Summer 2011

Hybrid Runge-Kutta and quasi-Newton methods for unconstrained nonlinear optimization

Darin Griffin Mohr
University of Iowa

HYBRID RUNGE-KUTTA AND QUASI-NEWTON METHODS FOR UNCONSTRAINED NONLINEAR OPTIMIZATION

by

Darin Griffin Mohr

An Abstract

Of a thesis submitted in partial fulfillment of the
requirements for the Doctor of Philosophy
degree in Applied Mathematical and Computational Sciences
in the Graduate College of
The University of Iowa

July 2011

Thesis Supervisor: Professor Laurent Jay

ABSTRACT

Finding a local minimizer in unconstrained nonlinear optimization and a fixed point of a gradient system of ordinary differential equations (ODEs) are two closely related problems. Quasi-Newton algorithms are widely used in unconstrained nonlinear optimization while Runge-Kutta methods are widely used for the numerical integration of ODEs. In this thesis, hybrid algorithms combining low-order implicit Runge-Kutta methods for gradient systems and quasi-Newton type updates of the Jacobian matrix such as the BFGS update are considered. These hybrid algorithms numerically approximate the gradient flow, but the exact Jacobian matrix is not used to solve the nonlinear system at each step. Instead, a quasi-Newton matrix is used to approximate the Jacobian matrix and matrix-vector multiplications are performed in a limited memory setting to reduce storage, computations, and the need to calculate Jacobian information.

For hybrid algorithms based on Runge-Kutta methods of order at least two, a curve search is implemented instead of the standard line search used in quasi-Newton algorithms. Stepsize control techniques are also performed to control the stepsize associated with the underlying Runge-Kutta method.

These hybrid algorithms are tested on a variety of test problems and their performance is compared with that of the limited memory BFGS algorithm.

Abstract Approved: _____

Thesis Supervisor

_____

Title and Department

_____

Date

HYBRID RUNGE-KUTTA AND QUASI-NEWTON METHODS FOR UNCONSTRAINED NONLINEAR OPTIMIZATION

by

Darin Griffin Mohr

A thesis submitted in partial fulfillment of the
requirements for the Doctor of Philosophy
degree in Applied Mathematical and Computational Sciences
in the Graduate College of
The University of Iowa

July 2011

Thesis Supervisor: Professor Laurent Jay

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

PH.D. THESIS

This is to certify that the Ph.D. thesis of

Darin Griffin Mohr

has been approved by the Examining Committee for the thesis requirement for the Doctor of Philosophy degree in Applied Mathematical and Computational Sciences at the July 2011 graduation.

Thesis Committee: _____
Laurent Jay, Thesis Supervisor

_____
Rodica Curtu

_____
Bruce Ayati

_____
Weimin Han

_____
Gerhard Strohmer

ACKNOWLEDGEMENTS

"Great people make you feel that, you too, can become great." This quote originates with Mark Twain and carries much truth.

Having been supported by wonderful and caring people throughout my life, I am truly blessed and filled with an energy to inspire others in the same fashion. I would like to thank my wife, Emma, and my family for their support, inspiration, kind words, and thoughtful actions. Lastly, I am very grateful for the patience and support I received from my advisor, Dr. Laurent Jay.

ABSTRACT

Finding a local minimizer in unconstrained nonlinear optimization and a fixed point of a gradient system of ordinary differential equations (ODEs) are two closely related problems. Quasi-Newton algorithms are widely used in unconstrained nonlinear optimization while Runge-Kutta methods are widely used for the numerical integration of ODEs. In this thesis, hybrid algorithms combining low-order implicit Runge-Kutta methods for gradient systems and quasi-Newton type updates of the Jacobian matrix such as the BFGS update are considered. These hybrid algorithms numerically approximate the gradient flow, but the exact Jacobian matrix is not used to solve the nonlinear system at each step. Instead, a quasi-Newton matrix is used to approximate the Jacobian matrix and matrix-vector multiplications are performed in a limited memory setting to reduce storage, computations, and the need to calculate Jacobian information.

For hybrid algorithms based on Runge-Kutta methods of order at least two, a curve search is implemented instead of the standard line search used in quasi-Newton algorithms. Stepsize control techniques are also performed to control the stepsize associated with the underlying Runge-Kutta method.

These hybrid algorithms are tested on a variety of test problems and their performance is compared with that of the limited memory BFGS algorithm.

TABLE OF CONTENTS

LIST OF TABLES

Table

LIST OF ALGORITHMS

Algorithm

CHAPTER 1
INTRODUCTION

Nonlinear optimization can be realized in many areas of life. In nature, systems tend to their lowest energy state. In business applications, finding minimum cost or maximum profits are crucial for the success of the business. These applications include transportation problems involving a set of goods, stock portfolio selections, and plant production management. In radiation therapy, a cancerous tumor can be hit with beams of radiation designed to kill the cancerous cells. Engineers wish to design machines that deliver these radiation beams so that the tumor is hit with the required radiation, but the radiation absorbed by the surrounding healthy tissue is minimized [19, 25, 27].

Given a nonlinear optimization problem, line search methods, trust-region methods, and methods based on solving a gradient system of differential equations are three classes of methods for the numerical solution of such problems. In one dimension, any student of calculus should know how to solve a minimization problem: to find a local minimizer of a given unconstrained, twice-continuously differentiable function of one variable, one differentiates the function, finds the zeros of this derivative, and checks whether these critical points yield local minima, local maxima, or local saddle points using, in general, second order sufficient conditions. In higher dimensions, the process is similar: taking the derivative corresponds to finding the gradient of the function, finding the zeros of the derivative corresponds to finding the zeros of the gradient, and characterizing the critical points using second order

sufficient conditions involves checking the positive definiteness of the Hessian matrix.

Line search methods and trust-region methods are based on locally approximating the objective function at each point with a local model function which has easily computable minimizers. This is usually a quadratic model. Line search methods use this approximation to generate a search direction and then search for an acceptable step length in this direction. These methods are sometimes taught in the undergraduate curriculum, for example, the steepest descent method for numerically finding local minima. However, this method is generally inefficient. Trust-region methods use a local model of the objective function to find a search direction and step length at the same time. This is done by solving a simple constrained minimization problem in a region around a current iterate. The size of this neighborhood, usually given by a certain radius, reflects how well the local model approximates the nonlinear function. If the local model is a poor approximation, the radius may be shrunk in hopes to more accurately approximate the objective function. If the approximation is good, the radius is either held constant or is increased. There is a wealth of literature on trust-region methods (for example, [7,11,15,20]), but they will not be considered in this thesis.

The main aim of this thesis is to develop new algorithms that combine techniques from both line search methods and numerical differential equations. These new hybrid algorithms are built on the implicit Runge-Kutta framework for numerically solving differential equations, but use a quasi-Newton matrix in place of a Jacobian matrix.

1.1 Definitions and Background

Throughout this thesis, $f : \mathbb{R}^n \to \mathbb{R}$ will denote the objective function that we want to minimize, $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$ denotes the gradient of f (column vector), and $\nabla^2 f : \mathbb{R}^n \to \mathbb{R}^{n\times n}$ denotes the Hessian matrix of f.

$$f(x) = f(x_1, x_2, \ldots, x_n), \qquad \nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(x_1, x_2, \ldots, x_n) \\ \frac{\partial f}{\partial x_2}(x_1, x_2, \ldots, x_n) \\ \vdots \\ \frac{\partial f}{\partial x_n}(x_1, x_2, \ldots, x_n), \end{bmatrix}$$

$$\nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1}(x_1, \ldots, x_n) & \frac{\partial^2 f}{\partial x_2 \partial x_1}(x_1, \ldots, x_n) & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_1}(x_1, \ldots, x_n) \\ \frac{\partial^2 f}{\partial x_1 \partial x_2}(x_1, \ldots, x_n) & \frac{\partial^2 f}{\partial x_2 \partial x_2}(x_1, \ldots, x_n) & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_2}(x_1, \ldots, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_1 \partial x_n}(x_1, \ldots, x_n) & \frac{\partial^2 f}{\partial x_2 \partial x_n}(x_1, \ldots, x_n) & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n}(x_1, \ldots, x_n) \end{bmatrix}$$

Unless otherwise stated, $x \in \mathbb{R}^n$, f is at least twice continuously differentiable, and the norm used in $\mathbb{R}^n$ is the Euclidean norm, $||x||_2 = \left(\sum_{i=1}^{n} x_i^2 \right)^{1/2}$. We begin by introducing some basic definitions.

Definition 1.1. Given a set $X \subseteq \mathbb{R}^n$, $x^* \in X$ is a *global minimizer of* f if and only if $f(x^*) \leq f(x)$ for all $x^* \in X$. If $f(x^*) < f(x)$ for all $x^* \in X \backslash \{x^*\}$, then x^* is called a *strict global minimizer of* f.

Definition 1.2. Given a set $X \subseteq \mathbb{R}^n$, $x^* \in X$ is a *global maximizer of* f if and only if $f(x^*) \geq f(x)$ for all $x^* \in X$. If $f(x^*) > f(x)$ for all $x^* \in X \backslash \{x^*\}$, then x^* is called a *strict global maximizer of* f.

Definition 1.3. Given a set $X \subseteq \mathbb{R}^n$, $x^* \in X$ is a *local minimizer of* f if and only if there exists $\delta > 0$ such that $f(x^*) \leq f(x)$ for all $x \in X \cap B(x^*, \delta)$. If $f(x^*) < f(x)$ for all $x \in (X \cap B(x^*, \delta))\backslash\{x^*\}$, then x^* is called a *strict local minimizer of* f.

Definition 1.4. Given a set $X \subseteq \mathbb{R}^n$, $x^* \in X$ is a *local maximizer of* f if and only if there exists $\delta > 0$ such that $f(x^*) \geq f(x)$ for all $x \in X \cap B(x^*, \delta)$. If $f(x^*) > f(x)$ for all $x \in (X \cap B(x^*, \delta))\backslash\{x^*\}$, then x^* is called a *strict local maximizer of* f.

The algorithms discussed in this thesis attempt to find local minimizers. Finding a global minimizer of a general nonlinear function f usually requires heuristics to escape basins of attraction corresponding to local, but nonglobal minimizers. This topic will not be discussed here. Notice that if x^* is a local maximizer of f, then x^* is a local minimizer of $-f$. Therefore, we only consider the problem of finding local minimizers of f since the maximization problem can be reformulated as a minimization problem.

Definition 1.5. Given a set $X \subseteq \mathbb{R}^n$, $x^* \in X$ is a *isolated local minimizer of* f if and only if there exists $\delta > 0$ such that x^* is the unique local minimizer in $X \cap B(x^*, \delta)$.

Ordinary differential equations (ODEs) are discussed in greater detail in Section 1.1.3, but we will introduce a few basic definitions concerning ODEs which are the basis of a large portion of this thesis.

Definition 1.6. A differential equation of the form

$$\dot{x} = -\nabla f(x) \tag{1.1}$$

is called a *gradient system*. The flow $\phi_t(x_0) := x(t, 0, x_0)$ to the gradient system is called the *gradient flow* where $x(t, t_0, x_0)$ denotes the solution at time t of the initial value problem with initial condition $x(t_0) = x_0$.

Definition 1.7. A point $x^* \in \mathbb{R}^n$ is called an *equilibrium point* (or *fixed point*, *critical point*, *stationary point*) of the differential equation $\dot{x} = g(x)$ if $g(x^*) = 0$.

Therefore, an equilibrium point of the gradient system (1.6) is any point x^* which satisfies $\nabla f(x^*) = 0$.

Definition 1.8. A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is *Lipschitz continuous at* x^* if there exists a constant $L > 0$ and $\delta > 0$ so that for all $x \in \mathbb{R}^n$ satisfying $||x - x^*|| < \delta$, we have

$$||f(x) - f(x^*)|| \leq L||x - x^*||.$$

Definition 1.9. Given a set $S \subseteq \mathbb{R}^n$, a function $f : \mathbb{R}^n \to \mathbb{R}^m$ is *Lipschitz continuous on* S if there exists a constant $L > 0$ so that

$$||f(x) - f(y)|| \leq L||x - y||$$

for all x and y in S.

Definition 1.10. A function $f : \mathbb{R}^n \to \mathbb{R}$ is called *convex* if for any $x,\ y \in \mathbb{R}^n$,

$$f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y)$$

holds for all $\alpha \in [0,\ 1]$. If this inequality is strict, than f is called *strictly convex*.

Definition 1.11. A continuously differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ is called *strongly convex* if there is an $m > 0$ so that

$$f(y) \geq f(x) + \nabla f(x)^T(y - x) + \frac{m}{2}||y - x||^2$$

holds for any $x,\ y \in \mathbb{R}^n$.

1.1.1 Line Search Algorithms

Line search algorithms are globalization methods used in nonlinear optimization to find local minima of functions. Given a current iterate x_k and a search direction p_k, these methods define the next iterate, x_{k+1}, by the formula

$$x_{k+1} = x_k + \alpha_k p_k \tag{1.2}$$

where α_k is called the step length. Often, $p_k = -M_k^{-1}\nabla f(x_k)$ where M_k is a symmetric positive definite matrix. We begin by introducing some background definitions regarding line search methods.

Definition 1.12. $p_k \in \mathbb{R}^n$ is a *direction of decrease* for the function f at x if there exists $\sigma > 0$ such that $f(x + \alpha p_k) < f(x)$ for all $\alpha \in (0, \sigma)$.

Definition 1.13. $p_k \in \mathbb{R}^n$ is a *descent direction* for the function f at x if $\nabla f(x)^T p_k < 0$.

Lemma 1.14. *If f is continuously differentiable at x and $p \in \mathbb{R}^n$ is a descent direction for f at x, then p is a direction of decrease for f at x.*

The converse of Lemma 1.14 is false in general.

Example 1.1. Consider $f(x) = x^6 - x^4$ where $x \in \mathbb{R}$. f has three relative extrema: a local maximum at $x = 0$ and two local minima $x = \pm\sqrt{2/3}$. It is easily shown that $x = 0$ is a strict local maximum and therefore, $p = 1$ is a direction of decrease for f at $x = 0$. However, $p = 1$ is not a descent direction for f since $\nabla f(x)^T p = f'(0) = 0 \not< 0$.

Now we present some choices for the search direction $p_k = -M_k^{-1}\nabla f(x_k)$.

Table 1.1: Popular Search Directions

1.	$M_k = I$ corresponds to the steepest descent direction, $p_k = -\nabla f(x_k)$.
2.	$M_k = \nabla^2 f(x_k)$ corresponds to Newton's direction, $p_k = -\nabla^2 f(x_k)^{-1}\nabla f(x_k)$. Notice that in general M_k is not necessarily positive definite.
3.	$M_k = (\lambda_k I + \nabla^2 f(x_k))$ corresponds to the modified Newton's direction, $p_k = -(\lambda_k I + \nabla^2 f(x_k))^{-1}\nabla f(x_k)$. For certain values of λ_k (see Lemma 1.15 below), we can ensure that M_k remains positive definite.
4.	M_k a symmetric positive definite matrix satisfying certain properties corresponds to quasi-Newton methods. This is further discussed in Section 1.1.2.

The following proposition provides a necessary and sufficient condition on λ so that the matrix $(\lambda I + Q)$ is positive definite where Q is a symmetric positive definite matrix.

Lemma 1.15. *Let $\beta_1, \beta_2, \ldots, \beta_n$ be the eigenvalues of a symmetric positive definite matrix $Q \in \mathbb{R}^{n\times n}$ and let $v_1, v_2, \ldots, v_n$ be the corresponding eigenvectors of Q. Then*

$\lambda > -\min_{i=1,\ldots,n} \beta_i$ *if and only if* $(\lambda I + Q)$ *is positive definite.*

Proof. Let $\sigma_1, \sigma_2, \ldots, \sigma_n$ be the eigenvalues of $(\lambda I + Q)$. Then

$$(\lambda I + Q)v_i = \lambda v_i + Qv_i = (\lambda + \beta_i)v_i = \sigma_i v_i$$

Hence, $\sigma_i = (\lambda + \beta_i)$ is an eigenvalue of $(\lambda I + Q)$ with corresponding eigenvector v_i for all $i = 1, \ldots, n$. Therefore, the matrix $(\lambda I + Q)$ is positive definite if and only if $\sigma_i > 0$; that is, if and only if $\lambda > \max_{i=1,\ldots,n} -\beta_i = -\min_{i=1,\ldots,n} \beta_i$. □

If we have a descent direction p_k at iterate k, consider $m_k(\alpha) := f(x_k + \alpha p_k)$. As shown above, $m_k'(0) = \nabla f(x_k)^T p_k < 0$ since p_k is a descent direction for the function f. Ideally, we choose α_k to be the solution to the scalar minimization problem,

$$\min_{\alpha > 0} m_k(\alpha)$$

and set the next iterate $x_{k+1} = x_k + \alpha_k p_k$. This is called an *exact line search*. However, finding this minimizer may be difficult and inefficient in general. Therefore, in practice, we perform an *inexact line search*; that is, we choose to accept a value of α_k which provides sufficient decrease of the objective function.

Definition 1.16. Let $\gamma_1 \in (0, 1)$ be given and consider $m_k(\alpha) = f(x_k + \alpha p_k)$. Then α^* provides *sufficient decrease* of the function $f(x)$ if

$$m_k(\alpha^*) \leq m(0) + \gamma_1 \alpha^* m_k'(0). \tag{1.3}$$

This sufficient decrease condition is also called the *Armijo condition*.

The coefficient γ_1 ensures that the accepted point decreases the objective function by at least $\gamma_1 \alpha^* m_k'(0) = \gamma_1 \alpha^* \nabla f(x_k)^T p_k$. The Armijo condition with $\gamma_1 \in (0, 1)$

prevents the line search method from accepting a point which does not decrease the objective function enough (see Example 1.2).

Example 1.2. This example shows the importance of the Armijo condition. Let $f(x) = x^2$, where $x \in \mathbb{R}$, $x_0 = 2$, and assume a line search method generates search directions $p_k = (-1)^{k+1}$ and step lengths $\alpha_k = 2 + 3(1/2)^{k+1}$ for all $k \geq 0$. This produces the iterates $x_k = (1 + (1/2)^k)(-1)^k$ for $k = 0, 1, 2, \ldots$.

$$\begin{aligned} f(x_{k+1}) - f(x_k) &= (1 + (1/2)^{k+1})^2 - (1 + (1/2)^k)^2 \\ &= 1 + (1/2)^k + (1/4)^{k+1} - 1 - (1/2)^{k-1} - (1/4)^k \\ &= (1/2)^{k-1}(1/2 - 1) + (1/4)^k(1/4 - 1) \\ &= -(1/2)^k - (3/4)(1/4)^k \\ &< 0 \end{aligned}$$

Thus, $\{f(x_k)\}_{k\geq 0}$ is a strictly decreasing sequence with $\lim_{k\to\infty} f(x_k) = 1$. However, the iterates fail to converge to a single point as the limit of x_k as $k \to \infty$ is the set $\{-1, 1\}$. Therefore, the condition that x_{k+1} decrease the objective function at each iteration, that is $f(x_{k+1}) < f(x_k)$, is not enough to guarantee convergence.

For this example, the Armijo condition is always violated if $\gamma_1 \in (1/8, 1)$ and if $\gamma_1 \in (0, 1/8]$, the Armijo condition is violated when

$$k \geq \frac{\ln\left(\frac{1}{4\gamma_1} - 1\right)}{\ln 2}. \tag{1.4}$$

To see this, we find conditions on k so that $f(x_k + \alpha_k p_k) > f(x_k) + \gamma_1 \alpha_k \nabla f(x_k)^T p_k$.

Using the definition of x_k, α_k, and p_k,

$$\begin{aligned} f(x_k + \alpha_k p_k) - f(x_k) - \gamma_1 \alpha_k \nabla f(x_k)^T p_k &= (x_k + \alpha_k p_k)^2 - x_k^2 - \gamma_1 \alpha_k (2x_k) p_k \\ &= (2x_k p_k - 2\gamma_1 x_k p_k + \alpha_k)\alpha_k \\ &= ((1/2)^k(-1 + 4\gamma_1) + 4\gamma_1)\left(\frac{\alpha_k}{2}\right) \end{aligned}$$

Therefore, $f(x_k + \alpha_k p_k) > f(x_k) + \gamma_1 \alpha_k \nabla f(x_k)^T p_k$ if and only if $(1/2)^k(-1 + 4\gamma_1) + 4\gamma_1 > 0$. The quantity $(1/2)^k(-1 + 4\gamma_1) + 4\gamma_1$ is positive for all values of $k \geq 0$ if $\gamma_1 > 1/8$. If $\gamma_1 \leq 1/8$, then by applying logarithms to this inequality, we find (1.4).

The Armijo condition is the first condition that we impose on line search methods. However, this condition is not sufficient to ensure convergence of line search methods to a local minimum. To ensure convergence, we introduce a second condition.

Definition 1.17. Let $\gamma_2 \in (\gamma_1, 1)$ where γ_1 is the constant from the Armijo condition (1.3) and consider $m_k(\alpha) = f(x_k + \alpha p_k)$. Then α^* satisfies the *curvature condition* for the function f at x_k if

$$m_k'(\alpha^*) \geq \gamma_2 m_k'(0). \tag{1.5}$$

The curvature condition ensures that the accepted value of α results in the quantity $m_k'(\alpha^*)$ being less negative than $\gamma_2 m_k'(0)$. This condition prevents a choice of α which results in the next iterate x_{k+1} being 'too close' to the current iterate x_k.

Example 1.3. This example illustrates that even with the Armijo condition, convergence of a line search algorithm is not guaranteed in the absence of the curvature condition. Consider the scalar function $f(x) = x^2$ and $x_0 = 2$ from Example 1.2.

Let $p_k = -1$ and $\alpha_k = (1/2)^{k+1}$ for $k \geq 0$ be the search directions and step lengths respectively. This generates the sequence of iterates defined by $x_k = 1 + (1/2)^k$ for all $k \geq 0$. Then, as in the proof from Example 1.2, $f(x_{k+1}) < f(x_k)$ for all k. We have $\lim_{k\to\infty} x_k = 1$ Additionally,

$$\begin{aligned} f(x_k + \alpha_k p_k) - f(x_k) - \gamma_1 \alpha_k \nabla f(x_k)^T p_k &= (x_k - \alpha_k)^2 - x_k^2 + \gamma_1 \alpha_k (2x_k) \\ &= (-2x_k + \alpha_k + 2\gamma_1 x_k)\alpha_k \\ &= ((1/2)^k(-3 + 4\gamma_1) + 4\gamma_1 - 4)\left(\frac{\alpha_k}{2}\right). \end{aligned}$$

For the Armijo condition to hold, we require $(1/2)^k(-3+4\gamma_1)+4\gamma_1-4 \leq 0$. Solving for γ_1, we find

$$\gamma_1 \leq \frac{2^k + 3/4}{2^k + 1}.$$

The right hand side of this inequality is a monotonically increasing function of k and is bounded below by $3/4$. Therefore, if $\gamma_1 \leq 3/4$, the step lengths satisfy the Armijo condition for all $k \geq 0$. The iterates $\{x_k\}_{k\geq 0}$ converge to $x = 1$ as $k \to \infty$, but the minimizer of f occurs at $x = 0$, not at $x = 1$. The problem with this example lies in the fact that the step lengths converge to zero too quickly. In fact, $|\nabla f(1)| = 2$ and we will see in Section 1.1.3 that $\nabla f(x^*) = 0$ is a necessary condition for x^* to be a local minimizer.

Definition 1.18. Together, the Armijo and curvature conditions (1.3) and (1.5) are

called the *first and second Wolfe conditions* respectively:

$$m_k(\alpha) \leq m_k(0) + \gamma_1 \alpha m_k'(0) \tag{1.6}$$

$$m_k'(\alpha) \geq \gamma_2 m_k'(0). \tag{1.7}$$

where $0 < \gamma_1 < \gamma_2 < 1$.

The *Goldstein and Price conditions* are similar to the Wolfe conditions except that $m_k'(\alpha)$ is replaced with an average slope,

$$m_k'(\alpha) \approx \frac{m_k(\alpha) - m_k(0)}{\alpha}.$$

This eliminates the need to perform one gradient evaluation per check.

Definition 1.19. Together, the Armijo condition and the discretized curvature condition are called the *Goldstein and Price conditions*. They are expressed as:

$$m_k(\alpha) \leq m_k(0) + \gamma_1 \alpha m_k'(0) \tag{1.8}$$

$$\frac{m_k(\alpha) - m_k(0)}{\alpha} \geq \gamma_2 m_k'(0). \tag{1.9}$$

Notice that for $\gamma_1 \to 0$, the Armijo condition is satisfied if the proposed iterate produces any decrease in the objective function f whereas for $\gamma_1 \to 1$, the proposed iterate must provide a decrease in the objective function that is more than $\alpha m_k'(0) = \alpha \nabla f(x_k)^T p_k$.

The curvature condition requires that the derivative $m'(\alpha)$ be greater than some fraction of $m'(0)$; that is, an increase in the derivative is required. For $\gamma_2 \to 0$, we search for an α such that $m_k'(\alpha)$ is nonnegative. Conversely, for $\gamma_2 \to 1$, the curvature

condition enforces the much looser requirement $m_k'(\alpha) \geq m_k'(0)$. In practice, γ_1 is usually chosen close to zero so that the Armijo condition is easily satisfied ($\gamma_1 = 10^{-4}$ for example) and the constant γ_2 is usually chosen close to one so that the curvature condition is easily satisfied ($\gamma_2 = 0.9$ for example) [25]. The following example, taken from [7], provides motivation for the choice $0 < \gamma_1 \leq 1/2$.

Lemma 1.20. *Assume that the univariate function $m_k(\alpha)$ is quadratic with $m_k'(0) < 0$. Let $0 < \gamma_1 < 1$, and α^* be the value of α which minimizes $m_k(\alpha)$. Then the Armijo condition is satisfied if and only if $\gamma_1 \in (0,\, 1/2]$.*

Proof. Since $m_k(\alpha)$ is a quadratic, we can write

$$m_k(\alpha) = m_k(0) + \alpha m_k'(0) + \frac{\alpha^2}{2} m_k''(0).$$

The minimum of m_k occurs at $\alpha^* = -m_k'(0)/m_k''(0)$ with minimum value

$$m_k(\alpha^*) = m_k(0) - \frac{m_k'(0)^2}{2m_k''(0)}.$$

Therefore,

$$\begin{aligned} m_k(\alpha^*) &= m_k(0) - \frac{m_k'(0)^2}{2m_k''(0)} \\ &= m_k(0) - \frac{\alpha^* m_k'(0)^2}{\alpha^* 2m_k''(0)} \\ &= m_k(0) + \frac{1}{2}\alpha^* m_k'(0) \\ &\leq m_k(0) + \gamma_1 \alpha^* m_k'(0) \end{aligned}$$

where the final inequality is true only if and only if $\gamma_1 \in (0,\, 1/2]$. □

Lemma 1.20 shows that the optimal step length is rejected in the case when $m_k(\alpha)$ is a quadratic and $\gamma_1 > 1/2$. Additionally, $f(x_k)$ can be approximated fairly

accurately by a quadratic function near a local minimizer so $m_k(\alpha)$ can be closely approximated by a quadratic. In this case, we do not want to rule out the optimal step length α^* by having $\gamma_1 > 1/2$.

One type of line search method is the *backtracking line search algorithm*. In this algorithm, α_0 is initialized as some positive constant, usually $\alpha_0 = 1$ at each iteration, or α is initialized by using some interpolating polynomial between the previous m iterates. The choice $\alpha_0 = 1$ is motivated by the fact that $\alpha = 1$ will satisfy either the Wolfe conditions or the Goldstein and Price conditions near a local minimizer x^* [7]. If the value of α^* is rejected, we backtrack towards zero by reducing the value of α until we find a value which satisfies either (1.18) or (1.19)

Another type of line search algorithm different from the backtracking line search is given in Algorithm 1.1. The main difference between this algorithm and the backtracking line search algorithm is the way the next α is calculated. We start by initializing α and endpoints $\alpha_L = 0$ and $\alpha_R = 0$. If the current value of α is rejected, we check which of the Wolfe conditions is violated and do the following: set $\alpha_R = \alpha$ if the Armijo condition is violated or set $\alpha_L = \alpha$ if the curvature condition is violated. If the current value of α has been rejected and $\alpha_R = 0$, we increase the value of α for the next proposed step length to check. Once $\alpha_R > 0$, we have a search interval $[\alpha_L, \alpha_R]$ and the next choice of α is chosen from this interval. Therefore, at each iteration, we change one of the endpoints and select a new value of α until an acceptable step length is found. The framework for Algorithm 1.1 is taken from [7].

When selecting $\alpha_{new} > \alpha$, we search for a step length that satisfies the Armijo

Algorithm 1.1 General Line Search Algorithm

Inputs: current point x_k, search direction p_k, constants $0 < \gamma_1 \leq 1/2 \leq \gamma_2 < 1$,

objective function f, and gradient ∇f

Initialize: $\alpha_L = 0$ and $\alpha_R = 0$

Generate initial $\alpha > 0$

while $m_k(\alpha) > m_k(0) + \gamma_1 \alpha m_k'(0)$ or $m_k'(\alpha) < \gamma_2 m_k'(0)$

if $m_k(\alpha) > m_k(0) + \gamma_1 \alpha m_k'(0)$

set $\alpha_R \leftarrow \alpha$

end

if $m_k(\alpha) \leq m_k(0) + \gamma_1 \alpha m_k'(0)$ and $m_k'(\alpha) < \gamma_2 m_k'(0)$

$\alpha_L \leftarrow \alpha$

end

if $t_R = 0$

select $\alpha_{new} > \alpha$

end

if $t_R \neq 0$

select $\alpha_{new} \in (\alpha_L, \alpha_R)$

end

$\alpha \leftarrow \alpha_{new}$

endwhile

Output: α

condition. Let $\alpha_{new}^{(i)}$ denote the proposed step length at iteration i of the general line search algorithm. To find an acceptable α_R, we choose $\alpha_{new}^{(i)} > \alpha$ so that in the event of repeated failure of the Armijo condition, $\lim_{i\to\infty} \alpha_{new}^{(i)} = \infty$. That is, in modifying $\alpha_R > 0$ to get $\alpha_R > 0$, we do not want to choose values for α_{new} which converge to a finite number. Once α_R is found, we select α_{new} from the interval $[\alpha_L, \alpha_R]$.

Algorithm 1.1 can be adapted for the Goldstein and Price line search conditions by replacing $m_k'(\alpha)$ with the difference approximation $(m_k(\alpha) - m_k(0))/\alpha$. The following lemma, taken from [7], proves some important properties about line searches using the Wolfe or Goldstein and Price conditions.

Lemma 1.21. *Let $f \in \mathcal{C}^1$ and assume that $m_k(\alpha) = f(x_k + \alpha p_k)$ is bounded below. Also assume that if $\alpha_R > 0$, the proposed step length for Algorithm 1.1 takes the midpoint value: $\alpha_{new} = (\alpha_L + \alpha_R)/2$. Then Algorithm 1.1 terminates. Additionally, the algorithm terminates if the Goldstein and Price conditions are used instead of the Wolfe conditions.*

Proof. We only show the result with the Wolfe conditions. Assume by way of contradiction that the algorithm does not terminate and that no $\alpha_R > 0$ exists. Then $m_k(\alpha^{(i)}) \leq \gamma_1 \alpha^{(i)} m_k'(0)$ for all sequences $\{\alpha^{(i)}\}$. Since $\alpha^{(i)} \to \infty$ as $i \to \infty$ and since $m_k'(0) < 0$, we have that $m_k(\alpha^{(i)}) \to -\infty$ as $i \to \infty$. This contradicts m_k being bounded below. Therefore, there must exist $\alpha > 0$ such that the Armijo condition is violated.

Given that some finite iterate forces $\alpha_R > 0$, all subsequent α values must to be contained in the interval $[\alpha_L, \alpha_R]$. Assume, by way of contradiction, that

the algorithm fails to find an acceptable α in this interval in a finite number of iterations. By definition, at each iteration, we either change the left endpoint of the right endpoint to the midpoint of this interval. Then the algorithm generates two sequences, $\{\alpha_L^{(i)}\}$ and $\{\alpha_R^{(i)}\}$, such that the interval $[\alpha_L, \alpha_R]$ is halved at each iteration and the sequences are monotonically increasing and decreasing respectively. Therefore,

$$\lim_{i\to\infty} \alpha_L^{(i)} = \alpha^* = \lim_{i\to\infty} \alpha_R^{(i)}.$$

Using the properties of α_L and α_R, this limit implies $m_k(\alpha^*) = m_k(0) + \gamma_1\alpha^* m_k'(0)$. For the right endpoints, we have

$$\begin{aligned} m_k(\alpha_R^{(i)}) &> m_k(0) + \gamma_1\alpha_R^{(i)} m_k'(0) \\ &= m_k(0) + \gamma_1(\alpha^* + \alpha_R^{(i)} - \alpha^*)m_k'(0) \\ &= m_k(\alpha^*) + \gamma_1(\alpha_R^{(i)} - \alpha^*)m_k'(0). \end{aligned}$$

Therefore, solving for $\gamma_1 m'(0)$, taking the limit as $i \to \infty$, and using $\gamma_1 < \gamma_2$ we have,

$$m_k'(\alpha^*) \geq \gamma_1 m_k'(0) > \gamma_2 m_k'(0). \tag{1.10}$$

However, for the left endpoints $\alpha_L(i)$, we know that $m_k'(\alpha_L^{(i)}) < \gamma_2 m_k'(0)$ for all $i \geq 0$. Taking limits on this inequality yields

$$m_k'(\alpha^*) \leq \gamma_2 m_k'(0). \tag{1.11}$$

Hence (1.10) and (1.11) produce a contradiction. Therefore, Algorithm 1.1 must terminate after a finite number of iterations. □

Now we present a theorem by Zoutendijk which guarantees convergence of line search methods of the form (1.2) when either the Wolfe or Goldstein and Price conditions are used. The proof of this theorem can be found in [25].

Theorem 1.22. *(Zoutendijk) Consider line search algorithms given by (1.2) and let p_k be a descent direction and α_k satisfy the Wolfe conditions, (1.6) and (1.7), or the Goldstein and Price conditions, (1.8) and (1.9), for all k. Let f be continuously differentiable with Lipschitz continuous gradient in an open set containing the level set $\mathcal{L}_{x_0} := \{x : f(x) \leq f(x_0)\}$. Lastly, assume f bounded below. Then*

$$\sum_{k\geq 0} c_k^2 ||\nabla f(x_k)||^2 < \infty \tag{1.12}$$

where c_k is defined by

$$c_k := -(\nabla f(x_k)^T p_k)/(||\nabla f(x_k)|| \cdot ||p_k||).$$

The BFGS Algorithm 1.2, which is discussed in Section 1.1.2, is locally convergent and in fact, it can be shown that the convergence rate is superlinear under certain assumptions. In Chapter 2, we investigate the convergence of various hybrid algorithms and their convergence rates. The proofs for the BFGS algorithm provide the basic framework for our proofs. Now we present a theorem by Dennis and Moré which provides a tool to check whether or not the convergence rate is superlinear. The proof can be found in [14].

Theorem 1.23. *(Dennis and Moré) Assume a line search method of the form $x_{k+1} = x_k + \alpha_k p_k$ is given, where p_k is a descent direction and α_k is chosen satisfying the*

Wolfe conditions with $0 < \gamma_1 \leq 1/2 < \gamma_2 < 1$. *If the sequence* $\{x_k\}$ *converges to* x^* *where* x^* *is an equilibrium point of the gradient system at which* $\nabla^2 f(x^*)$ *is positive definite, and if*

$$\lim_{k\to\infty} \frac{||\nabla f(x_k) + \nabla^2 f(x^*)p_k||}{||p_k||} = 0,$$

then x_k *converges to* x^* *superlinearly.*

If $p_k = B_k^{-1}\nabla f(x_k)$ as in quasi-Newton methods, Theorem 1.23 implies that a necessary condition for superlinear convergence is

$$\lim_{k\to\infty} \frac{||(B_k + \nabla^2 f(x^*))p_k||}{||p_k||} = 0.$$

Notice that superlinear convergence can be achieved without having the quasi-Newton matrices converge to the actual Hessian matrix at the local minimizer.

1.1.2 Quasi-Newton and Limited Memory Algorithms

Quasi-Newton directions method that use second order information to generate the search direction p_k at each iteration without using actual Hessian calculations. Given a current iterate x_k, consider the quadratic approximation

$$q_k(p) = f(x_k) + \nabla f(x_k)^T p + \frac{1}{2} p^T M_k p \tag{1.13}$$

where M_k is a symmetric positive definite matrix. The unique minimizer of this quadratic is $p_k = -M_k^{-1}\nabla f(x_k)$. Many line search algorithms are based on such quadratic approximations. For example, $M_k = I$ results in the steepest descent direction, $M_k = \nabla f(x_k)^2$ gives Newton's direction, and $M_k = \lambda_k I + \nabla f(x_k)^2$ produces

the modified Newton direction. Note that the matrix M_k of Newton's direction is not necessarily positive definite or even invertible and may not give a descent direction. This is the motivation behind the modified Newton direction which ensures the positive definiteness of the matrix M_k. The BFGS algorithm calculates $B_k := M_k$ with the following properties:

(1) B_{k+1} is updated from B_k based on current information of the objective function. Therefore, we consider the quadratic approximation (1.13) for the $k+1$ iteration of the line search method.

(2) $\nabla q_{k+1}(p)$ should match $\nabla f(x)$ at x_{k+1} and x_k. This results in the so called *secant equation* or *quasi-Newton condition*: $B_{k+1}s_k = y_k$ where $s_k = x_{k+1} - x_k$ and $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$.

(3) Since we want B_{k+1} to be positive definite, it is necessary to have $s_k^T B_{k+1} s_k > 0$. Using the secant equation, we can rewrite this inequality as the so called *curvature condition*: $s_k^T y_k > 0$ which we require.

(4) Compute B_{k+1} using the rank two correction $B_{k+1} = B_k + \gamma_k v_k v_k^T + \delta_k u_k u_k^T$ where v_k, u_k are vectors and γ_k and δ_k are constants to be determined.

The BFGS formula is given by

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{s_k^T y_k}. \tag{1.14}$$

To find the search direction, we compute $p_k = -B_k^{-1}\nabla f(x_k)$ which involves solving the linear system $B_k p_k = -\nabla f(x_k)$. Instead of calculating B_k^{-1}, we can use

the Sherman-Morrison-Woodbury formula to express the update formula (1.14) in terms of the approximate inverse Hessian $H_k = B_k^{-1} \approx \nabla^2 f(x_k)^{-1}$ [28].

Theorem 1.24. *(Sherman-Morrison-Woodbury) Let $A \in \mathbb{R}^{n\times n}$ be an invertible matrix and let $U, V \in \mathbb{R}^{n\times m}$. Then $I_m + V^T A^{-1} U$ is invertible if and only if $A + UV^T$ is invertible. Additionally,*

$$(A + UV^T)^{-1} = A^{-1} - A^{-1}U(I_m + V^T A^{-1} U)^{-1} V^T A^{-1}$$

$$(I_m + V^T A^{-1} U)^{-1} = I_m - V^T (A + UV^T)^{-1} U.$$

Proof. The expressions for $(A + UV^T)^{-1}$ and $(I_m + V^T A^{-1} U)^{-1}$ given above satisfy $(A + UV^T)^{-1}(A + UV^T) = I_n$ and $(I_m + V^T A^{-1} U)^{-1}(I_m + V^T A^{-1} U) = I_m$. Furthermore, since the inverse of one of these matrices depends on the invertibility of the other matrix, we have that $(I_m + V^T A^{-1} U)$ is invertible if and only if $(A + UV^T)$ is invertible. □

Corollary 1.25. *Let $A \in \mathbb{R}^{n\times n}$ be an invertible matrix and let $u, v \in \mathbb{R}^n$. Then $1 + v^T A^{-1} u \neq 0$ if and only if $A + uv^T$ is invertible. Additionally,*

$$(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1} u v^T A^{-1}}{1 + v^T A^{-1} u}$$

$$(1 + v^T A^{-1} u)^{-1} = 1 - v^T (A + uv^T)^{-1} u.$$

Lemma 1.26. *Let $H_k = B_k^{-1}$ for all $k \geq 0$ and suppose $y_k^T s_k > 0$. Then the BFGS update for H_{k+1} is given by,*

$$H_{k+1} = V_k^T H_k V_k + \rho_k s_k s_k^T \tag{1.15}$$

where

$$\rho_k = \frac{1}{y_k^T s_k}, \qquad V_k = I - \rho_k y_k s_k^T.$$

Proof. Using the BFGS update formula (1.14), let $A = B_k + y_k y_k^T / y_k^T s_k$, $u = -B_k s_k / (s_k^T B_k s_k)$, and $v = B_k s_k$. Then $B_{k+1} = A + uv^T$. In order to apply Corollary 1.25, we need A^{-1}. To find this, we apply Corollary 1.25 to $A = B_k + \bar{u}\bar{v}^T$ with $\bar{u} = y_k / y_k^T s_k$ and $\bar{v} = y_k$ where $\bar{u}, \bar{v} \in \mathbb{R}^n$.

$$\begin{aligned} A^{-1} &= B_k^{-1} - \frac{B_k^{-1}\bar{u}\bar{v}^T B_k^{-1}}{1 + \bar{v}^T B_k^{-1}\bar{u}} \\ &= B_k^{-1} - \left(1 + \frac{y_k^T B_k^{-1} y_k}{y_k^T s_k}\right)^{-1} \frac{B_k^{-1} y_k y_k^T B_k^{-1}}{y_k^T s_k} \\ &= B_k^{-1} - \frac{B_k^{-1} y_k y_k^T B_k^{-1}}{y_k^T s_k + y_k^T B_k^{-1} y_k} \end{aligned}$$

Notice that $1 + \bar{v}^T B_k^{-1}\bar{u} = 1 + (y_k^T B_k^{-1} y_k)/y_k^T s_k > 1$ since B_k^{-1} is positive definite and $y_k^T s_k > 0$. Now we find the inverse of B_{k+1} using u and v as above and Theorem 1.25 a second time. Setting $\rho_k = 1/(y_k^T s_k)$,

$$\begin{aligned} B_{k+1}^{-1} &= A^{-1} - \frac{A^{-1} u v^T A^{-1}}{1 + v^T A^{-1} u} \\ &= B_k^{-1} - \frac{B_k^{-1} y_k y_k^T B_k^{-1}}{y_k^T s_k + y_k^T B_k^{-1} y_k} + \frac{A^{-1} B_k s_k s_k^T B_k A^{-T}}{s_k^T B_k s_k - s_k B_k A^{-1} B_k s_k} \\ &= B_k^{-1} - \rho_k B_k^{-1} y_k s_k^T - \rho_k s_k y_k^T B_k^{-1} + \rho_k s_k s_k^T + \rho_k^2 s_k y_k^T B_k^{-1} y_k s_k^T \\ &= (I - \rho_k s_k y_k^T) B_k^{-1} (I - \rho_k s_k y_k^T) + \rho_k s_k s_k^T. \end{aligned}$$

Replacing B_{k+1}^{-1} and B_k^{-1} with H_{k+1} and H_k respectively gives the result. □

Using this update, we deal with the approximate inverse Hessian H_{k+1} and all inverse calculations are avoided. The BFGS algorithm written in terms of the approximate inverse Hessian matrix H_{k+1} is given in Algorithm 1.2.

Algorithm 1.2 BFGS Algorithm

Input: starting point x_0, convergence tolerance $tol > 0$, initial symmetric,

positive definite Hessian approximation H_0

$k \leftarrow 0$

while $||\nabla f(x_k)|| > tol$

$p_k \leftarrow -H_k \nabla f(x_k)$

Find α_k via Algorithm 1.1

$x_{k+1} \leftarrow x_k + \alpha_k p_k$

$s_k \leftarrow x_{k+1} - x_k$,

$y_k \leftarrow \nabla f(x_{k+1}) - \nabla f(x_k)$

Compute H_{k+1} via (1.15)

$k \leftarrow k + 1$

endwhile

Output: x_k

The following theorems summarize the properties of the BFGS algorithm.

Lemma 1.27. *If $p_k = -H_k \nabla f(x_k)$ is chosen according to Algorithm 1.2, then p_k is a descent direction for $f(x)$ for all k.*

Proof.

$$p_k^T \nabla f(x_k) = (-H_k \nabla f(x_k))^T \nabla f(x_k) = -\nabla f(x_k)^T H_k \nabla f(x_k) < 0$$

since H_k is symmetric, positive definite. □

Lemma 1.28. *If the Wolfe conditions are used in the line search of Algorithm 1.2 with $\gamma_2 < 1$, then the curvature condition $s_k^T y_k > 0$ is satisfied at each iteration.*

Proof. For the BFGS algorithm with Wolfe conditions, we use the curvature condition (1.5),

$$\begin{aligned} s_k^T y_k &= \alpha_k p_k^T \nabla f(x_{k+1}) - \alpha_k p_k^T \nabla f(x_k) \\ &= \alpha_k m_k'(\alpha_k) - \alpha_k m_k'(0) \\ &\geq \gamma_2 \alpha_k m_k'(0) - \alpha_k m_k'(0) \\ &= (\gamma_2 - 1)\alpha_k p_k^T \nabla f(x_k) \\ &> 0 \end{aligned}$$

where the last inequality follows from the fact that $\gamma_2 < 1$, $\alpha_k > 0$, and p_k is a descent direction for the function f at x_k by (1.27). □

Lemma 1.29. *If H_k is symmetric positive definite and $s_k^T y_k > 0$, then H_{k+1} given by formula (1.15) is symmetric positive definite.*

Proof. Let $q \in \mathbb{R}^n$. Since $s_k^T y_k > 0$, then $\rho_k = 1/(s_k^T y_k) > 0$. Using the hypothesis that H_k is positive definite,

$$\begin{aligned} q^T H_{k+1} q &= q^T V_k^T H_k V_k q + \rho_k q^T s_k s_k^T q \\ &= (V_k q)^T H_k (V_k q) + \rho_k (s_k^T q)^T (s_k^T q) \\ &= (V_k q)^T H_k (V_k q) + \rho_k (s_k^T q)^2 \\ &> 0. \end{aligned}$$

Therefore, H_{k+1} is positive definite. To show symmetry, notice that the outer product $s_k s_k^T$ is always symmetric. Also,

$$\begin{aligned}(V^T H_k V)_{ij} &= \sum_{p=1}^{n}\sum_{q=1}^{n} v_{pi} h_{pq} v_{qj} \\ &= \sum_{p=1}^{n}\sum_{q=1}^{n} v_{qj} h_{pq} v_{pi} \\ &= \sum_{q=1}^{n}\sum_{p=1}^{n} v_{pj} h_{qp} v_{qi} \\ &= (V^T H_k V)_{ji}\end{aligned}$$

where we use a change of variables and the fact that H_k is symmetric. Therefore, H_{k+1} is symmetric. □

From Lemmas 1.28 and 1.29, if the Wolfe conditions are used to determine the step length α_k, then the symmetry and positive definiteness of H_{k+1} is preserved from H_k, provided H_k is symmetric positive definite. Since the inverse of a symmetric positive definite matrix is symmetric positive definite, these properties are also preserved under (1.14) for the quasi-Newton matrix B_{k+1}.

Although no Hessian matrices are explicitly calculated, the BFGS algorithm requires the storage of a $n \times n$ symmetric matrix which uses $n(n-1)/2$ storage locations. Let $q \in \mathbb{R}^n$. In general, computing $H_k q$ requires $O(n^2)$ operations and updating H_k can be done in $O(n^2)$ operations. When n is very large, the storage, updating, and implementation costs associated with H_k are not practical. The limited memory BFGS algorithm (LBFGS) reduces both the memory and the cost per iteration of the standard BFGS algorithm when n is large. The motivation behind the LBFGS

algorithm lies in recursively applying the BFGS update (1.15),

$$\begin{aligned}
H_k &= V_{k-1}^T H_{k-1} V_{k-1} + \rho_{k-1} s_{k-1} s_{k-1}^T \\
&= (V_{k-1}^T V_{k-2}^T) H_{k-2} (V_{k-2} V_{k-1}) + \rho_{k-2} V_{k-1}^T s_{k-2} s_{k-2}^T V_{k-1} + \rho_{k-1} s_{k-1} s_{k-1}^T \\
&\vdots \\
&= (V_{k-1}^T \dots V_0^T) H_0 (V_{k-1} \dots V_0) \\
&\quad + \rho_0 (V_{k-1}^T \dots V_1^T) s_0 s_0^T (V_{k-1} \dots V_1) \\
&\quad + \rho_1 (V_{k-1}^T \dots V_2^T) s_1 s_1^T (V_{k-1} \dots V_2) \\
&\quad + \dots \\
&\quad + \rho_{k-2} (V_{k-1}^T) s_{k-2} s_{k-2}^T (V_{k-1}) \\
&\quad + \rho_{k-1} s_{k-1} s_{k-1}^T.
\end{aligned}$$

Instead of computing all $k+1$ recursions, we only consider m recursions. Rather than store the matrix H_{k-m}, a general matrix denoted by H_k^0 is used and we impose that H_k^0 be symmetric positive definite. This results in the reduced scheme,

$$\begin{aligned}
H_k =& (V_{k-1}^T \dots V_{k-m}^T) H_k^0 (V_{k-1} \dots V_{k-m}) \\
&+ \rho_{k-m} (V_{k-1}^T \dots V_{k-m+1}^T) s_{k-m} s_{k-m}^T (V_{k-m+1} \dots V_{k-1}) \\
&+ \rho_{k-m+1} (V_{k-1}^T \dots V_{k-m+2}^T) s_{k-m+1} s_{k-m+1}^T (V_{k-m+2} \dots V_{k-1}) \\
&+ \dots \\
&+ \rho_{k-2} (V_{k-1}^T) s_{k-2} s_{k-2}^T (V_{k-1}) \\
&+ \rho_{k-1} s_{k-1} s_{k-1}^T.
\end{aligned} \tag{1.16}$$

The product $H_k q$ can be computed easily using this scheme and is shown in Algorithm 1.3.

Algorithm 1.3 LBFGS Two Loop Recursion

Inputs: s, y, m, k, q, initial Hessian H_k^0

for $i = k-1, k-2, \ldots, k-m$

$\alpha_i \leftarrow \rho_i s_i^T q$

$q \leftarrow q - \alpha_i y_i$

endfor

$r \leftarrow H_k^0 q$

for $i = k-m, k-m+1, \ldots, k-1$

$\beta \leftarrow \rho_i y_i^T r$

$r \leftarrow r + s_i(\alpha_i - \beta)$

endfor

Output: $r := H_k q$

The choice of m controls how many vector pairs $\{s_k, y_k\}$ are stored. In practice, smaller values of m result in less memory requirements and fewer floating point operations per iteration, but requires a greater number of iterations to achieve the termination criteria. Further discussion on the impact of different choices of m can be found in [21, 24].

The LBFGS algorithm, shown in Algorithm 1.4, and Algorithm 1.2 are equivalent when $m = \infty$ and the choice of initial matrix H_k^0 for the LBFGS and H_0 for the BFGS are the same.

Algorithm 1.4 LBFGS Algorithm

Inputs: starting point x_0, $m > 0$, convergence tolerance $tol > 0$

$k \leftarrow 0$

while $||\nabla f(x_k)|| > tol$

Choose initial symmetric positive definite matrix H_k^0

$p_k \leftarrow -H_k \nabla f(x_k)$ via Algorithm 1.3

Find α_k via Algorithm 1.1

$x_{k+1} \leftarrow x_k + \alpha_k p_k$

if $k > m$

Discard $\{s_{k-m}, y_{k-m}\}$ from memory

endif

$s_k \leftarrow x_{k+1} - x_k$

$y_k \leftarrow \nabla f(x_{k+1}) - \nabla f(x_k)$

$k \leftarrow k + 1$

endwhile

Output: x_k

The main advantage behind the LBFGS algorithm is that it does not store the matrix H_k at each iteration. Instead, it only stores the latest m vectors s_i and y_i, $i = k-1, k-2, \ldots, k-m$ and uses these vectors to approximate matrix-vector products involving the inverse Hessian at the current iterate. If H_k^0 is chosen to be a diagonal matrix, which is often done in practice, Algorithm 1.3 calculates the product $H_k q$ in $O(mn)$ operations and requires $O(mn)$ storage locations.

Several methods for choosing the initial Hessian approximation H_k^0 in the LBFGS algorithm (1.4) have been proposed. We require this approximation to be symmetric and positive definite so that the product $-H_k \nabla f(x_k)$ is a descent direction. One popular choice, given in [25], is $H_k^0 = \gamma_k I$ where

$$\gamma_k = \frac{s_{k-1}^T y_{k-1}}{y_k^T y_k}. \tag{1.17}$$

This scaling factor γ_k can be interpreted as an estimate to one of the eigenvalues of $\nabla^2 f(x_k)^{-1}$.

The following theorem gives conditions under which the BFGS algorithm converges. The proof, found in [25] for example, relies on Zoutendijk's theorem.

Theorem 1.30. *Let $f \in \mathcal{C}^2$ and assume the hypotheses for Theorem 1.22 are satisfied and the initial quasi-Newton matrix B_0 is symmetric positive definite. Additionally, assume that the level set $\mathcal{L}_{x_0} = \{x \in \mathbb{R}^n | f(x) \leq f(x_0)\}$ is convex and that there exists constants m, and M such that*

$$m||w||^2 \leq w^T \nabla^2 f(x) w \leq M||w||^2$$

for all $w \in \mathbb{R}^n$ *and* $x \in \mathcal{L}_{x_0}$. *Then the sequence of iterates* $\{x_k\}$ *generated by Algorithm 1.2 has the property that* $\liminf_{k\to\infty} ||\nabla f(x_k)|| = 0$. *Additionally, if* f *is strongly convex, then* $\lim_{k\to\infty} x_k = x^*$, *where* x^* *is the unique minimizer of* f.

Notice that this theorem does not prove global convergence of the BFGS algorithm for general nonlinear functions as it relies on the convexity of $\mathcal{L}_{x_0}$. The following theorem by Absil et al. offers a separate set of conditions under which line search algorithms (including BFGS) converge to a local minimizer [1]. This is a global result that is independent of convexity, but relies on the objective function being real analytic.

Theorem 1.31. *Consider the line search Algorithm 1.2 and assume that it terminates at iteration* k *if* $\nabla f(x_k) = 0$. *For each* $k \geq 0$, *assume that*

$$c_k := -\frac{p_k^T \nabla f(x_k)}{||p_k|| \cdot ||\nabla f(x_k)||} \geq \delta > 0$$

where $\delta > 0$ *is some positive constant. Lastly, assume the Armijo condition for stepsize selection is implemented.*

i. If the objective function f *is real analytic, then either* $\lim_{k\to\infty} ||x_k|| = +\infty$, *or there exists a unique point* x^* *such that*

$$\lim_{k\to\infty} x_k = x^*.$$

ii. If the objective function f *is real analytic and the curvature condition for step length selection is implemented then either* $\lim_{k\to\infty} ||x_k|| = +\infty$, *or there exists a*

unique point x^* such that

$$\lim_{k\to\infty} x_k = x^*$$

and $\nabla f(x^*) = 0$. That is, if the sequence of iterates $\{x_k\}$ converges, it must converge to a equilibrium point of the gradient system.

Furthermore, if $\mathcal{L}_{x_0} = \{x \in \mathbb{R}^n | f(x) \leq f(x_0)\}$ is compact, the sequence of iterates in Theorem 1.31 will not diverge. The following theorem, by Absil and Kurdyka gives a more general relationship between the stability of equilibrium points of the gradient system and corresponding local minimizers [2].

Theorem 1.32. *If f is real analytic in a neighborhood of x^*, then x^* is a stable equilibrium point of the gradient system if and only if x^* is a local minimizer of f.*

The requirement that f be real analytic cannot be weakened to $f \in \mathcal{C}^\infty$. Absil and Kurdyka, in addition to proving Theorem 1.32, give an example of a smooth function which posses a local minimizer which is not a stable equilibrium point of the gradient system and vice versa.

The next theorem concerns the convergence rate of the BFGS algorithm. Its proof can be found in [25].

Theorem 1.33. *Let $f \in \mathcal{C}^2$ and assume that the sequence generated by Algorithm 1.2 converges to a local minimizer x^* at which the Hessian matrix $H(x)$ is Lipschitz continuous. That is, there exists a constants $L > 0$ and $\delta > 0$ so that*

$$||H(x) - H(x^*)|| \leq L||x - x^*||$$

for all $||x - x^*|| < \delta$. Then x_k converges to x^* superlinearly.

1.1.3 Runge-Kutta Methods

Runge-Kutta (RK) methods are a class of integration methods that can be used to numerically solve systems of ordinary differential equations,

$$\dot{x} = F(t, x).$$

General s-stage Runge-Kutta methods take the form

$$\begin{aligned} X_i &= x_0 + h \sum_{j=1}^{s} a_{ij} F(t_0 + c_j h, X_j), \qquad i = 1, \ldots, s \\ x_1 &= x_0 + h \sum_{j=1}^{s} b_j F(t_0 + c_j h, X_j) \end{aligned} \tag{1.18}$$

where $F : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$, s is the number of stages, the values X_i are referred to as the *internal stages*, h is the stepsize, and the coefficients $A = \{a_{ij}\}$, $c = \{c_i\}$, and $b = \{b_i\}$ define the RK method. In practice, to avoid round-off errors resulting in subtracting nearly equal quantities, we set $z_i := X_i - x_0$.

$$\begin{aligned} z_i &= h \sum_{j=1}^{s} a_{ij} F(t_0 + c_j h,\, x_0 + z_j), \qquad i = 1, \ldots, s \\ x_1 &= x_0 + \sum_{j=1}^{s} d_j z_j \\ d^T &= b^T A^{-1} \end{aligned} \tag{1.19}$$

When we refer to Runge-Kutta methods, we will refer to system (1.19) unless otherwise stated. For the remainder of this section, we only consider autonomous gradient systems. Therefore, we drop the parameter t from $F(t, z)$; that is, we adopt the notation $F(X_j) = -\nabla f(X_j)$.

If $a_{ij} = 0$ for all $j \geq i$ the RK method is *explicit* and the internal stages can be sequentially computed as the ith stage of z_i is a function of z_j for $j < i$. If $a_{ij} \neq 0$

for some $j \geq i$, the method is *implicit* and to solve (1.19), iterative schemes such as Newton iterations are often used. Since RK methods are defined by the coefficients $\{a_{ij}\}$, $\{c_i\}$, and $\{b_i\}$, it is often the practice to report these values in what is called a Butcher tableau shown below

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array} \tag{1.20}$$

For implicit methods, we seek an approximate solution to the possibly nonlinear system of equations defined by the internal stages of (1.19); that is, we seek an approximate solution to

$$\begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_s \end{pmatrix} = h(A \otimes I) \begin{pmatrix} F(x_0 + z_1) \\ F(x_0 + z_2) \\ \vdots \\ F(x_0 + z_s) \end{pmatrix}. \tag{1.21}$$

Let

$$Z := \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_s \end{pmatrix} \qquad \text{and} \qquad Q(Z) := Z - h(A \otimes I) \begin{pmatrix} F(x_0 + z_1) \\ F(x_0 + z_2) \\ \vdots \\ F(x_0 + z_s) \end{pmatrix}.$$

Therefore, we seek a solution to the system $Q(Z) = 0$. If Z^* is a solution to this system and if $Z^{(j)}$ is an approximate solution, then by Taylor expansion,

$$0 = Q(Z^*) \approx Q(Z^{(j)}) + DQ(Z^{(j)})(Z^* - Z^{(j)}).$$

Solving for Z^*,

$$Z^* \approx Z^{(j)} - DQ(Z^{(j)})^{-1}Q(Z^{(j)}).$$

This leads to Newton's method, an iterative method used to numerically solve the equation $Q(Z) = 0$:

$$Z^{(j+1)} = Z^{(j)} - DQ(Z^{(j)})^{-1}Q(Z^{(j)}). \tag{1.22}$$

Newton's method requires an initial guess $Z^{(0)}$ to start the iteration scheme. Possible choices for $Z^{(0)}$ and stopping criteria for the Newton iterations are discussed in Chapter 2.

Calculating $DQ(Z^{(j)})$ is costly in general as it requires the partial derivatives

$$\frac{\partial F}{\partial x}(x_0 + z_i^{(j)})$$

for $i = 1, \ldots, s$. In practice, these Jacobian evaluations are often replaced with an approximate Jacobian, $\dfrac{\partial F}{\partial x}(x_0)$ so that only one Jacobian is computed. For gradient systems, $F(x) = -\nabla f(x)$, finding a zero of $Q(Z)$ via Newton iterations leads to the iteration scheme

$$(I + hA \otimes J)\Delta Z^{(j)} = -(Z^{(j)} + h(A \otimes I)\hat{F}(Z^{(j)})) \tag{1.23}$$

where $J = \nabla^2 f(x_0)$, $\Delta Z^{(j)} = Z^{(j+1)} - Z^{(j)}$ and

$$\hat{F}(Z^{(j)}) = (\nabla f(x_0 + z_1^{(j)}), \nabla f(x_0 + z_2^{(j)}), \ldots, \nabla f(x_0 + z_s^{(j)}))^T.$$

System (1.23) is of size $sn \times sn$. If the matrix A is diagonalizable, that is, there exists a matrix T such that $T^{-1}AT = \Lambda$ where $\Lambda = \text{diag}(\mu_1, \mu_2, \ldots, \mu_s)$ is a

diagonal matrix with the diagonal entries being the eigenvalues of A, then we can further decouple system (1.23) into s systems of size $n \times n$,

$$\begin{aligned}
&(T^{-1} \otimes I)(I + hA \otimes J)(T \otimes I)(T^{-1} \otimes I)\Delta Z^{(j)} = -(T^{-1} \otimes I)(Z^{(j)} + h(A \otimes I)\hat{F}(Z^{(j)})) \\
&\Rightarrow (I + hT^{-1}AT \otimes J)\Delta W^{(j)} = -(W^{(j)} + h(T^{-1}A \otimes I)\hat{F}(Z^{(j)})) \qquad (1.24) \\
&\Rightarrow (\lambda I + \Lambda \otimes J)\Delta W^{(j)} = -(\lambda W^{(j)} + (T^{-1}A \otimes I)\hat{F}(Z^{(j)}))
\end{aligned}$$

where $\lambda = 1/h$, $\Delta W^{(j)} = (T^{-1} \otimes I)\Delta Z^{(j)}$ and $W^{(j)} = (T^{-1} \otimes I)Z^{(j)}$ [17]. The main advantage of diagonalizing A is the ability to further reduce the cost of solving for the left hand side of (1.24) since

$$\begin{aligned}
(\lambda I + \Lambda \otimes J)^{-1} &= \text{diag}((\lambda I + \mu_1 J)^{-1}, (\lambda I + \mu_2 J)^{-1}, \ldots, (\lambda I + \mu_s J)^{-1}) \\
&= \text{diag}\left(\frac{1}{\mu_1}\left(\frac{\lambda}{\mu_1}I + J\right)^{-1}, \frac{1}{\mu_2}\left(\frac{\lambda}{\mu_1}I + J\right)^{-1}, \ldots, \frac{1}{\mu_s}\left(\frac{\lambda}{\mu_1}I + J\right)^{-1}\right)
\end{aligned}$$

Let

$$\begin{pmatrix} v_1^{(j)} \\ v_2^{(j)} \\ \vdots \\ v_s^{(j)} \end{pmatrix} = (T^{-1}A \otimes I)\hat{F}(Z^{(j)})$$

where $v_i^{(j)} \in \mathbb{R}^n$ for $i = 1, \ldots, s$. Then, (1.24) can be expressed as

$$\Delta W_i^{(j)} = -\frac{1}{\mu_i}\left(\frac{\lambda}{\mu_i}I + J\right)^{-1}\left(\lambda W_i^{(j)} + v_i^{(j)}\right) \qquad (1.25)$$

for $i = 1, \ldots, s$. Thus, we only need to solve s systems of size $n \times n$.

Now we show a connection between line search methods discussed in Section

1.1.1 and RK methods. Consider the implicit Euler method with Butcher tableau

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array} \tag{1.26}$$

Applied to the gradient system, $\dot{x} = -\nabla f(x)$ at the point x_k, the implicit Euler method with stepsize h_k leads to the nonlinear equation $x_{k+1} = x_k - h_k \nabla f(x_{k+1})$. If we use one Newton iteration with initial guess $x_k^{(0)} = x_k$, we arrive at the following,

$$\begin{aligned} x_{k+1} &= x_k - (I + h_k \nabla^2 f(x_k))^{-1}(x_k - x_k + h_k \nabla f(x_k)) \\ &= x_k - (\lambda_k I + \nabla^2 f(x_k))^{-1} \nabla f(x_k) \\ &\approx x_k - M_k^{-1} \nabla f(x_k) \end{aligned} \tag{1.27}$$

where $\lambda_k = 1/h_k$, $M_k = \lambda_k I + B_k \approx \lambda_k I + \nabla^2 f(x_k)$, and $B_k \approx \nabla^2 f(x_k)$ represents a quasi-Newton matrix. Now if we consider a continuous output through x_k and x_{k+1},

$$x_{h_k}(\alpha) = x_k - \alpha M_k^{-1} \nabla f(x_k) \approx x(\alpha h_k),$$

we recover line search algorithms with α acting as the step length and $p_k = -M_k^{-1} \nabla f(x_k)$. That is, we can interpret line search algorithms as the implicit Euler method applied to the gradient system with one Newton iteration where the initial guess is the current iterate. Notice that if $h \to \infty$, $M_k \to B_k$ and the quasi-Newton matrix of quasi-Newton methods is recovered.

The relationship $M_k = \lambda_k I + B_k \approx \lambda_k I + \nabla^2 f(x_k)$ provides a posteriori justifications for the modified Newton's direction of Section 1.1.1 as well as the choice of initial quasi-Newton matrix $H_0 = \lambda_0 I$ of the modified Newton's algorithm; the choice

λ_0 can be chosen to be the inverse of the initial stepsize of the underlying gradient system, $\lambda_0 = 1/h_0$. This is developed further in Section 2.2.

We consider Runge-Kutta and quasi-Newton hybrid algorithms based on approximating the inverse of the modified Hessian of $f(x_k)$ by a quasi-Newton matrix $H_k(\lambda_k)$:

$$H_k(\lambda_k) \approx (\lambda_k I + \nabla^2 f(x_k))^{-1}.$$

For gradient systems and general RK methods, we seek a solution to the s nonlinear systems given in (1.25). Given a current iterate x_k, we can write these systems as

$$\Delta W_{i,k}^{(j)} = -\frac{1}{\mu_i}\left(\frac{\lambda_k}{\mu_i} I + J\right)^{-1} (\lambda_k W_{i,k}^{(j)} + v_{i,k}^{(j)}) \tag{1.28}$$

for $i = 1, \ldots, s$. Notice that in each system, we wish to use $\left(\frac{\lambda_k}{\mu_i} I + J\right)^{-1}$ as an approximation to the modified Hessian $\left(\frac{\lambda_k}{\mu_i} I + \nabla^2 f(x_k)\right)^{-1}$. As with the specific implementation of the implicit Euler method, we replace the matrix $\left(\frac{\lambda_k}{\mu_i} I + J\right)^{-1}$ with a quasi-Newton matrix $H_{i,k}(\lambda_k)$. This idea is the main topic of Chapter 2. We now present some of the basic theorems regarding gradient systems.

Lemma 1.34. *Let $x(t)$ be a solution to the gradient system $\dot{x} = -\nabla f(x)$. Then $f(x(t))$ is a nonincreasing function of t.*

Proof. Taking the derivative of $f(t)$ with respect to t gives,

$$\frac{d}{dt} f(x(t)) = \nabla f(x(t))^T \dot{x}(t) = \nabla f(x(t))^T \left(-\nabla f(x(t))\right) = -||\nabla f(x(t))||^2 \leq 0$$

□

Lemma 1.34 gives justification for following the gradient flow: If the solution to the gradient system is known starting from iterate x_0, then $\lim_{t\to\infty} ||\nabla f(x(t,0,x_0))|| = 0$ where $x(t,0,x_0)$ is the gradient flow. Additionally, if the level set

$$\mathcal{L}_{x_0} = \{x \in \mathbb{R}^n | f(x) \leq f(x_0)\}$$

is compact, then $\lim_{t\to\infty} x(t,0,x_0) = x^*$ where x^* is an equilibrium point of the system. The following theorems, both taken from [25], provide the necessary and sufficient conditions for a point to be a local minimizer. These theorems relate mathematical optimization with the equilibrium points of a corresponding gradient system.

Lemma 1.35. *Let f be continuously differentiable at x^* and let x^* be a local minimizer of f. Then $\nabla f(x^*) = 0$. Additionally, if f is twice continuously differentiable at x^*, then $\nabla^2 f(x^*)$ is positive semi-definite.*

Proof. Assume that $\nabla f(x^*) \neq 0$ and define $p = -\nabla f(x^*)$. Then $p^T \nabla f(x^*) = -||\nabla f(x^*)||^2 < 0$. Then by Lemma 1.14, p is a direction of decrease for f at x^*. That is, x^* is not a local minimizer of f, a contradiction. Thus, $\nabla f(x^*) = 0$. Now assume f is twice continuously differentiable at x^* and assume for contradiction that $\nabla^2 f(x^*)$ is negative definite. Then there exists a vector $p \in \mathbb{R}^n$ so that $p^T \nabla^2 f(x^*) p < 0$. Since $\nabla^2 f$ is continuous at x^*, there is a $T \in \mathbb{R}$ so that

$$p^T \nabla^2 f(x^* + tp) p < 0$$

for all $t \in [0, T)$. Choose $\bar{t} \in (0, T)$. Then using Taylor's theorem,

$$f(x^* + tp) = f(x^*) + tp^T \nabla f(x^*) + \frac{t^2}{2} p^T \nabla^2 f(x^* + \theta \bar{t} p) p$$

for some $\theta \in (0, 1)$. Let $\alpha = \theta\bar{t}$. Then $\alpha \in (0, T)$ and $p^T\nabla^2 f(x^* + \alpha p)p < 0$. Using the fact that $\nabla f(x^*) = 0$, we have $f(x^* + tp) < f(x^*)$. Therefore, p is a direction of decrease for the function f at x^*, which contradicts the hypothesis that x^* is a local minimizer of f. □

This lemma gives the necessary conditions for a local minimizer. The first part of this lemma, called the first-order necessary conditions, assumes f to be continuously differentiable at x^* whereas the second part of the lemma, called the second-order necessary conditions, assumes f to be twice continuously differentiable at x^*. The following lemma is called the second-order sufficient conditions for a local minimizer.

Lemma 1.36. *If f is twice continuously differentiable at x^*, $\nabla f(x^*) = 0$, and $\nabla^2 f(x^*)$ is positive definite then x^* is an isolated local minimizer of f.*

Proof. By Taylor's theorem we have for some $t \in (0, 1)$

$$f(x^* + p) = f(x^*) + p^T\nabla f(x^*) + \frac{1}{2}p^T\nabla^2 f(x^* + tp)p$$

for all $p \in \mathbb{R}^n$. By continuity of $\nabla^2 f$ at x^*, $\nabla^2 f(x)$ remains positive definite in a neighborhood of x^*; that is, there exists $\delta > 0$ so that if $p \in \mathbb{R}^n$ is a nonzero vector such that $||p|| < \delta$ then $p^T\nabla^2 f(x^* + tp)p > 0$ for $t \in [0, 1)$. Since $\nabla f(x^*) = 0$ by hypothesis, we have $f(x^* + p) < f(x^*)$ for all $x \neq x^*$ such that $||x - x^*|| < \delta$. Hence, x^* is a local minimizer.

Now we show that x^* is an isolated local minimizer. Again using Taylor's theorem and the hypothesis $\nabla f(x^*) = 0$, for any $||p|| < \delta$,

$$\nabla f(x^* + p) = \nabla^2 f(x^* + tp)^T p$$

for some $t \in (0, 1)$. Therefore $p^T \nabla f(x^* + p) = p^T \nabla^2 f(x^* + tp)^T p > 0$ which implies that we cannot have $\nabla f(x^* + p) = 0$. Hence, x^* is the unique local minimizer in the neighborhood $B(x^*, \delta)$. □

Lastly, we present some terminology and basic theorems regarding numerical algorithms for solving ODEs. Given a rooted tree diagram (see [18] for more information on trees), we can define the following:

Definition 1.37. Given a tree T, the *order* of T, denoted $\rho(T)$, is the number of nodes of T.

Definition 1.38. Given a tree T, $\gamma(T)$ is defined recursively as follows: Define $\gamma(\bullet) = 1$ and

$$\gamma(T) = \rho(T)\gamma(T_1) \cdots \gamma(T_m)$$

where $T_1, \ldots, T_m$ are the m subtrees generated by removing the root of T and all branches connected to this root.

Definition 1.39. Given a tree T and a Butcher tableau corresponding to some Runge-Kutta method, $\Phi_j(T)$ is defined recursively as follows: Define $\Phi_j(\bullet) = 1$ and

$$\Phi_j(T) = \sum_{k_1=1}^{s} a_{jk_1} \Phi_{k_1}(T_1) \cdots \sum_{k_m=1}^{s} a_{jk_m} \Phi_{k_m}(T_m)$$

where $T_1, \ldots, T_m$ are the m subtrees generated by removing the root of T and all branches connected to this root.

Definition 1.40. Given a s-stage implicit Runge-Kutta method, the *stability function*

for that method is

$$R(z) = 1 + zb^T(I - zA)^{-1}\mathbb{1}$$

where $\mathbb{1} = (1, 1, \ldots, 1)^T$ is a vector of length s.

The following theorem gives conditions on the coefficients c_i, b_i, and a_{ij} under which a Runge-Kutta method is of order p. The proof can be found in [18].

Theorem 1.41. *A Runge-Kutta method is of order p if and only if*

$$\sum_{j=1}^{s} b_j \Phi_j(t) = \frac{1}{\gamma(t)}$$

for all trees with $\rho(T) \leq p$.

Definition 1.42. A Runge-Kutta method is *A-stable* if and only if

$$|R(z)| \leq 1.$$

Definition 1.43. A Runge-Kutta method is *L-stable* if and only if it is A-stable and

$$\lim_{z \to \infty} R(z) = 0.$$

Definition 1.44. A s-stage Runge-Kutta method satifying

$$a_{sj} = b_j$$

for all $j = 1, \ldots, s$ is called *stiffly accurate*.

Definition 1.45. The *stage order* of a Runge-Kutta method is defined as the minimum of p and q where p is the order of the Runge-Kutta method and q is the largest

natural number such that

$$\sum_{j=1}^{s} a_{ij} c_j^{k-1} = \frac{c_i^k}{k}$$

holds for all $k = 1, \ldots, q$ and all $i = 1, \ldots, s$.

1.1.4 Hybrid Algorithms

Although research linking nonlinear optimization and numerical ODE methods exists, there is a comparatively small amount of literature on the topic. Most of the existing research is based on numerically approximating the local solution to the gradient system, but doing so in a fashion that avoids the costs associated with ODE implicit solvers.

Necessary and sufficient conditions for a local minimizer of an objective function to be a stable equilibrium point for the corresponding gradient system are shown in [2] and conditions under which line search methods converge to a fixed point of the gradient system are shown in [1].

Hybrid algorithms based on the linearization of the gradient system at the current iterate x_k are discussed in [9]. One particular algorithm, QUABOT, uses a BFGS matrix to approximate inverse Hessians and is shown to be competitive with standard line search algorithms. Quasi-Newton algorithms have also been developed that implement a new quasi-Newton equation arising from attempting to minimize the function

$$f_k(x) = f(x) + \frac{1}{2}(x - x_k)^T A_k (x_k - x)$$

at each iteration instead of minimizing $f(x)$ [32]. The choice of symmetric, positive definite matrix A_k determines the quasi-Newton equation.

Hybrid algorithms using ODE and trust region techniques are discussed in [22] and [33]. The former uses a trust region framework but computes the step length using a Rosenbrock implicit ODE method. Other algorithms that use elements from nonlinear optimization and numerical ODEs include [3, 5, 8].

The goal of this thesis is to combine certain Runge-Kutta implicit ODE methods with line search techniques to create a class of hybrid algorithms that converges to an equilibrium point of the gradient system while avoiding the cost of calculating Hessian information. This is possible by using the Runge-Kutta framework for solving the gradient system, but also using a quasi-Newton-like matrix that takes the place of the modified Hessian matrix.

1.2 Overview

In Chapter 2, we develop two hybrid algorithms for unconstrained nonlinear optimization, one based on the implicit Euler method for ODEs and the other based on a second order Runge-Kutta algorithm. We discuss the motivation behind these hybrid algorithms and then go into detail regarding each subroutine: finding line or curve search information by approximately solving the gradient flow, performing a line or curve search using either the Wolfe or Goldstein and Price conditions, and generating a stepsize for the following iteration by a stepsize control algorithm.

Chapter 3 compares the performance of the hybrid algorithms with standard

algorithms such as the LBFGS method. Chapter 4 summarizes this thesis and presents some directions for future work in the area of hybrid algorithms.

CHAPTER 2
HYBRID ALGORITHMS

2.1 Unconstrained Algorithms

The BFGS algorithm satisfies the secant and curvature conditions, $B_{k+1}s_k = y_k$ and $s_k^T y_k > 0$ respectively. Expressing the secant condition in terms of $H_{k+1} = B_{k+1}^{-1}$, we have

$$H_{k+1}y_k = s_k \tag{2.1}$$

$$s_k^T y_k > 0. \tag{2.2}$$

The relationship between the approximate dense output implicit Euler method and the modified Newton's algorithm of line search methods was shown in Section 1.1.3. In particular, the matrix $(\lambda_k I + J_k)^{-1}$, where $J_k = \nabla^2 f(x_k)$, appears in both algorithms but the parameter λ_k represents a positive definite controlling factor in Newton's algorithm whereas it is related to the stepsize for the numerical integration of the gradient system in the other. In general, the matrix $(\frac{\lambda_k}{\mu_i} I + J_k)^{-1}$, as shown in equation (1.28), appears when Newton iterations are used to solve the nonlinear system of equations in any Runge-Kutta algorithm. As stated in Section 1.1.3, we are not interested in approximating $\nabla^2 f(x_k)^{-1}$ with a matrix H_k, but rather in approximating $(\frac{\lambda_k}{\mu_i} I + \nabla^2 f(x_k))^{-1}$ with a quasi-Newton-like matrix. Therefore, from this point on, let

$$H_{i,\,k}(\lambda) \approx \left(\frac{\lambda}{\mu_i} I + \nabla^2 f(x_k) \right)^{-1}. \tag{2.3}$$

Since $H_{i,k}(\lambda) = B_{i,k}(\lambda)^{-1}$ does not approximate the Hessian but rather a modified Hessian, it is natural to consider the new secant-type condition,

$$B_{i,\,k+1}(\lambda_{k+1})s_k = \frac{\lambda_{k+1}}{\mu_i}s_k + y_k$$

coming from

$$\begin{aligned} B_{i,\,k+1}(\lambda_{k+1})s_k &\approx \left(\frac{\lambda_{k+1}}{\mu_i}I + \nabla^2 f(x_{k+1})\right)s_k \\ &\approx \frac{\lambda_{k+1}}{\mu_i}s_k + \hat{B}_{k+1}s_k \\ &= \frac{\lambda_{k+1}}{\mu_i}s_k + y_k \end{aligned}$$

where $\hat{B}_{k+1}$ is the standard quasi-Newton matrix introduced in Chapter 1 which satisfies the secant equation, $B_{k+1}s_k = y_k$ which is independent of i. Therefore, we obtain s quasi-Newton equations per step. Written in terms of $H_{i,k+1}(\lambda) := B^{-1}_{i,k+1}(\lambda)$, these s conditions are:

$$H_{i,\,k+1}(\lambda_{k+1})Y_{i,k}(\lambda_{k+1}) = s_k \tag{2.4}$$

where $Y_{i,j}(\lambda) := \dfrac{\lambda}{\mu_i}s_j + y_j$ and $i = 1, \ldots, s$. Under the definition of $Y_{i,j}(\lambda)$, the curvature condition can also be satisfied.

Lemma 2.1. *If $s_k^T y_k > 0$, $\lambda_{k+1} > 0$, and $\mu_i > 0$ for all $i = 1, \ldots, s$, then*

$$s_k^T Y_{i,k}(\lambda_{k+1}) > 0.$$

Proof.

$$s_k^T Y_{i,k}(\lambda_{k+1}) = \frac{\lambda_{k+1}}{\mu_i}s_k^T s_k + s_k^T y_k = \frac{\lambda_{k+1}}{\mu_i}||s_k||^2 + s_k^T y_k > 0. \tag{2.5}$$

□

Although the new secant condition (2.4) and the new curvature condition (2.5) can be used to define hybrid algorithms for any algorithm in the Broyden class, we will only consider the BFGS algorithm. The update for the matrix $B_{i,k}(\lambda)$ is given by

$$B_{i,\,k+1}(\lambda) = B_{i,k}(\lambda) - \frac{B_{i,k}(\lambda)s_k s_k^T B_{i,k}(\lambda)}{s_k^T B_{i,k}(\lambda) s_k} + \frac{Y_{i,k}(\lambda)Y_{i,k}(\lambda)^T}{s_k^T Y_{i,k}(\lambda)}. \tag{2.6}$$

Using the Sherman-Morrison-Woodbury Formula 1.25, we can express this update in terms of the matrix $H_{i,k+1}(\lambda)$:

$$H_{i,\,k+1}(\lambda) = V_k^T H_{i,\,k}(\lambda) V_k + \rho_k s_k s_k^T \tag{2.7}$$

where

$$\rho_k = \frac{1}{Y_{i,k}(\lambda)^T s_k}, \qquad V_k = (I - \rho_k Y_{i,k}(\lambda) s_k^T).$$

Since each $Y_{i,j}(\lambda_{k+1})$ depends on the *current* inverse stepsize λ_{k+1}, which can change between iterations of the hybrid algorithm, it is not practical to store the most recent m vectors $Y_{i,j}(\lambda_{k+1})$, in a limited memory scheme. Instead, we store the vectors $s_{k-m+1}, \ldots, s_k, y_{k-m+1}, \ldots, y_k$ and compute the $Y_{i,j}(\lambda_{k+1})$ using the formula,

$$Y_{i,j}(\lambda_{k+1}) = \frac{\lambda_{k+1}}{\mu_i} s_j + y_j. \tag{2.8}$$

The following corollary of Lemma 1.15 gives conditions on the stepsize of the differential equation so that the matrix $\left(\dfrac{\lambda_k}{\mu_j} I + \nabla^2 f\right)$ remains positive definite.

Corollary 2.2. *Let $\beta_1, \beta_2, \ldots, \beta_n$ be the eigenvalues of a symmetric positive definite matrix $Q \in \mathbb{R}^{n\times n}$, let $v_1, v_2, \ldots, v_n$ be the corresponding eigenvectors, and let $\mu_1, \mu_2, \ldots, \mu_s$ be positive real eigenvalues of the Runge-Kutta coefficient matrix A. Then $\lambda > -\min_{i=1,\ldots,n} \beta_i\mu$ if and only if $\left(\frac{\lambda}{\mu_j}I + Q\right)$ is positive definite for each $j = 1, \ldots, s$ where $\mu = \max_{j=1,\ldots,s} \mu_j$.*

Proof. Replace the λ in Lemma 1.15 with λ/μ. □

Using Corollary 2.2 and equation (2.3), we see that if the stepsize h_k is sufficiently small, then $H_{i,\,k+1}(\lambda)$ is positive definite. Additionally, when x_k is close to a local minimizer, $\nabla^2 f(x_k)$ is positive definite so the stepsize h_k is allowed to be large (λ_k small). Following the proof of Lemma 1.29, $H_{i,\,k+1}(\lambda)$ is symmetric when $H_{i,\,k}(\lambda)$ is symmetric.

As with the BFGS algorithm, we can apply the LBFGS two loop recursion Algorithm 1.3 to calculate $H_{i,k}(\lambda)q$ for any $q \in \mathbb{R}^n$. Notice that in this calculation, i is fixed and corresponds to the i^{th} eigenvalue.

For the initial Hessian $H^0_{i,k}(\lambda_k)$ in Algorithm (2.1), a reasonable choice is $H^0_{i,k}(\lambda_{k+1}) = \gamma_{i,k}(\lambda_{k+1})I$ where

$$\gamma_{i,k}(\lambda_{k+1}) = \frac{s_k^T Y_{i,k}(\lambda_{k+1})}{Y_{i,k}(\lambda_{k+1})^T Y_{i,k}(\lambda_{k+1})}. \tag{2.9}$$

This choice of $\gamma_{i,k}(\lambda)$ satisfies $\lim_{h\to 0} H^0_{i,k}(\lambda) = 0$, the zero matrix, and $\lim_{h\to\infty} H^0_{i,k}(\lambda) = \frac{s_k^T y_k}{y_k^T y_k} I$, which is the proposed initial matrix for the LBFGS algorithm (1.17).

The following sections detail two examples of hybrid algorithms: one based on the implicit Euler method for solving the gradient system and the second based on a

Algorithm 2.1 LBFGS Hybrid Two Loop Recursion

Inputs: Limited memory vectors $\{s_j\}$ and $\{y_j\}$, eigenvalue μ_i, λ_{k+1}, m, k, q,

and initial Hessian $H^0_{i,k}(\lambda_k)$

for $j = k-1, k-2, \ldots, k-m$

Calculate $Y_{i,j}(\lambda_{k+1})$ using (2.8)

$\rho_j \leftarrow \dfrac{1}{Y_{i,j}(\lambda_{k+1})^T s_j}$

$\alpha_j \leftarrow \rho_j s_j^T q$

$q \leftarrow q - \alpha_j Y_{i,j}(\lambda_{k+1})$

end

$r \leftarrow H^0_{i,k}(\lambda_{k+1})q$

for $j = k-m, k-m+1, \ldots, k-1$

$\beta \leftarrow \rho_j Y_{i,j}(\lambda_{k+1})^T r$

$r \leftarrow r + s_j(\alpha_j - \beta)$

end

Output: $r := H_{i,k}(\lambda_k)q$

specific implicit Runge-Kutta algorithm of order two.

2.2 Hybrid Algorithm of Order One

In this section, the outline of the hybrid method based on the implicit Euler algorithm from numerical quadrature is presented. The Butcher tableau for the

implicit Euler method is

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array} \tag{2.10}$$

Using Theorem 1.41, the implicit Euler algorithm satisfies

$$\Phi(\bullet) = 1 = \frac{1}{\gamma(\bullet)}$$
$$\Phi(\begin{smallmatrix}\bullet\\ \bullet\end{smallmatrix}) = 1 \neq \frac{1}{2} = \frac{1}{\gamma(\begin{smallmatrix}\bullet\\ \bullet\end{smallmatrix})}$$

and is therefore an order one method. The tableau (2.10) leads to the nonlinear system

$$x_{k+1} = x_k - h_k \nabla f(x_{k+1}) \tag{2.11}$$

where h_k is the stepsize of the differential equation. In a numerical ODE solver, this system is approximately solved at each iteration to find the next iterate. For the hybrid algorithm of order one, a numerical solution to (2.11) is quickly calculated and used as a suitable step direction for a line search. Notice that the coefficient matrix of (2.10) has only one eigenvalue, $\mu_1 = 1$. As such, for the remainder of this section, we adopt the following simpler notation:

$$H_k(\lambda) := H_{i,k}(\lambda), \qquad B_k(\lambda) := B_{i,k}(\lambda), \qquad \text{and} \qquad Y_k(\lambda) := Y_{i,k}(\lambda).$$

If $H_k(\lambda_k)$ is used to approximate $(\lambda_k I + \nabla^2 f(x_k))^{-1}$, the modified Newton iterations are,

$$\begin{aligned} x_{k+1}^{(j+1)} &= x_{k+1}^{(j)} - (\lambda_k I + \nabla^2 f(x_{k+1}^{(j)}))^{-1}(\lambda_k(x_{k+1}^{(j)} - x_k) + \nabla f(x_{k+1}^{(j)})) \\ &\approx x_{k+1}^{(j)} - H_k(\lambda_k) \cdot (\lambda_k(x_{k+1}^{(j)} - x_k) + \nabla f(x_{k+1}^{(j)})). \end{aligned} \tag{2.12}$$

The hybrid algorithm that we develop is similar in nature to quasi-Newton line search algorithms. The search direction p_k is chosen to be a descent direction and once selected, a line search in the direction of p_k from x_k is performed to find an suitable step length which satisfies either the Wolfe of the Goldstein and Price conditions. We will see that the line search in the direction of p_k is equivalent to the dense output approximation of the solution to the gradient system at x_k. Lastly, the quasi-Newton vectors s_k and y_k are updated along with the quasi-Newton matrix $H_k(\lambda)$. Unlike quasi-Newton algorithms, the search direction is given by the approximate solution to the gradient system at the current iterate. The matrix $H_k(\lambda)$ and the vectors $Y_j(\lambda) = \lambda s_j + y_j$, $j = 1, \ldots, m$ are functions of the stepsize.

Now we state an immediate result of (2.1).

Corollary 2.3. *Consider the univariate function $m_k(\alpha) = x_k - \alpha H_k(\lambda)\nabla f(x_k)$. If the Wolfe conditions are used to find an acceptable step length, then the updating scheme (2.7) satisfies the curvature condition $s_k^T Y_k(\lambda_{k+1}) > 0$.*

Proof. From Lemma 2.1, the curvature condition $s_k^T Y_k(\lambda_{k+1}) > 0$ is satisfied whenever the standard BFGS curvature condition $s_k^T y_k > 0$ is satisfied. From Lemma 1.28, we know that if the Wolfe conditions are used on the function $m_k(\alpha)$, then $s_k^T y_k > 0$ holds. □

Now we present the hybrid algorithm of order one as shown in Algorithm 2.2. The details regarding the choice of search direction p_k, the selection of α_k via line search, and the stepsize $h_k = 1/\lambda_k$ are discussed in the following sections.

Algorithm 2.2 Hybrid Algorithm of Order 1

Input: starting iterate x_0, termination tolerance $\epsilon > 0$,

initial inverse stepsize λ_0, and initial matrix $H_0(\lambda_0)$

Initialize $k \leftarrow 0$

while $||\nabla f(x_k)|| > \epsilon$

Compute search direction p_k via Algorithm 2.3

Find step length α_k via Algorithm 1.1 applied to (2.17)

$x_{k+1} \leftarrow x_k + \alpha_k p_k$

$s_k \leftarrow x_{k+1} - x_k$, and $y_k \leftarrow \nabla f(x_{k+1}) - \nabla f(x_k)$

Update h_k via equation (2.22)

$\lambda_{k+1} \leftarrow 1/h_{k+1}$, and $k \leftarrow k + 1$

end

Output: x_k

2.2.1 Search Direction Selection

The nonlinear system of equations (2.11) is solved using the modified Newton iteration scheme given in (2.12). From Lemma 1.34, stationary points of the gradient system occur when the solution at x_0, $x(t, 0, x_0)$, is evaluated at $t = \infty$. Therefore, it is not critically important that (2.11) is solved exactly. Instead, only a small number of Newton iterations are performed at each step. If one Newton iteration is executed,

(2.12) gives

$$x_{k+1}^{(1)} = x_{k+1}^{(0)} - (\lambda_k I + \nabla^2 f(x_{k+1}^{(0)}))^{-1}(\lambda_k(x_{k+1}^{(0)} - x_k) + \nabla f(x_{k+1}^{(0)})). \tag{2.13}$$

The search direction p_k is found by $p_k = x_{k+1}^{(1)} - x_k$. Since this search direction is then used in a line search to find an acceptable step length, p_k must be a descent direction for the function f at x_k; that is, the initial Newton guess, $x_{k+1}^{(0)}$ must be chosen so that $p_k^T \nabla f(x_k) < 0$. One choice is

$$x_{k+1}^{(0)} = x_k. \tag{2.14}$$

Using (2.13), this choice leads to the search direction

$$p_k(\lambda_k) = -H_k(\lambda_k)\nabla f(x_k). \tag{2.15}$$

Since $H_k(\lambda_k)$ is positive definite, $p_k(\lambda_k)$ is a descent direction since

$$p_k(\lambda_k)^T \nabla f(x_k) = -\nabla f(x_k) H_k(\lambda_k) \nabla f(x_k) < 0.$$

Therefore, when only performing one Newton iteration, the choice (2.14) guarantees a decrease in the objective function from x_k in the direction of $p_k(\lambda_k)$. What makes this choice particularly attractive is that it does not require the any additional gradient evaluations apart from $\nabla f(x_k)$, which is computed when evaluating the Wolfe or the Goldstein and Price conditions.

The general search direction algorithm is given in Algorithm 2.3. In practice, we use (2.14) to find an acceptable direction since it is carries a trivial amount of cost and does not require checking if the resulting vector $p_k(\lambda_k)$ is a descent direction.

If $x_{k+1}^{(0)}$ is predicted using previous information, additional gradient evaluations may be needed. Additionally, it may be necessary to perform multiple Newton iterations before a descent direction is found. If this is the case, these added computations can be used to produce additional limited memory vectors s_j and y_j; that is, if the gradient is evaluated at a point $\hat{x}$ during step k, we can use the vectors $\hat{x} - x_k$ and $\nabla f(\hat{x}) - \nabla f(x_k)$ as new limited memory vectors s_j and y_j respectively.

Algorithm 2.3 Order One Search Direction Algorithm

Input: iterate x_k, Newton guess $x_{k+1}^{(0)}$

inverse stepsize $\lambda_k > 0$, limited memory vectors $\{s_i\}$ and $\{y_i\}$,

Initialize $descent\_number \leftarrow 0$;

while $descent\_number \leq 0$

Compute $x_{k+1}^{(j+1)}$ using (2.13) and Algorithm 2.1

$p_k \leftarrow x_{k+1}^{(j+1)} - x_k$

$descent\_number \leftarrow p_k^T \nabla f(x_k)$

end

Output: p_k

In an ODE solver, a good approximation to the solution of the Newton system (1.21) is desired. Although we are not concerned with finding an accurate solution to this nonlinear system, it is desirable to have the Newton iterations of the hybrid

algorithm produce better approximations if more iterations are performed. Table 2.1 shows that often times, this the case. The triple appearing in each entry of the table represents the number of iterations, function evaluations, and gradient evaluations respectively to achieve convergence.

Table 2.1: Effect of Newton Iterations on Convergence (Algorithm 2.2)

Problem	One Iteration	Two Iterations	Ten Iterations
BIGGS	80/125/125	66/97/162	47/66/480
DIAG10	20/23/23	13/16/21	11/14/104
EXTWD500	76/93/93	55/77/131	26/39/264
PENAL5000	162/439/439	153/442/595	135/406/1618
POWER100	813/850/850	357/374/730	176/225/1800
PQUAD1000	298/318/318	166/181/346	72/94/733
RAYA5000	594/623/623	322/340/661	57/86/590
ROSENB250	633/650/650	550/601/816	760/2031/2824
VARDM100	41/85/85	30/74/103	23/67/265
ZAKHAR1000	142/411/411	122/392/513	114/383/1400

Results: iterations/function evaluations/gradient evaluations

The hybrid algorithm was tested on ten test problems with convergence criteria $||\nabla f(x_k)|| < 10^{-6}$ and $m = 6$ limited memory vectors. For each test function,

the number of Newton iterations is fixed at either one, two, or ten. The numbers reported in each cell of the table represent the number of iterates, function evaluations, and gradient evaluations respectively. These results suggest that increasing the number of Newton iterations performed at each iteration of the algorithm decreases the number of steps needed to achieve convergence, but increases the number of gradient evaluations needed. Hence, we prefer few Newton iterations (one or at most two).

2.2.2 Line Search

In Chapter 1.1.3, a connection between line search methods and the implicit Euler method for numerical integration was shown. In this section, we further develop this connection.

The general Runge-Kutta method represented by the Butcher tableau (1.20) gives a numerical approximation to the actual solution at $x(t_0 + h)$. Dense output Runge-Kutta methods provide approximations to $x(t_0 + \theta h)$ for any $\theta \in \mathbb{R}$, not just $[0, 1]$. This is done by considering Runge-Kutta methods of the form

$$\begin{aligned} z_i &= h \sum_{j=1}^{s} a_{ij} h(t_0 + c_j h,\, x_0 + z_j), \qquad i = 1, \ldots, s \\ x_1(\theta) &= x_0 + \sum_{i=1}^{s} d_i(\theta) z_i \\ d(\theta)^T &= b(\theta)^T A^{-1}. \end{aligned} \tag{2.16}$$

The coefficients $b_i(\theta)$ are polynomials in θ to be determined. We impose that $b(\theta)$ should provide a local error of order $\mathcal{O}(h^{p^*+1})$; that is,

$$x_1(\theta) - x(t_0 + \theta h) = \mathcal{O}(h^{p^*+1}).$$

The following theorem (found in [18]) offers conditions on θ under which the dense output approximation is of order p^*.

Theorem 2.4. *The dense output Runge-Kutta method given by (2.16) is of order p^* if and only if*

$$\sum_{i=1}^{s} b_i(\theta)\Phi_i(t) = \frac{\theta^{\rho(t)}}{\gamma(t)}, \qquad \textit{for all trees } t \textit{ with } \rho(t) \leq p^*.$$

For the implicit Euler method, we find a suitable search direction, p_k, by using Newton iterations to approximate the solution to system (2.11) as described in Section 2.2.1. After finding a suitable p_k,

$$x_{k+1}(\theta) = x_k - h_k b_1(\theta)\nabla f(x_{k+1}) \approx x_k + h_k b_1(\theta)p_k$$

and, by using Theorem 2.4, this approximation will be order $p^* = 1$ if

$$b_1(\theta) = \frac{\theta^{\rho(\bullet)}}{\gamma(\bullet)\Phi(\bullet)} = \frac{\theta}{1 \cdot 1} = \theta.$$

Therefore, the the hybrid algorithm of order one will perform the line search on the univariate function,

$$m_k(\alpha) = x_k + \alpha p_k. \tag{2.17}$$

where $\alpha_k := h_k\theta_k$. This line search, which is equivalent to the standard line search algorithm, can be performed with either the Wolfe or the Goldstein and Price conditions. For the initial value of α in the line search, we take $\theta_k = 1$ (corresponding to $x_k + h_k p_k$) with the hope that the stepsize h_k produces a reasonable approximation p_k to the Newton system. Once an acceptable value α_k has been chosen at iteration

k, we set

$$x_{k+1} = x_k + \alpha_k p_k$$

and proceed to update the value of h_k and the limited memory vectors $\{s_i\}$ and $\{y_i\}$.

In Chapter 3, Algorithm 2.2 is compared to the LBFGS Algorithm 1.4. On difficult test problems where the minimizer is found in a large shallow basin, many line search methods have trouble finding a point which satisfies the Wolfe or Goldstein and Price conditions. We impose the following safeguard: if the Wolfe or the Goldstein and Price conditions spend too long searching for an acceptable point (for example, placing an upper bound on the number of iterations within the line search), the line search is terminated and the current iteration is repeated with a higher number of Newton iterations with no line search. In effect, this switches the hybrid algorithm to a numerical integration routine.

This safeguard is very natural. In Table 2.1, it was shown that for the majority of the test problems, the Newton iterations appear to converge as the number of Newton iterations increased. This convergence, along with the increased number of Newton iterations with no line search allows the hybrid algorithm to function more as a numerical integration routine. Note that unlike numerical integration algorithms, we still use the limited memory vectors to approximate the Hessian-vector products appearing in the Newton system so that no Hessian evaluations are required.

When the line search is dropped, a criterion is needed to terminate the Newton iterations. If the Newton iterations converge, then speed of convergence is linear and

using the notation presented in Section 1.1.3,

$$||\Delta Z^{(j+1)}|| \leq \Theta ||\Delta Z^{(j)}|| \tag{2.18}$$

where Θ is a nonnegative number. If $0 < \Theta < 1$, then $||\Delta Z^{(j)}|| \to 0$ as $j \to \infty$ and if Z^* is the exact solution of the Newton system, this implies that $Z^{(j)} \to Z^*$ as $j \to \infty$. Using this, and (2.18),

$$\begin{aligned} ||Z^{(j+1)} - Z^*|| &\leq \sum_{i=1}^{\infty} ||Z^{(j+i)} - Z^{(j+i+1)}|| \leq \sum_{i=1}^{\infty} ||\Delta Z^{(j+i)}|| \leq ||\Delta Z^{(j)}|| \sum_{i=0}^{\infty} \Theta^i \\ &\leq \frac{\Theta}{1-\Theta} ||\Delta Z^{(j)}||. \end{aligned} \tag{2.19}$$

Therefore, we can bound the error at each iteration using (2.19). Additionally, formula (2.18) can be used to provide estimates of Θ; that is,

$$\Theta_j := \frac{||\Delta Z^{(j)}||}{||\Delta Z^{(j-1)}||}.$$

Given a tolerance for the Newton iterations, tol_N, we terminate the Newton iterations at iteration j if the following holds

$$\frac{\Theta_j}{1-\Theta_j} ||\Delta Z^{(j)}|| \leq tol_N. \tag{2.20}$$

Notice that estimating Θ_j requires at least two iterations to be performed so if the line search is temporarily stopped, we perform at least two Newton iterations per step. After a set number of iterations without implementing a line search, we can revert the algorithm back to its original state by once again performing a line search and only one Newton iteration with initial choice given by (2.14). If problems once again occur, the line search is temporarily dropped again.

2.2.3 Stepsize Control

The stepsize h_k associated with numerically solving the gradient system needs to be updated at each iteration of the algorithm. As we will see in Section 2.2.4, the property

$$\sum_{k=0}^{\infty} \lambda_k < \infty \tag{2.21}$$

where $\lambda_k = 1/h_k$ is not needed for convergence of the hybrid algorithm of order one, but it is a sufficient condition for superlinear convergence. Notice that since $\lambda_k > 0$ for all k, (2.21) implies $\lambda_k \to 0$ as $k \to \infty$. As such, it is necessary for superlinear convergence that the stepsizes satisfy $h_k \to \infty$ as x_k approaches x^*; that is, the hybrid algorithm of order one behaves in a similar fashion to the BFGS algorithm near a local minimizer.

One possibility is to start with a large value for λ_0 and decrease λ_k in a geometric fashion; for example, $\lambda_{k+1} = a\lambda_k$ for some constant $a \in (0, 1)$. In this case, the resulting algorithm will converge to an equilibrium point and if this point is an isolated local minimizer of $f(x)$, the convergence is superlinear (see Section 2.2.4). However, this choice often decreases λ_k too quickly and the resulting algorithm performs in a similar fashion to the BFGS algorithm. Instead, we control the stepsize using the equation

$$h_{k+1} = \frac{c}{||\nabla f(x_k)||} \tag{2.22}$$

where c is a scalar. This update formula is cheap as this norm has already been calculated to check the first order necessary condition for termination. It also has the

property that if $x_k \to x^*$ as $k \to \infty$, then $h_k \to \infty$. In Section 2.3, more accurate approximations of the local error are considered based on the underlying IRK method having order two.

2.2.4 Convergence of the Order One Method

For the Algorithm 2.2, which is based on the implicit Euler method, when $\lambda_{k+1} = 0$ ($h_{k+1} = +\infty$), we recover the original BFGS formula since $Y_k(0) = y_k$. Therefore, this hybrid algorithm can be viewed as a generalization of the BFGS method. As such, we expect the convergence properties of Algorithm 2.2 to be similar to those of the BFGS algorithm. This is in fact the case.

The following theorem gives conditions under which Algorithm 2.2 converges. These results rely on the theorem of Zoutendijk (Theorem 1.22) and therefore, we impose that the objective function f is bounded below, the gradient ∇f is Lipschitz continuous, and either the Wolfe (1.18) or Goldstein and Price (1.19) conditions are used for step length selection. This theorem, as well as other theorems in this section, use the BFGS matrix $B_k(\lambda) = H_k(\lambda)^{-1} \approx \lambda I + \nabla^2 f(x_k)$ and some of the proofs are based on those offered in [25] for similar theorems. For the theorems in this section, λ_{k+1} is chosen to satisfy the Wolfe or the Goldstein and Price conditions. As such, we will adopt the simpler notation $B_k := B_k(\lambda_k)$ and $H_k := H_k(\lambda_k)$ for the remainder of this section.

For the following theorem, assume for all $k \geq 0$, there exists a constant $N > 0$

such that

$$\lambda_k < N. \tag{2.23}$$

That is, we assume the stepsize of the differential equation does not go to zero. Let $\mathcal{L}_{x_0} = \{x \in \mathbb{R}^n | f(x) \leq f(x_0)\}$ denote the level set of f at x_0 and assume that there exists constants m, and M such that

$$0 < m||w||^2 \leq w^T \nabla^2 f(x) w \leq M||w||^2 \tag{2.24}$$

for all $w \in \mathbb{R}^n$ and $x \in \mathcal{L}_{x_0}$.

Theorem 2.5. *Let $f : \mathbb{R}^n \to \mathbb{R} \in \mathcal{C}^2$ satisfy the hypotheses of Zoutendijk's Theorem 1.22 and let B_0 be a symmetric positive definite matrix. Then the sequence of iterates $\{x_k\}$ generated by Algorithm 2.2 has the property that $\liminf_{k\to\infty} ||\nabla f(x_k)|| = 0$. Additionally, if f is strictly convex on $\mathcal{L}_{x_0}$, then $\lim_{k\to\infty} x_k = x^*$, where x^* is the unique minimizer of f on $\mathcal{L}_{x_0}$.*

Proof. Calculating the trace and the determinant of the BFGS updating formula (2.6), we find

$$\begin{aligned}
\text{trace}(B_{k+1}) &= \text{trace}(B_k) - \text{trace}\left(\frac{(B_k s_k)(B_k s_k)^T}{s_k^T B_k s_k}\right) + \text{trace}\left(\frac{Y_k(\lambda_{k+1}) Y_k(\lambda_{k+1})^T}{s_k^T Y_k(\lambda_{k+1})}\right) \\
&= \text{trace}(B_k) - \frac{||B_k s_k||^2}{s_k^T B_k s_k} + \frac{||Y_k(\lambda_{k+1})||^2}{s_k^T Y_k(\lambda_{k+1})} \qquad (2.25) \\
\det(B_{k+1}) &= \det\left(B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{Y_k(\lambda_{k+1}) Y_k(\lambda_{k+1})^T}{s_k^T Y_k(\lambda_{k+1})}\right) \\
&= \det\left(B_k \frac{Y_k(\lambda_{k+1})^T s_k}{s_k^T B_k s_k}\right). \qquad (2.26)
\end{aligned}$$

Define

$$c_k := \cos\beta_k = \frac{s_k^T B_k s_k}{||s_k|| \cdot ||B_k s_k||}, \qquad w_k := \frac{s_k^T B_k s_k}{s_k^T s_k}. \tag{2.27}$$

Thus, c_k is the cosine of the angle β_k between s_k and $B_k s_k$. Let

$$\phi(B) := \text{trace}(B) - \ln(\det(B)) \tag{2.28}$$

where B is any positive definite matrix. If $\{\beta_i\}$ are the eigenvalues of a B, then using the fact that the trace is the sum of the eigenvalues and the determinant is the product of the eigenvalues, we can express (2.28) as $\phi(B) = \sum_{i=1}^{n}(\beta_i - \ln\beta_i)$. Since the function $h(x) = x - \ln(x)$ has a global minimum at $x = 1$ and no local maximum, $\phi(B) \geq 1 > 0$.

Let $\bar{H}_k = \int_0^1 \nabla^2 f(x_k + \tau\alpha_k p_k)\, d\tau$ be the average Hessian of $f(x)$ on a line between x_k and $x_k + \alpha_k p_k$. By Taylor's theorem, $y_k = \nabla f(x_{k+1}) - \nabla f(x_k) = \bar{H}_k s_k$. Then using (2.24), for any k and $\lambda_{k+1} \geq 0$, we have

$$\frac{Y_k(\lambda_{k+1})^T s_k}{s_k^T s_k} = \frac{\lambda_{k+1} s_k^T s_k}{s_k^T s_k} + \frac{y_k^T s_k}{s_k^T s_k} = \lambda_{k+1} + \frac{s_k^T \bar{H}_k s_k}{s_k^T s_k} \geq \lambda_{k+1} + m \geq m. \tag{2.29}$$

Setting $v_k := \bar{H}_k^{1/2} s_k$, we also have

$$\begin{aligned}
\frac{Y_k(\lambda_{k+1})^T Y_k(\lambda_{k+1})}{Y_k(\lambda_{k+1})^T s_k} &= \frac{\lambda_{k+1}^2 s_k^T s_k + 2\lambda_{k+1} s_k^T y_k + y_k^T y_k}{\lambda_{k+1} s_k^T s_k + y_k^T s_k} \\
&= \frac{\lambda_{k+1}^2 s_k^T s_k + 2\lambda_{k+1} s_k^T \bar{H}_k s_k + s_k^T \bar{H}_k^2 s_k}{\lambda_{k+1} s_k^T s_k + s_k^T \bar{H}_k s_k} \\
&= \frac{\lambda_{k+1}||s_k||^2 + 2\lambda_{k+1} v_k^T v_k + v_k^T \bar{H}_k v_k}{\lambda_{k+1}||s_k||^2 + ||v_k||^2} \\
&\leq \frac{N||s_k||^2 + 2N||v_k||^2 + v_k^T \bar{H}_k v_k}{||v_k||^2} \\
&\leq \frac{N||s_k||^2}{||v_k||^2} + 2N + M \leq N(m+1) + M.
\end{aligned} \tag{2.30}$$

Define $C := N(m+1) + M$. Using (2.25), (2.26), (2.28), (2.29), and (2.30), we have

$$\begin{aligned}
\phi(B_{k+1}) &= \operatorname{trace}(B_k) - \frac{w_k}{c_k^2} + \frac{||Y_k(\lambda_{k+1})||^2}{Y_k(\lambda_{k+1})^T s_k} - \ln\left(\det(B_k)\frac{Y_k(\lambda_{k+1})^T s_k}{w_k \cdot s_k^T s_k}\right) \\
&\leq \operatorname{trace}(B_k) + C - \frac{w_k}{c_k^2} - \ln(\det(B_k)) + \ln(w_k) \\
&\quad - \ln\left(\frac{Y_k(\lambda_{k+1})^T s_k}{s_k^T s_k}\right) \qquad (2.31) \\
&= \phi(B_k) + \left(C - \ln\left(\frac{Y_k(\lambda_{k+1})^T s_k}{s_k^T s_k}\right) - 1\right) \\
&\quad + \left(1 - \frac{w_k}{c_k^2} + \ln\left(\frac{w_k}{c_k^2}\right)\right) + \ln c_k^2.
\end{aligned}$$

The function $h(x) = 1 - x + \ln x$ has a global maximum at $x = 1$ with value $h(1) = 0$. Therefore, $h(x) \leq 0$ for all $x \geq 0$. Using this fact and defining $\bar{C} := C - \ln m - 1$, (2.31) can be further reduced to

$$\begin{aligned}
0 < \phi(B_{k+1}) &\leq \phi(B_k) + \left(C - \ln\left(\frac{Y_k(\lambda_{k+1})^T s_k}{s_k^T s_k}\right) - 1\right) + \ln c_k^2 \\
&\leq \phi(B_k) + (C - \ln m - 1) + \ln c_k^2 \\
&\leq \phi(B_{k-1}) + 2(C - \ln m - 1) + \ln c_k^2 + \ln c_{k-1}^2 \qquad (2.32) \\
&\quad \vdots \\
&\leq \phi(B_0) + \bar{C}(k+1) + \sum_{j=0}^{k} \ln c_j^2.
\end{aligned}$$

The constant M may be chosen sufficiently large so that

$$\bar{C} = C - \ln m - 1 = N(m+1) + M - \ln m - 1 > 0.$$

Now assume by way of contradiction, assume that $\liminf_{k\to\infty} ||\nabla f(x_k)|| \neq 0$. By Zoutendijk's Theorem 1.22, $\lim_{k\to\infty} c_k = 0$. Therefore, there exists a positive integer K so

that for all $i > K$, $\ln c_i^2 < -2\bar{C}$. Using this inequality in (2.32),

$$\begin{aligned} 0 < \phi(B_{k+1}) &\leq \phi(B_0) + \bar{C}(k+1) + \sum_{j=0}^{K} \ln c_j^2 + \sum_{j=K+1}^{k} \ln c_j^2 \\ &< \phi(B_0) + \bar{C}(k+1) + \sum_{j=0}^{K} \ln c_j^2 - 2\bar{C}(k-K-1) \\ &= \phi(B_0) + \bar{C}(3+2K) - k\bar{C} + \sum_{j=0}^{K} \ln c_j^2. \end{aligned} \tag{2.33}$$

However, the right hand side of (2.33) depends only on k and for k sufficiently large, this quantity is negative. Therefore, we have a contradiction and we conclude that $\liminf_{k\to\infty} ||\nabla f(x_k)|| = 0$. Additionally, if f is strictly convex, $\lim_{k\to\infty} ||\nabla f(x_k)|| = 0$ and $\lim_{k\to\infty} x_k = x^*$ where x^* is the unique minimizer of f on $\mathcal{L}_{x_0}$. □

Theorem 2.5 relies on the convexity of the objective function f on the level set $\mathcal{L}_{x_0}$ and the existence of an upper bound on the inverse stepsizes $\lambda_k = 1/h_k$ is needed so that $\lim_{k\to\infty} ||\nabla f(x_k)|| = 0$ is guaranteed. For general nonlinear functions, convexity is only guaranteed in a sufficiently small neighborhood near a local minimizer. The following theorem gives a second convergence result for Algorithm 2.2. This theorem ensures $||\nabla f(x_k)|| \to 0$ without convexity assumptions, but does require the additional assumptions that a lower bound on the minimum eigenvalue B_k exists independent of k as well as an upper bound, only one Newton iteration is performed at each iteration, and (2.14) is used as the initial iterate of the Newton iterations.

Theorem 2.6. *Let $p_k = -B_k^{-1}\nabla f(x_k)$ be the search direction chosen at each iteration of Algorithm 2.2 where B_k^{-1} is a symmetric positive definite matrix and assume that the hypotheses of Zoutendijk's theorem (1.22) hold. If there exists constants m and*

M independent of k so that

$$0 < m \leq \mu_{min}(B_k) \leq ||B_k|| \leq M$$

where $\mu_{min}(B_k)$ is the minimum eigenvalue of the matrix B_k, then the iterates of Algorithm 2.2 converge to an equilibrium point of the gradient system.

Proof. Since B_k is symmetric, it is diagonalizable by an orthogonal matrix Q_k; that is, $B_k = Q_k^T D_k Q_k$ where $D_k = \text{diag}(\mu_1, \ldots, \mu_n)$ is a diagonal matrix containing the eigenvalues of B_k. Defining $q_k := Q_k p_k$, we have

$$\begin{aligned} -\nabla f(x_k)^T p_k &= |p_k^T B_k p_k| = |q_k^T D_k q_k| \\ &\geq \mu_{\min}(B_k)||q_k^2|| = \mu_{\min}(B_k)|p_k^T Q_k^T Q_k p_k| \geq m||p_k||^2. \end{aligned} \tag{2.34}$$

Using (2.34) and the definition of p_k,

$$c_k := -\frac{\nabla f(x_k)^T p_k}{||\nabla f(x_k)|| \cdot ||p_k||} \geq \frac{m||p_k||^2}{||B_k p_k|| \cdot ||p_k||} \geq \frac{m}{M} > 0.$$

By Zoutendijk's theorem, we have

$$\sum_{k \geq 0} c_k^2 ||\nabla f(x_k)||^2 < \infty.$$

Since $c_k \geq m/M > 0$ for all k, it must be true that

$$\lim_{k \to \infty} ||\nabla f(x_k)|| = 0.$$

That is, x_k must converge to a point x^* where $\nabla f(x^*) = 0$. □

Under certain assumptions, the convergence of Algorithm 2.2 to a local minimizer is superlinear. To prove this, we first prove a few lemmas that will be useful.

For the remainder of the section, we will continue to use the inequality (2.24), but will only require that it holds in a neighborhood of a local minimizer. That is, we assume there exists a $r > 0$ so that if x^* is a local minimizer of f,

$$m||w||^2 \leq w^T \nabla^2 f(x) w \leq M||w||^2 \tag{2.35}$$

holds for all w such that $||w - x^*|| < r$. If we know that $x_k \to x^*$ as $k \to \infty$, there is a whole number N so that (2.35) is satisfied for all $k \geq N$.

Lemma 2.7. *Assume $f \in \mathcal{C}^1$ and x^* is a local minimizer of $f(x)$ such that the sequence x_k generated by Algorithm 2.2 converges to x^* as $k \to \infty$. Suppose $\nabla f(x)$ is locally Lipschitz in a neighborhood of x^* with Lipschitz constant $L > 0$. Then for all k sufficiently large,*

$$\frac{1-\gamma_2}{L}||\nabla f(x_k)||c_k \leq ||s_k|| \leq \frac{2(1-\gamma_1)}{m}||\nabla f(x_k)||c_k \tag{2.36}$$

where c_k is defined as in Theorem 2.6.

Proof. Using (2.35), for all k sufficiently large,

$$\begin{aligned} f(x_{k+1}) &= f(x_k) + \nabla f(x_k)^T s_k + \frac{1}{2} s_k^T \nabla^2 f(x_k + \tau s_k) s_k \\ &\geq f(x_k) + \nabla f(x_k)^T s_k + \frac{m}{2}||s_k||^2 \end{aligned} \tag{2.37}$$

where $\tau \in (0, 1)$. Therefore,

$$f(x_{k+1}) - f(x_k) \geq \nabla f(x_k)^T s_k + \frac{m}{2}||s_k||^2. \tag{2.38}$$

Also, by the Armijo condition,

$$f(x_k) - f(x_{k+1}) \geq -\gamma_1 \nabla f(x_k)^T s_k. \tag{2.39}$$

Adding (2.38) and (2.39) gives

$$0 \geq (1-\gamma_1)\nabla f(x_k)^T s_k + \frac{m}{2}||s_k||^2.$$

Since $\nabla f(x_k)^T s_k < 0$, this implies

$$\begin{aligned} ||s_k||^2 &\leq \frac{2(\gamma_1 - 1)}{m}\nabla f(x_k)^T s_k \\ &= \frac{2(1-\gamma_1)}{m}|\nabla f(x_k)^T s_k| \\ &= \frac{2(1-\gamma_1)}{m}||\nabla f(x_k)|| \cdot ||s_k|| c_k. \end{aligned}$$

Dividing by $||s_k||$ gives the right inequality of (2.36).

To show the left inequality, the curvature condition and the Lipschitz continuity of the gradient imply

$$\begin{aligned} 0 &\leq \nabla f(x_{k+1})^T s_k - \gamma_2 \nabla f(x_k)^T s_k \\ &= (\nabla f(x_{k+1}) - \nabla f(x_k))^T s_k + (1-\gamma_2)\nabla f(x_k)^T s_k \\ &\leq ||\nabla f(x_{k+1}) - \nabla f(x_k)|| \cdot ||s_k|| + (1-\gamma_2)\nabla f(x_k)^T s_k. \\ &\leq L||s_k||^2 + (1-\gamma_2)\nabla f(x_k)^T s_k \\ &= L||s_k||^2 - (1-\gamma_2)(-\nabla f(x_k)^T s_k) \\ &= L||s_k||^2 - (1-\gamma_2)||\nabla f(x_k)|| \cdot ||s_k|| c_k \end{aligned}$$

where c_k is defined as in 2.6. Dividing by $||s_k||$ and then solving for $||s_k||$ gives the left inequality of (2.36). □

Lemma 2.8. *Assume that the hypotheses of Theorem 2.5 are satisfied and assume that the sequence $\{x_k\}$ generated by Algorithm 2.2 converges to x^* as $k \to \infty$. If*

θ_k denotes the angle between $-\nabla f(x_k)$ and s_k, then there exists a whole number N so that for each $k > N$, more than half of the angles θ_j, $j = 0, \ldots, k$ satisfy $c_j^2 := \cos^2 \theta_j \geq e^{-4-4\bar{C}} =: \gamma$ where $\bar{C}$ is the constant from Theorem 2.5.

Proof. We prove the lemma by contradiction. That is, for each whole number N, there is a $k_N > N$ so that at least half of the angles θ_j, $j = 0, \ldots, k_N$ satisfy

$$c_j^2 < e^{-4-4\bar{C}}. \tag{2.40}$$

Applying logarithms, $\ln c_j < -2 - 2\bar{C}$. Using (2.32) on k_N,

$$0 < \phi(B_0) + \bar{C}k_N + \bar{C} + \sum_{j=0}^{k_N} \ln c_j^2. \tag{2.41}$$

Now we partition the $k_N + 1$ angles into two sets based on whether they satisfy the inequality (2.40). To this end, let a_1 and a_2 be chosen so that $a_1 k_N + 1$ and $a_2 k_N + 1$ are the number of angles which respectively satisfy and do not satisfy (2.40). Since at least half of the angles satisfy this inequality, we have that $0 \leq a_2 \leq 1/2 \leq a_1 \leq 1$. After reindexing, let $j_- = 0, \ldots, a_1 k_N$ denote the indices which satisfy (2.40) and $j_+ = 0, \ldots, a_2 k_N$ denote the indices which do not satisfy this inequality. Then (2.41)

can be simplified to

$$\begin{aligned}
0 &< \phi(B_0) + \bar{C}k_N + \bar{C} + \sum_{j_-=0}^{a_1k_N} \ln c_{j_-}^2 + \sum_{j_+=0}^{a_2k_N} \ln c_{j_+}^2 \\
&\leq \phi(B_0) + \bar{C}k_N + \bar{C} + \sum_{j_-=0}^{a_1k_N} (-2 - 2\bar{C}) + \sum_{j_+=0}^{a_2k_N} 0 \\
&= \phi(B_0) + \bar{C}k_N + \bar{C} + (1 + a_1k_N)(-2 - 2\bar{C}) \\
&= \phi(B_0) + \bar{C}k_N + \bar{C} - 2 - 2\bar{C} - 2a_1k_N - 2a_1k_N\bar{C} \qquad (2.42) \\
&= \phi(B_0) - 2 - \bar{C} + (\bar{C} - 2a_1 - 2a_1\bar{C})k_N \\
&= \phi(B_0) - 2 - \bar{C} + ((1 - 2a_1)\bar{C} - 2a_1)k_N.
\end{aligned}$$

Since $a_1 \geq 1/2$ and $\bar{C} > 0$, $(1 - 2a_1)\bar{C} - 2a_1$ is negative. Inequality (2.42) holds for any $N > 0$ so when k_N is sufficiently large, the right hand side of (2.42) is negative, a contradiction. Therefore, there must exist a natural number N such that for all $k > N$, more than half of the angles θ_j, $j = 0, \ldots, k$ satisfy $c_j^2 \geq e^{-4-4\bar{C}}$. □

Lemma 2.8 ensures that 'enough' angles θ_j are bounded away from $\pi/2$. The reason for this will become apparent in Lemma 2.10.

Lemma 2.9. *Assume $f \in \mathcal{C}^2$ and x^* is a local minimizer of $f(x)$ such that the sequence x_k generated by Algorithm 2.2 converges to x^* as $k \to \infty$. Then for k sufficiently large,*

$$f(x_k) - f(x^*) \leq \frac{2}{m}||\nabla f(x_k)||^2. \qquad (2.43)$$

Proof. By (2.35) and (2.37),

$$f(x^*) \geq f(x_k) + \nabla f(x_k)^T(x^* - x_k)$$

for all k sufficiently large. This inequality implies

$$\nabla f(x_k)^T(x^* - x_k) \leq f(x^*) - f(x_k) < 0.$$

Again by (2.35) and (2.37), for all k sufficiently large,

$$\begin{aligned} f(x^*) &\geq f(x_k) + \nabla f(x_k)^T(x^* - x_k) + \frac{m}{2}||x^* - x_k||^2 \\ &\geq f(x_k) - |\nabla f(x_k)^T(x^* - x_k)| + \frac{m}{2}||x^* - x_k||^2 \\ &\geq f(x_k) - ||\nabla f(x_k)|| \cdot ||x^* - x_k|| + \frac{m}{2}||x^* - x_k||^2. \end{aligned}$$

Therefore,

$$0 < f(x_k) - f(x^*) \leq ||\nabla f(x_k)|| \cdot ||x^* - x_k|| - \frac{m}{2}||x^* - x_k||^2. \tag{2.44}$$

Dividing by $||x^* - x_k||$ and then solving for $||x^* - x_k||$ gives

$$||x^* - x_k|| \leq \frac{2}{m}||\nabla f(x_k)||$$

which, reintroduced in (2.44) leads to (2.43). □

When x_k approaches the local minimizer, Lemma 2.9 allows the quantity $f(x_k) - f(x^*)$ to be bounded by an expression involving only the norm of the gradient at the current iterate. The previous lemmas will be instrumental in showing the next lemma.

Lemma 2.10. *Let $f \in \mathcal{C}^2$ and assume that the sequence of iterates x_k generated by Algorithm 2.2 converges to a local minimizer x^*. If the assumptions of Lemmas 2.7 and 2.9 are satisfied, then*

$$\sum_{k=0}^{\infty} ||x_k - x^*|| < \infty.$$

Proof. Using the Armijo condition, Lemmas 2.7 and 2.9, and the definition of c_k as in Lemma 2.8, we have

$$\begin{aligned} f(x_{k+1}) - f(x^*) &\leq f(x_k) - f(x^*) + \gamma_1 \nabla f(x_k)^T s_k \\ &= f(x_k) - f(x^*) - \gamma_1 ||\nabla f(x_k)|| \cdot ||s_k|| c_k \\ &\leq f(x_k) - f(x^*) - \frac{(1-\gamma_2)\gamma_1}{L} ||\nabla f(x_k)||^2 c_k^2 \qquad (2.45) \\ &\leq f(x_k) - f(x^*) - \frac{(1-\gamma_2)\gamma_1 m}{2L} (f(x_k) - f(x^*)) c_k^2 \\ &= \left(1 - \frac{m(1-\gamma_2)\gamma_1 c_k^2}{2L}\right) (f(x_k) - f(x^*)). \end{aligned}$$

Recursively applying this inequality, we have,

$$\begin{aligned} f(x_{k+1}) - f(x^*) &\leq \left(1 - \frac{m(1-\gamma_2)\gamma_1 c_k^2}{2L}\right) (f(x_k) - f(x^*)) \\ &\leq (f(x_0) - f(x^*)) \prod_{j=0}^{k} \left[1 - \frac{m(1-\gamma_2)\gamma_1}{2L} c_j^2\right]. \qquad (2.46) \end{aligned}$$

Since $0 \leq f(x_k) - f(x^*)$ for all $k \geq 0$, the quantity in square brackets in (2.46) must be nonnegative. Additionally, this quantity is bounded by $m(1-\gamma_2)\gamma_1 c_j^2/2L$ which is positive for all $j \geq 0$. From Lemma 2.8, at least half of the angles θ_j, $j = 0, \ldots, k$ satisfy the inequality $\cos^2 \theta_j \geq \gamma > 0$. Therefore, (2.46) becomes

$$f(x_{k+1}) - f(x^*) \leq \begin{cases} (f(x_0) - f(x^*))\tau^{k/2}, & \text{if } k \text{ even} \\ (f(x_0) - f(x^*))\tau^{(k-1)/2}, & \text{if } k \text{ odd} \end{cases} \qquad (2.47)$$

where $\tau = 1 - \dfrac{m(1-\gamma_2)\gamma_1\gamma}{2L}$. Note that we may choose the Lipschitz constant L sufficiently large so that $\dfrac{m}{2L}(1-\gamma_2)\gamma_1 c_j^2 \leq 1$. Therefore, we have that $\tau \in [0, 1)$. Using (2.35) applied to x_k and x^* and the fact that $x_k \to x^*$ as $k \to \infty$, there is a

whole number N such that for $k \geq N$,

$$\begin{aligned} f(x_k) &= f(x^*) + \nabla f(x^*)^T(x_k - x^*) + \frac{1}{2}(x_k - x^*)^T \nabla^2 f(x^* + \delta(x_k - x^*))(x_k - x^*) \\ &\geq f(x^*) + \frac{m}{2}||x_k - x^*||^2 \end{aligned}$$

where $\delta \in [0, 1]$. Solving this inequality for $||x_k - x^*||$ gives

$$||x - x^*|| \leq \sqrt{\frac{2}{m}(f(x) - f(x^*))}. \tag{2.48}$$

Since $\sum_{k=0}^{N} ||x_k - x^*||$ is finite, we can assume without loss of generality that $N = 0$; that is, that x_k satisfies (2.48) for all $k \geq 0$. Using (2.47) and (2.48),

$$\begin{aligned} \sum_{k=0}^{\infty} ||x_k - x^*|| &\leq \sum_{k=0}^{\infty} \sqrt{\frac{2}{m}(f(x_k) - f(x^*))} \\ &\leq \sqrt{\frac{2}{m}(f(x_0) - f(x^*))} \left[\sum_{k \text{ even}}^{\infty} \tau^{k/4} + \sum_{k \text{ odd}}^{\infty} \tau^{(k-1)/4} \right] \\ &\leq \sqrt{\frac{2}{m}(f(x_0) - f(x^*))} \sum_{k=0}^{\infty} 2\tau^{k/2} \\ &= 2\sqrt{\frac{2}{m}(f(x_0) - f(x^*))} \left(\frac{1}{1 - \tau^{1/2}} \right) \\ &< \infty \end{aligned}$$

where we have used the fact that $0 \leq \tau < 1$ and therefore, $\sum_{k=0}^{\infty} \tau^{k/2}$ is a geometric series. □

From this point on, since $x_k \to x^*$ as $k \to \infty$, we assume without loss of generality that (2.37), and therefore, all previous lemmas hold for all $k \geq N = 0$.

Notice that if $\sum_{j=0}^{\infty} \lambda_j < \infty$ and $u > 0$ is a scalar, Lemma 2.10 implies that iterates of Algorithm 2.2 satisfy

$$\sum_{j=0}^{\infty} \left(\frac{\lambda_k}{u} + ||x_k - x^*|| \right) < \infty. \tag{2.49}$$

Now the main rate of convergence result is presented. Since Algorithm 2.2 is identical to BFGS algorithm when $\lambda_k = 0$ for all k and only one Newton iteration is performed with $x_k^{(0)} = x_k$, we would expect the rate of convergence to a local minimizer x^* to be similar if $\lambda_k \to 0$ sufficiently fast near x^*. In fact, this is the case. For simplicity, let $H(x) := \nabla^2 f(x_k)$.

Theorem 2.11. *Let $f \in \mathcal{C}^2$ and assume that the sequence generated by Algorithm 2.2 converges to a local minimizer x^* at which the Hessian matrix is Lipschitz continuous. That is, there exists a constants $L > 0$ and $\delta > 0$ so that*

$$||H(x) - H(x^*)|| \leq L||x - x^*||$$

for all $||x - x^*|| < \delta$. Lastly, assume that

$$\sum_{k=0}^{\infty} \lambda_k < \infty. \tag{2.50}$$

If only one Newton iteration is performed with initial Newton iterate satisfying (2.14) (so that $p_k = -H_k \nabla f(x_k)$), then x_k converges to x^* superlinearly.

Proof. Since x^* is a local minimizer, $H(x^*)$ is positive definite. Therefore, the square root, $H(x^*)^{1/2}$, exists and is positive definite. Define the following quantities:

$$\bar{s}_k = H(x^*)^{1/2} s_k, \qquad \bar{y}_k = H(x^*)^{-1/2} y_k, \qquad \bar{Y}_k(\lambda) = \lambda H(x^*)^{-1} \bar{s}_k + \bar{y}_k$$

$$\bar{B}_k = H(x^*)^{-1/2} B_k H(x^*)^{-1/2}.$$

By Taylor's theorem, $y_k = \bar{H}_k s_k$ where $\bar{H}_k$ is the average Hessian of f at x_k

as defined in the proof of Theorem 2.5. Using this, we have

$$\begin{aligned}
||\bar{Y}_k(\lambda_{k+1}) - \bar{s}_k|| &= ||\lambda_{k+1}H(x^*)^{-1}H(x^*)^{1/2}s_k + H(x^*)^{-1/2}\bar{H}_k s_k - H(x^*)^{1/2}s_k|| \\
&= ||H(x^*)^{-1/2}\left(\lambda_{k+1}I + \bar{H}_k - H(x^*)\right)s_k|| \\
&\leq ||H(x^*)^{-1/2}||^2||\bar{s}_k|| \cdot ||\lambda_{k+1}I + \bar{H}_k - H(x^*)|| \\
&\leq ||H(x^*)^{-1/2}||^2||\bar{s}_k||\left(\lambda_{k+1} + L\epsilon_k\right) \qquad (2.51) \\
&= L||H(x^*)^{-1/2}||^2||\bar{s}_k||\left(\frac{\lambda_{k+1}}{L} + \epsilon_k\right) \\
&= \bar{c}||\bar{s}_k||\left(\frac{\lambda_{k+1}}{L} + \epsilon_k\right) \\
&= \bar{c}v_k||\bar{s}_k||
\end{aligned}$$

where

$$\epsilon_k = \max\{||x_{k+1} - x^*||, ||x_k - x^*||\}, \quad \bar{c} = L||H(x^*)^{-1/2}||^2, \quad \text{and} \quad v_k = \lambda_{k+1}/L + \epsilon_k.$$

By Lemma 2.10 and (2.50), $v_k \to 0$ as $k \to \infty$. Therefore, there exists a whole number N so that for all $k \geq N$,

$$1 - \bar{c}v_k > \frac{1}{2}. \qquad (2.52)$$

Using (2.51),

$$||\bar{Y}_k(\lambda_{k+1})|| - ||\bar{s}_k|| \leq ||\bar{Y}_k(\lambda_{k+1}) - \bar{s}_k|| \leq \bar{c}v_k||\bar{s}_k||$$

$$||\bar{s}_k|| - ||\bar{Y}_k(\lambda_{k+1})|| \leq ||\bar{Y}_k(\lambda_{k+1}) - \bar{s}_k|| \leq \bar{c}v_k||\bar{s}_k||$$

and together, these two inequalities bound $||\bar{Y}_k(\lambda_{k+1})||$.

$$(1 - \bar{c}v_k)||\bar{s}_k|| \leq ||\bar{Y}_k(\lambda_{k+1})|| \leq (1 + \bar{c}v_k)||\bar{s}_k|| \qquad (2.53)$$

By (2.51), we have that if $k \geq N$, we can square the left inequality in (2.53). That is,

$$(1 - \bar{c}v_k)^2 ||\bar{s}_k||^2 \leq ||\bar{Y}_k(\lambda_{k+1})||^2 \tag{2.54}$$

Using (2.53), (2.54) and the definition of norm,

$$||\bar{Y}_k(\lambda_{k+1}) - \bar{s}_k||^2 = ||\bar{Y}_k(\lambda_{k+1})||^2 - 2\bar{Y}_k(\lambda_{k+1})\bar{s}_k + ||\bar{s}_k||^2 \leq \bar{c}^2 v_k^2 ||\bar{s}_k||^2$$

$$\begin{aligned} ||\bar{Y}_k(\lambda_{k+1}) - \bar{s}_k||^2 &= ||\bar{Y}_k(\lambda_{k+1})||^2 - 2\bar{Y}_k(\lambda_{k+1})\bar{s}_k + ||\bar{s}_k||^2 \\ &\geq (1 - \bar{c}v_k)^2 ||\bar{s}_k||^2 - 2\bar{Y}_k(\lambda_{k+1})\bar{s}_k + ||\bar{s}_k||^2 \end{aligned}$$

These two equations give a positive lower bound on the inner product $\bar{Y}_k(\lambda_{k+1})^T \bar{s}_k$.

$$\begin{aligned} \bar{Y}_k(\lambda_{k+1})^T \bar{s}_k &\geq \frac{1}{2}\left(-\bar{c}^2 v_k^2 ||\bar{s}_k||^2 + (1 - \bar{c}v_k)^2 ||\bar{s}_k||^2 + ||\bar{s}_k||^2\right) \\ &= \frac{1}{2}(-\bar{c}^2 v_k^2 + (1 - \bar{c}v_k)^2 + 1)||\bar{s}_k||^2 \\ &= (1 - \bar{c}v_k)||\bar{s}_k||^2 \end{aligned} \tag{2.55}$$

Therefore, using (2.54) and (2.55), we have the useful inequalities,

$$\frac{\bar{Y}_k(\lambda_{k+1})^T \bar{s}_k}{||\bar{s}_k||^2} \geq \frac{(1 - \bar{c}v_k)||\bar{s}_k||^2}{||\bar{s}_k||^2} = 1 - \bar{c}v_k \tag{2.56}$$

$$\frac{||\bar{Y}_k(\lambda_{k+1})||^2}{\bar{Y}_k(\lambda_{k+1})^T \bar{s}_k} \leq \frac{(1 + \bar{c}v_k)^2 ||\bar{s}_k||^2}{(1 - \bar{c}v_k)||\bar{s}_k||^2} = \frac{(1 + \bar{c}v_k)^2}{(1 - \bar{c}v_k)}. \tag{2.57}$$

Consider the scalar function $g(x) = (3\bar{c} + \bar{c}^2 x)/(1 - \bar{c}x)$. Taking the derivative with respect to x, we find

$$g'(x) = \frac{4\bar{c}^2}{(1 - \bar{c}x)^2} > 0.$$

Therefore, if $x < y$, then $g(x) < g(y)$ and since $v_k \to 0$ as $k \to \infty$, there exists a constant v_{max} so that $g(v_k) \leq g(v_{max})$ for all $k \geq N$. Let $c := \max\{g(v_{max}), \bar{c}\}$. Then $\bar{c} \leq c$ and (2.57) can be simplified to

$$\begin{aligned} \frac{||\bar{Y}_k(\lambda_{k+1})||^2}{\bar{Y}_k(\lambda_{k+1})^T \bar{s}_k} &\leq \frac{1 + 2\bar{c}v_k + \bar{c}^2 v_k^2}{(1 - \bar{c}v_k)} \\ &= 1 + \frac{3\bar{c} + \bar{c}^2 v_k}{(1 - \bar{c}v_k)} v_k \\ &= 1 + g(v_k)v_k \qquad (2.58) \\ &\leq 1 + cv_k. \end{aligned}$$

In the proof of Theorem 2.5, the function $h(t) = 1 - t + \ln t$ was shown to be nonpositive. Thus,

$$0 \geq h\left(\frac{1}{1 - \bar{c}v_k}\right) = \frac{-\bar{c}v_k}{1 - \bar{c}v_k} - \ln(1 - \bar{c}v_k)$$

and using (2.52) we conclude

$$\ln(1 - \bar{c}v_k) \geq \frac{-\bar{c}v_k}{1 - \bar{c}v_k} \geq -2\bar{c}v_k \geq -2cv_k. \qquad (2.59)$$

Now we return to the BFGS update formula. Using the update (2.6),

$$\begin{aligned}
\bar{B}_{k+1} &= H(x^*)^{-1/2} B_{k+1} H(x^*)^{-1/2} \\
&= H(x^*)^{-1/2} \left(B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{Y_k(\lambda_{k+1}) Y_k(\lambda_{k+1})^T}{s_k^T Y_k(\lambda_{k+1})} \right) H(x^*)^{-1/2} \\
&= \frac{H(x^*)^{-1/2} B_k H(x^*)^{-1/2} H(x^*)^{1/2} s_k s_k^T H(x^*)^{1/2} H(x^*)^{-1/2} B_k H(x^*)^{-1/2}}{s_k^T H(x^*)^{1/2} H(x^*)^{-1/2} B_k H(x^*)^{-1/2} H(x^*)^{1/2} s_k} \\
&\quad + \frac{H(x^*)^{-1/2} Y_k(\lambda_{k+1}) Y_k(\lambda_{k+1})^T H(x^*)^{-1/2}}{s_k^T H(x^*)^{1/2} H(x^*)^{-1/2} Y_k(\lambda_{k+1})} \\
&= \frac{\bar{B}_k \bar{s}_k \bar{s}_k^T \bar{B}_k}{\bar{s}_k^T \bar{B}_k \bar{s}_k} \\
&\quad + \frac{\left(\lambda_{k+1} H(x^*)^{-1/2} s_k + H(x^*)^{-1/2} y_k \right) \left(\lambda_{k+1} H(x^*)^{-1/2} s_k + H(x^*)^{-1/2} y_k \right)^T}{\bar{s}_k^T \left(\lambda_{k+1} H(x^*)^{-1/2} s_k + H(x^*)^{-1/2} y_k \right)} \\
&= \frac{\bar{B}_k \bar{s}_k \bar{s}_k^T \bar{B}_k}{\bar{s}_k^T \bar{B}_k \bar{s}_k} + \frac{\left(\lambda_{k+1} H(x^*)^{-1} \bar{s}_k + \bar{y}_k \right) \left(\lambda_{k+1} H(x^*)^{-1} \bar{s}_k + \bar{y}_k \right)^T}{\bar{s}_k^T \left(\lambda_{k+1} H(x^*)^{-1} \bar{s}_k + \bar{y}_k \right)} \\
&= \frac{\bar{B}_k \bar{s}_k \bar{s}_k^T \bar{B}_k}{\bar{s}_k^T \bar{B}_k \bar{s}_k} + \frac{\bar{Y}_k(\lambda_{k+1}) \bar{Y}_k(\lambda_{k+1})^T}{\bar{s}_k^T \bar{Y}_k(\lambda_{k+1})}
\end{aligned}$$

Therefore, $\bar{s}_k$, $\bar{Y}_k(\lambda_{k+1})$, and $\bar{B}_k$ as defined above preserve the BFGS update formula for $\bar{B}_{k+1}$. Similar to the notation used in Theorem 2.5, let

$$\bar{c}_k := \frac{\bar{s}_k^T \bar{B}_k \bar{s}_k}{||\bar{s}_k|| \cdot ||\bar{B}_k \bar{s}_k||}, \qquad \bar{w}_k = \frac{\bar{s}_k^T \bar{B}_k \bar{s}_k}{||\bar{s}_k^T||}.$$

Therefore, the formulas (2.25) and (2.26) may be applied to $\bar{B}_{k+1}$ and used in the equation (2.28) along with (2.55), (2.58), and (2.59). That is,

$$\begin{aligned}
\phi(\bar{B}_{k+1}) &= \phi(\bar{B}_k) - \frac{\bar{w}_k}{\bar{c}_k^2} + \frac{||\bar{Y}_k(\lambda_{k+1})||^2}{\bar{Y}_k(\lambda_{k+1})^T \bar{s}_k} + \ln \bar{w}_k - \ln \frac{\bar{Y}_k(\lambda_{k+1})^T \bar{s}_k}{\bar{s}_k^T \bar{s}_k} \\
&= \phi(\bar{B}_k) + \left(\frac{||\bar{Y}_k(\lambda_{k+1})||^2}{\bar{Y}_k(\lambda_{k+1})^T \bar{s}_k} - \ln \frac{\bar{Y}_k(\lambda_{k+1})^T \bar{s}_k}{\bar{s}_k^T \bar{s}_k} - 1 \right) + \left(1 - \frac{\bar{w}_k}{\bar{c}_k^2} + \ln \frac{\bar{w}_k}{\bar{c}_k^2} \right) + \ln \bar{c}_k^2 \\
&\leq \phi(\bar{B}_k) + (1 + cv_k + 2\bar{c}v_k - 1) + \left(1 - \frac{\bar{w}_k}{\bar{c}_k^2} + \ln \frac{\bar{w}_k}{\bar{c}_k^2} \right) + \ln \bar{c}_k^2 \\
&\leq \phi(\bar{B}_k) + 3cv_k + \left(1 - \frac{\bar{w}_k}{\bar{c}_k^2} + \ln \frac{\bar{w}_k}{\bar{c}_k^2} \right) + \ln \bar{c}_k^2.
\end{aligned}$$

Recursively applying this inequality gives

$$0 < \phi(\bar{B}_{k+1}) \leq \phi(\bar{B}_0) + \sum_{j=0}^{k+1} 3cv_j + \left(1 - \frac{\bar{w}_j}{\bar{c}_j^2} + \ln \frac{\bar{w}_j}{\bar{c}_j^2}\right) + \ln \bar{c}_j^2.$$

Rearranging this inequality and taking the limit as $k \to \infty$,

$$\sum_{j=0}^{\infty} \ln \frac{1}{\bar{c}_j^2} - \left(1 - \frac{\bar{w}_j}{\bar{c}_j^2} + \ln \frac{\bar{w}_j}{\bar{c}_j^2}\right) \leq \phi(\bar{B}_0) \sum_{j=0}^{\infty} 3cv_j < \infty. \tag{2.60}$$

Since the expression in the parentheses of the leftmost expression of (2.60) is nonpositive and $\ln(1/\bar{c}_j^2) \geq 0$ for all $j \geq 0$,

$$\sum_{k=0}^{\infty} \ln \frac{1}{\bar{c}_j^2} - \left(1 - \frac{\bar{w}_j}{\bar{c}_j^2} + \ln \frac{\bar{w}_j}{\bar{c}_j^2}\right) < \infty$$

implies that both

$$\lim_{j\to\infty} \ln \frac{1}{\bar{c}_j^2} = 0, \qquad \lim_{j\to\infty} \left(1 - \frac{\bar{w}_j}{\bar{c}_j^2} + \ln \frac{\bar{w}_j}{\bar{c}_j^2}\right) = 0.$$

Since $\ln(1/x) = 0$ implies $x = 1$, we have

$$\lim_{j\to\infty} \bar{c}_j^2 = 1. \tag{2.61}$$

Secondly, $\bar{c}_j^2 \to 1$ enforces $1 - \bar{w}_j + \ln \bar{w}_j \to 0$. Therefore,

$$\lim_{j\to\infty} \bar{w}_j = 1. \tag{2.62}$$

Using (2.61) and (2.62),

$$\begin{aligned}
\lim_{k\to\infty} \frac{||H(x^*)^{-1/2}\left(B_k - H(x^*)\right)s_k||^2}{||H(x^*)^{1/2}s_k||^2} &= \lim_{k\to\infty} \frac{||\left(\bar{B}_k - I\right)\bar{s}_k||^2}{||\bar{s}_k||^2} \\
&= \lim_{k\to\infty} \frac{||\bar{B}_k\bar{s}_k||^2 - 2\bar{s}_k^T\bar{B}_k\bar{s}_k + ||\bar{s}_k||^2}{||\bar{s}_k||^2} \\
&= \lim_{k\to\infty} \left(\frac{||\bar{B}_k\bar{s}_k||^2||\bar{s}_k||^2(\bar{s}_k^T\bar{B}_k\bar{s}_k)^2}{||\bar{s}_k||^4(\bar{s}_k^T\bar{B}_k\bar{s}_k)^2} \right. \\
&\quad \left. + \frac{-2\bar{s}_k^T\bar{B}_k\bar{s}_k}{||\bar{s}_k||^2} + 1 \right) \\
&= \lim_{k\to\infty} \left(\frac{\bar{w}_k^2}{\bar{c}_k^2} - 2\bar{w}_k + 1 \right) \\
&= 0.
\end{aligned}$$

Therefore,

$$\begin{aligned}
0 &= \lim_{k\to\infty} \frac{||H(x^*)^{-1/2}\left(B_k - H(x^*)\right)s_k||^2}{||H(x^*)^{1/2}s_k||^2} \\
&= \lim_{k\to\infty} \frac{||H(x^*)^{1/2}||}{||H(x^*)^{1/2}||} \frac{||H(x^*)^{-1/2}\left(B_k - H(x^*)\right)s_k||}{||H(x^*)^{1/2}s_k||} \\
&\geq \lim_{k\to\infty} \frac{||H(x^*)^{1/2}H(x^*)^{-1/2}\left(B_k - H(x^*)\right)s_k||}{||H(x^*)^{1/2}||^2||s_k||} \\
&= \lim_{k\to\infty} \frac{||\left(B_k - H(x^*)\right)s_k||}{||s_k||}.
\end{aligned}$$

Therefore, by the theorem of Dennis and Moré (Theorem 1.23), the convergence of x_k to x^* is superlinear. □

As long as p_k is chosen so that $p_k = -B_k^{-1}\nabla f(x_k)$ for all k greater than some constant N, Theorem 2.11 says that the speed of convergence is superlinear. Therefore if h_k is chosen so that $h_k \to \infty$ as $k \to \infty$, all that is needed to ensure superlinear convergence sufficiently close to an equilibrium point is the search direction (2.15). If p_k is chosen using methods other than (2.15), we still have superlinear convergence

as long as the eventual switch to the search direction (2.15) is made.

2.3 Hybrid Algorithm of Order Two

In this section, we extend the results based on the one-stage implicit Euler method to two-stage implicit Runge-Kutta methods. The corresponding Butcher tableau for these two-stage methods is given by

$$\begin{array}{c|cc} c_1 & a_{11} & a_{12} \\ c_2 & a_{21} & a_{22} \\ \hline & b_1 & b_2 \end{array} \tag{2.63}$$

The coefficient matrix A in (2.63) gives rise to two eigenvalues, μ_1 and μ_2. In Section 1.1.3, it was shown that for implicit Runge-Kutta methods, numerically solving the full $sn \times sn$ Newton system (1.21) is equivalent to solving s systems of size $n \times n$ where system i at step k depends on the matrix $\left(\frac{\lambda_k}{\mu_i} I + J\right)$. Considering two-stage Runge-Kutta methods, we use two quasi-Newton matrices, $H_{1,\,k}(\lambda)$ and $H_{2,\,k}(\lambda)$, so that

$$\begin{aligned} H_{1,\,k+1}(\lambda) &\approx \left(\frac{\lambda}{\mu_1} I + \nabla^2 f(x_k)\right)^{-1} \approx \left(\frac{\lambda}{\mu_1} I + J\right)^{-1} \\ H_{2,\,k+1}(\lambda) &\approx \left(\frac{\lambda}{\mu_2} I + \nabla^2 f(x_k)\right)^{-1} \approx \left(\frac{\lambda}{\mu_2} I + J\right)^{-1}. \end{aligned}$$

From (2.4), these matrices satisfy the four secant equations

$$\begin{aligned} H_{1,\,k+1}(\lambda_{k+1}) Y_{i,k}(\lambda_{k+1}) &= s_{i,k} \\ H_{2,\,k+1}(\lambda_{k+1}) Y_{i,k}(\lambda_{k+1}) &= s_{i,k}. \end{aligned} \tag{2.64}$$

where $Y_{i,k}(\lambda) = \dfrac{\lambda}{\mu_i} s_{i,k} + y_{i,k}$ for $i = 1, \ldots, 2$,

$$s_{1,k} = X^{(0)}_{1,k} - x_k, \qquad s_{2,k} = X^{(0)}_{2,k} - X^{(0)}_{1,k},$$

$$y_{1,k} = \nabla f(X^{(0)}_{1,k}) - \nabla f(x_k), \quad y_{2,k} = \nabla f(X^{(0)}_{2,k}) - \nabla f(X^{(0)}_{1,k}),$$

and $X_{i,k}$ denotes the i^{th} stage at step k.

We restrict the class of all two-stage Runge-Kutta methods to RK methods satisfying the conditions shown in Table 2.2. Notice that all stiffly accurate methods

Table 2.2: Assumptions for Two-Stage Hybrid Algorithms

1. A has real positive eigenvalues
2. Stage order 2
3. Stiffly accurate and hence L-stable
4. Order 2 dense output

are L-stable since

$$\lim_{z\to\infty} R(z) = \lim_{z\to\infty} 1 + b^T(z^{-1}I - A)^{-1}\mathbb{1} = 1 - b^T A^{-1}\mathbb{1} = 1 - e_s A A^{-1}\mathbb{1} = 0$$

where we use the fact that for stiffly accurate methods, $A^T e_s = b$, $e_s = (0, \ldots, 0, 1)^T$.

Theorem 2.12. *Let $c_2 > 0$. If a Runge-Kutta method satisfies the first two conditions in Table 2.2, then*

$$c_1 \leq (3 - 2\sqrt{2})c_2 \qquad or \qquad c_1 \geq (3 + 2\sqrt{2})c_2. \tag{2.65}$$

Additionally, if the Runge-Kutta method is stiffly accurate, then $c_2 = 1$.

Proof. Since the Runge-Kutta method has stage order two, by (1.45), we require

$$\sum_{j=1}^{s} a_{ij} c_j^{k-1} = \frac{c_i^k}{k} \tag{2.66}$$

to hold for $i = 1, 2$ and $k = 1, 2$. This is equivalent to

$$A \begin{pmatrix} 1 & c_1 \\ 1 & c_2 \end{pmatrix} = \begin{pmatrix} c_1 & c_1^2/2 \\ c_2 & c_2^2/2 \end{pmatrix}.$$

Let

$$V := \begin{pmatrix} 1 & c_1 \\ 1 & c_2 \end{pmatrix}^{-1}.$$

We seek conditions on c_1 and c_2 so that the eigenvalues of A are real, positive, and satisfy the stage order conditions. Let μ_1 and μ_2 denote the eigenvalues of A. Since these eigenvalues are real and positive, A is invertible. Therefore, V is also invertible and we have that $c_1 \neq c_2$. Calculating the eigenvalues of A, we have

$$\begin{aligned}
\det(\mu I - A) &= \frac{\det(\mu V - AV)}{\det(V)} \\
&= \frac{1}{c_2 - c_1} \det \begin{pmatrix} \mu - c_1 & \mu c_1 - c_1^2/2 \\ \mu - c_2 & \mu c_2 - c_2^2/2 \end{pmatrix} \\
&= \frac{1}{c_2 - c_1} \left(\mu^2 (c_2 - c_1) - \frac{\mu}{2}(c_2^2 - c_1^2) + \frac{c_1 c_2}{2}(c_2 - c_1) \right) \\
&= \mu^2 - \mu \frac{c_1 + c_2}{2} + \frac{c_1 c_2}{2}.
\end{aligned}$$

Define $p(\mu) := \mu^2 - \mu\frac{c_1+c_2}{2} + \frac{c_1c_2}{2}$. The eigenvalues of A are found by solving $p(\mu) = 0$ and we want to show that solutions to this equation are strictly positive. The solutions are given by

$$\mu = \frac{c_1+c_2}{4} \pm \frac{1}{2}\sqrt{\frac{1}{4}c_1^2 + \frac{1}{4}c_2^2 + \frac{1}{2}c_1c_2 - 2c_1c_2}.$$

Since we require these two eigenvalues to be real and positive, we require

$$c_1^2 + c_2^2 - 6c_1c_2 \geq 0. \tag{2.67}$$

Equality occurs when

$$c_1 = c_2(3 \pm 2\sqrt{2}).$$

Inequality (2.67) will be satisfied (and hence, $p(\mu)$ is real) when

$$c_1 \leq c_2(3 - 2\sqrt{2}) \qquad \text{or} \qquad c_1 \geq c_2(3 + 2\sqrt{2}).$$

All that remains is to check whether these eigenvalues are positive or negative. Let $\hat{p}(\mu) := p(-\mu)$. Using the Routh-Hurwitz conditions, $\hat{p}(\mu)$ has negative real roots if the first column of the following table contains only positive real numbers.

$$\begin{array}{|cc} 1 & \frac{c_1c_2}{2} \\ \frac{c_1+c_2}{2} & \\ \frac{c_1c_2}{2} & \end{array}$$

Therefore, the roots of $\hat{p}(\mu)$ are strictly negative if $c_1 > 0$ and $c_2 > 0$ and hence the roots of $p(\mu)$ are strictly positive when $c_1 > 0$ and $c_2 > 0$. If the Runge-Kutta method is stiffly accurate, then using (2.66), Definition 1.44, and Theorem 1.41,

$$c_2 = a_{21} + a_{22} = b_1(1) + b_2(1) = 1.$$

□

The two-stage hybrid Runge-Kutta algorithms we consider all satisfy the assumptions in Table 2.2 and therefore, the conclusions of Theorem 2.12. Notice that all two-stage Runge Kutta methods which satisfy the assumptions of Table 2.2 form a one parameter family based on the coefficient c_1. That is, the hybrid algorithms we consider are of the form

$$\begin{array}{c|cc} c_1 & \frac{c_1(2-c_1)}{2(1-c_1)} & -\frac{c_1^2}{2(1-c_1)} \\ 1 & \frac{1}{2(1-c_1)} & \frac{1-2c_1}{2(1-c_1)} \\ \hline & b_1(\theta) & b_2(\theta) \end{array} \tag{2.68}$$

The last condition of Table 2.2, in conjunction with Theorem 2.4, implies that the two-stage hybrid algorithms we consider satisfies the linear system:

$$\begin{pmatrix} \Phi_1(\bullet) & \Phi_2(\bullet) \\ \Phi_1(\overset{\bullet}{\bullet}) & \Phi_2(\overset{\bullet}{\bullet}) \end{pmatrix} \begin{pmatrix} b_1(\theta) \\ b_2(\theta) \end{pmatrix} = \begin{pmatrix} \theta^{\rho(\bullet)}/\gamma(\bullet) \\ \theta^{\rho(\overset{\bullet}{\bullet})}/\gamma(\overset{\bullet}{\bullet}) \end{pmatrix}$$

which is equivalent to

$$\begin{pmatrix} 1 & 1 \\ c_1 & 1 \end{pmatrix} \begin{pmatrix} b_1(\theta) \\ b_2(\theta) \end{pmatrix} = \begin{pmatrix} \theta \\ \theta^2/2 \end{pmatrix}.$$

From Theorem 2.12, the square matrix on the left hand side of this equation is invertible so we arrive at the dense output coefficients

$$b_1(\theta) = \frac{2\theta - \theta^2}{2(1-c_1)} \qquad \text{and} \qquad b_2(\theta) = \frac{-2c_1\theta + \theta^2}{2(1-c_1)}.$$

Therefore, the all two-stage Runge-Kutta methods which satisfy the conditions

in Table 2.2 are of the form

$$\begin{array}{c|cc} c_1 & \frac{c_1(2-c_1)}{2(1-c_1)} & -\frac{c_1^2}{2(1-c_1)} \\ 1 & \frac{1}{2(1-c_1)} & \frac{1-2c_1}{2(1-c_1)} \\ \hline & \frac{2\theta-\theta^2}{2(1-c_1)} & \frac{-2c_1\theta+\theta^2}{2(1-c_1)} \end{array} \tag{2.69}$$

If we choose $c_1 = 1/8$ and use (2.69), we arrive at the two-stage Runge-Kutta method that will form the framework for the hybrid algorithm of order two.

$$\begin{array}{c|cc} 1/8 & 15/112 & -1/112 \\ 1 & 4/7 & 3/7 \\ \hline & (16\theta - 8\theta^2)/14 & (-\theta + 4\theta^2)/7 \end{array} \tag{2.70}$$

The eigenvalues associated to the coefficient matrix A of this method are $\mu_1 = (9 - \sqrt{17})/32 \approx 0.1524$ and $\mu_2 = (9 + \sqrt{17})/32 \approx 0.4101$. Algorithm 2.4 is the hybrid algorithm of order two based on this choice for c_1. Details regarding the choice of search direction p_k, the step length θ_k, and the stepsize control of $h_k = 1/\lambda_k$ are discussed in the following sections.

2.3.1 Simplified Newton Iterations

Similar to the derivation of the search direction of the one-stage hybrid algorithm, the search direction for the order two, two-stage hybrid algorithm is found by numerically solving

$$\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = -h(A \otimes I) \begin{pmatrix} \nabla f(x_k + z_1) \\ \nabla f(x_k + z_2) \end{pmatrix}. \tag{2.71}$$

Using the framework developed in Chapter 1.1.3, the Newton iteration scheme leads to two nonlinear systems of size $n \times n$, one for each eigenvalue of A. We approximate

Algorithm 2.4 Hybrid Algorithm of Order 2

Input: starting iterate x_0, termination tolerance $\epsilon > 0$,

inverse stepsize λ_0, and initial matrix $H_0(\lambda_0)$

Initialize $k \leftarrow 0$

while $||\nabla f(x_k)|| > \epsilon$

Compute internal stages via Algorithm 2.5

If weak monotonicity conditions hold

Find acceptable θ_k along (2.81)

$x_{k+1} \leftarrow x_k + \theta_k Z_{1,k}$

else

Find acceptable θ_k along (2.77)

$x_{k+1} \leftarrow x_{h_k}(\theta_k)$

endif

Update $s_{i,k}$, and $y_{i,k}$ via (2.75)

Update h_k via (2.85)

$k \leftarrow k + 1$

endwhile

Output: x_k

the solution of (2.71) by the Newton iteration scheme

$$\Delta W_{i,k}^{(j)} = -\frac{1}{\mu_i}\left(\frac{\lambda_k}{\mu_i}I + J\right)^{-1}\left(\lambda_k W_{i,k}^{(j)} + v_{i,k}^{(j)}\right)$$

for each $i = 1, 2$ and where $\Delta W_{i,k}^{(j)} = (T^{-1} \otimes I)\Delta Z_{i,k}^{(j)}$, $W_{i,k}^{(j)} = (T^{-1} \otimes I)Z_{i,k}^{(j)}$, and $Z_{i,k}^{(j+1)} = Z_{i,k}^{(j)} + \Delta Z_{i,k}^{(j)}$. Approximating $\left(\frac{\lambda_k}{\mu_i}I + J\right)^{-1}$ with the quasi-Newton matrix $H_{i,k}(\lambda_k)$, we have

$$\Delta W_{i,k}^{(j)} = -\frac{1}{\mu_i}H_{i,k}(\lambda_k)\left(\lambda_k W_{i,k}^{(j)} + v_{i,k}^{(j)}\right) \tag{2.72}$$

where

$$\begin{pmatrix} v_{1,k}^{(j)} \\ \\ v_{2,k}^{(j)} \end{pmatrix} = (T^{-1}A \otimes I)\hat{F}(Z^{(j)})$$

and

$$\hat{F}(Z^{(j)}) = (\nabla f(x_0 + z_1^{(j)}), \nabla f(x_0 + z_2^{(j)}), \ldots, \nabla f(x_0 + z_s^{(j)}))^T.$$

This scheme is used to find a suitable curve at step k. Using the equation $T^{-1}A = \Lambda T^{-1}$,

$$v_{i,k}^{(j)} = \mu_i\left[(T^{-1})_{i1}\nabla f(x_k + z_{1,k}^{(j)}) + (T^{-1})_{i2}\nabla f(x_k + z_{2,k}^{(j)})\right]$$

As with the one-stage hybrid algorithm, we take $x_k^{(0)} = x_k$, or equivalently, $W_{i,k}^{(0)} = 0 \in \mathbb{R}^n$ for $i = 1, 2$. Therefore, if only one Newton iteration is performed, we have

$$\begin{aligned}
\Delta W_{i,k}^{(1)} &= -\frac{1}{\mu_i}H_{i,k}(\lambda_k)\left(\lambda_k W_{i,k}^{(0)} + v_{i,k}^{(0)}\right) \\
&= -\frac{1}{\mu_i}H_{i,k}(\lambda_k)\mu_i\left[(T^{-1})_{i1}\nabla f(x_k) + (T^{-1})_{i2}\nabla f(x_k)\right] \qquad (2.73) \\
&= -((T^{-1})_{i1} + (T^{-1})_{i2})H_{i,k}(\lambda_k)\nabla f(x_k)
\end{aligned}$$

for each $i = 1, 2$. Therefore,

$$\begin{aligned} Z_{i,k}^{(1)} &= \Delta Z_{i,k}^{(1)} \\ &= -\left[T_{i1}((T^{-1})_{11} + (T^{-1})_{12})H_{1,k}(\lambda_k) + T_{i2}((T^{-1})_{21} + (T^{-1})_{22})H_{2,k}(\lambda_k)\right] \nabla f(x_k) \\ &= -\frac{1}{\det(T)}\left[T_{i1}(T_{22} - T_{12})H_{1,k}(\lambda_k) + T_{i2}(T_{11} - T_{21})H_{2,k}(\lambda_k)\right] \nabla f(x_k). \end{aligned} \tag{2.74}$$

The matrix vector products in (2.74) are calculated using the LBFGS Hybrid Two Loop Recursion Algorithm 2.1 and as with one-stage hybrid algorithm, we perform a low number of Newton iterations (one if possible). Notice that under one Newton iteration, only one gradient evaluation is needed. Once an acceptable value of $Z_{i,k}^{(j)}$ has been calculated, we set $Z_{i,k} = Z_{i,k}^{(j)}$.

The presence of the internal stage $Z_{i,k} = X_{i,k} - x_k$ give rise to two quasi-Newton updates per step:

$$\begin{aligned} s_{1,k} &= X_{1,k}^{(j)} - x_k, & s_{2,k} &= X_{2,k}^{(j)} - X_{1,k}^{(j)}, \\ y_{1,k} &= \nabla f(X_{1,k}^{(j)}) - \nabla f(x_k), & y_{2,k} &= \nabla f(X_{2,k}^{(j)}) - \nabla f(X_{1,k}^{(j)}), \end{aligned} \tag{2.75}$$

This is based on the fact that the two quasi-Newton matrices $H_{i,k}(\lambda)$, $i = 1, 2$ satisfy the secant conditions (2.64). If more than one Newton iteration is performed, we can use the calculated internal stages $X_{i,k}^{(j)}$ and their respective gradients to further update the limited memory vectors $\{s_{i,k}\}$ and $\{y_{i,k}\}$. The algorithm for calculating the internal stages is shown in Algorithm 2.5.

The following definition will be useful when discussing curve searches versus line searches in the following section.

Definition 2.13. Let $Z_{i,k}$, $i = 1, 2$, be numerical approximations to the solution of (2.71) and let $f(x)$ be the corresponding objective function. The *monotonicity*

conditions at step k of Algorithm 2.4 are

$$f(x_k) > f(x_k + Z_{1,k}) > f(x_k + Z_{2,k}). \tag{2.76}$$

If only the left inequality is satisfied, we say that the *weak monotonicity condition* at step k is satisfied.

Algorithm 2.5 Order Two Numerical Integration Algorithm

Input: iterate x_0, Newton guess $x_{k+1}^{(0)}$,

stepsize $h_k > 0$, and limited memory vectors $\{s_{i,k}\}$ and $\{y_{i,k}\}$

Initialize: $descent \leftarrow 0$, $j \leftarrow 0$

while $descent$=0

Compute internal stages via (2.74),

If $Z_{i,k}^{(j)}$ satisfy the weak monotonicity or the monotonicity conditions

$descent \leftarrow 1$,

else

Decrease h_k, $j \leftarrow j + 1$,

endif

endwhile

Output: $Z_{i,k}$

2.3.2 Dense Output Curve Searches

When Newton iterations are used to approximately solve the nonlinear system, we can use the two internal stages to build a dense output approximation to the gradient flow from x_k. In the case of two-stage Runge-Kutta algorithms which satisfy the assumptions in Table 2.2, the coefficients $b_1(\theta)$ and $b_2(\theta)$ are given by

$$b_1(\theta) = \frac{2\theta - \theta^2}{2(1-c_1)} \qquad \text{and} \qquad b_2(\theta) = \frac{-2c_1\theta + \theta^2}{2(1-c_1)}.$$

Using these coefficients, the dense output curve is

$$x_{h_k}(\theta) = x_k + d_1(\theta)Z_{1,k} + d_2(\theta)Z_{2,k} \tag{2.77}$$

where the subscript h_k denotes stepsize dependence of $Z_{i,k}$ and

$$\begin{aligned}(d_1(\theta), d_2(\theta)) &= (b_1(\theta), b_2(\theta))A^{-1} \\ &= \frac{1}{2\det(A)(1-c_1)} \begin{pmatrix} \theta - \theta^2, & \theta^2 c_1 - \theta c_1^2 \end{pmatrix}.\end{aligned}$$

Now we show that under the choice $x_k^{(0)} = x_k$, the search curve defined by (2.77) has the property that $\dot{x}_{h_k}(0)$ is a descent direction for f at x_k.

Lemma 2.14. *Let*

$$K = \frac{-(T_{12} - c_1^2 T_{22})(T_{11} - T_{21})}{(T_{11} - c_1^2 T_{21})(T_{22} - T_{12})}$$

(for $c_1 = 1/8$, $K \approx 7.240799963$). If

$$\nabla f(x_k)^T H_{1,k}(\lambda_k) \nabla f(x_k) < K \cdot \nabla f(x_k)^T H_{2,k}(\lambda_k) \nabla f(x_k), \tag{2.78}$$

and one Newton iteration is performed at each step with choice $x_k^{(0)} = x_k$, then $\dot{x}_{h_k}(0)$ is a descent direction for f at x_k.

Proof.

$$\begin{aligned}
\nabla f(x_k)^T \dot{x}_{h_k}(0) &= \frac{1}{2\det(A)(1-c_1)}(\dot{d}_1(0)\nabla f(x_k)^T Z_{1,k}^{(1)} + \dot{d}_2(0)\nabla f(x_k)^T Z_{2,k}^{(1)}) \\
&= \frac{1}{2\det(A)(1-c_1)}(\nabla f(x_k)^T Z_{1,k}^{(1)} - c_1^2 \nabla f(x_k)^T Z_{2,k}^{(1)}) \\
&= c[T_{11}(T_{22}-T_{12})\nabla f(x_k)^T H_{1,k}(\lambda_k)\nabla f(x_k) \\
&\quad + T_{12}(T_{11}-T_{21})\nabla f(x_k)^T H_{2,k}(\lambda_k)\nabla f(x_k) \\
&\quad - c_1^2 T_{21}(T_{22}-T_{12})\nabla f(x_k)^T H_{1,k}(\lambda_k)\nabla f(x_k) \\
&\quad - c_1^2 T_{22}(T_{11}-T_{21})\nabla f(x_k)^T H_{2,k}(\lambda_k)\nabla f(x_k)] \\
&= c[(T_{11}-c_1^2 T_{21})(T_{22}-T_{12})\nabla f(x_k)^T H_{1,k}(\lambda_k)\nabla f(x_k) \\
&\quad + (T_{12}-c_1^2 T_{22})(T_{11}-T_{21})\nabla f(x_k)^T H_{2,k}(\lambda_k)\nabla f(x_k)]
\end{aligned}$$

where $c = -1/(2\det(A)\det(T)(1-c_1)$. For $c_1 = 1/8$, we have $c > 0$ and

$$\nabla f(x_k)^T \dot{x}_{h_k}(0) \approx c[\kappa_1 \nabla f(x_k)^T H_{1,k}(\lambda_k)\nabla f(x_k) + \kappa_2 \nabla f(x_k)^T H_{2,k}(\lambda_k)\nabla f(x_k)]$$

where

$$\kappa_1 = (T_{11}-c_1^2 T_{21})(T_{22}-T_{12}) \approx 0.06401082157$$

$$\kappa_2 = (T_{12}-c_1^2 T_{22})(T_{11}-T_{21}) \approx -0.4634895545.$$

Dividing $-\kappa_2/\kappa_1$ produces the value K and therefore, $\nabla f(x_k)^T \dot{x}_{h_k}(0) < 0$ when

$$\nabla f(x_k)^T H_{1,k}(\lambda_k)\nabla f(x_k) < K \cdot \nabla f(x_k)^T H_{2,k}(\lambda_k)\nabla f(x_k).$$

□

As with the one-stage hybrid algorithm, we choose $H_{i,k}^{(0)}(\lambda_k) = \gamma_{i,k}(\lambda_k)I$. Since we have control on these matrices, it is always possible to choose $H_{1,k}^{(0)}(\lambda_k)$ with norm small enough so that (2.78) is always satisfied.

We search for a point along the curve (2.77) satisfying the generalized Wolfe or the generalized Goldstein and Price conditions. Let

$$m(\theta) := f(x_{h_k}(\theta)).$$

Definition 2.15. A step length θ_k satisfies the *generalized Wolfe conditions* if

$$m(\theta_k) \leq m(0) + \gamma_1 \theta_k m'(0) \tag{2.79}$$

$$m'(\theta_k) \geq \gamma_2 m'(0) \tag{2.80}$$

where γ_1 and γ_2 are constants satisfying $0 < \gamma_1 < \gamma_2 < 1$.

Notice that if $x_{h_k}(\theta) = x_k + h_k \theta p_k$ where p_k is a descent direction for f at x_k independent of θ, the standard Wolfe conditions are recovered. Since the objective function along the true solution to the gradient flow at x_k is strictly decreasing (assuming x_k is not an equilibrium point), the monotonicity conditions (2.76) are used to evaluate how well $Z_{1,k}$ and $Z_{2,k}$ follow the gradient flow. There are three cases to consider.

Case One: The internal stages satisfy the monotonicity conditions (see Definition 2.13). In this case, the two stages appear to be reasonable approximations to the gradient flow from x_k and we use the dense output curve (2.77) to search for x_{k+1} satisfying the generalized Wolfe conditions.

Case Two: The internal stages satisfy the weak monotonicity condition. In this case, the second internal stage should not be used and we perform the line search

$$x_{h_k}(\theta) = x_k + h_k \theta Z_{1,k} \tag{2.81}$$

and accept any θ_k which satisfies the standard Wolfe or Goldstein and Price conditions. The fact that $Z_{1,k}$ is a descent direction is shown in Lemma 2.16.

Case Three: The internal stages do not satisfy the weak monotonicity conditions. In this case, the internal stages are not used in a line or curve search. Instead, the stepsize h_k is decreased and the system (2.71) is recalculated and the monotonicity conditions are rechecked.

Lemma 2.16. *Let*

$$K = -\frac{T_{12}(T_{11} - T_{21})}{T_{11}(T_{22} - T_{12})}$$

(for $c_1 = 1/8$, $K \approx 10.40388203$). $Z_{1,k}$ is a descent direction for f at x_k if the

$$\nabla f(x_k)^T H_{1,k}(\lambda_k) \nabla f(x_k) < K \cdot \nabla f(x_k)^T H_{2,k}(\lambda_k) \nabla f(x_k).$$

Proof.

$$\begin{aligned}
\nabla f(x_k)^T Z_{1,k} = \frac{-1}{\det(T)} [&T_{11}(T_{22} - T_{12}) \nabla f(x_k)^T H_{1,k}(\lambda_k) \nabla f(x_k) \\
&+ T_{12}(T_{11} - T_{21}) \nabla f(x_k)^T H_{2,k}(\lambda_k) \nabla f(x_k)] \\
\approx \frac{-1}{\det(T)} [&\kappa_1 \cdot \nabla f(x_k)^T H_{1,k}(\lambda_k) \nabla f(x_k) + \kappa_2 \cdot \nabla f(x_k)^T H_{2,k}(\lambda_k) \nabla f(x_k)]
\end{aligned}$$

where

$$\kappa_1 = T_{11}(T_{22} - T_{12}) \approx 0.04315448277$$

$$\kappa_2 = T_{12}(T_{11} - T_{21}) \approx -0.4489741479.$$

Note that $\det(T) < 0$. K is found by dividing $-\kappa_2/\kappa_1$. Therefore, $\nabla f(x_k)^T Z_{1,k} < 0$ when

$$\nabla f(x_k)^T H_{1,k}(\lambda_k) \nabla f(x_k) < K \cdot \nabla f(x_k)^T H_{2,k}(\lambda_k) \nabla f(x_k).$$

□

The choice of initial θ_k depends on if a curve search or a line search is performed. If the weak monotonicity conditions hold and a line search is performed in the direction $Z_{1,k}$, then we take $\theta_k = 1$ as with the initial line search of the one-stage hybrid algorithm. However, if the monotonicity conditions hold, than a curve search is performed and an initial value for θ_k can be predicted using Lyapunov functions.

Definition 2.17. A function $V : \mathbb{R}^n \to \mathbb{R}$ is a *Lyapunov function* for the differential equation $\dot{x} = g(x)$, $g : \mathbb{R}^n \to \mathbb{R}^n$ if

$$\frac{d}{dt}V(x) \leq 0.$$

If this inequality is strict outside the set of equilibrium points of f, then the Lyapunov function is a *strict Lyapunov function.*

For gradient systems, $V(x) = f(x)$ is a strict Lyapunov function since

$$\dot{V}(x) = \nabla f(x)^T(-\nabla f(x)) = -||\nabla f(x)||^2 < 0$$

when x is not an equilibrium point. The level curves of a Lyapunov function describe regions of stability for the differential equation; if a solution of the differential equation enters a region defined by a contour of $V(x)$, it will not leave this region.

Let $q(x) := -\nabla f(x)^T \nabla f(x)$. The internal stages $Z_{i,k}$ provide an approximation to the Lyapunov function at x_k:

$$V_{k+1}(\theta) := V(x_k) + h\sum_{i=1}^{s} b_i(\theta)q(Z_{i,k} + x_k).$$

This approximation is a quadratic in θ and has the minimizer

$$\theta_{\min} = \frac{c_1 q(Z_{2,k} + x_k) - q(Z_{1,k} + x_k)}{q(Z_{2,k} + x_k) - q(Z_{1,k} + x_k)} = 1 - \frac{\frac{7}{8} q(Z_{2,k} + x_k)}{q(Z_{2,k} + x_k) - q(Z_{1,k} + x_k)}.$$

If this quantity is positive, we can use this value of θ_k as the initial guess for the Wolfe curve search. Otherwise, we take $\theta_k = 1$. However, for x_k sufficiently close to the minimizer, we expect the stepsize to go to infinity and the algorithm to behave in a similar fashion to the limited memory BFGS algorithm. Since the unit step length $\alpha_k = 1$ is always accepted near a local minimizer for the BFGS algorithm, we impose that for large stepsizes, $\theta_k = 1$ [7]; for example, if $h_k > 10$.

When the hybrid algorithm of Section 2.2.2 has trouble locating a point which satisfies the Wolfe or the Goldstein and Price conditions, the line search is dropped and additional Newton iterations are performed. Table 2.3 suggests that this strategy will not work for the two-stage hybrid algorithm. Additionally, each Newton iteration of the two-stage algorithm carries two additional gradient evaluations whereas each Newton iteration of the one-stage algorithm carries only one additional gradient evaluation. Therefore, we safeguard Algorithm 2.4 by switching to the one-stage algorithm whenever there is difficulty satisfying the Wolfe or the Goldstein and Price conditions. If the switch is performed, the convergence follows from the results of Section 2.2.4.

2.3.3 Stepsize Control

When the gradient system is stiff, rapid decreases in the stepsize may be required to effectively navigate the phase space [17]. Instead of decreasing λ_k by a

Table 2.3: Effect of Newton Iterations on Convergence (Algorithm 2.4)

Problem	One Iteration	Two Iterations	Ten Iterations
DIAG10	17/47/42	21/64/109	Diverge
PQUAD250	211/1143/968	145/655/925	Diverge
ROSENB2	29/87/89	39/124/208	Diverge
TRIDIA10	41/217/230	117/1103/1428	182/1955/5214
TRIG5	11/24/19	29/79/169	9/41/211

Results: iterations/function evaluations/gradient evaluations

constant factor at each iteration (hence increasing h_k by the inverse of this constant), stepsize control algorithms can be used to dynamically increase or decrease the stepsize at each iteration.

In a numerical ODE solver, algorithms rely on an estimate of the local error at each iteration to monitor the stepsize h_k. If a Runge-Kutta method is expressed with Butcher tableau (1.20), we can estimate the local error using the tableau

$$\begin{array}{c|c} c & A \\ \hline & b \\ \hline & \hat{b} \end{array} \tag{2.82}$$

where the coefficients c, A, and $\hat{b}$ represent an embedded RK algorithm of order strictly less than that of the method using coefficients c, A, and b. Using (1.18)

applied to this Butcher tableau we have,

$$x_{k+1} = x_k - h_k \sum_{j=1}^{s} b_j \nabla f(X_j)$$

$$\hat{x}_{k+1} = x_k - h_k \sum_{j=1}^{s} \hat{b}_j \nabla f(X_j).$$

Subtracting these two equations and setting $e_j := b_j - \hat{b}_j$ gives

$$x_{k+1} - \hat{x}_{k+1} = -h_k \sum_{j=1}^{s} e_j \nabla f(X_j). \tag{2.83}$$

Using Theorem 1.41, the local order, p, of the approximation x_{k+1} and the local order, $\hat{p} \leq p-1$ of the approximation $\hat{x}_{k+1}$ satisfy

$$1 = \gamma(t) \sum_{j=1}^{s} b_j \Phi_j(t)$$

$$1 = \gamma(t) \sum_{j=1}^{s} \hat{b}_j \Phi_j(t)$$

for all trees t of order less than or equal to p and $\hat{p}$ respectively. Therefore, the quantities e_j will produce a local error estimate which is $\mathcal{O}(h^{\hat{p}+1})$ if and only if

$$\gamma(t) \sum_{j=1}^{s} e_j \Phi_j(t) \begin{cases} = 0, & \text{for all trees } t \text{ with order no greater than } \hat{p} \\ \neq 0, & \text{for some tree with order } \hat{p}+1. \end{cases} \tag{2.84}$$

The quantity $\hat{r}_{k+1} = ||x_{k+1} - \hat{x}_{k+1}||$ is an estimate of the norm of the local error per iteration. Most stepsize control algorithms aim to make this quantity as large possible, but also subject to a specified tolerance, tol. The current step is accepted if $\hat{r}_{k+1} \leq \epsilon \cdot tol$ where $\epsilon \in (0,1)$ is a safety factor. If this inequality is not satisfied, the step is rejected and the stepsize for that step is decreased and the step is recomputed.

In the interest of keeping computational costs low, and since $\dot{x}_{h_k}(0)$ is chosen to be a descent direction, the control tolerance is greater than what is found in standard

controllers ($tol = 10^1$ for example). For the safety factor, we take $\epsilon = 0.8$. The choice of large tolerance is motivated by the fact that we only wish to loosely control the stepsize; we are not concerned with exactly following the gradient flow, but rather loosely following the flow in a quick and efficient manner.

For calculating the stepsize of the next iteration, h_{k+1}, one control mechanism is the PC.4.7 controller proposed by Gustafsson [16]:

$$h_{k+1} = \left(\frac{0.8 \cdot tol}{\hat{r}_{k+1}}\right)^{0.4/q} \left(\frac{\hat{r}_k}{\hat{r}_{k+1}}\right)^{0.7/q} \frac{h_k^2}{h_{k-1}} \tag{2.85}$$

where $q = p + 1$. Since this controller depends on the previous stepsize, it is not implemented until the second iteration. For the first iteration, the elementary controller, which depends only on the current stepsize, is used to control the stepsize. The elementary controller is

$$h_{k+1} = \left(\frac{0.8 \cdot tol}{\hat{r}_{k+1}}\right)^{1/q} h_k. \tag{2.86}$$

For the one-stage hybrid algorithm, Algorithm 2.2, $\hat{p} = 0$ and using (2.84), we have

$$0 \neq \gamma(\bullet) e_1 \Phi_j(\bullet) = e_1$$

which implies $b_1 \neq \hat{b}_1$. Therefore, for the one-stage hybrid algorithm, we set $\hat{b} = 0$. This results in the local error estimator which is $\mathcal{O}(h_k)$. This is not a very accurate approximation to the local error and is the motivation behind the choice (2.22).

For Algorithm 2.4, $\hat{p} = 1$. Therefore, we require

$$0 = \gamma(\bullet)\,(e_1\Phi_1(\bullet) + e_2\Phi_2(\bullet)) = e_1 + e_2 \tag{2.87}$$

$$0 \neq \gamma(\begin{smallmatrix}\bullet\\\bullet\end{smallmatrix})\,(e_1\Phi_1(\begin{smallmatrix}\bullet\\\bullet\end{smallmatrix}) + e_2\Phi_2(\begin{smallmatrix}\bullet\\\bullet\end{smallmatrix})) = 2e_1(a_{11}+a_{12}) + 2e_2(a_{21}+a_{22})) = 2c_1e_1 + 2e_2.$$

If we set $2c_1e_1 + 2e_2 = 1$, the solution to (2.87) is given by

$$\hat{b}_1 = \frac{1}{1-c_1} = \frac{8}{7} \qquad \text{and} \qquad \hat{b}_2 = \frac{-c_1}{1-c_1} = -\frac{1}{7}.$$

The values of $\hat{b}_1$ and $\hat{b}_2$ define $\hat{x}_{k+1}$ and, in turn, the estimate of the local error $\hat{r}_{k+1}$. This error satisfies $\mathcal{O}(h_k^2)$. For more information on stepsize controllers for Runge-Kutta algorithms, see [29–31].

2.3.4 Convergence of the Order Two Method

Convergence of line search methods relies on Theorem 1.22. Since Algorithm 2.4 uses a search curve instead of a search direction, this theorem is not applicable to the two-stage hybrid algorithm. The presence of the search curve, which is nonlinear in θ, makes finding a lower bound on θ_k at each iteration difficult. However, if only the weak monotonicity conditions hold at each step, a line search is performed in the direction $Z_{1,k}$ and therefore, Zoutendijk's Theorem 1.22 holds. In this case, the theorems of Section 2.2.4 hold for $Z_{1,k}$ and $H_{1,k}(\lambda)$. As such, we have the following convergence result,

Theorem 2.18. *Consider an implementation of Algorithm 2.4 where at every q^{th} iteration, a line search is imposed using the direction $Z_{1,k}$ with corresponding hybrid BFGS matrix $H_{1,k}(\lambda_k)$. Additionally, suppose that $f \in \mathcal{C}^2$ is bounded below,*

∇f is Lipschitz continuous, and the initial BFGS matrix, $H_{1,0}$, is positive definite. Then the sequence of iterates generated by Algorithm 2.4 has the property that $\liminf_{k\to\infty} ||\nabla f(x_k)|| = 0$. *Additionally, if f is strictly convex on the level set $\mathcal{L}_{x_0}$, then* $\lim_{k\to\infty} x_k = x^*$ *where x^* is the unique minimizer of f on $\mathcal{L}_{x_0}$.*

Proof. By the generalized Wolfe conditions, we always have $f(x_{k+1}) < f(x_k)$. Restrict the sequence of iterates generated by Algorithm 2.4 to the subsequence of iterates $\{x_{k_j}\}_{j=0}^{\infty}$ generated by performing the line search at every q^{th} iteration. This sequence satisfies the hypothesis of Theorem 2.5 and hence, the conclusions hold for the subsequence $\{x_{k_j}\}_{j=0}^{\infty}$. Therefore,

$$\liminf_{k\to\infty} ||\nabla f(x_k)|| = 0$$

and if f is strongly convex on $\mathcal{L}_{x_0}$, then the iterates converge to the unique minimizer of f on $\mathcal{L}_{x_0}$. □

In practice, when the iterates become close to a local minimizer, we expect the local quadratic model of the objective function to be fairly accurate and the stepsize of the hybrid method to become unbounded. Therefore, we impose that when the stepsize, h_k, becomes large, we switch to the order one hybrid method. This will reduce the number of gradient evaluations and preserve the property that the hybrid algorithm behaves like the LBFGS algorithm near a local minimizer. If this switch is made, the theorems from Section 2.2.4 apply and we expect superlinear convergence near the local minimizer.

2.4 Higher Order Hybrid Algorithms

Although we do not consider quasi-Newton and Runge-Kutta hybrid algorithms of degree greater than two, the framework for such algorithms is in place. We may impose that any hybrid algorithm of order greater than two satisfy the conditions

Table 2.4: Assumptions for s-Stage Hybrid Algorithms

1. A has real positive eigenvalues
2. Stage order s
3. Stiffly accurate and hence L-stable
4. Order s dense output

Given an s-stage Runge-Kutta method with satisfies the assumptions in Table 2.4, the corresponding hybrid algorithm is described as follows: At each iteration, an approximate solution to the nonlinear system (1.21), defined by the Newton iterations, if calculated. This produces s internal stages which can be used to define a search curve which is an s degree polynomial in θ.

$$x_{h_k}(\theta) = x_k + \sum_{j=1}^{s} d_j(\theta) Z_{j,k} \tag{2.88}$$

Provided that these internal stages are reasonable approximations to the gradient flow from the current iterate, select the next iterate on this search curve using the

Wolfe or Goldstein and Price conditions. Lastly, update the current stepsize using, for example, the controller (2.85).

Algorithm 2.6 General Hybrid Algorithm for Unconstrained Optimization

Inputs: starting point x_0, termination tolerance $\epsilon > 0$

inverse stepsize λ_0, and initial matrix $H_0(\lambda_0)$

$k \leftarrow 0$

while $||\nabla f(x_k)|| \geq \epsilon$

Compute internal stages of (1.24)

Find acceptable step length θ_k along (2.88)

$x_{k+1} \leftarrow x_{h_k}(\theta_k)$

Update $s_{i,k}$ and $y_{i,k}$ via (2.75)

Update h_k via (2.85)

$k \leftarrow k + 1$

endwhile

Output: x_k

CHAPTER 3
NUMERICAL RESULTS

In this chapter, we test the hybrid algorithms of Chapter 2 and the LBFGS algorithm on a set of 59 test problems taken from [4], [12], and [23]. The number of iterations, function evaluations, gradient evaluations, and computational time needed to achieve convergence to an equilibrium point is compared in Tables 3.2 through 3.10. An algorithm is said to converge if

$$||\nabla f(x_k)|| < tol \tag{3.1}$$

where the tolerance *tol* is taken to be 10^{-3}, 10^{-6}, or 10^{-9}.

Each table has a column for the function name, the problem name, and one for each of the algorithms being compared. Furthermore, each column representing a specific algorithm is divided into three subcolumns denoted by a '1','2', or 'f'. The '1' column represents first-place finishes, the '2' column represents second-place finishes, and the 'f' column represents a failed search. If the value 1 appears in the '1' column for a given algorithm, that signifies that that algorithm performed best given the performance metric in question (number of iterations, function evaluations, gradient evaluations, or computational time). There were occasionally ties between the algorithms, in which case, both algorithms have a 1 in that respective row.

The '2' column contains numbers greater than one and denotes second-place finishes. These numbers represent how many times worse, with regards to the metric in question, that algorithm performed when compared to the first-place algorithm.

For example, if the second-place algorithm required 25 iterations to achieve convergence on a specific problem whereas the first-place algorithm required only 10 for that same problem, the second-place algorithm is given a score of 2.50, which would appear in the '2' column. If an algorithm failed to solve a specific problem, an f is reported.

Totals are summarized at the bottom of each table. The first row of of this summary reports how many first-place or second-place finishes each algorithm achieved as well as how many times the algorithm failed to find an equilibrium point of the gradient system. The second row compares the LBFGS algorithm to the safeguarded hybrid algorithm. For this row, the totals from the hybrid algorithm and the safeguarded hybrid algorithm are added together to provide a comparison between the LBFGS algorithm and the full hybrid algorithm.

The test functions were implemented in Matlab 7.11.10.584 (R2010b) and executed on a 2.40 GHz HP desktop with 2.00 GB of RAM running Microsoft Windows 7 version 6.1. The test problems and their size are reported in Table 3.1.

3.1 Hybrid Algorithm of Order One

In this section, we compare the performance of Algorithm 2.2 without the safeguard discussed in Section 2.2.2 (HYBRID1) and the LBFGS algorithm. The safeguarded hybrid algorithm (HYBRID1S) will be used whenever HYBRID1 fails. Note that HYBRID1S and HYBRID1 are identical so long as a point which satisfies the Wolfe or Goldstein and Price conditions is found in a reasonable number of

Table 3.1: Test Functions for Unconstrained Minimization

Function	Problem	Size		Function	Problem	Size
BIGGS	BIGGS6	6		POWBSC	POWBSC2	2
BROWND	BROWND4	4		POWSNG	POWSNG4	4
DIAGA	DIAGA10	10		POWSNG	POWSNG100	100
DIAGA	DIAGA1000	1000		POWSNG	POWSNG500	500
EXTRSN	EXTRSN50	50		POWSNG	POWSNG1000	1000
EXTRSN	EXTRSN250	250		POWER	POWER5	5
EXTRSN	EXTRSN1000	1000		POWER	POWER30	30
EXTRSN	EXTRSN5000	5000		POWER	POWER100	100
EXTWD	EXTWD40	40		RAYDA	RAYDA10	10
EXTWD	EXTWD100	100		RAYDA	RAYDA100	100
EXTWD	EXTWD500	500		RAYDA	RAYDA1000	1000
EXTWD	EXTWD1000	1000		RAYDA	RAYDA5000	5000
HIMMBG	HIMMBG10	10		ROSENB	ROSENB2	2
LIARWHD	LWHD5	5		TRIDIA	TRIDIA10	10
LIARWHD	LWHD250	250		TRIDIA	TRIDIA500	500
LIARWHD	LWHD1000	1000		TRIDIA	TRIDIA1000	1000
LIARWHD	LWHD5000	5000		TRIG	TRIG5	5
NONSCOMP	NONSCP10	10		TRIG	TRIG50	50
NONSCOMP	NONSCP500	500		TRIG	TRIG100	100
NONSCOMP	NONSCP1000	1000		VARDIM	VARDIM10	10
NONSCOMP	NONSCP5000	5000		VARDIM	VARDIM100	100
NONSCOMP	NONSCP10000	10000		VARDIM	VARDIM500	500
PENALA	PENALA10	10		VARDIM	VARDIM1000	1000
PENALA	PENALA250	250		VARDIM	VARDIM5000	5000
PENALA	PENALA1000	1000		WOOD	WOOD4	4
PENALA	PENALA5000	5000		ZAKHAR	ZAKHAR50	50
PQUAD	PQUAD50	50		ZAKHAR	ZAKHAR250	250
PQUAD	PQUAD250	250		ZAKHAR	ZAKHAR1000	1000
PQUAD	PQUAD1000	1000		ZAKHAR	ZAKHAR5000	5000
PQUAD	PQUAD5000	5000				

iterates. Therefore, there is no need to compare HYBRID1 and HYBRID1S. The safeguarded algorithm demonstrates how the switch to an ODE solver can improve

robustness of the hybrid algorithm of order one.

For the LBFGS algorithm, the number of function and gradient evaluations are the same if the Wolfe conditions are used. This is also true of the HYBRID1 algoirthm. The Wolfe conditions were implemented to generate the results in Tables 3.2 through 3.10 so separate tables for function and gradient evaluation comparison is not necessary. There was only one instance where HYBRID1 failed and LBFGS did not (`TRIG10` with tolerance = 10^{-6}). In this case, a comparison between LBFGS and HYBRID1S was performed. Since the number of function and gradient evaluations of HYBRID1S are not the same, two numbers are reported in the HYBRID1S column and represent the performance based on function evaluations and gradient evaluations respectively.

Tables 3.2 through 3.10 show that, as the tolerance decreases, the number of problems that LBFGS and HYBRID1 are able to solve also decreases. Notice that HYBRID1 was able to solve at least as many problems as LBFGS and when the tolerance is 10^{-9}, the HYBRID1 algorithm is able to solve 41 of the test problems whereas LBFGS converges on 34 of the problems. When considering HYBRID1S, the safegaurded algorithm is able to solve all but two of the test problems, `TRIDIA500` and `TRIDIA1000` with tolerance 10^{-9}.

As tolerance decreases, the number of first-place finishes for LBFGS also decreases given any of the performance metrics whereas the number of first-place finishes for HYBRID1 increases as tolerance decreases. Additionally, many of the second-place finishes for both algorithms were within ten percent of each other.

Table 3.2: Place Finishes for the Number of Iterations, Tolerance = 10^{-3}.

Function	Problem	LBFGS			HYBRID1			HYBRID1S		
		1	2	f	1	2	f	1	2	f
BIGGS	BIGGS6		1.02		1					
BROWND	BROWND4	1				1.62				
DIAGA	DIAGA10	1				1.15				
DIAGA	DIAGA100	1				1.08				
EXTRSN	EXTRSN50		1.04		1					
EXTRSN	EXTRSN250		1.09		1					
EXTRSN	EXTRSN1000		1.09		1					
EXTRSN	EXTRSN5000		1.08		1					
EXTWD	EXTWD40	1				1.14				
EXTWD	EXTWD100	1				1.17				
EXTWD	EXTWD500	1				1.15				
EXTWD	EXTWD1000		1.67		1					
HIMMBG	HIMMBG10	1				3.14				
LIARWHD	LWHD5			f			f	1		
LIARWHD	LWHD250			f			f	1		
LIARWHD	LWHD1000			f			f	1		
LIARWHD	LWHD5000			f			f	1		
NONSCOMP	NONSCP10		1.18		1					
NONSCOMP	NONSP500		1.21		1					
NONSCOMP	NONSP1000		1.41		1					
NONSCOMP	NONSP5000	1				1.05				
NONSCOMP	NONSP10000		1.21		1					
PENALA	PENALA10	1				1.67				
PENALA	PENALA250	1				2.23				
PENALA	PENALA1000	1				2.42				
PENALA	PENALA5000	1				2.50				
PQUAD	PQUAD50	1				1.18				
PQUAD	PQUAD250	1				1.06				
PQUAD	PQUAD1000	1				1.03				
PQUAD	PQUAD5000	1				1.04				

Table 3.2. Continued.

POWBSC	POWBSC2			f			f	1			
POWSNG	POWSNG4			f			f	1			
POWSNG	POWSNG100			f			f	1			
POWSNG	POWSNG500			f			f	1			
POWSNG	POWSNG1000			f			f	1			
POWER	POWER5	1				1.07					
POWER	POWER30	1				1.01					
POWER	POWER100	1				1.01					
RAYDA	RAYDA10	1				1.18					
RAYDA	RAYDA100	1				1.05					
RAYDA	RAYDA1000	1				1.08					
RAYDA	RAYDA5000	1				1.17					
ROSENB	ROSENB2	1				1.16					
TRIDIA	TRIDIA10			f			f	1			
TRIDIA	TRIDIA500			f			f	1			
TRIDIA	TRIDIA1000			f			f	1			
TRIG	TRIG5		1.07		1						
TRIG	TRIG20		1.18		1						
TRIG	TRIG100	1			1						
VARDIM	VARDIM10	1			1						
VARDIM	VARDIM100	1				1.24					
VARDIM	VARDIM500		1.18		1						
VARDIM	VARDIM1000		1.02		1						
VARDIM	VARDIM5000			f			f	1			
WOOD	WOOD4	1				1.19					
ZAKHAR	ZAKHAR50	1				1.64					
ZAKHAR	ZAKHAR250	1				2.16					
ZAKHAR	ZAKHAR1000	1				2.10					
ZAKHAR	ZAKHAR5000	1				1.84					
Totals:		32	14	13	16	30	13	13	0	0	
		32	14	13	29			30	0		

Table 3.3: Place Finishes for the Number of Iterations, Tolerance = 10^{-6}.

Function	Problem	LBFGS			HYBRID1			HYBRID1S		
		1	2	f	1	2	f	1	2	f
BIGGS	BIGGS6	1				1.57				
BROWND	BROWND4			f			f	1		
DIAGA	DIAGA10	1				1.11				
DIAGA	DIAGA100			f			f	1		
EXTRSN	EXTRSN50		1.05		1					
EXTRSN	EXTRSN250		1.09		1					
EXTRSN	EXTRSN1000		1.08		1					
EXTRSN	EXTRSN5000		1.08		1					
EXTWD	EXTWD40			f	1					
EXTWD	EXTWD100	1				1.17				
EXTWD	EXTWD500	1				1.11				
EXTWD	EXTWD1000			f	1					
HIMMBG	HIMMBG10	1				2.53				
LIARWHD	LWHD5			f			f	1		
LIARWHD	LWHD250			f			f	1		
LIARWHD	LWHD1000			f			f	1		
LIARWHD	LWHD5000			f			f	1		
NONSCOMP	NONSCP10	1				1.05				
NONSCOMP	NONSP500		1.65		1					
NONSCOMP	NONSP1000		1.69		1					
NONSCOMP	NONSP5000		1.59		1					
NONSCOMP	NONSP10000		1.68		1					
PENALA	PENALA10	1				1.09				
PENALA	PENALA250	1				1.26				
PENALA	PENALA1000	1				1.68				
PENALA	PENALA5000	1				1.64				
PQUAD	PQUAD50		1.02		1					
PQUAD	PQUAD250	1				1.10				
PQUAD	PQUAD1000	1				1.12				
PQUAD	PQUAD5000	1				1.19				

Table 3.3. Continued.

POWBSC	POWBSC2			f			f	1			
POWSNG	POWSNG4			f			f	1			
POWSNG	POWSNG100			f			f	1			
POWSNG	POWSNG500			f			f	1			
POWSNG	POWSNG1000			f			f	1			
POWER	POWER5		1.12		1						
POWER	POWER30	1				1.11					
POWER	POWER100	1				1.08					
RAYDA	RAYDA10	1				1.19					
RAYDA	RAYDA100	1				1.14					
RAYDA	RAYDA1000	1				1.15					
RAYDA	RAYDA5000	1				1.25					
ROSENB	ROSENB2	1				1.15					
TRIDIA	TRIDIA10			f			f	1			
TRIDIA	TRIDIA500			f			f	1			
TRIDIA	TRIDIA1000			f			f	1			
TRIG	TRIG5	1					f		1.64		
TRIG	TRIG20			f			f	1			
TRIG	TRIG100			f			f	1			
VARDIM	VARDIM10	1			1						
VARDIM	VARDIM100		1.06		1						
VARDIM	VARDIM500		1.15		1						
VARDIM	VARDIM1000	1				1.03					
VARDIM	VARDIM5000			f			f	1			
WOOD	WOOD4	1				1.27					
ZAKHAR	ZAKHAR50	1				1.59					
ZAKHAR	ZAKHAR250	1				2.12					
ZAKHAR	ZAKHAR1000	1				2.08					
ZAKHAR	ZAKHAR5000	1				1.92					
Totals:		28	12	19	15	26	18	17	1	0	
		28	12	19	32			27	0		

Table 3.4: Place Finishes for the Number of Iterations, Tolerance = 10^{-9}.

Function	Problem	LBFGS			HYBRID1			HYBRID1S		
		1	2	f	1	2	f	1	2	f
BIGGS	BIGGS6			f	1					
BROWND	BROWND4			f			f	1		
DIAGA	DIAGA10	1				1.04				
DIAGA	DIAGA100			f			f	1		
EXTRSN	EXTRSN50		1.04		1					
EXTRSN	EXTRSN250		1.09		1					
EXTRSN	EXTRSN1000		1.08		1					
EXTRSN	EXTRSN5000		1.08		1					
EXTWD	EXTWD40			f	1					
EXTWD	EXTWD100			f	1					
EXTWD	EXTWD500			f	1					
EXTWD	EXTWD1000			f	1					
HIMMBG	HIMMBG10	1				2.33				
LIARWHD	LWHD5			f			f	1		
LIARWHD	LWHD250			f			f	1		
LIARWHD	LWHD1000			f			f	1		
LIARWHD	LWHD5000			f			f	1		
NONSCOMP	NONSCP10	1				1.04				
NONSCOMP	NONSP500		1.86		1					
NONSCOMP	NONSP1000		1.94		1					
NONSCOMP	NONSP5000		1.77		1					
NONSCOMP	NONSP10000		1.58		1					
PENALA	PENALA10	1				1.05				
PENALA	PENALA250	1				1.25				
PENALA	PENALA1000	1				1.64				
PENALA	PENALA5000	1				1.63				
PQUAD	PQUAD50	1				1.07				
PQUAD	PQUAD250	1				1.01				
PQUAD	PQUAD1000		1.02		1					
PQUAD	PQUAD5000	1				1.04				

Table 3.4. Continued.

POWBSC	POWBSC2			f			f	1			
POWSNG	POWSNG4			f			f	1			
POWSNG	POWSNG100			f			f	1			
POWSNG	POWSNG500			f			f	1			
POWSNG	POWSNG1000			f			f	1			
POWER	POWER5	1				1.04					
POWER	POWER30		1.04		1						
POWER	POWER100		1.01		1						
RAYDA	RAYDA10	1				1.08					
RAYDA	RAYDA100	1				1.11					
RAYDA	RAYDA1000	1				1.06					
RAYDA	RAYDA5000	1				1.03					
ROSENB	ROSENB2	1				1.14					
TRIDIA	TRIDIA10			f			f	1			
TRIDIA	TRIDIA500			f			f				f
TRIDIA	TRIDIA1000			f			f				f
TRIG	TRIG5			f			f	1			
TRIG	TRIG20			f			f	1			
TRIG	TRIG100			f			f	1			
VARDIM	VARDIM10	1			1						
VARDIM	VARDIM100	1				1.24					
VARDIM	VARDIM500			f	1						
VARDIM	VARDIM1000			f	1						
VARDIM	VARDIM5000			f			f	1			
WOOD	WOOD4	1				1.27					
ZAKHAR	ZAKHAR50	1				1.62					
ZAKHAR	ZAKHAR250	1				2.13					
ZAKHAR	ZAKHAR1000	1				2.25					
ZAKHAR	ZAKHAR5000	1				2.26					
Totals:		23	11	25	19	22	18	16	0	2	
		23	11	25	35			22	2		

Table 3.5: Place Finishes for the Number of Function and Gradient Evaluations, Tolerance = 10^{-3}.

Function	Problem	LBFGS			HYBRID1			HYBRID1S		
		1	2	f	1	2	f	1	2	f
BIGGS	BIGGS6	1				1.17				
BROWND	BROWND4	1				1.43				
DIAGA	DIAGA10	1			1					
DIAGA	DIAGA100	1				1.13				
EXTRSN	EXTRSN50		1.07		1					
EXTRSN	EXTRSN250		1.11		1					
EXTRSN	EXTRSN1000		1.12		1					
EXTRSN	EXTRSN5000		1.12		1					
EXTWD	EXTWD40	1				1.12				
EXTWD	EXTWD100	1				1.04				
EXTWD	EXTWD500	1				1.13				
EXTWD	EXTWD1000		1.58		1					
HIMMBG	HIMMBG10	1				2.67				
LIARWHD	LWHD5			f			f	1		
LIARWHD	LWHD250			f			f	1		
LIARWHD	LWHD1000			f			f	1		
LIARWHD	LWHD5000			f			f	1		
NONSCOMP	NONSCP10		1.13		1					
NONSCOMP	NONSP500		1.28		1					
NONSCOMP	NONSP1000		1.37		1					
NONSCOMP	NONSP5000	1				1.11				
NONSCOMP	NONSP10000		1.15		1					
PENALA	PENALA10	1				1.42				
PENALA	PENALA250	1				2.68				
PENALA	PENALA1000	1				3.58				
PENALA	PENALA5000	1				4.46				
PQUAD	PQUAD50	1				1.15				
PQUAD	PQUAD250	1				1.04				
PQUAD	PQUAD1000	1				1.02				

Table 3.5. Continued.

PQUAD	PQUAD5000	1				1.06					
POWBSC	POWBSC2			f			f	1			
POWSNG	POWSNG4			f			f	1			
POWSNG	POWSNG100			f			f	1			
POWSNG	POWSNG500			f			f	1			
POWSNG	POWSNG1000			f			f	1			
POWER	POWER5	1			1						
POWER	POWER30		1.01		1						
POWER	POWER100	1				1.01					
RAYDA	RAYDA10	1				1.15					
RAYDA	RAYDA100	1				1.04					
RAYDA	RAYDA1000	1				1.08					
RAYDA	RAYDA5000	1				1.17					
ROSENB	ROSENB2	1				1.10					
TRIDIA	TRIDIA10			f			f	1			
TRIDIA	TRIDIA500			f			f	1			
TRIDIA	TRIDIA1000			f			f	1			
TRIG	TRIG5		1.16		1						
TRIG	TRIG20	1				1.38					
TRIG	TRIG100	1			1						
VARDIM	VARDIM10	1			1						
VARDIM	VARDIM100	1				1.10					
VARDIM	VARDIM500		1.01		1						
VARDIM	VARDIM1000		1.01		1						
VARDIM	VARDIM5000			f			f	1			
WOOD	WOOD4	1				1.16					
ZAKHAR	ZAKHAR50	1				1.31					
ZAKHAR	ZAKHAR250	1				2.26					
ZAKHAR	ZAKHAR1000	1				2.70					
ZAKHAR	ZAKHAR5000	1				2.88					
Totals:		33	13	13	17	29	13	13	0	0	
		33	13	13	30			29	0		

Table 3.6: Place Finishes for the Number of Function and Gradient Evaluations, Tolerance = 10^{-6}.

Function	Problem	LBFGS			HYBRID1			HYBRID1S		
		1	2	f	1	2	f	1	2	f
BIGGS	BIGGS6	1				2.08				
BROWND	BROWND4			f			f	1		
DIAGA	DIAGA10	1			1					
DIAGA	DIAGA100			f			f	1		
EXTRSN	EXTRSN50		1.07		1					
EXTRSN	EXTRSN250		1.11		1					
EXTRSN	EXTRSN1000		1.11		1					
EXTRSN	EXTRSN5000		1.12		1					
EXTWD	EXTWD40			f	1					
EXTWD	EXTWD100	1				1.07				
EXTWD	EXTWD500	1				1.10				
EXTWD	EXTWD1000			f	1					
HIMMBG	HIMMBG10	1				2.37				
LIARWHD	LWHD5			f			f	1		
LIARWHD	LWHD250			f			f	1		
LIARWHD	LWHD1000			f			f	1		
LIARWHD	LWHD5000			f			f	1		
NONSCOMP	NONSCP10	1				1.05				
NONSCOMP	NONSP500		1.65		1					
NONSCOMP	NONSP1000		1.65		1					
NONSCOMP	NONSP5000		1.53		1					
NONSCOMP	NONSP10000		1.67		1					
PENALA	PENALA10	1				1.05				
PENALA	PENALA250	1				1.45				
PENALA	PENALA1000	1				2.32				
PENALA	PENALA5000	1				2.78				
PQUAD	PQUAD50		1.03		1					
PQUAD	PQUAD250	1				1.10				
PQUAD	PQUAD1000	1				1.11				

Table 3.6. Continued.

PQUAD	PQUAD5000	1				1.19				
POWBSC	POWBSC2			f			f	1		
POWSNG	POWSNG4			f			f	1		
POWSNG	POWSNG100			f			f	1		
POWSNG	POWSNG500			f			f	1		
POWSNG	POWSNG1000			f			f	1		
POWER	POWER5		1.13		1					
POWER	POWER30	1				1.08				
POWER	POWER100	1				1.12				
RAYDA	RAYDA10	1				1.24				
RAYDA	RAYDA100	1				1.17				
RAYDA	RAYDA1000	1				1.15				
RAYDA	RAYDA5000	1				1.24				
ROSENB	ROSENB2	1				1.09				
TRIDIA	TRIDIA10			f			f	1		
TRIDIA	TRIDIA500			f			f	1		
TRIDIA	TRIDIA1000			f			f	1		
TRIG	TRIG5	1					f		2.07/3.17	
TRIG	TRIG20			f			f	1		
TRIG	TRIG100			f			f	1		
VARDIM	VARDIM10	1			1					
VARDIM	VARDIM100	1				1.05				
VARDIM	VARDIM500	1				1.01				
VARDIM	VARDIM1000	1				1.03				
VARDIM	VARDIM5000			f			f	1		
WOOD	WOOD4	1				1.27				
ZAKHAR	ZAKHAR50	1				1.30				
ZAKHAR	ZAKHAR250	1				2.24				
ZAKHAR	ZAKHAR1000	1				2.69				
ZAKHAR	ZAKHAR5000	1				2.96				
Totals:		30	10	19	14	27	18	17	1/1	0
		30	10	19		31		28/28	0	

Table 3.7: Place Finishes for the Number of Function and Gradient Evaluations, Tolerance = 10^{-9}.

Function	Problem	LBFGS			HYBRID1			HYBRID1S		
		1	2	f	1	2	f	1	2	f
BIGGS	BIGGS6			f	1					
BROWND	BROWND4			f			f	1		
DIAGA	DIAGA10	1				1.09				
DIAGA	DIAGA100			f			f	1		
EXTRSN	EXTRSN50		1.07		1					
EXTRSN	EXTRSN250		1.12		1					
EXTRSN	EXTRSN1000		1.12		1					
EXTRSN	EXTRSN5000		1.12		1					
EXTWD	EXTWD40			f	1					
EXTWD	EXTWD100			f	1					
EXTWD	EXTWD500			f	1					
EXTWD	EXTWD1000			f	1					
HIMMBG	HIMMBG10	1				2.24				
LIARWHD	LWHD5			f			f	1		
LIARWHD	LWHD250			f			f	1		
LIARWHD	LWHD1000			f			f	1		
LIARWHD	LWHD5000			f			f	1		
NONSCOMP	NONSCP10	1				1.04				
NONSCOMP	NONSP500		1.82		1					
NONSCOMP	NONSP1000		1.90		1					
NONSCOMP	NONSP5000		1.74		1					
NONSCOMP	NONSP10000		1.56		1					
PENALA	PENALA10	1				1.02				
PENALA	PENALA250	1				1.43				
PENALA	PENALA1000	1				2.28				
PENALA	PENALA5000	1				2.76				
PQUAD	PQUAD50	1				1.07				
PQUAD	PQUAD250	1				1.02				
PQUAD	PQUAD1000		1.02		1					

Table 3.7. Continued.

PQUAD	PQUAD5000	1				1.04					
POWBSC	POWBSC2			f			f	1			
POWSNG	POWSNG4			f			f	1			
POWSNG	POWSNG100			f			f	1			
POWSNG	POWSNG500			f			f	1			
POWSNG	POWSNG1000			f			f	1			
POWER	POWER5	1			1						
POWER	POWER30		1.04		1						
POWER	POWER100		1.02		1						
RAYDA	RAYDA10	1				1.11					
RAYDA	RAYDA100	1				1.12					
RAYDA	RAYDA1000	1				1.07					
RAYDA	RAYDA5000	1				1.03					
ROSENB	ROSENB2	1				1.09					
TRIDIA	TRIDIA10			f			f	1			
TRIDIA	TRIDIA500			f			f				f
TRIDIA	TRIDIA1000			f			f				f
TRIG	TRIG5			f			f	1			
TRIG	TRIG20			f			f	1			
TRIG	TRIG100			f			f	1			
VARDIM	VARDIM10	1				1.03					
VARDIM	VARDIM100	1				1.10					
VARDIM	VARDIM500			f	1						
VARDIM	VARDIM1000			f	1						
VARDIM	VARDIM5000			f			f	1			
WOOD	WOOD4	1				1.27					
ZAKHAR	ZAKHAR50	1				1.31					
ZAKHAR	ZAKHAR250	1				2.25					
ZAKHAR	ZAKHAR1000	1				2.83					
ZAKHAR	ZAKHAR5000	1				3.11					
Totals:		23	11	25	19	22	18	16	0	2	
		23	11	25	35			22	2		

Table 3.8: Place Finishes for Computational Time, Tolerance = 10^{-3}.

Function	Problem	LBFGS			HYBRID1			HYBRID1S		
		1	2	f	1	2	f	1	2	f
BIGGS	BIGGS6		1.24		1					
BROWND	BROWND4	1				1.37				
DIAGA	DIAGA10	1				1.10				
DIAGA	DIAGA100	1				1.32				
EXTRSN	EXTRSN50		1.03		1					
EXTRSN	EXTRSN250		1.05		1					
EXTRSN	EXTRSN1000		1.04		1					
EXTRSN	EXTRSN5000	1				1.04				
EXTWD	EXTWD40	1				1.13				
EXTWD	EXTWD100	1				1.20				
EXTWD	EXTWD500	1				1.25				
EXTWD	EXTWD1000		1.60		1					
HIMMBG	HIMMBG10	1				3.21				
LIARWHD	LWHD5			f			f	1		
LIARWHD	LWHD250			f			f	1		
LIARWHD	LWHD1000			f			f	1		
LIARWHD	LWHD5000			f			f	1		
NONSCOMP	NONSCP10		1.13		1					
NONSCOMP	NONSP500		1.15		1					
NONSCOMP	NONSP1000		1.27		1					
NONSCOMP	NONSP5000	1				1.17				
NONSCOMP	NONSP10000		1.06		1					
PENALA	PENALA10	1				1.73				
PENALA	PENALA250	1				2.00				
PENALA	PENALA1000	1				2.61				
PENALA	PENALA5000	1				2.51				
PQUAD	PQUAD50	1				1.27				
PQUAD	PQUAD250	1				1.08				
PQUAD	PQUAD1000	1				1.07				
PQUAD	PQUAD5000	1				1.14				

Table 3.8. Continued.

POWBSC	POWBSC2			f			f	1			
POWSNG	POWSNG4			f			f	1			
POWSNG	POWSNG100			f			f	1			
POWSNG	POWSNG500			f			f	1			
POWSNG	POWSNG1000			f			f	1			
POWER	POWER5	1				1.08					
POWER	POWER30		1.10		1						
POWER	POWER100	1				1.08					
RAYDA	RAYDA10	1				1.20					
RAYDA	RAYDA100	1			1						
RAYDA	RAYDA1000	1				1.25					
RAYDA	RAYDA5000	1				1.23					
ROSENB	ROSENB2	1				1.15					
TRIDIA	TRIDIA10			f			f	1			
TRIDIA	TRIDIA500			f			f	1			
TRIDIA	TRIDIA1000			f			f	1			
TRIG	TRIG5		1.05		1						
TRIG	TRIG20	1				1.15					
TRIG	TRIG100		1.04		1						
VARDIM	VARDIM10	1			1						
VARDIM	VARDIM100		1.78		1						
VARDIM	VARDIM500		2.09		1						
VARDIM	VARDIM1000	1				1.01					
VARDIM	VARDIM5000			f			f	1			
WOOD	WOOD4	1				1.26					
ZAKHAR	ZAKHAR50	1				1.43					
ZAKHAR	ZAKHAR250	1				2.02					
ZAKHAR	ZAKHAR1000	1				1.69					
ZAKHAR	ZAKHAR5000	1				2.63					
Totals:		32	14	13	16	30	13	13	0	0	
		32	14	13	29			30	0		

Table 3.9: Place Finishes for Computational Time, Tolerance = 10^{-6}.

Function	Problem	LBFGS			HYBRID1			HYBRID1S		
		1	2	f	1	2	f	1	2	f
BIGGS	BIGGS6	1				1.35				
BROWND	BROWND4			f			f	1		
DIAGA	DIAGA10		1.02		1					
DIAGA	DIAGA100			f			f	1		
EXTRSN	EXTRSN50		1.17		1					
EXTRSN	EXTRSN250		1.09		1					
EXTRSN	EXTRSN1000		1.01		1					
EXTRSN	EXTRSN5000		1.02		1					
EXTWD	EXTWD40			f	1					
EXTWD	EXTWD100	1				1.17				
EXTWD	EXTWD500	1				1.17				
EXTWD	EXTWD1000			f	1					
HIMMBG	HIMMBG10	1				3.17				
LIARWHD	LWHD5			f			f	1		
LIARWHD	LWHD250			f			f	1		
LIARWHD	LWHD1000			f			f	1		
LIARWHD	LWHD5000			f			f	1		
NONSCOMP	NONSCP10	1				1.03				
NONSCOMP	NONSP500		1.47		1					
NONSCOMP	NONSP1000		1.47		1					
NONSCOMP	NONSP5000		1.35		1					
NONSCOMP	NONSP10000		1.41		1					
PENALA	PENALA10	1				1.01				
PENALA	PENALA250	1				1.13				
PENALA	PENALA1000	1				1.93				
PENALA	PENALA5000	1				1.76				
PQUAD	PQUAD50		1.02		1					
PQUAD	PQUAD250	1				1.20				
PQUAD	PQUAD1000	1				1.12				
PQUAD	PQUAD5000	1				1.28				

Table 3.9. Continued.

POWBSC	POWBSC2			f			f	1			
POWSNG	POWSNG4			f			f	1			
POWSNG	POWSNG100			f			f	1			
POWSNG	POWSNG500			f			f	1			
POWSNG	POWSNG1000			f			f	1			
POWER	POWER5		1.17		1						
POWER	POWER30	1				1.09					
POWER	POWER100	1				1.22					
RAYDA	RAYDA10	1				1.35					
RAYDA	RAYDA100	1				1.01					
RAYDA	RAYDA1000	1				1.35					
RAYDA	RAYDA5000	1				1.40					
ROSENB	ROSENB2	1				1.15					
TRIDIA	TRIDIA10			f			f	1			
TRIDIA	TRIDIA500			f			f	1			
TRIDIA	TRIDIA1000			f			f	1			
TRIG	TRIG5	1					f		3.50		
TRIG	TRIG20			f			f	1			
TRIG	TRIG100			f			f	1			
VARDIM	VARDIM10		1.03		1						
VARDIM	VARDIM100		1.78		1						
VARDIM	VARDIM500		2.09		1						
VARDIM	VARDIM1000	1				1.14					
VARDIM	VARDIM5000			f			f	1			
WOOD	WOOD4	1				1.16					
ZAKHAR	ZAKHAR50	1				1.60					
ZAKHAR	ZAKHAR250	1				1.69					
ZAKHAR	ZAKHAR1000	1				1.58					
ZAKHAR	ZAKHAR5000	1				2.71					
Totals:		26	14	19	16	25	18	17	1	0	
		26	14	19	32			27	0		

Table 3.10: Place Finishes for Computational Cost, Tolerance = 10^{-9}.

Function	Problem	LBFGS			HYBRID1			HYBRID1S		
		1	2	f	1	2	f	1	2	f
BIGGS	BIGGS6			f	1					
BROWND	BROWND4			f			f	1		
DIAGA	DIAGA10	1				1.22				
DIAGA	DIAGA100			f			f	1		
EXTRSN	EXTRSN50		1.23		1					
EXTRSN	EXTRSN250		1.09		1					
EXTRSN	EXTRSN1000		1.01		1					
EXTRSN	EXTRSN5000		1.01		1					
EXTWD	EXTWD40			f	1					
EXTWD	EXTWD100			f	1					
EXTWD	EXTWD500			f	1					
EXTWD	EXTWD1000			f	1					
HIMMBG	HIMMBG10	1				1.98				
LIARWHD	LWHD5			f			f	1		
LIARWHD	LWHD250			f			f	1		
LIARWHD	LWHD1000			f			f	1		
LIARWHD	LWHD5000			f			f	1		
NONSCOMP	NONSCP10	1				1.03				
NONSCOMP	NONSP500		1.23		1					
NONSCOMP	NONSP1000		1.65		1					
NONSCOMP	NONSP5000		1.62		1					
NONSCOMP	NONSP10000		1.41		1					
PENALA	PENALA10	1				1.03				
PENALA	PENALA250	1				1.09				
PENALA	PENALA1000	1				1.91				
PENALA	PENALA5000	1				1.78				
PQUAD	PQUAD50	1				1.15				
PQUAD	PQUAD250	1				1.17				
PQUAD	PQUAD1000	1				1.04				
PQUAD	PQUAD5000	1				1.21				

Table 3.10. Continued.

POWBSC	POWBSC2			f			f	1			
POWSNG	POWSNG4			f			f	1			
POWSNG	POWSNG100			f			f	1			
POWSNG	POWSNG500			f			f	1			
POWSNG	POWSNG1000			f			f	1			
POWER	POWER5	1				1.01					
POWER	POWER30		1.04		1						
POWER	POWER100		1.02		1						
RAYDA	RAYDA10	1				1.06					
RAYDA	RAYDA100	1				1.12					
RAYDA	RAYDA1000	1				1.20					
RAYDA	RAYDA5000	1				1.13					
ROSENB	ROSENB2	1				1.10					
TRIDIA	TRIDIA10			f			f	1			
TRIDIA	TRIDIA500			f			f				f
TRIDIA	TRIDIA1000			f			f				f
TRIG	TRIG5			f			f	1			
TRIG	TRIG20			f			f	1			
TRIG	TRIG100			f			f	1			
VARDIM	VARDIM10	1				1.07					
VARDIM	VARDIM100	1				1.30					
VARDIM	VARDIM500			f	1						
VARDIM	VARDIM1000			f	1						
VARDIM	VARDIM5000			f			f	1			
WOOD	WOOD4	1				1.31					
ZAKHAR	ZAKHAR50	1				1.76					
ZAKHAR	ZAKHAR250	1				1.87					
ZAKHAR	ZAKHAR1000	1				1.47					
ZAKHAR	ZAKHAR5000	1				2.91					
Totals:		24	10	25	17	24	18	16	0	2	
		24	10	25	33			24	2		

3.2 Hybrid Algorithm of Order Two

In this section, we compare Algorithm 2.4 with the limited memory BFGS algorithm (LBFGS) (Table 3.11 - Table 3.22). As mentioned in Section 2.3.2, in the event that the two-stage hybrid algorithm has difficulty satisfying the Wolfe or the Goldstein and Price conditions, a switch is made to the one-stage safeguarded hybrid algorithm, HYBRID1S. The safeguarded two-stage hybrid algorithm is denoted by HYBRID2S. Algorithm HYBRID2S was able to solve all but two problems. In fact, these are the same two problems that HYBRID1S is unable to solve: `TRIDIA500` and `TRIDIA1000` with tolerance 10^{-9}. The number of function evaluations and the number of gradient evaluations needed for HYBRID2S to achieve convergence is rarely the same. As such, this section includes three tables comparing gradient evaluations between the two algorithms.

In general, the number of gradient and function evaluations needed to achieve convergence is greater with the two-stage hybrid algorithm than with the one-stage hybrid algorithm. However, the overall computational time can be decreased when the Newton system and the limited memory updates of Algorithm 2.4 are computed in parallel. This is a direction of future work and is briefly discussed in Chapter 4.

Table 3.11: Place Finishes for the Number of Iterations, Tolerance = 10^{-3}.

Function	Problem	LBFGS			HYBRID2S		
		1	2	f	1	2	f
BIGGS	BIGGS6		1.75		1		
BROWND	BROWND4	1				2.03	
DIAGA	DIAGA10	1				1.15	
DIAGA	DIAGA100	1				1.29	
EXTRSN	EXTRSN50	1				1.25	
EXTRSN	EXTRSN250	1				1.16	
EXTRSN	EXTRSN1000	1				1.18	
EXTRSN	EXTRSN5000	1				1.17	
EXTWD	EXTWD40	1				1.27	
EXTWD	EXTWD100	1				1.23	
EXTWD	EXTWD500	1				1.36	
EXTWD	EXTWD1000		1.30		1		
HIMMBG	HIMMBG10	1				3.29	
LIARWHD	LWHD5			f	1		
LIARWHD	LWHD250			f	1		
LIARWHD	LWHD1000			f	1		
LIARWHD	LWHD5000			f	1		
NONSCOMP	NONSCP10		1.03		1		
NONSCOMP	NONSP500		1.03		1		
NONSCOMP	NONSP1000		1.13		1		
NONSCOMP	NONSP5000	1				1.28	
NONSCOMP	NONSP10000	1				1.02	
PENALA	PENALA10		3.75		1		
PENALA	PENALA250		1.94		1		
PENALA	PENALA1000		1.65		1		
PENALA	PENALA5000	1			1		
PQUAD	PQUAD50	1				1.09	
PQUAD	PQUAD250		1.68		1		
PQUAD	PQUAD1000	1				1.11	
PQUAD	PQUAD5000	1				1.07	

Table 3.11. Continued.

POWBSC	POWBSC2			f	1		
POWSNG	POWSNG4			f	1		
POWSNG	POWSNG100			f	1		
POWSNG	POWSNG500			f	1		
POWSNG	POWSNG1000			f	1		
POWER	POWER5	1				1.64	
POWER	POWER30		1.10		1		
POWER	POWER100		1.06		1		
RAYDA	RAYDA10	1				1.27	
RAYDA	RAYDA100	1				1.18	
RAYDA	RAYDA1000	1				1.16	
RAYDA	RAYDA5000	1				1.22	
ROSENB	ROSENB2	1				1.31	
TRIDIA	TRIDIA10			f	1		
TRIDIA	TRIDIA500			f	1		
TRIDIA	TRIDIA1000			f	1		
TRIG	TRIG5	1			1		
TRIG	TRIG20		1.13		1		
TRIG	TRIG100	1			1		
VARDIM	VARDIM10	1				1.67	
VARDIM	VARDIM100	1				1.73	
VARDIM	VARDIM500	1				1.41	
VARDIM	VARDIM1000	1				1.51	
VARDIM	VARDIM5000			f	1		
WOOD	WOOD4	1				1.52	
ZAKHAR	ZAKHAR50	1				1.44	
ZAKHAR	ZAKHAR250	1				2.10	
ZAKHAR	ZAKHAR1000	1				2.39	
ZAKHAR	ZAKHAR5000	1				2.16	
Totals:		34	12	13	28	31	0

Table 3.12: Place Finishes for the Number of Iterations, Tolerance = 10^{-6}.

Function	Problem	LBFGS			HYBRID2S		
		1	2	f	1	2	f
BIGGS	BIGGS6	1				1.51	
BROWND	BROWND4			f	1		
DIAGA	DIAGA10	1				1.17	
DIAGA	DIAGA100			f	1		
EXTRSN	EXTRSN50	1				1.21	
EXTRSN	EXTRSN250	1				1.16	
EXTRSN	EXTRSN1000	1				1.18	
EXTRSN	EXTRSN5000	1				1.17	
EXTWD	EXTWD40			f	1		
EXTWD	EXTWD100	1				1.15	
EXTWD	EXTWD500	1				1.32	
EXTWD	EXTWD1000			f	1		
HIMMBG	HIMMBG10	1				2.59	
LIARWHD	LWHD5			f	1		
LIARWHD	LWHD250			f	1		
LIARWHD	LWHD1000			f	1		
LIARWHD	LWHD5000			f	1		
NONSCOMP	NONSCP10	1				1.16	
NONSCOMP	NONSP500		2.01		1		
NONSCOMP	NONSP1000		2.05		1		
NONSCOMP	NONSP5000		1.93		1		
NONSCOMP	NONSP10000		1.99		1		
PENALA	PENALA10		1.12		1		
PENALA	PENALA250		1.27		1		
PENALA	PENALA1000		1.12		1		
PENALA	PENALA5000		1.01		1		
PQUAD	PQUAD50	1				1.07	
PQUAD	PQUAD250	1				1.14	
PQUAD	PQUAD1000	1				1.24	
PQUAD	PQUAD5000	1				1.20	

Table 3.12. Continued.

POWBSC	POWBSC2			f	1		
POWSNG	POWSNG4			f	1		
POWSNG	POWSNG100			f	1		
POWSNG	POWSNG500			f	1		
POWSNG	POWSNG1000			f	1		
POWER	POWER5	1				1.47	
POWER	POWER30	1				1.05	
POWER	POWER100	1				1.14	
RAYDA	RAYDA10	1				1.25	
RAYDA	RAYDA100	1				1.23	
RAYDA	RAYDA1000	1				1.24	
RAYDA	RAYDA5000	1				1.34	
ROSENB	ROSENB2	1				1.35	
TRIDIA	TRIDIA10			f	1		
TRIDIA	TRIDIA500			f	1		
TRIDIA	TRIDIA1000			f	1		
TRIG	TRIG5	1				1.04	
TRIG	TRIG20			f	1		
TRIG	TRIG100			f	1		
VARDIM	VARDIM10	1				1.62	
VARDIM	VARDIM100	1				1.78	
VARDIM	VARDIM500	1				1.46	
VARDIM	VARDIM1000	1				1.59	
VARDIM	VARDIM5000			f	1		
WOOD	WOOD4	1				1.40	
ZAKHAR	ZAKHAR50	1				1.46	
ZAKHAR	ZAKHAR250	1				2.10	
ZAKHAR	ZAKHAR1000	1				2.41	
ZAKHAR	ZAKHAR5000	1				1.85	
Totals:		32	8	19	27	32	0

Table 3.13: Place Finishes for the Number of Iterations, Tolerance = 10^{-9}.

Function	Problem	LBFGS			HYBRID2S		
		1	2	f	1	2	f
BIGGS	BIGGS6			f	1		
BROWND	BROWND4			f	1		
DIAGA	DIAGA10	1				1.07	
DIAGA	DIAGA100			f	1		
EXTRSN	EXTRSN50	1				1.19	
EXTRSN	EXTRSN250	1				1.15	
EXTRSN	EXTRSN1000	1				1.18	
EXTRSN	EXTRSN5000	1				1.17	
EXTWD	EXTWD40			f	1		
EXTWD	EXTWD100			f	1		
EXTWD	EXTWD500			f	1		
EXTWD	EXTWD1000			f	1		
HIMMBG	HIMMBG10	1				2.41	
LIARWHD	LWHD5			f	1		
LIARWHD	LWHD250			f	1		
LIARWHD	LWHD1000			f	1		
LIARWHD	LWHD5000			f	1		
NONSCOMP	NONSCP10	1				1.15	
NONSCOMP	NONSP500		1.64		1		
NONSCOMP	NONSP1000		1.91		1		
NONSCOMP	NONSP5000		1.77		1		
NONSCOMP	NONSP10000		2.03		1		
PENALA	PENALA10		1.15		1		
PENALA	PENALA250		1.29		1		
PENALA	PENALA1000		1.14		1		
PENALA	PENALA5000		1.01		1		
PQUAD	PQUAD50	1				1.05	
PQUAD	PQUAD250	1				1.05	
PQUAD	PQUAD1000		1.02		1		
PQUAD	PQUAD5000	1				1.01	

Table 3.13. Continued.

POWBSC	POWBSC2			f	1		
POWSNG	POWSNG4			f	1		
POWSNG	POWSNG100			f	1		
POWSNG	POWSNG500			f	1		
POWSNG	POWSNG1000			f	1		
POWER	POWER5	1				1.24	
POWER	POWER30		1.10		1		
POWER	POWER100	1				1.03	
RAYDA	RAYDA10	1				1.21	
RAYDA	RAYDA100	1				1.13	
RAYDA	RAYDA1000	1				1.09	
RAYDA	RAYDA5000	1				1.07	
ROSENB	ROSENB2	1				1.38	
TRIDIA	TRIDIA10			f	1		
TRIDIA	TRIDIA500			f	1		
TRIDIA	TRIDIA1000			f	1		
TRIG	TRIG5			f	1		
TRIG	TRIG20			f			f
TRIG	TRIG100			f			f
VARDIM	VARDIM10	1				1.50	
VARDIM	VARDIM100	1				1.79	
VARDIM	VARDIM500			f	1		
VARDIM	VARDIM1000			f	1		
VARDIM	VARDIM5000			f	1		
WOOD	WOOD4	1				1.42	
ZAKHAR	ZAKHAR50	1				1.49	
ZAKHAR	ZAKHAR250	1				2.13	
ZAKHAR	ZAKHAR1000	1				2.44	
ZAKHAR	ZAKHAR5000	1				1.87	
Totals:		24	10	25	33	24	2

Table 3.14: Place Finishes for Function Evaluations, Tolerance = 10^{-3}.

Function	Problem	LBFGS			HYBRID2S		
		1	2	f	1	2	f
BIGGS	BIGGS6		1.20		1		
BROWND	BROWND4	1				1.96	
DIAGA	DIAGA10	1				2.22	
DIAGA	DIAGA100	1				2.82	
EXTRSN	EXTRSN50	1				3.57	
EXTRSN	EXTRSN250	1				3.41	
EXTRSN	EXTRSN1000	1				3.47	
EXTRSN	EXTRSN5000	1				3.43	
EXTWD	EXTWD40	1				3.27	
EXTWD	EXTWD100	1				2.88	
EXTWD	EXTWD500	1				3.30	
EXTWD	EXTWD1000	1				1.97	
HIMMBG	HIMMBG10	1				5.00	
LIARWHD	LWHD5			f	1		
LIARWHD	LWHD250			f	1		
LIARWHD	LWHD1000			f	1		
LIARWHD	LWHD5000			f	1		
NONSCOMP	NONSCP10	1				1.41	
NONSCOMP	NONSP500	1				1.61	
NONSCOMP	NONSP1000	1				1.46	
NONSCOMP	NONSP5000	1				2.18	
NONSCOMP	NONSP10000	1				1.76	
PENALA	PENALA10		1.44		1		
PENALA	PENALA250	1				1.16	
PENALA	PENALA1000	1				1.30	
PENALA	PENALA5000	1				1.96	
PQUAD	PQUAD50	1				2.39	
PQUAD	PQUAD250	1				2.07	
PQUAD	PQUAD1000	1				2.02	
PQUAD	PQUAD5000	1				1.70	

Table 3.14. Continued.

POWBSC	POWBSC2			f	1		
POWSNG	POWSNG4			f	1		
POWSNG	POWSNG100			f	1		
POWSNG	POWSNG500			f	1		
POWSNG	POWSNG1000			f	1		
POWER	POWER5	1				3.33	
POWER	POWER30	1				2.17	
POWER	POWER100	1				1.56	
RAYDA	RAYDA10	1				2.54	
RAYDA	RAYDA100	1				2.54	
RAYDA	RAYDA1000	1				2.32	
RAYDA	RAYDA5000	1				2.30	
ROSENB	ROSENB2	1				2.58	
TRIDIA	TRIDIA10			f	1		
TRIDIA	TRIDIA500			f	1		
TRIDIA	TRIDIA1000			f	1		
TRIG	TRIG5	1				2.54	
TRIG	TRIG20	1				1.81	
TRIG	TRIG100	1				1.13	
VARDIM	VARDIM10	1				1.43	
VARDIM	VARDIM100	1				1.31	
VARDIM	VARDIM500	1				1.22	
VARDIM	VARDIM1000	1				1.17	
VARDIM	VARDIM5000			f	1		
WOOD	WOOD4	1				2.58	
ZAKHAR	ZAKHAR50	1				1.18	
ZAKHAR	ZAKHAR250	1				2.18	
ZAKHAR	ZAKHAR1000	1				3.00	
ZAKHAR	ZAKHAR5000	1				3.17	
Totals:		44	2	13	15	44	0

Table 3.15: Place Finishes for Function Evaluations, Tolerance = 10^{-6}.

Function	Problem	LBFGS			HYBRID2S		
		1	2	f	1	2	f
BIGGS	BIGGS6	1				1.98	
BROWND	BROWND4			f	1		
DIAGA	DIAGA10	1				2.00	
DIAGA	DIAGA100			f	1		
EXTRSN	EXTRSN50	1				3.58	
EXTRSN	EXTRSN250	1				3.43	
EXTRSN	EXTRSN1000	1				3.47	
EXTRSN	EXTRSN5000	1				3.42	
EXTWD	EXTWD40			f	1		
EXTWD	EXTWD100	1				2.74	
EXTWD	EXTWD500	1				2.94	
EXTWD	EXTWD1000			f	1		
HIMMBG	HIMMBG10	1				3.47	
LIARWHD	LWHD5			f	1		
LIARWHD	LWHD250			f	1		
LIARWHD	LWHD1000			f	1		
LIARWHD	LWHD5000			f	1		
NONSCOMP	NONSCP10	1				1.35	
NONSCOMP	NONSP500		1.89		1		
NONSCOMP	NONSP1000		1.89		1		
NONSCOMP	NONSP5000		1.76		1		
NONSCOMP	NONSP10000		1.85		1		
PENALA	PENALA10		1.07		1		
PENALA	PENALA250	1			1		
PENALA	PENALA1000	1				1.18	
PENALA	PENALA5000	1				1.46	
PQUAD	PQUAD50	1				1.85	
PQUAD	PQUAD250	1				1.83	
PQUAD	PQUAD1000	1				1.89	
PQUAD	PQUAD5000	1				1.66	

Table 3.15. Continued.

POWBSC	POWBSC2			f	1		
POWSNG	POWSNG4			f	1		
POWSNG	POWSNG100			f	1		
POWSNG	POWSNG500			f	1		
POWSNG	POWSNG1000			f	1		
POWER	POWER5	1				2.92	
POWER	POWER30	1				1.89	
POWER	POWER100	1				1.63	
RAYDA	RAYDA10	1				2.29	
RAYDA	RAYDA100	1				2.15	
RAYDA	RAYDA1000	1				2.08	
RAYDA	RAYDA5000	1				2.03	
ROSENB	ROSENB2	1				2.69	
TRIDIA	TRIDIA10			f	1		
TRIDIA	TRIDIA500			f	1		
TRIDIA	TRIDIA1000			f	1		
TRIG	TRIG5	1				1.46	
TRIG	TRIG20			f	1		
TRIG	TRIG100			f	1		
VARDIM	VARDIM10	1				1.42	
VARDIM	VARDIM100	1				1.36	
VARDIM	VARDIM500	1				1.27	
VARDIM	VARDIM1000	1				1.20	
VARDIM	VARDIM5000			f	1		
WOOD	WOOD4	1				2.44	
ZAKHAR	ZAKHAR50	1				1.22	
ZAKHAR	ZAKHAR250	1				2.22	
ZAKHAR	ZAKHAR1000	1				3.03	
ZAKHAR	ZAKHAR5000	1				3.22	
Totals:		35	5	19	25	34	0

Table 3.16: Place Finishes for Function Evaluations, Tolerance = 10^{-9}.

Function	Problem	LBFGS			HYBRID2S		
		1	2	f	1	2	f
BIGGS	BIGGS6			f	1		
BROWND	BROWND4			f	1		
DIAGA	DIAGA10	1				1.73	
DIAGA	DIAGA100			f	1		
EXTRSN	EXTRSN50	1				3.36	
EXTRSN	EXTRSN250	1				3.34	
EXTRSN	EXTRSN1000	1				3.46	
EXTRSN	EXTRSN5000	1				3.43	
EXTWD	EXTWD40			f	1		
EXTWD	EXTWD100			f	1		
EXTWD	EXTWD500			f	1		
EXTWD	EXTWD1000			f	1		
HIMMBG	HIMMBG10	1				3.00	
LIARWHD	LWHD5			f	1		
LIARWHD	LWHD250			f	1		
LIARWHD	LWHD1000			f	1		
LIARWHD	LWHD5000			f	1		
NONSCOMP	NONSCP10	1				1.34	
NONSCOMP	NONSP500		1.61		1		
NONSCOMP	NONSP1000		1.87		1		
NONSCOMP	NONSP5000		1.73		1		
NONSCOMP	NONSP10000		2.05		1		
PENALA	PENALA10		1.09		1		
PENALA	PENALA250		1.01		1		
PENALA	PENALA1000	1				1.16	
PENALA	PENALA5000	1				1.46	
PQUAD	PQUAD50	1				1.64	
PQUAD	PQUAD250	1				1.54	
PQUAD	PQUAD1000	1				1.40	
PQUAD	PQUAD5000	1				1.31	

Table 3.16. Continued.

POWBSC	POWBSC2			f	1		
POWSNG	POWSNG4			f	1		
POWSNG	POWSNG100			f	1		
POWSNG	POWSNG500			f	1		
POWSNG	POWSNG1000			f	1		
POWER	POWER5	1				2.47	
POWER	POWER30	1				1.49	
POWER	POWER100	1				1.30	
RAYDA	RAYDA10	1				1.88	
RAYDA	RAYDA100	1				1.75	
RAYDA	RAYDA1000	1				1.62	
RAYDA	RAYDA5000	1				1.49	
ROSENB	ROSENB2	1				2.78	
TRIDIA	TRIDIA10			f	1		
TRIDIA	TRIDIA500			f	1		
TRIDIA	TRIDIA1000			f	1		
TRIG	TRIG5			f	1		
TRIG	TRIG20			f			f
TRIG	TRIG100			f			f
VARDIM	VARDIM10	1				1.38	
VARDIM	VARDIM100	1				1.40	
VARDIM	VARDIM500			f	1		
VARDIM	VARDIM1000			f	1		
VARDIM	VARDIM5000			f	1		
WOOD	WOOD4	1				2.41	
ZAKHAR	ZAKHAR50	1				1.24	
ZAKHAR	ZAKHAR250	1				2.25	
ZAKHAR	ZAKHAR1000	1				3.05	
ZAKHAR	ZAKHAR5000	1				3.24	
Totals:		28	6	25	29	28	2

Table 3.17: Place Finishes for Gradient Evaluations, Tolerance = 10^{-3}.

Function	Problem	LBFGS			HYBRID2S		
		1	2	f	1	2	f
BIGGS	BIGGS6		1.20		1		
BROWND	BROWND4	1				2.31	
DIAGA	DIAGA10	1				2.11	
DIAGA	DIAGA100	1				2.60	
EXTRSN	EXTRSN50	1				3.15	
EXTRSN	EXTRSN250	1				3.03	
EXTRSN	EXTRSN1000	1				3.13	
EXTRSN	EXTRSN5000	1				3.08	
EXTWD	EXTWD40	1				2.97	
EXTWD	EXTWD100	1				2.57	
EXTWD	EXTWD500	1				3.04	
EXTWD	EXTWD1000	1				1.79	
HIMMBG	HIMMBG10	1				5.00	
LIARWHD	LWHD5			f	1		
LIARWHD	LWHD250			f	1		
LIARWHD	LWHD1000			f	1		
LIARWHD	LWHD5000			f	1		
NONSCOMP	NONSCP10	1				1.35	
NONSCOMP	NONSP500	1				1.55	
NONSCOMP	NONSP1000	1				1.38	
NONSCOMP	NONSP5000	1				2.06	
NONSCOMP	NONSP10000	1				1.67	
PENALA	PENALA10		1.44		1		
PENALA	PENALA250	1				1.16	
PENALA	PENALA1000	1				1.30	
PENALA	PENALA5000	1				1.96	
PQUAD	PQUAD50	1				2.10	
PQUAD	PQUAD250	1				1.89	
PQUAD	PQUAD1000	1				1.77	
PQUAD	PQUAD5000	1				1.55	

Table 3.17. Continued.

POWBSC	POWBSC2			f	1		
POWSNG	POWSNG4			f	1		
POWSNG	POWSNG100			f	1		
POWSNG	POWSNG500			f	1		
POWSNG	POWSNG1000			f	1		
POWER	POWER5	1				3.14	
POWER	POWER30	1				1.73	
POWER	POWER100	1				1.37	
RAYDA	RAYDA10	1				2.54	
RAYDA	RAYDA100	1				2.38	
RAYDA	RAYDA1000	1				2.12	
RAYDA	RAYDA5000	1				2.04	
ROSENB	ROSENB2	1				2.32	
TRIDIA	TRIDIA10			f	1		
TRIDIA	TRIDIA500			f	1		
TRIDIA	TRIDIA1000			f	1		
TRIG	TRIG5	1				2.45	
TRIG	TRIG20	1				1.67	
TRIG	TRIG100	1				1.13	
VARDIM	VARDIM10	1				2.14	
VARDIM	VARDIM100	1				1.44	
VARDIM	VARDIM500	1				1.66	
VARDIM	VARDIM1000	1				1.57	
VARDIM	VARDIM5000			f	1		
WOOD	WOOD4	1				2.44	
ZAKHAR	ZAKHAR50	1				1.36	
ZAKHAR	ZAKHAR250	1				2.98	
ZAKHAR	ZAKHAR1000	1				3.56	
ZAKHAR	ZAKHAR5000	1				3.63	
Totals:		44	2	13	15	44	0

Table 3.18: Place Finishes for Gradient Evaluations, Tolerance = 10^{-6}.

Function	Problem	LBFGS			HYBRID2S		
		1	2	f	1	2	f
BIGGS	BIGGS6	1				1.98	
BROWND	BROWND4			f	1		
DIAGA	DIAGA10	1				1.91	
DIAGA	DIAGA100			f	1		
EXTRSN	EXTRSN50	1				3.13	
EXTRSN	EXTRSN250	1				3.04	
EXTRSN	EXTRSN1000	1				3.13	
EXTRSN	EXTRSN5000	1				3.08	
EXTWD	EXTWD40			f	1		
EXTWD	EXTWD100	1				2.49	
EXTWD	EXTWD500	1				2.73	
EXTWD	EXTWD1000			f	1		
HIMMBG	HIMMBG10	1				3.47	
LIARWHD	LWHD5			f	1		
LIARWHD	LWHD250			f	1		
LIARWHD	LWHD1000			f	1		
LIARWHD	LWHD5000			f	1		
NONSCOMP	NONSCP10	1				1.32	
NONSCOMP	NONSP500		1.93		1		
NONSCOMP	NONSP1000		1.98		1		
NONSCOMP	NONSP5000		1.79		1		
NONSCOMP	NONSP10000		1.65		1		
PENALA	PENALA10		1.07		1		
PENALA	PENALA250	1			1		
PENALA	PENALA1000	1				1.18	
PENALA	PENALA5000	1				1.47	
PQUAD	PQUAD50	1				1.67	
PQUAD	PQUAD250	1				1.70	
PQUAD	PQUAD1000	1				1.97	
PQUAD	PQUAD5000	1				1.55	

Table 3.18. Continued.

POWBSC	POWBSC2			f	1		
POWSNG	POWSNG4			f	1		
POWSNG	POWSNG100			f	1		
POWSNG	POWSNG500			f	1		
POWSNG	POWSNG1000			f	1		
POWER	POWER5	1				2.77	
POWER	POWER30	1				1.59	
POWER	POWER100	1				1.49	
RAYDA	RAYDA10	1				2.29	
RAYDA	RAYDA100	1				2.04	
RAYDA	RAYDA1000	1				1.97	
RAYDA	RAYDA5000	1				1.87	
ROSENB	ROSENB2	1				2.38	
TRIDIA	TRIDIA10			f	1		
TRIDIA	TRIDIA500			f	1		
TRIDIA	TRIDIA1000			f	1		
TRIG	TRIG5	1				1.98	
TRIG	TRIG20			f	1		
TRIG	TRIG100			f	1		
VARDIM	VARDIM10	1				2.11	
VARDIM	VARDIM100	1				1.49	
VARDIM	VARDIM500	1				1.72	
VARDIM	VARDIM1000	1				1.65	
VARDIM	VARDIM5000			f	1		
WOOD	WOOD4	1				2.31	
ZAKHAR	ZAKHAR50	1				1.41	
ZAKHAR	ZAKHAR250	1				3.31	
ZAKHAR	ZAKHAR1000	1				3.59	
ZAKHAR	ZAKHAR5000	1				3.70	
Totals:		35	5	19	25	34	0

Table 3.19: Place Finishes for Gradient Evaluations, Tolerance = 10^{-9}.

Function	Problem	LBFGS			HYBRID2S		
		1	2	f	1	2	f
BIGGS	BIGGS6			f	1		
BROWND	BROWND4			f	1		
DIAGA	DIAGA10	1				1.67	
DIAGA	DIAGA100			f	1		
EXTRSN	EXTRSN50	1				2.95	
EXTRSN	EXTRSN250	1				2.97	
EXTRSN	EXTRSN1000	1				3.12	
EXTRSN	EXTRSN5000	1				3.08	
EXTWD	EXTWD40			f	1		
EXTWD	EXTWD100			f	1		
EXTWD	EXTWD500			f	1		
EXTWD	EXTWD1000			f	1		
HIMMBG	HIMMBG10	1				3.00	
LIARWHD	LWHD5			f	1		
LIARWHD	LWHD250			f	1		
LIARWHD	LWHD1000			f	1		
LIARWHD	LWHD5000			f	1		
NONSCOMP	NONSCP10	1				1.34	
NONSCOMP	NONSP500		1.61		1		
NONSCOMP	NONSP1000		1.87		1		
NONSCOMP	NONSP5000		1.73		1		
NONSCOMP	NONSP10000		2.00		1		
PENALA	PENALA10		1.09		1		
PENALA	PENALA250		1.01		1		
PENALA	PENALA1000	1				1.16	
PENALA	PENALA5000	1				1.47	
PQUAD	PQUAD50	1				1.50	
PQUAD	PQUAD250	1				1.45	
PQUAD	PQUAD1000	1				1.30	
PQUAD	PQUAD5000	1				1.24	

Table 3.19. Continued.

POWBSC	POWBSC2			f	1		
POWSNG	POWSNG4			f	1		
POWSNG	POWSNG100			f	1		
POWSNG	POWSNG500			f	1		
POWSNG	POWSNG1000			f	1		
POWER	POWER5	1				2.34	
POWER	POWER30	1				1.28	
POWER	POWER100	1				1.21	
RAYDA	RAYDA10	1				1.88	
RAYDA	RAYDA100	1				1.67	
RAYDA	RAYDA1000	1				1.55	
RAYDA	RAYDA5000	1				1.39	
ROSENB	ROSENB2	1				2.33	
TRIDIA	TRIDIA10			f	1		
TRIDIA	TRIDIA500			f	1		
TRIDIA	TRIDIA1000			f	1		
TRIG	TRIG5			f	1		
TRIG	TRIG20			f			f
TRIG	TRIG100			f			f
VARDIM	VARDIM10	1				2.05	
VARDIM	VARDIM100	1				1.55	
VARDIM	VARDIM500			f	1		
VARDIM	VARDIM1000			f	1		
VARDIM	VARDIM5000			f	1		
WOOD	WOOD4	1				2.29	
ZAKHAR	ZAKHAR50	1				1.46	
ZAKHAR	ZAKHAR250	1				3.36	
ZAKHAR	ZAKHAR1000	1				3.63	
ZAKHAR	ZAKHAR5000	1				3.74	
Totals:		28	6	25	29	28	2

Table 3.20: Place Finishes for Computational Time, Tolerance = 10^{-3}.

Function	Problem	LBFGS			HYBRID2S		
		1	2	f	1	2	f
BIGGS	BIGGS6		1.65		1		
BROWND	BROWND4	1				1.63	
DIAGA	DIAGA10	1				1.57	
DIAGA	DIAGA100	1				2.19	
EXTRSN	EXTRSN50	1				2.29	
EXTRSN	EXTRSN250	1				2.15	
EXTRSN	EXTRSN1000	1				3.26	
EXTRSN	EXTRSN5000	1				2.86	
EXTWD	EXTWD40	1				2.18	
EXTWD	EXTWD100	1				2.01	
EXTWD	EXTWD500	1				2.69	
EXTWD	EXTWD1000	1				1.70	
HIMMBG	HIMMBG10	1				3.72	
LIARWHD	LWHD5			f	1		
LIARWHD	LWHD250			f	1		
LIARWHD	LWHD1000			f	1		
LIARWHD	LWHD5000			f	1		
NONSCOMP	NONSCP10		1.12		1		
NONSCOMP	NONSP500	1				1.52	
NONSCOMP	NONSP1000	1				1.18	
NONSCOMP	NONSP5000	1				1.97	
NONSCOMP	NONSP10000	1				1.69	
PENALA	PENALA10		1.54		1		
PENALA	PENALA250		1.17		1		
PENALA	PENALA1000	1				1.33	
PENALA	PENALA5000	1				1.75	
PQUAD	PQUAD50	1				1.68	
PQUAD	PQUAD250	1				1.48	
PQUAD	PQUAD1000	1				1.59	
PQUAD	PQUAD5000	1				1.36	

Table 3.20. Continued.

POWBSC	POWBSC2			f	1		
POWSNG	POWSNG4			f	1		
POWSNG	POWSNG100			f	1		
POWSNG	POWSNG500			f	1		
POWSNG	POWSNG1000			f	1		
POWER	POWER5	1				2.31	
POWER	POWER30	1				1.25	
POWER	POWER100	1				1.08	
RAYDA	RAYDA10	1				1.36	
RAYDA	RAYDA100	1				2.00	
RAYDA	RAYDA1000	1				1.41	
RAYDA	RAYDA5000	1				1.91	
ROSENB	ROSENB2	1				2.11	
TRIDIA	TRIDIA10			f	1		
TRIDIA	TRIDIA500			f	1		
TRIDIA	TRIDIA1000			f	1		
TRIG	TRIG5	1				1.75	
TRIG	TRIG20	1				1.23	
TRIG	TRIG100	1				1.43	
VARDIM	VARDIM10		1.65		1		
VARDIM	VARDIM100	1				2.78	
VARDIM	VARDIM500		1.07		1		
VARDIM	VARDIM1000		1.06		1		
VARDIM	VARDIM5000			f	1		
WOOD	WOOD4	1				1.92	
ZAKHAR	ZAKHAR50	1				1.31	
ZAKHAR	ZAKHAR250	1				2.29	
ZAKHAR	ZAKHAR1000	1				7.00	
ZAKHAR	ZAKHAR5000	1				2.63	
Totals:		39	7	13	20	39	0

Table 3.21: Place Finishes for Computational Time, Tolerance = 10^{-6}.

Function	Problem	LBFGS			HYBRID2S		
		1	2	f	1	2	f
BIGGS	BIGGS6	1				1.28	
BROWND	BROWND4			f	1		
DIAGA	DIAGA10	1				1.30	
DIAGA	DIAGA100			f	1		
EXTRSN	EXTRSN50	1				1.93	
EXTRSN	EXTRSN250	1				2.17	
EXTRSN	EXTRSN1000	1				3.18	
EXTRSN	EXTRSN5000	1				2.84	
EXTWD	EXTWD40			f	1		
EXTWD	EXTWD100	1				1.85	
EXTWD	EXTWD500	1				2.70	
EXTWD	EXTWD1000			f	1		
HIMMBG	HIMMBG10	1				2.93	
LIARWHD	LWHD5			f	1		
LIARWHD	LWHD250			f	1		
LIARWHD	LWHD1000			f	1		
LIARWHD	LWHD5000			f	1		
NONSCOMP	NONSCP10	1				1.04	
NONSCOMP	NONSP500		1.93		1		
NONSCOMP	NONSP1000		1.98		1		
NONSCOMP	NONSP5000		1.80		1		
NONSCOMP	NONSP10000		1.65		1		
PENALA	PENALA10		1.36		1		
PENALA	PENALA250		1.40		1		
PENALA	PENALA1000		1.05		1		
PENALA	PENALA5000	1				1.29	
PQUAD	PQUAD50	1				1.31	
PQUAD	PQUAD250	1				1.35	
PQUAD	PQUAD1000	1				1.48	
PQUAD	PQUAD5000	1				1.49	

Table 3.21. Continued.

POWBSC	POWBSC2			f	1		
POWSNG	POWSNG4			f	1		
POWSNG	POWSNG100			f	1		
POWSNG	POWSNG500			f	1		
POWSNG	POWSNG1000			f	1		
POWER	POWER5	1				1.41	
POWER	POWER30	1				1.03	
POWER	POWER100	1				1.28	
RAYDA	RAYDA10	1				1.76	
RAYDA	RAYDA100	1				1.33	
RAYDA	RAYDA1000	1				1.74	
RAYDA	RAYDA5000	1				1.82	
ROSENB	ROSENB2	1				2.09	
TRIDIA	TRIDIA10			f	1		
TRIDIA	TRIDIA500			f	1		
TRIDIA	TRIDIA1000			f	1		
TRIG	TRIG5	1				1.57	
TRIG	TRIG20			f	1		
TRIG	TRIG100			f	1		
VARDIM	VARDIM10	1				2.26	
VARDIM	VARDIM100	1				1.83	
VARDIM	VARDIM500	1				1.54	
VARDIM	VARDIM1000	1				2.11	
VARDIM	VARDIM5000			f	1		
WOOD	WOOD4	1				1.75	
ZAKHAR	ZAKHAR50	1				1.73	
ZAKHAR	ZAKHAR250	1				2.67	
ZAKHAR	ZAKHAR1000	1				6.01	
ZAKHAR	ZAKHAR5000	1				2.65	
Totals:		33	7	19	26	33	0

Table 3.22: Place Finishes for Computational Time, Tolerance = 10^{-9}.

Function	Problem	LBFGS			HYBRID2S		
		1	2	f	1	2	f
BIGGS	BIGGS6			f	1		
BROWND	BROWND4			f	1		
DIAGA	DIAGA10	1				1.44	
DIAGA	DIAGA100			f	1		
EXTRSN	EXTRSN50	1				1.77	
EXTRSN	EXTRSN250	1				2.12	
EXTRSN	EXTRSN1000	1				3.21	
EXTRSN	EXTRSN5000	1				2.85	
EXTWD	EXTWD40			f	1		
EXTWD	EXTWD100			f	1		
EXTWD	EXTWD500			f	1		
EXTWD	EXTWD1000			f	1		
HIMMBG	HIMMBG10	1				2.13	
LIARWHD	LWHD5			f	1		
LIARWHD	LWHD250			f	1		
LIARWHD	LWHD1000			f	1		
LIARWHD	LWHD5000			f	1		
NONSCOMP	NONSCP10	1				1.03	
NONSCOMP	NONSP500		1.64		1		
NONSCOMP	NONSP1000		2.80		1		
NONSCOMP	NONSP5000		1.52		1		
NONSCOMP	NONSP10000		1.74		1		
PENALA	PENALA10		1.29		1		
PENALA	PENALA250		1.39		1		
PENALA	PENALA1000	1			1		
PENALA	PENALA5000	1				1.34	
PQUAD	PQUAD50	1				1.36	
PQUAD	PQUAD250	1				1.13	
PQUAD	PQUAD1000	1				1.30	
PQUAD	PQUAD5000	1				1.44	

Table 3.22. Continued.

POWBSC	POWBSC2			f	1		
POWSNG	POWSNG4			f	1		
POWSNG	POWSNG100			f	1		
POWSNG	POWSNG500			f	1		
POWSNG	POWSNG1000			f	1		
POWER	POWER5	1				1.76	
POWER	POWER30	1				1.11	
POWER	POWER100	1				1.10	
RAYDA	RAYDA10	1				1.56	
RAYDA	RAYDA100	1				1.35	
RAYDA	RAYDA1000	1				1.40	
RAYDA	RAYDA5000	1				1.44	
ROSENB	ROSENB2	1				2.33	
TRIDIA	TRIDIA10			f	1		
TRIDIA	TRIDIA500			f	1		
TRIDIA	TRIDIA1000			f	1		
TRIG	TRIG5			f	1		
TRIG	TRIG20			f			f
TRIG	TRIG100			f			f
VARDIM	VARDIM10	1				2.58	
VARDIM	VARDIM100	1				4.81	
VARDIM	VARDIM500			f	1		
VARDIM	VARDIM1000			f	1		
VARDIM	VARDIM5000			f	1		
WOOD	WOOD4	1				2.02	
ZAKHAR	ZAKHAR50	1				1.74	
ZAKHAR	ZAKHAR250	1				3.17	
ZAKHAR	ZAKHAR1000	1				6.17	
ZAKHAR	ZAKHAR5000	1				2.69	
Totals:		28	6	25	30	27	2

CHAPTER 4
CONCLUSIONS AND FUTURE WORK

In this thesis, we have combined the robust properties of numerical integration methods, specifically implicit Runge-Kutta methods, with the low cost properties of quasi-Newton methods. The hybrid algorithms we developed all use a quasi-Newton matrix to approximate the matrix of the nonlinear system (1.21) arising from the implementation of Newton's method in Runge-Kutta algorithms. In particular, the matrix we approximate is given by

$$B_{i,k}(\lambda) \approx \left(\frac{\lambda}{\mu_i} I + \nabla^2 f(x_k) \right)$$

where μ_i, $i = 1, \ldots, s$ are the eigenvalues of the coefficient matrix A associated with the Runge-Kutta method and $\lambda_k = 1/h_k$ where h_k is the stepsize associated with numerically integrating the gradient system. Quasi-Newton methods use a quasi-Newton matrix to approximate the actual Hessian of the objective function,

$$B_k \approx \nabla^2 f(x_k).$$

The comparison of the limited memory implementations of these methods shows that the hybrid methods can be more robust than limited memory BFGS. Although hybrid algorithms show much promise, there are many directions for future research in this area.

As mentioned in Section 2.3.4, developing convergence proofs for two-stage hybrid algorithms is difficult because of the presence of a search curve rather than a

search line. For example, Zoutendijk's theorem does not hold for search curves. The idea behind the proof of Zoutendijk's theorem is to find a lower bound on α_k. For line search methods, the second Wolfe condition and the Lipschitz continuity of the gradient produce

$$\alpha_k \geq \frac{\gamma_2 - 1}{L} \frac{\nabla f(x_k)^T p_k}{||p_k||^2}.$$

However, using the same tools on the search curve $x_{h_k}(\theta)$, one finds

$$\gamma_2 m'(0) - \nabla f(x_k)^T \dot{x}_{h_k}(\theta_k) \leq L||x_{h_k}(\theta_k) - x_k||\ ||\dot{x}_{h_k}(\theta_k)||.$$

This inequality leads to a cubic inequality in θ_k which does not produce a lower bound for θ_k. Developing reasonable hypotheses under which these algorithms will converge is a future direction of research. Also, hybrid algorithms can be developed that do not diagonalize the coefficient matrix A. In a limited memory setting, this results in $s_j,\ y_j \in \mathbb{R}^{sn}$ for all $j = 1, \ldots, s$ rather than $s_j,\ y_j \in \mathbb{R}^n$. This increase in storage can be offset by using only $m/2$ limited memory vectors. Proving convergence properties for this class of hybrid algorithms may be easier as the choice $x_k^{(0)} = x_k$ for the Newton iterations always produces a descent direction when one Newton iteration is performed.

Parallelizing the two-stage hybrid algorithm, Algorithm 2.4, is possible. In this algorithm, we calculate the internal stages of the differential equation at step k via the formula

$$\Delta W_{i,k}^{(j)} = -\frac{1}{\mu_i} H_{i,k}(\lambda_k) \left(\lambda_k W_{i,k}^{(j)} + v_{i,k}^{(j)} \right),$$

for $i = 1, 2$. Since these two formulas are independent of each other, they can be calculated in parallel [13]. Additionally, the limited memory vectors, $s_{i,k}$ and $y_{i,k}$ for $i = 1, 2$ can be updated in parallel. Parallelizing this algorithm can provide an increase in performance.

In developing hybrid algorithms, we relied on the BFGS update because of its limited memory capabilities. However, the SR1 quasi-Newton method also has a compact representation which would lend itself to possible hybrid algorithms [10]. Although the SR1 method does not possess the positive definite heredity property that the BFGS method possesses, the SR1 matrices often produce better approximations to the actual Hessian matrix. Therefore, hybrid trust-region algorithms based on the SR1 method can be developed where the SR1 method is used to solve the quadratic subproblem at each step. Line search hybrid algorithms can also be developed based on the SR1 method, but care must be taken so that a descent direction is always produced.

The hybrid algorithms developed in this thesis can also be adapted to constrained problems. The nonlinear constrained optimization problem (NLP) can be stated as

$$\text{Minimize} \quad f(x)$$

$$\text{Subject to} \quad g(x) = 0$$

where f is the objective function and $g : \mathbb{R}^n \to \mathbb{R}^m$ is a system of equality constraints. Quasi-Newton matrices have been used to solve NLP where the search direction at each step is given by the solution to a constrained quadratic subproblem. Convergence

properties have also been developed [6, 25, 26]. The use of quasi-Newton matrices eliminates the need to calculate the second derivative of the Lagrangian function and may prove useful in solving NLP.

In unconstrained optimization, finding a local minimum is closely related to finding an equilibrium point of the corresponding gradient system. Similarly, the NLP is related to differential-algebraic equations (DAEs) which have the form

$$\dot{x} = \nabla f(x) - g_x(x)^T \lambda$$

$$0 = g(x).$$

DAEs can be considered to devise hybrid algorithms for NLP.

REFERENCES

[1] Mahony R. Absil, P.-A and Andrews B. Convergence of the iterates of descent methods for analytic cost functions. *SIAM J. Optim.*, 16(2):531–547, 2005.

[2] P.-A Absil and Kurdyka K. On the stable equilibrium points of gradient systems. *Systems Control Lett.*, 55(7):573–577, 2006.

[3] N. Andrei. Gradient flow algorithm for unconstrained optimization. *ICI Technical Report*, 2004.

[4] N. Andrei. An unconstrained optimization test function collection. *Adv. Mod. and Optim.*, 10(1):147–161, 2008.

[5] W. Behrman. An efficient gradient flow method for unconstrained optimization. *Ph.D. Thesis, Stanford University*, 1998.

[6] Tolle J.W. Boggs, P.T. and P. Wang. On the local convergence of quasi-Newton methods for constrained optimization. *SIAM J. Control Optim.*, 20(2):161–171, 1982.

[7] Gilbert J.C. Lemaréchal C. Bonnans, J.F. and C.A. Sagastizábal. *Numerical Optimization*. Springer, 2000.

[8] C.A. Botsaris and D.H. Jacobson. A newton-type curvilinear search method for optimization. *J. Math. Anal. Appl.*, 54:217–229, 1976.

[9] B.B. Brown and Bartholomew-Biggs M.C. Some effective methods for unconstrained optimization based on the solution of systems of ordinary differential equations. *J Optim. Theory Appl.*, 62(2):211–223, 1989.

[10] Nocedal J. Byrd, R.H. and R.B. Schnabel. Representations of quasi-Newton matrices and their use in limited memory methods. *Math. Program.*, 63(4):129–156, 1994.

[11] Gould N.I.M Conn, A.R. and Ph.L. Toint. *Trust-Region Methods*. SIAM, 2000.

[12] Gould N.I.M. Conn, A.R. and P.L. Toint. Testing a class of methods for solving minimization problems with simple bounds on the variables. *Math Comput.*, 50(182):399–430, 1988.

[13] G.J. Cooper and R. Vignesvaran. On the use of parallel processors for implicit Runge-Kutta methods. *Computing*, 51:135–150, 1993.

[14] J. Dennis and J. More. A characterization of superlinear convergence of quasi-Newton methods. *Math. Comp.*, 28:549–560, 1974.

[15] N.I.M. Gould and An Introduction to Algorithms for Nonlinear Optimization Leyffer, S. *Frontiers in Numerical Analysis*. Blowey, J.F, and Craig, A.W., Springer, 109-199, 2003.

[16] K. Gustafsson. Control-theoretic techniques for stepsize selection in implicit runge-kutta methods. *ACM T Math Software*, 20(4):496–517, 1994.

[17] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II, Second Edition*. Springer, 2002.

[18] Wanner G. Hairer, E. and S.P. Nørsett. *Solving Ordinary Differential Equations I, Second Edition*. Springer, 2000.

[19] F.S. Hillier and G.J. Lieberman. *Introduction to Operations Research*. McGraw Hill, 1990.

[20] C.T. Kelley. *Iterative Methods for Optimization*. SIAM, 1999.

[21] D.C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Math. Prog.*, 45:503–528, 1989.

[22] Kelly C.T. Liao L.-Z. Luo, X.-L. and H.W. Tam. Combining trust region techniques and rosenbrock methods to compute stationary points. *J Optim. Theory Appl.*, 140:265–286, 2009.

[23] Garbow B.S. Moré, J.J. and K.E. Hillstrom. Testing unconstrained optimization software. *ACM T. Math Software*, 7:17–41, 1981.

[24] J. Nocedal. Updating quasi-Newton matrices with limited storage. *Math. Comp.*, 35(151):773–782, 1980.

[25] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer, 2006.

[26] M.J.D. Powell. *Variable metric methods for constrained optimization, In Mathematical Programming: The State of the Art*. Springer, 1983.

[27] Chelikowsky J.R. Saad, Y. and S.M. Shontz. Numerical methods for electronic structure calculations of materials. *SIAM Rev.*, 52(1):3–54, 2010.

[28] J. Sherman and W.J. Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *Ann. Math. Statist.*, 21(1):124–127, 1950.

[29] G. Söderlind. The automatic control of numerical integration. *CWI Quarterly*, 11(1):55–74, 1998.

[30] G. Söderlind and L. Wang. Adaptive time-stepping and computational stability. *J Comput Appl Math*, 185:225–243, 2006.

[31] Gustaf Söderlind. Automatic control and adaptive time-stepping. *Numer Algorithms*, 31:281–310, 2002.

[32] Li G. Wei, Z. and L. Qi. New quasi-newton methods for unconstrained optimization problems. *Appl. Math. and Comput.*, 175:1156–1188, 2006.

[33] Qian Z. Yigui, Ou and L. Haichan. An ode-based trust region method for unconstrained optimization problems. *J Comput. and Appl. Math*, 232:318–326, 2009.