the methods act more like the Newton method in terms of performance.

Table 2.2 shows the steady-state problem results at a Reynolds number of 10. Max Rate is the largest value of convergence rate allowed before a restart is performed. S/F stands for successful convergence or failure of the simulation. NS, F, and J are the number of nonlinear steps, the number of residual (function) evaluations, and the number of Jacobian evaluations, respectively. F Time, J Time, Prec Time, LS Time, and Total Time are the time taken for residual evaluations, time taken for all Jacobian evaluations, time taken for all preconditioner factorizations, time taken for the linear solve (excluding preconditioner factorization time), and the total simulation time, respectively. All times are reported in seconds.

We observed that Newton's method performs the most efficiently while Broyden's method with a convergence rate restart value of 1.0 performed the least. This can be attributed to the application code's efficient implementation of the Jacobian evaluation and fast preconditioners. Broyden's method is expected to out-perform Newton's method when the Jacobian evaluations are expensive. In the case of MPSalsa, the Jacobian evaluations were extremely efficient and did not factor into the overall run time. In our experience with MPSalsa, we found that the residual and Jacobian fill times were less than 10% of the total run time. A similar situation exists for the preconditioner. In Table 2.2 the time spent in the preconditioner is also negligible to the overall time. Our conclusion is that in terms of efficiency, MPSalsa should be using Newton's method for steady-state solves.

Transient simulations were also performed for the CJR problem. Transient problems are different from steady-state problems in that the nonlinear solves have a good initial guess from the last time step and are much more robust. This allows for good efficiency gains through reuse of the Jacobian and preconditioner because the initial values should not change appreciably due to the good initial guess (unless very large time steps are used).

Run	Method	Max Rate	S/F	NS	F	J	F Time	J Time	Prec Time	LS Time	Total Time
1	Newton	NA	S	52	72	52	3.8	6.0	23.8	111.9	151.9
2	Mod. Newton	0.1	S	46	66	22	3.5	2.5	10.1	89.3	110.1
3	Broyden	0.1	S	46	66	22	3.5	2.6	10.0	87.8	109.8
4	Mod. Newton	1.0	S	46	66	20	3.5	2.3	9.1	86.4	107.2
5	Broyden	1.0	S	49	69	20	3.7	2.3	9.0	95.6	117.0

Table 2.3. Transient performance characteristics.

A no-flow solution (zero initial guess) was used as the starting solution in this test. The flow was turned on and the system was tracked for 0.1 seconds which corresponded to about 20 time steps with a constant step size. Table 2.3 shows the results with the true Jacobian times in MPsalsa (Jacobians were not artificially inflated). In transient problems, the time step size controls the amount of nonlinearity in the solution. In this case, the Broyden and modified Newton methods perform much better than Newton. The efficiency is based on the fact the each time step is using only one Jacobian evaluation and then reuses the Jacobian and preconditioner for each successive linear solve. The difference in efficiency between the Broyden and modified Newton methods was negligible. For transient runs, the rank-1 updates in Broyden's method do not have enough iterations to accumulate information to speed up convergence.

2.3.2 Thermal Convection

This section compares Newton's method with Broyden's method on a benchmark problem for fluid flow: thermal convection (or buoyancy driven) flow of a fluid in a differentially heated square cavity in the presence of gravity. The thermal convection problem is used in many demonstration studies since it has a well defined problem space. MPSalsa simulations on this problem are documented in [62], [45] and [46]. These resources provide a more thorough description and additional references.

The governing PDEs are the incompressible Navier-Stokes equations, given below. The unknown quantities in these equations are the fluid velocity vector ($\mathbf{u}$), the hydrodynamic pressure (P), and the temperature (T).

$$\text{Conservation of mass:} \quad \nabla \cdot \mathbf{u} = 0 \tag{2.27}$$

$$\text{Momentum transport:} \quad \rho\, \mathbf{u} \cdot \nabla \mathbf{u} - \nabla \cdot \mathbf{T} - \rho \mathbf{g} = 0 \tag{2.28}$$

$$\text{Energy transport:} \quad \rho C_p \mathbf{u} \cdot \nabla T + \nabla \cdot \mathbf{q} = 0 \tag{2.29}$$

In these equations, $\mathbf{g}$ is the gravity vector, and ρ and C_p are the density and specific heat

Run	Method	Max Rate	S/F	NS	F	J	F Time	J Time	LS Time	Prec Time	Total Time
1	Newton	NA	S	152	202	152	14.3	23.2	227.0	119.5	395.4
2	Mod. Newton	1.0	S	104	154	50	11.5	7.8	155.3	40.7	228.0
3	Broyden	1.0	S	106	156	52	11.3	8.0	157.7	41.9	231.6
4	Mod. Newton	0.1	S	104	154	52	11.2	8.1	154.8	41.9	228.7
5	Broyden	0.1	S	104	154	52	11.2	8.1	155.7	41.9	230.0

Table 2.4. Transient thermal convection in a box.

at constant pressure of the bulk fluid, respectively. The constitutive equations for the stress tensor **T** and heat flux **q** are

$$\begin{aligned} \mathbf{T} &= -P\mathbf{I} + \mu(\nabla\mathbf{u} + \nabla\mathbf{u}^T), \\ \mathbf{q} &= -\kappa\nabla T, \end{aligned}$$

where μ is the dynamic viscosity and κ is the thermal conductivity of the fluid.

The left and right walls of the box are kept at constant temperatures (one cold and the other hot) and the top and bottom walls are insulated. This corresponds to the following Dirichlet and Neumann boundary conditions:

$$T = T_{cold},\ \mathbf{u} = 0 \quad \text{at} \quad x = 0, \tag{2.30}$$

$$T = T_{hot},\ \mathbf{u} = 0 \quad \text{at} \quad x = 1, \tag{2.31}$$

$$\frac{\partial T}{\partial y} = 0,\ \mathbf{u} = 0 \quad \text{at} \quad y = 0, 1. \tag{2.32}$$

Once the governing equations and boundary conditions are nondimensionalized, two parameters appear: the Prandtl number (Pr) and the Rayleigh number (Ra). In our experiments, we fixed Pr = 1 and varied the Rayleigh number, which increases the nonlinear effects of the convection terms and makes the solution more difficult to obtain. We used a 100×100 equally spaced mesh, which has 40,804 unknowns.

Figure 2.3 shows a plot of the streamlines and temperature contours for the thermal convection for $\mathrm{Ra} = 1.0 \times 10^5$. The fluid is heated on the right side and rises to the top due to buoyancy-driven forces. The fluid on the left side is cooled and drops toward the bottom. This induces a circular motion in the box as depicted by the streamlines.

The thermal convection was first run in transient mode, toward a steady state solution. Table 2.4 shows the timings for the problem at $\mathrm{Ra} = 1.0 \times 10^3$. The execution times

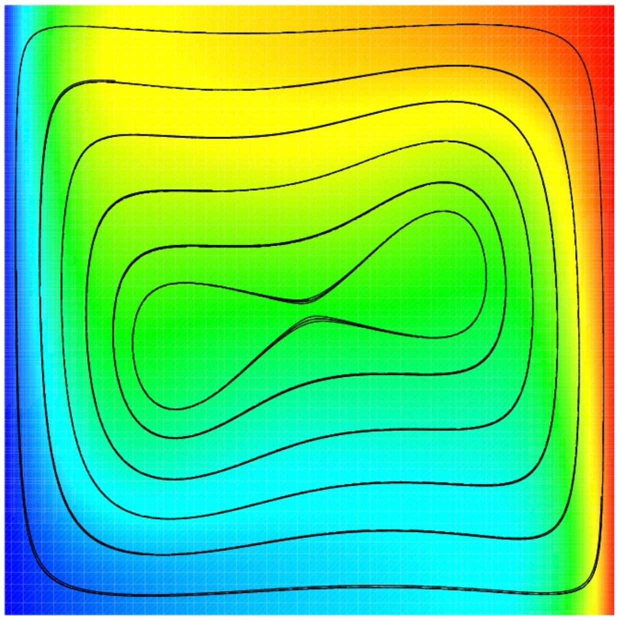

Figure 2.3. Thermal convection problem. Color contour plot of temperature with streamlines at Ra=10E5

show that Broyden's method and the modified Newton's method both are more efficient that Newton's method. We observe that the extra information in the rank-1 updates of Broyden's method has a minimal effect on the number of iterations required for convergence. In this case, Broyden's method did not accumulate enough information to impact the performance because each nonlinear problem was solved in a very small number of iterations (approximately 2 nonlinear steps per time step). For this problem, we conclude that it would be best to apply a modified Newton method to to avoid the complexities of Broyden's method.

2.3.3 Circuit Simulation

This section documents the testing of Broyden's method on Sandia's circuit simulation code, Xyce [36]. Circuit simulation represents a very difficult challenge to gradient-based nonlinear solvers. The equations are highly nonlinear and the solutions generate steep gradients. Some models can incorporate discontinuities into the equations. These difficulties will push the Broyden's method and help expose any drawbacks.

We have chosen four test problems from the regression test suite to perform our comparison. In this case, we compare the current default of Newton's method against Broyden's

Test Circuit	Method	S/F	NS	F	J	Total Time
Comparator	Newton	S	3439	5120	3439	0.80
	Broyden	S	3528	5206	1729	0.79
Dual Channel	Newton	S	9047	11495	9047	8.34
	Broyden	F	12572	14311	4985	16.7
Single Channel	Newton	F	30702	57731	31942	135.0
	Broyden	S	100589	264003	53563	417.0
ssu_none	Newton	S	385165	556754	385165	181.4
	Broyden	F	227328	327575	102702	101.7

Table 2.5. Performance comparison for various circuits using the Xyce circuit simulator.

method. We note that voltage limiting was disabled for the Newton's method runs, since a similar formulation for Broyden's method could not be formed. Voltage limiting is an algorithm designed to alter the Newton direction based on individual device performance; see [36]. Voltage limiting is, by default, enabled in Xyce since it causes Newton's method to behave in a more robust manner.

All simulations are transient runs. The results are shown in Table 2.5. Failed runs, marked by F in the S/F column are additionally highlighted in red. All observed failures come from the transient solver prematurely ending the run due to the error "time step is too small." This condition is triggered when the the nonlinear solver fails to converge to the requested tolerance and the time step is halved so many times that it is too small to make any reasonable progress.

The results are mixed. The comparator circuit was the only circuit where both Broyden's method and Newton's method successfully converged to the correct solution. For this circuit, the only difference in performance statistics was the number of Jacobian iterations. Broyden's method took about half as many iterations as Newton's method. But the total run times remained about equal. The reason for this is that Jacobian evaluation time for circuit simulation is almost negligible. All the work is done to evaluate the residual. Once a residual is computed, the Jacobian can be computed for almost nothing. For the case of the comparator circuit, for Newton's method the Jacobian evaluation accounted for 4% of the run time while for the Broyden's method it accounted for 2% of the run time. From an algorithmic perspective, a 50% reduction in Jacobian evaluation time is very good, but it is a negligible gain in the overall time.

A similar argument for reuse of the factorization holds. The Xyce runs used a direct linear solver that saved the factorization for reuse. For the dual channel run that converged, approximately 20% of the total run time is spent in the linear solve using Newton's method.

If Broyden's method had converged, it could only improve on a fraction of that 20%. The leads us to conclude that for circuit simulation, with respect to efficiency, the main advantages in using Broyden's method is lost. The ability to reuse the Jacobian and factorization each lead to little gain, because most of the work occurs in the residual evaluation.

The other problems either converged with Newton or converged with Broyden but not both. This shows that in circuits, the path an algorithm traverses is important in achieving convergence. If circuit simulators have convergence problems with Newton's method, one option would be to try using Broyden's method. In our experience with circuit simulation, voltage limiting has been the most successful option.

The Xyce circuit simulator constructs analytic Jacobians and therefore does not need to use JFNK or Broyden's method. Therefore, we do not recommend the general use of Broyden's method in Xyce. The only situation it may be of some assistance is if Newton's method fails.

2.4 Discussion

We implemented a modified version of Kelley's [37] limited-memory Broyden algorithm and tested it on large-scale problems on parallel machines. We made certain implementation choices to make the method robust. Broyden's method is similar to Newton's method but does not require a Jacobian be explicitly evaluated. Instead, a Jacobian estimate is constructed using a series of rank-1 updates to some initial guess. An added expense of storing one extra vector per nonlinear iteration is incurred.

Testing has demonstrated that Broyden's method can be used as a successful alternative to the JFNK method for solving systems where the Jacobian either cannot be computed explicitly or the estimation of the Jacobian is poor. Broyden's method may be more efficient than JFNK in cases where a large fraction of the run time is associated with the residual evaluations. In addition, the Broyden method does not require an iterative linear solver, as does JFNK, making it a more generally applicable algorithm.

An efficiency study has demonstrated that problems exist where Broyden's method can be more efficient than Newton's method. The efficiency comes from the fact that Broyden can reuse the Jacobian and the factorization/preconditioner for the linear solves. These methods are most useful when the problems are mildly nonlinear. Such cases arise most often from transient solves since there is a good initial guess and the time step size can control the degree of nonlinearity.

If the Broyden updates are not stored, Broyden's algorithm reduces to a modified Newton method. Comparing against modified Newton, we could isolate the effects of the rank-1 updates. In terms of efficiency, the updates do not play a critical role. These cases typically require very few nonlinear iterations (less than 10) and the updates fail to generate a good approximation to the Jacobian in such a small space. We do note, however, that in terms of robustness, the rank-1 updates can be critical if the Jacobian is inaccurate. As demonstrated in section 2.3.1.1, the updates aggregate information on the missing/inexact Jacobian terms to attain convergence at the cost of taking a larger number of Newton steps.

The Broyden algorithm has been implemented in the NOX Nonlinear Solver Library [41] and is available for download under the GNU LPGL license.

Chapter 3

Tensor Methods

In this chapter, we present our work with tensor methods [57] for solving large-scale systems of nonlinear equations. We discuss methods that were implemented in the nonlinear software package NOX [41] and our modifications to the tensor-Krylov method of Bouaricha [9], which considers a special case in a new way and improves its numerical performance. This research is aimed at studying the efficiency and robustness of tensor methods for problems at Sandia. In addition, we hope to bring tensor methods more into the mainstream by offering an implementation that can use any linear solver.

3.1 Introduction

Standard methods for solving the nonlinear equations problem (1.1), such as Newton's method, base each iteration upon a local, linear model $M(x_k + d)$ of the function $F(x)$ around the current iterate $x_k \in \mathbb{R}^n$. These methods work well for problems with well-conditioned Jacobians at the solution (they have a quadratic convergence rate), but they face difficulties when the Jacobian is singular or nearly singular at the solution. Many authors have analyzed the behavior of Newton's method on singular problems and have proposed acceleration techniques as remedies (see, e.g., Decker, Keller, and Kelley [14]; Decker and Kelley [15, 16, 17]; Griewank [29]; Griewank and Osborne [31]; Kelley and Suresh [39]; and Reddien [47]). Their collective analysis shows that Newton's method without acceleration is locally q-linearly convergent with the ratio of the norm of consecutive residuals converging to $\frac{1}{2}$. Acceleration techniques address this deficiency to some extent, but they require *a priori* knowledge that the problem is singular, which is not practical for general problem solving.

Tensor methods for nonlinear equations [57] do not require *a priori* knowledge of whether the Jacobian at the solution is singular or ill-conditioned (henceforth we just refer to the problem as being "singular" or "ill-conditioned"). They bypass this precondition by always including second-order information at each iteration in the local model:

$$M_T(x_k + d) = F(x_k) + J(x_k)d + \tfrac{1}{2}T_k dd,$$

where $T_k \in \mathbb{R}^{n\times n\times n}$ is a low-rank approximation to $F''(x_k)$, usually formed by a secant approximation. As a result, tensor methods have local superlinear convergence for a large class of singular problems under mild conditions [23]. Specifically, [23] shows that on problems where the rank of the Jacobian at the root is $n-1$, "practical" tensor methods (i.e., those using secant approximations for the tensor term T_k) have three-step superlinear convergence behavior with q-order $\frac{3}{2}$. In practice, one-step superlinear convergence frequently is observed on these problems. (Recall that Newton's method without any acceleration techniques on such problems exhibits only q-linear convergence.) In addition, the analysis in [23] shows that tensor methods have at least quadratic convergence on nonsingular problems.

The additional term T_k present in tensor methods provides second-order information in recent step directions, which aids in cases where the Jacobian is (nearly) singular at the solution. As the iterates approach the solution, the Jacobian lacks information in the null space direction, but the second-order term supplies useful information for a better quality step. Computational evidence in [57] on small problems shows that tensor methods provide 21–23% average improvement (in terms of function evaluations and/or nonlinear iterations) over standard methods on nonsingular problems and 40–43% improvement on problems with $\operatorname{rank}(J(x_*)) = n-1$. Thus, while tensor methods are not widely adopted, they generally outperform standard methods, particularly on ill-conditioned and singular problems.

Tensor methods that solve the local model with a direct factorization of the Jacobian matrix (like standard Newton's method) cannot efficiently solve large-scale problems due to large storage considerations and the expensive direct solution of the tensor model. To this end, several versions of inexact tensor methods have been developed for solving large problems. Bouaricha [9] describes an implementation that uses a Krylov-based linear solver and constructs an inexact tensor step from the approximate solutions of two linear systems (with the same Jacobian matrix). Feng and Pulliam [24] have developed a "tensor-GMRES" method, which first finds the Newton-GMRES step and then solves for an approximate tensor step. More recently, Bader has developed a class of methods called tensor-Krylov methods [2, 3] that calculates a tensor step from a specially chosen Krylov subspace. All of these algorithms are an amalgamation of various techniques, including tensor methods for nonlinear equations [57], Krylov subspace techniques [12], and an inexact solver framework [18], that make them well-suited for large-scale problems.

These large-scale tensor methods have advantages and disadvantages, especially when considering the development cost of implementing them in a production code, such as NOX. For example, while the tensor-Krylov methods generally have superior performance and better theoretical properties than the other methods, they are more complicated to implement due to a specialized local solver. On the other hand, Bouaricha's tensor method is simple to implement because it can use any linear solver, but it generally requires twice the total number of linear iterations as the others and gives up some theoretical guarantees.

It is with these tradeoffs in mind that we focused our research to develop a new tensor method resulting in a modification to Bouaricha's method that has good theoretical properties. The modified Bouaricha tensor method is implemented in the NOX Nonlinear Solver Library [41], which is available for download under the GNU LPGL license.

3.2 Background on Tensor Methods

Some background on tensor methods will be helpful and is described here. In the first subsection, we review the tensor methods as they were introduced in [57] and also discuss an alternative algorithm introduced in [9]. Since these standard approaches use direct factorizations of the Jacobian matrix, we refer to these methods as direct tensor methods. Due to the storage and linear algebra costs, these methods are only practical for solving small, dense problems. The last subsection discusses simple global strategies used by tensor methods, which require a modification to the standard global strategies because the tensor step is not guaranteed to be a descent direction.

3.2.1 Tensor Methods

Tensor methods were first proposed by Schnabel and Frank [57] and since have been studied by many others [2, 3, 9, 23, 24]. In deriving tensor methods for nonlinear equations, we consider the Taylor series expansion of $F(x)$ around x_k:

$$F(x_k+d) = F(x_k) + F'(x_k)d + \tfrac{1}{2}F''(x_k)dd + O(d^3). \tag{3.1}$$

Whereas Newton's method solves the first-order Taylor series approximation at each iteration, tensor methods include a special, restricted form of the quadratic term from the Taylor's expansion. While it would be impractical to actually form and store $F''(x_k)$ (i.e., $\frac{1}{2}n^3$ second partial derivatives of $F(x)$), further complications for solving a system of n quadratic equations in n unknowns would render this direct approach prohibitive. Thus, practical tensor methods store a secant approximation to $F''(x_k)dd$ based on information from previous iterations, similar to the concept of secant approximations for approximating the Jacobian (as in Broyden's method). At each iteration, a low-rank approximation to $F''(x_k)$ is formed, which requires considerably less storage and allows the model to be solved efficiently. Augmenting the linear model with this term yields what we call the local tensor model:

$$M_T(x_k+d) = F(x_k) + J(x_k)d + \tfrac{1}{2}T_k dd, \tag{3.2}$$

where $T_k \in \mathbb{R}^{n\times n\times n}$ is a low-rank approximation to $F''(x_k)$. The term T_k is selected so that the model interpolates $p \leq \sqrt{n}$ previous function values in the recent history of iterates, which makes T_k a rank-p tensor. Computational evidence in [57] suggests that $p > 1$ adds little to the computational performance of a direct tensor method, so we focus on the case of $p = 1$, which uses only one secant update and creates a rank-1 tensor. There may be scenarios in which a higher-rank approximation may be useful, such as dealing with a Jacobian having a rank less than $n-1$.

In the case of $p=1$, the tensor model about x_k reduces to

$$M_T(x_k+d) = F_k + J_k d + \tfrac{1}{2} a_k (s_k^T d)^2, \tag{3.3}$$

where

$$a_k \in \mathbb{R}^n = \frac{2(F_{k-1} - F_k - J_k s_k)}{(s_k^T s_k)^2}, \tag{3.4}$$

$$s_k \in \mathbb{R}^n = x_{k-1} - x_k. \tag{3.5}$$

After forming the model, we use it to determine the step to the next trial point. Because (3.3) may not have a root, we solve the minimization subproblem

$$\min_{d \in \mathbb{R}^n} \|M_T(x_k+d)\|_2, \tag{3.6}$$

and a root or minimizer of the model is the tensor step. Due to the special form of (3.3), the solution of (3.6) in the nonsingular case reduces to minimizing a quadratic equation followed by solving a system of $n-1$ linear equations in as many unknowns.

This minimization problem may be solved in a variety of ways. Schnabel and Frank [57], for example, show how to find the solution for arbitrary p using orthogonal transformations, and this method is described first. An alternative approach by Bouaricha [9], based on a reduced system when $p=1$, is described later. Background in both approaches is useful for introducing the large-scale tensor methods.

3.2.1.1 Method of Orthogonal Transformations

The first approach for solving (3.6) uses two orthogonal transformations to reduce the problem to two subproblems that are more easily solved. We refer to [57] for more details, but briefly the approach is as follows. The first transformation finds an orthogonal $Q_1 \in \mathbb{R}^{n\times n}$ such that $s_k/\|s_k\|$ is the last column and permits a change in variables

$$d = Q_1 \hat{d}.$$

The second transformation finds an orthogonal $Q_2 \in \mathbb{R}^{n\times n}$ such that $Q_2 J_k Q_1$ is upper triangular. Applying these two transformations to (3.3) and setting the system equal to zero, yields the following triangular system of n equations in n unknowns

$$Q_2 F_k + Q_2 J_k Q_1 \hat{d} + \tfrac{1}{2} Q_2 a_k \|s_k\|^2 \hat{d}_n^2 = 0, \tag{3.7}$$

where $\hat{d}_n \in \mathbb{R}$ is the unknown appearing in the quadratic equation.

Partitioning (3.7) into two smaller problems, the solution to (3.6) continues by first solving for $\hat{d_n}$ by minimizing the quadratic equation appearing in the last row of (3.7) and choosing the smaller magnitude minimizer if there are two. Using the value of $\hat{d_n}$ in (3.7), a triangular linear system of size $(n-1) \times (n-1)$ is revealed. Finally, the complete solution to (3.6) is found by solving this resultant system for the remaining components of $\hat{d}$ and then reversing the variable space transformation from the first step, $d = Q_1 \hat{d}$.

3.2.1.2 Reduction Method

The second approach due to Bouaricha [9] for solving the tensor model (3.6) involves solving a reduced system of quadratic equations after first solving two linear systems with the same Jacobian matrix. The method may handle an arbitrary size of p, but we will restrict ourselves to the case $p = 1$. We will call this approach the "reduction method" because it solves two linear systems and then reduces the system of quadratic equations to a single quadratic equation.

In this approach, the tensor step is found by multiplying (3.3) by $s_k^T J_k^{-1}$ and finding the root or minimizer of the resulting equation,

$$s_k^T J_k^{-1} F_k + s_k^T d + \tfrac{1}{2} s_k^T J_k^{-1} a_k (s_k^T d)^2 = 0. \tag{3.8}$$

Defining the quantity $s_k^T d$ as β, a quadratic equation in β—call it $q(\beta)$—is revealed,

$$\begin{aligned} \beta &\equiv s_k^T d \\ q(\beta) &\equiv s_k^T J^{-1} F_k + \beta + \tfrac{1}{2} s_k^T J_k^{-1} a_k \beta^2. \end{aligned} \tag{3.9}$$

Since (3.9) may or may not have a root, solving the single-variable minimization problem

$$\min_{\beta \in \mathbb{R}} |q(\beta)| \tag{3.10}$$

provides the value β_*. If $q(\beta)$ has two real roots, then the root of smaller magnitude is taken. To solve the original problem and find the tensor step, we calculate the value of $q(\beta_*)$ and substitute this value and β_* into the explicit step

$$d_T = -J_k^{-1} \left[F_k + \tfrac{1}{2} a_k \beta_*^2 - \frac{J_k^{-T} s_k q(\beta_*)}{s_k^T (J_k^T J_k)^{-1} s_k} \right]. \tag{3.11}$$

We omit the proof that this step solves the tensor model (see [9] for details), but one can verify that the step (3.11) is a root or minimizer of (3.3) by substituting it into the model, simplifying, and recalling the definition $\beta_* = s_k^T d_T$ and that $q(\beta_*)$ has minimum norm. Quite often $q(\beta_*) = 0$, which avoids computing the last term. This term requires a non-trivial amount of extra arithmetic to form, so the large-scale implementation in [9] actually neglects this term without apparent detriment.

The operation count of this process when $p = 1$ is the cost of a matrix factorization of the Jacobian J_k, two back substitutions using this factorization, plus the cost of minimizing (3.10), which for this case of $p = 1$ a closed form solution exists. Thus, the total cost beyond a standard method is about n^2 multiplications due to the extra linear solve (ignoring the extra term if $q(\beta_*)$ is nonzero).

If during an iteration the Jacobian is singular, then a modified approach may be taken with this $q(\beta)$ formulation. It involves changing the Jacobian to be nonsingular, modifying the function value, and redefining β, all in a way that still results in solving the original model. The modified tensor model becomes

$$M_T(x_k + d) = \hat{F}_k + \hat{J}_k d + \tfrac{1}{2} a_k \hat{\beta},$$

where $\hat{J}_k$ equals J_k plus a low-rank matrix. This technique is not new, but it has not been published in much detail. We will discuss it in more detail in section 4.1.1.

3.2.2 Global Strategies

To achieve global convergence, most tensor method implementations use a line search strategy for its simple implementation and reasonable robustness. To complete this section's background review of tensor methods, we provide brief descriptions of a common tensor method line search strategy and the more advanced curvilinear line search.

First, we present the line search strategy introduced in [57], which chooses between the tensor step and the Newton step for the search direction for backtracking. This strategy always attempts the full tensor step first to preserve convergence properties of the tensor method. However, because the tensor step is not guaranteed to be a descent direction on the merit function $f(x) = \|F(x)\|$, a "backup" descent direction is needed if the first trial point is not acceptable and the tensor step is not a descent direction. Fortunately, the Newton direction is guaranteed to be a descent direction on the merit function, which would be used in place of the tensor step in such cases. An important aspect of tensor methods for small, dense systems is that the Newton step can be computed readily with an extra linear solve by ignoring the tensor term (or the Newton step is already computed with the method of Bouaricha).

To be precise, the line search algorithm for step selection from [57], which we call the "standard tensor line search," is presented here:

Algorithm 2.1: STANDARD TENSOR LINE SEARCH

```
If (no root or minimizer of the tensor model could be computed)
Or ((minimizer of tensor model that is not a root was found)
And ($\|M_T(x_k + d_T)\|_2 > \frac{1}{2}\|F(x_k)\|_2$)),
Then $x_{k+1} \leftarrow x_k + \lambda d_N, \lambda \in (0,1]$ selected by line search;
Else $x_{k+1} \leftarrow x_k + d_T$.
If $x_{k+1}$ is not acceptable, then
    If $d_T$ is a sufficient descent direction.
    Then $x_{k+1} \leftarrow x_k + \lambda d_T, \lambda \in (0,1]$ selected by line search;
    Else $x_{k+1} \leftarrow x_k + \lambda d_N, \lambda \in (0,1]$ selected by line search.
```

As can be seen, different conditions necessitate line searches with either the tensor step or the Newton step (d_T or d_N).

The second global strategy we present is the curvilinear line search [4]. The curvilinear line search involves the solution of the modified tensor model $\lambda F + Jd + \frac{1}{2}a(s^T d)^2$, where $\lambda \in (0,1]$ is the line search parameter. As the value of λ changes according to some prescribed procedure (e.g., quadratic backtracking or interval halving), the solution to the modified tensor model is the curvilinear step $d_T(\lambda)$. The curvilinear step has some nice theoretical properties, including guaranteed descent as $\lambda \to 0$ and monotonicity on the merit function. Unlike the standard tensor line search, the curvilinear line search obviates the need for special cases because it sweeps out a path spanned by the Newton direction and the tensor step. In that regard, the curvilinear line search is similar to a trust region method (cf. the optimal hook step [19]), which spans the steepest descent direction and Newton step.

3.3 Review of Large-scale Methods

In the previous section, we reviewed direct tensor methods. We now shift our focus to solving large, sparse problems, which requires a different approach to be computationally efficient and have low storage requirements. Three classes of large-scale tensor methods already exist and are predicated upon Krylov subspace methods.

We begin the discussion by reviewing inexact Newton methods [18] and, in particular, Newton-Krylov methods [12], where linear Krylov subspace methods solve the local model to some relative tolerance. Then we review the tensor-Krylov method of Bouaricha [9], tensor-GMRES of Feng and Pulliam [24], and the tensor-Krylov methods of Bader [2, 3].

3.3.1 Newton-Krylov Methods

Large, sparse nonlinear systems often are solved successfully using a class of "inexact" Newton methods:

$$x_{k+1} = x_k + d_k, \quad \text{where} \quad J(x_k)d_k = -F(x_k) + r_k, \quad \|r_k\| \le \eta_k \|F(x_k)\|. \tag{3.12}$$

The local model is solved only approximately at each step using an iterative linear solver.

A common approach for approximately solving the local Newton model in (3.12) uses Krylov-based methods [12]. A linear Krylov subspace method is a projection method that seeks an approximate solution x_m to the linear system $Ax = b$ from an m-dimensional affine subspace $x_0 + \mathcal{K}_m$. Here, $\mathcal{K}_m$ is the Krylov subspace

$$\mathcal{K}_m(A, r_0) = span\{r_0, Ar_0, A^2 r_0, \ldots, A^{m-1} r_0\},$$

where $r_0 = b - Ax_0$ is the residual at an initial guess x_0. A popular Krylov subspace method is the Generalized Minimum Residual method (GMRES) [52], which computes a solution $x_m \in x_0 + \mathcal{K}_m$ such that the residual norm over all vectors in $x_0 + \mathcal{K}_m$ is minimized. That is, at the mth step, GMRES finds x_m such that $\|b - Ax_m\|_2$ is minimized for all $x_m \in x_0 + \mathcal{K}_m$. One drawback of GMRES is the storage requirement of an orthogonal basis, which could be larger than a sparse Jacobian matrix unless m is kept small, leading to the class of restarted GMRES methods [52]. Other Krylov methods, such as BiCGSTAB and TFQMR (see, e.g., [51]), do not have these additional storage requirements but may not be as robust. A Newton-based method that uses GMRES as its local solver is called Newton-GMRES. It is a popular algorithm for solving large-scale problems and will be used as the standard Newton-based algorithm in our numerical experiments.

3.3.2 Bouaricha's Method

As presented in section 3.2.1, the tensor step is calculated in one of two ways: (1) using orthogonal transformations for small, dense problems, or (2) using the reduction method for both sparse and dense systems (but with more difficulty if the Jacobian is singular or if the model does not have a root).

With the reduction method, one needs the values $s^T J^{-1} F$ and $s^T J^{-1} a$ in equation (3.9) to find the value of β_*. The advantage of the reduction method is that any linear solver may be used to solve these linear systems, which makes the method noninvasive. For instance, efficient Krylov methods may be used for solving the linear systems $J^{-1}F$ and $J^{-1}a$, positioning them for use on large-scale problems. Indeed, this is the approach adopted by Bouaricha in [9] for a large-scale tensor method, which he calls a tensor-Krylov method. We believe this is an unfortunate name because the method is not restricted to using Krylov-based linear solvers. For the sake of clarity to distinguish it from the tensor-Krylov methods of [2, 3], we will refer to his tensor-Krylov method as "Bouaricha's method."

Bouaricha's method uses principles from inexact Newton methods and expects only approximate solutions to the two linear systems. Thus, when calculating β_* from (3.10), the value of β_* will be inexact, but hopefully a good enough estimate.

A key point of Bouaricha's algorithm, however, differentiates it from a straightforward implementation that uses just the concepts above. For this discussion let the previous nonlinear iterate be denoted with the subscript $k - 1$. So instead of approximately solving the system $J_k^{-1}a_k$, his method solves a different linear problem, $J_k^{-1}F_{k-1}$, and then reaches $J_k^{-1}a_k$ via the linear combination: $2(J_k^{-1}F_{k-1} - J_k^{-1}F_k - s_k)/(s_k^T s_k)^2$. The beauty of this approach is due to a clever initial guess for the second system, $J_k^{-1}F_{k-1}$. Because the Jacobian typically changes very little from iteration to iteration near the solution, the previous Newton direction $J_{k-1}^{-1}F_{k-1}$ will be close to $J_k^{-1}F_{k-1}$ if the Jacobian at the root is nonsingular. If the Jacobian is singular, then the directions are likely to be collinear but of different magnitudes. Thus, Bouaricha's method uses the approximate Newton step calculated from the previous step as an initial guess for the system $J_k w = F_{k-1}$. The complete algorithm for solving the local model at each nonlinear iteration is presented below.

Algorithm 3.1: BOUARICHA'S TENSOR-KRYLOV METHOD

1. Approximately solve $J_k d_N = -F_k$ for d_N using a Krylov subspace method.
2. Approximately solve $J_k y = -F_{k-1}$ for y using a Krylov subspace method and starting from the initial guess $y_0 = -d_{N_{k-1}} = J_{k-1}^{-1}F_{k-1}$.
3. Form the term $w = J_k^{-1}a_k$ by using the following relationship:

$$\begin{aligned} w &= J_k^{-1}a_k \\ &= \frac{2}{(s_k^T s_k)^2}(J_k^{-1}F_{k-1} - J_k^{-1}F_k - s_k) \\ &= \frac{2}{(s_k^T s_k)^2}(-y + d_N - s_k). \end{aligned}$$

4. Form the quadratic equation

$$\begin{aligned} q(\beta) &= \tfrac{1}{2}s_k^T J_k^{-1}a_k\beta^2 + \beta + s_k^T J_k^{-1}F_k && (3.13) \\ &= \tfrac{1}{2}s_k^T w\beta^2 + \beta - s_k^T d_N, && (3.14) \end{aligned}$$

and find the smallest magnitude real root or minimizer of $q(\beta)$.

5. Calculate the tensor step

$$\begin{aligned} d_T &= -J_k^{-1}F_k - \tfrac{1}{2}J_k^{-1}a_k\beta_*^2 \\ &= d_N - \tfrac{1}{2}w\beta_*^2, \end{aligned}$$

where β_* is the solution of (3.14).

6. Select the next iterate x_{k+1} using a line search global strategy (as outlined in section 3.3.4.3).

When calculating the tensor step, Bouaricha's method also neglects the complicated term in (3.11) involving $q(\beta_*)$,

$$\frac{(J_k^T J_k)^{-1} s_k}{s_k^T (J_k^T J_k)^{-1} s_k} q(\beta_*). \tag{3.15}$$

The justification for this omission is that the term has a considerable computational cost for large-scale problems and that there appears to be no way to calculate it without having all of J_k available. Also, it is proved in [9] that the step calculated without (3.15) still retains the same convergence properties of the tensor method. As a sketch of the proof, the expression (3.15) is negligible in the neighborhood of the solution and, therefore, has no effect on the tensor method. In the numerical experiments in [9], omitting the term had a seemingly negligible impact on the performance of the algorithm. Our modification to Bouaricha's method discussed in section 3.4 also seeks to avoid this complicated term.

Despite favorable results in [9], studies in [2] have found the method to be less impressive on more practical problems. The two main disadvantages of Bouaricha's method stem from the fact that two linear systems must be solved for each outer iteration and that an accurate value of β is not calculated. These disadvantages are discussed next.

As specified in the algorithm above, solving the two linear systems required by Bouaricha's method ($J_k d = -F_k$ and $J_k y = -F_{k-1}$) is roughly twice the cost per nonlinear iteration of Newton's method. The clever initial guess for the second linear system usually does not help until the iterates get close to the solution, at which point typically only a few more nonlinear iterations are needed. Until then, solving the two linear systems separately doubles the cost per nonlinear iteration as compared with Newton's method. Thus, for these methods to be competitive, the number of nonlinear iterations would need to be halved, which does not happen very frequently on practical problems.

In addition to the added cost of solving two linear systems independently, it is not clear how precise the solutions need to be to provide an accurate value for β in Bouaricha's method. This feature can be detrimental to the method by making it potentially unstable. That is, if both linear systems are solved only approximately, then β may be inaccurate, affecting the quality of the computed tensor step since the scalar coefficient of the vector $J^{-1}a$ is $\frac{1}{2}\beta^2$. Any errors in β are magnified once the value is squared, particularly if the approximate value is much larger than the true value. Thus, Bouaricha's method can and does compute spurious steps and performs worse than corresponding Newton-Krylov methods.

Numerical results in [2] show that these disadvantages often outweigh any benefit of calculating the tensor step. Indeed, testing on medium-sized problems ($n < 1000$) showed that the performance of Bouaricha's method usually trailed the performance of Newton's method as well as the other large-scale tensor methods. Due to the theoretical disadvantages mentioned above and the lackluster numerical performance, this algorithm has not received much attention. In section 3.4, we will examine modifications to Bouaricha's method that improve it.

3.3.3 Tensor-GMRES Method

Another large-scale tensor method is that of Feng and Pulliam [24]. It is also a tensor-Krylov method but employs a different strategy. It uses Krylov subspace projection techniques for solving the Newton equations; and, in particular, it uses GMRES to find the approximate Newton step $d_N = d_0 + V_m y_m$. Once again, the columns of V_m form an orthonormal basis for the Krylov subspace $\mathcal{K}_m$ generated by the corresponding Arnoldi process, and the Hessenberg matrix H_m is also generated from the Arnoldi process. Given these key matrices, their tensor-GMRES algorithm proceeds to solve a projected version of the tensor model (3.3) along a subspace that spans the Newton step direction (i.e., the approximate tensor step is in the span of the Krylov subspace $\mathcal{K}_m^N$ and d_0, or equivalently the span of the matrix $[V_m, d_0]$). Thus, their algorithm solves the least-squares problem

$$\min_{d \in \{d_0\} \cup \mathcal{K}_m^N} \left\| F_k + J_k d + \tfrac{1}{2} P a (s^T d)^2 \right\|, \tag{3.16}$$

where P is the projection matrix

$$P = Y(Y^T Y)^{-1} Y^T, \quad \text{where } Y = J_k [V_m, d_0]. \tag{3.17}$$

Despite the algorithm's daunting algebra, the design is actually rather straightforward. The algorithm may be viewed as an extension of Newton-GMRES, where the inexact Newton step is calculated via GMRES in the standard way. An approximate tensor step is calculated subsequently using the Krylov subspace information generated for the Newton step. In this way, the method is also consistent with preconditioning techniques and a matrix-free implementation, which makes it appealing for general use. However, because it requires the orthogonal basis V_m and Hessenberg matrix H_m from the linear solver, we classify the tensor-GMRES method as an "invasive" method.

The Feng and Pulliam tensor-GMRES method for solving the local tensor model involves some difficult algebra, but the extra work and storage beyond GMRES is actually quite small. The extra work is at most $4mn + 5n + 2m^2 + O(m)$ multiplications plus a single Jacobian-vector product for evaluating the rank-one tensor vector a_k. The extra storage amounts to two extra n-vectors for a_k and s_k plus a few smaller working vectors of length m.

The analysis in [24] shows that the same superlinear convergence properties for the unprojected tensor model considered in [23] also hold for the projected tensor model (3.16). Both analyses consider ideal tensor models (i.e., when a_k is formed using information from the singular value decomposition of J_k instead of from a secant approximation) and extend the ideal result to practical implementations that use a secant approximation for a_k. When considering only ideal tensor models, the only difference between the model in [23] and the projected model (3.16) is a projection matrix P in front of a_k. In the ideal case, the projection matrix P is the matrix WW^T, such that the left singular vector corresponding to the least singular value of J_k is in the span of the orthonormal column vectors of W. Feng and Pulliam show that this difference does not affect the proofs of Lemmas 4.1 and 4.2 in

[23], so the rest of the convergence analysis follows exactly the proof of Theorem 4.4 in [23].

Feng and Pulliam extend this convergence result of an ideal, projected tensor model to the practical, projected tensor model of (3.16) with the use of two approximations. First, the Newton direction found via GMRES, $d_N \in [V_m, d_0]$, will be arbitrarily close to v_N^c, the right singular vector corresponding to the smallest singular value of J_k. Thus, the difference between two consecutive iterates, $s_k = x_{k-1} - x_k$, is likely to be along the null space when approaching a rank-deficient solution, so a practical implementation uses s_k to approximate the null space direction. Second, the ideal projection matrix WW^T projects onto the subspace containing the left singular vector corresponding to the smallest singular value of J_k, which is in the same direction as $J_k v_N^c$. Hence, a reasonable approximation to WW^T is the projection matrix (3.17). While a rigorous proof is not provided in [24], these two approximations for a practical tensor-GMRES method appear to work well in their numerical tests. See [23] and [24] for more theoretical details.

The Feng-Pulliam tensor-GMRES method has been tested in a variety of numerical experiments, including comparisons in [2, 3, 4, 24]. In all, the results suggest that tensor-GMRES is a competitive algorithm that usually performs on par with other large-scale tensor methods and in some cases even better. However, there are instances when performance is hindered. These cases stem from a few theoretical deficiencies of tensor-GMRES that are related to using a Krylov subspace generated from the solution of the Newton step. We discuss these disadvantages in more detail next.

First, the variable space restriction on d in the minimization problem (3.16) illustrates a possible disadvantage of the Feng-Pulliam method, particularly when using preconditioners or restarted GMRES. Instead of solving (3.16) in the full variable space (which would be expensive), Feng and Pulliam restrict the solution d along a subspace that spans the Krylov subspace generated from the Newton-GMRES step since the Newton step usually provides good directional information. Hence, the norm of the projected tensor model is only minimized to the extent that the Krylov subspace for the Newton-GMRES step is large enough to capture important directional information of the second-order tensor term.

Consider, for example, using an *exact* preconditioner, i.e., $M = J_k$. One iteration of GMRES solves the Newton equations exactly, and the Newton step direction is v_1 from the orthonormal basis V_m. Then, according to the Feng-Pulliam method, the approximate tensor step that solves (3.16) could only be a scalar multiple of the direction v_1 (assuming that $d_0 = 0$). A similar example may be developed when using restarted GMRES in the Feng-Pulliam method—if GMRES converges soon after a restart, then the orthonormal basis V_m is smaller than before the restart. A smaller basis may lead to a tensor step that solves (3.16) with more error due to fewer degrees of freedom. Some limited testing in [2] suggests that solving (3.16) in a smaller variable space adversely affects the practical performance of this method when using preconditioners or restarted GMRES.

Second, the Feng-Pulliam method solves the *projected* tensor model (3.16) instead of the true tensor model without a projection. Undoubtedly, some information is lost with the projection, which may lead to a less accurate tensor step. In the limit, this projection is not supposed to be detrimental to the convergence rate of tensor-GMRES. Because the Newton step tends to undershoot (or overshoot) when first-order information is lacking in the local model, the solution to the tensor model is often nearly along the Newton direction, so the subspace restriction on d might not be a problem.

A third and final disadvantage is that the relative stopping tolerance η_k in the Newton-GMRES step has no direct relationship with the error in the tensor model. That is, solving the Newton step to a relative tolerance of 10^{-1} does not guarantee that the tensor step is solved to a tolerance of at least 10^{-1}. The implication is that the step may be of poorer quality than the local tolerance suggests.

3.3.4 Tensor-Krylov Methods

The tensor-Krylov methods introduced in [2, 3] differ from previous large-scale tensor methods due to their ability to solve the local tensor model to a specified tolerance. Using either Bouaricha's method or tensor-GMRES, the residual error $\|M_T(x_k+d)\|$ must be computed explicitly, making it difficult to assess the quality of the approximate tensor step that is computed. In addition, tensor-Krylov methods avoid the costly solution of two linear systems (as opposed to two in Bouaricha's method) and compute the solution to the full tensor model, as opposed to a projected tensor model (as in the Feng-Pulliam method). However, tensor-Krylov methods are invasive and require a customized solver.

In the same manner that GMRES is an algorithm for solving linear systems and Newton-GMRES is the nonlinear solver, a distinction is made between the solver for the local tensor model and the nonlinear solver. Here, we outline one of three procedures [2] for iteratively solving the local tensor model that use the concepts from linear Krylov subspace methods. All three procedures share the same conceptual design, so discussing only one will suffice. Then, we consider the issues of the tensor-Krylov nonlinear solver that implements the local solver.

Once again, we restrict ourselves to the rank-one tensor model in (3.3)–(3.5), which only interpolates the function value at the previous iterate. Tensor-Krylov methods find a solution to the minimization problem

$$\min_{d\in\mathcal{K}_m} \|M_T(x_k+d)\|_2 = \min_{d\in\mathcal{K}_m} \left\|F_k + J_k d + \tfrac{1}{2}a_k(s_k^T d)^2\right\|_2, \tag{3.18}$$

where $\mathcal{K}_m$ is a specially chosen Krylov subspace that facilitates the solution of the quadratic model. The three tensor-Krylov methods differ in their choice of $\mathcal{K}_m$, which becomes their signature difference and dictates the algorithm. The three variants are differentiated by the size of their initial block subspace, identifying them as block-2, block-2+, and block-3. The three variants have different complexities and usefulness as block algorithms. For instance, the block-3 method is the most straightforward and capable block implementation, while the block-2 methods are more complex but work better in scalar implementations.

Here, we provide only the basic details of the block-3 methods, and we refer to [2, 3] for more detailed information on all three algorithms. We discuss issues that are important to a nonlinear equations solver, including block-Krylov subspace issues, residual calculation, stopping conditions, preconditioning and scaling techniques, computation of the Newton step, and cost. Section 3.3.4.2 wraps the local solver into a complete tensor-Krylov algorithm for solving large-scale systems of nonlinear equations, and section 3.3.4.3 discusses the global strategies for the tensor-Krylov algorithm.

3.3.4.1 Block-3 Local Solver

The block-3 algorithm for solving (3.18) proceeds in a block-Krylov-like fashion, operating on a matrix of three initial vectors instead of the single residual r_0 of a linear system. By choosing three vectors, we may include information on the three known vectors in the local tensor model (s, a, and F_k) and allow a transformation of the variable space and function space in a manner similar to the method of orthogonal transformations of section 3.2.1. To that end, we consider the block of initial vectors

$$R_0 = [s,\ (Jd_0 + F_{k-1}),\ (Jd_0 + F_k)]. \tag{3.19}$$

The rationale for choosing these specific vectors is as follows. The vector s is listed first in order to isolate the inner product $s^T d$ (via $\|s\|\, v_1^T d$) and later create a single quadratic equation in a single unknown. The second vector is the residual involving the previous function value F_{k-1} and is needed for computing the tensor term a, (3.4). The third vector is the residual of the Newton equations, and it may be placed as the second or third column in R_0. Collectively, these three vectors are chosen specifically to compute the tensor term a later in the algorithm in addition to fully characterizing the local tensor model (i.e., representing the three known vectors F_k, a, s) with this initial subspace.

The first step of the algorithm computes the QR-factorization of R_0,

$$R_0 = VR = [v_1, v_2, v_3]R, \tag{3.20}$$

where $V \in \mathbb{R}^{n\times 3} = [v_1, v_2, v_3]$ is unitary and $R \in \mathbb{R}^{3\times 3}$ is upper triangular. A block-Arnoldi process then creates additional columns of an orthonormal basis V_m that spans the block-Krylov subspace

$$span\{V, JV, J^2V, J^3V, \dots\}. \tag{3.21}$$

There are several block-Arnoldi versions available for implementation, and the particular variant is not critical to the implementation of the tensor-Krylov method. The standard procedure works on a whole block $V \in \mathbb{R}^{n\times t}$ and adds t vectors ($t = 3$ in this case) to the subspace at a time. This block procedure may work well when considering cache memory performance, and the single-vector version of block-Arnoldi more closely corresponds with the scalar implementation of GMRES. The version in Algorithm 3.2 is very similar to the standard Arnoldi algorithm, which operates on a single vector at a time and is due to Ruhe [48] (see also [50]) for the symmetric case (block Lanczos).

Algorithm 3.2: BLOCK ARNOLDI PROCESS—RUHE'S VARIANT

```
1. Choose $t$ initial orthonormal vectors $\{v_i\}_{i=1,\dots,t}$.
2. Choose a number of Arnoldi iterations to perform and set to $m$.
3. For $k = 1,\dots,m$ :
    (a) Set $j := k+t-1$
    (b) Compute $w := Jv_k$
    (c) For $i = 1,2,\dots,j$
        i. $h_{ik} := (w, v_i)$
        ii. $w := w - h_{ik}v_i$
    (d) $h_{j+1,k} := \|w\|_2$
    (e) If $h_{j+1,k} \neq 0$, then set $v_{j+1} := w/h_{j+1,k}$;
        Else if $t = 1$, then Stop;
        Else set $t := t-1$ and continue.
```

The first step of the algorithm is to multiply a single vector, v_1, by the Jacobian matrix J and orthonormalize the resulting vector w against all j vectors $v_1,\dots,v_j$ ($j = t$ at the first iteration) in the orthonormal basis, building the subspace one vector at a time. Thus, a vector from the initial block $\{v_i\}_{i=1,\dots,t}$ is multiplied by J every t steps. The last step 3e avoids a division by zero and is commonly referred to as the breakdown condition. In the scalar case ($t = 1$), a breakdown condition indicates that the solution is in the subspace spanned by the k basis vectors computed thus far. Here in the block case, we must modify the usual condition to reduce the block dimension by one until it eventually reduces to the scalar case.

After m steps on the initial matrix $V \in \mathbb{R}^{n\times 3}$ defined in (3.20), the block-Arnoldi process produces an orthogonal matrix $V_{m+3} \in \mathbb{R}^{n\times(m+3)}$ and a matrix $\bar{H}_m \in \mathbb{R}^{(m+3)\times m}$ whose nonzero entries are the elements h_{ik} computed in the process. It is important to note that $\bar{H}_m$ is banded upper Hessenberg with three subdiagonals. The orthonormal basis V_{m+3} and the matrix $\bar{H}_m$ have an important relationship,

$$JV_m = V_{m+3}\bar{H}_m. \tag{3.22}$$

The block-3 algorithm uses orthogonal transformations and permutation matrices to switch rows and columns to isolate a quadratic equation in the mth row. After the block-Arnoldi process adds a basis vector and an extra column to $\bar{H}_m$, we perform a series of plane rotations to put the matrix $\bar{H}_m$ in upper triangular form. In its current ordering, the quadratic equation would not be isolated to a single variable in the first row and should be switched to the mth row. So we permute the first row and column with the mth row and column to facilitate an easier solution. After all of the orthogonal transformations and row/column permutations, the structure of the simplified problem is

$$\begin{pmatrix} \star \\ \star \\ \star \\ \vdots \\ \star \\ \star \\ \star \\ \star \end{pmatrix} + \begin{pmatrix} \star & \star & \star & \cdots & \star \\ & \star & \star & & \star \\ & & \star & \cdots & \star \\ & & & \ddots & \vdots \\ & & & & \star \\ & & & & \\ & & & & \\ & & & & \end{pmatrix} \hat{y} + \begin{pmatrix} \star \\ \star \\ \star \\ \vdots \\ \star \\ \star \\ \star \\ \star \end{pmatrix} (s^T d_0 + \|s\| \hat{y}_m)^2, \tag{3.23}$$

where $\hat{y}$ is the vector of unknowns and $\hat{y}_m$ is the last element in $\hat{y}$. Similar to the last step in the orthogonal transformations approach of section 3.2.1, one finds the minimizer $\hat{y}_m$ and then solves a linear system for the remaining elements in $\hat{y}$. Finally, one uses V_m and previous permutations/transformations to compute the optimal solution x from $\hat{y}$.

The decision for stopping the Arnoldi process so that the approximate step solves the tensor model to a specified tolerance appears before the computation of the explicit step, which is at an inconvenient location.

There are two possible implementations for computing a stopping condition in these Krylov-based methods, and they are fundamentally similar. Both may be checked without explicitly computing the approximate step d_m after each step in the Arnoldi process. We briefly mention one approach here, which is used in our numerical tests.

The idea is analogous to what is done in GMRES, which uses an efficient approach in its least-squares solution. With GMRES, the least-squares error $\|b - Ax\|_2$ is equal to the last element of $Qe_1 \|b\|$, where Q is the product of all Givens rotations to transform the Hessenberg matrix to upper triangular form and e_1 is the unit vector $(1, 0, 0, \dots)^T$.

The approach with the block-3 method is similar in that it involves computing the norm of the remaining rows below the triangular part of $\tilde{H}_m$. The last four rows of the $m \times (m+3)$ system (3.23) pertain to the least-squares error of the local tensor model. Hence, one neglects the contribution from the quadratic equation in row m of (3.23) (because it is already minimal) and calculates the norm of the last three rows for use in a stopping condition.

As a final remark, these local solvers are amenable to preconditioning and scaling. The modifications to the block-Arnoldi method are no different from what is done with preconditioning and scaling in GMRES. There are a few additional steps that deal with handling s_k and a_k in the preconditioned mode, but they are minor.

3.3.4.2 Tensor-Krylov Nonlinear Solver

Now that the Krylov-based iterative methods for solving the local tensor model has been introduced, we return to solving the general nonlinear equations problem (1.1). The following algorithm outlines the tensor-Krylov algorithm, which at every outer iteration calls a Krylov-based iterative method for solving the local tensor model.

Algorithm 3.3: TENSOR-KRYLOV METHOD

1. Given the nonlinear equations problem $F(x)$, choose a starting point x_0 and set the maximum iteration counter k_{max}.
2. For $k = 0, 1, 2, \ldots, k_{max}$, do:
    (a) Choose a forcing term tolerance $\eta_k \in [0, 1)$.
    (b) If $k = 0$, then calculate the Newton-GMRES step d_N according to the relative tolerance η_k and proceed to step 2e.
    (c) Form the local tensor model $M_T(x_k + d) = F_k + Jd + \frac{1}{2}a(s^T d)^2$, where $F_k = F(x_k)$, $F_{k-1} = F(x_{k-1})$, $J = F'(x_k)$, $s = x_{k-1} - x_k$, and $a = \frac{2(F_{k-1} - F_k - Js)}{(s^T s)^2}$.
    (d) Compute the inexact tensor step d_T according to the relative tolerance η_k by approximately solving the local tensor model according to the block-2, block-2+, or block-3 methods.
    (e) Set $x_{k+1} = x_k + \lambda d$, where d and λ are chosen according to a line search strategy that uses the directions d_T and/or d_N.
    (f) If x_{k+1} is an acceptable approximation to a root of $F(x)$, then stop and signal a success.

When referring to Algorithm 3.3 that uses a specific Krylov-based local solver in step 2d (i.e., the block-2, block-2+, or block-3 methods), we will abbreviate the method as TK2, TK2+, and TK3, respectively.

The main advantage of the tensor-Krylov method over other inexact tensor methods is that the inexact tensor step d_T satisfies the local tensor model to within the specified tolerance η_k. The tensor-GMRES method, on the other hand, computes the solution of a *projected* tensor model, which is missing second-order information in the direction of s, and may compute a less desirable step. The Bouaricha method [9] uses the exact model, but the relationship $\|M(x + d_T)\| < \eta_k \|F_k\|$ is not guaranteed, thereby raising the possibility of less accurate steps.

3.3.4.3 Global Strategy and Step Selection

Algorithm 3.3 needs a robust strategy for global convergence if neither the full tensor step nor the Newton step is satisfactory in step 2e. While step 2e uses a line search strategy, a trust region strategy is still viable, albeit less straightforward. Here we discuss details regarding a line search implementation in the tensor-Krylov method.

The standard tensor line search of [57] and the TENSOLVE line search of [9, 10] are straightforward applications of backtracking along the tensor step, if it is a descent direction, or otherwise along the Newton direction. The curvilinear line search implementation in [4] requires a little adaptation. The curvilinear step $d_T(\lambda)$ is the solution of the modified tensor model $\lambda F + Jd + \frac{1}{2}a(s^T d)^2$, where λ is the line search parameter. Thus, in the tensor-Krylov algorithm, the local tensor model is likewise changed and recomputed. Fortunately, the scalar λ is carried through the process in a straightforward manner, irrespective of method. Specific details are covered in [2] Because the curvilinear line search for tensor methods has posted encouraging results and has a nice theoretical basis, we will use this line search implementation in the tensor-Krylov algorithm.

It should be noted that other large-scale tensor methods, such as the tensor-GMRES method of Feng and Pulliam [24], could employ the curvilinear line search even though these other methods have subtle differences in calculating an inexact tensor step. This is because the curvilinear step is calculated from a simple scalar multiplication of the function value in the local tensor model and may be carried through the algebra of the step calculation to arrive at a parametric form of the curvilinear step.

3.4 Modified Bouaricha Method

Tensor methods have a reputation for being difficult to understand and cumbersome to implement. Our goal was to implement a tensor method for general use at Sandia that was easy to understand (as far as tensor methods go), computationally efficient, robust on a wide range of problems, capable of running in parallel, and capable of using preexisting linear solvers with all of their advanced features.

Our experience has shown us that we cannot use just any linear solver for the difficult applications at Sandia. Advanced linear solvers such as AZTEC [64] are needed. Unfortunately, these linear solvers are not easily modified and typically do not provide the Hessenberg matrices and other information needed by the tensor-Krylov and tensor-GMRES methods. More precisely, the tensor-Krylov methods [2, 3] require their own specialized Krylov-based solver, which prevents them from serious consideration. Tensor-GMRES [24], while more flexible with its linear solver requirements, still needs the Hessenberg matrix and orthogonal basis, which are usually not accessible from a standard interface. Bouaricha's method [9], on the the other hand, is capable of using any stand-alone linear solver, be it GMRES, TFQMR, BiCGSTAB, etc.

However, in experiments performed in [2], Bouaricha's method usually lagged in performance, sometimes by as much as twice the running time of the other algorithms. Thus, here we propose modifications to Bouaricha's method [9] that seek to address two aspects of the original algorithm. First, we address a theoretical deficiency that involves neglecting a complicated term for an exact step. Second, Bouaricha's method suffers from slow convergence due to the sequential solution of two linear systems per outer iteration, which we propose to address with block linear solvers. We describe both of these changes next.

Previously we saw that, when calculating the tensor step, Bouaricha's method neglects the complicated term in (3.11) involving $q(\beta_*)$,

$$\frac{(J_k^T J_k)^{-1} s_k}{s_k^T (J_k^T J_k)^{-1} s_k} q(\beta_*). \tag{3.24}$$

Bouaricha justifies this omission by noting that the term has a considerable computational cost for large-scale problems and that there appears to be no way to calculate it without having all of J_k available. While it can be proven that the step calculated without (3.24) still retains the same superlinear convergence properties of a standard tensor method, neglecting this term means that the resultant step is no longer a minimizer of the local tensor model. We believe there is a better approach than dropping (3.24).

Our modification to Bouaricha's algorithm approaches the problem from a practical standpoint. When the iterates are close to the solution, the tensor model in nearly all cases has a root. The nonroot cases usually occurs when the solver is far from the real solution and the solver is taking large steps. In this case, the second-order tensor term a_k, which is formed by secant approximation and depends on the step length, is most likely a poor approximation. Hence, the local model may deviate significantly from the actual function, and $M_T(x_k + d)$ may no longer have a root.

Intuition tells us that when the local model is significantly wrong, we should back off and be less aggressive in attempts at modeling the function around x_k. Thus, one of our modifications to Bouaricha's method attenuates the local tensor model by scaling the second-order information closer to a linear model (i.e., closer to Newton's method). We accomplish this by multiplying the tensor term $\frac{1}{2} a_k (s_k^T d)^2$ by a scalar parameter α:

$$M_T(x_k + d) = F_k + J_k d + \tfrac{1}{2} \alpha a_k (s_k^T d)^2.$$

This parameter is initially set to one, which implies solving the standard tensor model as before, but if a real root is not found, then we choose $\alpha \in (0,1]$ such that the model has a real root. This is done easily by forming the parameterized quadratic equation

$$q(\beta;\alpha) = \tfrac{1}{2} \alpha s_k^T J_k^{-1} a_k \beta^2 + \beta + s_k^T J_k^{-1} F_k. \tag{3.25}$$

If (3.25) does not have a real root for $\alpha = 1$, then the value of α that admits a single real root is

$$\alpha = \frac{1}{4(s_k^T J_k^{-1} a_k)(s_k^T J_k^{-1} F_k)}. \tag{3.26}$$

A second limitation of Bouaricha's original method is its performance—two linear systems are solved per iteration. As mentioned in section 3.3.2, a clever technique attempts to address this issue by using the previous Newton step as an initial guess to the second linear system, $J_k w = F_{k-1}$. In practice, this technique does not play a significant role until the iterates are very close to the solution, which leaves a lot of early iterations where the algorithm solves two linear systems with roughly equal work.

To address this, we propose a second modification of using a block linear solver to solve the two linear systems in tandem. For instance, a block-Krylov method [51] works with a block of initial vectors V to produce the block-Krylov subspace

$$span\{V, JV, J^2V, J^3V, \dots\}. \tag{3.27}$$

The block-Arnoldi process creates additional columns of the orthonormal basis for (3.27), with many versions available for implementation. The standard block-Arnoldi procedure works on a whole block $V \in \mathbb{R}^{n\times p}$ and adds p vectors—$p = 3$ in this case—to the subspace at a time.

Block solvers are more efficient with cache memory than scalar linear solvers solving multiple right hand sides [6]. The reason is that, as memory latency costs begin to dominate an algorithm, it becomes more expensive to retrieve a Jacobian matrix from memory. Block solvers take advantage of this by multiplying multiple vectors with the parts of the matrix that reside in cache memory. This extra computation is almost free compared to the time that the matrix would have to be fetched and pulled through cache memory in subsequent linear solves with a scalar method.

The option to use a block linear solver may or may not be available, depending on the availability of a library. The complete algorithm for solving the local model at each nonlinear iteration is presented below.

Algorithm 4.1: MODIFIED BOUARICHA METHOD

1. If a block solver is available, approximately solve the block system

$$J_k[d_N\, y] = -[F_k\, F_{k-1}]$$

for d_N and y using a block linear solver, such as block GMRES. Otherwise, solve each system independently using a standard linear solver. The initial guess

$$y_0 = -d_{N_{k-1}} = J_{k-1}^{-1}F_{k-1}$$

may be used for the second right hand side in the system.

2. Form the term $w = J_k^{-1}a_k$ by using the following relationship:

$$\begin{aligned} w &= J_k^{-1}a_k \\ &= \frac{2}{(s_k^T s_k)^2}(J_k^{-1}F_{k-1} - J_k^{-1}F_k - s_k) \\ &= \frac{2}{(s_k^T s_k)^2}(-y + d_N - s_k). \end{aligned}$$

3. Form the quadratic equation

$$q(\beta;\alpha) = \tfrac{1}{2}\alpha s_k^T J_k^{-1} a_k \beta^2 + \beta + s_k^T J_k^{-1} F_k \quad (3.28)$$
$$= \tfrac{1}{2}\alpha s_k^T w \beta^2 + \beta - s_k^T d_N, \quad (3.29)$$

with $\alpha = 1$ and find the smallest magnitude real root $q(\beta;\alpha)$. If a real root does not exist, then choose $\alpha \in (0,1]$ such that a single real root exists:

$$\alpha = \quad (3.30)$$

4. Calculate the tensor step

$$d_T = -J_k^{-1} F_k - \tfrac{1}{2} J_k^{-1} a_k \beta_*^2$$
$$= d_N - \tfrac{1}{2}\alpha w \beta_*^2,$$

where β_* is a real root of (3.29).

5. Select the next iterate x_{k+1} using a line search global strategy (as outlined in section 3.3.4.3).

3.5 Computational Results

This section describes numerical tests aimed at comparing the modified Bouaricha method with Newton-GMRES. We provide a limited comparison of the other methods and primarily focus on our modified Bouaricha method. Additional numerical tests comparing the other tensor methods may be found in [3, 2], and results on several ill-conditioned problems are included in [5].

3.5.1 Test Results on Fluid Flow Benchmark Problems

For an initial comparison, we consider a couple of CFD benchmark problems described in [62] that are used for verification of fluid flow codes and solution algorithms: the 2D backward-facing step problem and the 2D thermal convection problem.

We implemented the algorithms in a software package called NOX [41], which is a C++ object-oriented nonlinear solver package being developed at Sandia National Laboratories. For objective comparisons, all of the methods, including Newton-GMRES and tensor-GMRES, used the same Arnoldi process (modified Gram–Schmidt) as the tensor-Krylov method. That choice granted us more control over the algorithm and assured us of a controlled experiment. Thus, the results in this section do not reflect the most efficient and optimized implementations available.

We set up the numerical experiments to closely correspond to those in [62], using many of the same conditions and parameters. A successful termination was declared when both of the following stopping conditions were satisfied:

$$\|F(x_k)\| \leq \varepsilon_F \|F(x_0)\| \tag{3.31}$$

and

$$\frac{1}{\sqrt{n}} \|Wd_k\| < 1, \tag{3.32}$$

where n is the total number of unknowns, d_k is the full Newton or tensor step, and W is a diagonal scaling matrix with entries

$$W_{ii} = \frac{1}{\varepsilon_r |x_{k_i}| + \varepsilon_a},$$

in which x_{k_i} is the ith element of the current solution x_k. We used the same parameters as in [62]: $\varepsilon_F = 10^{-2}$, $\varepsilon_r = 10^{-3}$, and $\varepsilon_a = 10^{-8}$.

In practice, the step length criterion (3.32) is more stringent than (3.31) and is necessary to resolve finer details of the fluid flow and transport by requiring that each ith element of the Newton or tensor step be small relative to its current value x_{k_i}. All successful runs, except for two noted below in section 3.5.1.2, satisfied (3.31) with at least $\varepsilon_F = 10^{-8}$ and converged to the same solution. The last several iterations in each run were needed to satisfy (3.32).

If the test problem required more than 200 nonlinear (outer) iterations or if there was a line search failure (i.e., $f(x_k + \lambda d) \leq f(x_k) + \alpha\lambda\nabla f(x_k)^T d$, where $f(x) \equiv \frac{1}{2}\|F(x)\|_2$ and $\alpha = 10^{-4}$, could not be satisfied with $\lambda > 10^{-12}$ in at most 40 backtracks), then we declared a failure for the run.

The tests in [62] used a variety of forcing terms, particularly the adaptive forcing terms of Eisenstat and Walker [20]. As mentioned in section 3.3.4.3, more research is needed to determine how best to apply adaptive forcing terms to tensor methods. Consequently, we have used a constant forcing term of $\eta_k = 10^{-4}$ in the 2D problems and $\eta_k = 10^{-2}$ in the 3D problem. As in [62], we allowed the local solver (i.e., GMRES or its tensor-Krylov equivalent) a restart value of 200 with a maximum of 600 total iterations. If the local solver did not satisfy the desired tolerance within the 600 iterations, then we used the step computed thus far and tested for step acceptance with our global strategies. Restarting became more of an issue as the problem difficulty increased.

We used an explicit Jacobian, which our PDE code computed efficiently by a combination of analytic evaluation and numerical differentiation, and enabled the option for maximum accuracy in the Jacobian. We employed right preconditioning in all cases using an ILUT preconditioner [49], and we performed no variable or function scaling in the problems. The initial approximation was the zero vector for all cases.

While we recommended using a block linear solver for our modified Bouaricha method, one was not available for these experiments. Belos [22] is a suite of block solvers in the Trilinos software framework [34], but it was not interfaced to NOX in time for these experiments. Thus, we have tallied a worst-case type estimate for the method if a block linear solver were used. For each nonlinear iteration, we recorded the number of linear iterations for both linear solves. Then we took the larger of the two and added that to our running tally. We believe this is a reasonable estimate for two reasons. First, this number would be greater than or equal to the actual number of block iterations because the union of the two Krylov subspaces that solve the two right-hand sides in the scalar case is contained within the span of the block Krylov subspace generated in this many iterations. Second, the time per block linear iteration would be just slightly more than a scalar linear iteration due to the simultaneous computation of matrix-vector products as the matrix is pulled through cache memory only once [33].

We used a standard backtracking line search procedure for Newton-GMRES and used the complete tensor-GMRES algorithm in [24], including their globalization. For the tensor-Krylov methods, we used the curvilinear line search due to favorable theoretical and performance considerations in [4]. For selecting the line search parameter at each trial step, we used the λ-halving procedure (dividing λ by two at each inner iteration). Quadratic backtracking was an option but generally required more iterations and function evaluations than λ-halving across all methods in preliminary tests on these problems, so it was not used.

All single processor tests were performed on a dual 3GHz Pentium Xeon desktop computer with 2GB of RAM, which was more than sufficient for these problems. However, the computer was not dedicated to these tests, so the timing statistics provided are only approximate and could be off by 10% or more relative to each other. The parallel processor test was performed on a dedicated 16 node/32 processor cluster of 1 GHz Pentium III processors.

The fluid flow problems are set up using a particular spatial discretization of the governing steady-state transport equations for momentum and heat transfer in flowing fluids. These governing PDEs are given below. The unknown quantities in these equations are the fluid velocity vector ($\mathbf{u}$), the hydrodynamic pressure (P), and the temperature (T).

$$\text{Conservation of mass:} \quad \nabla \cdot \mathbf{u} = 0 \tag{3.33}$$

$$\text{Momentum transport:} \quad \rho\, \mathbf{u} \cdot \nabla \mathbf{u} - \nabla \cdot \mathbf{T} - \rho \mathbf{g} = 0 \tag{3.34}$$

$$\text{Energy transport:} \quad \rho C_p \mathbf{u} \cdot \nabla T + \nabla \cdot \mathbf{q} = 0 \tag{3.35}$$

In these equations, $\mathbf{g}$ is the gravity vector, and ρ and C_p are the density and specific heat at constant pressure of the bulk fluid, respectively. The constitutive equations for the stress tensor $\mathbf{T}$ and heat flux $\mathbf{q}$ are

$$\begin{aligned} \mathbf{T} &= -P\mathbf{I} + \mu(\nabla \mathbf{u} + \nabla \mathbf{u}^T), \\ \mathbf{q} &= -\kappa \nabla T, \end{aligned}$$

where μ is the dynamic viscosity and κ is the thermal conductivity of the fluid.

The particular spatial discretization of (3.33)–(3.35) that we use is from a finite element reacting flow code called MPSalsa [60] developed at Sandia. MPSalsa generates an algebraic system of equations by a pressure-stabilized Petrov–Galerkin finite element formulation of the low Mach number Navier–Stokes equations with heat transport. This scheme uses equal-order interpolation of velocity and pressure, and we enabled the option for streamline upwinding to limit oscillations due to high grid Reynolds numbers. Since the publication of [62], the pressure-stabilized streamline upwinding Petrov–Galerkin formulation in MPSalsa has been changed to a Galerkin least squares–type method [59]. This stabilization method is slightly less dissipative, and the nonlinear convergence behavior for difficult problems can be less robust at higher Reynolds numbers. Consequently, this change precludes direct comparisons with results in [62].

To complete a problem's specification, boundary conditions are imposed on the governing PDEs, which we discuss in the subsections that follow. The problems differ only in their boundary conditions and in whether they use (3.33)–(3.35) or only (3.33)–(3.34). The next two subsections describe the test problems and their results.

3.5.1.1 Thermal Convection Problem

This problem consists of the thermal convection (or buoyancy driven) flow of a fluid in a differentially heated square cavity in the presence of gravity. It requires the solution of (3.33)–(3.35) on the unit square with the following Dirichlet and Neumann boundary conditions:

$$T = T_{cold},\ \mathbf{u} = 0 \quad \text{at} \quad x = 0, \tag{3.36}$$

$$T = T_{hot},\ \mathbf{u} = 0 \quad \text{at} \quad x = 1, \tag{3.37}$$

$$\frac{\partial T}{\partial y} = 0,\ \mathbf{u} = 0 \quad \text{at} \quad y = 0, 1. \tag{3.38}$$

Once the governing equations and boundary conditions are nondimensionalized, two parameters appear: the Prandtl number (Pr) and the Rayleigh number (Ra). In our experiments, we fixed Pr = 1 and increased the Rayleigh number from Ra = 10^4 up to 2×10^7, which increases the nonlinear effects of the convection terms and makes the solution more difficult to obtain. The range in [62] is Ra = 10^3 to 10^6, but we shifted the range to explore the effectiveness of tensor methods on more difficult problems. We used a 100×100 equally spaced mesh, which has 40,804 unknowns. On this size mesh, it is unclear whether the choice of Ra $> 10^6$ admits a physically accurate and/or stable solution, but we are interested only in the relative performance of the numerical methods on this problem, which remain valid comparisons.

Figure 3.1 shows the overall performance of Newton-GMRES and all of the tensor methods on this problem. As the Rayleigh number increases, Newton-GMRES and tensor-GMRES require increasingly more iterations to solve the problem. For Ra $\geq 10^7$, Newton-GMRES fails to solve the problem in 200 iterations. Thus, the trend for Newton-GMRES is a clear degradation in performance as the problem becomes more difficult to solve. Tensor-GMRES is unable to solve the problem for Ra $\geq 8 \times 10^6$. In contrast, the other tensor methods (due in part to the curvilinear line search) are much less affected by the transition and see a much smaller increase in nonlinear iterations.

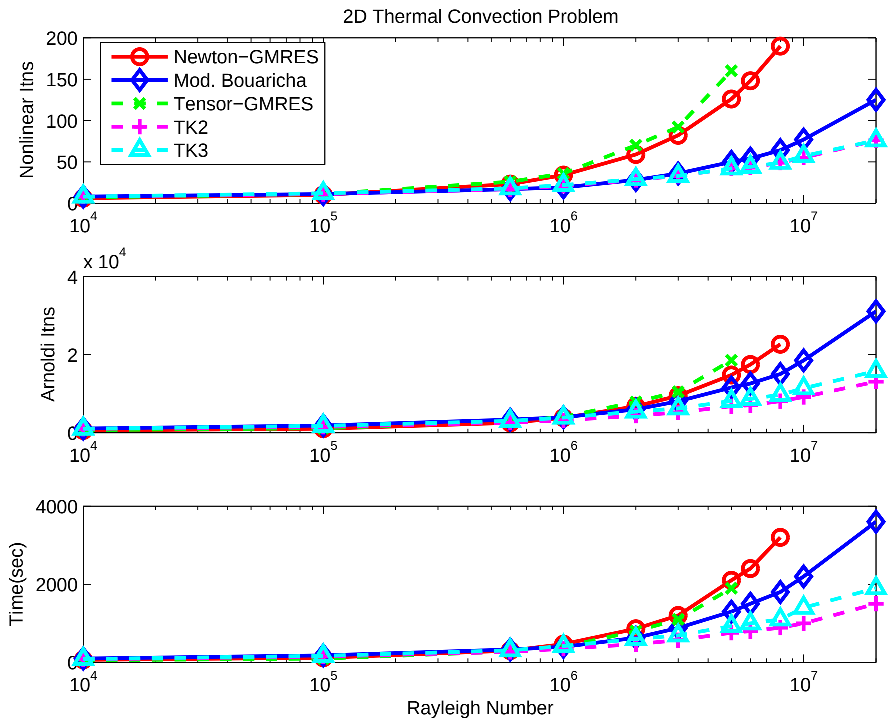

Figure 3.1. 2D Thermal convection problem results for Newton-GMRES and all tensor methods.

Figure 3.2 shows just the results for the modified Bouaricha method and Newton-GMRES. The middle plot shows the results of the modified Bouaricha method with a standard linear solver (scalar implementation rather than block). The results are less impressive but still better than Newton-GMRES on the most difficult problems. As seen in Figure 3.1, other tensor methods perform better in a scalar implementation due to solving only one system as opposed to two linear systems.

The bottom plot of Figure 3.2 shows the expected performance of a block linear solver implementation of the modified Bouaricha method. One block iteration (unit on the y-axis) is just slightly more expensive than an iteration from a scalar implementation. This is because the cost of multiplying an extra vector with a sparse Jacobian that already resides in cache memory is small compared to the cost of retrieving the matrix from main memory.

We show similar results in Figure 3.3 for results run on 8 processors of a parallel com-

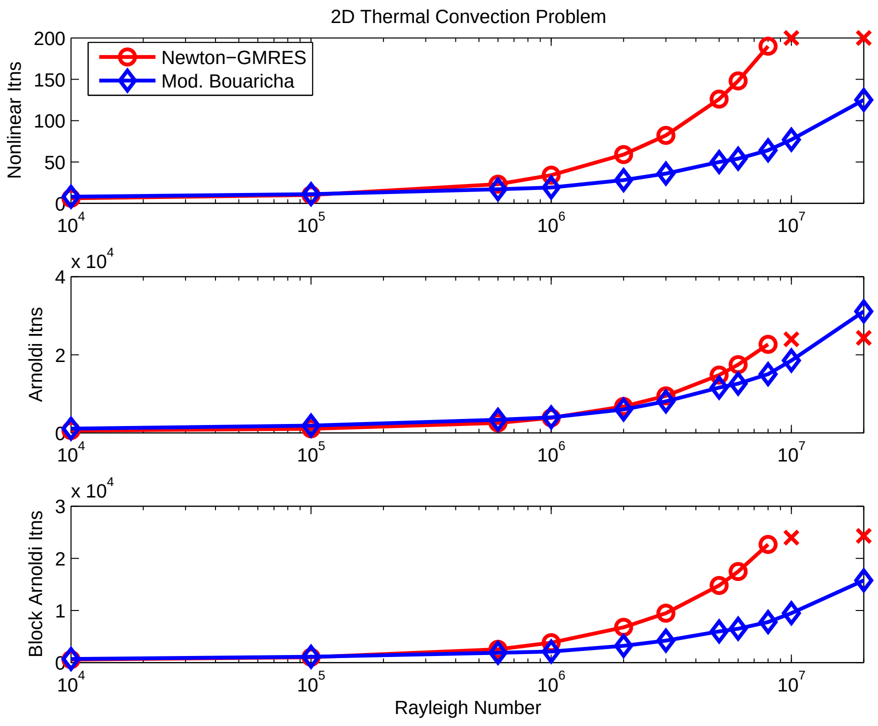

Figure 3.2. 2D Thermal convection problem results for Newton-GMRES (○) and the modified Bouaricha method (◇).

puter. The significance of this plot is that the other tensor methods (TK2, TK3, and tensor-GMRES) are not capable of running in parallel as they are implemented in NOX. Significant changes would be needed to convert the tensor-Krylov methods to run in parallel, and some customizations of the linear solver interface would be necessary to parallelize tensor-GMRES. Here the plots show the same behavior as in the single processor case—Newton-GMRES requires dramatically more iterations as the problem difficulty increases whereas the modified Bouaricha method is much less affected. In comparison to the single processor results, the modified Bouaricha method happened to display even better performance relative to Newton-GMRES.

3.5.1.2 Backward-facing Step Problem

This second fluid flow problem consists of a rectangular channel with a 1×30 aspect ratio in which a reentrant backward-facing step (i.e., a sudden expansion in the channel width) is simulated by injecting fluid with a fully developed parabolic velocity profile in the upper half of the inlet boundary and imposing a zero velocity on the lower half. The channel geometry and flowing fluid produce recirculation zones beneath the entering flow on the

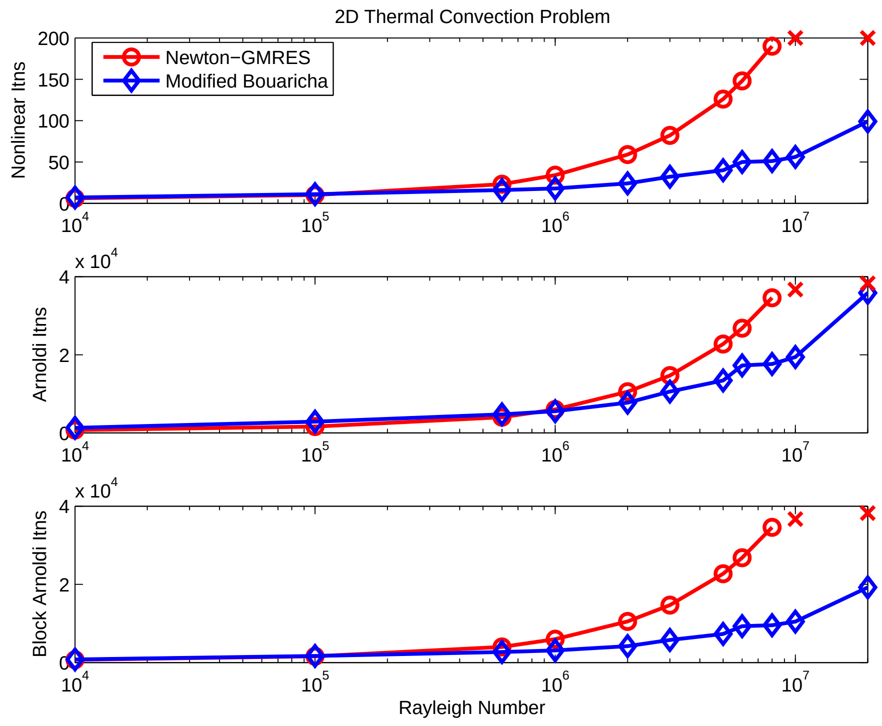

Figure 3.3. 2D Thermal convection problem results for Newton-GMRES (○) and the modified Bouaricha method (◇) on 8 processors of a parallel computer.

lower wall and, for sufficiently fast flow, farther downstream on the upper wall. This problem requires the solution of (3.33)–(3.34) on the unit square with the following Dirichlet boundary conditions:

$$\begin{aligned}
\mathbf{u} = 24y(\tfrac{1}{2} - y)U_0\hat{x} \quad &\text{at} \quad x = 0,\ 0 \le y \le \tfrac{1}{2},\\
\mathbf{u} = 0 \quad &\text{at} \quad x = 0,\ -\tfrac{1}{2} \le y < 0,\\
\mathbf{u} = 0 \quad &\text{at} \quad y = -\tfrac{1}{2}, \tfrac{1}{2},\\
\mathbf{T}_{xx} = \mathbf{T}_{xy} = 0 \quad &\text{at} \quad x = 30,
\end{aligned}$$

where $\hat{x}$ is the unit vector in the x-direction. Once the governing equations and boundary conditions are nondimensionalized, the Reynolds number (Re) appears, which is a measure of inertial forces to viscous forces. In our experiments, we increased the Reynolds number up to 800, which increases the nonlinear inertial terms in the momentum equation and makes the solution more difficult to obtain. Beyond Re = 800, it is not clear that the problem is stable and admits a physical solution. All solutions for this problem were computed on a

20×400 unequally spaced mesh, which has 25,263 unknowns. All methods converged to the same point except for the case Re = 400 and so was not included in the results.

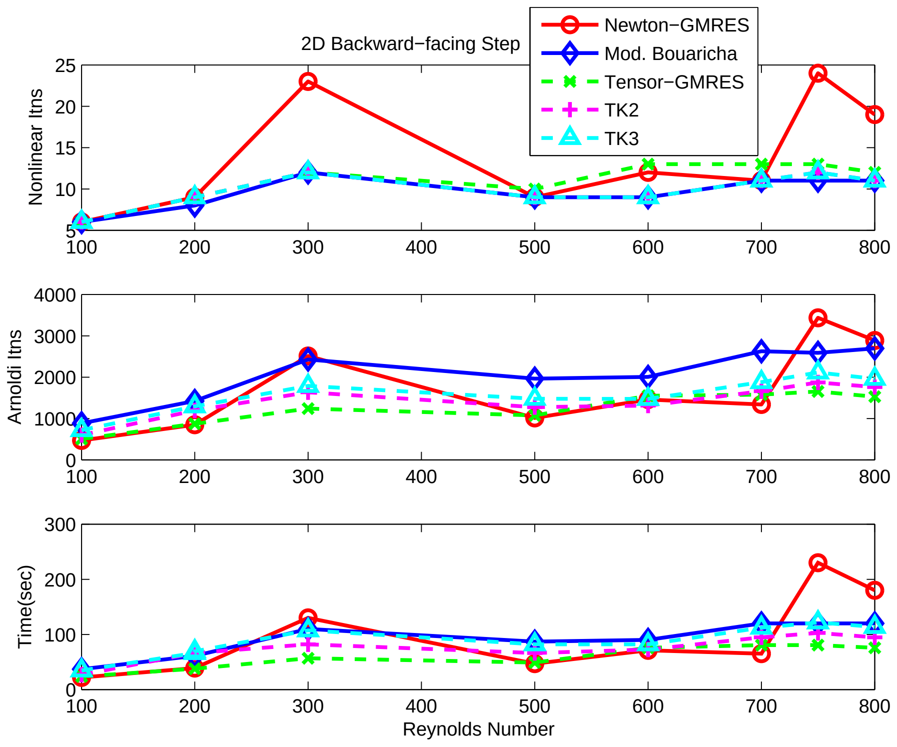

Figure 3.4. 2D Backward-facing step convection problem results for Newton-GMRES and all tensor methods.

The plots in Figure 3.4 show that all of the methods require about 10–12 nonlinear iterations, on average, to solve, with Newton-GMRES requiring considerably more iterations in some cases. Newton's method tended to be more erratic, having slight difficulty at Re = 300, improvements at Re = 500 and 600, and then more difficulty on the three hardest problems. The two Newton solutions at Re = 700 and 750 actually converged to a local minimizer of the line search merit function yet still satisfied the relative residual reduction criterion of 10^{-2}. If ε_F in (3.31) were 10^{-3}, then these two runs would have been line search failures.

Figure 3.5 shows just the results for Newton-GMRES and the modified Bouaricha method. The bottom plot shows the predicted performance for a block implementation of the modified Bouaricha method, which is rather favorable and comparable to the performance of the other tensor methods.

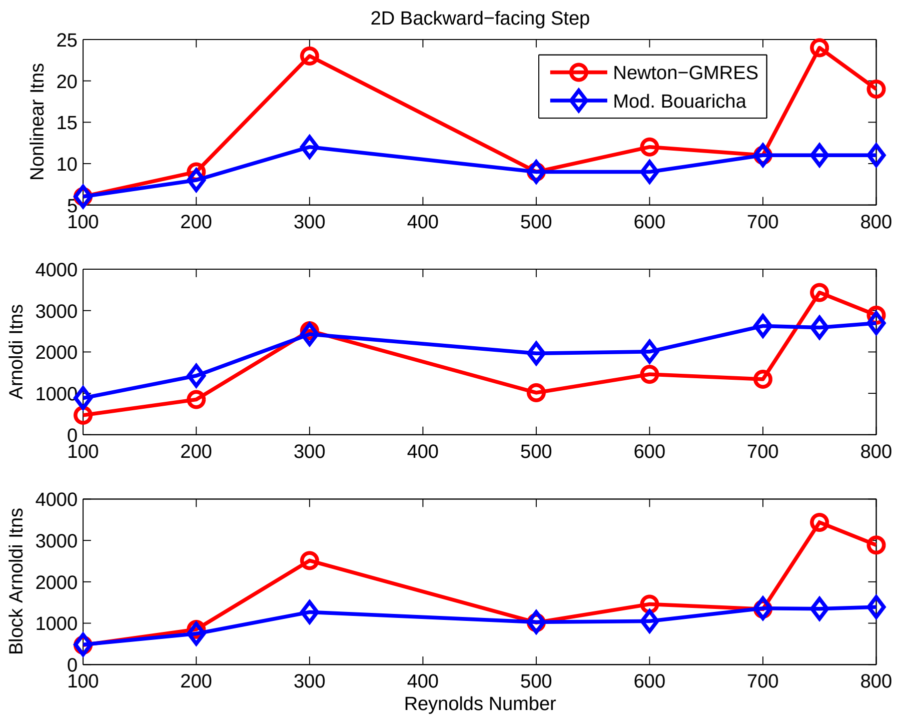

Figure 3.5. 2D Backward-facing step convection problem results for Newton-GMRES (○) and the modified Bouaricha method (◇).

3.5.2 Test Results on a Circuit Simulation Problem

For a numerical experiment from a different problem domain, we considered an elementary circuit problem from Xyce [36], which is a circuit simulator being developed at Sandia. We investigated the performance of the algorithms on a comparator circuit. The switches in the circuit cause strong nonlinearities and in some case near discontinuities, all of which are difficult for Newton-based solvers to handle. In particular, the Jacobian can be very ill-conditioned or singular during an iteration. Some linear algebra techniques, such as singleton elimination, help address these issues, but the nonlinear solver still faces some difficulties. For example, the computed Newton step still can be very long in the direction of the singular vectors corresponding to the smallest singular values.

Due to the second-order information in the local model at each step, tensor methods tend to have less difficulty with singular and ill-conditioned Jacobian matrices. Most often the secant approximation contains information that is missing in the (near) null space of the Jacobian. This prevents the nonlinear step from growing too large and requiring cutbacks in the line search routine. While ill-conditioned and singular matrices in tensor methods may be handled easily using a technique described in section 4.1.1, the technique was not implemented here. Instead, we tested the pure performance of a tensor method versus Newton's method.

Because circuit simulation problems are very difficult for standard nonlinear solvers, tuning is necessary to avoid catastrophic failures and/or achieve satisfactory performance. Because tuning is usually problem specific, it is not known *a priori* which set of parameters will work the best on a particular problem. Thus, we chose the conditions of this experiment to mimic this idea of parameter tuning. One often tries different global strategies and/or parameters to see what works best in a trial-and-error fashion. Here we run each algorithm with the same solver options and parameters (linear solver tolerance, full step vs. global strategy, etc.) and compare their performance for that run only. Because different parameter sets affect the algorithms differently, we are looking for the algorithm that has the best average performance over all parameter sets.

Figure 3.6 shows a binary comparison over fifteen runs with different algorithmic tuning parameters. The same set of parameters are used for both algorithms for that particular run. In the plot we record the logarithm (base 2) of the ratio of nonlinear iterations of Newton to tensor. The resulting plot shows which method performed better and by how much. Blue bars extending above zero show the extent to which the tensor method was better. On average, the modified Bouaricha method converged in 39% fewer nonlinear iterations than Newton-GMRES.

Figure 3.7 takes the same results and shows the binary comparison for execution time. On average, the modified Bouaricha method converged in 10% less time than Newton-GMRES. This is due once again to the fact that two linear systems are being solved per nonlinear iteration in the Bouaricha method. If a block linear solver were used to solve the two linear systems simultaneously, then each nonlinear iteration would take just slightly more time than Newton's method, and this plot would look more like Figure 3.6.

3.6 Discussion

The results of the fluid flow problems are encouraging for our modified Bouaricha tensor method. They show a preliminary indication that the modified Bouaricha method is more robust and oftentimes more efficient than standard Newton-GMRES. In general, the tensor method would perform on par with Newton-GMRES on the easier problems (lower Rayleigh or Reynolds numbers), but as the difficulty increased, the modified Bouaricha method would outperform Newton-GMRES. This general trend is seen also with the other tensor methods to varying degrees; see [3].

Part of the success is due to the curvilinear line search, which produces a trial step that has a direction in the span of the Newton step and tensor step and that reduces to the Newton direction as $\lambda \to 0$, which is a descent direction on the merit function. In a separate study of line search procedures for tensor methods [4], it was found that the curvilinear line search

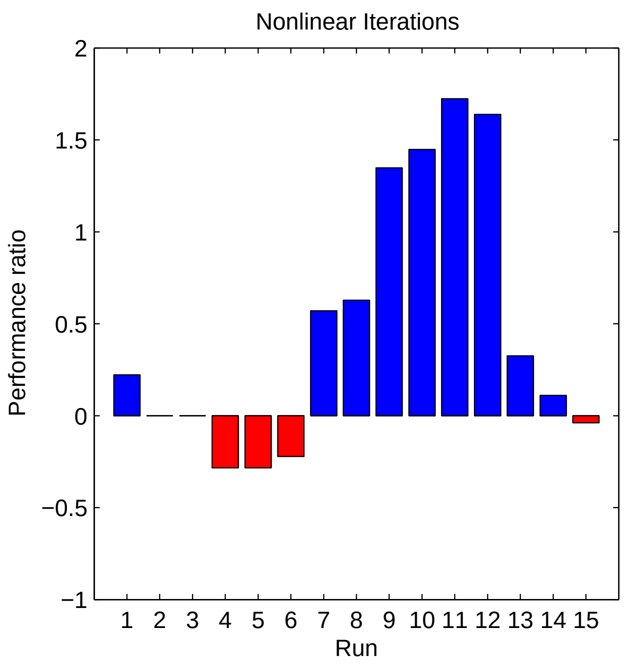

Figure 3.6. Log-ratio of the number of nonlinear iterations on the Xyce comparator circuit problem for the following methods: Newton-GMRES and the modified Bouaricha method. Red bars extending below zero favor Newton-GMRES and blue bars extending above zero favor the tensor method.

was the most robust and efficient. In many of the numerical experiments, Newton-GMRES had several line search failures, which are more catastrophic than exceeding the maximum iteration limit because the algorithm cannot make forward progress. The second-order information in the tensor model tends to provide a correction to the Newton step in difficult areas, such as bogging down in a line search, that helps the tensor method "escape" faster. Coincidentally, the accuracy of this second-order information increases during difficult regions because, as the line search cuts the step back, the secant approximation in the tensor method becomes more accurate.

Tensor methods are especially effective at solving large-scale problems that possess Jacobians at the solution that are highly ill-conditioned or singular. For theoretical reasons, algorithms based on Newton's method exhibit very slow convergence on such problems. In the next chapter, we discuss a technique that allows tensor methods to deal with highly ill-conditioned or singular Jacobians during the iterative process.

From our experience with Xyce, tensor methods perform modestly on circuit simulation problems, but our observation is that tuning algorithm parameters is a key ingredient to success on these types of problems. In addition, very ill-conditioned and singular Jacobian

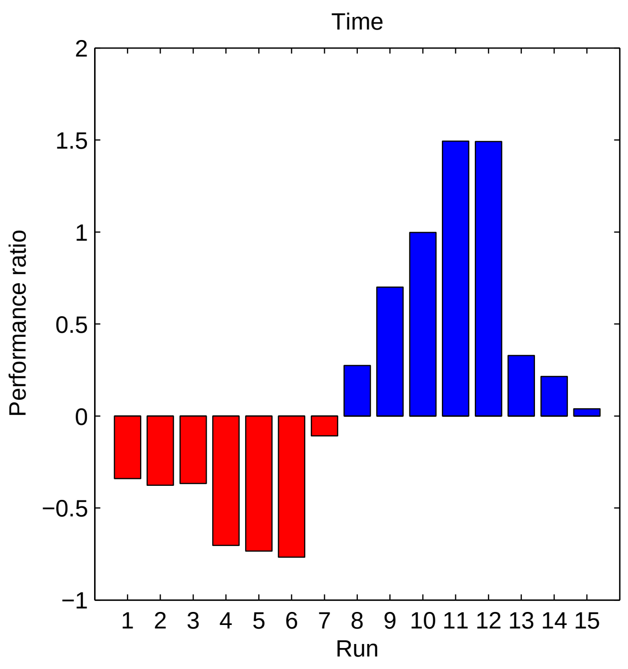

Figure 3.7. Log-ratio of execution time on the Xyce comparator circuit problem for the following methods: Newton-GMRES and the modified Bouaricha method. Red bars extending below zero favor Newton-GMRES and blue bars extending above zero favor the tensor method.

matrices frequently turn up in these problems. Ultimately, this causes problems for Newton's method, but it also caused problems for our implementation of Bouaricha's method. The reason is that we were still solving two linear systems with this matrix. When the Jacobian was ill-conditioned, then the two solutions were accurate only to a limited number of digits (which depends on the condition number). The tensor step is formed from a linear combination of these two vectors of limited accuracy, and so some of the significant digits may be lost, yielding a potentially spurious step.

The next chapter discusses a technique to avoid such difficulties associated with ill-conditioned and (mildly) singular matrices. Unfortunately, we were not able to implement this technique for testing on this class of difficult problems. We would expect to see better performance provided that the rank of the Jacobian is (or near) $n-1$. If the rank deficiency is greater, then a higher-order tensor term should be computed to account for the extra missing information in the null space. Following the paradigm outlined in the next chapter, additional rank-1 matrices may be added to the Jacobian to make it well-conditioned. With this approach, we would expect to see a more robust and stable algorithm for dealing with circuit simulation problems.

There are several research questions left at this point to explore. We mention two future extensions here. First, adaptive forcing terms like the form by Eisenstat and Walker [20] may help improve robustness. Second, there is still the question of accuracy of β as a function of linear solver tolerance η. If the two linear subproblems are solved very approximately, then it is possible that a large value of β may be calculated, which could unduly bias the tensor correction and yield a step far from the Newton direction.

Chapter 4

Turning Point Algorithm

In this chapter, we present research on a modified turning point algorithm that works with ill-conditioned and singular Jacobians. We first describe an approach that tensor methods may use to deal with singular and ill-conditioned Jacobian matrices. We then extend that idea to a general linear system and apply it to a turning point identification algorithm.

To be consistent with previous chapters, we will keep the subscript k when discussing the tensor model at iteration k in the nonlinear solver. Later in this chapter when we are dealing strictly with a local model at the same iteration k, we will drop the subscript k on J_k, F_k, a_k, and s_k.

4.1 Singular and Ill-conditioned Jacobians

4.1.1 Tensor Methods

Tensor methods [57] incorporate second-order information in a localized model about some current point x_k:

$$M_T(x_k + d) = F_k + J_k d + \tfrac{1}{2} T_k dd, \tag{4.1}$$

where $T_k \in \mathbb{R}^{n \times n \times n}$ holds some second-order information computed from a secant approximation; see section 3.2.1. Unlike local models that are strictly linear, (4.1) can still have a single real root even if the Jacobian is singular, provided that T_k contains information that is missing in the null space of J_k. The standard approach for solving the model in this case is to use orthogonal transformations, as outlined in section 3.2.1.

Briefly, the solution of a rank-1 tensor model can be reduced to the solution of a system of q quadratic equations in 1 unknown, plus the solution of a system of $n - q$ linear equations in $n - 1$ unknowns; see [57]. Most often $q = 1$, but a singular Jacobian with a rank deficiency greater than one will make $q > 1$. In this case, the number of linear equations is decreased by this rank deficiency from the normal case, and the number of quadratic equations is increased correspondingly, leaving the number of variables in each system unchanged.

The other approach for solving a rank-1 tensor model, what we will call the "reduction method" and is used in Bouaricha's method [9], solves two linear systems with J_k to form a reduced system of quadratic equations. Clearly, if J_k is singular, then the linear systems are unsolvable and this method breaks down.

We outline an approach that makes the reduction method feasible for a singular (or ill-conditioned) Jacobian under mild conditions. Once again, we will concentrate on the rank-1 tensor model formed by a secant approximation as derived in section 3.2.1:

$$M_T(x_k+d) = F_k + J_k d + \tfrac{1}{2}a_k(s_k^T d)^2. \tag{4.2}$$

In the nonsingular case, the solution to (4.2) is found by solving two linear systems with F_k and a_k (or F_{k-1} as we have shown before) as right hand sides. Then we form the quadratic equation

$$q(\beta) \equiv s_k^T J_k^{-1} F_k + \beta + \tfrac{1}{2} s_k^T J_k^{-1} a_k \beta^2$$

solve for the smallest magnitude real root, which we call β_*. For this section, we consider only the case of real roots; see section 3.2.1 for the minimizing case. Then the solution to (4.2) is

$$d_T = -J_k^{-1} F_k - \tfrac{1}{2} J_k^{-1} a_k \beta_*^2.$$

Clearly, this method breaks down when J_k is singular, but there can be problems when J_k is ill-conditioned, too. These problems are caused by the accuracy of the two linear solutions and the fact that they are added together. If they are of opposite signs, then there may be cancellation of the accurate digits, leaving one with fewer (or no) accurate digits.

The approach to solving (4.2) when J_k is singular or ill-conditioned deals with modifying the system such that the underlying system is unchanged. We start by adding a clever form of zero, $[a_k s_k^T d - a_k s_k^T d]$, and rearrange terms.

$$\begin{aligned}
0 &= F_k + J_k d + \tfrac{1}{2}a_k(s_k^T d)^2 \\
0 &= F_k + J_k d + \tfrac{1}{2}a_k(s_k^T d)^2 + [a_k s_k^T d - a_k s_k^T d] \\
0 &= F_k + (J_k + a_k s_k^T)d + \tfrac{1}{2}a_k(s_k^T d)^2 - a_k s_k^T d \\
0 &= F_k + \tilde{J}_k d + \tfrac{1}{2}a_k(s_k^T d)^2 - a_k s_k^T d \qquad (4.3)
\end{aligned}$$

The matrix $\tilde{J}_k \equiv (J_k + a_k s_k^T)$ is the original Jacobian plus a rank-1 matrix, $a_k s_k^T$. It is this rank-1 matrix that makes $\tilde{J}_k$ better conditioned than J_k. We will comment later on conditions that make it so.

We continue by noting that (4.3) may be simplified further by "completing the square" of the last two terms. By adding another clever form of zero, this time $[\frac{1}{2}a_k - \frac{1}{2}a_k]$, the last two terms may be collected and written as a square:

$$\begin{aligned}
0 &= F_k + \tilde{J}_k d + \tfrac{1}{2} a_k (s_k^T d)^2 - a_k (s_k^T d) + [\tfrac{1}{2} a_k - \tfrac{1}{2} a_k], \\
0 &= F_k - \tfrac{1}{2} a_k + \tilde{J}_k d + \tfrac{1}{2} a_k (s_k^T d)^2 - a_k (s_k^T d) + \tfrac{1}{2} a_k, \\
0 &= \tilde{F}_k + \tilde{J}_k d + \tfrac{1}{2} a_k (s_k^T d - 1)^2. \qquad (4.4)
\end{aligned}$$

Equation (4.4) is identical to our original system (4.2) because we have added nothing but zeros to the system. Indeed, using these definitions,

$$\begin{aligned}
\tilde{J}_k &\equiv J_k + a_k s_k^T, \\
\tilde{F}_k &\equiv F_k - \tfrac{1}{2} a_k, \\
\tilde{\beta}_* &\equiv s_k^T d_T - 1,
\end{aligned}$$

the model in (4.2) may be written as

$$M_T(x_k + d) = \tilde{F}_k + \tilde{J}_k d + \tfrac{1}{2} a_k \tilde{\beta}^2. \qquad (4.5)$$

Because (4.5) is identical to (4.2) in form, we may solve (4.5) using the reduction method. That is, we form the quadratic equation

$$q(\tilde{\beta}) \equiv s_k^T \tilde{J}_k^{-1} \tilde{F}_k + \tilde{\beta} + \tfrac{1}{2} s_k^T \tilde{J}_k^{-1} a_k \tilde{\beta}^2$$

and find its smallest magnitude real root $\tilde{\beta}_*$. Then the root of both (4.2) and (4.5) is given by

$$d_T = -\tilde{J}_k^{-1} \tilde{F}_k - \tfrac{1}{2} \tilde{J}_k^{-1} a_k \tilde{\beta}_*^2.$$

We now discuss the conditions that make $\tilde{J}_k$ have a lower condition number than J_k. We start our analysis with the singular value decomposition (SVD) of J_k. Let $J_k = U \Sigma V^T = \sum_{i=1}^n u_i \sigma_i v_i$, where U and V are unitary matrices with orthogonal columns and Σ is a diagonal matrix of singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$. The condition number of J_k is given by $\kappa(J_k) = \frac{\sigma_1}{\sigma_n}$. Thus, if the smallest singular value σ_n is zero, then the matrix is singular and $\kappa(J_k) = \infty$.

The rank-1 matrix $a_k s_k^T$ that is added to J_k will add some amount to one or more of the singular values of J_k. If a_k is a multiple of u_n and s_k is a multiple of v_n, then the contribution is purely in the last singular value and the condition number of

$$\tilde{J}_k = \frac{\max(\sigma_1, u_n^T a_k s_k^T v_n)}{\min(\sigma_{n-1}, u_n^T a_k s_k^T v_n)}.$$

In practice, it is never the case that a_k is a multiple of u_n, but there is some theory for s_k.

An ideal tensor method [23] would construct the tensor term T_k from exact second derivatives as well as the null space of J_k. More precisely, s_k would correspond to v_n, the least singular vector of J_k, and a_k would be $F''(x_k)v_nv_n$. Normally, this is too expensive to compute so a practical approach uses a secant approximation to obtain s_k and a_k. The choice of $s_k = x_k - x_{k-1}$ approximately lies in the direction of v_n because the difference between two consecutive iterates is likely to be along the null space when consecutive iterates are in the funnel around the null space near the solution [23, 24]. Close to the solution, the vector $F''(x_k)v_nv_n$ is likely to be approached by our a_k in (3.4).

Thus, one approximately has s_k proportional to v_n but no such condition on a_k. Provided that $u_n^T a_k s_k^T v_n \neq 0$, then the condition number of $\tilde{J}$ will no longer be infinite. In effect, the rank-1 matrix boosts the smallest singular value of J_k, which makes $\tilde{J}_k$ nonsingular. For ill-conditioned matrices, it is likely, but not guaranteed, that the condition number will improve.

4.1.2 General Linear Systems

We now apply this technique to general linear systems that are very ill-conditioned. In keeping with our notation from all previous sections on nonlinear equations, where the matrix is a Jacobian, we write the linear system as $Jx = b$ instead of the more traditional $Ax = b$. For this discussion we assume that $J \in \mathbb{R}^{n\times n}$ is either singular with rank $n-1$ or very ill-conditioned with the smallest singular value $\sigma_n \ll \sigma_{n-1}$. However, the approach outlined here is generalizable to a larger rank-deficiency.

The approach to solving (4.2) when J is singular or ill-conditioned is similar to the approach above. It is actually a variation of the Sherman-Morrison-Woodbury formula [27]. A similar but unrelated idea is contained in the Interlocking Eigenvalue Lemma due to Loewner (stated in [42]), which relates the eigenvalues of a symmetric matrix A to the eigenvalues of A plus a rank-1 matrix $e_ie_i^T$, where e_i is the ith unit vector.

We start by choosing two vectors $u, v \in \mathbb{R}^n$ such that their inner product with the smallest left and right singular vectors of J (resp.) is nonzero. For simplicity we'll assume that u, v have unit length. Ideally we would like $u = u_n$ and $v = v_n$. We also choose a scale factor α and form the rank-1 matrix αuv^T. Then we take its product with the unknown x, add it to both sides of the linear equation, rearrange terms, and solve for x:

$$\begin{aligned}
Jx &= b, \\
Jx + \alpha uv^T x &= b + \alpha uv^T x, \\
(J + \alpha uv^T)x &= b + \alpha uv^T x, \\
\tilde{J}x &= b + \alpha uv^T x, \\
x &= \tilde{J}^{-1}b + \alpha \tilde{J}^{-1}u(v^T x).
\end{aligned}$$

Because uv^T provides information in the smallest singular vectors of J, the matrix $\tilde{J} \equiv (J + \alpha uv^T)$ is nonsingular and hopefully better conditioned than J. Later, we will discuss issues for choosing α, u, and v to make $\tilde{J}$ better conditioned than J.

Now, in a manner similar to the reduction method for solving a rank-1 tensor model, we take the inner product of both sides with v and solve for the unknown $\beta \equiv v^T x$,

$$\begin{aligned} v^T x &= v^T \tilde{J}^{-1} b + \alpha v^T \tilde{J}^{-1} u (v^T x) \\ \beta \equiv v^T x &= \frac{v^T \tilde{J}^{-1} b}{1 - \alpha v^T \tilde{J}^{-1} u} \end{aligned}$$

Finally, the solution to the linear system is given by

$$x = \tilde{J}^{-1} b + \alpha \tilde{J}^{-1} u \beta \tag{4.6}$$

where

$$\begin{aligned} \tilde{J} &\equiv J + \alpha u v^T, \\ \beta &\equiv \frac{v^T \tilde{J}^{-1} b}{1 - \alpha v^T \tilde{J}^{-1} u}. \end{aligned}$$

Some comments are in order. First, as mentioned above, the vectors u and v must not be orthogonal to the smallest singular vectors of J. If they are orthogonal, then $\tilde{J}$ is no better conditioned than J. Second, it is best to have u and v close to u_n and v_n. Third, the scaling factor α is used to scale the contribution of this matrix so that the smallest singular value of J is boosted between σ_1 and σ_{n-1}. Fourth, this technique cannot be used to solve $Jx = b$ if J is singular. For such a system, there is either no solution or an infinite number of solutions. However, even if b is in the range of J, the denominator of β would be zero. Last, this technique involves the solution of two linear systems with the same matrix $\tilde{J}$ but different right hand sides, b and u. The cost associated with solving two systems may be amortized over a single loop with block linear solvers.

The benefit of this technique is that many iterative methods are adversely affected by a large condition number of the matrix. Hence, preconditioning techniques have been developed to address these issues with linear systems and to help speed up convergence. This technique is a different approach that may address many of the same issues.

4.2 Turning Point Identification

We turn our attention to an application of this technique for solving ill-conditioned linear systems. In particular, the identification of turning points in physical systems involves finding the point on a curve of a family of solutions to a physical problem such that the Jacobian matrix has rank $n-1$.

More precisely, we are interested in a general physical problem described by the system of nonlinear equations $F(x, \lambda)$, where x is the state vector and λ is a parameter. We may trace out a family of solutions to x for a range of values of λ. If the curve turns back on itself, then there is a critical value λ_* that has only one solution. On one side of λ_* there are two solutions and on the other side there are no solutions to $F(x,\lambda)$. The point λ_* is called a turning point, and the Jacobian at this point is singular with rank deficiency of one.

A system of nonlinear equations that explicitly defines a turning point may be written down and solved:

$$\begin{aligned} F(x,\lambda) &= 0, & (4.7)\\ Jv &= 0, & (4.8)\\ \phi^T v &= 1. & (4.9)\end{aligned}$$

The first equation is the state equation of the physical problem. The second equation specifies that the Jacobian of $F(x,\lambda)$ with respect to x is singular with a null vector v. The last condition is a scalar equation to keep v away from the trivial solution of $v = 0$. This is a system of $2n+1$ equations, which may be solved by Newton's method:

$$\begin{pmatrix} J & 0 & \frac{\partial F}{\partial \lambda} \\ \frac{\partial (Jv)}{\partial x} & J & \frac{\partial (Jv)}{\partial \lambda} \\ 0 & \phi^T & 0 \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta v \\ \Delta \lambda \end{pmatrix} = - \begin{pmatrix} F \\ Jv \\ \phi^T v - 1 \end{pmatrix} \tag{4.10}$$

4.2.1 Bordered Algorithm

In practice, it is difficult to solve (4.7)–(4.9) strictly by Newton's method because some entries of the full Jacobian matrix are difficult to calculate. We discuss a practical algorithm in LOCA [54, 55] for solving this system. It is called a bordered algorithm, and it may be derived from a block elimination procedure. We provide a brief outline of the algorithm here; for more details see [54, 55];

The bordered algorithm involves four intermediate solves with the Jacobian matrix:

$$\begin{aligned} Ja &= -F, \\ Jb &= -\frac{\partial F}{\partial \lambda}, \\ Jc &= -\frac{\partial (Jv)}{\partial x} a, \\ Jd &= -\frac{\partial (Jv)}{\partial x} b - \frac{\partial (Jv)}{\partial \lambda}. \end{aligned}$$

Then we may compute the Newton step with the four solutions to a, b, c, and d:

$$\begin{aligned}
\Delta\lambda &= \frac{1-\phi^T c}{\phi^T d} \\
\Delta x &= a + \Delta\lambda b \\
\Delta v &= c + \Delta\lambda d - v
\end{aligned}$$

The Jacobian J grows more ill-conditioned and eventually singular as we approach the turning point, but the $(2n+1)\times(2n+1)$ "system" Jacobian in (4.10) remains nonsingular. Because there are four solves with an ill-conditioned matrix that is being driven to a singularity, the linear combination of these four solutions to form the Newton step would be increasingly inaccurate.

4.2.2 Modified Bordered Algorithm

To address this issue of an increasingly ill-conditioned system, we propose solving the four linear systems using the approach outlined in section 4.1.2. At the current stage of this research, this technique only works with an iterative solver that solves each system only approximately.

The approach involves adding a nonzero vector to both sides of the Newton equation to facilitate a way to improve the conditioning of the matrix (the Jacobian) appearing in the linear subproblems:

$$\begin{pmatrix} uv^T\Delta x \\ uv^T\Delta v \\ 0 \end{pmatrix} + \begin{pmatrix} J & 0 & \frac{\partial F}{\partial \lambda} \\ \frac{\partial(Jv)}{\partial x} & J & \frac{\partial(Jv)}{\partial \lambda} \\ 0 & \phi^T & 0 \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta v \\ \Delta\lambda \end{pmatrix} = -\begin{pmatrix} F \\ Jv \\ \phi^T v - 1 \end{pmatrix} + \begin{pmatrix} uv^T\Delta x \\ uv^T\Delta v \\ 0 \end{pmatrix}. \qquad (4.11)$$

The solution to this problem is the same as if it were calculated by standard Newton's method.

Using the same bordering algorithm approach, we may solve (4.11) by solving smaller subproblems with the same matrix. That is, we solve the four linear systems

$$\begin{aligned}
\tilde{J}a &= -F + uv^T a \\
\tilde{J}b &= -\frac{\partial F}{\partial\lambda} + uv^T b \\
\tilde{J}c &= -\frac{\partial(Jv)}{\partial x}a + uv^T c \\
\tilde{J}d &= -\frac{\partial(Jv)}{\partial x}b - \frac{\partial(Jv)}{\partial\lambda} + uv^T d
\end{aligned}$$

for a, b, c, d, where $\tilde{J} \equiv J + uv$. In this case we let v equal the current estimate of the null space vector being solved for in the turning point problem, and we choose u to be a scaled approximation to the smallest left singular vector of J. We use the approximation $u = \alpha Jv / \|Jv\|$, where α is a scaling factor to make the condition number of $\tilde{J}$ be acceptable. The actual value of α is not important so long as it is not too high and not too low. Ideally, α should be between the largest singular value of J and second smallest singular value of J, i.e., $\sigma_1 > \alpha > \sigma_{n-1}$.

We point out that for each linear system, the solution is implicitly defined, and one must calculate its solution in the manner of section 4.1.2. That is, to solve for a, b, c, d, one also needs the solution to $J^{-1}u$, or at least an approximation. Thus, this approach requires five linear solves per Newton step.

Once the solutions for a, b, c, d have been computed, one forms the Newton step in the standard way:

$$\begin{aligned}
\Delta\lambda &= \frac{1 - \phi^T c}{\phi^T d}, \\
\Delta x &= a + \Delta\lambda b, \\
\Delta v &= c + \Delta\lambda d - v.
\end{aligned}$$

As a final remark, we consider why this technique works. We mentioned in the beginning that this technique is only useful with iterative solvers that approximately solve a system to a specified tolerance. It has not been fully analyzed why this is the case, but it is related to the solver tolerance. The solution to each linear system computed by the modified method is no more accurate than a standard solution. Thus, as the solver approaches the turning point, each individual solution to a, b, c, d is increasingly less accurate. However, the contribution by $\tilde{J}^{-1}u$ is computed separately and used in each system. Hence, when $\Delta\lambda, \Delta x$, and Δv are computed, it is believed that the numerical error related to the null space direction cancels out, leaving an accurate Newton step.

4.3 Computational Results

This section describes a numerical experiment aimed at comparing the modified turning point method with the original bordering method used in LOCA [54, 55]. We use both approaches to solve for the turning point in the Bratu problem. The conclusions from these results extend to other types turning point identification problems that we tested. We performed all experiments in MATLAB.

The Bratu problem is a simplified model for nonlinear diffusion phenomena occurring, for example, in semiconductors and combustion, where the source term is related to the Arrhenius law for modeling exothermic reactions. The following version is taken from the set of nonlinear model problems collected by Moré [43]. The problem is the nonlinear partial differential equation in u

$$-\nabla^2 u = \lambda e^u \text{ in } \Omega, \quad u = 0 \text{ on } \partial\Omega, \tag{4.12}$$

where $\nabla^2 = \sum_{i=1}^{n} \partial^2/\partial x_i^2$ is the Laplace operator, $\lambda \in \mathbb{R}$ is a parameter, Ω is the bounded domain $(0,1) \times (0,1)$, and $\partial\Omega$ the boundary of Ω.

Problem (4.12) has a unique solution for $\lambda \leq 0$, but for $\lambda > 0$, there may be zero, one, or two solutions (cf., [26]). The critical value $\lambda^* = 6.80812$ is a turning point such that for $0 < \lambda < \lambda^*$, problem (4.12) has two solutions; and for $\lambda > \lambda^*$, it has no solutions. Also, the Jacobian at the limit point is singular with a rank of $n-1$, and as λ approaches the limit point, the discretized problem becomes harder to solve. To investigate the effects of ill-conditioning on the turning point algorithms, we used both the standard bordering algorithm of [54] as well as our modified approach to solve for the turning point λ^*.

When testing the Bratu problem, the initial approximate solution as well as the initial null space vector was a scaled vector of ones on a uniform grid of size 31×31. We chose an initial value of $\lambda = 3$, which was far from the critical value.

The Laplace operator was discretized using centered differences (5-point stencil), and we computed an analytic Jacobian at each step. We used GMRES [52] with a relative tolerance of 10^{-4} for each linear subproblem, and the linear systems were solved without a preconditioner although one could have been used. We used the current approximation of the null space vector in our rank-1 augmentation matrix to the Jacobian for the modified method.

Figure 4.1 presents the results of these tests, comparing the progress of each method by residual norm (top plot) as well as the condition number of the Jacobian at each iteration (bottom plot). Up to the fourth iteration, both methods perform virtually identically. After that the original method diverges from the modified method, and we start to see a "see-saw" motion of the original solver's progress. This behavior is caused by the condition number of the Jacobian matrix. As the solution becomes more accurate, the condition number is larger because it is closer to the turning point. If the condition number is greater than say 10^{10}, then the computed step is bad and takes us away from the exact solution. Thus, the solver can only get so close to the turning point until the next (bad) step kicks it away again.

On the other hand, the modified method always deals with a matrix that is well conditioned (roughly 10^3). Here the error in each linear system is determined largely by the relative tolerance passed into GMRES (10^{-4}), and the nonlinear step computed from the linear combination of these five subproblems is much more accurate. These factors enable the modified algorithm to find the turning point to within machine precision.

Figure 4.2 tabulates the total number of linear iterations for each step of the two algorithms. Each nonlinear iteration on the x-axis in Figure 4.2 corresponds to the same subproblem in Figure 4.1. Thus, for step 6 the condition number of the actual Jacobian is roughly 10^{13}, and the condition number of the effective Jacobian used in the modified method is roughly 10^3.

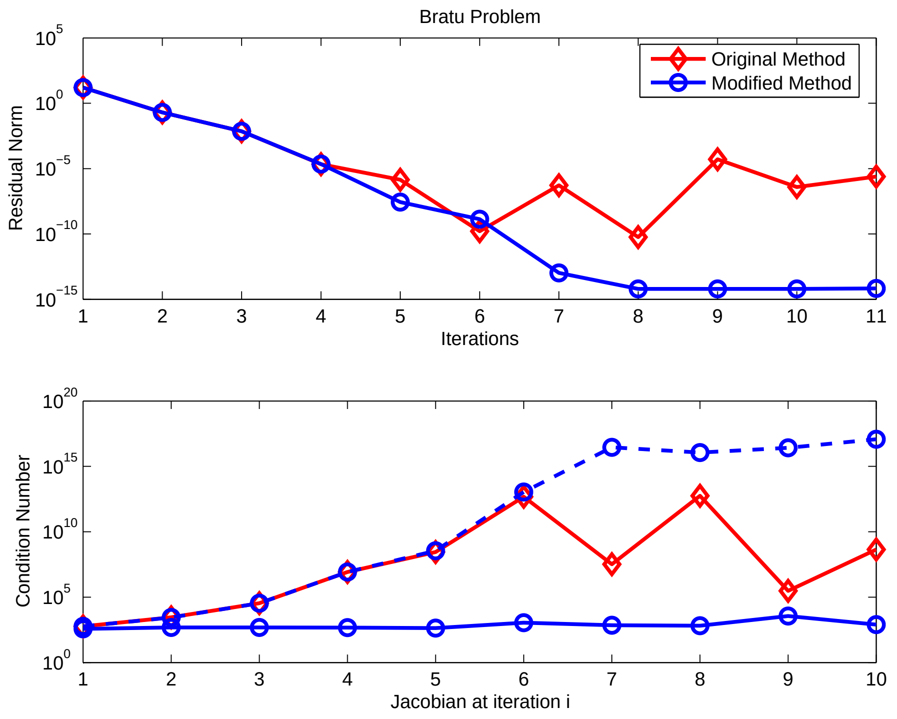

Figure 4.1. Comparison of turning point methods (residual norm and condition number) on the Bratu problem: Original bordering algorithm (◇) and our modified method (○). The solid lines in the bottom plot track the condition number of the matrix used in the linear solver (augmented Jacobian for the modified method). The dashed line indicates the condition number of the actual Jacobian (modified method).

It is evident that even though the modified method must solve five linear systems rather than four like the original method, the condition number of the matrix makes a large difference in the overall efficiency. For instance, at step 6 our modified method needed only 134 total linear iterations versus 206 for the original method.

4.4 Discussion

This chapter outlined an approach for solving ill-conditioned linear systems. It is based on a technique from tensor methods for dealing with ill-conditioned and singular Jacobians.

The main benefit of this modified approach is that it effectively lowers the condition

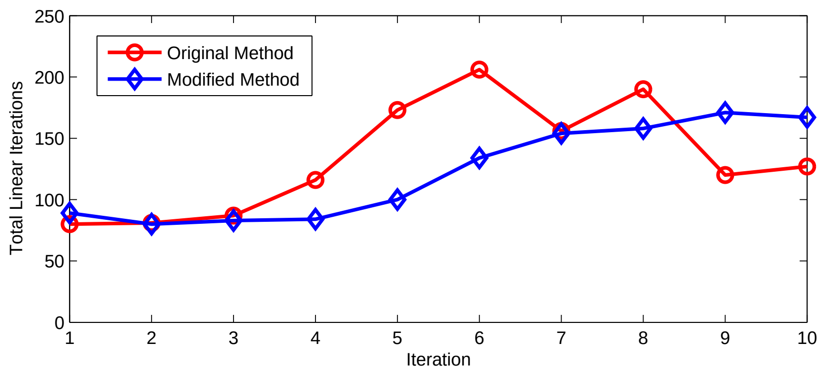

Figure 4.2. Comparison of turning point methods (total linear iterations per step) on the Bratu problem: Original bordering algorithm (◇) and our modified method (○).

number of the matrix used in a linear solver. For iterative solvers, such as GMRES [52], the distribution of eigenvalues of the matrix (and consequently its condition number) plays an important role in determining how many iterations are necessary to reach convergence [28]. From our experience, as the condition number increases, the number of iterations necessary to satisfy a fixed tolerance also increases. Thus, lowering the condition number from say 10^{12} to 10^3 can improve the performance of an iterative solver. In a similar approach, preconditioning techniques attempt to change the linear system to one that is closer to $Ix = A^{-1}b$ so that the eigenvalues are clustered around one, which also makes the matrix better conditioned.

The modified approach has two principal disadvantages, namely that two linear systems must be solved and that it may be difficult to determine an appropriate rank-1 matrix for a general problem. The first disadvantage may be mitigated through the use of block linear solvers.

More research is needed to determine the viability of this approach in practical applications. We have described one application by applying it to the turning point identification algorithm, which conveniently computes the null vector.

Chapter 5

Summary

The first objective of our research was to investigate Broyden's method for use in large-scale simulations. We implemented a parallel version and evaluated its applicability and performance on reacting flow problems and circuit simulation problems. The important finding was that Broyden's method could be used as a replacement for Newton's method when the Jacobian required for Newton's method either could not be evaluated or was inaccurate. In this case, the rank-1 update vectors could approximate the missing information as the iteration sequence progressed. This application of Broyden's method can be seen as a competitor to the Jacobian-free Newton-Krylov method since neither require a Jacobian. The advantage for Broyden's method is that it does not require an iterative linear solver.

Broyden's method was also evaluated for efficiency. The results were mixed in that some problems performed better with Newton's method and others better with Broyden's method. In general, Broyden's method was more efficient for systems that are dominated by the Jacobian evaluation time and for transient simulations. Broyden's method requires fewer Jacobian evaluations than Newton's method, thus enhancing run times for codes with costly Jacobian evaluations. For transient simulations, Broyden's method has a good initial guess for each nonlinear solve and should only require one single Jacobian approximation on the first step. This again saves time due to fewer Jacobian evaluations. The modified Newton method was just as efficient as Broyden's method on transient problems and is much less complex to implement. Therefore, unless the Jacobian is unavailable of inaccurate, we would recommend using a modified Newton method over Broyden's method in terms of efficiency.

The second objective of our research was to investigate tensor methods for solving a number of difficult problems at Sandia. We implemented a variety of tensor methods and investigated their performance and usefulness. The ability to use state-of-the-art, stand-alone linear solvers in serial or in parallel without modification was a key requirement. A secondary goal was to simplify these methods to make them more understandable and accessible to researchers.

Of the several tensor method implementations available, all but one need to access or manipulate data structures that are inaccessible in many linear solver packages. This requirement effectively narrowed the field to Bouaricha's tensor-Krylov method. However, Bouaricha's method neglects a term in certain situations, which is theoretically undesirable, and because it solves two linear systems per iteration, it could require up to twice as much execution time as other methods. We have devised a modified Bouaricha method that is more satisfying from a theoretical standpoint and can potentially exploit block linear solvers for faster performance. Because block linear solvers are not yet widely available, we analyzed the predicted performance of a block implementation.

The modified Bouaricha method proposed here eliminates an argument against Bouaricha's method, namely that it neglects a term in certain circumstances. Moreover, the modified Bouaricha method retains the capability of using stand-alone linear solvers, including all of their associated technology, such as preconditioning and restarting techniques. We have also proposed using block linear solvers and investigated their expected worst-case performance.

Our numerical results suggest that tensor methods clearly have some advantages over Newton-GMRES method, especially as the problem becomes more difficult to solve or the Jacobian grows more ill-conditioned. In addition, the modified Bouaricha method has a potential advantage over other tensor method implementations that makes it likely to be beneficial on some important problems. Specifically, it modifies the step so that in cases where the tensor model does not have a root, which usually occurs far from the solution where the secant approximation may be poor and a Newton step may be better, it attenuates the second-order information and changes the direction to be closer to the Newton step.

A third objective of our research was to apply tensor methods to the turning point algorithms used in LOCA. Because a turning point is located at a point where the Jacobian is singular, tensor methods seemed a natural fit for this problem. Over the course of our research with the turning point problem, the tensor method did not produce the expected breakthroughs. The reason is that while the Jacobian of the set of state equations is singular, the Jacobian of the overall system of equations that describes the turning point is nonsingular. Thus, there is no theoretical advantage for a tensor method in this case.

Instead, over the course of this research, we developed a novel technique for dealing with ill-conditioned linear systems. By adding a rank-1 matrix to the linear system matrix, one may change a linear system to one that is better conditioned and more amenable to iterative solvers.

We applied this technique to the turning point identification algorithm and developed a modified method. The standard bordering algorithm, which solves four subproblems with the Jacobian and forms the Newton step from their solution, is plagued by an increasingly ill-conditioned Jacobian matrix. Ill-conditioned matrices cause the solution to blow up in the direction of the null vector and cause the standard algorithm to be unstable the closer it approaches the turning point. On the other hand, our modified method was stable and robust. It was able to solve the problem to within machine precision, which meant that the Jacobian was effectively singular. For all iterative solves, instead of dealing with an ill-conditioned matrix, our method dealt with a nicely conditioned matrix.

We believe that more research is needed to understand the nuances of our modified turning point identification algorithm. In addition, we believe there are more potential applications of this modified approach to solving ill-conditioned linear problems. Lowering the condition number of a matrix involved in a linear solve is a powerful idea. Our experience has shown us that iterative solvers require more iterations as a matrix becomes more ill-conditioned, and this technique is a potential form of “preconditioning” for such systems.

References

[1] AIAA-2002-3292. *SIERRA/Premo - A New General Purpose Compressible Flow Simulation Code*, June 2002.

[2] Brett W. Bader. *Tensor-Krylov methods for solving large-scale systems of nonlinear equations*. PhD thesis, University of Colorado, Boulder, Department of Computer Science, 2003.

[3] Brett W. Bader. Tensor-Krylov methods for solving large-scale systems of nonlinear equations. *SIAM J. Numer. Anal.*, accepted 2005. To appear.

[4] Brett W. Bader and Robert B. Schnabel. Curvilinear linesearch for tensor methods. *SIAM J. Sci. Comput.*, 25(2):604–622, 2003.

[5] Brett W. Bader and Robert B. Schnabel. On the performance of tensor methods for ill-conditioned problems. Technical Report SAND2004-1944, Sandia National Laboratories, Albuquerque, NM, September 2004.

[6] Allison Baker, John Dennis, and Elizabeth R. Jessup. Toward memory-efficient linear solvers. *Lecture Notes in Computer Science*, 2565:315–327, 2003.

[7] C. Bischof, A. Carle, P. Hovland, P. Khademi, and A. Mauer. ADIFOR 2.0 user's guide. Technical Report ANL/MCS-TM-192, Argonne National Laboratory, Argonne, IL, June 1998.

[8] C. Bischof, L. Roh, and A. Mauer. ADIC — An extensible automatic differentiation tool for ANSI-C. Technical Report ANL/MCS-P626-1196, Argonne National Laboratory, Argonne, IL, 1996.

[9] Ali Bouaricha. *Solving large sparse systems of nonlinear equations and nonlinear least squares problems using tensor methods on sequential and parallel computers*. PhD thesis, University of Colorado, Boulder, Department of Computer Science, 1992.

[10] Ali Bouaricha and Robert B. Schnabel. Algorithm 768: TENSOLVE: A software package for solving systems of nonlinear equations and nonlinear least-squares problems using tensor methods. *ACM Trans. Math. Soft.*, 23:174–195, 1997.

[11] K. E. Brennam, S. L. Cambell, and L. R. Petzold. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. Classics in Applied Mathematics. SIAM, Englewood Cliffs, NJ, 1996.

[12] P. N. Brown and Y. Saad. Hybrid Krylov methods for nonlinear systems of equations. *SIAM J. Sci. Stat. Comput.*, 11:450–481, 1990.

[13] C. G. Broyden. A class of methods for solving nonlinear simultaneous equations. *Mat. Comput.*, 19:577–593, 1965.

[14] D. W. Decker, H. B. Keller, and C. T. Kelley. Convergence rate for Newton's method at singular points. *SIAM J. Numer. Anal.*, 20:296–314, 1983.

[15] D. W. Decker and C. T. Kelley. Newton's method at singular points I. *SIAM J. Numer. Anal.*, 17:66–70, 1980.

[16] D. W. Decker and C. T. Kelley. Newton's method at singular points II. *SIAM J. Numer. Anal.*, 17:465–471, 1980.

[17] D. W. Decker and C. T. Kelley. Convergence acceleration for Newton's method at singular points. *SIAM J. Numer. Anal.*, 19:219–229, 1982.

[18] R. S. Dembo, S. C. Eisenstat, and T. Steihaug. Inexact Newton methods. *SIAM J. Numer. Anal.*, 19:400–408, 1982.

[19] J. E. Dennis, Jr. and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Series in Automatic Computation. Prentice-Hall, Englewood Cliffs, NJ, 1983.

[20] Stanley C. Eisenstat and Homer F. Walker. Choosing the forcing terms in an inexact Newton method. *SIAM J. Sci. Comput.*, 17:16–32, 1996.

[21] M. Heroux et al. Trilinos solver project. http://software.sandia.gov/trilinos.

[22] Mike Heroux et al. Belos: Next generation iterative solver package. http://software.sandia.gov/belos/, 2005.

[23] Dan Feng, Paul D. Frank, and Robert B. Schnabel. Local convergence analysis of tensor methods for nonlinear equations. *Math. Prog.*, 62:427–459, 1993.

[24] Dan Feng and Thomas H. Pulliam. Tensor-GMRES method for large systems of nonlinear equations. *SIAM J. Optim.*, 7:757–779, 1997.

[25] A. H. Gebremedhin, F. Manne, and A. Pothen. Graph coloring in optimization revisited. Technical Report 226, University of Bergen, January 2002.

[26] R. Glowinski, H. B. Keller, and L. Reinhart. Continuation-conjugate gradient methods for the least squares solution of nonlinear boundary value problems. *SIAM J. Sci. Statist. Comput.*, 6:793–832, 1985.

[27] G. Golub and C. Van Loan. *Matrix Computations*. The John Hopkins University Press, Baltimore, 1989.

[28] Anne Greenbaum. *Iterative Methods for Solving Linear Systems*. Frontiers in Applied Mathematics. SIAM, Philadelphia, 1997.

[29] A. Griewank. On solving nonlinear equations with simple singularities or nearly singular solutions. *SIAM Review*, 27:537–563, 1985.

[30] A. Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Frontiers in Applied Mathematics. SIAM, Philadelphia, 2000.

[31] A. Griewank and M. R. Osborne. Analysis of Newton's method at irregular singularities. *SIAM J. Numer. Anal.*, 20:747–773, 1983.

[32] R. Y. Grimes, G. A. Pulido, R. A. Levine, and A. P. Yoganathan. Quasisteady behavior of pulsatile, confined, counterflowing jets: Implications for the assessment of mitral and tricuspid regurgitation. *Journal of Biomechanical Engineering*, 118(4):498–505, November 1996.

[33] William D. Gropp, Dennis K. Kaushik, David E. Keyes, and Barry F. Smith. Toward realistic performance bounds for implicit CFD codes. In A. Ecer et al., editor, *Proceedings of Parallel CFD'99*, pages 233–240. Elsevier, 1999.

[34] Michael A. Heroux, Roscoe A. Bartlett, Vicki E. Howle, Robert J. Hoekstra, Jonathan J. Hu, Tamara G. Kolda, Richard B. Lehoucq, Kevin R. Long, Roger P. Pawlowski, Eric T. Phipps, Andrew G. Salinger, Heidi K. Thornquist, Ray S. Tuminaro, James M. Willenbring, Alan Williams, and Kendall S. Stanley. An overview of the trilinos project. *ACM Trans. Math. Soft.*, 31(3):397–423, September 2005.

[35] T. J. R. Hughes, L. P. Franca, and G. M. Hulbert. A new finite element formulation for computational fluid dynamics: VII. the Galerkin/Least-Squares method for advective-diffusive equations. *Computer Methods in Applied Mechanics and Engineering*, 73:173–189, 1989.

[36] Eric R. Keiter, Scott A. Hutchinson, Robert J. Hoekstra, Lon J. Waters, Thomas V. Russo, Eric L. Rankin, Roger P. Pawlowski, and Steven D. Wix. Xyce parallel electronic simulator: User's guide, version 2.1. Technical Report SAND2005-3916, Sandia National Laboratories, Albuquerque, NM, January 2005.

[37] C. T. Kelley. *Iterative Methods for Linear and Nonlinear Equations*, volume 16 of *Frontiers in Applied Mathematics*. SIAM, Philadelphia, 1995.

[38] C. T. Kelley. *Solving Nonlinear Equations with Newton's Method*, volume 1 of *Fundamentals of Algorithms*. SIAM, Philadelphia, 2003.

[39] C. T. Kelley and R. Suresh. A new acceleration method for Newton's method at singular points. *SIAM J. Numer. Anal.*, 20:1001–1009, 1983.

[40] D.A. Knoll and D.E. Keyes. Jacobian-free Newton-Krylov methods: a survey of approaches and applications. *J. Comput. Phys.*, 193(2):357–397, January 2004.

[41] Tamara Kolda and Roger Pawlowski et al. NOX: An object-oriented nonlinear solver package. http://software.sandia.gov/nox/, 2005.

[42] David G. Luenberger. *Linear and Nonlinear Programming, Second Edition*. Addison-Wesley, Reading, MA, 1984.

[43] Jorge J. Moré. A collection of nonlinear model problems. *Lectures Appl. Math.*, 26:723–762, 1990.

[44] R. P. Pawlowski, A. G. Salinger, T. J. Mountziaris, and J. N. Shadid. A bifurcation and stability analysis of laminar isothermal counterflowing jets. *Journal of Fluid Mechnics*, 2005.

[45] R. P. Pawlowski, J. N. Shadid, J. P. Simonis, and H. F. Walker. Globalization techniques for Newton–Krylov methods and applications to the fully-coupled solution of the Navier–Stokes equations. Technical Report SAND2004-1777, Sandia National Laboratories, Albuquerque, NM, May 2004.

[46] R. P. Pawlowski, J. N. Shadid, J. P. Simonis, and H. F. Walker. Globalization techniques for Newton–Krylov methods and applications to the fully-coupled solution of the Navier–Stokes equations. *SIAM Review*, 2005. Accepted.

[47] G. W. Reddien. On Newton's method for singular problems. *SIAM J. Numer. Anal.*, 15:993–996, 1978.

[48] A. Ruhe. Implementation aspects of band Lanczos algorithms for computation of eigenvalues of large sparse symmetric matrices. *Mathematics of Computations*, 33:680–687, 1979.

[49] Yousef Saad. ILUT: A dual threshold incomplete ILU preconditioner. *Numer. Linear Algebra Appl.*, 1:387–402, 1994.

[50] Yousef Saad. Analysis of augmented Krylov subspace methods. *SIAM J. Matrix Anal. Appl.*, 18:435–449, 1997.

[51] Yousef Saad. *Iterative Methods for Sparse Linear Systems, Second Edition*. SIAM, Philadelphia, 2003.

[52] Yousef Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 7:856–869, 1986.

[53] S. A. Safvi and T. J. Mountziaris. A new reactor for purely homogeneous kinetic studies of endothermic reactions. *AIChE Journal*, 40(9):1535–1548, September 1994.

[54] A. G. Salinger, N. M. Bou-Rabee, R. P. Pawlowski, E. D. Wilkes, E. A. Burroughs, R. B. Lehoucq, and L. A. Romero. LOCA 1.0 A library of continuation algorithms: Theroy and implementation manual. Technical Report SAND2002-0396, Sandia National Laboratories, Albuquerque, NM, March 2002.

[55] A. G. Salinger, E. A. Burroughs, R. P. Pawlowski, E. T. Phipps, and L. A. Romero. Bifurcation tracking algorithms and software for large scale applications. *Int. J. Bifurcat. Chaos*, 15(3):1015–1032, 2005.

[56] D. Sarigiannis, T. J. Mountziaris, G. Kioseoglou, and A. Petrou. Synthesis of crystalline ZnSe nanoparticles in a counterflow jet reactor. *Applied Physics Letters*, 80(21):4024–6, 2002.

[57] Robert B. Schnabel and Paul D. Frank. Tensor methods for nonlinear equations. *SIAM J. Numer. Anal.*, 21:815–843, 1984.

[58] K. Seshadri, C. Trevino, and M. D. Smooke. Analysis of the structure and mechanisms of extinction of a counterflow methane-air diffusion flame. *Combustion and Flame*, 76:111–132, 1989.

[59] J. N. Shadid. A fully-coupled Newton-Krylov solution method for parallel unstructured finite element fluid flow, heat and mass transfer simulations. *Int. J. Comput. Fluid Dyn.*, 12:199–211, 1999.

[60] J. N. Shadid, H. K. Moffat, S. A. Hutchinson, G. L. Hennigan, K. D. Devine, and A. G. Salinger. MPSalsa: A finite element computer program for reacting flow problems part 1: Theoretical development. Technical Report SAND95-2752, Sandia National Laboratories, Albuquerque, NM, May 1996.

[61] J. N. Shadid, A.G. Salinger, R.P. Pawlowski, P.T. Lin, G.L. Hennigan, R.S. Tuminaro, and R.B. Lehoucq. Large-scale stablilized FE computational analysis of nonlinear steady-state transport/reaction systems. *CMAME*, 2005. In press.

[62] John N. Shadid, Ray S. Tuminaro, and Homer F. Walker. An inexact Newton method for fully coupled solution of the Navier–Stokes equations with heat and mass transport. *J. Comput. Phys.*, 137:155–185, 1997.

[63] T. E. Tezduyar. Stabilized finite element formulations for incompressible flow computations. *Adv. App. Mech.*, 28:1, 1992.

[64] R. S. Tuminaro, M. A. Heroux, S. A. Hutchinson, and J. N. Shadid. *Official Aztec User's Guide, Version 2.1*. Sandia National Laboratories, Albuquerque, NM, 1999.

DISTRIBUTION:

1 MS 9159
Tamara G. Kolda, 08962

5 MS 0316
Roger Pawlowski, 01416

5 MS 0316
Brett Bader, 01416

1 MS 9018
Central Technical Files, 8945-1

2 MS 0899
Technical Library, 9616