Improving the Efficiency of Runge-Kutta Methods for the Solution of BVPs for Higher-Order ODEs

by

Khalid Zuberi

A thesis submitted in conformity
with the requirements for the degree of
Master of Science
Graduate Department of Computer Science
University of Toronto

Abstract

Improving the Efficiency of
Runge-Kutta Methods for the Solution of
BVPs for Higher-Order ODEs

Khalid Zuberi
Master of Science
Graduate Department of Computer Science
University of Toronto, 1996

Runge-Kutta methods are often used to solve two-point boundary value problems (BVPs) for ordinary differential equations (ODEs.) These methods are usually applied directly to first-order systems; to solve problems for higher-order ODEs, the differential equations are often reduced to an equivalent first-order system. We exploit the structure of the reduced system to construct an approximate Jacobian matrix that is significantly cheaper to construct for general higher order BVPs. Second-order ODEs are examined in detail, and a specialised linear system solver is suggested to provide additional computational savings. Numerical results on sample BVPs for second-order ODEs show significant computational savings with no loss of solution accuracy.

Acknowledgements

I would like to thank my supervisor Prof. Ken Jackson for suggesting this project and his guidance throughout this work, my second reader Tom Fairgrieve for many suggestions that improved the overall thesis, and Paul Muir for some insightful discussions. I also thank the Natural Science and Engineering Research Council for providing financial support, and finally my wife Ameena Philips for constantly brightening my life.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Runge-Kutta (RK) methods are often used to solve two-point boundary value problems (BVPs) for ordinary differential equations (ODEs.) These methods apply directly to first-order systems; to solve problems consisting of higher-order equations, the ODEs are often reduced to an equivalent first-order system. We exploit the structure of the resulting reduced equations, in particular, the structure of the *Jacobian* of the discretized form of the reduced system, to find an approximation that will be cheaper to construct and for which linear system solves can be computed efficiently.

A variety of techniques have been developed to solve BVPs for ODEs. These fall into two major classes: initial value methods and finite difference methods.

The simplest of the initial value methods is simple shooting, in which a method for solving a related initial value problem (IVP) is used to guess at a solution to the BVP and then to iteratively improve the solution. However, this method suffers from a number of drawbacks: it can be unstable and it suffers from the accumulation of roundoff error. The multiple shooting method overcomes these difficulties by dividing the problem interval into a set of sub-intervals and applying simple shooting on each of them. Multiple shooting is stable if the BVP is well conditioned and the shooting points satisfy certain criteria.

In finite difference methods, the problem interval is divided into a set of sub-intervals and finite difference equations are formulated over these sub-intervals to approximate the ODE. The resulting system of finite difference equations is solved to produce a solution over the entire problem interval. For this reason, these methods are also referred to as global methods. Examples of some simple one-step finite difference schemes are the midpoint and trapezoidal methods, which are second-order methods. Higher-order methods are preferable to solve problems to higher accuracy. Various higher-order methods are obtained from the class of Runge-Kutta methods. Explicit Runge-Kutta methods are easier to evaluate than the more general implicit methods, however they suffer stability problems.

Another finite difference method is the method of collocation. It can be shown to be equivalent to a subclass of RK methods. In these methods, an interpolant is fit through a mesh of 'collocation' points. An advantage of collocation is that it can be applied directly to higher-order systems of equations, while most of the other methods require the system be reduced to first-order, increasing its size. Finally, note that a one-step difference method can be regarded as multiple shooting with integration over each sub-interval performed by the single application of a difference method. Hence there is a relationship between these two classes.

Our work concentrates on a class of Runge-Kutta (RK) methods known as mono-implicit Runge-Kutta (MIRK) [12] methods, but can be extended to more general RK methods. For BVPs, MIRK methods have the desirable stability properties of implicit RK methods, while having efficiency comparable to explicit RK methods.

Particular attention is given to the case of second-order systems, and a specialised and efficient linear system solver is developed for systems that use the approximate Jacobian that we derive. The specialised solver requires that extra care be taken to guarantee an accurate solution to the problem, and our current implementation requires a specific, but common, arrangement of boundary conditions. It remains to find a general technique for solving linear systems involving the approximate Jacobians that arise from ODEs with more general arrangements of boundary conditions, and with order greater than two.

In Chapter 2 we define the relevant RK methods and review their use for solving BVPs. In Chapter 3 we derive the form of an approximate Jacobian by dropping terms that contain second or higher powers of the mesh width h from the exact

Jacobian. We also outline a specialised solver for linear systems formed using the approximate Jacobian when it arises from second-order systems of equations. Chapter 4 contains the results of numerical testing and Chapter 5 concludes our work and suggests directions for further investigation.

Chapter 2

Using RK methods to solve BVPs

2.1 Runge-Kutta methods

Runge-Kutta (RK) methods are used to approximate solutions to systems of ODEs. Given a system of m ODEs, $y' = f(t, y)$, the general form of an implicit RK method is

$$\phi(y_n, y_{n+1}) = y_{n+1} - y_n - h_n \sum_{r=1}^{s} \omega_r k_r = 0, \tag{2.1}$$

with

$$k_r = f(x_n + c_r h_n, y_n + h_n \sum_{j=1}^{s} \beta_{rj} k_j), \qquad r = 1, \ldots, s, \tag{2.2}$$

where $y_n \approx y(x_n)$, $y_{n+1} \approx y(x_{n+1})$ and $h_n = x_{n+1} - x_n$. The subscript n ranges from 0 to N, and is used to denote discrete values of the independent variable and solution. The parameters ω_r, β_{rj}, and c_r define the particular formula used, s is the number of stages, and k_r are the stage values. In our application, it is convenient to not always write the explicit dependence on the independent variable x in the evaluations of the function f. With this understanding, equations (2.1)-(2.2) become

$$\phi(y_n, y_{n+1}) = y_{n+1} - y_n - h_n \sum_{r=1}^{s} \omega_r k_r = 0, \tag{2.3}$$

with

$$k_r = f(y_n + h_n \sum_{j=1}^{s} \beta_{rj} k_j), \qquad r = 1, \ldots, s. \tag{2.4}$$

The subclass of *explicit* methods is similarly defined by

$$\phi(y_n, y_{n+1}) = y_{n+1} - y_n - h_n \sum_{r=1}^{s} \omega_r k_r = 0, \tag{2.5}$$

with

$$k_r = f(y_n + h_n \sum_{j=1}^{r-1} \beta_{rj} k_j), \qquad r = 1, \ldots, s. \tag{2.6}$$

Note that in the latter case the matrix of coefficients $[\beta_{rj}]$ is strictly lower triangular, hence each stage k_j is defined explicitly in terms of previous stages and can be computed easily. In the case of implicit formulas, however, each stage k_j is defined implicitly in terms of itself and all the other stages, and, in general, a non-linear system must be solved to calculate the stage values.

2.2 MIRK methods

Mono-implicit Runge-Kutta (MIRK) methods are a subclass of RK methods that can be written in a special form that includes an additional set of parameters, which we denote as γ_r. MIRK methods are defined by

$$\phi(y_n, y_{n+1}) = y_{n+1} - y_n - h_n \sum_{r=1}^{s} \omega_r k_r = 0, \tag{2.7}$$

with

$$k_r = f((1-\gamma_r)y_n + \gamma_r y_{n+1} + h_n \sum_{j=1}^{r-1} \beta_{rj} k_j), \qquad r = 1, \ldots, s, \tag{2.8}$$

where the matrix of coefficients $[\beta_{rj}]$ is lower triangular, as in explicit RK methods, and γ_r is a scalar parameter. It can be shown that any MIRK method is equivalent to an implicit RK method (the converse is not true). MIRK methods share the favourable stability properties of implicit methods, while providing improved computational efficiency when applied to the solution of BVPs because of the explicit definition of the stages. Implicitly defined stages would require an additional system solve and add a level of iteration at the innermost loop of the solution code, resulting in a significant impact on performance. See [12] for a description of the use of implicit RK and MIRK formulas for solving BVPs.

The coefficients that define a MIRK method can be written in the form of a tableau as

$$\begin{array}{cc|cccccc}
c_1 & \gamma_1 & 0 & 0 & \ldots & 0 & 0 \\
c_2 & \gamma_2 & \beta_{21} & 0 & \ldots & 0 & 0 \\
\vdots & \vdots & \vdots & \ddots & & \vdots & \vdots \\
c_s & \gamma_s & \beta_{s1} & \ldots & & \beta_{s\ s-1} & 0 \\
\hline
 & & \omega_1 & \ldots & & \omega_{s-1} & \omega_s
\end{array}\ . \tag{2.9}$$

A discussion of order results for MIRK methods and a characterisation of low-order methods can be found in [2].

2.3 Solving two-point BVPs using RK methods: an overview

Although originally used to solve initial value problems, RK methods are often used to solve BVPs for ODEs. In this section we briefly describe how RK methods can be used to solve BVPs and how a RK-BVP solver would function.

Since the method described here applies to first-order systems of equations, arbitrary systems are reduced to first-order by introducing additional variables.

Consider a system of m equations

$$y' = f(x, y) \tag{2.10}$$

over the interval $[a, b]$ of the independent variable. The equations are subject to the separated boundary conditions

$$g_a(y(a)) = 0, \quad g_b(y(b)) = 0, \tag{2.11}$$

where $y \in \Re^m,\ f : \Re^{m+1} \to \Re^m,\ g_a : \Re^m \to \Re^{m_1},\ g_b : \Re^m \to \Re^{m_2},\ m_1 + m_2 = m.$

The interval $[a, b]$ is divided into a mesh $T = (x_0, x_1, x_2, \ldots x_N)^T$ with $a = x_0 < x_1 < \ldots < x_N = b$, and the mesh interval widths are defined to be $h_i = x_{i+1} - x_i$. A system of equations $\Phi(Y) = 0$ is formulated whose solution $W = (w_0, w_1, \ldots, w_N)^T$ is a discrete approximation to the solution of the BVP. That is, $w_i \approx y(x_i)$, for each i in $0, 1, \ldots, N$. To create the system Φ, a RK formula such as (2.3)-(2.4) is applied to the BVP (2.10)-(2.11) over each mesh interval. Since Runge-Kutta formulas depend on two consecutive mesh-points only, Φ has the form

$$\Phi(W) = \begin{pmatrix} \Phi_a(W) \\ \Phi_0(W) \\ \Phi_1(W) \\ \vdots \\ \Phi_{N-1}(W) \\ \Phi_b(W) \end{pmatrix} = \begin{pmatrix} g_a(w_0) \\ \phi(w_0, w_1) \\ \phi(w_1, w_2) \\ \vdots \\ \phi(w_{N-1}, w_N) \\ g_b(w_N) \end{pmatrix} = 0. \tag{2.12}$$

To solve the system, we require an initial approximation to the solution, $W^{[0]}$ over an initial mesh $T^{[0]}$. We generate subsequent approximations using an iterative method as described in Section 2.4, which under favourable conditions will converge to a solution. The iteration is halted when a test such as $\|\delta W^{[k]}\| < TOL$ is satisfied, where TOL is a parameter. An error estimate is then typically generated, and used along with a user specified error control criteria to decide if the solution computed is satisfactory. If the solution is not satisfactory, the mesh $T^{[k]}$ is refined or redistributed, and the above process is repeated.

2.4 Iterative methods for solving nonlinear systems

In general, the computation of a solution to a BVP by RK methods involves solving a nonlinear system of equations. This is usually accomplished using an iterative technique such as one of those briefly outlined below. We focus on Newton's method, as its simplicity and convergence properties make it widely used. Also mentioned are related methods in which approximations are used to reduce the cost of the iterative process.

We consider the solution to a nonlinear system

$$\Phi(W) = 0. \tag{2.13}$$

The general family of one-step stationary iterative methods are of the form

$$W^{[i+1]} = G(W^{[i]}). \tag{2.14}$$

One method of significance is Newton's method, in which

$$G(W) = W - \Phi'(W)^{-1}\Phi(W). \tag{2.15}$$

In practice, Φ is not explicitly inverted. Instead, the formulation

$$\Phi'(W^{[i]})\delta W^{[i]} = -\Phi(W^{[i]}) \tag{2.16}$$

is used, where $\delta W^{[i]} = (W^{[i+1]} - W^{[i]})$. The update $\delta W^{[i]}$ is computed from (2.16) using a linear system solver. The next iterate is computed using

$$W^{[i+1]} = W^{[i]} + \delta W^{[i]}. \tag{2.17}$$

A variation, called the damped Newton method, is often used to improve the convergence properties of the iteration. In this case (2.17) is replaced with

$$W^{[i+1]} = W^{[i]} + \lambda^{[i]} \delta W^{[i]}, \tag{2.18}$$

where $0 < \lambda^{[i]} < 1$. A predictor-corrector scheme is used, along with a function denoted g called the objective function, to select the $\lambda^{[i]}$ for each iteration. A more complete discussion of the damped Newton method is found in [1]. Here we mention only the natural criterion function, defined to be

$$g(W, \lambda \delta W) = \frac{1}{2} \|\Phi'(W)^{-1} \Phi(W + \lambda \delta W)\|^2. \tag{2.19}$$

Note that g varies from iteration to iteration, though here we have dropped the superscripts. Again, Φ' is not explicitly inverted, and, since a factorisation of Φ' is usually already available from the linear system solver to compute the solution for (2.16), this computation is not very expensive, requiring just forward and back substitutions. In addition to improving the interval of convergence of the iteration, the damped Newton method is also useful for the early detection of a failed iteration.

Newton's method is said to exhibit quadratic convergence. That is, for a large class of equations, when the iterates $W^{[i]}$ are close enough to the solution W^*, the relation

$$\|W^{[i+1]} - W^*\| \leq c \|W^{[i]} - W^*\|^2 \tag{2.20}$$

holds, for some constant $c < +\infty$. Not all iterative methods have this advantageous property. A disadvantage for using Newton's method is that Φ' must be evaluated at every step, which is an expensive process.

A family of related Newton-type iterative methods are defined by

$$G(W) = W - \tilde{\Phi}'(W)^{-1}\Phi(W), \tag{2.21}$$

where $\tilde{\Phi}'$ approximates Φ' in some way, and is cheaper to compute. Although cheaper, the disadvantage of these methods is the convergence rate drops to being only linear or superlinear. That is, the iterates satisfy

$$\lim_{i\to\infty} \frac{\|W^{[i+1]} - W^*\|}{\|W^{[i]} - W^*\|} = c, \tag{2.22}$$

with $c = 0$ when superlinear and $c < +\infty$ when linear, but do not display the quadratic convergence property (2.20) of Newton's method. Another common simplification is to hold the Jacobian constant for several iterations. In this case we no longer have a one-step station iteration.

Iterative schemes such as those described above are only locally convergent. That is, if the initial guess $W^{[0]}$ is close enough to a solution W^* of (2.13), then the series of iterates $W^{[i]}$ converges to W^*. Ostrowski's theorem provides a sufficient condition for the local convergence of an iteration at W^*, namely

$$\rho(G'(W^*)) < 1, \tag{2.23}$$

where ρ denotes the spectral radius. This result requires only that G be continuously differentiable at W^* and is valid for the general iteration (2.14). See [14] for a much more complete discussion of iterative methods for solving nonlinear equations.

2.5 Simple improvements for reduced systems

Recent work by Enright and Hu [5] demonstrates a simple way to take advantage of the structure of reduced systems of differential equations when solving BVPs. Although we do not apply their technique in our work, their results demonstrate some of the potential gains when taking advantage of the special structure of reduced systems of equations. Their technique is applicable to a wide range of solution methods. They identify a sparsity pattern in a matrix that is used frequently in the inner loops of a solution code, and suggest how optimising numerical operations on this matrix can lead to substantial savings. The sparsity pattern arises as a direct result of the reduction of a higher-order system of ODEs to first-order. We present a brief explanation of their method as it applies to an autonomous p^{th}-order scalar differential equation, noting that the results generalise easily to systems of equations.

Consider the p^{th}-order scalar differential equation

$$\frac{d^p y}{dx^p} = f(y, \frac{dy}{dx}, \frac{d^2 y}{dx^2}, \dots, \frac{d^{p-1} y}{dx^{p-1}}). \tag{2.24}$$

We transform this to a first-order system by introducing the new variables

$$\begin{aligned} u^{(0)} &= y, \\ u^{(1)} &= \tfrac{dy}{dx}, \\ u^{(2)} &= \tfrac{d^2 y}{dx^2}, \\ &\vdots \\ u^{(p-1)} &= \tfrac{d^{p-1} y}{dx^{p-1}}. \end{aligned} \tag{2.25}$$

The transformed system is

$$w' = \begin{pmatrix} u^{(0)} \\ u^{(1)} \\ \vdots \\ u^{(p-2)} \\ u^{(p-1)} \end{pmatrix}' = \begin{pmatrix} u^{(1)} \\ u^{(2)} \\ \vdots \\ u^{(p-1)} \\ f(u^{(0)}, u^{(1)}, \dots, u^{(p-1)}) \end{pmatrix} = F(w). \tag{2.26}$$

Now observe the particularly simple form of the Jacobian matrix of this system, namely

$$\frac{\partial F}{\partial w} = J = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 0 & 1 \\ \frac{\partial f}{\partial u^{(0)}} & \frac{\partial f}{\partial u^{(1)}} & \frac{\partial f}{\partial u^{(2)}} & \dots & \frac{\partial f}{\partial u^{(p-2)}} & \frac{\partial f}{\partial u^{(p-1)}} \end{pmatrix}. \tag{2.27}$$

Many schemes that solve BVPs must at some stage perform matrix multiplications involving the Jacobian. If the system has the form of (2.27), then an efficient multiplication procedure can be used. A multiplication of the form $J \cdot A$ where A is a dense $p \times p$ matrix is a simple operation in light of the structure of J. The first $p-1$ rows of the product are the same as the rows of A shifted up one level, while each element of the last row of the product is computed by the inner-product of two p-vectors. The cost of this product is then proportional to p^2 floating-point operations, while a regular dense matrix multiplication would be proportional to p^3 floating-point operations. Similarly, a factor of p is saved if a system of m p^{th}-order equations is being solved.

This simple optimisation lends itself to implementation in a variety of solution codes, including solvers that employ Runge-Kutta schemes and multiple shooting codes. This optimisation is not restricted to uniform-order systems of equations; very similar results are obtained for mixed-order systems.

Enright and Hu tested their scheme by modifying MIRKDC [6], a code that solves boundary value problems using MIRK methods, and also a shooting code, with favourable results. A savings of between 10 and 34 percent was reported when the method was tested on a small set of problems with the MIRKDC code with a sixth-order MIRK method, and between 12 and 44 percent with a fourth-order MIRK method.

Chapter 3

Approximate Jacobians for MIRK methods

3.1 Derivation of an approximate Jacobian

A large portion of the total cost of a BVP code using RK methods lies in constructing the Jacobian matrix of the discrete system that approximates the problem. Profiling reported for MIRKDC in [13] indicates the setup of the residual Φ and the Jacobian $\Phi' = \partial\Phi/\partial W$ accounts for approximately 55% of the total cost of computing a solution. For systems of large size m, this percentage may be even higher. In this section we will construct an approximation to the Jacobian matrix of a p^{th}-order system of ODEs by discarding terms containing second or higher powers of h. The calculations in this section require only that Φ is differentiable. The approximations converge to the exact values as the mesh width $h \to 0$. For convenience of notation we assume the mesh width is uniform, although the results are valid for a non-uniform mesh. A derivation similar to the one presented here for the particular case $p = 2$, on which this section is largely based, is due to Ken Jackson [8].

Consider a p^{th}-order system of ODEs as in (2.24), but where $y \in \Re^m$, so that it is a system of m p^{th}-order ODEs. Let the boundary conditions be

$$g_a(y(a), \frac{dy}{dx}(a), \dots, \frac{d^{p-1}y}{dx^{p-1}}(a)) = 0, \quad g_b(y(b), \frac{dy}{dx}(b), \dots, \frac{d^{p-1}y}{dx^{p-1}}(b)) = 0. \tag{3.1}$$

The ODEs can be transformed to a first-order system using (2.25) to get a system as in (2.26) but where each $u^{(i)} \in \Re^m$. The boundary conditions become

$$g_a(u^{(0)}(a), u^{(1)}(a), \dots, u^{(p-1)}(a)) = 0, \quad g_b(u^{(0)}(b), u^{(1)}(b), \dots, u^{(p-1)}(b)) = 0. \tag{3.2}$$

The approximate solution to the BVP over the mesh $T = (x_0, x_1, \dots, x_N)^T$ is $W = (w_0, w_1, \dots, w_N)^T$. We write the discretized system as $\Phi(W) = 0$. To solve the system a Newton iteration as in (2.16) is used.

The interior of the Jacobian matrix $\Phi'(W)$ (ignoring the equations associated with the boundary conditions) has the structure

$$\begin{pmatrix} \frac{\partial\phi(w_0,w_1)}{\partial w_0} & \frac{\partial\phi(w_0,w_1)}{\partial w_1} & & & \\ & \frac{\partial\phi(w_1,w_2)}{\partial w_1} & \frac{\partial\phi(w_1,w_2)}{\partial w_2} & & \\ & \ddots & \ddots & & \\ & & & \frac{\partial\phi(w_{N-1},w_N)}{\partial w_{N-1}} & \frac{\partial\phi(w_{N-1},w_N)}{\partial w_N} \end{pmatrix}. \tag{3.3}$$

We are concerned here with the form of each block row, namely

$$\begin{bmatrix} 0 & \dots & 0 & L & R & 0 & \dots & 0 \end{bmatrix} = \\ \begin{bmatrix} 0 & \dots & 0 & \frac{\partial\phi(w_n,w_{n+1})}{\partial w_n} & \frac{\partial\phi(w_n,w_{n+1})}{\partial w_{n+1}} & 0 & \dots & 0 \end{bmatrix}, \tag{3.4}$$

and, since $w_n = (u_n^{(0)}, u_n^{(1)}, \ldots, u_n^{(p-1)})^T$, we have

$$L = \begin{pmatrix} \frac{\partial \phi^{(0)}(w_n,w_{n+1})}{\partial u_n^{(0)}} & \frac{\partial \phi^{(0)}(w_n,w_{n+1})}{\partial u_n^{(1)}} & \cdots & \frac{\partial \phi^{(0)}(w_n,w_{n+1})}{\partial u_n^{(p-1)}} \\ \frac{\partial \phi^{(1)}(w_n,w_{n+1})}{\partial u_n^{(0)}} & \frac{\partial \phi^{(1)}(w_n,w_{n+1})}{\partial u_n^{(1)}} & \cdots & \frac{\partial \phi^{(1)}(w_n,w_{n+1})}{\partial u_n^{(p-1)}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \phi^{(p-1)}(w_n,w_{n+1})}{\partial u_n^{(0)}} & \frac{\partial \phi^{(p-1)}(w_n,w_{n+1})}{\partial u_n^{(1)}} & \cdots & \frac{\partial \phi^{(p-1)}(w_n,w_{n+1})}{\partial u_n^{(p-1)}} \end{pmatrix} \tag{3.5}$$

$$\tag{3.6}$$

and

$$R = \begin{pmatrix} \frac{\partial \phi^{(0)}(w_n,w_{n+1})}{\partial u_{n+1}^{(0)}} & \frac{\partial \phi^{(0)}(w_n,w_{n+1})}{\partial u_{n+1}^{(1)}} & \cdots & \frac{\partial \phi^{(0)}(w_n,w_{n+1})}{\partial u_{n+1}^{(p-1)}} \\ \frac{\partial \phi^{(1)}(w_n,w_{n+1})}{\partial u_{n+1}^{(0)}} & \frac{\partial \phi^{(1)}(w_n,w_{n+1})}{\partial u_{n+1}^{(1)}} & \cdots & \frac{\partial \phi^{(1)}(w_n,w_{n+1})}{\partial u_{n+1}^{(p-1)}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \phi^{(p-1)}(w_n,w_{n+1})}{\partial u_{n+1}^{(0)}} & \frac{\partial \phi^{(p-1)}(w_n,w_{n+1})}{\partial u_{n+1}^{(1)}} & \cdots & \frac{\partial \phi^{(p-1)}(w_n,w_{n+1})}{\partial u_{n+1}^{(p-1)}} \end{pmatrix}. \tag{3.7}$$

If we discretized the system using a MIRK formula, then

$$\Phi_n^{(i)} = \phi^{(i)}(u_n^{(i)}, u_{n+1}^{(i)}) = u_{n+1}^{(i)} - u_n^{(i)} - h \sum_{r=1}^{s} \omega_r F^{(i)}(U_{n,r}), \tag{3.8}$$

where

$$U_{n,r}^{(i)} = (1-\gamma_r)u_n^{(i)} + \gamma_r u_{n+1}^{(i)} + h \sum_{j=1}^{r-1} \beta_{rj} F^{(i)}(U_{n,j}). \tag{3.9}$$

Following (2.26), in the above we have used

$$F(W) = \begin{pmatrix} F^{(0)}(W) \\ F^{(1)}(W) \\ \vdots \\ F^{(p-1)}(W) \end{pmatrix}. \tag{3.10}$$

Similarly,

$$U_{n,r} = \begin{pmatrix} U_{n,r}^{(0)} \\ U_{n,r}^{(1)} \\ \vdots \\ U_{n,r}^{(p-1)} \end{pmatrix}. \tag{3.11}$$

Our goal is to compute an approximation to Φ' by dropping terms containing second or higher powers of h. To compute the general form of the L blocks, we observe from (3.5) that it consists of p block-rows, labelled 0 through $p-1$. Since the system of equations is formed by reducing a p^{th} order ODE to first-order, we find that much of the structure of block rows 0 through $p-2$ arises from this reduction process. The last row, number $p-1$, depends directly on the original BVP. Hence we examine separately block rows number 0 through $p-2$ and row number $p-1$. For the first set case, rows 0 through $p-2$, we examine the subcases for the partial derivatives $\partial\Phi_n^{(i)}/\partial u_n^{(k)}$ given by $i=k$, $i+1=k$, and otherwise. The last row of L has a more complicated structure. For this case, we examine the first $p-1$ columns, which have a similar structure, and then the last column, which has a slightly different structure. The same procedure is applied to compute an approximation to R.

In order to perform these calculations, we make use of approximations for the partial derivatives of $U_{n,r}^{(i)}$ with respect to $u_n^{(k)}$ and $u_{n+1}^{(k)}$. These can be broken into the cases $i=k$ and $i \neq k$. We deal first with

$$\frac{\partial U_{n,r}^{(i)}}{\partial u_n^{(k)}} = (1-\gamma_r)\frac{\partial u_n^{(i)}}{\partial u_n^{(k)}} + h\sum_{j=1}^{r-1}\beta_{rj}\frac{\partial F^{(i)}(U_{n,j})}{\partial U_{n,j}}\frac{\partial U_{n,j}}{\partial u_n^{(k)}}. \tag{3.12}$$

When $i=k$, (3.12) becomes

$$\frac{\partial U_{n,r}^{(i)}}{\partial u_n^{(i)}} = (1-\gamma_r)I + O(h), \tag{3.13}$$

where we have used $O(h)$ to represent terms which contain positive powers of h. For the other cases, $i \neq k$, (3.12) becomes

$$\frac{\partial U_{n,r}^{(i)}}{\partial u_n^{(k)}} = 0 + O(h). \tag{3.14}$$

Similarly,

$$\frac{\partial U_{n,r}^{(i)}}{\partial u_{n+1}^{(k)}} = \gamma_r \frac{\partial u_{n+1}^{(i)}}{\partial u_{n+1}^{(k)}} + h \sum_{j=1}^{r-1} \beta_{rj} \frac{\partial F^{(i)}(U_{n,j})}{\partial U_{n,j}} \frac{\partial U_{n,j}}{\partial u_{n+1}^{(k)}}. \tag{3.15}$$

When $i = k$, (3.15) becomes

$$\frac{\partial U_{n,r}^{(i)}}{\partial u_{n+1}^{(i)}} = \gamma_r I + O(h). \tag{3.16}$$

Otherwise, for $i \neq k$, (3.15) becomes

$$\frac{\partial U_{n,r}^{(i)}}{\partial u_{n+1}^{(k)}} = 0 + O(h). \tag{3.17}$$

Now, returning to (3.8), and using (2.26) we have for $i = 0, \ldots, p-2$,

$$\Phi_n^{(i)} = u_{n+1}^{(i)} - u_n^{(i)} - h \sum_{r=1}^{s} \omega_r U_{n,r}^{(i+1)}, \tag{3.18}$$

and taking partial derivatives with respect to $u_n^{(k)}$ gives

$$\frac{\partial \Phi_n^{(i)}}{\partial u_n^{(k)}} = -\frac{\partial u_n^{(i)}}{\partial u_n^{(k)}} - h \sum_{r=1}^{s} \omega_r \frac{\partial U_{n,r}^{(i+1)}}{\partial u_n^{(k)}}. \tag{3.19}$$

For $i = k$, substituting (3.14) into (3.19) we obtain the approximation

$$\begin{aligned} \frac{\partial \Phi_n^{(i)}}{\partial u_n^{(i)}} &= -I - h \sum_{r=1}^{s} \omega_r (0 + O(h)) \\ &= -I + O(h^2) \\ &\approx -I, \end{aligned} \tag{3.20}$$

and when $i+1=k$, we obtain from (3.19) using (3.13)

$$\begin{aligned}
\frac{\partial \Phi_n^{(i)}}{\partial u_n^{(i+1)}} &= -h\sum_{r=1}^{s}\omega_r\big((1-\gamma_r)I+O(h)\big) \\
&= \left(-h\sum_{r=1}^{s}\omega_r(1-\gamma_r)I\right)+O(h^2) \\
&= -h(1-\gamma)I+O(h^2) \\
&\approx -h(1-\gamma)I, \qquad (3.21)
\end{aligned}$$

where we have used

$$\sum_{r=1}^{s}\omega_r = 1, \qquad (3.22)$$

which follows from the order conditions (see, for example, [1]), and we have defined

$$\gamma = \sum_{r=1}^{s}\omega_r\gamma_r. \qquad (3.23)$$

Otherwise, from (3.19), using (3.14), when $i \neq k$ and $i+1 \neq k$

$$\begin{aligned}
\frac{\partial \Phi_n^{(i)}}{\partial u_n^{(k)}} &= -h\sum_{r=1}^{s}\omega_r(0+O(h)) \\
&= O(h^2) \\
&\approx 0. \qquad (3.24)
\end{aligned}$$

We proceed in a similar fashion for the other partial derivatives. Again from (3.18) for $i=0,\dots,p-2$ we obtain

$$\frac{\partial \Phi_n^{(i)}}{\partial u_{n+1}^{(k)}} = \frac{\partial u_{n+1}^{(i)}}{\partial u_{n+1}^{(k)}} - h\sum_{r=1}^{s}\omega_r\frac{\partial U_{n,r}^{(i+1)}}{\partial u_{n+1}^{(k)}}. \qquad (3.25)$$

For $i = k$, we obtain from (3.25) using (3.17)

$$\begin{aligned} \frac{\partial \Phi_n^{(k)}}{\partial u_{n+1}^{(k)}} &= I - h \sum_{r=1}^{s} (0 + O(h)) \\ &= I + O(h^2) \\ &\approx I, \end{aligned} \tag{3.26}$$

and, when $i + 1 = k$, we obtain from (3.25) using (3.16)

$$\begin{aligned} \frac{\partial \Phi_n^{(i)}}{\partial u_{n+1}^{(i+1)}} &= -h \sum_{r=1}^{s} \omega_r(\gamma_r I + O(h)) \\ &= \left(-h \sum_{r=1}^{s} \omega_r(\gamma_r I) \right) + O(h^2) \\ &= -h\gamma I + O(h^2) \\ &\approx -h\gamma I. \end{aligned} \tag{3.27}$$

Otherwise, for $i \neq k$ and $i + 1 \neq k$, we obtain from (3.25) using (3.17)

$$\begin{aligned} \frac{\partial \Phi_n^{(i)}}{\partial u_{n+1}^{(k)}} &= -h \sum_{r=1}^{s} \omega_r(0 + O(h)) \\ &= O(h^2) \\ &\approx 0. \end{aligned} \tag{3.28}$$

Finally, again returning to (3.8), we have for $i = p - 1$,

$$\Phi_n^{(p-1)} = u_{n+1}^{(p-1)} - u_n^{(p-1)} - h \sum_{r=1}^{s} \omega_r f(U_{n,r}^{(0)}, \ldots, U_{n,r}^{(p-1)}) \tag{3.29}$$

Taking partial derivatives with respect to $u_n^{(k)}$ gives

$$\begin{aligned} \frac{\partial \Phi_n^{(p-1)}}{\partial u_n^{(k)}} = & -\frac{\partial u_n^{(p-1)}}{\partial u_n^{(k)}} - h \sum_{r=1}^{s} \omega_r \Bigg(f_0(\ldots) \frac{\partial U_{n,r}^{(0)}}{\partial u_n^{(k)}} + f_1(\ldots) \frac{\partial U_{n,r}^{(1)}}{\partial u_n^{(k)}} + \ldots \\ & + f_{p-1}(\ldots) \frac{\partial U_{n,r}^{(p-1)}}{\partial u_n^{(k)}} \Bigg). \end{aligned} \tag{3.30}$$

Here we are using the abbreviation $f_i(\dots)$ to mean the partial derivative of the function $f(U_{n,r}^{(0)}, U_{n,r}^{(1)}, \dots, U_{n,r}^{(p-1)})$ with respect to the i^{th} block-variable, evaluated at $(U_{n,r}^{(0)}, U_{n,r}^{(1)}, \dots, U_{n,r}^{(p-1)})$. Consider now terms of the form

$$-h \sum_{r=1}^{s} \omega_r f_q(\dots) \frac{\partial U_{n,r}^{(q)}}{\partial u_n^{(k)}}, \tag{3.31}$$

where $q = 0, \dots, p-1$. When have, using (3.14),

$$\begin{aligned} -h \sum_{r=1}^{s} \omega_r f_q(\dots) \frac{\partial U_{n,r}^{(q)}}{\partial u_n^{(k)}} &= -h \sum_{r=1}^{s} \omega_r f_q(\dots)\big(0 + O(h)\big) \\ &= O(h^2) \\ &\approx 0. \end{aligned} \tag{3.32}$$

When $q = k$, we have, using (3.13),

$$\begin{aligned} -h \sum_{r=1}^{s} \omega_r f_k(\dots) \frac{\partial U_{n,r}^{(k)}}{\partial u_n^{(k)}} &= -h \sum_{r=1}^{s} \omega_r f_k(\dots)\big((1-\gamma_r)I + O(h)\big) \\ &= \Big(-h \sum_{r=1}^{s} (\omega_r f_k(\dots)(1-\gamma_r)I\Big) + O(h^2) \\ &\approx -h \sum_{r=1}^{s} \omega_r f_k(\dots)(1-\gamma_r)I \\ &\approx -h(1-\gamma) f_k(\dots), \end{aligned} \tag{3.33}$$

where in the last step we have taken all the evaluations of f_k at a single point in the sub-interval, typically one of the end-points or the centre, though any point will do. Now returning to (3.30), for $k = 0, \dots, p-2$ using first (3.32) then (3.33)

$$\begin{aligned} \frac{\partial \Phi_n^{(p-1)}}{\partial u_n^{(k)}} &= \Big(-h \sum_{r=1}^{s} (\omega_r (f_k(\dots) \frac{\partial U_{n,r}^{(k)}}{\partial u_n^{(k)}})\Big) + O(h^2) \\ &= \Big(-h \sum_{r=1}^{s} \omega_r f_k(\dots)(1-\gamma_r)I\Big) + O(h^2) \\ &\approx -h(1-\gamma) f_k(u_n^{(0)}, u_n^{(1)}, \dots, u_n^{(p-1)}). \end{aligned} \tag{3.34}$$

For the case $k = p-1$ in (3.30), similar substitutions of (3.32) and (3.33) give

$$\begin{aligned}
\frac{\partial \Phi_n^{(p-1)}}{\partial u_n^{(p-1)}} &= -I - \Big(h \sum_{r=1}^{s} \omega_r (f_{p-1}(\dots) \frac{\partial U_{n,r}^{(p-1)}}{\partial u_n^{(p-1)}}\Big) + O(h^2) \\
&= -I - \Big(h \sum_{r=1}^{s} \omega_r (f_{p-1}(\dots)(1-\gamma_r) I\Big) + O(h^2) \\
&\approx -I - h \sum_{r=1}^{s} \omega_r f_{p-1}(\dots)(1-\gamma_r) I \\
&\approx -I - h(1-\gamma) f_{p-1}(u_n^{(0)}, \dots, u_n^{(p-1)}). \qquad (3.35)
\end{aligned}$$

What remains are the partial derivatives of (3.29) with respect to u_{n+1}. Similar calculations to the preceding (which we omit) yield

$$\frac{\partial \Phi_n^{(p-1)}}{\partial u_{n+1}^{(k)}} \approx -h\gamma f_k(u_n^{(0)}, u_n^{(1)}, \dots, u_n^{(p-1)}) \qquad (3.36)$$

for $k = 0, \dots, p-2$, and

$$\frac{\partial \Phi_n^{(p-1)}}{\partial u_{n+1}^{(p-1)}} \approx I - h\gamma f_{p-1}(u_n^{(0)}, u_n^{(1)}, \dots, u_n^{(p-1)}) \qquad (3.37)$$

We can now construct our desired approximation for the Jacobian. Substituting (3.20), (3.21), (3.24), (3.26), (3.27), (3.28), (3.34), (3.35), (3.36), (3.37) into (3.5), we find that the block rows of the approximate Jacobian, which we call $\tilde{\Phi}'$, are of the form $\begin{bmatrix} 0 & \dots & 0 & \tilde{L} & \tilde{R} & 0 & \dots & 0 \end{bmatrix}$. Letting $\alpha = 1-\gamma$, we have

$$\tilde{L} = \begin{pmatrix} -I & -h\alpha I & & & \\ & -I & -h\alpha I & & \\ & & \ddots & & \\ & & & -I & -h\alpha I \\ -h\alpha f_0 & -h\alpha f_1 & \dots & -h\alpha f_{p-2} & -I - h\alpha f_{p-1} \end{pmatrix}, \qquad (3.38)$$

and

$$\tilde{R} = \begin{pmatrix} I & -h\gamma I & & & \\ & I & -h\gamma I & & \\ & & \ddots & & \\ & & & I & -h\gamma I \\ -h\gamma f_0 & -h\gamma f_1 & \dots & -h\gamma f_{p-2} & I - h\gamma f_{p-1} \end{pmatrix}. \tag{3.39}$$

Clearly the approximate Jacobian $\tilde{\Phi}'$ has a special sparse structure. Each of $\tilde{L}$ and $\tilde{R}$ consist of p block rows, of which only the last may be dense. The rest of $\tilde{L}$ or $\tilde{R}$ is Almost Block Diagonal (ABD) in structure; the blocks on the matrices main diagonal and the first upper diagonal contain the only nonzero elements. These nonzero blocks themselves are diagonal $m \times m$ blocks and all the elements on the diagonal are the same. $\tilde{L}$ and $\tilde{R}$ each contain $mp \times mp$ elements, but at most, $p \times m^2 + 2(p-1) \times m$ of these are non-zero. Contrast this to the exact Jacobian, in which L and R remain dense.

There are two areas in which the use of the approximate Jacobian in the Newton iteration has the potential to provide a savings when solving BVPs. The first source of savings arises from $\tilde{\Phi}'$ being cheaper to compute than Φ'. The second potential source of savings comes from the efficient solution of linear systems having coefficient matrices with the specific structure of $\tilde{\Phi}'$. The latter point is addressed in the next section.

3.2 Efficient system solves using the approximate Jacobian

In Section 3.1 we found the form of an approximate Jacobian matrix for p^{th}-order systems of ODEs. Here we examine in more detail the approximate Jacobian matrix for second-order systems. For this case, the structure of the $\tilde{L}$ and $\tilde{R}$ blocks simplifies and is no longer ABD, although the Jacobian matrix $\tilde{\Phi}'$ is itself still ABD. We examine a factoring algorithm that is commonly used for ABD matrices, and propose a faster algorithm particular to matrices with the special form of $\tilde{\Phi}'$.

The algorithms are illustrated by example, and we select for convenience the matrix of a system discretized over $N = 2$ mesh intervals. It is straightforward to show using (3.38) and (3.39) that for a second-order system of m equations, the Jacobian matrix has the form

$$\begin{pmatrix} B_1 & B_2 & & & & \\ -I & -h(1-\gamma)I & I & -h\gamma I & & \\ -h(1-\gamma)f_0 & -I-h(1-\gamma)f_1 & -h\gamma f_0 & I-h\gamma f_1 & & \\ & & -I & -h(1-\gamma)I & I & -h\gamma I \\ & & -h(1-\gamma)f_0 & -I-h(1-\gamma)f_1 & -h\gamma f_0 & I-h\gamma f_1 \\ & & & & B_3 & B_4 \end{pmatrix} \tag{3.40}$$

Each block is $m \times m$ and the B's come from the boundary conditions. Note that the f_0 and f_1 in the third block row differ from those in the fifth in that they are evaluated at different points. We assume for simplicity the number of boundary conditions is evenly divided between the end-points. With this assumption, the boundary blocks have the same size as the rest of the blocks in (3.40), so algebraic manipulations on the block-rows and block-columns of the matrix are convenient. Current codes might solve this system by modified alternate row-column elimination as discussed in [4]. This procedure is based on the familiar Gaussian elimination and has accuracy properties similar to Gaussian elimination, but alternates row and column eliminations to avoid fill-in, and so performs efficiently for matrices with an ABD structure. We point out that newer algorithms exist that also efficiently solve ABD systems, especially using parallelism, such as those based on stable local factorisations [15], and eigenvalue rescaling [9]. We are focussed, however, on the sequential case, and find that modified alternate row-column elimination is amenable to modifications to make use of additional structure in the system being solved. We outline this procedure briefly in the next subsection. In Subsections 3.2.2 and 3.2.3 we describe our specialised algorithm, and in Section 3.3 we discuss concerns relating to the accuracy of our solver.

3.2.1 Procedure 1: Modified alternate row-column elimination

To solve a linear system of the form

$$Ax = b \tag{3.41}$$

where A has the form of (3.40), modified alternate row-column elimination may be applied. This procedure involves alternately performing row eliminations and column eliminations on the matrix, to avoid fill-in in the zero-blocks of the ABD matrix A. However our A has an even more specialised form; the non-zero blocks themselves have a sparse structure, and this may be destroyed by the solution procedure.

The modified alternate row-column elimination procedure consists of first factoring the matrix A, and then solving for the required variables. The procedure begins by first performing column eliminations with partial column pivoting on the first two block-columns of A to put the system into the form

$$\begin{pmatrix} C_0 & & & & & \\ M_1 & S_{1a} & I & -h\gamma I & & \\ M_2 & S_{1b} & -h\gamma f_0 & I - h\gamma f_1 & & \\ & & -I & -h(1-\gamma)I & I & -h\gamma I \\ & & -h(1-\gamma)f_0 & -I - h(1-\gamma)f_1 & -h\gamma f_0 & I - h\gamma f_1 \\ & & & & B_3 & B_4 \end{pmatrix} \tag{3.42}$$

where C_0 is lower triangular. These column eliminations are analogous to the row eliminations that are commonly described in introductory linear algebra texts. Since we perform partial pivoting, column interchanges can take place anywhere within the subcolumns of the first two block columns.

The first block row and block column from the resulting matrix (3.42) are not manipulated in the following calculations, and we consider now the submatrix formed when they are dropped. Row eliminations with partial row pivoting are applied to the first two block rows of this submatrix (that is, to block rows two and three of the over-all matrix, but not affecting the first block column). The structure of the entire matrix is then

$$\left(\begin{array}{c|ccccc} C_0 & & & & & \\ \hline M_{1a} & R_1 & N_{1a} & N_{1b} & & \\ M_{1b} & & T_{1a} & T_{1b} & & \\ & & -I & -h(1-\gamma)I & I & -h\gamma I \\ & & -h(1-\gamma)f_0 & -I-h(1-\gamma)f_1 & -h\gamma f_0 & I-h\gamma f_1 \\ & & & & B_3 & B_4 \end{array}\right) \tag{3.43}$$

where R_1 is upper triangular. Unfortunately, we've lost the sparse structure of the second block row because of the pivoting. Since the pivoting may interchange any two subrows of block rows two and three of the matrix, elements from block row three may be swapped into block row two, and the subsequent row operations on this subrow may fill in all the zeros in the subrows below it. Even though blocks in the third and fourth block column of block row 2 are originally diagonal, they may in general become completely filled-in. Any advantage we might have had as a result of this sparsity is lost.

The process continues alternating between column and row eliminations resulting in the final form of the matrix

$$\begin{pmatrix} C_0 & & & & & & \\ M_{1a} & R_1 & N_{1a} & N_{1b} & & & \\ M_{1b} & & C_1 & & & & \\ & & M_{2a} & R_2 & N_{2a} & N_{2b} & \\ & & M_{2b} & & C_2 & & \\ & & & & M_{3a} & R_3 & \end{pmatrix} \tag{3.44}$$

The solution is computed by first solving for the variables corresponding to the odd-block rows in a forward fashion, then solving for the remaining variables in a backward fashion.

3.2.2 Procedure 2: Modified alternate row-column elimination with restricted column pivoting

The loss of structure in the first procedure is caused by pivoting. Pivoting is important for the calculation of accurate solutions. It is possible to compute cheaper solutions if we compromise the pivoting somewhat.

The procedure we outline in this section is a modification of the alternate row-column elimination procedure of the previous section. It increases efficiency by reducing the width of elements over which column pivoting is performed. This prevents certain blocks from being filled-in, thereby simplifying and speeding up the solution procedure.

Starting again with the original matrix (3.40), first block columns $2i+1$ are swapped with block columns $2i+2$, for $i=0$ through N. This small rearrangement prevents multiplication by large values when h is small in later operations. This rearrangement yields

$$\begin{pmatrix} B_2 & B_1 & & & & \\ -h(1-\gamma)I & -I & -h\gamma I & I & & \\ -I-h(1-\gamma)f_1 & -h(1-\gamma)f_0 & I-h\gamma f_1 & -h\gamma f_0 & & \\ & & -h(1-\gamma)I & -I & -h\gamma I & I \\ & & -I-h(1-\gamma)f_1 & -h(1-\gamma)f_0 & I-h\gamma f_1 & -h\gamma f_0 \\ & & & & B_4 & B_3 \end{pmatrix} \tag{3.45}$$

Now $-h(1-\gamma)I$ times the second block column is added to the first block column to produce

$$
\begin{pmatrix}
\tilde{B}_2 & B_1 & & & & \\
 & -I & -h\gamma I & I & & \\
\tilde{M} & -h(1-\gamma)f_0 & I-h\gamma f_1 & -h\gamma f_0 & & \\
 & & -h(1-\gamma)I & -I & -h\gamma I & I \\
 & & -I-h(1-\gamma)f_1 & -h(1-\gamma)f_0 & I-h\gamma f_1 & -h\gamma f_0 \\
 & & & & B_4 & B_3
\end{pmatrix}
\tag{3.46}
$$

Column eliminations are performed as usual, but only the *first* block column is used for pivoting instead of the *first two* block columns as usual. This preserves the structure of the second block row, and no operations need to be performed on it. We now have the matrix

$$
\begin{pmatrix}
\tilde{C}_0 & & & & & \\
 & -I & -h\gamma I & I & & \\
\tilde{M}_1 & \tilde{S}_1 & I-h\gamma f_1 & -h\gamma f_0 & & \\
 & & -h(1-\gamma)I & -I & -h\gamma I & I \\
 & & -I-h(1-\gamma)f_1 & -h(1-\gamma)f_0 & I-h\gamma f_1 & -h\gamma f_0 \\
 & & & & B_4 & B_3
\end{pmatrix}
\tag{3.47}
$$

The row eliminations are now performed, but since the block $-I$ in the second block row and column is already diagonal, we need only eliminate $\tilde{S}_1$ below it. Pivoting may cause the top half of the block $-I$ to fill in. The matrix is now

$$
\left(\begin{array}{c|ccccc}
\tilde{C}_0 & & & & & \\
\hline
 & \tilde{R}_1 & \tilde{N}_{1a} & \tilde{N}_{1b} & & \\
\tilde{M}_1 & & \tilde{T}_{1a} & \tilde{T}_{1b} & & \\
 & & -h(1-\gamma)I & -I & -h\gamma I & I \\
 & & -I-h(1-\gamma)f_1 & -h(1-\gamma)f_0 & I-h\gamma f_1 & -h\gamma f_0 \\
 & & & & B_4 & B_3
\end{array}\right)
\tag{3.48}
$$

The procedure is then repeated, adding $-h(1-\gamma)I$ times block column four to block column three. After all the eliminations are completed, the matrix looks like

$$\begin{pmatrix} \tilde{C}_0 & & & & & \\ & \tilde{R}_1 & \tilde{N}_{1a} & \tilde{N}_{1b} & & \\ \tilde{M}_1 & & \tilde{C}_1 & & & \\ & & & \tilde{R}_2 & \tilde{N}_{2a} & \tilde{N}_{2b} \\ & & \tilde{M}_2 & & \tilde{C}_2 & \\ & & & & \tilde{M}_3 & \tilde{R}_3 \end{pmatrix} \tag{3.49}$$

The solve stage of the algorithm is also simplified by the absence of nonzero values in the blocks below the $\tilde{C}$'s.

3.2.3 Procedure 3: Modified alternate row-column elimination with restricted column pivoting and simple block-row eliminations

A further simplification can be made at the level of the row eliminations. Instead of performing row eliminations to remove the S blocks, we could eliminate them by adding S_i times block row $2i$ to block row $2i+1$, for $i = 1$ through N. Starting from (3.47) again, this gives

$$\left(\begin{array}{c|ccccc} \tilde{C}_0 & & & & & \\ \hline & -I & -h\gamma I & I & & \\ \tilde{M}_1 & & \hat{T}_{1a} & \hat{T}_{1b} & & \\ & & -h(1-\gamma)I & -I & -h\gamma I & I \\ & & -I-h(1-\gamma)f_1 & -h(1-\gamma)f_0 & I-h\gamma f_1 & -h\gamma f_0 \\ & & & & B_4 & B_3 \end{array}\right) \tag{3.50}$$

Completing the eliminations in this way gives the final matrix

$$\begin{pmatrix} \tilde{C}_0 & & & & & \\ & -I & -h\gamma I & I & & \\ \tilde{M}_1 & & \hat{C}_1 & & & \\ & & & -I & -h\gamma I & I \\ & & \hat{M}_2 & & \hat{C}_2 & \\ & & & & \hat{M}_3 & \hat{R}3 \end{pmatrix} \tag{3.51}$$

Note that since the simple structure of the even block rows are never changed, these rows do not need to be stored explicitly, thereby reducing the storage space required. Also, the presence of the additional zero blocks below the C's, and the diagonal submatrices in the second and fourth block rows of (3.51) simplify the back and forward solve phase of the algorithm.

3.3 Accuracy of linear system solves

In the previous two subsections, we suggested two methods for solving the special ABD system of equations that arises when the approximate Jacobian of Section 3.1 is applied to a second-order system of ODEs. In doing so, we modified the pivoting strategy used by the factorisation. This change has the potential to affect the accuracy of the solutions.

Partial pivoting is normally applied by codes that do some form of Gaussian elimination in order to prevent element growth that can lead to a loss of accuracy and incorrect solutions. Given a matrix defined as

$$\begin{pmatrix} a_{1,1} & \dots & a_{1,m} & a_{1,m+1} & \dots & a_{1,2m} \\ \vdots & & \vdots & \vdots & & \vdots \\ a_{m,1} & \dots & a_{m,m} & a_{m,m+1} & \dots & a_{m,2m} \\ a_{m+1,1} & \dots & a_{m+1,m} & a_{m+1,m+1} & \dots & a_{m+1,2m} \\ \vdots & & \vdots & \vdots & & \vdots \\ a_{2m,1} & \dots & a_{2m,m} & a_{2m,m+1} & \dots & a_{2m,2m} \end{pmatrix} \tag{3.52}$$

suppose we apply column elimination with partial column pivoting (the column-wise analog of the more familiar row elimination with partial row pivoting). In the first step we calculate the pivot column by finding p such that

$$|a_{1,p}| = \max_{i=1..2m} |a_{1,i}| \tag{3.53}$$

Columns 1 and p are exchanged before performing column operations to eliminate the elements in columns 2 through $2m$ in row 1. This guarantees that the multipliers m_{1j} used in performing the eliminations satisfy

$$|m_{1j}| = \left| \frac{a_{1,j}}{a_{1,1}} \right| \leq 1 \tag{3.54}$$

and so the column operations in general do not cause excessive element growth. In the worst case element growth can be proportional to 2^{2m-1}, as shown in [7].

In our specialised method, we have arranged for the matrix on the bottom left quadrant to be zero, so the eliminations are carried out on matrices of the form

$$\begin{pmatrix} a_{1,1} & \dots & a_{1,m} & a_{1,m+1} & \dots & a_{1,2m} \\ \vdots & & \vdots & \vdots & & \vdots \\ a_{m,1} & \dots & a_{m,m} & a_{m,m+1} & \dots & a_{m,2m} \\ & & & a_{m+1,m+1} & \dots & a_{m+1,2m} \\ & & & \vdots & & \vdots \\ & & & a_{2m,m+1} & \dots & a_{2m,2m} \end{pmatrix}. \tag{3.55}$$

Using restricted partial column pivoting causes the pivot column to be calculated over the first m columns only, so now we find $\hat{p}$ such that

$$|a_{1,\hat{p}}| = \max_{i=1..m} |a_{1,i}| \tag{3.56}$$

and columns 1 and $\hat{p}$ are interchanged. However, in carrying out the column eliminations, it is possible that we have a multiplier greater than 1 for eliminations on columns $m+1$ to $2m$.

A possible modification to our procedure would check for unacceptably large multipliers and apply the normal pivoting strategy if they are present. In this case, we compute

$$m_{test} = \max_{i=m+1,2m} \frac{|a_{1,i}|}{|a_{1,\hat{p}}|}, \tag{3.57}$$

which is the largest multiplier over the last m columns. A simple test can then be applied, for example $m_{test} < MAXMULT$, which must be satisfied for the eliminations to continue. If the test fails, then the column that produced the largest multiplier is exchanged with the first column and the normal elimination scheme is used. The value of the constant $MAXMULT$ is a matter of tuning. In addition to addressing the problem of element growth, this modification also addresses a potentially worse case that arises in the event that $\tilde{B}_2$ in (3.46) is singular, which is possible depending on the particular boundary conditions for the problem being solved.

When partial column pivoting is applied, instead of restricted partial column pivoting, the set of row eliminations that follow must be performed with partial row pivoting instead of the simplified block-row operation of our specialised strategy. However, the next set of column eliminations may proceed as usual. The amount of savings that this method yields will depend on the number of blocks with which the new strategy can be successfully applied, and would vary from matrix to matrix.

Finally, we note that when the specialised strategy of procedure 3 is successfully applied, the row elimination phase is applied to blocks with the structure

$$\begin{pmatrix} -I & -h\gamma I & I \\ S_i & I - h\gamma f_1 & -h\gamma f_0 \end{pmatrix} \tag{3.58}$$

where each block is $m \times m$. The block S_i is eliminated by adding S_i times the first block row to the second block row. Note here that if $h\gamma \leq 1$, which would be the case as the mesh is refined and h becomes smaller, then all the values in the first block-row are less than or equal to one in magnitude. Hence multiplication by S_i should not cause excessive element growth and we expect that the lack of pivoting here will not cause accuracy problems.

We have not made any guarantees on the stability of this linear-equation solver. With the modifications mentioned above, we only attempt to control error growth while still retaining a gain in efficiency.

3.4 Computational efficiency

A calculation of the number of floating point operations required by the methods we suggest is instructive in evaluating their potential merit. In the following sections we present operation counts for performing a typical row-column elimination, and for performing our fastest specialised version, procedure 3. Also presented are the costs of evaluating the approximate Jacobian, the complete Jacobian, which we denote MIRK-S, and also a commonly used approximation to MIRK-S denoted MIRK-1. The MIRK-1 approximation differs from the MIRK-S Jacobian in that only one function evaluation per sub-interval is performed, instead of the usual s evaluations. These are theoretical counts. In practice, the cost of calculating array indices and computing loop counters contributes only a small amount to the total cost, and we therefore ignore them. However, these costs may be significant on small systems of ODEs, an effect we will observe during testing in a later section.

Both the use of the approximate Jacobian and performing the the use of the specialised system solver yield savings on each iteration of the Newton iteration process. However we may pay a penalty for having to compute extra iterations to get a good solution, especially for coarse meshes on which the approximate Jacobian might differ significantly from the exact Jacobian. This tradeoff is examined in numerical testing later.

The following results apply to second-order systems. We count floating-point operations (FLOPs), and define a FLOP to be the cost of performing a singe arithmetic operation.

3.4.1 System solves

The total cost of system solves is divided in two parts: performing the factorisation, and solving for the variables. The cost of a single factorisation using procedure 1 on a matrix A computed from and ODE system of size m discretized over N intervals, is $N(\frac{23}{3}m^3 - m^2 - \frac{2}{3}m) + \frac{8}{3}m^3 - \frac{2}{3}m$ FLOPs. Procedure 3 requires $N(\frac{7}{3}m^3 - \frac{1}{3}m + 3m^2) + \frac{8}{3}m^3 - \frac{2}{3}m$ FLOPs. The solve phase of procedure 1 costs $N(18m^2 + 3m) + 9m^2 + m$ FLOPs, while for procedure 3 it costs $N(11m^2 + 5m) + 7m^2$ FLOPs.

The factorisation portion of the algorithms dominates the total cost. In both cases, we achieve a savings in floating-point operations by using procedure 3. For large systems and mesh sizes the dominant terms are $\frac{23}{3}Nm^3$ for procedure 1 and $\frac{7}{3}Nm^3$ for procedure 3, so the relative improvement between the two can be as much as 70%.

3.4.2 Evaluation of Jacobians

The approximate Jacobian is simpler to compute than either the MIRK-S or MIRK-1 Jacobians. All require the same computation of partial derivatives of the boundary conditions. In addition, both MIRK-1 and the approximate Jacobian require one evaluation one each of the N subintervals of the system Jacobians $\partial F/\partial w$ from (2.26). The MIRK-S Jacobian, however, requires s evaluations per subinterval, and also requires the computation of s stages on each subinterval, if they are not already available.

The algebraic cost of computing the Jacobians is presented next. This assumes that the boundary conditions and the system Jacobians mentioned above have already been evaluated. Under this assumption, the MIRK-S and MIRK-1 Jacobians both require about $N(16(s-1)m^3 + 4(s^2+s)m^2)$ FLOPs. The approximate Jacobian requires only $N(4m^2 + 4m)$ FLOPs. Clearly, the latter is better by an order of magnitude in m, does not contain factors s or s^2, and therefore represents a significant computational savings.

It should be noted that the calculation of the Jacobians mentioned here require many evaluations of the Jacobian $\partial F/\partial w$ of the reduced ODE (2.27). The cost of these evaluations are not included in the above calculations, and in practice we may find that this cost actually dominates the overall cost, depending on the complexity of the BVP being solved.

Chapter 4

Numerical testing

4.1 A simple test implementation

Simple BVP solvers were implemented using MATLAB [11] to test the behaviour and performance of the experimental solution techniques described in the previous chapter. Two types of nonlinear system solvers were used in the tests: Newton's method and a damped Newton method based on one presented in [1]. The details of these non-linear system solvers are given in Tables 4.1 and 4.2. The main algorithm of the BVP solver is given in Table 4.3. For the tests, we used values of $TOL = 10^{-7}$, $MAXITS = 40$, and $MAXN = 10^6$.

Tests conducted using the approximate Jacobian are denoted NEW-A in the tables containing the test results. For comparison, test results using the exact Jacobian, denoted MIRK-S, are presented. Also, test results using a common approximation to the exact Jacobian, denoted MIRK-1, are also presented. MIRK-1 differs from MIRK-S in that a single function evaluation per subinterval is made instead of the s function evaluations used by MIRK-S.

The modified row-column elimination strategy of procedure 1 was used to solve the linear systems for each of the three coefficient matrices. In addition, tests were conducted using the specialised solver of procedure 3 in conjunction with the approximate Jacobian, and results from these tests are labelled NEW-B in the tables.

The codes were designed to solve BVPs consisting of systems of second-order

ODEs, with an equal number of boundary conditions at both end-points. For each problem, the codes require functions which evaluate the system $f(u^{(0)}, u^{(1)})$, its derivatives $\partial f/\partial u^{(1)}$ and $\partial f/\partial u^{(0)}$, as well as the boundary conditions g_a and g_b and their derivatives with respect to $u^{(0)}$ and $u^{(1)}$. The codes also require an initial guess for the solution, which in these tests was a straight line through the boundary conditions. This initial guess was sufficient for the problems considered in our testing.

Three test problems are presented. Most of the tests use a uniform mesh. The number of mesh intervals N ranges from 2 to 64. For two of the problems, additional tests on a fine nonuniform mesh are also included. We report the number of Newton iterations required for each solution, the error in the final solution compared to a solution obtained from MIRKDC [6] for the same problem computed with a tolerance of 10^{-8}, and we report its cost in terms of number of floating-point operations. The MATLAB *flops* command was used to obtain the latter result. For each of MIRK-1, NEW-A, and NEW-B we also compute the savings as a percentage of the cost of applying MIRK-S.

- Input: A Matrix A and a vector b, and current approximate solution W.
- Output: an updated approximate solution W, flag $status$, and norm n.
- Algorithm
  1. Set $status = 0$.
  2. Solve $A\delta W = -b$ using a linear system solver, either procedure 1 or procedure 3.
  3. Compute the norm $n = \|\delta W\|^2$.
  4. Set $W = W + \delta W$.

Table 4.1: Algorithm for calculating a single Newton step

The actual MIRK method used in the tests is the five-stage sixth-order method defined by the tableau given in Table 4.4, which is also the method used by the package MIRKDC [6].

- Input: A Matrix A, vector b, iteration counter k, and current approximate solution W.
- Output: An updated approximate solution W, flag $status$, and norm n.
- Algorithm
  1. Set $status = 0$, $\sigma = 0.01$, $\lambda_{min} = 0.01$, $\tau = 0.1$.
  2. Solve $A\delta W = -b$ using a linear system solver, either procedure 1 or procedure 3. Temporarily store factors of A for additional solves when evaluating g.
  3. Compute $g_0 = \frac{1}{2}\|\delta W\|^2$.
  4. Predict λ_k using
  $$\lambda_k = \begin{cases} \lambda_{k-1} & \text{if} \quad \lambda_{k-1} < \lambda_{k-2}(1-\sigma) \\ \min(1, 2\lambda_{k-1}) & \text{otherwise} \end{cases}$$
  5. Loop:
     (a) Compute $\hat{W} = W + \lambda_k \delta W$
     (b) Compute $g_\lambda(\hat{W})$ using (2.19).
     (c) Accept λ_k if $g_\lambda \leq (1 - 2\lambda_k\sigma)$, exit loop.
     (d) Otherwise, update λ_k using
     $$\lambda_k = \max\left(\tau\lambda_k, \frac{\lambda_k^2 g_0}{(2\lambda_k - 1)g_0 + g_\lambda}\right)$$
     (e) Failure if $\lambda_k < \lambda_{min}$, set $status = 4$, exit.
  6. Set $W = \hat{W}$.
  7. Compute the norm $n = \|\delta W\|_2$.

Table 4.2: Algorithm for computing a single damped Newton step

- Input: An initial mesh and initial solution W.
- Output: A final solution W and flag $status$.
- Algorithm
  1. Set iteration counter k=0.
  2. Loop:
     (a) Calculate the residual function $b = \Phi(W)$.
     (b) Compute the Jacobian $A = \Phi'(W)$ for the MIRK-S or MIRK-1 Jacobians, or $A = \tilde{\Phi}'(W)$ for the approximate Jacobian of Section 3.1.
     (c) Take a Newton step or damped Newton step with the values (A, b, W), returning an updated W, flag $status$, and norm n.
        - Failure if $status = 4$, exit.
        - Success if $n < TOL$, set $status = 1$, exit.
        - Failure if $k > MAXITS$, $status = 2$, exit.
        - Failure if $n > MAXN$, $status = 3$, exit.
     (d) increment iteration counter k.

Table 4.3: Main algorithm for solving a BVP

$$
\begin{array}{cc|ccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{4} & \frac{5}{32} & \frac{9}{64} & -\frac{3}{64} & 0 & 0 & 0 \\
\frac{3}{4} & \frac{27}{32} & \frac{3}{64} & -\frac{9}{64} & 0 & 0 & 0 \\
\frac{2}{5} & \frac{1}{2} & -\frac{5}{24} & \frac{5}{24} & \frac{2}{3} & -\frac{2}{3} & 0 \\
\hline
 & & \frac{7}{90} & \frac{7}{90} & \frac{32}{90} & \frac{32}{90} & \frac{12}{90}
\end{array}
$$

Table 4.4: five-stage sixth-order MIRK formula

4.2 Test problems and results

4.2.1 Daniel and Martin 75

A simple test problem from [3] is

$$y'' = \frac{(y+x+1)^3}{2}, \qquad 0 < x < 1, \tag{4.1}$$

$$y(0) = 0, \quad y(1) = 0, \tag{4.2}$$

which has the solution

$$y(x) = \frac{2}{(2-x)} - x - 1, \quad y'(x) = \frac{2}{(2-x)^2} - 1. \tag{4.3}$$

This problem was tested a second time, with the original system of equations repeated 10 times to make the system artificially larger. This is useful in observing the improvement in performance which we expect for systems of larger size m.

4.2.2 Kreiss '81

Kreiss '81 [10] is a singular perturbation problem which has a parameter with which we can control difficulty. The problem is defined by

$$y'' = \frac{(1-y')y}{\epsilon}, \qquad 0 < x < 1, \tag{4.4}$$

for ϵ a small parameter. The boundary conditions are

$$y(-1) = -1, \quad y(1) = 2. \tag{4.5}$$

No closed form solution is available for this problem.

4.2.3 Shock Wave

This problem is from [1]. It also contains a parameter with which we can control difficulty. The problem is defined by

$$\epsilon A(x)uu'' - [1 + \frac{\gamma}{2} - \epsilon A'(x)]uu' + \frac{u'}{u} + \frac{A'(x)}{A(x)u}(1 - \frac{\gamma - 1}{2}u^2) = 0, \ 0 < x < 1, \quad (4.6)$$

$$A(x) = 1 + x^2, \quad \gamma = 1.4, \quad (4.7)$$

and

$$u(0) = 0.9129, \quad u(1) = 0.375. \quad (4.8)$$

4.3 Test results for the approximate Jacobian

In this section we review test results from the use of the approximate Jacobian, (labelled NEW-A in the tables), and compare them to the results when the MIRK-1 and MIRK-S Jacobians are used. We defer our discussion of the effects of the specialised linear system solver to a following section.

4.3.1 Effects of system size m

A significant savings in cost is possible for BVPs of large size m. For the Daniel and Martin problem repeated 10 times, we see from the results presented in Table A.2, NEW-A is able to achieve a savings of more than 90% for all mesh sizes compared to MIRK-S. In comparison, the results for the unrepeated system in Table A.1 shows a savings of 65% for meshes of size 16 and larger. the results of Section 3.4 for $m = 10$ and $s = 5$, the ratio of algebraic costs for evaluating the approximate Jacobian to evaluating the complete Jacobian is roughly $\frac{1}{200}$. Thus we should achieve a savings of close to 100% in the Jacobian evaluation portion of the algorithm. Evidently this portion dominates the overall cost of computing a solution to the BVP.

Similar significant savings are shown for the Kreiss test problem repeated 10 times in Tables A.17 and A.18, and for the Shock Wave test problem repeated 10 times in Tables A.31 and A.32.

4.3.2 Effects of mesh size

Since an approximation to the Jacobian is being used in which we discard terms containing second or higher powers of the mesh width h, it is expected that the iterates from using the approximate Jacobian would behave more like the iterates from using the true Jacobian (MIRK-S) as h becomes smaller. This effect appears clearly in the test data. Refer for example to the Daniel and Martin problem results given in Table A.1, and in particular to the results for the coarsest mesh, in which there are $N = 2$ mesh intervals. The MIRK-S Jacobian converges in 4 iterations, while the approximate Jacobian tested in NEW-A requires 8 iterations, twice as many. However, the difference in the number of Newton iterations decreases between the two methods as the mesh becomes finer. Both methods take 4 iterations when a mesh of $N = 16$ subintervals is used. This effect can also be observed in Tables A.5, A.6, A.8, and A.9 for the Kriess test problem, and in Tables A.19, A.20, A.22, and A.23 for the Shock Wave test problem.

Again referring to Table A.1 for the Daniel and Martin problem, note that, even while doing twice as many iterations as MIRK-S for $N = 2$, NEW-A still achieves roughly a 25% performance advantage. At $N = 16$, the advantage is about 65%, and number of Newton iterations and the savings stay fixed at this amount for meshes with larger N. That is, the performance of the two methods does not depend on the mesh size except in its effect on the convergence of the Newton iterations. This is as expected since in the calculation of computational costs, the Jacobian construction costs for all the methods contained a common factor of N.

A significant result is apparent from simple cases of the Kreiss problem. For the case shown in Table A.5 in which $\epsilon = 0.1$, we note that for $N = 4$ and $N = 8$ mesh intervals, MIRK-S converged to a solution while NEW-A failed to reach a solution. It is apparent from this that if Newton iterations with the true Jacobian converge, it is not necessarily true that Newton iterations with the approximate Jacobian will also converge. The same holds for damped Newton iterations, as shown in Table A.8 with $N = 4$, and similarly for the Shock Wave problem in Tables A.19 and A.22. However, again as N becomes larger the mesh widths become smaller and the approximate Jacobian performs better. In Table A.5, convergence requires only 1 extra iteration for $N = 64$. At this last mesh width we save roughly 60% in overall cost.

The rates of convergence of the three methods are qualitatively illustrated in a set of plots. Given that a sequence of n Newton iterations lead to a solution W^*, we plot the norms $\|W^{[i]} - W^*\|_2$, for $i = 0$ through $n-1$. Graphs for MIRK-S, MIRK-1, and NEW-A are presented on the same axes. Referring to Figures A.9 and A.10, we can see that the results for the approximate Jacobian better approximate the results for the true Jacobian when the mesh width is halved. Note that the NEW-A results actually come to approximate the MIRK-1 results quite closely, while the results for MIRK-S remains slightly better. This is reasonable in light of the fact that in the derivation of the approximate Jacobian, the simplification of only one function evaluation per sub-interval was made, as is also the case for the MIRK-1 Jacobian. We can thus view NEW-A as an approximation to MIRK-1, which is itself an approximation to MIRK-S. Note also in Figures A.9 and A.10 that the iterations with the true Jacobian appear to be converging quadratically, while the others converge linearly. Finally, we can estimate the order of the underlying MIRK formula we are using with some quick calculations using the values in the *error* column of tables such as A.5, noting that the h is being halved with each new mesh. We obtain an estimate of approximately 5.5, which confirms that our sixth-order MIRK formula is performing as expected.

4.3.3 Effects of using damped Newton steps

Experiments with both Newton and damped Newton iterations were conducted. It was observed that the damped Newton iterations lead to the various methods converging for coarser meshes than for the undamped case, seen for example in Table A.6 compared to Table A.9, as would be expected. The damped Newton iteration seemed to have a particularly positive affect on the MIRK-S method, sometimes allowing it to converge much faster than the others; see for example Table A.9. In fact, MIRK-S was cheaper in overall cost than the approximate Jacobian when both converged for some cases. As h became smaller, however, use of the approximate Jacobian did lead to a savings. Note also that for the particular example mentioned above, the system size is $m = 1$, and for larger systems as in Tables A.17 and A.18, using the approximate Jacobian still leads to significant savings, even though the iteration using the MIRK-S Jacobian converges much faster than the iteration using the approximate Jacobian.

4.3.4 Effects of problem difficulty

For the Shock Wave and Kreiss test problems, a parameter ϵ is available with which we are able to control difficulty. Comparing the results for $\epsilon = 0.1$, 0.05, and 0.01 contained in Tables A.5, A.6, A.7 and A.8, A.9, A.10 for the Kreiss problem, and Tables A.19, A.20, A.21 and A.22, A.23, A.24 for the Shock Wave problem, we find that as the problems become more difficult, all the methods have convergence difficulties for successively finer meshes. The approximate Jacobian iterations do not appear to suffer more severely than the other methods, and is able to achieve savings when it does converge. None of the methods were able to converge for $\epsilon = 0.005$ using uniform meshes for either the Shock Wave or Kreiss test problems. However, given a very good starting guess obtained from the true solution to the problem for $\epsilon = 0.006$, MIRK-S did succeed in converging for both damped and non-damped cases, though to poor solutions, while the others still diverged as shown in Tables A.13 and A.14. The same is true for the Shock Wave problem as is shown in Tables A.27 and A.28. We conclude that it is better for some meshes to use the complete Jacobian of MIRK-S so that the method will converge to a solution before refining to a finer mesh. Eventually, for finer meshes, NEW-A becomes effective again.

Further tests on a fine non-uniform mesh were conducted for the Kreiss and Shock Wave test problems for $\epsilon = 0.005$. The fine mesh was obtained from the output from MIRKDC for the same problem. Both damped and full Newton iterations were used. Results are shown in Tables A.11, A.12, A.25, and A.26. For all of these except the Kreiss problem with damped Newton iterations, the MIRK-S Jacobian failed to converge. This is an unexpected result and unexplained at this point in our analysis. The approximate Jacobian was unable to converge for the Kreiss test problems, but succeeded in converging for the Shock Wave problem. The same problems were solved using a good initial guess, obtained from solving the same problems for $\epsilon = 0.006$, shown in Tables A.15, A.16, A.29, and A.30. We observe for most cases of the good starting guess, the approximate Jacobian converged in almost the same number of iterations as the complete Jacobian and attained a computational savings. The exception is the Kreiss problem with full Newton iterations shown in Table A.15, in which case the approximate Jacobian failed to converge to a solution. It is apparent that some of the convergence difficulties when using the approximate Jacobian are overcome when close enough to the final solution.

In Section 2.4, we stated a condition (2.23) for the convergence of an iterative method. The associated spectral radius parameter was calculated for both the MIRK-1 and approximate Jacobians for a series of meshes, and plotted against mesh size for various problems. See Figures A.2, A.4, A.6, and A.8 for the Kreiss problem and Figures A.15, A.17, A.19, and A.21 for the Shock Wave problem. It is apparent that for some of the coarser meshes the spectral radii are larger than one, which would indicate that the Newton iteration at these meshes may not converge even when starting very close to the the solution. As the problems become more difficult, the spectral radii are larger than one even for finer meshes.

Also included in our test results are plots of the condition numbers of the Jacobian matrices, for example in Figures A.1 and A.3 for the Kreiss problem and Figures A.14 and A.16. These plots may be useful in assessing the difficulty of the problems attempted. The larger condition numbers correspond to more difficult problems. Comparing these plots, we can see the effect of smaller values of the parameter ϵ on the difficulty of the problems attempted.

4.4 Test results for the specialised solver

We now turn to the results of using the specialised linear system solver, shown in the tables as NEW-B. The benefits of using the specialised solver are sensitive to the system size m and are marginal compared to the benefits from using the approximate Jacobian. For example in Table A.5 for the Kreiss problem, NEW-B actually performed worse than NEW-A. This is because, for $m = 1$, the extra work done performing block column manipulations is not offset by any benefit. The Daniel and Martin problem provides more positive results for the large system with $m = 10$ in Table A.2. In this case, if we compare the results of the two methods directly rather than indirectly through MIRK-S, NEW-B yields savings of 23% for $N = 64$ over NEW-A. Similar improvements are observed for the repeated Kreiss problem in Table A.17, and the repeated Shock Wave problem in Table A.31.

Finally, we note that we did not observe additional convergence difficulties when using the specialised solver. This is noted since no special care was taken in the test implementations to guarantee accuracy as described in Section 3.3. However, such considerations need to be taken in a production code, and would be expected to have a negative impact on the performance of the specialised row-column elimination procedure. Even so, we expect to obtain better cost performance than the standard approach, especially for large systems.

4.5 Miscellaneous considerations

A few factors must be mentioned before drawing conclusions from these test results. The test codes used were relatively simple. The implementation of MIRK-S could be made more efficient by saving the values of the stages when they are computed for evaluation of the residual Φ, instead of computing them again, although MIRK-S at best would still cost more to evaluate that MIRK-1. We have not considered the possible benefits of using fixed Jacobian steps for some of the Newton iterations on any of the methods, as would be done in a production version of the software.

Chapter 5

Conclusions and future work

5.1 Conclusions

Our numerical test results show it may be possible to obtain a significant reduction in the number of floating-point operations required when using MIRK methods to solve BVPs. These savings are made possible by exploiting the structure of ODEs that arise from the reduction of higher-order systems of equations to first-order. Most of the improvement is from the use of an approximate Jacobian $\tilde{\Phi}'$ instead of the true Jacobian Φ' of a discretized BVP. A smaller improvement is from the use of a specialised row-column elimination method to solve the linear systems that arise when solving the discretized BVP.

Using approximate Jacobians can have the drawback of increasing the number of Newton iterations required to compute a solution to the BVP. For some cases, the solver that uses the approximate Jacobian does not converge while the one that uses the exact Jacobian does. This is particularly noticeable on coarser meshes. However, as the meshes become finer, the approximate Jacobian solver becomes more accurate and performs better. Moreover, in our tests, the cost of performing the extra iterations was usually outweighed by the savings from constructing the cheaper approximate Jacobian at each iteration. Hence the use of the approximate Jacobian has a positive effect on performance. We were able to obtain improvements of 90% for some problems with large system size m. The problem of convergence can be addressed by adaptively applying a full MIRK-S Jacobian when the solver with the approximate Jacobian fails to converge. For iterations with finer meshes, the approximate Jacobian may again provide more efficient solutions, if it is able to converge. For any particular problem, the effectiveness of applying the approximate Jacobian appears to depend on the size of the mesh widths and the difficulty of the problem.

Because of possible convergence difficulties with the Newton iteration using the approximate Jacobian, we expect that at times full Jacobian evaluations may be necessary for some problems. Further, it is not clear how the mesh refinement strategies used by most codes to compute solutions efficiently to a desired accuracy would affect the performance of the approximate Jacobian solver. It may be that for many cases a refined mesh would lead to some mesh intervals being sufficiently small for the approximation to be effective, while some other mesh intervals may remain relatively large.

Using a specialised solver may result in computational savings, but these are typically less significant than the savings from evaluating the approximate Jacobian, and only become of value when the system of ODEs is large. Investigation of element growth resulting from the use of restricted partial pivoting would be useful in determining the observed stability of this technique.

5.2 Future work

We have presented the general form of an approximate Jacobian for p^{th}-order systems of equations, and tested it on the particular case of second-order systems. The general form needs to be extended to include mixed-order systems, and testing on systems of higher-order is desirable. For these higher-order systems, a large amount of information is discarded in producing the approximation, and its effect on performance requires further evaluation. The implementation of this work is straightforward in light of what has already been provided here. The development of an adaptive technique to alternate between complete and approximate Jacobian evaluations may prove to be of practical use.

Of secondary interest is the identification of a general method to efficiently solve the linear systems arising when the approximate Jacobian is used. The solver we have developed, Procedure 3, works for the particular case of second-order systems with separated boundary conditions and the same number of conditions at each end-point of the problem interval. A better solver would work on higher-order and mixed-order systems, and more general boundary conditions. Unfortunately, our solver generalises to an unusual arrangement of boundary conditions for problems of order greater than two.

Since the dominant portion of the computational savings we have observed arises from using the approximate Jacobian, its incorporation into a production code without the additional benefit of a corresponding specialised solver could be a practical outcome of our work. Once such a code is developed, the potential additional savings arising from the use of a specialised solver should become significant enough to merit further investigation.

Bibliography

[1] U.M. Ascher, R.M.M. Mattheij and R.D. Russell, *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*, Prentice Hall, Englewood Cliffs, 1988.

[2] K. Burrage, F.H. Chipman and P.H. Muir, *Order results for mono-implicit Runge-Kutta methods*, SIAM J. Numer. Anal., Vol. 31, No. 3, pp. 876-891, 1994.

[3] J.W. Daniel and A.J. Martin, *Implementing deferred corrections for Numberov's difference method for second-order two-point boundary value problems*, Center for Numerical Analysis, The University of Texas at Austin, Technical Report CNA-107, November 1975.

[4] J.C. Diaz, G. Fairweather and P. Keast, *FORTRAN packages for solving certain almost block diagonal linear systems by modified alternate row and column elimination*, ACM Trans. Math. Software, Vol. 9, No. 3, pp. 358-375, 1983.

[5] W.H. Enright and Min Hu, *Improving performance when solving high order and mixed order boundary value problems in ODEs*, Department of Computer Science Technical Report, University of Toronto, 1995.

[6] W.H. Enright and Paul Muir, *A Runge-Kutta type boundary value ODE solver with defect control*, Department of Computer Science Technical Report 267/93, University of Toronto, 1993.

[7] Gene H. Golub and Charles F. van Loan, *Matrix Computations*, The John Hopkins University Press, 1983.

[8] Ken Jackson, *Solving higher order ODEs in a BVP code: some preliminary remarks*, unpublished note, 1993.

[9] K.R. Jackson and R.N. Pancer, *The parallel solution of ABD systems arising in numerical methods for BVPs for ODEs*, Department of Computer Science Technical Report 255/91, University of Toronto, 1992.

[10] Barbro Kreiss and Heinz-Otto Kreiss, *Numerical methods for singular perturbation problems*, SAM J. Numer. Anal., Vol. 18, No. 2, April 1981.

[11] The MathWorks, Inc., *MATLAB*, 24 Prime Park Way, Natick, MA 01760.

[12] Paul Muir, *Implicit Runge-Kutta methods for two-point boundary value problems*, Department of Computer Science Technical Report 175/84, University of Toronto, 1984.

[13] Paul Muir and Karin Remington, *A parallel implementation of a Runge-Kutta code for systems of nonlinear boundary value ODEs*, Congressus Numerantium 99, pp. 291-305, 1994.

[14] J. M. Ortega and W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York and London, 1970.

[15] S. J. Wright, *Stable parallel algorithms for two-point boundary value problems*, SIAM J. Sci. Statist. Comput., Vol.13, No. 2, pp. 742-764, 1992.

Appendix A

Numerical test data

A.1 Daniel and Martin problem test results

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	2	4	6.640e-05	1.137e+04	
MIRK-1:	2	6	6.640e-05	1.196e+04	-5.2%
NEW-A:	2	8	6.640e-05	8.416e+03	26.0%
NEW-B:	2	8	6.640e-05	8.976e+03	21.0%
MIRK-S:	4	4	1.765e-06	2.215e+04	
MIRK-1:	4	5	1.765e-06	1.918e+04	13.4%
NEW-A:	4	5	1.765e-06	9.850e+03	55.5%
NEW-B:	4	5	1.765e-06	1.044e+04	52.9%
MIRK-S:	8	4	4.106e-08	4.372e+04	
MIRK-1:	8	5	4.105e-08	3.762e+04	13.9%
NEW-A:	8	5	4.105e-08	1.903e+04	56.5%
NEW-B:	8	5	4.105e-08	2.010e+04	54.0%
MIRK-S:	16	4	9.119e-10	8.686e+04	
MIRK-1:	16	4	9.067e-10	5.960e+04	31.4%
NEW-A:	16	4	9.263e-10	2.991e+04	65.6%
NEW-B:	16	4	9.263e-10	3.154e+04	63.7%
MIRK-S:	32	4	2.009e-11	1.731e+05	
MIRK-1:	32	4	1.978e-11	1.186e+05	31.5%
NEW-A:	32	4	1.996e-11	5.929e+04	65.8%
NEW-B:	32	4	1.996e-11	6.245e+04	63.9%
MIRK-S:	64	4	4.426e-13	3.457e+05	
MIRK-1:	64	4	4.188e-13	2.366e+05	31.5%
NEW-A:	64	4	4.327e-13	1.180e+05	65.9%
NEW-B:	64	4	4.327e-13	1.243e+05	64.0%

Table A.1: Results for the Daniel and Martin problem, using full Newton iterations

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	2	4	2.100e-04	1.636e+06	
MIRK-1:	2	6	2.100e-04	1.207e+06	26.2%
NEW-A:	2	8	2.100e-04	1.233e+05	92.5%
NEW-B:	2	8	2.100e-04	1.017e+05	93.8%
MIRK-S:	4	4	5.580e-06	3.259e+06	
MIRK-1:	4	5	5.580e-06	1.997e+06	38.7%
NEW-A:	4	6	5.580e-06	1.668e+05	94.9%
NEW-B:	4	6	5.580e-06	1.335e+05	95.9%
MIRK-S:	8	4	1.298e-07	6.506e+06	
MIRK-1:	8	5	1.298e-07	3.980e+06	38.8%
NEW-A:	8	5	1.298e-07	2.630e+05	96.0%
NEW-B:	8	5	1.298e-07	2.066e+05	96.8%
MIRK-S:	16	4	2.884e-09	1.300e+07	
MIRK-1:	16	4	2.867e-09	6.355e+06	51.1%
NEW-A:	16	5	2.884e-09	5.110e+05	96.1%
NEW-B:	16	5	2.884e-09	3.974e+05	96.9%
MIRK-S:	32	4	6.352e-11	2.599e+07	
MIRK-1:	32	4	6.255e-11	1.270e+07	51.1%
NEW-A:	32	4	6.313e-11	8.055e+05	96.9%
NEW-B:	32	4	6.313e-11	6.231e+05	97.6%
MIRK-S:	64	4	1.399e-12	5.196e+07	
MIRK-1:	64	4	1.324e-12	2.538e+07	51.1%
NEW-A:	64	4	1.368e-12	1.599e+06	96.9%
NEW-B:	64	4	1.368e-12	1.234e+06	97.6%

Table A.2: Results for the Daniel and Martin problem, repeated 10 times, using full Newton iterations

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	2	4	6.640e-05	1.404e+04	
MIRK-1:	2	6	6.640e-05	1.596e+04	-13.7%
NEW-A:	2	8	6.640e-05	1.375e+04	2.0%
NEW-B:	2	8	6.640e-05	1.432e+04	-2.0%
MIRK-S:	4	4	1.765e-06	2.719e+04	
MIRK-1:	4	5	1.765e-06	2.548e+04	6.3%
NEW-A:	4	5	1.765e-06	1.614e+04	40.6%
NEW-B:	4	5	1.765e-06	1.667e+04	38.7%
MIRK-S:	8	4	4.106e-08	5.349e+04	
MIRK-1:	8	5	4.105e-08	4.984e+04	6.8%
NEW-A:	8	5	4.105e-08	3.124e+04	41.6%
NEW-B:	8	5	4.105e-08	3.211e+04	40.0%
MIRK-S:	16	4	9.119e-10	1.061e+05	
MIRK-1:	16	4	9.067e-10	7.885e+04	25.7%
NEW-A:	16	4	9.263e-10	4.916e+04	53.7%
NEW-B:	16	4	9.263e-10	5.039e+04	52.5%
MIRK-S:	32	4	2.009e-11	2.113e+05	
MIRK-1:	32	4	1.978e-11	1.568e+05	25.8%
NEW-A:	32	4	1.996e-11	9.748e+04	53.9%
NEW-B:	32	4	1.996e-11	9.980e+04	52.8%
MIRK-S:	64	4	4.426e-13	4.217e+05	
MIRK-1:	64	4	4.188e-13	3.127e+05	25.9%
NEW-A:	64	4	4.327e-13	1.941e+05	54.0%
NEW-B:	64	4	4.327e-13	1.986e+05	52.9%

Table A.3: Results for the Daniel and Martin problem, using damped Newton iterations

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	2	4	2.100e-04	1.665e+06	
MIRK-1:	2	6	2.100e-04	1.252e+06	24.8%
NEW-A:	2	8	2.100e-04	1.825e+05	89.0%
NEW-B:	2	8	2.100e-04	1.543e+05	90.7%
MIRK-S:	4	4	5.580e-06	3.311e+06	
MIRK-1:	4	5	5.580e-06	2.063e+06	37.7%
NEW-A:	4	6	5.580e-06	2.452e+05	92.6%
NEW-B:	4	6	5.580e-06	2.015e+05	93.9%
MIRK-S:	8	4	1.298e-07	6.603e+06	
MIRK-1:	8	5	1.298e-07	4.101e+06	37.9%
NEW-A:	8	5	1.298e-07	3.848e+05	94.2%
NEW-B:	8	5	1.298e-07	3.108e+05	95.3%
MIRK-S:	16	4	2.884e-09	1.319e+07	
MIRK-1:	16	4	2.867e-09	6.543e+06	50.4%
NEW-A:	16	5	2.884e-09	7.458e+05	94.3%
NEW-B:	16	5	2.884e-09	5.966e+05	95.5%
MIRK-S:	32	4	6.352e-11	2.636e+07	
MIRK-1:	32	4	6.255e-11	1.307e+07	50.4%
NEW-A:	32	4	6.313e-11	1.174e+06	95.5%
NEW-B:	32	4	6.313e-11	9.347e+05	96.5%
MIRK-S:	64	4	1.399e-12	5.269e+07	
MIRK-1:	64	4	1.324e-12	2.611e+07	50.4%
NEW-A:	64	4	1.368e-12	2.329e+06	95.6%
NEW-B:	64	4	1.368e-12	1.849e+06	96.5%

Table A.4: Results for the Daniel and Martin problem, repeated 10 times, using damped Newton iterations

A.2 Kreiss problem test results

A.2.1 Tables

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	2	** 6	5.980e+08	1.688e+04	
MIRK-1:	2	** 4	1.883e+06	7.932e+03	53.0%
NEW-A:	2	** 4	6.232e+22	4.168e+03	75.3%
NEW-B:	2	** 4	6.232e+22	4.448e+03	73.7%
MIRK-S:	4	7	5.663e-02	3.836e+04	
MIRK-1:	4	** 10	1.953e+17	3.815e+04	0.5%
NEW-A:	4	** 6	7.850e+08	1.169e+04	69.5%
NEW-B:	4	** 6	7.850e+08	1.240e+04	67.7%
MIRK-S:	8	6	1.013e-03	6.487e+04	
MIRK-1:	8	15	1.013e-03	1.122e+05	-72.9%
NEW-A:	8	* 41	5.631e-01	1.542e+05	-137.6%
NEW-B:	8	* 41	5.631e-01	1.629e+05	-151.2%
MIRK-S:	16	5	9.529e-06	1.074e+05	
MIRK-1:	16	8	9.529e-06	1.185e+05	-10.3%
NEW-A:	16	13	9.528e-06	9.599e+04	10.6%
NEW-B:	16	13	9.528e-06	1.013e+05	5.7%
MIRK-S:	32	5	1.962e-07	2.140e+05	
MIRK-1:	32	7	1.962e-07	2.062e+05	3.6%
NEW-A:	32	7	1.964e-07	1.024e+05	52.1%
NEW-B:	32	7	1.964e-07	1.080e+05	49.6%
MIRK-S:	64	5	4.213e-09	4.273e+05	
MIRK-1:	64	6	4.221e-09	3.526e+05	17.5%
NEW-A:	64	6	4.215e-09	1.748e+05	59.1%
NEW-B:	64	6	4.215e-09	1.841e+05	56.9%

Table A.5: Results for the Kreiss problem, with $\epsilon = 0.1$, using full Newton iterations. The symbol $*$ indicates the maximum allowed number of iterations was exceeded ($k > MAXITERS$). The symbol $**$ indicates that the iteration diverged ($n > MAXN$).

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	2	* 41	2.091e+05	1.154e+05	
MIRK-1:	2	** 2	3.429e+07	3.966e+03	96.6%
NEW-A:	2	** 2	4.533e+07	2.084e+03	98.2%
NEW-B:	2	** 2	4.533e+07	2.223e+03	98.1%
MIRK-S:	4	** 4	2.489e+07	2.192e+04	
MIRK-1:	4	** 6	2.784e+15	2.289e+04	-4.4%
NEW-A:	4	** 4	9.574e+21	7.792e+03	64.5%
NEW-B:	4	** 4	9.574e+21	8.264e+03	62.3%
MIRK-S:	8	** 14	1.418e+06	1.514e+05	
MIRK-1:	8	** 6	3.635e+07	4.487e+04	70.4%
NEW-A:	8	** 6	1.534e+10	2.256e+04	85.1%
NEW-B:	8	** 6	1.534e+10	2.384e+04	84.2%
MIRK-S:	16	6	5.154e-04	1.289e+05	
MIRK-1:	16	13	5.154e-04	1.925e+05	-49.4%
NEW-A:	16	* 41	3.254e+00	3.027e+05	-134.9%
NEW-B:	16	* 41	3.254e+00	3.194e+05	-147.9%
MIRK-S:	32	6	4.876e-06	2.568e+05	
MIRK-1:	32	9	4.876e-06	2.652e+05	-3.2%
NEW-A:	32	10	4.876e-06	1.463e+05	43.0%
NEW-B:	32	10	4.876e-06	1.542e+05	40.0%
MIRK-S:	64	6	9.922e-08	5.128e+05	
MIRK-1:	64	7	9.927e-08	4.114e+05	19.8%
NEW-A:	64	8	9.922e-08	2.330e+05	54.6%
NEW-B:	64	8	9.922e-08	2.455e+05	52.1%

Table A.6: Results for the Kreiss problem, with $\epsilon = 0.05$, using full Newton iterations. The symbol $*$ indicates the maximum allowed number of iterations was exceeded ($k > MAXITERS$). The symbol $**$ indicates that the iteration diverged ($n > MAXN$).

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	2	* 41	1.662e+03	1.154e+05	
MIRK-1:	2	** 2	3.774e+22	3.966e+03	96.6%
NEW-A:	2	** 2	6.704e+22	2.084e+03	98.2%
NEW-B:	2	** 2	6.704e+22	2.224e+03	98.1%
MIRK-S:	4	** 6	1.233e+08	3.288e+04	
MIRK-1:	4	** 2	1.215e+16	7.630e+03	76.8%
NEW-A:	4	** 2	1.352e+15	3.896e+03	88.2%
NEW-B:	4	** 2	1.352e+15	4.132e+03	87.4%
MIRK-S:	8	** 3	1.085e+07	3.244e+04	
MIRK-1:	8	** 3	2.267e+24	2.244e+04	30.8%
NEW-A:	8	** 3	7.136e+24	1.128e+04	65.2%
NEW-B:	8	** 3	7.136e+24	1.192e+04	63.2%
MIRK-S:	16	** 5	1.450e+06	1.074e+05	
MIRK-1:	16	** 4	6.761e+20	5.923e+04	44.8%
NEW-A:	16	** 3	2.694e+08	2.215e+04	79.4%
NEW-B:	16	** 3	2.694e+08	2.337e+04	78.2%
MIRK-S:	32	** 5	1.614e+06	2.140e+05	
MIRK-1:	32	** 6	1.543e+10	1.768e+05	17.4%
NEW-A:	32	** 5	5.630e+16	7.316e+04	65.8%
NEW-B:	32	** 5	5.630e+16	7.711e+04	64.0%
MIRK-S:	64	** 5	2.386e+08	4.273e+05	
MIRK-1:	64	** 5	1.431e+07	2.939e+05	31.2%
NEW-A:	64	** 28	9.051e+09	8.156e+05	-90.9%
NEW-B:	64	** 28	9.051e+09	8.592e+05	-101.1%

Table A.7: Results for the Kreiss problem, with $\epsilon = 0.01$, using full Newton iterations. The symbol $*$ indicates the maximum allowed number of iterations was exceeded ($k > MAXITERS$). The symbol $**$ indicates that the iteration diverged ($n > MAXN$).

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	2	*** 16	3.480e-01	5.815e+04	
MIRK-1:	2	*** 3	7.817e-01	9.865e+03	83.0%
NEW-A:	2	* 41	4.172e-01	7.681e+04	-32.1%
NEW-B:	2	* 41	4.172e-01	7.974e+04	-37.1%
MIRK-S:	4	7	5.663e-02	4.826e+04	
MIRK-1:	4	24	5.663e-02	1.274e+05	-164.0%
NEW-A:	4	* 41	5.663e-02	1.453e+05	-201.1%
NEW-B:	4	* 41	5.663e-02	1.495e+05	-209.7%
MIRK-S:	8	6	1.013e-03	7.930e+04	
MIRK-1:	8	15	1.013e-03	1.483e+05	-87.0%
NEW-A:	8	20	1.013e-03	1.327e+05	-67.4%
NEW-B:	8	20	1.013e-03	1.360e+05	-71.5%
MIRK-S:	16	5	9.529e-06	1.310e+05	
MIRK-1:	16	8	9.529e-06	1.563e+05	-19.3%
NEW-A:	16	13	9.528e-06	1.575e+05	-20.2%
NEW-B:	16	13	9.528e-06	1.615e+05	-23.3%
MIRK-S:	32	5	1.962e-07	2.610e+05	
MIRK-1:	32	7	1.962e-07	2.720e+05	-4.2%
NEW-A:	32	7	1.964e-07	1.681e+05	35.6%
NEW-B:	32	7	1.964e-07	1.722e+05	34.0%
MIRK-S:	64	5	4.213e-09	5.208e+05	
MIRK-1:	64	6	4.221e-09	4.649e+05	10.7%
NEW-A:	64	6	4.215e-09	2.870e+05	44.9%
NEW-B:	64	6	4.215e-09	2.937e+05	43.6%

Table A.8: Results for the Kreiss problem, with $\epsilon = 0.1$, using damped Newton iterations. The symbol $*$ indicates the maximum allowed number of iterations was exceeded ($k > MAXITERS$). The symbol $***$ indicates the damped Newton iteration failed with the damping factor becoming less than the allowed minimum ($\lambda < \lambda_{min}$).

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	2	*** 2	9.511e-01	8.241e+03	
MIRK-1:	2	*** 1	1.010e+00	3.938e+03	52.2%
NEW-A:	2	*** 1	1.050e+00	2.997e+03	63.6%
NEW-B:	2	*** 1	1.050e+00	3.066e+03	62.8%
MIRK-S:	4	21	6.840e-01	1.436e+05	
MIRK-1:	4	*** 1	3.160e+00	7.492e+03	94.8%
NEW-A:	4	* 41	6.925e-01	1.441e+05	-0.4%
NEW-B:	4	* 41	6.925e-01	1.483e+05	-3.3%
MIRK-S:	8	8	4.296e-02	1.081e+05	
MIRK-1:	8	28	4.296e-02	2.909e+05	-169.1%
NEW-A:	8	* 41	4.296e-02	2.786e+05	-157.8%
NEW-B:	8	* 41	4.296e-02	2.853e+05	-163.9%
MIRK-S:	16	6	5.154e-04	1.573e+05	
MIRK-1:	16	13	5.154e-04	2.540e+05	-61.5%
NEW-A:	16	28	5.154e-04	3.624e+05	-130.5%
NEW-B:	16	28	5.154e-04	3.706e+05	-135.7%
MIRK-S:	32	6	4.876e-06	3.132e+05	
MIRK-1:	32	9	4.876e-06	3.497e+05	-11.7%
NEW-A:	32	10	4.876e-06	2.402e+05	23.3%
NEW-B:	32	10	4.876e-06	2.460e+05	21.4%
MIRK-S:	64	6	9.922e-08	6.250e+05	
MIRK-1:	64	7	9.927e-08	5.423e+05	13.2%
NEW-A:	64	8	9.922e-08	3.826e+05	38.8%
NEW-B:	64	8	9.922e-08	3.916e+05	37.3%

Table A.9: Kreiss problem with $\epsilon = 0.05$, using damped Newton iterations. The symbol $*$ indicates the maximum allowed number of iterations was exceeded ($k > MAXITERS$). The symbol $***$ indicates the damped Newton iteration failed with the damping factor becoming less than the allowed minimum ($\lambda < \lambda_{min}$).

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	2	*** 3	2.536e+00	1.171e+04	
MIRK-1:	2	*** 1	1.951e+02	3.938e+03	66.4%
NEW-A:	2	*** 1	9.913e+01	2.997e+03	74.4%
NEW-B:	2	*** 1	9.913e+01	3.070e+03	73.8%
MIRK-S:	4	*** 10	1.248e+01	7.207e+04	
MIRK-1:	4	*** 1	7.272e+01	7.492e+03	89.6%
NEW-A:	4	*** 1	3.312e+01	5.625e+03	92.2%
NEW-B:	4	*** 1	3.312e+01	5.704e+03	92.1%
MIRK-S:	8	*** 2	1.918e+01	3.115e+04	
MIRK-1:	8	*** 2	1.363e+01	2.448e+04	21.4%
NEW-A:	8	*** 6	1.363e+01	4.406e+04	-41.4%
NEW-B:	8	*** 6	1.363e+01	4.497e+04	-44.4%
MIRK-S:	16	*** 3	1.661e+01	8.790e+04	
MIRK-1:	16	*** 1	1.347e+01	3.345e+04	61.9%
NEW-A:	16	*** 11	1.278e+01	1.565e+05	-78.0%
NEW-B:	16	*** 11	1.278e+01	1.594e+05	-81.3%
MIRK-S:	32	13	2.241e-01	6.969e+05	
MIRK-1:	32	*** 20	3.967e-01	8.414e+05	-20.7%
NEW-A:	32	*** 9	1.136e+01	2.622e+05	62.4%
NEW-B:	32	*** 9	1.136e+01	2.663e+05	61.8%
MIRK-S:	64	14	2.349e-03	1.495e+06	
MIRK-1:	64	34	2.349e-03	2.671e+06	-78.7%
NEW-A:	64	37	2.349e-03	1.861e+06	-24.5%
NEW-B:	64	37	2.349e-03	1.901e+06	-27.1%

Table A.10: Results for the Kreiss problem, with $\epsilon = 0.01$, using damped Newton iterations. The symbol $***$ indicates the damped Newton iteration failed with the damping factor becoming less than the allowed minimum ($\lambda < \lambda_{min}$).

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	476	** 4	3.920e+21	2.539e+06	
MIRK-1:	476	11	3.214e-12	4.798e+06	-89.0%
NEW-A:	476	** 4	2.792e+07	8.631e+05	66.0%
NEW-B:	476	** 4	2.792e+07	9.088e+05	64.2%

Table A.11: Results for the Kreiss problem, with $\epsilon = 0.005$, solved over a fine mesh from MIRKDC, using full Newton iterations. The symbol $**$ indicates that the iteration diverged ($n > MAXN$).

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	476	13	3.216e-12	1.032e+07	
MIRK-1:	476	14	3.932e-10	8.454e+06	18.1%
NEW-A:	476	*** 31	1.319e-01	1.234e+07	-19.6%
NEW-B:	476	*** 31	1.319e-01	1.256e+07	-21.7%

Table A.12: Results for the Kreiss problem, with $\epsilon = 0.005$, solved over a fine mesh from MIRKDC, using damped Newton iterations. The symbol $***$ indicates the damped Newton iteration failed with the damping factor becoming less than the allowed minimum ($\lambda < \lambda_{min}$).

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	2	3	1.802e-03	8.442e+03	
MIRK-1:	2	** 3	3.371e+26	5.949e+03	29.5%
NEW-A:	2	** 2	1.091e+06	2.084e+03	75.3%
NEW-B:	2	** 2	1.091e+06	2.224e+03	73.7%
MIRK-S:	4	** 7	1.158e+07	3.836e+04	
MIRK-1:	4	** 2	5.468e+17	7.620e+03	80.1%
NEW-A:	4	** 2	3.389e+21	3.886e+03	89.9%
NEW-B:	4	** 2	3.389e+21	4.132e+03	89.2%
MIRK-S:	8	** 6	1.034e+07	6.486e+04	
MIRK-1:	8	** 2	3.499e+08	1.493e+04	77.0%
NEW-A:	8	** 2	9.303e+14	7.490e+03	88.5%
NEW-B:	8	** 2	9.303e+14	7.948e+03	87.7%
MIRK-S:	16	8	1.292e+01	1.718e+05	
MIRK-1:	16	** 3	1.205e+19	4.439e+04	74.2%
NEW-A:	16	** 2	6.611e+12	1.471e+04	91.4%
NEW-B:	16	** 2	6.611e+12	1.558e+04	90.9%
MIRK-S:	32	4	5.517e+00	1.712e+05	
MIRK-1:	32	** 8	8.836e+18	2.357e+05	-37.7%
NEW-A:	32	** 4	1.725e+09	5.846e+04	65.9%
NEW-B:	32	** 4	1.725e+09	6.169e+04	64.0%
MIRK-S:	64	4	3.524e-01	3.416e+05	
MIRK-1:	64	** 12	1.690e+06	7.046e+05	-106.2%
NEW-A:	64	** 10	1.690e+14	2.913e+05	14.7%
NEW-B:	64	** 10	1.690e+14	3.069e+05	10.2%

Table A.13: Results for the Kreiss problem, with $\epsilon = 0.005$, using full Newton iterations, with a good initial guess. The symbol $**$ indicates that the iteration diverged ($n > MAXN$).

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	2	3	1.802e-03	1.042e+04	
MIRK-1:	2	*** 1	1.678e-02	3.938e+03	62.2%
NEW-A:	2	*** 1	1.713e-02	2.997e+03	71.2%
NEW-B:	2	*** 1	1.713e-02	3.070e+03	70.5%
MIRK-S:	4	*** 9	1.670e+01	6.413e+04	
MIRK-1:	4	*** 1	5.569e+01	7.470e+03	88.4%
NEW-A:	4	*** 1	1.225e+03	5.603e+03	91.3%
NEW-B:	4	*** 1	1.225e+03	5.704e+03	91.1%
MIRK-S:	8	*** 4	2.079e+01	5.756e+04	
MIRK-1:	8	*** 2	1.389e+01	2.442e+04	57.6%
NEW-A:	8	*** 1	8.692e+01	1.082e+04	81.2%
NEW-B:	8	*** 1	8.692e+01	1.097e+04	80.9%
MIRK-S:	16	8	1.292e+01	2.143e+05	
MIRK-1:	16	*** 11	1.818e+01	2.334e+05	-8.9%
NEW-A:	16	*** 1	2.516e+01	2.126e+04	90.1%
NEW-B:	16	*** 1	2.516e+01	2.151e+04	90.0%
MIRK-S:	32	4	5.517e+00	2.088e+05	
MIRK-1:	32	*** 11	5.494e+00	4.642e+05	-122.3%
NEW-A:	32	*** 19	5.205e+00	5.114e+05	-145.0%
NEW-B:	32	*** 19	5.205e+00	5.214e+05	-149.7%
MIRK-S:	64	4	3.524e-01	4.163e+05	
MIRK-1:	64	*** 7	3.524e-01	5.968e+05	-43.3%
NEW-A:	64	* 41	3.524e-01	2.162e+06	-419.4%
NEW-B:	64	* 41	3.524e-01	2.204e+06	-429.3%

Table A.14: Results from the Kreiss problem, with $\epsilon = 0.005$, using damped Newton iterations, with a good initial guess. The symbol $*$ indicates the maximum allowed number of iterations was exceeded ($k > MAXITERS$). The symbol $***$ indicates the damped Newton iteration failed with the damping factor becoming less than the allowed minimum ($\lambda < \lambda_{min}$).

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	476	4	3.204e-12	2.538e+06	
MIRK-1:	476	4	5.246e-12	1.745e+06	31.3%
NEW-A:	476	** 15	2.350e+09	3.236e+06	-27.5%
NEW-B:	476	** 15	2.350e+09	3.408e+06	-34.3%

Table A.15: Results for the Kreiss problem, with $\epsilon = 0.005$, solved over a fine mesh from MIRKDC, using full Newton iterations, and a good initial guess. The symbol $**$ indicates that the iteration diverged ($n > MAXN$).

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	476	4	3.204e-12	3.093e+06	
MIRK-1:	476	4	5.246e-12	2.299e+06	25.7%
NEW-A:	476	6	8.777e-08	2.398e+06	22.5%
NEW-B:	476	6	8.777e-08	2.440e+06	21.1%

Table A.16: Results from the Kreiss problem, with $\epsilon = 0.005$, solved over a fine mesh from MIRKDC, using damped Newton iterations, and a good initial guess.

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	2	*** 16	1.101e+00	6.684e+06	
MIRK-1:	2	*** 3	2.472e+00	6.465e+05	90.3%
NEW-A:	2	* 41	1.319e+00	1.006e+06	85.0%
NEW-B:	2	* 41	1.319e+00	8.603e+05	87.1%
MIRK-S:	4	7	1.791e-01	5.804e+06	
MIRK-1:	4	25	1.791e-01	1.037e+07	-78.6%
NEW-A:	4	* 41	1.791e-01	1.823e+06	68.6%
NEW-B:	4	* 41	1.791e-01	1.507e+06	74.0%
MIRK-S:	8	6	3.203e-03	9.902e+06	
MIRK-1:	8	16	3.203e-03	1.312e+07	-32.5%
NEW-A:	8	23	3.203e-03	1.855e+06	81.3%
NEW-B:	8	23	3.203e-03	1.509e+06	84.8%
MIRK-S:	16	5	3.013e-05	1.648e+07	
MIRK-1:	16	9	3.013e-05	1.472e+07	10.7%
NEW-A:	16	14	3.013e-05	2.081e+06	87.4%
NEW-B:	16	14	3.013e-05	1.668e+06	89.9%
MIRK-S:	32	5	6.206e-07	3.294e+07	
MIRK-1:	32	7	6.205e-07	2.286e+07	30.6%
NEW-A:	32	8	6.206e-07	2.343e+06	92.9%
NEW-B:	32	8	6.206e-07	1.867e+06	94.3%
MIRK-S:	64	5	1.332e-08	6.586e+07	
MIRK-1:	64	6	1.335e-08	3.917e+07	40.5%
NEW-A:	64	7	1.332e-08	4.069e+06	93.8%
NEW-B:	64	7	1.332e-08	3.231e+06	95.1%

Table A.17: Results for the Kreiss problem repeated 10 times, with $\epsilon = 0.1$, using damped Newton iterations. The symbol $*$ indicates the maximum allowed number of iterations was exceeded ($k > MAXITERS$). The symbol $***$ indicates the damped Newton iteration failed with the damping factor becoming less than the allowed minimum ($\lambda < \lambda_{min}$).

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	2	*** 2	3.008e+00	8.462e+05	
MIRK-1:	2	*** 1	3.193e+00	2.231e+05	73.6%
NEW-A:	2	*** 1	3.321e+00	3.723e+04	95.6%
NEW-B:	2	*** 1	3.321e+00	3.200e+04	96.2%
MIRK-S:	4	21	2.163e+00	1.740e+07	
MIRK-1:	4	*** 1	9.992e+00	4.376e+05	97.5%
NEW-A:	4	* 41	2.190e+00	1.813e+06	89.6%
NEW-B:	4	* 41	2.190e+00	1.496e+06	91.4%
MIRK-S:	8	8	1.359e-01	1.323e+07	
MIRK-1:	8	29	1.359e-01	2.392e+07	-80.8%
NEW-A:	8	* 41	1.359e-01	3.399e+06	74.3%
NEW-B:	8	* 41	1.359e-01	2.768e+06	79.1%
MIRK-S:	16	6	1.630e-03	1.978e+07	
MIRK-1:	16	14	1.630e-03	2.289e+07	-15.8%
NEW-A:	16	29	1.630e-03	4.586e+06	76.8%
NEW-B:	16	29	1.630e-03	3.688e+06	81.4%
MIRK-S:	32	6	1.542e-05	3.953e+07	
MIRK-1:	32	9	1.542e-05	2.939e+07	25.6%
NEW-A:	32	11	1.542e-05	3.222e+06	91.8%
NEW-B:	32	11	1.542e-05	2.566e+06	93.5%
MIRK-S:	64	6	3.138e-07	7.903e+07	
MIRK-1:	64	8	3.138e-07	5.222e+07	33.9%
NEW-A:	64	8	3.138e-07	4.651e+06	94.1%
NEW-B:	64	8	3.138e-07	3.693e+06	95.3%

Table A.18: Results for the Kreiss problem repeated 10 times, with $\epsilon = 0.05$, using damped Newton iterations. The symbol $*$ indicates the maximum allowed number of iterations was exceeded ($k > MAXITERS$). The symbol $***$ indicates the damped Newton iteration failed with the damping factor becoming less than the allowed minimum ($\lambda < \lambda_{min}$).

A.2.2 Condition and Spectral Radius

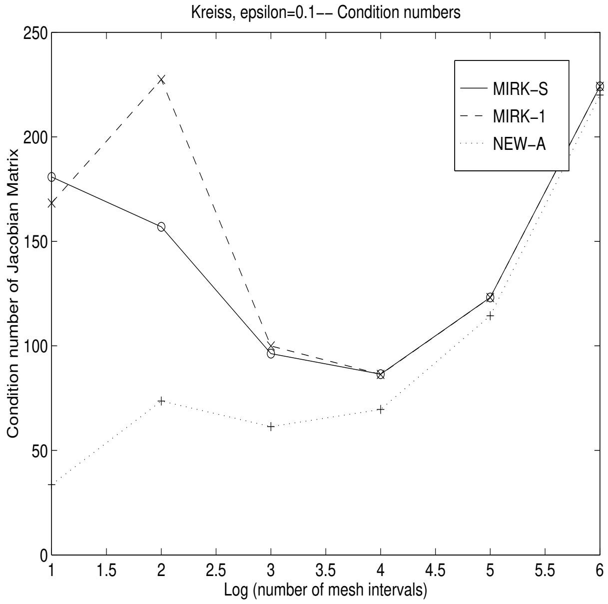

Figure A.1: Condition numbers of the Jacobian matrices, for the Kreiss problem with $\epsilon = 0.1$.

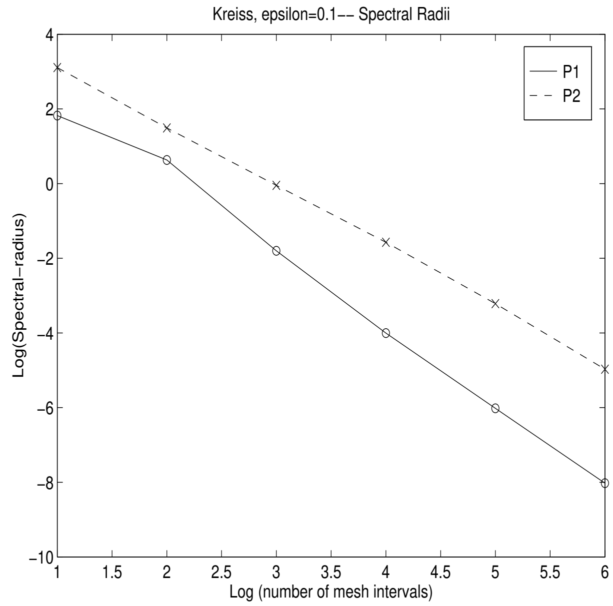

Figure A.2: Kreiss problem with $\epsilon = 0.1$. P1=spectral radius of G' calculated using MIRK-1 Jacobian. P2=spectral radius of G' calculated using approximate Jacobian.

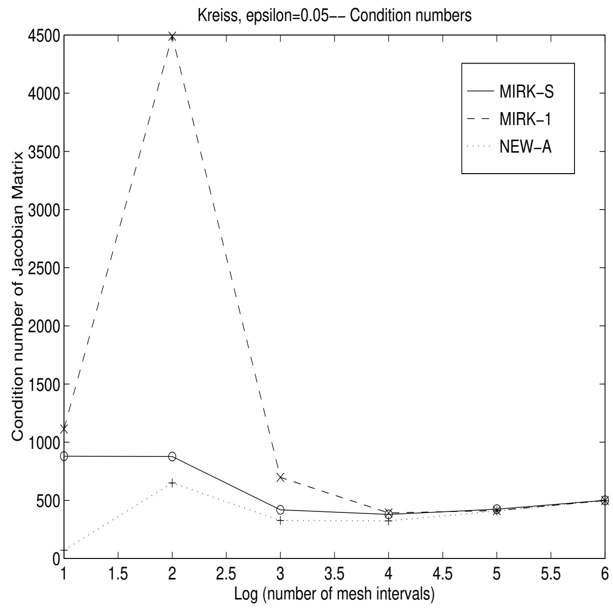

Figure A.3: Condition numbers of the Jacobian matrices, for the Kreiss problem with $\epsilon = 0.05$.

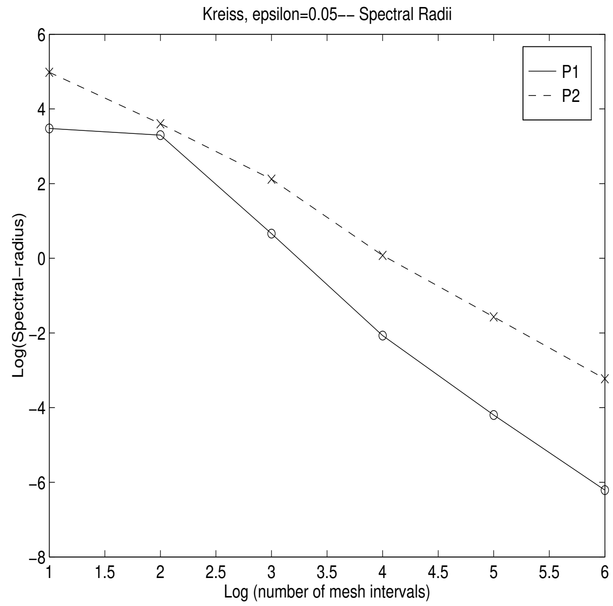

Figure A.4: Kreiss problem with $\epsilon = 0.05$. P1=spectral radius of G' calculated using MIRK-1 Jacobian. P2=spectral radius of G' calculated using approximate Jacobian.

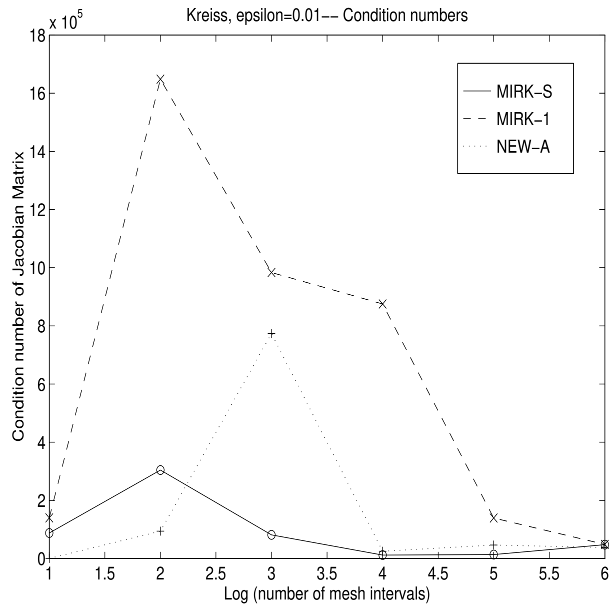

Figure A.5: Condition numbers of the Jacobian matrices, for the Kreiss problem with $\epsilon = 0.01$.

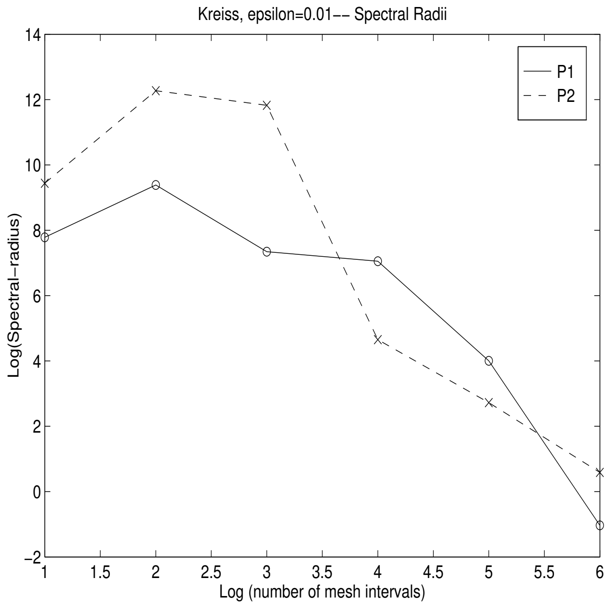

Figure A.6: Kreiss problem with $\epsilon = 0.01$.P1=spectral radius of G' calculated using MIRK-1 Jacobian. P2=spectral radius of G' calculated using approximate Jacobian.

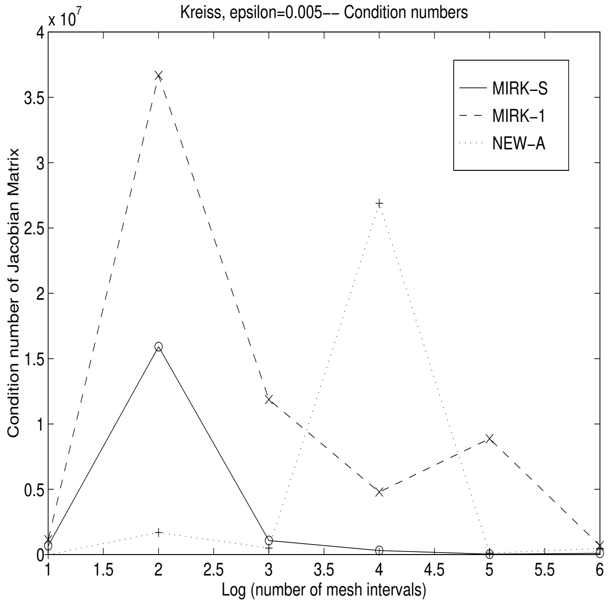

Figure A.7: Condition numbers of the Jacobian matrices, for the Kreiss problem with $\epsilon = 0.005$.

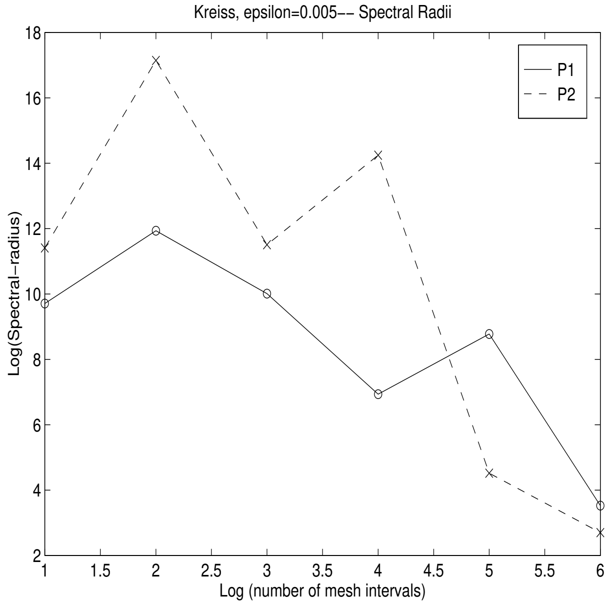

Figure A.8: Kreiss problem with $\epsilon = 0.005$. P1=spectral radius of G' calculated using MIRK-1 Jacobian. P2=spectral radius of G' calculated using approximate Jacobian.

A.2.3 Error in Newton steps

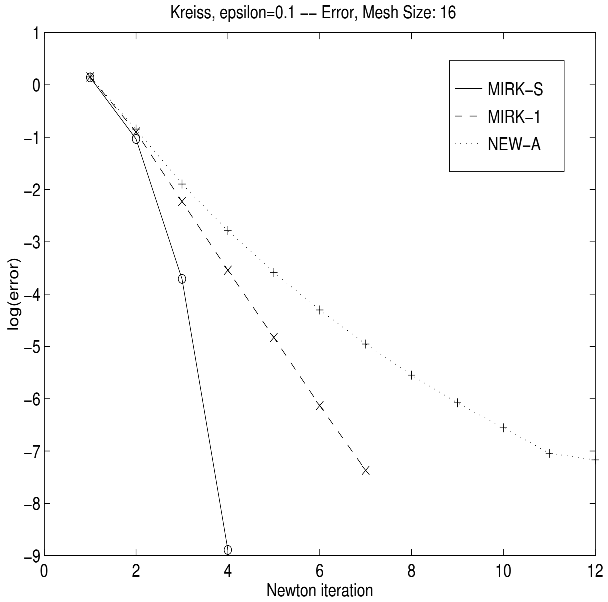

Figure A.9: Error in solution to the nonlinear system, for the Kreiss problem with $\epsilon = 0.1$, and 16 mesh intervals, using full Newton iterations.

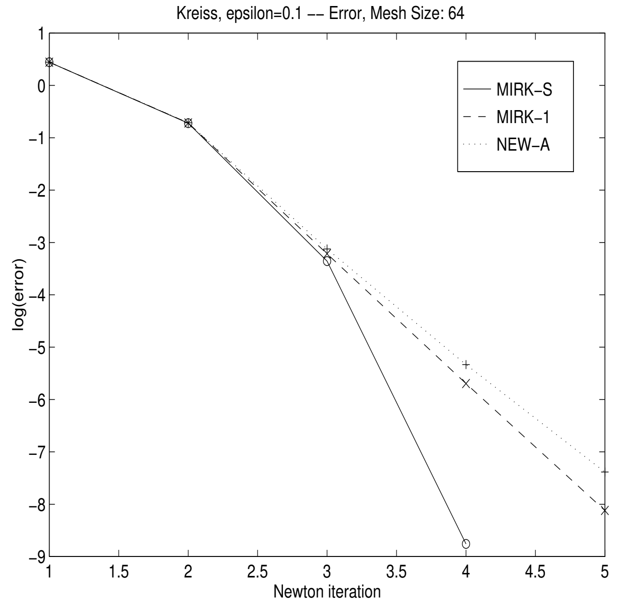

Figure A.10: Error in solution to the nonlinear system, for the Kreiss problem with $\epsilon = 0.1$, and 64 mesh intervals, using full Newton iterations.

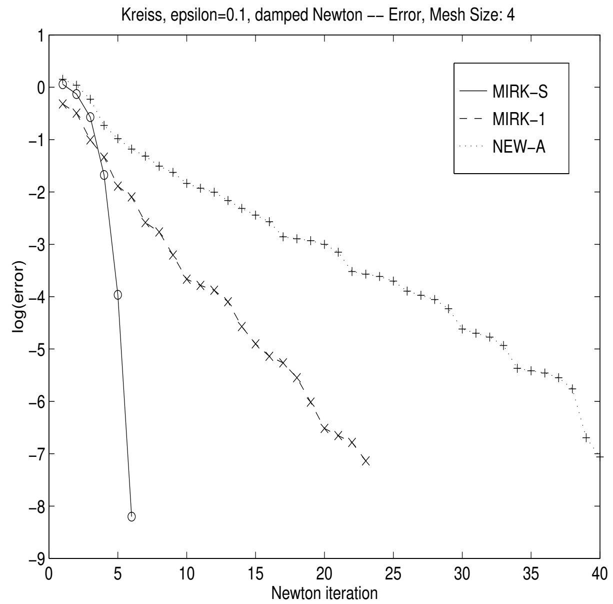

Figure A.11: Error in solution to the nonlinear system, for the Kreiss problem with $\epsilon = 0.1$, and 4 mesh intervals, using damped Newton iterations.

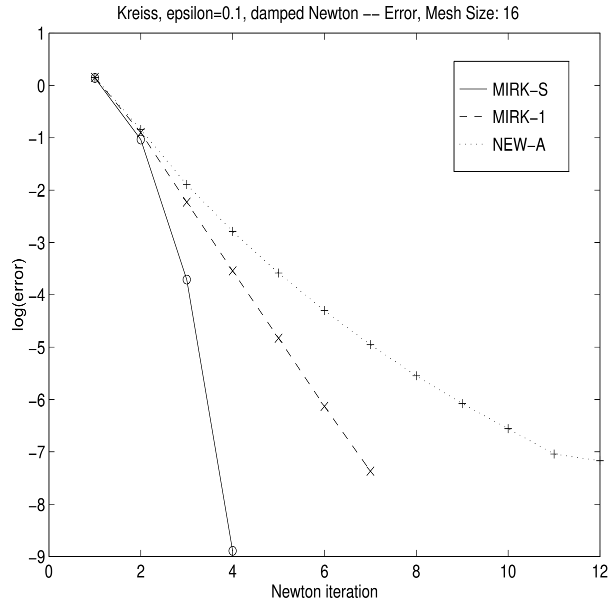

Figure A.12: Error in solution to the nonlinear system, for the Kreiss problem with $\epsilon = 0.1$, and 16 mesh intervals, using damped Newton iterations.

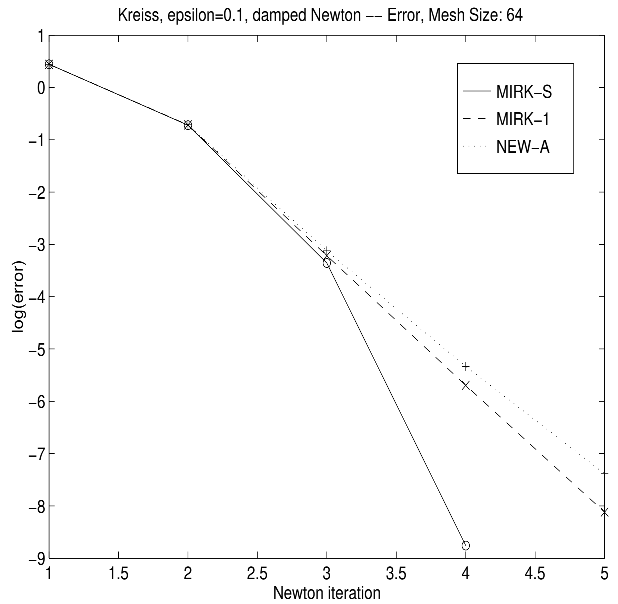

Figure A.13: Error in solution to the nonlinear system, for the Kreiss problem with $\epsilon = 0.1$, and 64 mesh intervals, using damped Newton iterations.

A.3 Shock Wave problem test results

A.3.1 Tables

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	2	6	7.299e-01	2.186e+04	
MIRK-1:	2	** 13	1.213e+06	2.973e+04	-36.0%
NEW-A:	2	** 5	1.064e+06	6.730e+03	69.2%
NEW-B:	2	** 5	1.064e+06	7.080e+03	67.6%
MIRK-S:	4	7	6.141e-02	4.998e+04	
MIRK-1:	4	* 41	9.759e+03	1.813e+05	-262.8%
NEW-A:	4	* 41	6.712e-01	1.048e+05	-109.7%
NEW-B:	4	* 41	6.712e-01	1.096e+05	-119.4%
MIRK-S:	8	7	3.665e-03	9.892e+04	
MIRK-1:	8	19	3.665e-03	1.652e+05	-67.0%
NEW-A:	8	23	3.665e-03	1.144e+05	-15.7%
NEW-B:	8	23	3.665e-03	1.194e+05	-20.7%
MIRK-S:	16	6	2.910e-05	1.687e+05	
MIRK-1:	16	9	2.910e-05	1.552e+05	8.0%
NEW-A:	16	11	2.910e-05	1.080e+05	36.0%
NEW-B:	16	11	2.910e-05	1.124e+05	33.3%
MIRK-S:	32	6	5.051e-07	3.365e+05	
MIRK-1:	32	8	5.049e-07	2.746e+05	18.4%
NEW-A:	32	9	5.052e-07	1.755e+05	47.9%
NEW-B:	32	9	5.052e-07	1.826e+05	45.7%
MIRK-S:	64	6	8.742e-08	6.721e+05	
MIRK-1:	64	7	8.730e-08	4.795e+05	28.7%
NEW-A:	64	7	8.746e-08	2.720e+05	59.5%
NEW-B:	64	7	8.746e-08	2.829e+05	57.9%

Table A.19: Results for the Shock Wave problem, with $\epsilon = 0.1$, using full Newton iterations. The symbol $*$ indicates the maximum allowed number of iterations was exceeded ($k > MAXITERS$). The symbol $**$ indicates that the iteration diverged ($n > MAXN$).

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	2	* 41	4.118e+01	1.494e+05	
MIRK-1:	2	** 6	1.145e+06	1.372e+04	90.8%
NEW-A:	2	** 4	3.133e+06	5.384e+03	96.4%
NEW-B:	2	** 4	3.133e+06	5.664e+03	96.2%
MIRK-S:	4	8	1.322e+00	5.712e+04	
MIRK-1:	4	** 10	6.786e+05	4.423e+04	22.6%
NEW-A:	4	** 9	1.447e+06	2.300e+04	59.7%
NEW-B:	4	** 9	1.447e+06	2.407e+04	57.9%
MIRK-S:	8	9	1.160e-01	1.272e+05	
MIRK-1:	8	** 21	5.515e+07	1.826e+05	-43.6%
NEW-A:	8	** 27	6.198e+05	1.344e+05	-5.6%
NEW-B:	8	** 27	6.198e+05	1.401e+05	-10.2%
MIRK-S:	16	* 41	8.741e+02	1.153e+06	
MIRK-1:	16	** 11	2.477e+06	1.896e+05	83.5%
NEW-A:	16	** 26	7.701e+06	2.552e+05	77.9%
NEW-B:	16	** 38	4.543e+06	3.884e+05	66.3%
MIRK-S:	32	8	4.596e-05	4.487e+05	
MIRK-1:	32	13	4.596e-05	4.463e+05	0.5%
NEW-A:	32	15	4.596e-05	2.924e+05	34.8%
NEW-B:	32	15	4.596e-05	3.043e+05	32.2%
MIRK-S:	64	8	1.007e-06	8.962e+05	
MIRK-1:	64	10	1.006e-06	6.850e+05	23.6%
NEW-A:	64	10	1.002e-06	3.886e+05	56.6%
NEW-B:	64	10	1.002e-06	4.041e+05	54.9%

Table A.20: Results for the Shock Wave problem, with $\epsilon = 0.05$, using full Newton iterations. The symbol $*$ indicates the maximum allowed number of iterations was exceeded ($k > MAXITERS$). The symbol $**$ indicates that the iteration diverged ($n > MAXN$).

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	2	10	1.931e+00	3.644e+04	
MIRK-1:	2	** 3	1.945e+06	6.861e+03	81.2%
NEW-A:	2	** 3	2.160e+07	4.038e+03	88.9%
NEW-B:	2	** 3	2.160e+07	4.248e+03	88.3%
MIRK-S:	4	* 41	5.008e+02	2.927e+05	
MIRK-1:	4	** 4	2.132e+06	1.769e+04	94.0%
NEW-A:	4	** 3	1.164e+07	7.668e+03	97.4%
NEW-B:	4	** 3	1.164e+07	8.022e+03	97.3%
MIRK-S:	8	15	1.481e+01	2.120e+05	
MIRK-1:	8	** 4	1.636e+06	3.478e+04	83.6%
NEW-A:	8	** 4	1.598e+06	1.990e+04	90.6%
NEW-B:	8	** 4	1.598e+06	2.076e+04	90.2%
MIRK-S:	16	** 30	2.562e+06	8.435e+05	
MIRK-1:	16	** 3	6.514e+06	5.172e+04	93.9%
NEW-A:	16	** 5	2.605e+06	4.908e+04	94.2%
NEW-B:	16	** 5	2.605e+06	5.111e+04	93.9%
MIRK-S:	32	* 41	7.250e+02	2.299e+06	
MIRK-1:	32	** 9	4.144e+06	3.089e+05	86.6%
NEW-A:	32	** 16	4.841e+07	3.119e+05	86.4%
NEW-B:	32	** 16	6.436e+05	3.246e+05	85.9%
MIRK-S:	64	* 41	3.417e+03	4.593e+06	
MIRK-1:	64	** 13	1.205e+06	8.905e+05	80.6%
NEW-A:	64	** 19	4.650e+06	7.383e+05	83.9%
NEW-B:	64	** 19	4.652e+06	7.679e+05	83.3%

Table A.21: Results for the Shock Wave problem with $\epsilon = 0.01$, using full Newton iterations. The symbol $*$ indicates the maximum allowed number of iterations was exceeded ($k > MAXITERS$). The symbol $**$ indicates that the iteration diverged ($n > MAXN$).

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	2	6	7.299e-01	2.720e+04	
MIRK-1:	2	* 41	1.091e+00	1.398e+05	-414.1%
NEW-A:	2	*** 12	1.035e+00	3.294e+04	-21.1%
NEW-B:	2	*** 12	1.035e+00	3.380e+04	-24.3%
MIRK-S:	4	7	6.141e-02	6.356e+04	
MIRK-1:	4	* 41	6.141e-02	2.678e+05	-321.4%
NEW-A:	4	* 41	6.141e-02	1.879e+05	-195.7%
NEW-B:	4	* 41	6.141e-02	1.921e+05	-202.3%
MIRK-S:	8	7	3.665e-03	1.222e+05	
MIRK-1:	8	19	3.665e-03	2.284e+05	-86.9%
NEW-A:	8	23	3.665e-03	1.909e+05	-56.2%
NEW-B:	8	23	3.665e-03	1.949e+05	-59.5%
MIRK-S:	16	6	2.910e-05	2.081e+05	
MIRK-1:	16	9	2.910e-05	2.143e+05	-3.0%
NEW-A:	16	11	2.910e-05	1.803e+05	13.4%
NEW-B:	16	11	2.910e-05	1.837e+05	11.8%
MIRK-S:	32	6	5.051e-07	4.149e+05	
MIRK-1:	32	8	5.049e-07	3.792e+05	8.6%
NEW-A:	32	9	5.052e-07	2.931e+05	29.4%
NEW-B:	32	9	5.052e-07	2.983e+05	28.1%
MIRK-S:	64	6	8.742e-08	8.285e+05	
MIRK-1:	64	7	8.730e-08	6.619e+05	20.1%
NEW-A:	64	7	8.746e-08	4.544e+05	45.2%
NEW-B:	64	7	8.746e-08	4.623e+05	44.2%

Table A.22: Results for the Shock Wave problem, with $\epsilon = 0.1$, using damped Newton iterations. The symbol $*$ indicates the maximum allowed number of iterations was exceeded ($k > MAXITERS$). The symbol $***$ indicates the damped Newton iteration failed with the damping factor becoming less than the allowed minimum ($\lambda < \lambda_{min}$).

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	2	12	1.814e+00	5.614e+04	
MIRK-1:	2	*** 2	1.547e+00	8.107e+03	85.6%
NEW-A:	2	*** 2	1.363e+00	6.225e+03	88.9%
NEW-B:	2	*** 2	1.363e+00	6.369e+03	88.7%
MIRK-S:	4	*** 7	6.455e-01	6.859e+04	
MIRK-1:	4	*** 6	9.153e-01	4.010e+04	41.5%
NEW-A:	4	*** 3	2.406e+00	1.613e+04	76.5%
NEW-B:	4	*** 3	2.406e+00	1.642e+04	76.1%
MIRK-S:	8	13	1.160e-01	2.335e+05	
MIRK-1:	8	*** 14	4.480e-02	1.945e+05	16.7%
NEW-A:	8	*** 18	6.163e-01	1.789e+05	23.4%
NEW-B:	8	*** 18	6.163e-01	1.816e+05	22.2%
MIRK-S:	16	11	9.907e-03	3.945e+05	
MIRK-1:	16	* 41	9.907e-03	9.892e+05	-150.7%
NEW-A:	16	*** 15	5.286e-02	2.847e+05	27.8%
NEW-B:	16	*** 15	5.286e-02	2.887e+05	26.8%
MIRK-S:	32	10	4.596e-05	7.044e+05	
MIRK-1:	32	15	4.596e-05	7.238e+05	-2.8%
NEW-A:	32	17	4.596e-05	5.665e+05	19.6%
NEW-B:	32	17	4.596e-05	5.761e+05	18.2%
MIRK-S:	64	10	1.007e-06	1.406e+06	
MIRK-1:	64	12	1.006e-06	1.160e+06	17.5%
NEW-A:	64	12	1.006e-06	8.047e+05	42.8%
NEW-B:	64	12	1.006e-06	8.177e+05	41.9%

Table A.23: Results for the Shock Wave problem, with $\epsilon = 0.05$, using damped Newton iterations. The symbol $*$ indicates the maximum allowed number of iterations was exceeded ($k > MAXITERS$). The symbol $***$ indicates the damped Newton iteration failed with the damping factor becoming less than the allowed minimum ($\lambda < \lambda_{min}$).

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	2	13	1.931e+00	5.980e+04	
MIRK-1:	2	*** 1	3.996e+00	4.932e+03	91.8%
NEW-A:	2	*** 1	1.091e+01	3.991e+03	93.3%
NEW-B:	2	*** 1	1.091e+01	4.064e+03	93.2%
MIRK-S:	4	*** 6	2.028e+00	5.807e+04	
MIRK-1:	4	*** 1	1.700e+01	9.480e+03	83.7%
NEW-A:	4	*** 1	1.276e+01	7.613e+03	86.9%
NEW-B:	4	*** 1	1.276e+01	7.692e+03	86.8%
MIRK-S:	8	15	1.827e+01	2.684e+05	
MIRK-1:	8	*** 5	3.307e+01	6.666e+04	75.2%
NEW-A:	8	*** 3	1.820e+01	3.146e+04	88.3%
NEW-B:	8	*** 3	1.820e+01	3.190e+04	88.1%
MIRK-S:	16	24	2.314e+01	8.649e+05	
MIRK-1:	16	*** 3	1.904e+01	8.439e+04	90.2%
NEW-A:	16	*** 5	1.973e+01	9.490e+04	89.0%
NEW-B:	16	*** 5	1.973e+01	9.625e+04	88.9%
MIRK-S:	32	*** 4	3.012e+01	3.024e+05	
MIRK-1:	32	*** 4	2.695e+01	2.282e+05	24.5%
NEW-A:	32	*** 5	3.209e+01	2.143e+05	29.1%
NEW-B:	32	*** 5	3.209e+01	2.164e+05	28.4%
MIRK-S:	64	*** 8	6.129e+01	1.182e+06	
MIRK-1:	64	*** 9	8.029e+01	9.281e+05	21.5%
NEW-A:	64	*** 14	4.699e+01	1.012e+06	14.4%
NEW-B:	64	*** 14	4.699e+01	1.026e+06	13.2%

Table A.24: Results for the Shock Wave problem, with $\epsilon = 0.01$, using damped Newton iterations. The symbol $***$ indicates the damped Newton iteration failed with the damping factor becoming less than the allowed minimum ($\lambda < \lambda_{min}$).

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	796	* 41	1.548e+05	5.705e+07	
MIRK-1:	796	22	2.613e-05	1.871e+07	67.2%
NEW-A:	796	24	2.613e-05	1.156e+07	79.7%
NEW-B:	796	24	2.613e-05	1.202e+07	78.9%

Table A.25: Results for the Shock Wave problem, with $\epsilon = 0.005$, solved over a fine mesh from MIRKDC, using full Newton iterations. The symbol $*$ indicates the maximum allowed number of iterations was exceeded ($k > MAXITERS$).

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	796	*** 7	5.433e+02	1.296e+07	
MIRK-1:	796	29	2.613e-05	3.562e+07	-174.9%
NEW-A:	796	30	2.613e-05	2.574e+07	-98.6%
NEW-B:	796	30	2.613e-05	2.612e+07	-101.6%

Table A.26: Results for the Shock Wave problem, with $\epsilon = 0.005$, solved over a fine mesh from MIRKDC, using damped Newton iterations. The symbol $***$ indicates the damped Newton iteration failed with the damping factor becoming less than the allowed minimum ($\lambda < \lambda_{min}$).

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	2	10	5.530e-01	3.644e+04	
MIRK-1:	2	** 3	2.389e+06	6.861e+03	81.2%
NEW-A:	2	** 3	9.544e+07	4.038e+03	88.9%
NEW-B:	2	** 3	9.544e+07	4.248e+03	88.3%
MIRK-S:	4	* 41	1.506e+03	2.927e+05	
MIRK-1:	4	** 4	1.051e+06	1.769e+04	94.0%
NEW-A:	4	** 3	1.142e+08	7.668e+03	97.4%
NEW-B:	4	** 3	1.142e+08	8.022e+03	97.3%
MIRK-S:	8	19	2.470e+01	2.685e+05	
MIRK-1:	8	** 2	1.451e+06	1.739e+04	93.5%
NEW-A:	8	** 1	7.250e+06	4.976e+03	98.1%
NEW-B:	8	** 1	7.250e+06	5.190e+03	98.1%
MIRK-S:	16	7	2.068e+01	1.968e+05	
MIRK-1:	16	** 5	2.556e+06	8.620e+04	56.2%
NEW-A:	16	** 4	8.709e+06	3.926e+04	80.0%
NEW-B:	16	** 4	8.709e+06	4.089e+04	79.2%
MIRK-S:	32	* 41	3.430e+03	2.299e+06	
MIRK-1:	32	** 10	6.418e+05	3.433e+05	85.1%
NEW-A:	32	** 7	1.237e+06	1.365e+05	94.1%
NEW-B:	32	** 7	1.237e+06	1.420e+05	93.8%
MIRK-S:	64	8	1.123e+01	8.962e+05	
MIRK-1:	64	** 10	1.066e+06	6.850e+05	23.6%
NEW-A:	64	** 4	4.852e+06	1.554e+05	82.7%
NEW-B:	64	** 4	4.852e+06	1.617e+05	82.0%

Table A.27: Results for the Shock Wave problem, with $\epsilon = 0.005$, using full Newton iterations, with a good initial guess. The symbol $*$ indicates the maximum allowed number of iterations was exceeded ($k > MAXITERS$). The symbol $**$ indicates that the iteration diverged ($n > MAXN$).

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	2	*** 11	4.756e-01	5.336e+04	
MIRK-1:	2	*** 1	9.215e-02	4.932e+03	90.8%
NEW-A:	2	*** 1	4.207e-01	3.991e+03	92.5%
NEW-B:	2	*** 1	4.207e-01	4.064e+03	92.4%
MIRK-S:	4	13	3.927e+00	1.183e+05	
MIRK-1:	4	*** 1	1.832e-01	9.480e+03	92.0%
NEW-A:	4	*** 1	2.524e-01	7.613e+03	93.6%
NEW-B:	4	*** 1	2.524e-01	7.692e+03	93.5%
MIRK-S:	8	*** 12	2.090e+01	2.259e+05	
MIRK-1:	8	*** 2	1.491e+03	3.060e+04	86.5%
NEW-A:	8	** 1	7.250e+04	1.486e+04	93.4%
NEW-B:	8	** 1	7.250e+04	1.495e+04	93.4%
MIRK-S:	16	9	2.068e+01	3.187e+05	
MIRK-1:	16	*** 1	6.076e+00	3.677e+04	88.5%
NEW-A:	16	*** 1	1.134e+01	2.934e+04	90.8%
NEW-B:	16	*** 1	1.134e+01	2.946e+04	90.8%
MIRK-S:	32	7	1.298e+01	4.969e+05	
MIRK-1:	32	*** 1	1.138e+01	7.315e+04	85.3%
NEW-A:	32	*** 7	7.102e+00	2.666e+05	46.4%
NEW-B:	32	*** 7	7.102e+00	2.700e+05	45.7%
MIRK-S:	64	8	2.510e+00	1.130e+06	
MIRK-1:	64	*** 3	6.888e+00	3.350e+05	70.4%
NEW-A:	64	*** 1	7.034e+00	1.163e+05	89.7%
NEW-B:	64	*** 1	7.034e+00	1.165e+05	89.7%

Table A.28: Results for the Shock Wave problem, with $\epsilon = 0.005$, using damped Newton iterations, with a good initial guess. The symbol $***$ indicates the damped Newton iteration failed with the damping factor becoming less than the allowed minimum ($\lambda < \lambda_{min}$).

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	796	5	2.613e-05	6.958e+06	
MIRK-1:	796	5	2.613e-05	4.251e+06	38.9%
NEW-A:	796	5	2.613e-05	2.409e+06	65.4%
NEW-B:	796	5	2.613e-05	2.504e+06	64.0%

Table A.29: Results for the Shock Wave problem with $\epsilon = 0.005$, solved over a fine mesh from MIRKDC, using full Newton iterations, with a good initial guess.

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	796	5	2.613e-05	8.893e+06	
MIRK-1:	796	5	2.613e-05	5.868e+06	34.0%
NEW-A:	796	5	2.613e-05	4.025e+06	54.7%
NEW-B:	796	5	2.613e-05	4.093e+06	54.0%

Table A.30: Results for the Shock Wave problem, with $\epsilon = 0.005$, solved over a fine mesh from MIRKDC, using damped Newton iterations, with a good inial guess.

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	2	6	2.308e+00	2.504e+06	
MIRK-1:	2	* 31	3.450e+00	6.551e+06	-161.6%
NEW-A:	2	*** 12	3.273e+00	3.320e+05	86.7%
NEW-B:	2	*** 12	3.273e+00	2.840e+05	88.7%
MIRK-S:	4	8	1.942e-01	6.652e+06	
MIRK-1:	4	* 31	1.942e-01	1.293e+07	-94.3%
NEW-A:	4	* 41	1.942e-01	1.823e+06	72.6%
NEW-B:	4	* 41	1.942e-01	1.511e+06	77.3%
MIRK-S:	8	7	1.159e-02	1.158e+07	
MIRK-1:	8	20	1.159e-02	1.645e+07	-42.0%
NEW-A:	8	24	1.159e-02	1.896e+06	83.6%
NEW-B:	8	24	1.159e-02	1.541e+06	86.7%
MIRK-S:	16	6	9.203e-05	1.983e+07	
MIRK-1:	16	10	9.203e-05	1.640e+07	17.3%
NEW-A:	16	12	9.203e-05	1.839e+06	90.7%
NEW-B:	16	12	9.203e-05	1.481e+06	92.5%
MIRK-S:	32	6	1.597e-06	3.963e+07	
MIRK-1:	32	8	1.597e-06	2.620e+07	33.9%
NEW-A:	32	9	1.598e-06	2.716e+06	93.1%
NEW-B:	32	9	1.598e-06	2.177e+06	94.5%
MIRK-S:	64	7	2.764e-07	9.244e+07	
MIRK-1:	64	8	2.764e-07	5.236e+07	43.4%
NEW-A:	64	8	2.764e-07	4.790e+06	94.8%
NEW-B:	64	8	2.764e-07	3.830e+06	95.9%

Table A.31: Results for the Shock Wave problem repeated 10 times, with $\epsilon = 0.1$, using damped Newton iterations. The symbol $*$ indicates the maximum allowed number of iterations was exceeded ($k > MAXITERS$). The symbol $***$ indicates the damped Newton iteration failed with the damping factor becoming less than the allowed minimum ($\lambda < \lambda_{min}$).

	# sub intervals	# Newton iterations	error in computed sol'n	total FLOPs	savings
MIRK-S:	2	12	5.737e+00	5.021e+06	
MIRK-1:	2	*** 2	4.893e+00	4.332e+05	91.4%
NEW-A:	2	*** 2	4.311e+00	6.155e+04	98.8%
NEW-B:	2	*** 2	4.311e+00	5.286e+04	98.9%
MIRK-S:	4	*** 7	2.041e+00	5.861e+06	
MIRK-1:	4	*** 6	2.894e+00	2.508e+06	57.2%
NEW-A:	4	*** 3	7.609e+00	1.521e+05	97.4%
NEW-B:	4	*** 3	7.609e+00	1.268e+05	97.8%
MIRK-S:	8	13	3.669e-01	2.156e+07	
MIRK-1:	8	*** 14	1.417e-01	1.171e+07	45.7%
NEW-A:	8	*** 18	1.949e+00	1.645e+06	92.4%
NEW-B:	8	*** 18	1.949e+00	1.347e+06	93.8%
MIRK-S:	16	11	3.133e-02	3.645e+07	
MIRK-1:	16	* 31	3.133e-02	5.093e+07	-39.7%
NEW-A:	16	*** 15	1.672e-01	2.585e+06	92.9%
NEW-B:	16	*** 15	1.672e-01	2.095e+06	94.3%
MIRK-S:	32	10	1.453e-04	6.652e+07	
MIRK-1:	32	15	1.453e-04	4.922e+07	26.0%
NEW-A:	32	17	1.453e-04	5.224e+06	92.1%
NEW-B:	32	17	1.453e-04	4.191e+06	93.7%
MIRK-S:	64	10	3.183e-06	1.322e+08	
MIRK-1:	64	12	3.183e-06	7.873e+07	40.5%
NEW-A:	64	12	3.182e-06	7.371e+06	94.4%
NEW-B:	64	12	3.182e-06	5.902e+06	95.5%

Table A.32: Results for the Shock Wave problem repeated 10 times, with $\epsilon = 0.05$, using damped Newton iterations. The symbol $*$ indicates the maximum allowed number of iterations was exceeded ($k > MAXITERS$). The symbol $***$ indicates the damped Newton iteration failed with the damping factor becoming less than the allowed minimum ($\lambda < \lambda_{min}$).

A.3.2 Condition and Spectral Radius

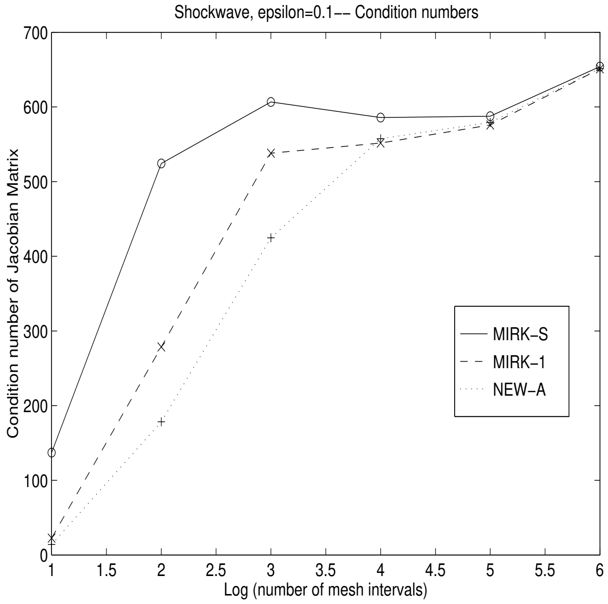

Figure A.14: Condition Numbers for the Jacobian matrices, for the Shock Wave problem with $\epsilon = 0.1$.

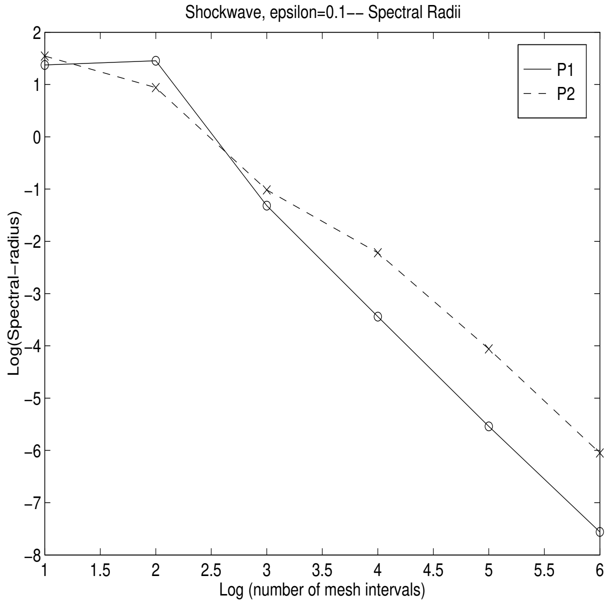

Figure A.15: Shock Wave with $\epsilon = 0.1$. P1=spectral radius of G' calculated using MIRK-1 Jacobian. P2=spectral radius of G' calculated using approximate Jacobian.

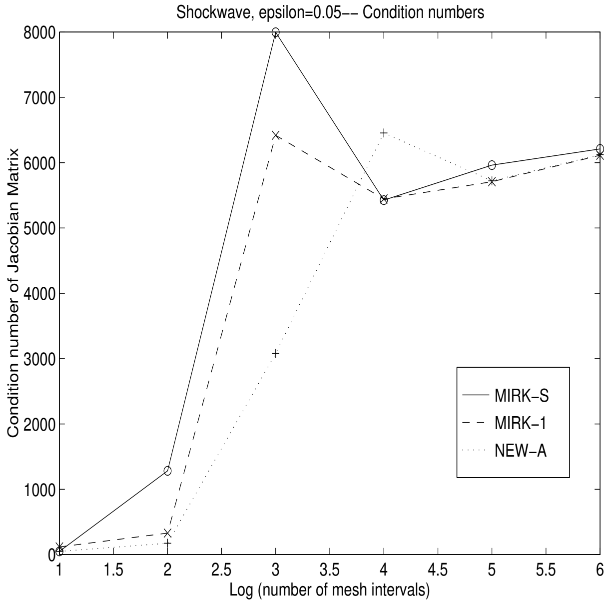

Figure A.16: Condition Numbers of the Jacobian matrices, for the Shock Wave problem with $\epsilon = 0.05$.

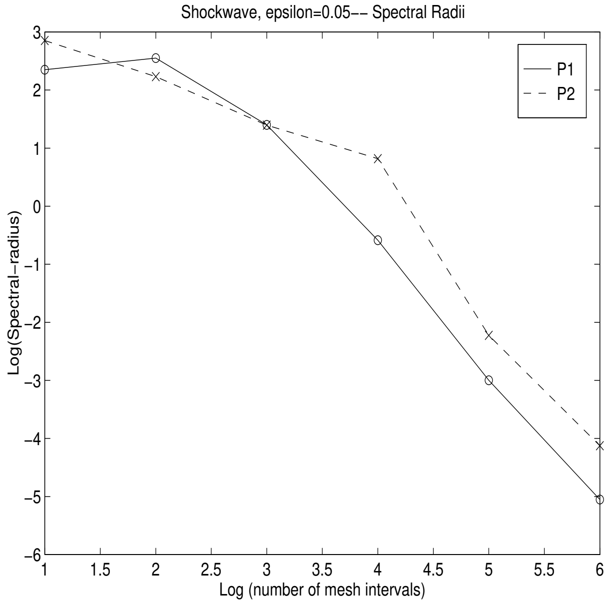

Figure A.17: Shock Wave problem with $\epsilon = 0.05$. P1=spectral radius of G' calculated using MIRK-1 Jacobian. P2=spectral radius of G' calculated using approximate Jacobian.

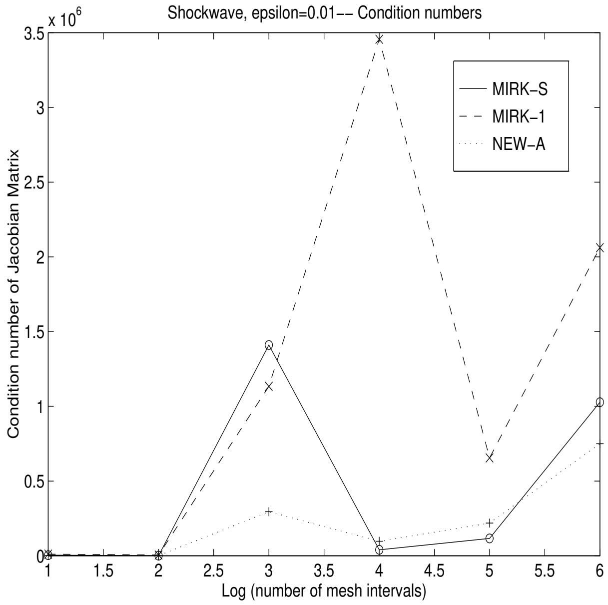

Figure A.18: Condition Numbers of the Jacobian matrices, for the Shock Wave problem with $\epsilon = 0.01$.

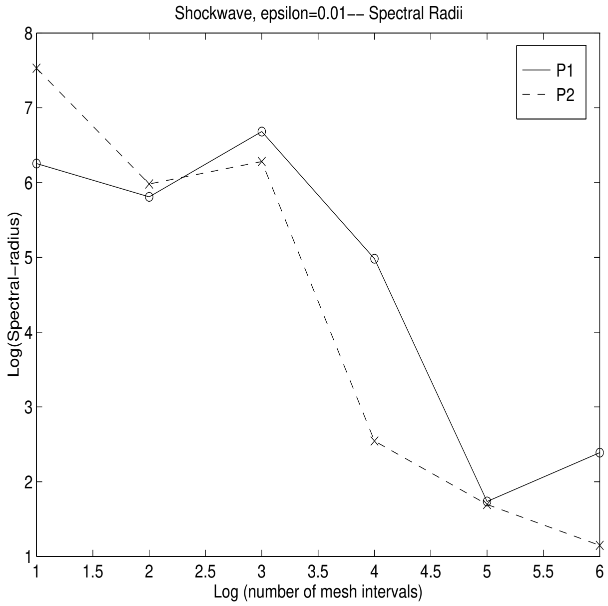

Figure A.19: Shock Wave problem with $\epsilon = 0.01$. P1=spectral radius of G' calculated using MIRK-1 Jacobian. P2=spectral radius of G' calculated using approximate Jacobian.

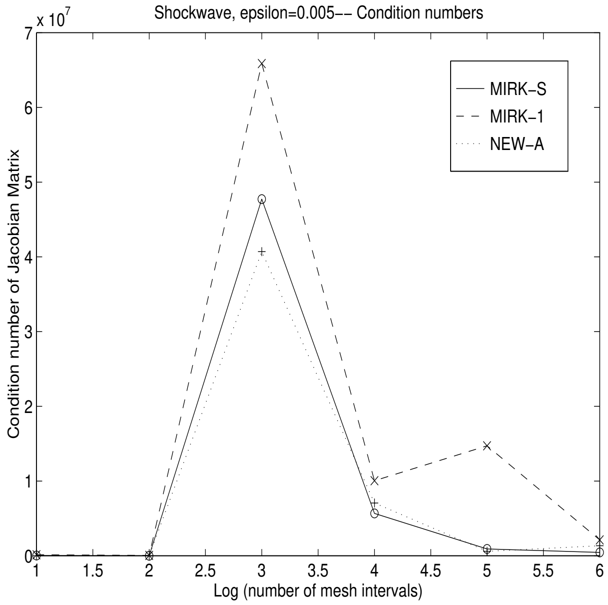

Figure A.20: Condition numbers of the Jacobian matrices, for the Shock Wave problem with $\epsilon = 0.005$.

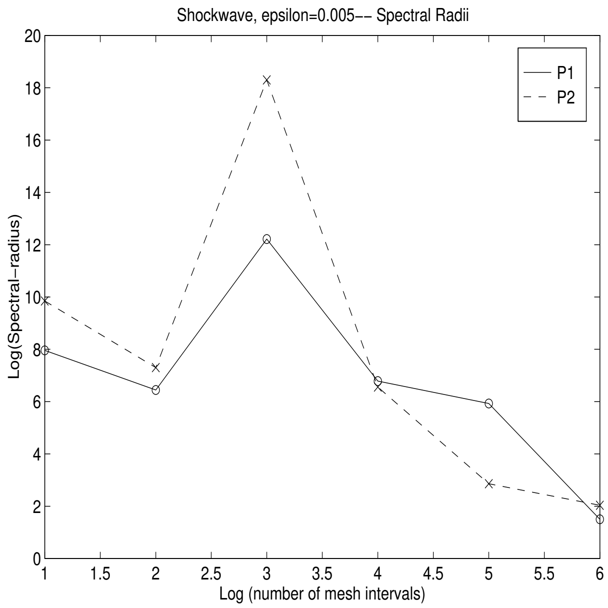

Figure A.21: Shock Wave problem with $\epsilon = 0.005$. P1=spectral radius of G' calculated using MIRK-1 Jacobian. P2=spectral radius of G' calculated using approximate Jacobian.

A.3.3 Error in Newton steps

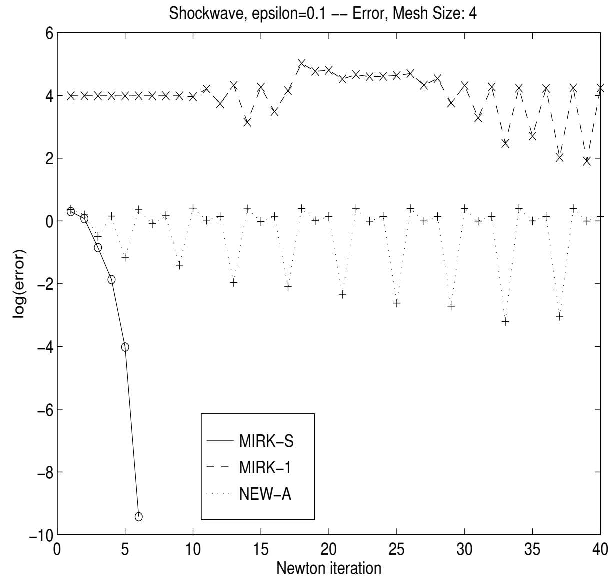

Figure A.22: Error in solution to the nonlinear system, for the Shock Wave problem with $\epsilon = 0.1$, and 4 mesh intervals, using full Newton iterations.

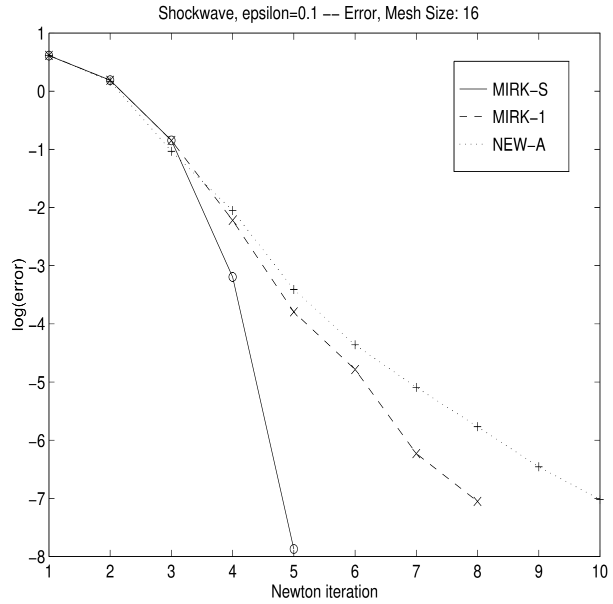

Figure A.23: Error in solution to the nonlinear system, for the Shock Wave problem with $\epsilon = 0.1$, and 16 mesh intervals, using full Newton iterations.

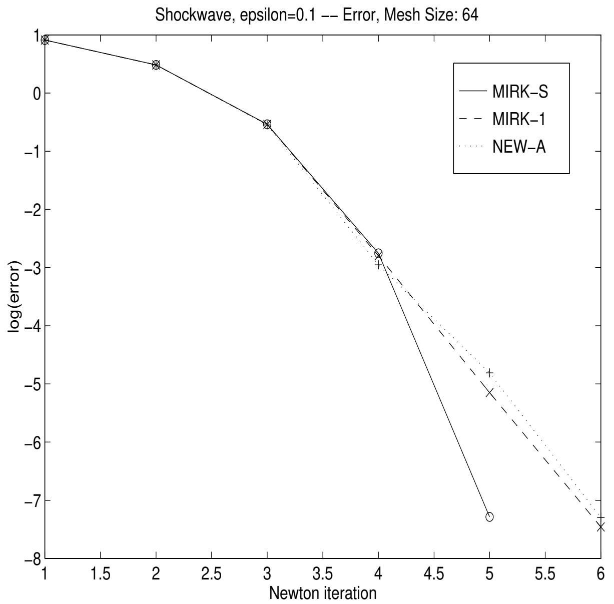

Figure A.24: Error in solution to the nonlinear system, for the Shock Wave problem with $\epsilon = 0.1$, and 64 mesh intervals, using full Newton iterations.

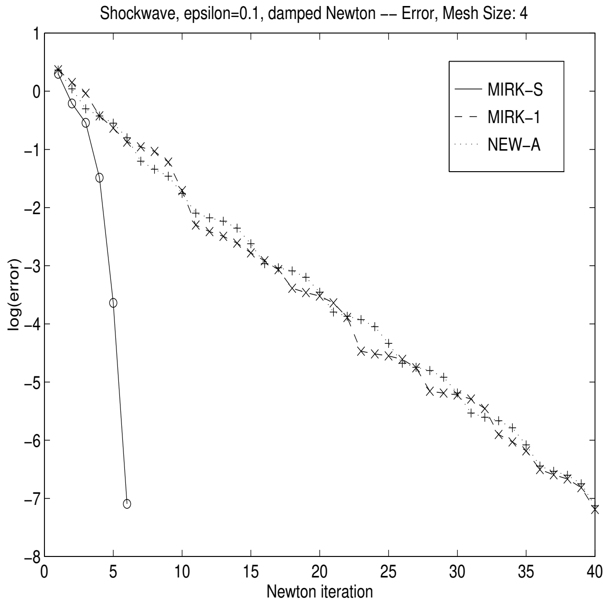

Figure A.25: Error in solution to the nonlinear system, for the Shock Wave problem with $\epsilon = 0.1$, and 4 mesh intervals, using damped Newton iterations.

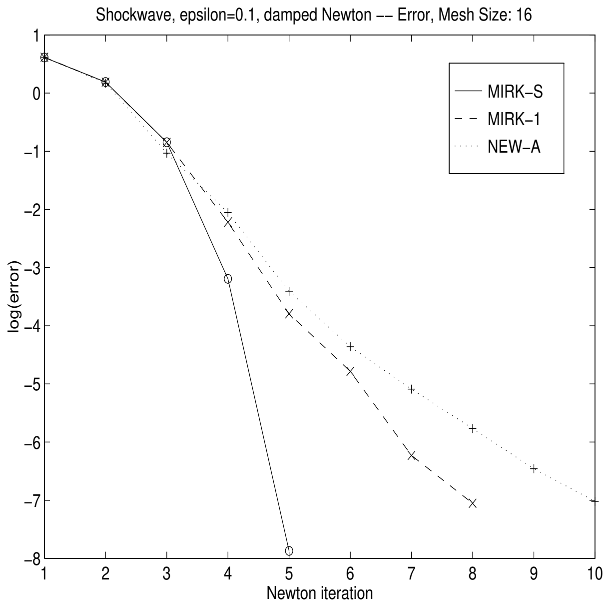

Figure A.26: Error in solution to the nonlinear system, for the Shock Wave problem with $\epsilon = 0.1$, and 16 mesh intervals, using damped Newton iterations.

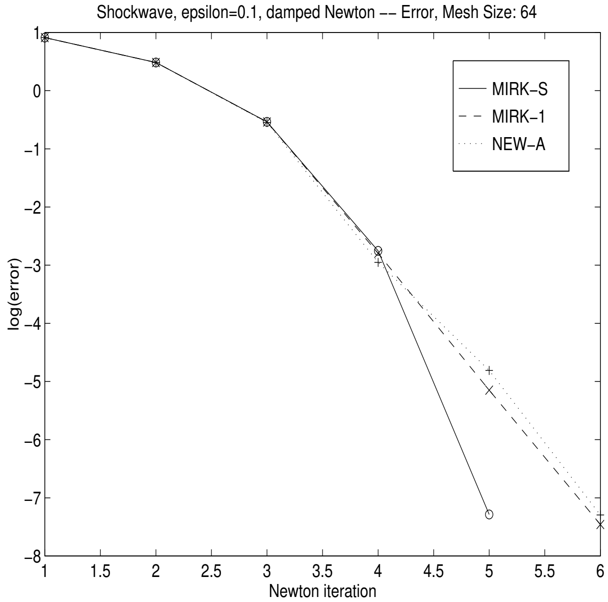

Figure A.27: Error in solution to the nonlinear system, for the Shock Wave problem with $\epsilon = 0.1$, and 64 mesh intervals, using damped Newton iterations.