Numerical Methods for Solving Systems of Nonlinear Equations

by
Courtney Remani

A project submitted to the Department of Mathematical Sciences in conformity with the requirements for Math 4301 (Honour's Seminar)

Lakehead University
Thunder Bay, Ontario, Canada

Abstract

This Honours Seminar Project will focus on the numerical methods involved in solving systems of nonlinear equations. First, we will study Newton's method for solving multivariable nonlinear equations, which involves using the Jacobian matrix. Second, we will examine a Quasi-Newton which is called Broyden's method; this method has been described as a generalization of the Secant Method. And third, to s solve for nonlinear boundary value problems for ordinary differential equations, we will study the Finite Difference method. We will also give an application of Newton's method and the Finite Difference method. Using the computer program Matlab, we will solve a boundary value problem of a nonlinear ordinary differential system.

Acknowledgements

I would like to acknowledge and thank everyone who has been involved with this project. I would like to express my sincere gratitude to my supervisor Dr. Liping Liu. Without her knowledge, direction, guidance, and all of her help, this project would not have been achieveable. I would also like to show a great deal of appreciation to my project coordinator, Dr. Adam Van Tuyl. He has been a great professor to me over the years. I would also like to acknowledge how thankful I am for my Mom and Dad. Without their unconditional love and support, and always believing in me throughout the years, I would not have been able to achieve a lot of my goals here at Lakehead University.

Contents

CHAPTER 1

Introduction

Over the years, we have been taught on how to solve equations using various algebraic methods. These methods include the substitution method and the elimination method. Other algebraic methods that can be executed include the quadratic formula and factorization. In Linear Algebra, we learned that solving systems of linear equations can be implemented by using row reduction as an algorithm. However, when these methods are not successful, we use the concept of numerical methods.

Numerical methods are used to approximate solutions of equations when exact solutions can not be determined via algebraic methods. They construct successive approximations that converge to the exact solution of an equation or system of equations. In Math 3351, we focused on solving nonlinear equations involving only a single variable. We used methods such as Newton's method, the Secant method, and the Bisection method. We also examined numerical methods such as the Runge-Kutta methods, that are used to solve initial-value problems for ordinary differential equations. However these problems only focused on solving nonlinear equations with only one variable, rather than nonlinear equations with several variables.

The goal of this paper is to examine three different numerical methods that are used to solve systems of nonlinear equations in several variables. The first method we will look at is Newton's method. This will be followed by Broyden's method, which is sometimes called a Quasi-Newton method; it is derived from Newton's method. Lastly, we will study the Finite Difference method that is used to solve boundary value problems of nonlinear ordinary differential equations. For each method, a breakdown of each numerical procedure will be provided. In addition, there will be some discussion of the convergence of the numerical methods, as well as the advantages and disadvantages of each method. After a discussion of each of the three methods, we will use the computer program Matlab to solve an example of a nonlinear ordinary differential equation using both the Finite Diffference method and Newton's method.

CHAPTER 2

Preliminaries

In this section, we present the definitions and terms that will be used throughout the project will be presented.

§2.1 A system of nonlinear equations

Definition 2.1. A function $f\colon \mathbb{R}^n \to \mathbb{R}$ is defined as being *nonlinear* when it does not satisfy the *superposition principle* that is

$$f(x_1 + x_2 + ...) \neq f(x_1) + f(x_2) + ...$$

Now that we know what the term *nonlinear* refers to we can define a *system of nonlinear equations*.

Definition 2.2. A *system of nonlinear equations* is a set of equations as the following:

$$\begin{gathered} f_1(x_1, x_2, ..., x_n) = 0, \\ f_2(x_1, x_2, ..., x_n) = 0, \\ \vdots \\ f_n(x_1, x_2, ..., x_n) = 0, \end{gathered}$$

where $(x_1, x_2, ..., x_n) \in \mathbb{R}^n$ and each f_i is a nonlinear real function, $i = 1, 2, ..., n$.

Example 2.3. Here is an example of a nonlinear system from Burden and Faires in [**3**]:

$$\begin{aligned} 3x_1 - \cos(x_2x_3) - \frac{1}{2} &= 0 \\ x_1^2 - 81(x_2 + 0.1)^2 + \sin x_3 + 1.06 &= 0 \\ e^{-x_1x_2} + 20x_3 + \frac{10\pi - 3}{3} &= 0 \end{aligned} \tag{2.1}$$

In this article we will use the term *root* or *solution* frequently to describe the final result of solving the systems.

Definition 2.4. A *solution* of a system of equations $f_1, f_2, ..., f_n$ in n variables is a point $(a_1, ..., a_n) \in \mathbb{R}^n$ such that $f_1(a_1, ..., a_n) = \cdots = f_n(a_1, ..., a_n) = 0$.

Because systems of nonlinear equations can not be solved as nicely as linear systems, we use procedures called *iterative methods*.

Definition 2.5. An *iterative method* is a procedure that is repeated over and over again, to find the root of an equation or find the solution of a system of equations.

Definition 2.6. Let $\mathbf{F}$ be a real function from $D \subset \mathbb{R}^n$ to $\mathbb{R}^n$. If $\mathbf{F}(\mathbf{p}) = \mathbf{p}$, for some $p \in D$, then $\mathbf{p}$ is said to be a *fixed point* of $\mathbf{F}$.

§2.2 Convergence

One of the things we will discuss is the convergence of each of the numerical methods.

Definition 2.7. We say that a sequence *converges* if it has a limit.

Definition 2.8. Let p_n be a sequence that converges to p, where $p_n \neq p$. If constants $\lambda, \alpha > 0$ exist such that

$$\lim_{n\to\infty} \frac{|p_{n+1} - p|}{|p_n - p|^\alpha} = \lambda.$$

Then it is said that p_n *converges to p of order α with a constant λ.*

There are three different orders of convergences.

Definition 2.9. A sequence p_n is said to be *linearly convergent* if p_n converges to p with order $\alpha = 1$, for a constant $\lambda < 1$ such that

$$\lim_{n\to\infty} \frac{|p_{n+1} - p|}{|p_n - p|^\alpha} = \lambda$$

Definition 2.10. A sequence p_n is said to be *quadratically convergent* if p_n converges to p with order $\alpha = 2$ such that

$$\lim_{n\to\infty} \frac{|p_{n+1} - p|}{|p_n - p|^\alpha} = \lambda$$

Definition 2.11. A sequence p_n is said to be *superlinearly convergent* if

$$\lim_{n\to\infty} \frac{|p_{n+1} - p|}{|p_n - p|} = 0$$

Remark 2.12. The value of α measures how fast a sequence converges. Thus the higher the value of α is, the more rapid the convergence of the sequence is. In the case of numerical methods, the sequence of approximate solutions is converging to the root. If the convergence of an iterative method is more rapid, then a solution may be reached in less interations in comparison to another method with a slower convergence

§2.3 Jacobian Matrix

The *Jacobian matrix*, is a key component of numerical methods in the next section.

Definition 2.13. The *Jacobian matrix* is a matrix of first order partial derivatives

$$J(x) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(x) & \frac{\partial f_1}{\partial x_2}(x) & \cdots & \frac{\partial f_1}{\partial x_n}(x) \\ \frac{\partial f_2}{\partial x_1}(x) & \frac{\partial f_2}{\partial x_2}(x) & \cdots & \frac{\partial f_2}{\partial x_n}(x) \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial f_n}{\partial x_1}(x) & \frac{\partial f_n}{\partial x_2}(x) & \cdots & \frac{\partial f_n}{\partial x_n}(x) \end{bmatrix}.$$

Example 2.14. If we take the system from Example 2.3 we are able to obtain the following Jacobian Matrix:

$$J(\mathbf{x}) = \begin{bmatrix} 3 & x_3\sin(x_2x_3) & x_2\sin(x_2x_3) \\ 2x_1 & -162(x_2+0.1) & \cos x_3 \\ -x_2e^{-x_1x_2} & -x_1e^{-x_1x_2} & 20 \end{bmatrix}$$

§2.4 Hessian Matrix

The *Hessian matrix*, will be discussed in a future proof.

Definition 2.15. The *Hessian matrix* is a matrix of second order partial derivatives $\mathbf{H} = \left[\frac{\partial^2 f}{\partial x_i \partial x_j}\right]_{ij}$ such that

$$H(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f_1}{\partial x_1^2} & \frac{\partial^2 f_1}{\partial x_1\partial x_2} & \cdots & \frac{\partial^2 f_1}{\partial x_1\partial x_n} \\ \frac{\partial^2 f_2}{\partial x_2\partial x_1} & \frac{\partial^2 f_2}{\partial x_2^2} & \cdots & \frac{\partial^2 f_2}{\partial x_2\partial x_n} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial^2 f_n}{\partial x_n\partial x_1} & \frac{\partial^2 f_n}{\partial x_n\partial x_2} & \cdots & \frac{\partial^2 f_n}{\partial x_n^2} \end{bmatrix}.$$

§2.5 Norms of Vectors

Let $\mathbf{x} \in \mathbb{R}^n$ where

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

.

Definition 2.16. A *vector norm* on $\mathbb{R}^n$ is a function, $||\cdot||$, from $\mathbb{R}^n$ into $\mathbb{R}$ that has the following properties:

(1) $||\mathbf{x}|| \geq 0$ for all $\mathbf{x} \in \mathbb{R}^n$,
(2) $||\mathbf{x}|| = 0$ if and only if $\mathbf{x} = \mathbf{0}$,
(3) $||\alpha\mathbf{x}|| = |\alpha|||\mathbf{x}||$ for all $\alpha \in \mathbb{R}$ and $\mathbf{x} \in \mathbb{R}^n$,
(4) $||\mathbf{x}+\mathbf{y}|| \leq ||\mathbf{x}|| + ||\mathbf{y}||$ for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$

There are two types of vector norms we will discuss, the l_2 and l_∞ norms.

Definition 2.17. The l_2 norm for the vector $\mathbf{x}$ is called the *Euclidean norm* because it represents the length of the vector denoted by

$$||\mathbf{x}|| = ||\mathbf{x}||_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

Definition 2.18. The l_∞ norm represents the absolute value of the largest component in the vector $\mathbf{x}$. It is denoted by

$$||\mathbf{x}||_\infty = \max_{1\leq i\leq n} |x_i|.$$

The following is an example demonstrating the vector norms.

Example 2.19. The vector

$$\mathbf{x} = \begin{bmatrix} -2 \\ -1 \\ 1 \\ 3 \end{bmatrix}$$

has vector norms

$$||\mathbf{x}||_2 = \sqrt{(-2)^2 + (-1)^2 + (1)^2 + (3)^2} = \sqrt{15}$$
$$||\mathbf{x}||_\infty = \max\left(|-2|, |-1|, |1|, |3|\right) = 3$$

In the next couple of sections, we will examine three different numerical methods that will apply the terms we discussed in this section. These methods include: Newton's method, Broyden's method, and the Finite Difference method.

CHAPTER 3

Newton's Method

Newton's method is one of the most popular numerical methods, and is even referred by Burden and Faires [**3**] as the most powerful method that is used to solve for the equation $f(x) = 0$. This method originates from the Taylor's series expansion of the function $f(x)$ about the point x_1:

$$f(x) = f(x_1) + (x - x_1)f'(x_1) + \frac{1}{2!}(x - x_1)^2 f''(x_1) + \cdots \tag{3.1}$$

where f, and its first and second order derivatives, f' and f'' are calculated at x_1. If we take the first two terms of the Taylor's series expansion we have:

$$f(x) \approx f(x_1) + (x - x_1)f'(x_1). \tag{3.2}$$

We then set (3.2) to zero (i.e $f(x) = 0$) to find the root of the equation which gives us:

$$f(x_1) + (x - x_1)f'(x_1) = 0. \tag{3.3}$$

Rearranging the (3.3) we obtain the next approximation to the root, giving us:

$$x = x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} \tag{3.4}$$

Thus generalizing (3.4) we obtain Newton's iterative method:

$$x_i = x_{i-1} - \frac{f(x_{i-1})}{f'(x_{i-1})}, \ i \in \mathbb{N} \tag{3.5}$$

where $x_i \to \overline{x}$ (as $i \to \infty$), and $\overline{x}$ is the approximation to a root of the function $f(x)$.

Remark 3.1. As the iterations begin to have the same repeated values i.e. as $x_i = x_{i+1} = \overline{x}$ this is an indication that $f(x)$ converges to $\overline{x}$. Thus x_i is the root of the function $f(x)$.

Proof of Remark 3.1

Since $x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$ and if $x_i = x_{i+1}$, then

$$x_i = x_i - \frac{f(x_i)}{f'(x_i)}$$

This implies that

$$\frac{f(x_i)}{f'(x_i)} = 0$$

and thus $f(x_i) = 0$. $\square$

Another indicator that x_i is the root of the function is if it satisfies that $|f(x_i)| < \epsilon$, where $\epsilon > 0$ is a given tolerance.

However, (3.5) can only be used to solve nonlinear equations involving only a single variable. This means we have to take (3.5) and alter it, in order to use it to solve a set of nonlinear algebraic equations involving multiple variables

We know from Linear Algebra that we can take systems of equations and express those systems in the form of matrices and vectors. With this in mind and using Definition 2.2, we can express the nonlinear system as a matrix with a corresponding vector. Thus, the following equation is derived:

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} - J(\mathbf{x}^{(k-1)})^{-1}\mathbf{F}(\mathbf{x}^{(k-1)})$$

where $k = 1, 2, ..., n$ represents the iteration, $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{F}$ is a vector function, and $J(\mathbf{x})^{-1}$ is the inverse of the Jacobian matrix . This equation represents the procedure of Newton's method for solving nonlinear algebraic systems. However, instead of solving the equation $f(x) = 0$, we are now solving the system $\mathbf{F}(\mathbf{x}) = 0$. We will now go through the equation and define each component.

(1) Let $\mathbf{F}$ be a function which maps $\mathbb{R}^n$ to $\mathbb{R}^n$.

$$\mathbf{F}(x_1, x_2, ..., x_n) = \begin{bmatrix} f_1(x_1, x_2, ..., x_n) \\ f_2(x_1, x_2, ..., x_n) \\ \vdots \\ f_n(x_1, x_2, ..., x_n) \end{bmatrix}$$

where $f_i : \mathbb{R}^n \to \mathbb{R}$.

(2) Let $\mathbf{x} \in \mathbb{R}^n$. Then $\mathbf{x}$ represents the vector

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

where $x_i \in \mathbb{R}$ and $i = 1, 2, \ldots, n$.

(3) From Definition 2.13 we know that $J(\mathbf{x})$ is the Jacobian matrix. Thus $J(\mathbf{x})^{-1}$ is

$$J(\mathbf{x})^{-1} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) & \frac{\partial f_1}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_1}{\partial x_n}(\mathbf{x}) \\ \frac{\partial f_2}{\partial x_1}(\mathbf{x}) & \frac{\partial f_2}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_2}{\partial x_n}(\mathbf{x}) \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial f_n}{\partial x_1}(\mathbf{x}) & \frac{\partial f_n}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_n}{\partial x_n}(\mathbf{x}) \end{bmatrix}^{-1}$$

Now we describe the steps of Newton's method:

Step 1:
Let $\mathbf{x}^{(0)} = (x_1^{(0)}, x_2^{(0)}, ..., x_n^{(0)})$ be a given initial vector.

Step 2:
Calculate $J(\mathbf{x}^{(0)})$ and $\mathbf{F}(\mathbf{x}^{(0)})$.

Step 3:
We now have to calculate the vector $\mathbf{y}^{(0)}$, where

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

In order to find $\mathbf{y}^{(0)}$, we solve the linear system $J(\mathbf{x}^{(0)})\mathbf{y}^{(0)} = -\mathbf{F}(\mathbf{x}^{(0)})$, using Gaussian Elimination.

REMARK 3.2. Rearranging the system in Step 3, we get that $\mathbf{y}^{(0)} = -J(\mathbf{x}^{(0)})^{-1}\mathbf{F}(\mathbf{x}^{(0)})$. The significance of this is that, since $\mathbf{y}^{(0)} = -J(\mathbf{x}^{(0)})^{-1}\mathbf{F}(\mathbf{x}^{(0)})$, we can replace $-J(\mathbf{x}^{(0)})^{-1}\mathbf{F}(\mathbf{x}^{(0)})$ in our iterative formula with $\mathbf{y}^{(0)}$. This result will yield that

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} - J(\mathbf{x}^{(k-1)})^{-1}\mathbf{F}(\mathbf{x}^{(k-1)}) = \mathbf{x}^{(k-1)} - \mathbf{y}^{(k-1)}$$

Step 4:
Once $\mathbf{y}^{(0)}$ is found, we can now proceed to finish the first iteration by solving for $\mathbf{x}^{(1)}$. Thus using the result from Step 3, we have that

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \mathbf{y}^{(0)} = \begin{bmatrix} x_1^{(0)} \\ x_2^{(0)} \\ \vdots \\ x_n^{(0)} \end{bmatrix} + \begin{bmatrix} y_1^{(0)} \\ y_2^{(0)} \\ \vdots \\ y_n^{(0)} \end{bmatrix}$$

Step 5:
Once we have calculated $\mathbf{x}^{(1)}$, we repeat the process again, until $\mathbf{x}^{(k)}$ converges to $\overline{\mathbf{x}}$. This indicates we have reached the solution to $\mathbf{F}(\mathbf{x}) = \mathbf{0}$, where $\overline{\mathbf{x}}$ is the solution to the system.

Remark 3.3. When a set of vectors converges, the norm $||\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}|| = 0$. This means that

$$||\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}|| = \sqrt{(x_1^{(k)} - x_1^{(k-1)})^2 + \cdots + (x_n^{(k)} - x_n^{(k-1)})^2} = 0$$

§3.2 Convergence of Newton's Method

Newton's method converges quadratically, (refer to definition 2.10). When carrying out this method the system converges quite rapdily once the approximation is close to the actual solution of the nonlinear system. This is seen as a advantage because Newton's method may require less iterations, compared to another method with a lower rate of convergence, to reach the solution. However, when the system does not converge, this is an indicator that an error in the computations has occured, or a solution may not exist.

In the following proof, we will prove that Newton's method does indeed converge quadratically.

Proof of Newton's Method Quadratic Convergence

In order for Newton's method to converge quadratically, the initial vector $\mathbf{x}^{(0)}$ must be sufficiently close to a the solution of the system $\mathbf{F}$=$\mathbf{0}$, which is denoted by $\overline{\mathbf{x}}$. As well, the Jacobian matrix at must not be singular, that is, $J(\mathbf{x})^{-1}$ must exist. The goal of this proof is to show that

$$\frac{||\mathbf{x}^{(k+1)} - \overline{\mathbf{x}}||}{||\mathbf{x}^{(k)} - \overline{\mathbf{x}}||^2} = \lambda$$

where λ denotes a positive constant.
We have that

$$||\mathbf{e}^{(k+1)}|| = ||\mathbf{x}^{(k+1)} - \overline{\mathbf{x}}|| = ||\mathbf{x}^{(k)} - J(\mathbf{x}^{(k)})^{-1}\mathbf{F}(\mathbf{x}^{(k)}) - \overline{\mathbf{x}}||.$$

If we set $||\mathbf{e}^{(k)}|| = ||\mathbf{x}^{(k)} - \overline{\mathbf{x}}||$ then have we that

$$||\mathbf{e}^{(k)} - J(\mathbf{x}^{(k)})^{-1}\mathbf{F}(\mathbf{x}^{(k)})||. \tag{3.6}$$

Next, we want to define the second-order Taylor series as

$$\mathbf{F}(\mathbf{x}^{(k)}) \approx \mathbf{F}(\overline{\mathbf{x}}) + \mathbf{J}\mathbf{e}^{(k)} + \frac{1}{2}(\mathbf{e}^{(k)})^T\mathbf{H}(\mathbf{e}^{(k)})$$

where $\mathbf{J} = J(\mathbf{x}^{(k)})$ and $\mathbf{H}$ is the Hessian tensor, which is similiar to the Hessian matrix, i.e. $\mathbf{H} = \left[\frac{\partial^2 f}{\partial x_i \partial x_j}\right]_{ij}$, when $\mathbf{F} = f$. We then have to multiply each side of the Taylor's series by $\mathbf{J}^{-1}$, which yields

$$\begin{aligned}\mathbf{J}^{-1}(\mathbf{F}(\mathbf{x}^{(k)})) &\approx \mathbf{J}^{-1}\left[\mathbf{F}(\overline{\mathbf{x}}) + \mathbf{J}\mathbf{e}^{(k)} + \frac{1}{2}(\mathbf{e}^{(k)})^T\mathbf{H}(\mathbf{e}^{(k)})\right] \\ &= \mathbf{e}^{(k)} + \frac{\mathbf{J}^{-1}}{2}(\mathbf{e}^{(k)})^T\mathbf{H}(\mathbf{e}^{(k)}) \end{aligned} \tag{3.7}$$

Using (3.6) and (3.7) we obtain our last result such that,

$$\begin{aligned}||\mathbf{x}^{(k+1)} - \overline{\mathbf{x}}|| &= ||\mathbf{e}^{(k+1)}|| \\ &= \left|\left|\frac{\mathbf{J}^{-1}}{2}(\mathbf{e}^{(k)})^T\mathbf{H}(\mathbf{e}^{(k)})\right|\right| \\ &\leq \frac{||\mathbf{J}^{-1}||||\mathbf{H}||}{2}||\mathbf{e}^{(k)}||^2.\end{aligned}$$

Thus is shows that Newton's method converges quadratically. □

§3.3 Advantages and Disadvantages of Newton's Method

One of the advantages of Newton's method is that its not too complicated in form and it can be used to solve a variety of problems. The major disadvantage associated with Newton's method, is that $J(\mathbf{x})$, as well as its inversion has, to be calculated for each iteration. Calculating both the Jacobian matrix and its inverse can be quite time consuming depending on the size of your system is. Another problem that we may be challenged with when using Newton's method is that it may fail to converge. If Newton's method fails to converge this will result in an oscillation between points.

§3.4 A Numerical Example of Newton's Method

The following example is a numerical application of Newton's method from [**3**].

Example 3.4. Solve the following nonlinear system

$$\begin{aligned}3x_1 - \cos(x_2x_3) - \frac{1}{2} &= 0,\\ x_1^2 - 81(x_2+0.1)^2 + \sin x_3 + 1.06 &= 0,\\ e^{-x_1x_2} + 20x_3 + \frac{10\pi - 3}{3} &= 0,\end{aligned}$$

when the initial approximation is

$$\mathbf{x}^{(0)} = \begin{bmatrix} 0.1 \\ 0.1 \\ -0.1 \end{bmatrix}$$

Solution

Step 1: We have our initial vector

$$\mathbf{x}^{(0)} = \begin{bmatrix} 0.1 \\ 0.1 \\ -0.1 \end{bmatrix}.$$

Step 2: Define $\mathbf{F}(\mathbf{x})$ and $J(\mathbf{x})$:

$$\mathbf{F}(\mathbf{x}) = \begin{bmatrix} 3x_1 - \cos(x_2x_3) - \frac{1}{2} \\ x_1^2 - 81(x_2+0.1)^2 + \sin x_3 + 1.06 \\ e^{-x_1x_2} + 20x_3 + \frac{10\pi-3}{3} \end{bmatrix}$$

$$J(\mathbf{x}) = \begin{bmatrix} 3 & x_3\sin(x_2x_3) & x_2\sin(x_2x_3) \\ 2x_1 & -162(x_2+0.1) & \cos x_3 \\ -x_2e^{-x_1x_2} & -x_1e^{-x_1x_2} & 20 \end{bmatrix}$$

Now that we have defined $\mathbf{F}(\mathbf{x})$ and $J(\mathbf{x})$, we now want to calculate $\mathbf{F}(\mathbf{x}^{(0)})$ and $J(\mathbf{x}^{(0)})$, where $\mathbf{x}^{(0)} = (0.1, 0.1, -0.1)^{\intercal}$:

$$\begin{aligned}\mathbf{F}(\mathbf{x}^{(0)}) &= \begin{bmatrix} 0.3 - \cos(-0.01) - \frac{1}{2} \\ 0.01 - 3.24 + \sin(-0.1) + 1.06 \\ e^{(-0.01)} - 2 + \frac{10\pi-3}{3} \end{bmatrix}\\ &= \begin{bmatrix} -1.19995 \\ -2.269833417 \\ 8.462025346 \end{bmatrix}\end{aligned}$$

and

$$J(\mathbf{x}^{(0)}) = \begin{bmatrix} 3 & (-0.1)\sin(-0.01) & 0.1\sin(-0.01) \\ 0.2 & -32.4 & \cos(-0.1) \\ -0.1e^{-0.01} & -0.1e^{-0.01} & 20 \end{bmatrix}$$
$$= \begin{bmatrix} 3 & 0.000999983 & -0.000999983 \\ 0.2 & -32.4 & 0.995004165 \\ -0.099004984 & -0.099004983 & 20 \end{bmatrix}$$

Step 3: Solve the system $J(\mathbf{x}^{(0)})\mathbf{y}^{(0)} = -\mathbf{F}(\mathbf{x}^{(0)})$, using Gaussian Elimination:

$$\begin{bmatrix} 3 & 0.000999983 & -0.000999983 \\ 0.2 & -32.4 & 0.995004165 \\ -0.099004984 & -0.099004983 & 20 \end{bmatrix} \begin{bmatrix} y_1^{(0)} \\ y_2^{(0)} \\ y_3^{(0)} \end{bmatrix} = - \begin{bmatrix} -1.19995 \\ -2.269833417 \\ 8.462025346 \end{bmatrix}$$

After solving the linear system above it yields the result

$$\mathbf{y}^{(0)} = \begin{bmatrix} 0.40003702 \\ -0.08053314 \\ -0.42152047 \end{bmatrix}$$

Step 4: Using the result in Step 3, compute $\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \mathbf{y}^{(0)}$:

$$\mathbf{x}^{(1)} = \begin{bmatrix} 0.1 \\ 0.1 \\ -0.1 \end{bmatrix} + \begin{bmatrix} 0.40003702 \\ -0.08053314 \\ -0.42152047 \end{bmatrix}$$
$$= \begin{bmatrix} 0.50003702 \\ 0.01946686 \\ -0.52152047 \end{bmatrix}$$

We can use the results of $\mathbf{x}^{(1)}$ to find our next iteration $\mathbf{x}^{(2)}$ by using the same procedure.

Step 5: If we continue to repeat the process, we will get the following results:

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$	$\|\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\|\|$
0	0.10000000	0.10000000	-0.10000000	–
1	0.50003702	0.01946686	-0.52152047	0.422
2	0.50004593	0.00158859	-0.52355711	0.0179
3	0.50000034	0.00001244	-0.52359845	0.00158
4	0.50000000	0.00000000	-0.52359877	0.0000124
5	0.50000000	0.00000000	-0.52359877	0

From Remark 3.3 we know that when a set of vectors converges the norm

$$||\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}|| = 0.$$

Thus by our table above, the norm is equal to zero at the fifth iteration. This indicates that our system $\mathbf{F}(\mathbf{x})$ has converged to the solution, which will be denoted by $\overline{\mathbf{x}}$.

Therefore, from our table of our results we know that

$$\overline{\mathbf{x}} = \begin{bmatrix} 0.50000000 \\ 0.00000000 \\ -0.52359877 \end{bmatrix}$$

is an approximation solution of $\mathbf{F}(\mathbf{x}) = 0$.

There are methods that are in the same family of Newton's method, identified as Quasi-Newton methods. A specific Quasi-Newton method, known as Broyden's method, will be examined in the next section.

CHAPTER 4

Broyden's Method

In the last chapter, we examined the numerical method known as Newton's method. We established that one of the major disadvantages of this method was that that $J(\mathbf{x})$ and its inverse must be computed at each iteration. We, therefore want to avoid this problem. There are methods known as Quasi-Newton methods, in which Burden and Faires in [**3**] describe as methods that use an approximation matrix that is updated at each iteration in place of the Jacobian matrix. This implies that the form of the iterative procedure for Broyden's method is almost identical to that used in Newton's method. The only exception being that an approximation matrix A_i is implemented instead of $J(\mathbf{x})$. With that said the following equation is derived:

$$\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - A_i^{-1}\mathbf{F}(\mathbf{x}^{(i)}).$$

This is defined as *Broyden's iterative procedure.* .

In [**3**], A_i is defined as

$$A_i = A_{i-1} + \frac{\mathbf{y}_i - A_{i-1}\mathbf{s}_i}{||\mathbf{s}_i||_2^2}\mathbf{s}_i^t$$

$\mathbf{y}_i = \mathbf{F}(\mathbf{x}^{(i)}) - \mathbf{F}(\mathbf{x}^{(i-1)})$ and $\mathbf{s}_i = \mathbf{x}^{(i)} - \mathbf{x}^{(i-1)}$. However, in Broyden's method it involves that computation A_i^{-1}, not A_i, which brings us to the next theorem.

THEOREM 4.1. *(Sherman-Morrison Forumula) If A is a nonsingular matrix and $\boldsymbol{x}$ and $\boldsymbol{y}$ are vectors, then $A + \boldsymbol{x}\boldsymbol{y}^t$ is nonsingular provided that $\boldsymbol{y}^t A^{-1}\boldsymbol{x} \neq -1$ and*

$$(A + \boldsymbol{x}\boldsymbol{y}^t)^{-1} = A^{-1} - \frac{A^{-1}\boldsymbol{x}\boldsymbol{y}^t A^{-1}}{1 + \boldsymbol{y}^t A^{-1}\boldsymbol{x}}.$$

The Sherman-Morrison Formula from [**3**], is a matrix inversion formula. It allows A_i^{-1} to be computed directly using A_{i-1}^{-1}, rather than computing A_i and then its inverse at each iteration. Now by using Theorem 4.1 and letting $A = A_{i-1}$, $\mathbf{x} = \frac{\mathbf{y}_i - A_{i-1}\mathbf{s}_i}{||\mathbf{s}_i||_2^2}$, and $\mathbf{y} = \mathbf{s}_i$, as well as using A_i as defined above we have that

$$A_i^{-1} = \left(A_{i-1} + \frac{\mathbf{y}_i - A_{i-1}\mathbf{s}_i}{||\mathbf{s}_i||_2^2}\mathbf{s}_i^t\right)^{-1}$$

$$= A_{i-1}^{-1} - \frac{A_{i-1}^{-1}\left(A_{i-1} + \frac{\mathbf{y}_i - A_{i-1}\mathbf{s}_i}{||\mathbf{s}_i||_2^2}\mathbf{s}_i^t\right)A_{i-1}^{-1}}{1 + \mathbf{s}_i^t A_{i-1}^{-1}\left(\frac{\mathbf{y}_i - A_{i-1}\mathbf{s}_i}{||\mathbf{s}_i||_2^2}\right)}$$

$$= A_{i-1}^{-1} - \frac{(A_{i-1}^{-1}y_i - \mathbf{s}_i)\mathbf{s}_i^t A_{i-1}^{-1}}{||\mathbf{s}_i||_2^2 + \mathbf{s}_i^t A_{i-1}^{-1}\mathbf{y}_i - ||\mathbf{s}_i||_2^2}$$

This leaves us with

$$A_i^{-1} = A_{i-1}^{-1} + \frac{(\mathbf{s}_i - A_{i-1}^{-1}\mathbf{y}_i)\mathbf{s}_i^t A_{i-1}^{-1}}{\mathbf{s}_i^t A_{i-1}^{-1}\mathbf{y}_i}.$$

We compute the inverse of the approximation matrix at each iteration with this equation.

We now desribe the steps of Broyden's method:

Step 1:
Let $\mathbf{x}^{(0)} = (x_1^{(0)}, x_2^{(0)}, ..., x_n^{(0)})$ be the initial vector given.

Step 2:
Calculate $\mathbf{F}(\mathbf{x}^{(0)})$.

Step 3:
In this step we compute A_0^{-1}. Because we do not have enough information to compute A_0 directly, Broyden's method permits us to let $A_0 = J(\mathbf{x}^{(0)})$, which implies that $A_0^{-1} = J(\mathbf{x}^{(0)})^{-1}$.

Step 4:
Calculate $\mathbf{x}^{(1)} = \mathbf{x}^{(0)} - A_0^{-1}\mathbf{F}(\mathbf{x}^{(0)})$.

Step 5:
Calculate $\mathbf{F}(\mathbf{x}^{(1)})$.

Step 6:
Take $\mathbf{F}(\mathbf{x}^{(0)})$ and $\mathbf{F}(\mathbf{x}^{(1)})$ and calculate $\mathbf{y}_1 = \mathbf{F}(\mathbf{x}^{(1)}) - \mathbf{F}(\mathbf{x}^{(0)})$. Next, take the first two iterations of $\mathbf{x}^{(i)}$ and calculate $\mathbf{s}_1 = \mathbf{x}^{(1)} - \mathbf{x}^{(0)}$.

Step 7:
Calculate $\mathbf{s}_1^t A_0^{-1}\mathbf{y}_1$.

Step 8:
Compute $A_1^{-1} = A_0^{-1} + \left(\frac{1}{\mathbf{s}_1^t A_0^{-1}\mathbf{y}_1}\right)\left[(\mathbf{s}_1 - A_0^{-1}\mathbf{y}_1)\mathbf{s}_1^t A_0^{-1}\right]$

Step 9:
Take A_1^{-1} that we found in Step 8, and calculate $\mathbf{x}^{(2)} = \mathbf{x}^{(1)} - A_1^{-1}\mathbf{F}(\mathbf{x}^{(1)})$.

Step 10:
Repeat the process until we converge to $\overline{\mathbf{x}}$, i.e. when $\mathbf{x}^{(i)} = \mathbf{x}^{(i+1)} = \overline{\mathbf{x}}$. This will indicate that we have reached the solution of the system (refer to Remark 3.3).

§4.2 Convergence of Broyden's Method

Unlike Newton's method, Broyden's method as well as all of the Quasi-Newton methods converge superlinearlly. This means that

$$\lim_{i\to\infty} \frac{||\mathbf{x}^{(i+1)} - \mathbf{p}||}{||\mathbf{x}^{(i)} - \mathbf{p}||} = 0$$

where $\mathbf{p}$ is the solution to $\mathbf{F}(\mathbf{x}) = 0$, and $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(i+1)}$ are successive approximations to $\mathbf{p}$. This can be proved in a similar manner that proved the convergence of Newton's method.

§4.3 Advantages and Disadvantages Of Broyden's Method

The main advantage of Broyden's method is the reduction of computations. More specifically, the way the inverse of the approximation matrix, A_i^{-1} can be computed directly from the previous iteration, A_{i-1}^{-1} reduces the number of computations needed for this method in comparison to Newton's Method. One thing that is seen as a disadvantage of this Quasi-Newton method is that it does not converge quadratically. This may mean that more iterations may be needed to reach the solution, when compared to the number of iterations Newton's method requires. Another disadvantage of Broyden's method is that as described in [**3**] by Burden and Faires, is that it is not self-correcting. This means that in contrast to Newton's method, it does not correct itself for round off errors with consecutive interations. This may cause only a slight inaccuracy in the iterations compared to Newton's, but the final iteration will be the same.

Now that we have taken a look at numerical methods for solving multivariable nonlinear equations, in the next section we will focus on a numerical method that is used to nonlinear boundary value problems for ordinary differential equations.

CHAPTER 5

Finite-Difference Method

In this section, we will examine a numerical method that is used to approximate the solution of a boundary-value problem. We will focus on a two-point boundary-value problem with a second order differential equation which takes the form

$$y'' = f(x, y, y'),\ a \leq x \leq b,$$

$$y(a) = \alpha,\ y(b) = \beta$$

where f is a function, a and b are the end points, and $y(a) = \alpha$ and $y(b) = \beta$ are the boundary conditions.

Example 5.1. The following example is of a two-point boundary value problem with a second order differential equation from [**4**]:

$$y'' = \frac{1}{8}(32 + 2x^3 - yy'),\ 1 \leq x \leq 3$$

$$y(1) = 17,\ y(3) = \frac{43}{3}$$

Before we can solve a boundary value problem we have to be sure it has a unique solution. The following theorem from [**4**] ensures that a solution indeed does exist and is unique.

Theorem 5.2. *Suppose the function f in the boundary-value problem*

$$y'' = f(x, y, y'),\ a \leq x \leq b,\ y(a) = \alpha,\ y(b) = \beta$$

is continuous on the set

$$D = ((x, y, y')|a \leq x \leq b, -\infty < y < \infty, -\infty < y' < \infty)$$

and that the partial derivatives f_y and $f_{y'}$ are also continuous in D. If

(1) $f_y(x, y, y') > 0$ *for all* $(x, y, y') \in D$*, and*

(2) *a constant M exists with* $|f_{y'}(x, y, y')| \leq M$ *for all* $(x, y, y') \in D$*,*

then the boundary-value problem has a unique solution.

The numerical method we will be looking at is the Finite-Difference method. This method can be used to solve both linear and nonlinear ordinary differential equations. We will just survey the nonlinear Finite-Difference method.

A nonlinear boundary-value problem takes on the form of

$$y'' = f(x, y, y'),\ a \leq x \leq b,\ y(a) = \alpha,\ y(b) = \beta$$

In order for the Finite-Difference method to be carried out we have to assume f satisfies the following conditions as described in [**4**]:

(1) f and the partial derivatives f_y and $f_{y'}$ are all continuous on

$$D = ((x, y, y')|a \leq x \leq b, -\infty < y < \infty, -\infty < y' < \infty)$$

(2) $f_y(x, y, y') \geq \delta$ on D,for some $\delta > 0$.

(3) Constants k and L exist, with

$$k = \max_{(x,y,y')\in D} |f_y(x, y, y')|,\ \mathit{and}\ L = \max_{(x,y,y')\in D} |f_{y'}(x, y, y')|$$

With f satisfying these conditions, Theorem 5.2 implies that a unique solution exists.

When solving a linear boundary-value problem using the Finite-Difference, the second-order boundary-value equation

$$y'' = p(x)y' + q(x)y + r(x)$$

is expanded using y in a third Taylor polynomial about x_i evaluated at x_{i+1} and x_{i-1}, where a formula called the *centered-difference formula* for both $y''(x_i)$ and $y'(x_i)$ is derived. Burden and Faires in [**4**] define the *centered-difference formula* for $y''(x_i)$ and $y'(x_i)$ as follows

$$y''(x_i) = \frac{1}{h^2}[y(x_{i+1}) - 2y(x_i) + y(x_{i-1})] - \frac{h^2}{12}y^{(4)}(\xi_i) \tag{5.1}$$

for some ξ_i in (x_{i-1}, x_{i+1}), and

$$y'(x_i) = \frac{1}{2h}[y(x_{i+1}) - y(x_{i-1})] - \frac{h^2}{6}y'''(\eta_i) \tag{5.2}$$

for some η_i in (x_{i-1}, x_{i+1}).

Now we can begin to form the procedure for the Finite-Difference method.

Step 1:

We first want to divide the interval $[a, b]$ into $(N + 1)$ equal subintervals which gives us

$$h = \frac{(b - a)}{(N + 1)}$$

with end points at $x_i = a + ih$ for $i = 0, 1, 2, ..., N + 1$.

Step 2:
Next we will take

$$y''(x_i) = f(x_i, y(x_i), y'(x_i))$$

and substitute equations (5.1) and (5.2) into it. This will give us:

$$\frac{y(x_{i+1}) - 2y(x_i) + y(x_{i-1})}{h^2} = f\left(x_i, y(x_i), \frac{y(x_{i+1}) - y(x_{i-1})}{2h} - \frac{h^2}{6}y'''(\eta_i)\right) + \frac{h^2}{12}y^{(4)}(\xi_i) \tag{5.3}$$

for some ξ_i and η_i in the interval (x_{i-1}, x_{i+1}).

Step 3:
The Finite-Difference method results by using (5.3), and the boundary conditions to define:

$$w_0 = \alpha,\ w_{N+1} = \beta$$

and

$$-\frac{w_{i+1} - 2w_i + w_{i-1}}{h^2} + f\left(x_i, w_i, \frac{w_{i+1} - w_{i-1}}{2h}\right) = 0$$

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

Step 4:
Once we define the boundary conditions in Step 3, an $N \times N$ nonlinear system, $F(\mathbf{w})$, is produced from the Finite Difference method defined in [**4**] as:

$$\begin{aligned}
2w_1 - w_2 + h^2 f\left(x_1, w_1, \frac{w_2 - \alpha}{2h}\right) - \alpha &= 0 \\
-w_1 + 2w_2 - w_3 + h^2 f\left(x_2, w_2, \frac{w_3 - w_1}{2h}\right) &= 0 \\
&\vdots \\
-w_{N-2} + 2w_{N-1} - w_N + h^2 f\left(x_{N-1}, w_{N-1}, \frac{w_N - w_{N-2}}{2h}\right) &= 0 \\
-w_{N-1} + 2w_N + h^2 f\left(x_N, w_N, \frac{\beta - w_{N-1}}{2h}\right) - \beta &= 0
\end{aligned} \tag{5.4}$$

Step 5:
We can take $\mathbf{F}(\mathbf{w})$, and implement Newton's method to approximate the solution to this system. We can do this by taking an initial approximation $\mathbf{w}^{(0)} = (w_1^{(0)}, w_2^{(0)}, ..., w_N^{(0)})^t$, $\mathbf{F}(\mathbf{w}^{(0)})$ and defining the Jacobian matrix as follows:

$$J(w_1, w_2, ..., w_N)_{ij} = -1 + \frac{h}{2} f_{y'}\left(x_i, w_i \frac{w_{i+1} - w_{i-1}}{2h}\right),\ \mathit{for}\ i = j-1\ \mathit{and}\ j = 2, ..., N$$
$$J(w_1, w_2, ..., w_N)_{ij} = 2 + h^2 f_y\left(x_i, w_i \frac{w_{i+1} - w_{i-1}}{2h}\right),\ \mathit{for}\ i = j\ \mathit{and}\ j = 1, ..., N$$
$$J(w_1, w_2, ..., w_N)_{ij} = -1 - \frac{h}{2} f_{y'}\left(x_i, w_i \frac{w_{i+1} - w_{i-1}}{2h}\right),\ \mathit{for}\ i = j+1\ \mathit{and}\ j = 1, ..., N-1 \tag{5.5}$$

where $w_0 = \alpha$ and $w_{N+1} = \beta$.

REMARK 5.3. We can find the initial approximation $\mathbf{w}^{(0)}$ by using the following equation equation

$$\mathbf{w}^{(0)} = \alpha + \frac{\beta - \alpha}{b - a}(x_i - a)$$

where $x_i = a + ih$ for $i = 1, 2, ..., N$

In the Finite-Difference method, $J(w_1, w_2, ..., w_N)$ is *tridiagonal* with ijth entry. This means that there are non-zero entries on the main diagonal, non-zero entries on the diagonal directly below the main diagonal, and there are non-zero entries on the diagonal directly above the main diagonal.

If we look at Step 3 of Newton's method in Chapter 3, we solve the system $J(\mathbf{x})\mathbf{y} = -\mathbf{F}(\mathbf{x})$. Now for the Finite Difference method we solve a similiar system that is

$$J(w_1, ..., w_N)(v_1, ..., v_n)^t = -\mathbf{F}(w_1, w_2, ..., w_N)$$

where $\mathbf{w}_i^{(k)} = \mathbf{w}_i^{(k-1)} + \mathbf{v}_i$, for each $i = 1, 2, ..., N$. However, we do not use Gaussian Elimination to solve this system. Since the Jacobian matrix is tridiagonal, we can solve it using Crout LU factorization for matrices such that $J(\mathbf{w}) = LU$.

Crout LU Factorization

Since $J(\mathbf{w})$ is tridiagonal, it takes on the form:

$$J(\mathbf{w}) = \begin{bmatrix} a_{11} & a_{12} & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ a_{21} & a_{22} & a_{23} & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 & \cdots & \cdots & 0 \\ \vdots & 0 & \ddots & \ddots & \ddots & 0 & \cdots & 0 \\ \vdots & \vdots & 0 & \ddots & \ddots & \ddots & 0 & 0 \\ \vdots & \vdots & \vdots & 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \vdots & \vdots & \vdots & 0 & \ddots & \ddots & a_{i-1,j} \\ 0 & 0 & 0 & 0 & 0 & 0 & a_{i,j-1} & a_{ij} \end{bmatrix}.$$

Crout's LU Factorization factors the matrix above into two triangular matrices L and U. These two matrices can be found in the form:

$$L = \begin{bmatrix} l_{11} & 0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ l_{21} & l_{22} & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & l_{32} & l_{33} & 0 & \cdots & \cdots & \cdots & 0 \\ \vdots & 0 & \ddots & \ddots & \ddots & 0 & \cdots & 0 \\ \vdots & \vdots & 0 & \ddots & \ddots & \ddots & 0 & 0 \\ \vdots & \vdots & \vdots & 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \vdots & \vdots & \vdots & 0 & \ddots & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & l_{i,j-1} & l_{ij} \end{bmatrix}.$$

and

$$U = \begin{bmatrix} 1 & u_{12} & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & 1 & u_{23} & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & 0 & 1 & u_{34} & 0 & \cdots & \cdots & 0 \\ \vdots & 0 & \ddots & \ddots & \ddots & 0 & \cdots & 0 \\ \vdots & \vdots & 0 & \ddots & \ddots & \ddots & 0 & 0 \\ \vdots & \vdots & \vdots & 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \vdots & \vdots & \vdots & 0 & \ddots & \ddots & u_{i-1,j} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Once we have expressed our original matrix $J(\mathbf{w})$ in terms of L and U, we need to compute the entries of each of these matrices. This procedure involves:

(1) Computing the first column of L, where $l_{i1} = a_{i1}$

(2) Computing the first row of U, where $u_{1j} = \frac{a_{1j}}{l_{11}}$

(3) Alternately computing the columns of L and the rows of U, where

$$l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj},\ \ for\ j \leq i,\ \ i = 1, 2, ..., N$$

$$u_{ij} = \frac{a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj}}{l_{ii}},\ \ for\ i \leq j,\ \ j = 2, 3, ..., N$$

Once the entries of the LU matrices are determined, we want to solve the system $J(w_1, ..., w_N)(v_1, ..., v_n)^t = -\mathbf{F}(w_1, w_2, ..., w_N)$.

We solve this system using the following procedure:

(1) Set up and solve the system $L\mathbf{z} = \mathbf{F}(\mathbf{w}^{(k)})$, where $\mathbf{z} \in \mathbb{R}^n$.

(2) Set up and solve the system $U\mathbf{v} = \mathbf{z}$. (Remember $\mathbf{v} = (v_1, ..., v_n)^t$)

Once we are able to obtain $\mathbf{v}$, we can proceed with computing $\mathbf{w}_i^{(k)} = \mathbf{w}_i^{(k-1)} + \mathbf{v}_i$, and thus repeating Newton's method for the next iteration.

As a result once we can obtain the initial approximation $\mathbf{w}^{(0)}$ and form a $N \times N$ system, we can follow the iterative process for Newton's method described in Chapter 3, with the addition of Crout's LU factorization in place of the Gaussian Elimination, to solve the boundary-value probem, i.e. the values of $y(x_i)$, where $x_i = a + ih$ and $i = 0, 1, 2, ..., N + 1$. This implies that the procedure for the Finite-Difference method consists of converting the boundary-value problem into a nonlinear algebraic system. Once a nonlinear algebraic system is formulated, we can use Newton's method to solve this system.

In the next section we will take a numerical example and solve a nonlinear boundary-value problem using the computer program Matlab.

CHAPTER 6

Matlab Application

In this section, we will solve the boundary value problem of nonlinear ordinary differential equation from Example 5.1:

$$y'' = \frac{1}{8}(32 + 2x^3 - yy'),\ 1 \leq x \leq 3$$
$$y(1) = 17,\ y(3) = \frac{43}{3}$$

with $h = 0.1$

There are a few things that we have to compute before we can solve this problem using Matlab.

Step 1:
Since we know that $h = 0.1$, this means that our interval $[1,3]$ is divided into $N+1 = 19+1 = 20$ equal subintervals. We also know from Chapter 5, that $x_i = a + ih$, where $i = 0, 1, 2, ..., N+1$. This implies that our values of x_i are as follows (refer to next page):

i	x_i
0	1.0
1	1.1
2	1.2
3	1.3
4	1.4
5	1.5
6	1.6
7	1.7
8	1.8
9	1.9
10	2.0
11	2.1
12	2.2
13	2.3
14	2.4
15	2.5
16	2.6
17	2.7
18	2.8
19	2.9
20	3.0

Step 2:
Next we will define the boundary conditions such that $w_0 = 17$ and $w_{20} = 14.333333$.

Step 3:
Using the equation from Remark 5.3 we want to define our initial approximation $\mathbf{w}^{(0)}$. The equation yields the following results:

$$\begin{aligned}\mathbf{w}^{(0)} = (&16.86666667, 16.73333333, 16.6, 16.46666667, 16.33333333,\\ &16.2, 16.06666667, 15.9333333, 15.8, 15.66666667, 15.53333333,\\ &15.4, 15.26666667, 15.13333333, 15, 14.86666667, 14.733333333,\\ &14.6, 14.46666667)^t\end{aligned}$$

Step 4:
We know that $N = 19$, which implies that $F(\mathbf{w})$ is 19×19 nonlinear system. Using (5.4)

we get that $F(\mathbf{w})$ is:

$$2w_1 - w_2 + 0.01\left(4 + 0.33275 + \frac{w_1(w_2 - 17)}{1.6}\right) - 17 = 0$$

$$-w_1 + 2w_2 - w_3 + 0.01\left(4 + 0.432 + \frac{w_2(w_3 - w_1)}{1.6}\right) = 0$$

$$-w_2 + 2w_3 - w_4 + 0.01\left(4 + 0.5495 + \frac{w_3(w_4 - w_2)}{1.6}\right) = 0$$

$$-w_3 + 2w_4 - w_5 + 0.01\left(4 + 0.686 + \frac{w_4(w_5 - w_3)}{1.6}\right) = 0$$

$$-w_4 + 2w_5 - w_6 + 0.01\left(4 + 0.84375 + \frac{w_5(w_6 - w_4)}{1.6}\right) = 0$$

$$-w_5 + 2w_6 - w_7 + 0.01\left(4 + 1.024 + \frac{w_6(w_7 - w_5)}{1.6}\right) = 0$$

$$-w_6 + 2w_7 - w_8 + 0.01\left(4 + 1.22825 + \frac{w_7(w_8 - w_6)}{1.6}\right) = 0$$

$$-w_7 + 2w_8 - w_9 + 0.01\left(4 + 1.458 + \frac{w_8(w_9 - w_7)}{1.6}\right) = 0$$

$$-w_8 + 2w_9 - w_{10} + 0.01\left(4 + 1.71475 + \frac{w_9(w_{10} - w_8)}{1.6}\right) = 0$$

$$-w_9 + 2w_{10} - w_{11} + 0.01\left(4 + 2 + \frac{w_{10}(w_{11} - w_9)}{1.6}\right) = 0$$

$$-w_{10} + 2w_{11} - w_{12} + 0.01\left(4 + 2.31525 + \frac{w_{11}(w_{12} - w_{10})}{1.6}\right) = 0$$

$$-w_{11} + 2w_{12} - w_{13} + 0.01\left(4 + 2.662 + \frac{w_{12}(w_{13} - w_{11})}{1.6}\right) = 0$$

$$-w_{12} + 2w_{13} - w_{14} + 0.01\left(4 + 3.04175 + \frac{w_{13}(w_{14} - w_{12})}{1.6}\right) = 0$$

$$-w_{13} + 2w_{14} - w_{15} + 0.01\left(4 + 3.456 + \frac{w_{14}(w_{15} - w_{13})}{1.6}\right) = 0$$

$$-w_{14} + 2w_{15} - w_{16} + 0.01\left(4 + 3.90625 + \frac{w_{15}(w_{16} - w_{14})}{1.6}\right) = 0$$

$$-w_{15} + 2w_{16} - w_{17} + 0.01\left(4 + 4.394 + \frac{w_{16}(w_{17} - w_{15})}{1.6}\right) = 0$$

$$-w_{16} + 2w_{17} - w_{18} + 0.01\left(4 + 4.92075 + \frac{w_{17}(w_{18} - w_{16})}{1.6}\right) = 0$$

$$-w_{17} + 2w_{18} - w_{19} + 0.01\left(4 + 5.488 + \frac{w_{18}(w_{19} - w_{17})}{1.6}\right) = 0$$

$$-w_{18} + 2w_{19} + 0.01\left(4 + 6.09725 + \frac{w_{19}(14.333333 - w_{18})}{1.6}\right) - 14.333333 = 0$$

Step 5:

Lastly we will define $J(\mathbf{w})$ using (5.5):

$$J(\mathbf{w}) = \begin{bmatrix} 2+0.01\left(\frac{w_2-17}{1.6}\right) & -1+0.05\left(\frac{1}{8}w_1\right) & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ -1-0.05\left(\frac{1}{8}w_2\right) & 2+0.01\left(\frac{w_3-w_1}{1.6}\right) & -1+0.05\left(\frac{1}{8}w_2\right) & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 & \cdots & \cdots & 0 \\ \vdots & 0 & \ddots & \ddots & \ddots & 0 & \cdots & 0 \\ \vdots & \vdots & 0 & \ddots & \ddots & \ddots & 0 & 0 \\ \vdots & \vdots & \vdots & 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \vdots & \vdots & \vdots & 0 & \ddots & \ddots & a_{i-1j} \\ 0 & 0 & 0 & 0 & 0 & 0 & a_{i,j-1} & a_{ij} \end{bmatrix}.$$

Now that we have defined each of the components, $\mathbf{w}^{(0)}$, $\mathbf{F}(\mathbf{w})$, and $J(\mathbf{w})$, we can input this data into our Matlab program (See Appendix).

The following data are the results from our output:

x_i	w_i	$\mathbf{w}^{(0)}$	$\mathbf{w}^{(1)}$	$\mathbf{w}^{(2)}$	$\mathbf{w}^{(3)}$	$\mathbf{w}^{(4)}$	$\mathbf{w}^{(5)}$
1.0	w_0	17.0000	17.0000	17.0000	17.0000	17.0000	17.0000
1.1	w_1	16.8667	16.7641	16.7606	16.7605	16.7605	16.7605
1.2	w_2	16.7333	16.5212	16.5135	16.5134	15.5134	16.5134
1.3	w_3	16.6000	16.2714	16.2859	16.2859	16.2589	16.2589
1.4	w_4	16.4667	16.0152	15.9974	15.9974	15.9974	15.9974
1.5	w_5	16.3333	15.7532	15.7299	15.7298	15.7298	15.7298
1.6	w_6	16.2000	15.4867	15.4578	15.4577	15.4577	15.4577
1.7	w_7	16.0667	15.2175	15.1831	15.1829	15.1829	15.1829
1.8	w_8	15.9333	14.9477	14.9085	14.9083	14.9083	14.9083
1.9	w_9	15.8000	14.6808	14.6377	14.6375	14.6375	14.6375
2.0	w_{10}	15.6667	14.4208	14.3752	14.3750	14.3750	14.3750
2.1	w_{11}	15.5333	14.1733	14.1269	14.1266	14.1266	14.1266
2.2	w_{12}	15.4000	13.9449	13.8997	13.8994	13.8993	13.8993
2.3	w_{13}	15.2667	13.7443	13.7022	13.7018	13.7018	13.7018
2.4	w_{14}	15.1333	13.5820	13.5448	13.5443	13.5443	13.5443
2.5	w_{15}	15.0000	13.4710	13.4397	13.4392	13.4391	13.4391
2.6	w_{16}	14.8667	13.4271	13.4017	13.4010	13.4010	13.4010
2.7	w_{17}	14.7333	13.4694	13.4483	13.4475	13.4475	13.4475
2.8	w_{18}	13.6209	13.6008	13.5999	13.5999	13.5999	13.5999
2.9	w_{19}	13.9089	13.8854	13.8844	13.8843	13.8843	13.8843
3.0	w_{20}	14.3333	14.3333	14.3333	14.3333	14.3333	14.3333

From the table above we can observe that $||\mathbf{w}^{(5)} - \mathbf{w}^{(4)}|| = 0$. This indicates that our sequence of iterates has converged. Thus, the solution to our boundary value problem of the nonlinear ordinary differential equation is

$$\begin{aligned}\overline{\mathbf{w}} = (&17.0000, 16.7605, 16.5134, 16.2589, 15.9974,\\ &15.7298, 15.4577, 15.1829, 14.9083, 14.6375,\\ &14.3750, 14.1266, 13.8993, 13.7018, 13.5443,\\ &13.439113.401013.447513.599913.8843)^t.\end{aligned} \tag{6.1}$$

The significance of this answer, is that it gives the approximation to the solutions of $y(x_i)$, where $x_i = a + ih$ and $i = 0, 1, 2, ..., N + 1$. Each numerical value in (6.1) gives the corresponding approximation of $y(x_0), y(x_1), y(x_2), ..., y(x_{N+1})$.

CHAPTER 7

Conclusion

From this paper, it is safe to say that numerical methods are a vital strand of mathematics. They are a powerful tool in not only solving nonlinear algebraic equations with one variable, but also systems of nonlinear algebraic equations. Even equations or systems of equations that may look simplistic in form, may in fact need the use of numerical methods in order to be solved. Numerical methods are also influential in solving for boundary value problems of nonlinear ordinary differential equations. Solving for boundary vaulue problems of linear ordinary differential equations can be difficult enough. Thus, it would be nearly impossible to solve boundary value problems of nonlinear ordinary differential equations without implementing numerical methods. In this paper, we only examined three numerical methods, however, there are several other ones that we have yet to take a closer look at.

The main results of this paper can be highlighted in to two different areas: Convergence and the role of Newton's method. With regards to convergence, we can summarize that a numerical method with a higher rate of convergence may reach the solution of a system in less iterations in comparison to another method with a slower convergence. For example, Newton's method converges quadratically and Broyden's method only converges superlinerally. The implication of this would be that given the exact same nonlinear system of equations denoted by $\mathbf{F}$, Newton's method would arrive at the solution of $\mathbf{F}{=}\mathbf{0}$ in less iterations compared to Broyden's method.

The second key result from this paper, is the significance of Newton's method in numerical methods. In the case of both Broyden 's method and the Finite-Difference method, Newton's method is incorporated into each of their algorithms. Broyden's method had an almost identical algorithm as Newton's method, with the exception of the use of approximation matrix. The Finite-Difference method implemented Newton's method once the boundary value problem was converted into a nonlinear algebraic system. Not only was Newton's method a part of these methods, but also other various numerical methods that I had come across. This demonstrates the diversity that Newton's method possesses; it can be applied to many problems. This mean we can make a conjecture that Newton's method is a notable process in the area of numerical methods.

What I would like to do next in regards to numerical methods, is that I would like to explore more methods that are used to solve boundary-value problems of nonlinear ordinary differential equations. In this paper I briefly looked at one, but I would like to focus more of my attention on these types of iterative methods. For example, the Shooting Method is definitely one method I would like to take a closer look at.

After all the material examined in this paper, we can conclude that numerical methods are a key component in the area of nonlinear mathematics.

Appendix

The following are the Matlab functions that were used solve the boundary value problem in Chapter 6.

File Name: Newton_sys.m

```
function w = Newton _sys(F, JF, w0, tol, max _it)
% Solve the nonliner system F(w)=0 using Newton's Method
% vectors w and w0 are row vectors (for display purposes)
% function F returns a column vector , [fl(w), ..fn(w)]'
% stop if norm of change in solution vector is less than tol
% solve JF(w) y = - F(w) using Matlab's "backlash operator"
% v = - feval(JF, wold)  feval(F, wold);
% the next approximate solution is w _new = wold + v';

    F='Newton _sys _F';
JF='Newton _sys _JF';
w0=[16.86666667, 16.73333333, 16.6, 16.46666667, 16.33333333, 16.2,
16.06666667, 15.9333333, 15.8, 15.66666667, 15.53333333, 15.4,
15.26666667, 15.13333333, 15, 14.86666667, 14.733333333, 14.6, 14.46666667];
tol=0.00001;
max _it=5000;

    w _old = w0;
disp([0 w _old]);
iter = 1;
while (iter ≤= max _it)
v = - feval(JF, w _old)  feval(F, w _old);
w _new = w _old + v';
dif = norm(w _new - w _old);
disp([iter w _new dif]);
if dif ≤ = tol
w = w _new;
disp('Newton method has converged')
```

```
return;
else
w _old = w _new;
end
iter = iter + 1;
end
disp('Newton method did not converge')
w = w _new;
```

File Name: Newton_sys_F.m

```
function y = Newton_sys_F(w)
% test function used for Newton method for a system
y = [ ( 2*w(1) - w(2) + 0.01*(4+0.33275+(w(1)*(w(2)-17)/1.6)) - 17 )
( -w(1) + 2*w(2) - w(3) + 0.01*(4+0.432+(w(2)*(w(3)-w(1))/1.6)) )
( -w(2) + 2*w(3) - w(4) + 0.01*(4+0.54925+(w(3)*(w(4)-w(2))/1.6)) )
( -w(3) + 2*w(4) - w(5) + 0.01*(4+0.686+(w(4)*(w(5)-w(3))/1.6)) )
( -w(4) + 2*w(5) - w(6) + 0.01*(4+0.84375+(w(5)*(w(6)-w(4))/1.6)) )
( - w(5) + 2*w(6) - w(7) + 0.01*(4+1.024+(w(6)*(w(7)-w(5))/1.6)) )
( -w(6) + 2*w(7) - w(8) + 0.01*(4+1.22825+(w(7)*(w(8)-w(6))/1.6)) )
( -w(7) + 2*w(8) - w(9) + 0.01*(4+1.458+(w(8)*(w(9)-w(7))/1.6)) )
( -w(8) + 2*w(9) - w(10) + 0.01*(4+1.71475+(w(9)*(w(10)-w(8))/1.6) ))
( -w(9) + 2*w(10) - w(11) + 0.01*(4+2+(w(10)*(w(11)-w(9))/1.6) ))
( -w(10) + 2*w(11) - w(12) + 0.01*(4+2.31525+(w(11)*(w(12)-w(10))/1.6)) )
( -w(11) + 2*w(12) - w(13) + 0.01*(4+2.662+(w(12)*(w(13)-w(11))/1.6)) )
( -w(12) + 2*w(13) - w(14) + 0.01*(4+3.04175+(w(13)*(w(14)-w(12))/1.6)) )
( -w(13) + 2*w(14) - w(15) + 0.01*(4+3.456+(w(14)*(w(15)-w(13))/1.6)) )
( -w(14) + 2*w(15) - w(16) + 0.01*(4+3.90625+(w(15)*(w(16)-w(14))/1.6)) )
( -w(15) + 2*w(16) - w(17) + 0.01*(4+4.394+(w(16)*(w(17)-w(15))/1.6)) )
( -w(16) + 2*w(17) - w(18) + 0.01*(4+4.92075+(w(17)*(w(18)-w(16))/1.6)) )
( -w(17) + 2*w(18) - w(19) + 0.01*(4+5.488+(w(18)*(w(19)-w(17))/1.6)) )
( -w(18) + 2*w(19) + 0.01*(4+6.09725+(w(19)*(14.333333-w(18))/1.6)) - 14.333333 )];
```

File Name: Newton_sys _JF.m

```
function y = Newton_sys _JF(w)
% test function used for Newton method for a system
% find JF and write it down y= the matrix JF
y =[ 2+0.01*((w(2)-17)/1.6) -1+0.05*(0.125*w(1)) 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; -1-
0.05*(0.125*w(2)) 2+0.01*((w(3)-w(1))/1.6) -1+0.05*(0.125*w(2)) 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0; 0 -1-0.05*(0.125*w(3)) 2+0.01*((w(4)-w(2))/1.6) -1+0.05*(0.125*w(3)) 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0; 0 0 -1-0.05*(0.125*w(4)) 2+0.01*((w(5)-w(3))/1.6) -1+0.05*(0.125*w(4))
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 0 0 0 -1-0.05*(0.125*w(5)) 2+0.01*((w(6)-w(4))/1.6) -
1+0.05*(0.125*w(5)) 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 0 0 0 0 -1-0.05*(0.125*w(6)) 2+0.01*((w(7)-
w(5))/1.6) -1+0.05*(0.125*w(6)) 0 0 0 0 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 -1-0.05*(0.125*w(7))
2+0.01*((w(8)-w(6))/1.6) -1+0.05*(0.125*w(7)) 0 0 0 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 -
1-0.05*(0.125*w(8)) 2+0.01*((w(9)-w(7))/1.6) -1+0.05*(0.125*w(8)) 0 0 0 0 0 0 0 0 0 0
0; 0 0 0 0 0 0 0 -1-0.05*(0.125*w(9)) 2+0.01*((w(10)-w(8))/1.6) -1+0.05*(0.125*w(9))
0 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 -1-0.05*(0.125*w(10)) 2+0.01*((w(11)-w(9))/1.6) -
1+0.05*(0.125*w(10)) 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 0 -1-0.05*(0.125*w(11)) 2+0.01*((w(12)-
w(10))/1.6) -1+0.05*(0.125*w(11)) 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 0 0 -1-0.05*(0.125*w(12))
2+0.01*((w(13)-w(11))/1.6) -1+0.05*(0.125*w(12)) 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 0 0 0 -1-
0.05*(0.125*w(13)) 2+0.01*((w(14)-w(12))/1.6) -1+0.05*(0.125*w(13)) 0 0 0 0 0; 0 0 0 0
0 0 0 0 0 0 0 0 -1-0.05*(0.125*w(14)) 2+0.01*((w(15)-w(13))/1.6) -1+0.05*(0.125*w(14))
0 0 0 0; 0 0 0 0 0 0 0 0 0 0 0 0 0 -1-0.05*(0.125*w(15)) 2+0.01*((w(16)-w(14))/1.6) -
1+0.05*(0.125*w(15)) 0 0 0; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1-0.05*(0.125*w(16)) 2+0.01*((w(17)-
w(15))/1.6) -1+0.05*(0.125*w(16)) 0 0; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1-0.05*(0.125*w(17))
2+0.01*((w(18)-w(16))/1.6) -1+0.05*(0.125*w(17)) 0; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
-1-0.05*(0.125*w(18)) 2+0.01*((w(19)-w(17))/1.6) -1+0.05*(0.125*w(18)); 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 -1-0.05*(0.125*w(19)) 2+0.01*(14.333333-w(18)/1.6 ) ] ;
```

The following are the actual Matlab results that were outputted from the data:

```
Columns 1 through 11
0 16.8667 16.7333 16.6000 16.4667 16.3333 16.2000 16.0667 15.9333 15.8000 15.6667
Columns 12 through 20
15.5333 15.4000 15.2667 15.1333 15.0000 14.8667 14.7333 14.6000 14.4667
Columns 1 through 11
1.0000 16.7641 16.5212 16.2714 16.0152 15.7532 15.4867 15.2175 14.9477 14.6808 14.4208
Columns 12 through 21
14.1733 13.9449 13.7443 13.5820 13.4710 13.4271 13.4694 13.6209 13.9089 4.6561
Columns 1 through 11
2.0000 16.7606 16.5135 16.2589 15.9974 15.7299 15.4578 15.1831 14.9085 14.6377 14.3752
Columns 12 through 21
14.1269 13.8997 13.7022 13.5448 13.4397 13.4017 13.4483 13.6008 13.8854 0.1377
Columns 1 through 11
3.0000 16.7605 16.5134 16.2589 15.9974 15.7298 15.4577 15.1829 14.9083 14.6375 14.3750
Columns 12 through 21
14.1266 13.8994 13.7018 13.5443 13.4392 13.4010 13.4475 13.5999 13.8844 0.0020
Columns 1 through 11
4.0000 16.7605 16.5134 16.2589 15.9974 15.7298 15.4577 15.1829 14.9083 14.6375 14.3750
Columns 12 through 21
14.1266 13.8993 13.7018 13.5443 13.4391 13.4010 13.4475 13.5999 13.8843 0.0001
```

```
Columns 1 through 11
5.0000 16.7605 16.5134 16.2589 15.9974 15.7298 15.4577 15.1829 14.9083 14.6375 14.3750
Columns 12 through 21
14.1266 13.8993 13.7018 13.5443 13.4391 13.4010 13.4475 13.5999 13.8843 0.0000
Newton method has converged
ans =
Columns 1 through 11
16.7605 16.5134 16.2589 15.9974 15.7298 15.4577 15.1829 14.9083 14.6375 14.3750 14.1266
Columns 12 through 19
13.8993 13.7018 13.5443 13.4391 13.4010 13.4475 13.5999 13.8843
```

Bibliography

[1] Atkinson, K.E. (1978). An Introduction to Numerical Analysis. *Nonlinear Systems of Equations* (pp. 88-95). Canada: John Wiley & Sons.

[2] Press, W.H., Flannery, B.P., Teukolsky, S.A., Vetterling, W.T. (1988). Numerical Recipes in C. *Newton-Raphson Method for Nonlinear Systems of Equations.* (pp.286-289). New York: Cambridge University Press.

[3] Burden, R.L., Faires, J.D (2005). Numerical Analysis. *Numerical Solutions of Nonlinear Systems of Equations*, (pp. 597-640). Belmount: Thomson Brooks/Cole.

[4] Burden, R.L., Faires, J.D (2005). Numerical Analysis. *Boundary-Value Problems for Ordinary Differential Equations*, (pp. 641-685). Belmount: Thomson Brooks/Cole

[5] Burden, R.L., Faires, J.D (2005). Numerical Analysis. *Special Types of Matrices*, (pp. 389-413). Belmount: Thomson Brooks/Cole

3, 7, 11, 15, 17
18, 19, 20