7-21-2010

A Comparison of Some Numerical Methods for Solving Volterra Integral Equations.

Scotty Glen Houston

To the University Council:

The Thesis Committee for Scotty Glen Houston certifies that this is the final approved version of the following electronic thesis: "A Comparison of Some Numerical Methods for Solving Volterra Integral Equations."

_____

Alistair Windsor, Ph.D.
Major Professor

We have read this thesis and recommend
its acceptance:

_____

John R. Haddock, Ph.D.

_____

Maria F. Botelho, Ph.D.

Accepted for the Graduate Council:

_____

Karen D. Weddle-West, Ph.D.
Vice Provost for Graduate Programs

A COMPARISON OF SOME NUMERICAL METHODS FOR SOLVING VOLTERRA INTEGRAL EQUATIONS.

A Thesis

Presented for the

Master of Science

Degree

The University of Memphis

Scotty Glen Houston

August, 2010

Abstract

Houston, Scotty Glen. M.S., The University of Memphis, August 2010.
A Comparison of Some Numerical Methods for Solving Volterra Integral Equations.
Major Professor: Alistair Windsor.

We implement several methods for solving Volterra integral equations based on standard techniques for solving ordinary differential equations. In particular

- we implement a version of the modified Euler method,
- we implement the standard fourth order Runge-Kutta method.

We compare these with a novel collocation method using Bernstein polynomials.

We also look at Volterra integral equations with weakly singular kernels. Here the standard methods developed above do not apply though the collocation method can still be used. We develop two "product integration" methods, derived from numerical integration methods that can be applied to these weakly singular Volterra integral equations and compare them to the collocation method.

Contents

List of Tables

List of Figures

1 Introduction

When exact solutions of Volterra integral equations cannot be found analytically, we use numerical methods to find approximate solutions. We compare a novel collocation method that uses Bernstein polynomials as a basis with more traditional methods such as the trapezoidal rule, the Runge-Kutta method, and product integration schemes for solving weakly singular problems.

Collocation methods are very fast in their running time, however, it is difficult to prove convergence results and to obtain error bounds.

We mimic the original paper by Mandal and Bhattacharrya [MB07], and we compare the various numerical methods on a variety of test problems.

2 Bernstein Polynomial Preliminaries

We will begin with a description of Bernstein polynomials and of their properties.

There are two possible approaches to Bernstein polynomials. One may work with the fixed interval $[0, 1]$ or with a general interval $[a, b]$. To simplify the notation we treat only the interval $[0, 1]$. Any interval $[a, b]$ may be mapped on to the interval $[0, 1]$ by an affine map.

We define the degree n Bernstein polynomials P_k^n by

$$P_k^n(x) = \binom{n}{k} x^k (1-x)^{n-k}$$

for $k = 0, 1, \ldots, n$. Bernstein polynomials take only non-negative values on $[0, 1]$.

Below we show the Bernstein polynomials $P_0^5, P_1^5, P_2^5, P_3^5, P_4^5, P_5^5$.

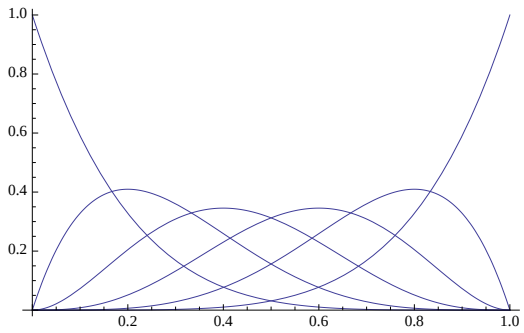

Figure 2.1: The 6 Bernstein Polynomials of Degree 5.

It appears from this that the Bernstein polynomials achieve their maximums at $0, \frac{1}{5}, \frac{2}{5}, \frac{3}{5}, \frac{4}{5}, 1$. This is the content of the next theorem.

Theorem 2.1. *The Bernstein polynomial P_n^k achieves its maximum at $\frac{k}{n}$.*

Proof. To find the maximum we perform the usual process of differentiating and setting equal to zero. Differentiating P_n^k we obtain

$$\begin{aligned}
\frac{d}{dx} P_k^n(x) &= \binom{k}{n} \big(k x^{k-1} (1-x)^{n-k} - (n-k) x^k (1-x)^{n-k-1} \big) \\
&= \binom{k}{n} x^{k-1} (1-x)^{n-k-1} \big(k(1-x) - (n-k)x \big).
\end{aligned}$$

This has three roots; 0, 1, and the solution of

$$k(1-x)-(n-k)x=0.$$

We solve this to find the interior critical point.

$$\begin{aligned} k-kx-nx+kx &= 0 \\ k-nx &= 0 \\ x &= \frac{k}{n}. \end{aligned}$$

Since $P_n^k(0) = P_n^k(1) = 0$ and $P_n^k(x) \geq 0$ the interior critical point must be a local maximum. □

Theorem 2.1 motivates the following definition: Given a continuous function f on $[0,1]$, we define a polynomial $B_n f$ by

$$\big(B_n f\big)(x) = \sum_{k=0}^{n} f\Big(\frac{k}{n}\Big) P_k^n(x) = \sum_{k=0}^{n} f\Big(\frac{k}{n}\Big) \binom{n}{k} x^k (1-x)^{n-k}.$$

This is a linear combination of the polynomials P_k^n. So $B_n f$ is a polynomial of at most degree n.

Theorem 2.2. *The following properties of the Bernstein polynomials need to be proved. Suppose that the map B_n is linear and monotone. Then for all $f, g \in C[0,1]$ and $\alpha \in \mathbb{R}$, the following are true:*

1. $B_n 1 = 1$.
2. $B_n(f+g) = B_n f + B_n g$.
3. $B_n(\alpha f) = \alpha B_n f$.
4. *if* $f \geq 0$ *then* $B_n f \geq 0$.
5. *if* $f \geq g$ *then* $B_n f \geq B_n g$.
6. *if* $|f| \leq g$ *then* $|B_n f| \leq B_n g$.

Proof. We will address each property in order.

1. From the definition of $B_n 1$

$$B_n 1 = \sum_{k=0}^{n} 1 \binom{n}{k} x^k (1-x)^{n-k} = \sum_{k=0}^{n} \binom{n}{k} x^k (1-x)^{n-k}.$$

Recall the Binomial Formula,

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k}.$$

Now, let $y = 1 - x$ in the Binomial formula Then,

$$(1)^n = \sum_{k=0}^{n} \binom{n}{k} x^k (1-x)^{n-k}.$$

So,

$$1 = \sum_{k=0}^{n} \binom{n}{k} x^k (1-x)^{n-k} = B_n 1.$$

2. Let f,g $\in C[0,1]$. Then, by the properties of the sum,

$$\begin{aligned}
B_n(f+g) &= \sum_{k=0}^{n} (f+g)\left(\frac{k}{n}\right) \binom{n}{k} x^k (1-x)^{n-k} \\
&= \sum_{k=0}^{n} \left(f\left(\frac{k}{n}\right) + g\left(\frac{k}{n}\right) \right) \binom{n}{k} x^k (1-x)^{n-k} \\
&= \sum_{k=0}^{n} \left[f\left(\frac{k}{n}\right) \binom{n}{k} x^k (1-x)^{n-k} + g\left(\frac{k}{n}\right) \binom{n}{k} x^k (1-x)^{n-k} \right] \\
&= \sum_{k=o}^{n} f\left(\frac{k}{n}\right) \binom{n}{k} x^k (1-x)^{n-k} + \sum_{k=0}^{n} g\left(\frac{k}{n}\right) \binom{n}{k} x^k (1-x)^{n-k} \\
&= B_n f + B_n g
\end{aligned}$$

as required.

3. Let f,g $\in C[0,1]$, and let $\alpha \in \mathbb{R}$. Then,

$$\begin{aligned}
B_n(\alpha f) &= \sum_{k=0}^{n} \left(\alpha f\right)\left(\frac{k}{n}\right) \binom{n}{k} x^k (1-x)^{n-k} \\
&= \sum_{k=0}^{n} \alpha f\left(\frac{k}{n}\right) \binom{n}{k} x^k (1-x)^{n-k} \\
&= \alpha \sum_{k=0}^{n} f\left(\frac{k}{n}\right) \binom{n}{k} x^k (1-x)^{n-k} \\
&= \alpha\, B_n f.
\end{aligned}$$

Thus,

$$B_n(\alpha f) = \alpha\, B_n f$$

as required.

4. Recall the definition of $B_n f$,

$$(B_n f)(x) = \sum_{k=0}^{n} f\Big(\frac{k}{n}\Big) P_k^n(x),$$

where P_k^n is a Bernstein polynomial. Since the Bernstein polynomials are all nonnegative on [0,1], and since $f \geq 0$ on [0,1], we have for each k that $f\big(\frac{k}{n}\big) P_k^n(x) \geq 0$. Then,

$$\sum_{k=0}^{n} f\Big(\frac{k}{n}\Big) P_k^n(x) \geq 0.$$

Thus, $B_n f \geq 0$.

5. Since $f \geq g$ we have $f - g \geq 0$. By Property (4) we must have

$$B_n(f - g) \geq 0.$$

By Properties (2) and (3) we have

$$B_n(f - g) = B_n f - B_n g.$$

Thus,

$$B_n f - B_n g \geq 0$$

so,

$$B_n f \geq B_n g$$

as we were to show.

6. Notice that if $|f| \leq g$, then $-g \leq f \leq g$. So, by Property (5) we have $-B_n g \leq B_n f \leq B_n g$. Therefore, $|B_n f| \leq B_n g$.

□

3 Bernstein's Proof of Weirstraß' Theorem

One of the principle reasons for defining Bernstein polynomials is to give a nice proof of the Weierstraß' Theorem. The following follows the presentation given in [DD01].

Theorem 3.1 (Weierstraß' Theorem)**.** *Let f be any continuous real-valued function on [a,b]. Then there is a sequence of polynomials, p_n, that converges to f uniformly on [a,b].*

In actuality what we will prove is that for a continuous function $f : [0, 1] \to \mathbb{R}$ the sequence of polynomials $B_n f$ converges uniformly to f on $[0, 1]$. The case for a general interval can be obtained from this by rescaling.

Proof. Fix f as a continuous function on [0,1]. What is needed is to prove that for every $\epsilon > 0$, there exists an $N > 0$, such that $|f(x) - (B_n f)(x)| < \epsilon$ for all $n \geq N$.

Since [0,1] is a compact interval, f is uniformly continuous over the interval. So for the given $\epsilon > 0$, there is some $\delta > 0$, such that $|f(x) - f(y)| \leq \frac{\epsilon}{2}$ if $|x - y| \leq \delta$ for $x, y \in [0, 1]$. Now, since f is bounded on [0,1], let $M = \|f\|_\infty = \sup_{x\in[0,1]} |f(x)|$. Fix any $a \in [0, 1]$. Now, notice that $|f(x) - f(a)| \leq \frac{\epsilon}{2} + \frac{2M}{\delta^2}(x - a)^2$. By linearity and the fact that $B_n 1 = 1, B_n(f - f(a))(x) = B_n f(x) - f(a)$. Using the fact that B_n is a positive mapping on [0,1],

$$\begin{aligned} |B_n f(x) - f(a)| &\leq B_n(\frac{\epsilon}{2} + \frac{2M}{\delta^2}(x - a)^2) \\ &= \frac{\epsilon}{2} + \frac{2M}{\delta^2}(x^2 + \frac{x - x^2}{n} - 2ax + a^2) \\ &= \frac{\epsilon}{2} + \frac{2M}{\delta^2}(x - a)^2 + \frac{2M}{\delta^2}\frac{x - x^2}{n}. \end{aligned}$$

Evaluate this at $x = a$ to get,

$$\begin{aligned} |B_n f(a) - f(a)| &\leq \frac{\epsilon}{2} + \frac{2M}{\delta^2}\frac{(a - a^2)}{n} \\ &\leq \frac{\epsilon}{2} + \frac{M}{2\delta^2 n}. \end{aligned}$$

As a calculus exercise shows, $a - a^2$ achieves a maximum at $\frac{1}{4}$ over $[0, 1]$. Now, notice that this estimate does not depend on the point a. So, $\|B_n f - f\|_\infty \leq \frac{\epsilon}{2} + \frac{M}{2\delta^2 n}$. Choose $N \geq \frac{M}{2\delta^2\epsilon}$ so that $\frac{M}{2\delta^2 N} < \frac{\epsilon}{2}$. Then, $\|B_n f - f\|_\infty \leq \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon$ for all $n \geq N$. $\square$

4 Bernstein polynomial Method for Solving Integral Equations

4.1 Volterra Integral Equations

We outline here the method proposed in [MB07] for solving integral equations using Bernstein polynomials.

The method may be applied to a variety of integral equations. We propose to study the two types of Volterra integral equations:

1. Volterra integral equations of the first kind

$$f(x) = \int_a^x K(x,t)\, u(t)\, dt$$

2. Volterra integral equations of the second kind

$$u(x) = f(x) + \int_a^x K(x,t)\, u(t)\, dt$$

These are Volterra integral equations because of the presence of the variable in the limits of integration.

We propose to study only linear Volterra equations though our numerical methods can be extended to non-linear equations without much difficulty.

4.2 Solving Volterra Integral Equations using Bernstein Polynomials

Bernstein polynomials have recently been used to solve certain classes of integral equations of both the first and the second kind. Recall that Bernstein polynomials are defined on [0,1] by

$$P_k^n(x) = \binom{n}{k} x^k (1-x)^{n-k}, \qquad k = 0, 1, \ldots, n.$$

These polynomials may be used to approximate any function on [0,1]. The following method uses approximation of the unknown function on the Bernstein polynomial basis for the solution of Volterra integral equations.

Consider the integral equation of the first kind given by

$$\int_0^x K(x,t)\, u(t)\, dt = f(x) \tag{4.1}$$

where $u(t)$ is the unknown function to be determined, $K(x,t)$, the kernel, is a continuous and square integrable function, and $f(x)$ is a known function satisfying $f(0) = 0$. To determine the approximate solution of (4.1) on $[0,1]$, $u(t)$ is approximated in the Bernstein polynomial basis on $[0,1]$ as

$$u(t) = \sum_{k=0}^{n} a_k\, P_k^n(t) \tag{4.2}$$

where $a_0, \ldots, a_n$ are constants to be determined. After substituting (4.2) in (4.1),

$$\sum_{k=0}^{n} a_k\, \alpha_k(x) = f(x) \tag{4.3}$$

where

$$\alpha_k(x) = \int_0^x K(x,t)\, P_k^n(t)\, dt.$$

We evaluate (4.3) at $n+1$ points

$$0 < x_0 < x_1 < \cdots < x_{n-1} < x_n < 1.$$

In this way we obtain a system of $n+1$ linear equations

$$\sum_{k=0}^{n} a_k\, \alpha_{k,j} = f_j, \qquad j = 0, 1, \ldots, n \tag{4.4}$$

where

$$\alpha_{k,j} = \alpha_k(x_j)$$

and

$$f_j = f(x_j).$$

The linear system in (4.4) can be easily solved by standard methods for the constants $a_0, \ldots, a_n$. These constants are then used in (4.2) to obtain our approximate solution $u(t)$.

Now consider the Volterra integral equation of the second kind given by

$$u(x) = f(x) + \int_0^x K(x,t)\, u(t)\, dt \tag{4.5}$$

where $K(x,t)$ is a kernel and $f(x)$ is a known function, then applying the method

from above, obtain

$$\sum_{k=0}^{n} a_k \beta_k(x) = f(x) \tag{4.6}$$

where

$$\beta_k(x) = P_k^n(x) - \int_0^x K(x,t)\, P_k^n(t)\, dt.$$

Choosing $n+1$ points

$$0 < x_0 < x_1 < \cdots < x_{n-1} < x_n < 1$$

as above, we obtain the following linear system

$$\sum_{k=0}^{n} a_k \beta_{k,j} = f_j, \quad j = 0, 1, 2, \ldots, n \tag{4.7}$$

where

$$\beta_{kj} = \beta_k(x_j)$$

and

$$f_j = f(x_j).$$

The linear system in (4.7) can be easily solved to obtain the unknown constants $a_0, \ldots, a_n$, that are then used to approximate the unknown function, $u(t)$.

5 Suite of Test Problems

We will compare the performance of the new algorithm with some standard numerical methods for approximating the solutions to Volterra integral equations. We will use a suite of four test problems.

5.1 Problem 1

Consider the Volterra integral equation of the second kind given by

$$u(x) = 1 - x - \frac{3}{2}x^2 + \frac{x^3}{2} + \int_0^x \frac{1+x}{1+t}u(t)dt.$$

The exact solution of this problem is given by $u(x) = 1 - x^2$. This is [MB07, Example 4].

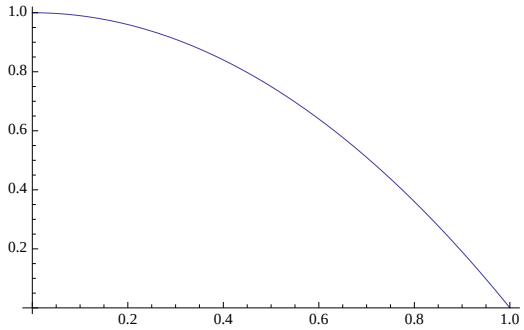

Figure 5.1: Exact Solution of Problem 1.

5.2 Problem 2

We now look at a Volterra equation from [Jer85, Problem 2, Page 93]. This problem does not have a polynomial solution.

$$u(x) = x + 2\int_0^x u(t)e^{x-t}dt. \tag{5.1}$$

To solve this problem it is convenient to recognize that the kernel $K(x,t) = e^{x-t}$ can

be thought of as a function of the difference $x - t$. Any kernel $K(x,t)$ that can be written in the form $K(x,t) = k(x-t)$ is called a difference kernel.

Difference kernels are particularly convenient from the perspective of Laplace transforms.

Lemma 5.1. *Let $k, u : [0,\infty) \to \mathbb{R}$ be of exponential type:*

$$\mathcal{L}\Big(\int_0^x k(x-t)u(t)\,dt\Big) = \mathcal{L}(k)\cdot\mathcal{L}(u).$$

Proof. The definition of the Laplace transform gives

$$\begin{aligned}\mathcal{L}\Big(\int_0^x k(x-t)u(t)\,dt\Big) &= \int_0^\infty \int_0^x k(x-t)u(t)\,dt\,e^{-sx}\,dx \\ &= \int_0^\infty \int_t^\infty k(x-t)u(t)e^{-sx}\,dx\,dt\end{aligned} \tag{5.2}$$

after the order of integration is switched. Now, let $y = x - t$.Then, (5.2) becomes

$$\begin{aligned}\int_0^\infty \int_0^\infty k(y)u(t)e^{-s(y+t)}\,dy\,dt &= \int_0^\infty u(t)e^{-st}\,dt \int_0^\infty k(y)e^{-sy}\,dy \\ &= \mathcal{L}(k)\cdot\mathcal{L}(u).\end{aligned}$$

□

Since

$$K(x,t) = e^{x-t}$$

is a difference kernel; using Lemma 5.1 with $k(x) = e^x$ we obtain

$$\begin{aligned}\mathcal{L}\Big(\int_0^x u(t)e^{x-t}\,dt\Big) &= \mathcal{L}\Big(\int_0^x k(x-t)u(t)\,dt\Big)\\ &= \mathcal{L}(k)\mathcal{L}(u)\\ &= \mathcal{L}(e^x)\mathcal{L}(u)\\ &= \frac{1}{(s-1)}\mathcal{L}(u).\end{aligned} \tag{5.3}$$

For notational convenience we will denote $\mathcal{L}(u)$ by U. Applying the Laplace transform to each term in (5.1) and using (5.3) we obtain

$$U(s) = \frac{1}{s^2} + \frac{2}{(s-1)}U(s).$$

We solve this for $U(s)$

$$\begin{aligned}U(s) - \frac{2}{s-1}U(s) &= \frac{1}{s^2}\\ U(s)\Big(1 - \frac{2}{s-1}\Big) &= \frac{1}{s^2}\\ U(s)\frac{s-3}{s-1} &= \frac{1}{s^2}\\ U(s) &= \frac{s-1}{s^2(s-3)}\end{aligned}$$

Now, use the inverse Laplace Transform on $U(s)$ to obtain

$$\begin{aligned}u(x) &= \mathcal{L}^{-1}(U(s))\\ &= \mathcal{L}^{-1}\Big(\frac{s-1}{s^2(s-3)}\Big)\end{aligned}$$

We will use the method of partial fractions. So,

$$\frac{s-1}{s^2(s-3)} = \frac{A}{s^2} + \frac{B}{s} + \frac{C}{s-3}.$$

Then, multiplying by $s^2(s-3)$ we get the equation

$$s - 1 = A(s-3) + Bs(s-3) + Cs^2$$

Evaluating this at $s = 0$ we get $-1 = -3A$ so $A = \frac{1}{3}$. Evaluating this at $s = 3$ we get $2 = 9C$ so $C = \frac{2}{9}$. Equating coefficients of s^2 we get $0 = B + C$ and consequently $B = -\frac{2}{9}$. Thus

$$\frac{s-1}{s^2(s-3)} = \frac{\frac{1}{3}}{s^2} - \frac{\frac{2}{9}}{s} + \frac{\frac{2}{9}}{s-3}.$$

Then,

$$\begin{aligned}\mathcal{L}^{-1}\Big(\frac{s-1}{s^2(s-3)}\Big) &= \frac{1}{3}\mathcal{L}^{-1}\Big(\frac{1}{s^2}\Big) - \frac{2}{9}\mathcal{L}^{-1}\Big(\frac{1}{s}\Big) + \frac{2}{9}\mathcal{L}^{-1}\Big(\frac{1}{s-3}\Big), \\ &= \frac{1}{3}x - \frac{2}{9} + \frac{2}{9}e^{3x}.\end{aligned}$$

Consequently,

$$u(x) = \frac{1}{3}x + \frac{2}{9}e^{3x} - \frac{2}{9}.$$

The solution of this problem is shown below.

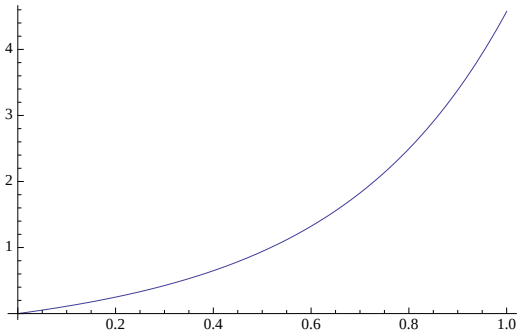

Figure 5.2: Exact Solution of Problem 2

5.3 Problem 3

Consider the following Volterra integral equation of the second kind given by,

$$u(x) = x^2 + \int_0^x \cos(x-t)\, u(t)\, dt.$$

The solution is unbounded and oscillatory. This problem can be solved using the method of Laplace transforms, however, we will follow a differential equations approach.

Notice that

$$\frac{du}{dx} = 2x + u(x) + \int_0^x -\sin(x-t)\, u(t)\, dt$$

and

$$\frac{d^2u}{dx^2} = 2 + \frac{du}{dx} + \int_0^x -\cos(x-t)\, u(t)\, dt.$$

Now,

$$\frac{d^2u}{dx^2} + u = 2 + x^2 + \frac{du}{dx},$$

or

$$\frac{d^2u}{dx^2} - \frac{du}{dx} + u = 2 + x^2.$$

This is a second order constant coefficient linear ordinary differential equation.

First we solve the homogeneous equation,

$$u_c^{''} - u_c^{'} + u_c = 0,$$

to find the complementary function u_c. The characteristic equation for this constant coefficient homogeneous linear system has roots $\frac{1}{2} \pm \frac{\sqrt{3}}{2}\imath$. So,

$$u_c = c_1 e^{(\frac{1}{2}x)} \cos\,(\frac{\sqrt{3}}{2}x) + c_2 e^{(\frac{1}{2}x)} \sin\,(\frac{\sqrt{3}}{2}x).$$

To find a particular solution we use the method of undetermined coefficients. This method says we may look for a solution of the form $u_p = Ax^2 + Bx + C$. Notice that $u_p^{'} = 2Ax + B$, and $u_p^{''} = 2A$.

Then, $2A - (2Ax + B) + Ax^2 + Bx + C = 2 + x^2$. After equating coefficients, it is found that $A = 1, B = 2$, and $C = 2$. Now, the particular solution is $u_p = x^2 + 2x + 2$. The solution is

$$u(x) = x^2 + 2x + 2 + c_1 e^{(\frac{1}{2}x)} \cos\,(\frac{\sqrt{3}}{2}x) + c_2 e^{(\frac{1}{2}x)} \sin\,(\frac{\sqrt{3}}{2}x).$$

To find the values of c_1 and c_2, notice from the given integral equation that

$u(0) = u'(0) = 0$. Then, $c_1 = -2$ and $c_2 = -\frac{2}{\sqrt{3}}$. The final solution is

$$u(x) = x^2 + 2x + 2 - 2e^{(\frac{1}{2}x)} \cos\,(\frac{\sqrt{3}}{2}x) - \frac{2}{\sqrt{3}} e^{(\frac{1}{2}x)} \sin\,(\frac{\sqrt{3}}{2}x).$$

We consider the problem on the interval $[0, 8]$ so that we can see at least one complete oscillation.

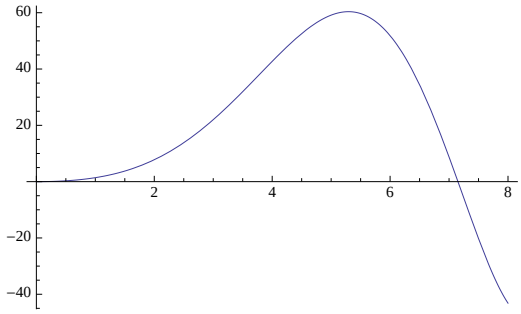

Figure 5.3: Exact solution to Problem 3 on the interval $[0, 8]$.

5.4 Problem 4

Consider the weakly singular Volterra integral equation of the second kind given by

$$u(x) = x^7(1 - \frac{4096}{6435}x^{1/2}) + \int_0^x \frac{u(t)}{\sqrt{x-t}}\,dt$$

for $0 < x < 1$ which has an exact solution given by $u(x) = x^7$. This is Example 7 from [MB07]. Note that this problem has a polynomial solution.

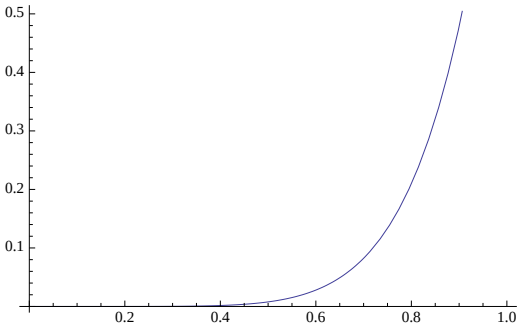

Figure 5.4: Exact Solution of Problem 4.

6 Classical Numerical Methods for Solving Volterra Equations

The previous problems were selected so that an exact solution could be found. When the problems are not so "nice," approximate methods are used where the integral equation is replaced or approximated by another closely related integral equation which can hopefully be handled by one of the usual methods. If this procedure is not reasonable for a given equation, we must resort to numerical methods which are also approximate methods and where the integral equation may be reduced to a set of N equations in $u(x_i)$, which are the samples of the approximate solution.

Volterra integral equations of the second kind can be thought of as a generalization of initial value problems for differential equations. As such many of the methods for solving initial value problems may be generalized to Volterra equations.

6.1 Trapezoidal Rule for Solving Integral Equations

We will now illustrate an example using one of the simplest and most familiar methods of numerical integration, the trapezoidal rule. Consider the Volterra integral equation of the second kind,

$$u(x) = f(x) + \int_a^x K(x,t)\,u(t)\,dt. \tag{6.1}$$

We fix a natural number n and interval of integration, $[a,b]$. The interval of integration is subdivided into n equal subintervals of equal width $h = \frac{b-a}{n}$. Define $x_i = a + ih$ so that

$$a = x_0 < x_1 < \cdots < x_{n-1} < x_n = b.$$

We compute the solution iteratively. Clearly $u(x_0) = f(x_0)$ since the integral does not appear. Imagine that values of u have been determined for $x_0, \ldots, x_{k-1}$. The x_k must be determined from the equation

$$u(x_k) = f(x_k) + \int_a^{x_k} K(x_k,t)\,u(t)\,dt.$$

We approximate the integral by using the trapezoidal rule. We use the same division in t as we used for x. Let $t_j = a + jh$. Notice that $t_j = x_j$. Using the trapezoidal rule to approximate the integral, we have

$$\begin{aligned}\int_0^{x_k} K(x_k,t)\,u(t)\,dt \approx \\ h\Big(&\frac{1}{2}K(x_k,t_0)\,u(t_0) + K(x_k,t_1)\,u(t_1) \\ &+ \cdots + K(x_k,t_{k-1})\,u(t_{k-1}) + \frac{1}{2}K(x_k,t_k)\,u(t_k)\Big),\end{aligned} \tag{6.2}$$

and the integral equation in (6.1) is then approximated by the sum

$$\begin{aligned} u(x_k) =& f(x_k) + h\left(\frac{1}{2}K(x_k, x_0)\, u(x_0) + K(x_k, x_1)\, u(x_1)\right. \\ &\left. + \cdots + K(x_k, x_{n-1})\, u(x_{n-1}) + \frac{1}{2}K(x_k, x_k)\, u(x_k)\right) \end{aligned} \tag{6.3}$$

where we have used $t_j = x_j$. Obviously, (6.3) is only an approximate solution of (6.1) since there is an error involved in replacing the integral in (6.1) by the terms of the trapezoidal rule.

This system of equations may be written in a more compact form as

$$\begin{aligned} u_0 &= f_0 \\ u_k &= f_k + h\left(\frac{1}{2}K_{k,0}\, u_0 + K_{k,1}\, u_1 + ... + K_{k,k-1}\, u_{k-1} + \frac{1}{2}K_{k,k}\, u_k\right) \end{aligned} \tag{6.4}$$

where

$$\begin{aligned} u_i &= u(x_i) \\ f_i &= f(x_i) \\ K_{i,j} &= K(x_i, x_j). \end{aligned}$$

Though we have used the notation of linear Volterra equations (all of our test problems will be linear Volterra equations) nothing up to this point of the discussion has used this linearity at all. At this point however we need to solve (6.4) for u_k. In our linear

case this is easy

$$u_k = \frac{f_k + h\left(\frac{1}{2}K_{k,0}\,u_0 + K_{k,1}\,u_1 + ... + K_{k,k-1}\,u_{k-1}\right)}{1 - \frac{h}{2}K_{k,k}}.$$

For nonlinear problems this step must be accomplished by some numerical solver. We will not pursue nonlinear problems further.

6.2 Runge-Kutta Method for Solving Integral Equations

We begin by discussing the Runge-Kutta method for solving initial value problems and then explain how to extend this to the case of a Volterra equation of the second kind.

Exactly as in the previous case we fix a natural number n and an interval $[a, b]$. The interval is subdivided into n equal subintervals of equal width $h = \frac{b-a}{n}$. Define $x_i = a + i\,h$. A p-stage Runge-Kutta method for the initial value problem

$$u'(x) = f\big(x, u(x)\big),$$
$$u(a) = u_0,$$

approximates the solution u at $x_0, \ldots, x_n$ by generating approximations at p intermediate points in $[x_i, x_{i+1}]$. The intermediate points are given by $x_i + \theta_r\,h$ were

$$0 = \theta_0 \leq \theta_1 \leq \cdots \leq \theta_{p-1} \leq 1.$$

The method gives

$$u_{i+1} = u_i + h \sum_{l=0}^{p-1} A_{p,l}\, k_l^i \tag{6.5}$$

where

$$k_r^i = \begin{cases} f(x_i, u_i), & r = 0 \\ f\big(x_i + \theta_r\, h, u_i + \ h \sum_{l=0}^{r-1}\ A_{r,l}\, k_l^i\big), & 1 \le r \le p-1. \end{cases}$$

and where the weights A_{rl} satisfy

$$\sum_{l=0}^{r-1} A_{rl} = \begin{cases} \theta_r, & \text{if } r = 1, 2, \ldots, p-1, \text{ or} \\ 1 & \text{if } r = p. \end{cases}$$

This can be written out in an alternative form that makes explicit the approximate solutions at intermediate point. We denote by $u_{i,r}$ the approximate solution at the point $x_{i,r} = x_i + \theta_r\, h$. It is given by

$$u_{i,r} = u_i + h \sum_{l=0}^{r-1} A_{r,l}\, f(x_{i,l}, u_{i,l}),$$

for $1 \le r \le p-1$. Finally u_{i+1} is given by

$$u_{i+1} = u_i + h \sum_{l=0}^{p-1} A_{p,l}\, f(x_{i,l}, u_{i,l}).$$

The parameters $A_{r,l}$, θ_l are chosen in practice to yield a final approximation of specified order; that is, with a local truncation error of $O(h^{q+1})$ for some chosen q which is the order of the method. This requirement yields a set of nonlinear equations

for the unknown parameters; for a given pair (p, q), no solutions, one solution, or a family of solutions may exist.

When $p = q = 4$, the classical fourth order Runge-Kutta method is obtained with the following choice of parameters: For the intermediate points we take:

$$\theta_0 = 0, \qquad \theta_1 = \frac{1}{2}, \qquad \theta_2 = \frac{1}{2}, \qquad \theta_3 = 1$$

and for weights:

$$\begin{aligned}
&A_{1,0} = \frac{1}{2} \\
&A_{2,0} = 0 && A_{2,1} = \frac{1}{2} \\
&A_{3,0} = 0 && A_{3,1} = 0 && A_{3,2} = 1 \\
&A_{4,0} = \frac{1}{6} && A_{4,1} = \frac{1}{3} && A_{4,2} = \frac{1}{3} && A_{4,3} = \frac{1}{6}
\end{aligned}$$

This is the particular Runge-Kutta method that we implement.

Now we show how to use this method to solve Volterra equations of the second kind

$$u(x) = f(x) + \int_a^x K\big(x, t, u(t)\big)\, dt. \tag{6.6}$$

For one of our points x_i this becomes

$$u(x_i) = f(x_i) + \int_a^{x_i} K\big(x_i, t, u(t)\big)\, dt. \tag{6.7}$$

We consider the Volterra equation of the second kind

$$y_i(x) = f(x_i) + \int_a^x K\big(x_i, t, y_i(t)\big)\, dt. \tag{6.8}$$

From (6.7) and (6.8) it is clear that $y_i(x_i) = u(x_i)$. However since x does not appear in the kernel of (6.8) we may differentiate (6.8) to obtain the following initial value problem.

$$\begin{aligned} y_i'(x) &= K\big(x_i, x, y_i(x)\big), \\ y_i(a) &= f(x_i) \end{aligned} \tag{6.9}$$

Now the solution y_i to the initial value problem (6.9) can be approximated at x_i using the Runge-Kutta method. We note that we have a separate and distinct initial value problem for each x_i.

6.3 Product Integration Method for Solving Integral Equations

Neither the trapezoidal rule nor the Runge-Kutta method work for a Volterra integral equation with a singular kernel since $K(x, x)$ is evaluated. We consider a method called product integration to solve the problem with the weakly singular kernel, Problem 4. The product integration idea is described in [DM85, Section 5.5].

Consider the numerical solution of

$$u(x) = f(x) + \int_a^x K(x, t)\, u(t)\, dt \tag{6.10}$$

when the kernel function, $K(x, t, u(t))$, is singular. Assume that the kernel function

or one of its lower order derivatives is badly behaved, and suppose that $K(x,t,u(t)) = p(s,t)m(x,t,u(t))$ where p and m are respectively singular and well-behaved functions of their arguments. Then the method of product integration may be used to solve problems of the form

$$u(x) = f(x) + \int_a^x p(s,t)\, m(x,t,u(t))\, dt \tag{6.11}$$

where $a \leq x \leq b$. Let the interval, [a,b] be divided into n equal subintervals. Let $h = \frac{b-a}{n}$ and define $x_i = a + i\,h$. Then the method proceeds by approximating the integral term in (6.11), quadrature rule of the form

$$\int_0^{x_i} p(x_i,t)\, m(x_i,t,u(t))\, dt \approx \sum_{j=0}^{i} w_{i,j}\, m(x_i,x_j,u(x_j)) \tag{6.12}$$

The weights are constructed by insisting that the rule in (6.12) be exact when $m(x_i,t,u(t))$ is a polynomial in t of degree $\leq$ some r. For each value of i, this requires the existence of the $r+1$ moments

$$\mu_{i,j} = \int_0^{x_i} t^j p(x_i,t) dt$$

for $j = 0, 1, \ldots, r$.

Thus we obtain

$$u(x_i) = f(x_i) + \sum_{j=0}^{i} w_{i,j}\, m\big(x_i,x_j,u(x_j)\big)$$

which may be solved for $u(x_i)$. For non-linear problems this will require a method such

as Newton-Raphson. In our problems we have $m\big(x,t,u(t)\big) = u(t)$ so the equation is

$$u(x_i) = f(x_i) + \sum_{j=0}^{i} w_{i,j}\, u(x_j)$$

which can be explicitly solved to give

$$u(x_i) = \frac{f(x_i) + \sum_{j=0}^{i-1} w_{i,j}\, u(x_j)}{1 - w_{i,i}}.$$

Product Integration Analogue of the Trapezoidal Rule

The product integration analogue of the trapezoidal rule proceeds by approximating the non-singular part of the integrand in (6.11) namely $m\big(x,t,u(t)\big)$ at $x = x_i$ by

$$m\big(x_i,t,u(t)\big) \approx \frac{t_{j+1}-t}{h}\, m\big(x_i,t_j,u(t_j)\big) + \frac{t-t_j}{h}\, m\big(x_i,t_{j+1},u(t_{j+1})\big). \tag{6.13}$$

it follows then that

$$\begin{aligned}
\int_0^{x_i} p(x_i,t)\, m\big(x_i,&t,u(t)\big)\, dt \\
&= \sum_{j=0}^{i-1} \int_{t_j}^{t_{j+1}} p(x_i,t) m\big(x_i,t,u(t)\big) dt \\
&\approx \sum_{j=0}^{i-1} \int_{t_j}^{t_{j+1}} p(x_i,t) \Big(\frac{t_{j+1}-t}{h} m\big(x_i,t_j,u(t_j)\big) \\
&\qquad\qquad + \frac{t-t_j}{h} m\big(x_i,t_{j+1},u(t_{j+1})\big)\Big)\, dt \\
&= \sum_{j=0}^{i} w_{i,j}\, m\big(x_i,t_j,u(t_j)\big).
\end{aligned}$$

where the weights are calculated from the expressions

$$
\begin{aligned}
w_{i,0} &= \frac{1}{h}\int_{t_0}^{t_1} p(x_i,t)(t_1-t)\,dt \\
w_{i,j} &= \frac{1}{h}\int_{t_j}^{t_{j+1}} p(x_i,t)(t_{j+1}-t)\,dt + \frac{1}{h}\int_{t_{j-1}}^{t_j} p(x_i,t)(t-t_{j-1})\,dt, \\
w_{i,i} &= \frac{1}{h}\int_{t_{i-1}}^{t_i} p(x_i,t)(t-t_{i-1})\,dt.
\end{aligned}
$$

where $j = 1, 2, \ldots, i-1$.

In order to speed up the product integration method we can compute the weight integrals analytically. The resulting weights may be used for any problem, linear or non-linear, where the weakly singular part of the kernel has the form $p(t,s) = \frac{1}{\sqrt{t-s}}$.

Letting $t_i = t_0 + ih$, we obtain the following weights,

$$
\begin{aligned}
w_{i,0} &= \frac{1}{h}\int_{t_0}^{t_0+h} p(t_0+ih,s)(t_0+h-s)\,ds \\
&= \frac{1}{h}\int_{t_0}^{t_0+h} \frac{1}{\sqrt{t_0+ih-s}}(t_0+h-s)\,ds.
\end{aligned}
$$

Using integration by parts on this, we obtain that

$$
w_{i,0} = 2\sqrt{hi} + \frac{4}{3}(i-1)\sqrt{h(i-1)} - \frac{4}{3}i\sqrt{hi}.
$$

Also,

$$
w_{i,j} = \frac{1}{h}\int_{t_j}^{t_{j+1}} \frac{1}{\sqrt{t_0+ih-s}}(t_{j+1}-s)\,ds + \frac{1}{h}\int_{t_{j-1}}^{t_j} \frac{1}{\sqrt{t_0+ih-s}}(s-t_{j-1})\,ds.
$$

Again, after using integration by parts, we obtain that

$$w_{i,j} = \frac{4}{3}\sqrt{h}\Big((i-j-1)^{\frac{3}{2}} - 2(i-j)^{\frac{3}{2}} + (i-j+1)^{\frac{3}{2}}\Big).$$

And,

$$w_{i,i} = \frac{1}{h}\int_{t_{i-1}}^{t_i} \frac{1}{\sqrt{t_0 + ih - s}}(s - t_{i-1})\, ds.$$

Finally, we obtain that,

$$w_{i,i} = \frac{4}{3}\sqrt{h}.$$

Higher Order Product Integration

Computing the weights analytically makes the product integration version of the trapezoidal rule run at a speed comparable to that of the regular trapezoidal rule. However, we found for non-singular problems that there was an advantage in using higher order methods. Though they take longer to run for a given number of steps the number of steps required to achieve a given error was reduced sufficiently that the higher order methods resulted in a net gain.

There is no obvious way to generalize the Runge-Kutta method to our weakly singular problem. However we may duplicate the work done above with the trapezoidal rule using a higher order numerical integration scheme. We choose to use the Simpson's rule numerical integration scheme.

This raises two issues that must be addressed:

1. The method is not self-starting. Using the Trapezoidal rule knowing $u(x_0)$ is sufficient to determine $u(x_1)$. However using Simpson's Rule $u(x_0)$ and $u(x_1)$ are required in order to determine $u(x_2)$. This raises the question of how to determine $u(x_1)$.

2. Simpson's rule requires that the number of subdivision for our integral be even. However we will have to approximate integrals where the number of subdivisions is naturally odd.

We address these issues in the following manners:

1. We use one step of the product integration Trapezoidal method to compute $u(x_1)$ from $u(x_0)$. We tried more complicated methods of starting including using a smaller step size and multiple steps to find $u(x_1)$ from $u(x_0)$. There was no discernible change in the final error bounds so we used the simplest solution.

2. To address the odd number of intervals it would be possible to simply place one Trapezoidal step in the integration scheme. However in order to preserve the higher order properties of the method we felt that we should use a four-point method that uses three intervals and has the same order as the Simpson's rule.

Weights for the i even case

To compute $u(x_i)$ for $i = 2m$ even we need to approximate the integral

$$\begin{aligned}
&\int_{x_0}^{x_i} p(x_i, s)\, u(s)\, ds \\
&\quad \approx \sum_{k=1}^{m} \int_{x_{2(k-1)}}^{x_{2k}} p(x_i, s)\Big(\frac{(x_{2k-1} - s)(x_{2k} - s)}{(x_{2k-1} - x_{2k-2})(x_{2k} - x_{2k-2})}\, u(x_{2k-2}) \\
&\qquad + \frac{(s - x_{2k-2})(x_{2k} - s)}{(x_{2k-1} - x_{2k-2})(x_{2k} - x_{2k-1})} x(x_{2k-1}) \\
&\qquad\quad + \frac{(s - x_{2k-2})(s - x_{2k-1})}{(x_{2k} - x_{2k-2})(x_{2k} - x_{2k-1})} u(x_{2k}) \Big) ds.
\end{aligned}$$

Distributing the integral we obtain,

$$\begin{aligned}
&= \sum_{k=1}^{m} \Big(\int_{x_{2(k-1)}}^{x_{2k}} p(x_i, s) \frac{(x_{2k-1} - s)(x_{2k} - s)}{(x_{2k-1} - x_{2k-2})(x_{2k} - x_{2k-2})} ds\, u(x_{2k-2}) \\
&\qquad + \int_{x_{2(k-1)}}^{x_{2k}} p(x_i, s) \frac{(s - x_{2k-2})(x_{2k} - s)}{(x_{2k-1} - x_{2k-2})(x_{2k} - x_{2k-1})} ds\, u(x_{2k-1}) \\
&\qquad\quad + \int_{x_{2(k-1)}}^{x_{2k}} p(x_i, s) \frac{(s - x_{2k-2})(s - x_{2k-1})}{(x_{2k} - x_{2k-2})(x_{2k} - x_{2k-1})} ds\, u(x_{2k}) \Big)
\end{aligned}$$

Now, pull out the first term of the first sum and the last term of the final sum to obtain the following:

$$\begin{aligned}
&\int_{x_0}^{x_i} p(x_i,s)\,u(s)\,ds \\
&\quad= \int_{x_0}^{x_2} p(x_i,s)\frac{(x_1-s)(x_2-s)}{(x_1-x_0)(x_2-x_0)}ds\,u(x_0) \\
&\qquad+\sum_{k=2}^{m}\int_{x_{2(k-1)}}^{x_{2k}} p(x_i,s)\frac{(x_{2k-1}-s)(x_{2k}-s)}{(x_{2k-1}-x_{2k-2})(x_{2k}-x_{2k-2})}ds\,u(x_{2k-2}) \\
&\qquad+\sum_{k=1}^{m}\int_{x_{2(k-1)}}^{x_{2k}} p(x_i,s)\frac{(s-x_{2k-2})(x_{2k}-s)}{(x_{2k-1}-x_{2k-2})(x_{2k}-x_{2k-1})}ds\,u(x_{2k-1}) \\
&\qquad+\sum_{i=1}^{m-1}\int_{x_{2(k-1)}}^{x_{2k}} p(x_i,s)\frac{(s-x_{2k-2})(s-x_{2k-1})}{(x_{2k}-x_{2k-2})(x_{2k}-x_{2k-1})}ds\,u(x_{2k}) \\
&\qquad+\int_{x_{i-2}}^{x_i} p(x_i,s)\frac{(s-x_{i-1})(s-x_{i-2})}{(x_i-x_{i-1})(x_i-x_{i-2})}ds\,u(x_i)
\end{aligned}$$

Reindexing the second term above to begin at $k=1$ and combining with the fourth term we obtain:

$$\begin{aligned}
&\quad= \int_{x_0}^{x_2} p(x_i,s)\frac{(x_1-s)(x_2-s)}{(x_1-x_0)(x_2-x_0)}ds\,u(x_0) \\
&\qquad+\sum_{k=1}^{m-1}\Big(\int_{x_{2k}}^{x_{2(k+1)}} p(x_i,s)\frac{(x_{2k+1}-s)(x_{2(k+1)}-s)}{(x_{2k+1}-x_{2k})(x_{2(k+1)-x_{2k}})}ds \\
&\qquad\qquad+\int_{x_{2(k-1)}}^{x_{2k}} p(x_i,s)\frac{(s-x_{2k-2})(s-x_{2k-1})}{(x_{2k}-x_{2k-2})(x_{2k}-x_{2k-1})}ds\Big)\;u(x_{2k}) \\
&\qquad+\sum_{k=1}^{m}\int_{x_{2(k-1)}}^{x_{2k}} p(x_i,s)\frac{(s-x_{2k-2})(x_{2k}-s)}{(x_{2k-1}-x_{2k-2})(x_{2k}-x_{2k-1})}ds\;u(x_{2k-1}) \\
&\qquad\qquad+\int_{x_{i-2}}^{x_i} p(x_i,s)\frac{(s-x_{i-1})(s-x_{i-2})}{(x_i-x_{i-1})(x_i-x_{i-2})}ds\;u(x_i)
\end{aligned}$$

From this the following weights are obtained for the case $i = 2m$

$$
\begin{aligned}
w_{i,0} &= \int_{x_0}^{x_2} p(x_i, s) \frac{(x_1 - s)(x_2 - s)}{(x_1 - x_0)(x_2 - x_0)} ds \\
w_{i,j} &= \begin{cases} \displaystyle\int_{x_{2(k-1)}}^{x_{2k}} p(x_i, s) \frac{(s - x_{2k-2})(x_{2k} - s)}{(x_{2k-1} - x_{2k-2})(x_{2k} - x_{2k-1})} ds & j = 2k - 1 \\ \displaystyle\int_{x_{2k}}^{x_{2(k+1)}} p(x_i, s) \frac{(x_{2k+1} - s)(x_{2(k+1)} - s)}{(x_{2k+1} - x_{2k})(x_{2(k+1)} - x_{2k})} ds & j = 2k \\ \qquad + \displaystyle\int_{x_{2(k-1)}}^{x_{2k}} p(x_i, s) \frac{(s - x_{2k-2})(s - x_{2k-1})}{(x_{2k} - x_{2k-2})(x_{2k} - x_{2k-1})} ds & j \neq 0, j \neq i \end{cases} \\
w_{i,i} &= \int_{x_{i-2}}^{x_i} p(x_i, s) \frac{(s - x_{i-1})(s - x_{i-2})}{(x_i - x_{i-1})(x_i - x_{i-2})} ds
\end{aligned}
$$

Analytic Weights for the i even case

After performing integration by parts on each of the given weights by hand with $x_i = ih$ and $p(t, s) = \frac{1}{\sqrt{t-s}}$, the following are obtained for each of the weights for the case where n is even:

$$
\begin{aligned}
w_{i,0} &= 2\sqrt{ih} - \frac{2}{3}\sqrt{h}(i-2)^{\frac{3}{2}} - \frac{8}{15}\sqrt{h}(i-2)^{\frac{5}{2}} + \frac{8}{15}\sqrt{h}i^{\frac{5}{2}} - 2\sqrt{h}(i)^{\frac{3}{2}} \\
w_{i.j} &= \begin{cases} \frac{8}{3}\sqrt{h}(i-2k)^{\frac{3}{2}} + \frac{8}{3}\sqrt{h}(i-2k+2)^{\frac{3}{2}} & \\ \quad + \frac{16}{15}\sqrt{h}(i-2k)^{\frac{5}{2}} - \frac{16}{15}\sqrt{h}(i-2k+2)^{\frac{5}{2}} & j = 2k-1 \\ -\frac{2}{3}\sqrt{h}(i-2k-2)^{\frac{3}{2}} - \frac{8}{15}\sqrt{h}(i-2k-2)^{\frac{5}{2}} - 4\sqrt{h}(i-2k)^{\frac{3}{2}} & j = 2k, \\ \quad - \frac{2}{3}\sqrt{h}(i-2k+2)^{\frac{3}{2}} + \frac{8}{15}\sqrt{h}(i-2k+2)^{\frac{5}{2}} & j \neq 0, j \neq i \end{cases} \\
w_{i,i} &= \frac{4}{5}\sqrt{2h}
\end{aligned}
$$

Weights for the i odd cases

To compute $u(x_i)$ for i odd we need to approximate the integral

$$\int_{x_0}^{x_i} p(x_i, s)\, u(s)\, ds.$$

Unlike when i is even we have to break the case of i odd down into three subcases:

1. $i = 1$ – we use a product integration Trapezoidal rule.
2. $i = 3$ – we use a four point product integration rule over the interval $[x_0, x_3]$. The weight $w_{3,3}$ arises from the fact that x_3 is the right hand endpoint of the four point integration interval.
3. $i \geq 5$ – we use a four point integration rule over the interval $[x_0, x_3]$ and Simpson's rule over the remaining even number of intervals. The weight $w_{3,3}$ arises from the fact that x_3 is the right hand endpoint of the four point integration rule interval and the left hand endpoint of the Simpson's rule interval $[x_3, x_5]$.

Case $i = 1$: When $i = 1$, we are reduced to a trapezoidal rule on $[x_0, x_1]$where

$$u(s) \approx \frac{x_1 - s}{x_1 - x_0} u(x_0) + \frac{s - x_0}{x_1 - x_0} u(x_1).$$

Then,

$$\int_{x_0}^{x_1} p(x_1, s)\, u(s)\, ds$$
$$= \int_{x_0}^{x_1} p(x_1, s) \frac{x_1 - s}{x_1 - x_0}\, ds\, u(x_0) + \int_{x_0}^{x_1} p(x_1, s) \frac{s - x_0}{x_1 - x_0}\, ds\, u(x_1).$$

From this, the following weights are obtained for the $i = 1$ case,

$$w_{1,0} = \int_{x_0}^{x_1} p(x_1, s) \frac{x_1 - s}{x_1 - x_0} ds$$
$$w_{1,1} = \int_{x_0}^{x_1} p(x_1, s) \frac{s - x_0}{x_1 - x_0} ds$$

Case $i = 3$: When $i = 3$ we estimate u on $[x_0, x_3]$ by the cubic polynomial

$$u(s) = \frac{(x_1 - s)(x_2 - s)(x_3 - s)}{(x_1 - x_0)(x_2 - x_0)(x_3 - x_0)} u(x_0)$$
$$+ \frac{(s - x_0)(x_2 - s)(x_3 - s)}{(x_1 - x_0)(x_2 - x_1)(x_3 - x_1)} u(x_1)$$
$$+ \frac{(s - x_0)(s - x_1)(x_3 - s)}{(x_2 - x_0)(x_2 - x_1)(x_3 - x_2)} u(x_2)$$
$$+ \frac{(s - x_0)(s - x_1)(s - x_2)}{(x_3 - x_0)(x_3 - x_1)(x_3 - x_2)} u(x_3)$$

Then,

$$\begin{aligned}\int_{x_0}^{x_3} p(x_3,s)\,u(s)\,ds &= \int_{x_0}^{x_3} p(x_3,s)\frac{(x_1-s)(x_2-s)(x_3-s)}{(x_1-x_0)(x_2-x_0)(x_3-x_0)}\,ds\,u(x_0)\\ &+ \int_{x_0}^{x_3} p(x_3,s)\frac{(s-x_0)(x_2-s)(x_3-s)}{(x_1-x_0)(x_2-x_1)(x_3-x_1)}\,ds\,u(x_1)\\ &+ \int_{x_0}^{x_3} p(x_3,s)\frac{(s-x_0)(s-x_1)(x_3-s)}{(x_2-x_0)(x_2-x_1)(x_3-x_2)}\,ds\,u(x_2)\\ &+ \int_{x_0}^{x_3} p(x_3,s)\frac{(s-x_0)(s-x_1)(s-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)}\,ds\,u(x_3)\end{aligned}$$

Consequently we obtain the following weights:

$$\begin{aligned}w_{3,0} &= \int_{x_0}^{x_3} p(x_3,s)\frac{(x_1-s)(x_2-s)(x_3-s)}{(x_1-x_0)(x_2-x_0)(x_3-x_0)}\,ds\\ w_{3,1} &= \int_{x_0}^{x_3} p(x_3,s)\frac{(s-x_0)(x_2-s)(x_3-s)}{(x_1-x_0)(x_2-x_1)(x_3-x_1)}\,ds\\ w_{3,2} &= \int_{x_0}^{x_3} p(x_3,s)\frac{(s-x_0)(s-x_1)(x_3-s)}{(x_2-x_0)(x_2-x_1)(x_3-x_2)}\,ds\\ w_{3,3} &= \int_{x_0}^{x_3} p(x_3,s)\frac{(s-x_0)(s-x_1)(s-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)}\,ds\end{aligned}$$

Case i odd and $i \geq 5$: We approximate $u(s)$ on $[x_0, x_3]$ using the cubic polynomial expressed above, and we estimate $u(s)$ on $[x_{2k+1}, x_{2k+3}]$ for $k \geq 1$ using the Simpson's rule. Thus on $[x_{2k+1}, x_{2k+3}]$ we approximate

$$\begin{aligned}u(s) \approx\ & \frac{(x_{2k+3}-s)(x_{2k+2}-s)}{(x_{2k+3}-x_{2k+1})(x_{2k+2}-x_{2k+1})}\,u(x_{2k+1})\\ &+ \frac{(x_{2k+3}-s)(s-x_{2k+1})}{(x_{2k+3}-x_{2k+2})(x_{2k+2}-x_{2k+1})}\,u(x_{2k+2})\\ &+ \frac{(s-x_{2k+2})(s-x_{2k+1})}{(x_{2k+3}-x_{2k+2})(x_{2k+3}-x_{2k+1})}u(x_{2k+3}).\end{aligned}$$

Thus, if we write $i = 2m + 3$, we obtain

$$\int_{x_0}^{x_i} p(x_i, s) = \int_{x_0}^{x_3} p(x_i, s)\, u(s)\, ds + \sum_{k=1}^{m} \int_{x_{2k+1}}^{x_{2k+3}} p(x_i, s)\, u(s)\, ds.$$

Estimating as above we see that the weights $w_{i,0}$,$w_{i,1}$, and $w_{i,2}$ are virtually unchanged from the $i = 3$ case:

$$\begin{aligned}
w_{i,0} &= \int_{x_0}^{x_3} p(x_i, s) \frac{(x_1 - s)(x_2 - s)(x_3 - s)}{(x_1 - x_0)(x_2 - x_0)(x_3 - x_0)}\, ds \\
w_{i,1} &= \int_{x_0}^{x_3} p(x_i, s) \frac{(s - x_0)(x_2 - s)(x_3 - s)}{(x_1 - x_0)(x_2 - x_1)(x_3 - x_1)}\, ds \\
w_{i,2} &= \int_{x_0}^{x_3} p(x_i, s) \frac{(s - x_0)(s - x_1)(x_3 - s)}{(x_2 - x_0)(x_2 - x_1)(x_3 - x_2)}\, ds
\end{aligned}$$

Continue on with the summation portion of the estimation.

$$\begin{aligned}
\sum_{k=1}^{m} \int_{x_{2k+1}}^{x_{2k+3}} p(x_i, s) \Big(& \frac{(x_{2k+3} - s)(x_{2k+2} - s)}{(x_{2k+3} - x_{2k+1})(x_{2k+2} - x_{2k+1})}\, u(x_{2k+1}) \\
& + \frac{(x_{2k+3} - s)(s - x_{2k+1})}{(x_{2k+3} - x_{2k+2})(x_{2k+2} - x_{2k+1})}\, u(x_{2k+2}) \\
& + \frac{(s - x_{2k+2})(s - x_{2k+1})}{(x_{2k+3} - x_{2k+2})(x_{2k+3} - x_{2k+1})}\, u(x_{2k+3}) \Big)\, ds.
\end{aligned}$$

This approximation then becomes

$$\begin{aligned}
\sum_{k=1}^{m} \Big(& \int_{x_{2k+1}}^{x_{2k+3}} p(x_i, s) \frac{(x_{2k+3} - s)(x_{2k+2} - s)}{(x_{2k+3} - x_{2k+1})(x_{2k+2} - x_{2k+1})}\, ds\; u(x_{2k+1}) \\
& + \int_{x_{2k+1}}^{x_{2k+3}} p(x_i, s) \frac{(x_{2k+3} - s)(s - x_{2k+1})}{(x_{2k+3} - x_{2k+2})(x_{2k+2} - x_{2k+1})}\, ds\; u(x_{2k+2}) \\
& + \int_{x_{2k+1}}^{x_{2k+3}} p(x_i, s) \frac{(s - x_{2k+2})(s - x_{2k+1})}{(x_{2k+3} - x_{2k+2})(x_{2k+3} - x_{2k+1})}\, ds\; u(x_{2k+3}) \Big).
\end{aligned}$$

Now, pull out the first term of the first sum and the last term of the final sum to obtain the following:

$$\begin{aligned}
&\int_{x_3}^{x_5} p(x_i,s)\frac{(x_5-s)(x_4-s)}{(x_5-x_3)(x_4-x_3)}ds\ u(x_3)\\
&\quad+\sum_{k=2}^{m}\int_{x_{2k+1}}^{x_{2k+3}} p(x_i,s)\frac{(x_{2k+3}-s)(x_{2k+2}-s)}{(x_{2k+3}-x_{2k+1})(x_{2k+2}-x_{2k+1})}ds\ u(x_{2k+1})\\
&\quad+\sum_{k=1}^{m}\int_{x_{2k+1}}^{x_{2k+3}} p(x_i,s)\frac{(x_{2k+3}-s)(s-x_{2k+1})}{(x_{2k+3}-x_{2k+2})(x_{2k+2}-x_{2k+1})}ds\ u(x_{2k+2})\\
&\quad+\sum_{k=1}^{m-1}\int_{x_{2k+1}}^{x_{2k+3}} p(x_i,s)\frac{(s-x_{2k+2})(s-x_{2k+1})}{(x_{2k+3}-x_{2k+2})(x_{2k+3}-x_{2k+1})}ds\ u(x_{2k+3})\\
&\qquad+\int_{x_{i-2}}^{x_i} p(x_i,s)\frac{(s-x_{i-1})(s-x_{i-2})}{(x_i-x_{i-1})(x_i-x_{i-2})}ds\ u(x_i).
\end{aligned}$$

Reindex the second term above to begin at $i=1$ and combine with the fourth term to obtain:

$$\begin{aligned}
&\int_{x_3}^{x_5} p(x_i,s)\frac{(x_5-s)(x_4-s)}{(x_5-x_3)(x_4-x_3)}ds\ u(x_3)+\\
&\quad+\sum_{k=1}^{m-1}\Big(\int_{x_{2k+3}}^{x_{2k+5}} p(x_i,s)\frac{(x_{2k+5}-s)(x_{2k+4}-s)}{(x_{2k+5}-x_{2k+3})(x_{2k+4}-x_{2k+3})}ds\\
&\qquad+\int_{x_{2k+1}}^{x_{2k+3}} p(x_i,s)\frac{(s-x_{2k+2})(s-x_{2k+1})}{(x_{2k+3}-x_{2k+2})(x_{2k+3}-x_{2k+1})}ds\Big)\ u(x_{2k+3})\\
&\quad+\sum_{k=1}^{m}\int_{x_{2k+1}}^{x_{2k+3}} p(x_i,s)\frac{(x_{2k+3}-s)(s-x_{2k+1})}{(x_{2k+3}-x_{2k+2})(x_{2k+2}-x_{2k+1})}ds\ u(x_{2k+2})+\\
&\quad+\int_{x_{i-2}}^{x_i} p(x_i,s)\frac{(s-x_{i-1})(s-x_{i-2})}{(x_i-x_{i-1})(x_i-x_{i-2})}ds\ u(x_i).
\end{aligned}$$

From this, the following weights are obtained for the case $i = 2m + 3$ for $m \geq 1$:

$$
\begin{aligned}
w_{i,3} &= \int_{x_0}^{x_3} p(x_i, s) \frac{(s - x_2)(s - x_1)(s - x_0)}{(x_3 - x_2)(x_3 - x_1)(x_3 - x_0)}\, ds \\
&\quad + \int_{x_3}^{x_5} p(x_i, s) \frac{(x_5 - s)(x_4 - s)}{(x_5 - x_3)(x_4 - x_3)} ds \\
w_{i,j} &= \begin{cases}
\displaystyle\int_{x_{2k+1}}^{x_{2k+3}} p(x_i, s) \frac{(x_{2k+3} - s)(s - x_{2k+1})}{(x_{2k+3} - x_{2k+2})(x_{2k+2} - x_{2k+1})}\, ds & j = 2k + 2 \\
\displaystyle\int_{x_{2k+3}}^{x_{2k+5}} p(x_i, s) \frac{(x_{2k+5} - s)(x_{2k+4} - s)}{(x_{2k+5} - x_{2k+3})(x_{2k+4} - x_{2k+3})} ds & j = 2k + 3 \\
\quad + \displaystyle\int_{x_{2k+1}}^{x_{2k+3}} p(x_i, s) \frac{(s - x_{2k+2})(s - x_{2k+1})}{(x_{2k+3} - x_{2k+2})(x_{2k+3} - x_{2k+1})} ds & j \neq 3, j \neq i
\end{cases} \\
w_{i,i} &= \int_{x_{i-2}}^{x_i} p(x_i, s) \frac{(s - x_{i-1})(s - x_{i-2})}{(x_i - x_{i-1})(x_i - x_{i-2})}\, ds.
\end{aligned}
$$

Analytic weights for the i odd case

As before we have $p(x, t) = \frac{1}{\sqrt{x-t}}$.

For the $i = 1$ case we obtain

$$
\begin{aligned}
w_{1,0} &= \frac{2}{3}\sqrt{h} \\
w_{1,1} &= \frac{4}{3}\sqrt{h}
\end{aligned}
$$

For the case $i \geq 3$ we obtain the following formulas:

$$
\begin{aligned}
w_{i,0} &= \sqrt{h}\Big(\frac{4}{9}(i-3)^{\frac{3}{2}} + \frac{8}{15}(i-3)^{\frac{5}{2}} + \frac{16}{105}(i-3)^{\frac{7}{2}} + 2\sqrt{i} \\
&\qquad - \frac{22}{9}i^{\frac{3}{2}} + \frac{16}{15}i^{\frac{5}{2}} - \frac{16}{105}i^{\frac{7}{2}}\Big) \\
w_{i,1} &= \sqrt{h}\Big(-2(i-3)^{\frac{3}{2}} - \frac{32}{15}(i-3)^{\frac{5}{2}} - \frac{16}{35}(i-3)^{\frac{7}{2}} + 4i^{\frac{3}{2}} - \frac{8}{3}i^{\frac{5}{2}} + \frac{16}{35}i^{\frac{7}{2}}\Big) \\
w_{i,2} &= \sqrt{h}\Big(4(i-3)^{\frac{3}{2}} + \frac{8}{3}(i-3)^{\frac{5}{2}} + \frac{16}{35}(i-3)^{\frac{7}{2}} - 2i^{\frac{3}{2}} + \frac{32}{15}i^{\frac{5}{2}} - \frac{16}{35}i^{\frac{7}{2}}\Big) \\
w_{i,3} &= \begin{cases}
\sqrt{h}\Big(-2(i-3)^{\frac{1}{2}} - \frac{22}{9}(i-3)^{\frac{3}{2}} - \frac{16}{15}(i-3)^{\frac{5}{2}} & \\
\qquad - \frac{16}{105}(i-3)^{\frac{7}{2}} + \frac{4}{9}i^{\frac{3}{2}} - \frac{8}{15}i^{\frac{5}{2}} + \frac{16}{105}i^{\frac{7}{2}}\Big)', & i = 3 \\
-\frac{40}{9}\sqrt{h}(i-3)^{\frac{3}{2}} - \frac{8}{15}\sqrt{h}(i-3)^{\frac{5}{2}} - \frac{16}{105}\sqrt{h}(i-3)^{\frac{7}{2}} + \frac{4}{9}\sqrt{h}(i)^{\frac{3}{2}} & \\
\qquad - \frac{8}{15}\sqrt{h}(i)^{\frac{5}{2}} + \frac{16}{105}\sqrt{h}(i)^{\frac{7}{2}} - \frac{2}{3}\sqrt{h}(i-5)^{\frac{3}{2}} - \frac{8}{15}\sqrt{h}(i-5)^{\frac{5}{2}} & i \geq 5
\end{cases} \\
w_{i,j} &= \begin{cases}
\frac{8}{3}\sqrt{h}(i-2k-3)^{\frac{3}{2}} + \frac{16}{15}\sqrt{h}(i-2k-3)^{\frac{5}{2}} & \\
\qquad + \frac{8}{3}\sqrt{h}(i-2k-1)^{\frac{3}{2}} - \frac{16}{15}\sqrt{h}(i-2k-1)^{\frac{5}{2}} & j = 2k+2 \\
-\frac{2}{3}\sqrt{h}(i-2k-5)^{\frac{3}{2}} - \frac{8}{15}\sqrt{h}(i-2k-5)^{\frac{5}{2}} - 4\sqrt{h}(i-2k-3)^{\frac{3}{2}} & j = 2k+3 \\
\qquad - \frac{2}{3}\sqrt{h}(i-2k-1)^{\frac{3}{2}} + \frac{8}{15}\sqrt{h}(i-2k-1)^{\frac{5}{2}} & j \neq 3, j \neq i
\end{cases} \\
w_{i,i} &= \frac{4}{5}\sqrt{2h}.
\end{aligned}
$$

7 Numerical Results

7.1 Tables for Problem 1

Table 7.1: Trapezoidal Rule Timing Table for Problem 1

Steps	Timing(secs)	Max. Error
10	0.000855	$3.505967446184705 \times 10^{-16}$
20	0.001786	$4.554761126667309 \times 10^{-16}$
40	0.005772	$5.551115123125783 \times 10^{-16}$
80	0.018147	$9.43689570931383 \times 10^{-16}$
160	0.066771	$7.771561172376096 \times 10^{-16}$
320	0.263254	$1.3322676295501878 \times 10^{-15}$
640	1.04509	$8.257283745649602 \times 10^{-16}$
1280	4.10863	$9.992007221626409 \times 10^{-16}$
2560	16.3114	$1.4432899320127035 \times 10^{-15}$

Working with infinite precision arithmetic we can confirm that the numerical method using the Trapezoidal rule does indeed produce an "exact" solution to the problem, that is the approximate solution is equal to the actual solution at the x values sampled.

Table 7.2: Error Reduction and Timing Increase Table for Problem 1 Using the Trapezoidal Rule

Beginning Steps	Ending Steps	Error Reduction Factor	Timing Increase Factor
10	20	0.769737	2.31746
20	40	0.820513	3.28253
40	80	0.588235	3.46183
80	160	1.21429	3.4065
160	320	0.583333	3.86082
320	640	1.61345	3.9388
640	1280	0.826389	3.95298
1280	2560	0.692308	3.93339

Since the error reported above is solely rounding error it is not expected to exhibit any pattern. It would appear that doubling the number of steps increases the running time fourfold. This is exactly what is expected for all of the "traditional" methods. If we divide the integration interval into n subintervals then we in fact perform n numerical integrations. The first takes only 1 step, with each successive integration takes 1 additional step, until the last integration which takes n steps. Thus the number of integration steps is actually

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2} = O(n^2).$$

Consequently we see that halving the step size, or equivalently doubling the number of subintervals, will result in the number of integration steps increasing by a factor of about four and, consequently the running time increasing by a factor of about four.

Table 7.3: Bernstein Method Timing Table for Problem 1

Order	Timing(secs)	Max. Error
1	0.009212	0.333929
2	0.017809	$1.7763568394002505 \times 10^{-15}$
3	0.031762	$1.685826127601827 \times 10^{-15}$
4	0.049423	$1.9165052215823697 \times 10^{-14}$
5	0.071519	$5.462412705096758 \times 10^{-15}$
6	0.097439	$3.266428033715291 \times 10^{-15}$
7	0.127807	$2.611489965036635 \times 10^{-15}$
8	0.162286	$2.7825166147512697 \times 10^{-15}$
9	0.201781	$3.234614473025316 \times 10^{-16}$
10	0.245167	$4.286681382243591 \times 10^{-15}$
11	0.293525	$1.2889193212718668 \times 10^{-14}$
12	0.345751	$4.223690468837967 \times 10^{-14}$
13	0.402289	$9.634082007031313 \times 10^{-14}$
14	0.4641	$1.0695938984814067 \times 10^{-13}$
15	0.529304	$1.0808184644531635 \times 10^{-11}$
16	0.599249	$5.3668775505309415 \times 10^{-11}$
17	0.674211	$8.231963667243426 \times 10^{-13}$
18	0.75341	$1.7066753643343196 \times 10^{-12}$
19	0.836884	$2.919290016655155 \times 10^{-11}$
20	0.924948	$3.792079438742732 \times 10^{-11}$

Were it not for the numerical integration, an exact solution should be found via the Bernstein collocation method using polynomials of degree 2 or higher since the solution is a quadratic polynomial. We see that the error is basically at the machine error until the order goes high enough that the matrix inversion becomes ill conditioned.

The choice of a problem that should be solved exactly by the collocation method as a "test" problem is questionable. Surprisingly the graph reported in [MB07] shows errors of size at least 10^{-11} though it achieves 0 error at the collocation points. This should be impossible.

Table 7.4: Runge-Kutta Timing Table for Problem 1

Order	Timing(secs)	Max. Error
10	0.008104	$3.4037473637393845 \times 10^{-6}$
20	0.024975	$2.1818785111982208 \times 10^{-7}$
40	0.094617	$1.3809348997639859 \times 10^{-8}$
80	0.367468	$8.6851259517573 \times 10^{-10}$
160	1.44324	$5.445288664418513 \times 10^{-11}$
320	5.75127	$3.4079405963893805 \times 10^{-12}$
640	22.6469	$2.1360690993788012 \times 10^{-13}$
1280	90.8496	$1.3877787807814457 \times 10^{-14}$
2560	364.26	$6.661338147750939 \times 10^{-16}$

Surprisingly the higher order method is far inferior to the simple Trapezoidal rule for this problem. The method does however converge to machine error with sufficiently many steps.

Table 7.5: Error Reduction and Timing Increase Table for Problem 1 Using the Runge Kutta Method

Beginning Steps	Ending Steps	Error Reduction Factor	Timing Increase Factor
10	20	15.6001	3.08181
20	40	15.8	3.78847
40	80	15.9	3.88374
80	160	15.9498	3.92752
160	320	15.9782	3.98497
320	640	15.9543	3.93773
640	1280	15.392	4.01157
1280	2560	20.8333	4.00949

The Runge-Kutta method implemented is a fourth order method. Thus we would expect that halving the step size would reduce the error by a factor of 16 which is what is observed. Halving the step size also increases the running time by a factor of

about four, for the same reason as discussed for the Trapezoidal rule.

7.2 Tables for Problem 2

Table 7.6: Trapezoidal Rule Timing Table for Problem 2 over the interval [0,1]

Order	Timing(secs)	Max. Error
10	0.000691	0.021557
20	0.001669	0.00534552
40	0.005291	0.00133367
80	0.017845	0.000333249
160	0.067059	0.0000833017
320	0.264632	0.0000208248
640	1.05398	$5.2061487165744325 \times 10^{-6}$
1280	4.17183	$1.301534598319165 \times 10^{-6}$
2560	16.5217	$3.2538348637700665 \times 10^{-7}$

Table 7.7: Error Reduction and Timing Increase Table for Problem 2 Using the Trapezoidal Rule over the interval [0,1]

Beginning Steps	Ending Steps	Error Reduction Factor	Timing Increase Factor
10	20	4.03272	2.41534
20	40	4.00812	3.17016
40	80	4.00203	3.37271
80	160	4.00051	3.75786
160	320	4.00013	3.94626
320	640	4.00003	3.98281
640	1280	4.00001	3.95817
1280	2560	4	3.9603

This time we see the behavior that we expect from the second order Trapezoidal method. Halving the step size reduces the error by a factor of about four. Halving the step size increases the running time by a factor of about four.

Table 7.8: Bernstein Method Timing Table for Problem 2 over the interval [0,1]

Order	Timing(secs)	Max. Error
1	0.019845	2.97795
2	0.017827	0.92154
3	0.031647	0.439376
4	0.049405	0.0375763
5	0.071359	0.0172435
6	0.097469	0.000986258
7	0.127991	0.000356974
8	0.163213	0.0000153604
9	0.202942	$1.5893401581479338 \times 10^{-6}$
10	0.247013	$1.5166496591945133 \times 10^{-7}$
11	0.295541	$1.2752749078970282 \times 10^{-8}$
12	0.348597	$1.0659786386923997 \times 10^{-9}$
13	0.4045	$7.807532398373951 \times 10^{-11}$
14	0.467772	$5.078826248450241 \times 10^{-12}$
15	0.533793	$9.724443472691746 \times 10^{-13}$
16	0.604626	$9.196865491389872 \times 10^{-12}$
17	0.679106	$1.1101675134739253 \times 10^{-11}$
18	0.758819	$2.185884806493732 \times 10^{-11}$
19	0.843523	$2.2247759190463512 \times 10^{-11}$
20	0.932505	$2.79180789597433 \times 10^{-10}$

Notice that the best error bound achieved with the Bernstein method is of the order of 10^{-13}. This is better than anything achieved with the Trapezoidal method but is not yet on the order of machine error.

Table 7.9: Runge-Kutta Timing Table for Problem 2 over the interval [0,1]

Order	Timing(secs)	Max. Error
10	0.008096	0.000165707
20	0.025218	0.0000110231
40	0.095702	$7.10713931262319 \times 10^{-7}$
80	0.3738	$4.5115519320404474 \times 10^{-8}$
160	1.47685	$2.841717083867934 \times 10^{-9}$
320	5.89666	$1.7829915321954104 \times 10^{-10}$
640	23.4157	$1.1163514557210874 \times 10^{-11}$
1280	93.9771	$7.007727731433988 \times 10^{-13}$
2560	374.72	$5.062616992290714 \times 10^{-14}$

The Runge-Kutta method achieves the best error bound but takes a very long time to do so. The method shows every sign of converging to machine error.

Table 7.10: Error Reduction and Timing Increase Table for Problem 2 by the Runge-Kutta Method over the interval [0,1]

Beginning Steps	Ending Steps	Error Reduction Factor	Timing Increase Factor
10	20	15.0328	3.11487
20	40	15.5099	3.79499
40	80	15.7532	3.90587
80	160	15.8761	3.95092
160	320	15.9379	3.99271
320	640	15.9716	3.971
640	1280	15.9303	4.01343
1280	2560	13.8421	3.98735

The drop in error reduction shows that round off error is starting to be a factor.

Table 7.11: Trapezoidal Rule Timing Table for Problem 2 over the interval [0,3]

Order	Timing(secs)	Max. Error
10	0.000682	339.954
20	0.001646	75.8224
40	0.005372	18.451
80	0.017589	4.58215
160	0.066291	1.14364
320	0.262284	0.285792
640	1.04602	0.0414405
1280	4.14747	0.178597
2560	16.4727	0.00446489

Table 7.12: Error Reduction and Timing Increase Table for Problem 2 Using the Trapezoidal Rule over the interval [0,3]

Beginning Steps	Ending Steps	Error Reduction Factor	Timing Increase Factor
10	20	4.48355	2.41349
20	40	4.10939	3.26367
401	80	4.02671	3.2742
80	160	4.00664	3.76889
160	320	4.00166	3.95656
320	640	4.00041	3.98814
640	1280	4.0001	3.96498
1280	2560	4.00003	3.97174

Table 7.13: Bernstein Method Timing Table for Problem 2 over the interval [0,3]

Order	Timing	Error
1	0.009534	1800.16
2	0.018746	1801.96
3	0.032831	1797.81
4	0.05142	1807.23
5	0.074562	1785.57
6	0.101703	1839.79
7	0.133106	1702.65
8	0.169308	2173.1
9	0.210015	1158.27
10	0.254166	1860.03
11	0.304261	211.23
12	0.3579	59.5576
13	0.416215	12.9417
14	0.479364	0.000820602
15	0.549089	0.000438734
16	0.620964	0.000146027
17	0.699087	0.0180487
18	0.782406	$9.548744188394009 \times 10^{-7}$
19	0.865245	$1.380227908775548 \times 10^{-7}$
20	0.956476	$2.5209547429109432 \times 10^{-8}$

The Bernstein method does very well here. Notice the continual improvement. With a larger interval, higher order polynomial approximations may be taken without the problem of ill conditioning of the matrix inversion arising.

Table 7.14: Runge-Kutta Timing Table for Problem 2 over the interval [0,3]

Order	Timing(secs)	Max. Error
10	0.008103	8.95919
20	0.02856	0.699196
40	0.097466	0.0487999
80	0.379152	0.00322316
160	1.49313	0.000207093
320	5.99036	0.0000131236
640	23.5635	$8.259155492851278 \times 10^{-7}$
1280	93.6748	$5.180049811315257 \times 10^{-8}$
2560	372.355	$3.2446223485749215 \times 10^{-9}$

The Runge-Kutta method achieves an excellent error bound and shows continuous improvement. However it is already taking a very long time to run.

Table 7.15: Error Reduction and Timing Increase Table for Problem 2 by the Runge-Kutta Method over the interval [0,3]

Beginning Steps	Ending Steps	Error Reduction Factor	Timing Increase Factor
10	20	12.8136	3.52462
20	40	14.3278	3.41268
40	80	15.1404	3.8901
80	160	15.5638	3.93809
160	320	15.7803	4.01193
320	640	15.8897	3.93357
640	1280	15.9442	3.97542
1280	2560	15.965	3.97498

7.3 Tables for Problem 3

Table 7.16: Trapezoidal Rule Timing Table for Problem 3 over the interval [0,1]

Order	Timing(secs)	Max. Error
10	0.000699	0.00429285
20	0.001537	0.00107101
40	0.005007	0.000267615
80	0.017799	0.0000668952
160	0.060912	0.0000167233
320	0.236339	$4.180783034657409 \times 10^{-6}$
640	0.941749	$1.0451936658384398 \times 10^{-6}$
1280	3.78179	$2.6129828589738224 \times 10^{-7}$
2560	14.9894	$6.532456398034014 \times 10^{-8}$

Table 7.17: Error Reduction and Timing Increase Table for Problem 3 Using the Trapezoidal Rule over the interval [0,1]

Beginning Steps	Ending Steps	Error Reduction Factor	Timing Increase Factor
10	20	4.00822	2.19886
20	40	4.00205	3.25764
40	80	4.00051	3.55482
80	160	4.00013	3.42221
160	320	4.00003	3.88001
320	640	4.00001	3.98474
640	1280	4	4.01571
1280	2560	4	3.96358

Table 7.18: Bernstein Method Timing Table for Problem 3 over the interval [0,1]

Order	Timing(secs)	Max. Error
1	0.023748	0.489434
2	0.040261	0.0398365
3	0.071993	0.00275528
4	0.112654	0.000174949
5	0.162781	0.0000272952
6	0.222144	$2.72038753745818 \times 10^{-8}$
7	0.292348	$1.967189300344785 \times 10^{-9}$
8	0.37032	$1.4878409615448618 \times 10^{-10}$
9	0.457422	$2.5697222127973873 \times 10^{-12}$
10	0.55481	$7.838174553853605 \times 10^{-14}$
11	0.661717	$8.881784197001252 \times 10^{-16}$
12	0.778853	$4.6629367034256575 \times 10^{-15}$
13	0.904019	$3.552713678800501 \times 10^{-15}$
14	1.03709	$3.774758283725532 \times 10^{-15}$
15	1.1825	$1.7763568394002505 \times 10^{-15}$
16	1.3376	$4.218847493575595 \times 10^{-14}$
17	1.5012	$4.1522341120980855 \times 10^{-14}$
18	1.67566	$2.544631172440859 \times 10^{-13}$
19	1.85913	$2.449351832467528 \times 10^{-11}$
20	2.05144	$1.1219913886861832 \times 10^{-12}$

Table 7.19: Runge-Kutta Timing Table for Problem 3 over the interval [0,1]

Order	Timing(secs)	Max. Error
10	0.007286	$2.110080666861691 \times 10^{-6}$
20	0.025679	$1.3615100380448553 \times 10^{-7}$
40	0.086431	$8.639460924442233 \times 10^{-9}$
80	0.33524	$5.439686479036254 \times 10^{-10}$
160	1.32383	$3.4121816483434486 \times 10^{-11}$
320	5.31318	$2.136069099378801 \times 10^{-12}$
640	21.3136	$1.3322676295501878 \times 10^{-13}$
1280	85.2636	$8.659739592076221 \times 10^{-15}$
2560	341.8	$1.6653345369377348 \times 10^{-15}$

Table 7.20: Error Reduction and Timing Increase Table for Problem 3 using the Runge-Kutta Method over the interval [0,1]

Beginning Steps	Ending Steps	Error Reduction Factor	Timing Increase Factor
10	20	15.4981	3.52443
20	40	15.7592	3.36582
40	80	15.8823	3.8787
80	160	15.942	3.94889
160	320	15.9741	4.0135
320	640	16.0333	4.01146
640	1280	15.3846	4.00043
1280	2560	5.2	4.00874

Table 7.21: Trapezoidal Method Timing Table for Problem 3 over the interval [0,8]

Order	Timing(secs)	Max. Error
10	0.000724	31.7243
20	0.001558	7.38853
40	0.00499	1.79668
80	0.015874	0.446042
160	0.060119	0.111459
320	0.241465	0.0278531
640	0.949773	0.00696255
1280	3.76136	0.00174059
2560	14.7772	0.000435145

Table 7.22: Error Reduction and Timing Increase Table for Problem 3 Using the Trapezoidal Rule over the interval [0,8]

Beginning Steps	Ending Steps	Error Reduction Factor	Timing Increase Factor
10	20	4.29372	2.15193
20	40	4.11232	3.20282
40	80	4.02806	3.18116
80	160	4.00186	3.78726
160	320	4.00166	4.01645
320	640	4.00042	3.93338
640	1280	4.0001	3.96027
1280	2560	4.00003	3.92869

Table 7.23: Bernstein Method Timing Table for Problem 3 over the interval [0,8]

Order	Timing	Error
1	0.021429	10.9985
2	0.046075	61.1172
3	0.079967	4.68055
4	0.126766	21.5964
5	0.180314	8.29718
6	0.247763	4.53152
7	0.320999	1.55252
8	0.409963	7.79191
9	0.504563	0.186727
10	0.613133	0.0170569
11	0.727627	0.0153657
12	0.861085	0.00240647
13	0.99173	0.000174676
14	1.14266	0.000121832
15	1.29805	0.000015451
16	1.47621	$8.784348035639766 \times 10^{-7}$
17	1.65022	$5.017704307608284 \times 10^{-7}$
18	1.83886	$5.578385886551018 \times 10^{-8}$
19	2.02976	$5.83988790658907 \times 10^{-9}$
20	2.24905	$5.782021528233372 \times 10^{-9}$

Table 7.24: Runge-Kutta Timing Table for Problem 3 over the interval [0,8]

Order	Timing(secs)	Max. Error
10	0.007394	2.26074
20	0.024582	0.135629
40	0.086318	0.00811976
80	0.340262	0.000493281
160	1.33524	0.0000303034
320	5.37228	$1.8769566523246795 \times 10^{-6}$
640	21.357	$1.1677632016926509 \times 10^{-7}$
1280	85.8606	$7.281499847522355 \times 10^{-9}$
2560	347.688	$4.546336640487425 \times 10^{-10}$

Table 7.25: Error Reduction and Timing Increase Table for Problem 3 Using the Runge-Kutta Method over the interval [0,8]

Beginning Steps	Ending Steps	Error Reduction Factor	Timing Increase Factor
10	20	16.6685	3.32459
20	40	16.7036	3.51143
40	80	16.4607	3.94196
80	160	16.2781	3.92416
160	320	16.1449	4.02345
320	640	16.0731	3.97541
640	1280	16.0374	4.02025
1280	2560	16.0162	4.04945

7.4 Tables for Problem 4

Table 7.26: Timing Table for the Product Integration Using the Trapezoidal Rule for Problem 4

Steps	Time(Sec)
10	0.003591
20	0.012515
40	0.050578
80	0.212187
160	0.8863
320	3.6838
640	15.2905
1280	62.9799

Table 7.27: Timing Increase Table for the Product Integration Using the Trapezoidal Rule for Problem 4

Starting Steps	Ending Steps	Time Increase Factor
10	20	3.4851
20	40	4.04139
40	80	4.19524
80	160	4.17698
160	320	4.15638
320	640	4.15073
640	1280	4.1189

Table 7.28: Error Table for the Product Integration Using the Trapezoidal Rule for Problem 4

Steps	Error
10	0.0651713
20	0.0169585
40	0.00440897
80	0.00113511
160	0.000289715
320	0.0000734801
640	0.0000185553
1280	$4.671491402152839 \times 10^{-6}$

Table 7.29: Error Reduction Table for the Product Integration Using the Trapezoidal Rule for Problem 4

Starting Steps	Ending Steps	Error Reduction Factor
10	20	3.84299
20	40	3.84636
40	80	3.88419
80	160	3.91801
160	320	3.94277
320	640	3.96005
640	1280	3.97203

Table 7.30: Bernstein Method Timing Table for Problem 4

Order	Timing	Error
1	0.012898	1.10984
2	0.026643	6.49126
3	0.047695	1.40834
4	0.074391	0.598712
5	0.110449	0.173146
6	0.15129	0.0146718
7	0.197972	$3.2613609084761326 \times 10^{-7}$
8	0.252735	$2.0768546874530767 \times 10^{-8}$
9	0.314958	$6.119413598172586 \times 10^{-7}$
10	0.382049	$2.652869509358843 \times 10^{-7}$
11	0.457308	$1.3223874682131288 \times 10^{-6}$
12	0.538063	$3.488993169681961 \times 10^{-6}$
13	0.631346	$4.342837784791201 \times 10^{-7}$
14	0.727892	$2.151348543755205 \times 10^{-11}$
15	0.822648	0.0000105119
16	0.92826	$2.4965141710912953 \times 10^{-6}$
17	1.0408	0.0000655347
18	1.1678	0.000122196
19	1.29	0.000281478
20	1.42722	0.000128001

Table 7.31: Timing Table for Product Integration Using Simpson's Method for the Problem 4

Steps	Timing(secs)
10	0.008141
20	0.026987
40	0.104847
80	0.423822
160	1.74383
320	7.18449
640	29.1522
1280	119.477

Table 7.32: Timing Increase Table for the Product Integration Using Simpson's Method for the Problem 4

Staring Steps	Ending Steps	Time Increase Factor
10	20	3.31495
20	40	3.88509
40	80	4.04229
80	160	4.11455
160	320	4.11993
320	640	4.05765
640	1280	4.0984

Table 7.33: Error Table for the Product Integration Using Simpson's Method for Problem 4

Steps	Error
10	0.00722974
20	0.000790282
40	0.0000796226
80	$7.643984999194942 \times 10^{-6}$
160	$7.128824821567292 \times 10^{-7}$
320	$6.529845020253333 \times 10^{-8}$
640	$5.911497513721997 \times 10^{-9}$
1280	$5.320657248120142 \times 10^{-10}$

Table 7.34: Error Reduction Table for the Product Integration Using Simpson's Method for Problem 4

Starting Steps	Ending Steps	Error Reduction Factor
10	20	9.1483
20	40	9.92535
40	80	10.4164
80	160	10.7226
160	320	10.9173
320	640	11.046
640	1280	11.1105

8 Conclusions

In this paper, the Bernstein polynomial method was compared to multiple methods of approximation for Volterra integral equations. The Bernstein method, or collocation method, produced very fast results. The timings are not directly comparable to our method since the collocation method uses optimized routines for both integration and matrix inversion whereas our numerical methods do not. Also, the collocation method does not reliably converge to machine-level error, so the error is not stable. Our higher-order Runge-Kutta method and Simpson's product method produce better error bounds at the expense of a longer execution time. Thus, the Runge-Kutta and Simpson's product methods provide highly accurate results for Volterra integral equations of the second kind with regular and weakly singular kernels, respectively.

A Appendix of Numerical Codes in Mathematica

A.1 Code for the Modified Euler/ Trapezoidal Rule Method

```
VolterraSolve[a_, K_, t0_, t1_, steps_] := Module[
   {s, dt = N[(t1 - t0)/steps], tcoord, xcoord, i, j}
   (*Local Variables *),
   (* Initialize the array of t values *)

   tcoord = Range[t0, t1, dt];(*
   Initialize an empty array to hold x values*)

   xcoord = ConstantArray[0, steps + 1];
   (* First x value does not involve an integral *)

   xcoord[[1]] = a[t0];
   (* For each step compute the new value of x by using the \
trapezoidal approximation to the integral *)

   For[i = 2, i <= steps + 1, i++,
    xcoord[[i]] =
     2/(2 - K[tcoord[[i]], tcoord[[i]]] dt)*(a[tcoord[[i]]] +
        1/2 K[tcoord[[i]], tcoord[[1]]] xcoord[[1]] dt +
        Sum[ K[tcoord[[i]], tcoord[[j]]] xcoord[[j]] dt , {j, 2,
```

```
          i - 1}])];
   (* Print[xcoord,tcoord];(* Debugging Code *) *)

   Return[Transpose[{tcoord, xcoord}]]];
```

A.2 Code for the Runge-Kutta Method

```
Clear[A, \[Theta]]; order = 4;
\[Theta][0] = 0; \[Theta][1] = 1/2; \[Theta][2] =
 1/2 ; \[Theta][3] = 1;
A[1, 0] = 1/2;
A[2, 0] = 0;
A[2, 1] = 1/2;
A[3, 0] = 0;
A[3, 1] = 0;
A[3, 2] = 1;
A[4, 0] = 1/6;
A[4, 1] = 1/3;
A[4, 2] = 1/3;
A[4, 3] = 1/6;
rSolve[a_, k_, t0_, t1_, maxSteps_] :=
 Module[{h = N[(t1 - t0)/maxSteps], x, t, i, j, l, m},
  Do[t[i, j] = t0 + (i + \[Theta][j])*h, {i, 0, maxSteps}, {j, 0,
    order - 1}]; x[0, 0] = a[t[0, 0]];
```

```
x[i_, 0] :=
 x[i, 0] =
  a[t[i, 0]] +
   h*Sum[A[order, l] k[t[i, 0], t[m, l], x[m, l]], {m, 0,
      i - 1}, {l, 0, order - 1}];
x[i_, j_] :=
 x[i, j] =
  a[t[i, j]] +
   h*Sum[A[order, l] k[t[i, j], t[m, l], x[m, l]], {m, 0,
      i - 1}, {l, 0, order - 1}] +
   h*Sum[A[j, l] k[t[i, j], t[i, l], x[i, l]], {l, 0, j - 1}];
Return[ Table[{t[i, 0], x[i, 0]}, {i, 0, maxSteps}]]]
```

A.3 Bernstein Polynomial Method

Here is the code used to define our own Bernstein polynomial. Mathematica contains its own Bernstein polynomial but it is scaled to $[0, 1]$.

```
BernsteinPolynomial[i_, n_, {a_, b_}] := (Binomial[n, i]
 (# - a)^i (b - #)^(n - i) /(b - a)^n ) &
```

The following is the solution method.

```
bSolve[a_, k_, t0_, t1_, order_] :=
 Module[{\[Alpha], \[Alpha]Soln, eqns, i, t,
   h = (t1 - t0)/(order + 2)}, Do[t[i] = (i + 1) h, {i, 0, order}];
  eqns = Table[Sum[\[Alpha][i] (
        BernsteinPolynomial[i, order , {t0, t1}][t[j]] -
         NIntegrate[
          k[t[j], s] BernsteinPolynomial[i, order, {t0, t1}][s], {s,
           0, t[j]}]), {i, 0, order}] == a[t[j]], {j, 0, order}];
  \[Alpha]Soln = Solve[eqns, Table[\[Alpha][i], {i, 0, order}]];
  Return[Evaluate[
     Sum[\[Alpha][i] BernsteinPolynomial[i, order, {t0, t1}][#], {i,
        0, order}] /. \[Alpha]Soln[[1]]] &]]
```

A.4 Product Trapezoidal Method

This first code uses numerical integration to compute the weights $w_{i,j}$ and as a consequence runs very slowly.

```
TrapProductSolve[a_, p_, t0_, t1_, steps_] := Module[
   {s, h = N[(t1 - t0)/steps], tcoord, xcoord, i, j, w} (*
   Local Variables *),
   (* Initialize the array of t values *)

   tcoord = Range[t0, t1, h];(*
```

```
   Initialize an empty array to hold x values*)

   xcoord = ConstantArray[0, steps + 1];
   (* these are the weights for the product integration *)

   w[i_, j_] :=
    1/h    NIntegrate[
       p[tcoord[[i + 1]], s] (tcoord[[2]] - s), {s, tcoord[[1]],
        tcoord[[2]]}] /; (j == 0);
   w[i_, j_] :=
    1/h  NIntegrate[
        p[tcoord[[i + 1]], s] (tcoord[[j + 2]] - s), {s,
         tcoord[[j + 1]], tcoord[[j + 2]]}] +
      1/h   NIntegrate[
        p[tcoord[[i + 1]], s] (s - tcoord[[j]]), {s, tcoord[[j]],
         tcoord[[j + 1]]}] /; (0 < j < i);
   w[i_, j_] :=
    1/h   NIntegrate[
       p[tcoord[[i + 1]], s] (s - tcoord[[i]]), {s, tcoord[[i]],
        tcoord[[i + 1]]}] /; (j == i);
   (* First x value does not involve an integral *)

   xcoord[[1]] = a[t0];
   (* For each step compute the new value of x by using the \
trapezoidal approximation to the integral *)
```

```
   For[i = 1, i <= steps, i++,
    xcoord[[
       i + 1]] = (a[tcoord[[i + 1]]] +
         Sum[w[i, j] xcoord[[j + 1]], {j, 0, i - 1}])/(1 -
         w[i, i]);];
   (* Print[xcoord,tcoord];(* Debugging Code *) *)

   Return[Transpose[{tcoord, xcoord}]]];
```

This uses the weights that were computed analytically in order to dramatically speed up execution.

```
SpecialTrapProductSolve[ a_, t0_, t1_, steps_] := Module[
   {s, dt = N[(t1 - t0)/steps], tcoord, xcoord, i, j, w} (*
   Local Variables *),
   (* Initialize the array of t values *)

   tcoord = Range[t0, t1, dt];(*
   Initialize an empty array to hold x values*)

   xcoord = ConstantArray[0, steps + 1];
   (* these are the weights for the product integration *)

   w[i_, j_] :=
```

```
    4/3 Sqrt[
       dt] ( (i - j - 1)^(3/2) -
        2 (i - j)^(3/2) + (i - j + 1)^(3/2)) /; (0 < j < i);
   w[i_, j_] :=
    2 Sqrt[dt i] + 4/3 (i - 1) Sqrt[dt (i - 1)] -
      4/3 i Sqrt[dt i] /; (j == 0);
   w[i_, j_] := (4 dt^(1/2))/3 /; (j == i);
   (* First x value does not involve an integral *)

   xcoord[[1]] = a[0];
   (* For each step compute the new value of x by using the \
trapezoidal approximation to the integral *)

   For[i = 1, i <= steps, i++,
    xcoord[[
       i + 1]] = (a[tcoord[[i + 1]]] +
         Sum[w[i, j] xcoord[[j + 1]], {j, 0, i - 1}])/(1 -
         w[i, i]);];
   (* Print[xcoord,tcoord];(* Debugging Code *) *)

   Return[Transpose[{tcoord, xcoord}]]];
```

A.5 Product Simpson's Method

This first code uses numerical integration to compute the weights $w_{i,j}$ and as a consequence runs very slowly.

```
SimpProductSolve[a_, p_, t0_, t1_, steps_] := Module[
   {s, h = N[(t1 - t0)/steps], tcoord, xcoord, i, j, w} (*
   Local Variables *),
   (* Initialize the array of t values *)

   tcoord = Range[t0, t1, h];(*
   Initialize an empty array to hold x values*)

   xcoord = ConstantArray[0, steps + 1];
   (*To get the Simpson's Method Started we need to use a trapezoidal \
step to find x[t[1]]. *)

   w[1, 0] =
    1/h NIntegrate[
      p[tcoord[[2]], s] (tcoord[[2]] - s), {s, tcoord[[1]],
       tcoord[[2]]}];
   w[1, 1] =
    1/h NIntegrate[
      p[tcoord[[2]], s] (s - tcoord[[1]]), {s, tcoord[[1]],
       tcoord[[2]]}];
```

```
(* If i is even we simply use the standard Simpson's rule *)
(*
These are the weights for the product integration *)

w[i_, j_] :=
 1/(2 h^2)    NIntegrate[
    p[tcoord[[i + 1]], s] (tcoord[[2]] - s) (tcoord[[3]] - s ), {s,
      tcoord[[1]], tcoord[[3]]}] /; (j == 0 && EvenQ[i]);
w[i_, j_] :=
 1/h^2  NIntegrate[
    p[tcoord[[i + 1]],
      s] (s - tcoord[[j]]) (tcoord[[j + 2]] - s), {s, tcoord[[j]],
     tcoord[[j + 2]]}] /; (OddQ[j] && EvenQ[i] );
w[i_, j_] :=
 1/(2 h^2) (
    NIntegrate[
      p[tcoord[[i + 1]],
        s] (s - tcoord[[j - 1]]) (s - tcoord[[j]]), {s,
       tcoord[[j - 1]], tcoord[[j + 1]]}] +
     NIntegrate[
      p[tcoord[[i + 1]],
        s] (tcoord[[j + 2]] - s) (tcoord[[j + 3]] - s), {s,
       tcoord[[j + 1]], tcoord[[j + 3]]}]) /; (EvenQ[j] &&
    EvenQ[i] && (0 < j < i) );
w[i_, j_] :=
```

```
 1/(2 h^2)   NIntegrate[
    p[tcoord[[i + 1]],
      s] (s - tcoord[[i]]) (s - tcoord[[i - 1]]), {s,
     tcoord[[i - 1]], tcoord[[i + 1]]}] /; (j == i && EvenQ[i]);
(* If i is odd (and greater than 1 then we use a four step method \
to generate x[t[
3]] and then we use the standard Simpson's rule *)
(*
These are the weights for the 4 point product integration *)

w[i_, j_] :=
 1/(6 h^3)    NIntegrate[
    p[tcoord[[i + 1]],
      s] (tcoord[[4]] - s) (tcoord[[3]] - s) (tcoord[[2]] -
       s ), {s, tcoord[[1]], tcoord[[4]]}] /; (j == 0 && OddQ[i] &&
     i > 1);
w[i_, j_] :=
 1/(2 h^3)  NIntegrate[
    p[tcoord[[i + 1]],
      s] (s - tcoord[[1]]) (tcoord[[3]] - s) (tcoord[[4]] - s), {s,
      tcoord[[1]], tcoord[[4]]}] /; (j == 1 && OddQ[i] && i > 1);
w[i_, j_] :=
 1/(2 h^3)  NIntegrate[
    p[tcoord[[i + 1]],
      s] (s - tcoord[[1]]) (s - tcoord[[2]]) (tcoord[[4]] - s), {s,
```

```
        tcoord[[1]], tcoord[[4]]}] /; (j == 2 && OddQ[i] && i > 1);
  (* There are two cases for the last point of the 4 point \
integration depending on whether it is the last point of the integral \
or not *)
  (* If last point *)

  w[i_, j_] :=
   1/(6 h^3)  NIntegrate[
      p[tcoord[[i + 1]],
        s] (s - tcoord[[1]]) (s - tcoord[[2]]) (s - tcoord[[3]]), {s,
        tcoord[[1]], tcoord[[4]]}] /; (j == 3 && i == 3);
  (* If not last point then this is intermediate between the 4 point \
integration and a 3 point integration *)

  w[i_, j_] :=
   1/(6 h^3)  NIntegrate[
       p[tcoord[[i + 1]],
         s] (s - tcoord[[1]]) (s - tcoord[[2]]) (s -
          tcoord[[3]]), {s, tcoord[[1]], tcoord[[4]]}] +
     1/(2 h^2)    NIntegrate[
       p[tcoord[[i + 1]],
         s] (tcoord[[5]] - s) (tcoord[[6]] - s ), {s, tcoord[[4]],
        tcoord[[6]]}] /; (j == 3 && OddQ[i] && i > 3);
  (* The remaining computations are as they were in the i even case *)
```

```
      w[i_, j_] :=
    1/h^2  NIntegrate[
       p[tcoord[[i + 1]],
         s] (s - tcoord[[j]]) (tcoord[[j + 2]] - s), {s, tcoord[[j]],
        tcoord[[j + 2]]}] /; (EvenQ[j] && OddQ[i] && j > 3 );
   w[i_, j_] :=
    1/(2 h^2) (
       NIntegrate[
         p[tcoord[[i + 1]],
           s] (s - tcoord[[j - 1]]) (s - tcoord[[j]]), {s,
          tcoord[[j - 1]], tcoord[[j + 1]]}] +
        NIntegrate[
         p[tcoord[[i + 1]],
           s] (tcoord[[j + 2]] - s) (tcoord[[j + 3]] - s), {s,
          tcoord[[j + 1]], tcoord[[j + 3]]}]) /; (OddQ[j] && OddQ[i] &&
        3 < j < i );
   (* If the last point of the integral is not the last point of the \
4 point integration then it is the last point of a 3 point Simpson's \
step and is dealt with here *)

   w[i_, j_] :=
    1/(2 h^2)   NIntegrate[
       p[tcoord[[i + 1]],
         s] (s - tcoord[[i]]) (s - tcoord[[i - 1]]), {s,
        tcoord[[i - 1]], tcoord[[i + 1]]}] /; (j == i && OddQ[i] &&
```

```
       i > 3);
   (* First x value does not involve an integral *)

   xcoord[[1]] = a[t0];
   (* For each step compute the new value of x by using the \
trapezoidal approximation to the integral *)

   For[i = 1, i <= steps, i++,
    xcoord[[
       i + 1]] = (a[tcoord[[i + 1]]] +
         Sum[w[i, j] xcoord[[j + 1]], {j, 0, i - 1}])/(1 -
         w[i, i]);];
   (* Print[xcoord,tcoord];(* Debugging Code *) *)

   Return[Transpose[{tcoord, xcoord}]]];
```

This uses the weights that were computed analytically in order to dramatically speed up execution.

```
SpecialSimpProductSolve[a_, t0_, t1_, steps_] := Module[
   {s, h = N[(t1 - t0)/steps], tcoord, xcoord, i, j, w} (*
   Local Variables *),
   (* Initialize the array of t values *)

   tcoord = Range[t0, t1, h];(*
```

```
   Initialize an empty array to hold x values*)

   xcoord = ConstantArray[0, steps + 1];
   (*To get the Simpson's Method Started we need to use a trapezoidal \
step to find x[t[1]]. *)
   w[1, 0] = 2/3 Sqrt[h];
   w[1, 1] = (4 h^(1/2))/3;
   (* If i is even we simply use the standard Simpson's rule *)
   (*
   These are the weights for the product integration *)

   w[i_, j_] :=
    2 Sqrt[i h] - 2/3 Sqrt[h] (i - 2)^(3/2) -
      8/15 Sqrt[h] (i - 2)^(5/2) + 8/15 Sqrt[h] i ^(5/2) -
      2 Sqrt[h] i^(3/2) /; (j == 0 && EvenQ[i]);
   w[i_, j_] :=
    8/3 Sqrt[h] (i - j - 1)^(3/2) + 8/3 Sqrt[h] (i - j + 1)^(3/2) +
      16/15 Sqrt[h] (i - j - 1)^(5/2) -
      16/15 Sqrt[h] (i - j + 1)^(5/2) /; (OddQ[j] && EvenQ[i] );
   w[i_, j_] := -2/3 Sqrt[h] (i - j - 2)^(3/2) -
      8/15 Sqrt[h] (i - j - 2)^(5/2) - 4 Sqrt[h] (i - j)^(3/2) -
      2/3 Sqrt[h] (i - j + 2)^(3/2) +
      8/15 Sqrt[h] (i - j + 2)^(5/2) /; (EvenQ[j] &&
       EvenQ[i] && (0 < j < i) );
   w[i_, j_] := 4/5 Sqrt[2 h] /; (j == i && EvenQ[i]);
```

```
   (* If i is odd (and greater than 1 then we use a four step method \
to generate x[t[
   3]] and then we use the standard Simpson's rule *)
   (*
   These are the weights for the 4 point product integration *)

   w[i_, j_] :=
    Sqrt[h] (4/9 (i - 3)^(3/2) + 8/15 (i - 3)^(5/2) +
        16/105 (i - 3)^(7/2) + 2 Sqrt[i] - 22/9 i^(3/2) +
        16/15 i^(5/2) - 16/105 i^(7/2)) /; (j == 0 && OddQ[i] &&
       i > 1);
   w[i_, j_] :=
    Sqrt[h] (-2 (i - 3)^(3/2) - 32/15 (i - 3)^(5/2) -
        48/105 (i - 3)^(7/2) + 4 i^(3/2) + 48/105 i^(7/2) -
        8/3 i^(5/2)) /; (j == 1 && OddQ[i] && i > 1);
   w[i_, j_] :=
    Sqrt[h] (4 (i - 3)^(3/2) + 40/15 (i - 3)^(5/2) +
        16/35 (i - 3)^(7/2) - 2 i^(3/2) + 32/15 i^(5/2) -
        16/35 i^(7/2)) /; (j == 2 && OddQ[i] && i > 1);
   (* There are two cases for the last point of the 4 point \
integration depending on whether it is the last point of the integral \
or not *)
   (* If last point *)

   w[i_, j_] :=
```

```
    Sqrt[h] (-2 (i - 3)^(1/2) - 22/9 (i - 3)^(3/2) -
        16/15 (i - 3)^(5/2) - 16/105 (i - 3)^(7/2) + 4/9 i^(3/2) -
        8/15 i^(5/2) + 16/105 i^(7/2)) /; (j == 3 && i == 3);
   (* If not the last point then this is intermediate between the 4  \
point integration and a 3 point integration *)

   w[i_, j_] := -40/9 Sqrt[h] (i - 3)^(3/2) -
      8/15 Sqrt[h] (i - 3)^(5/2) - 16/105 Sqrt[h] (i - 3)^(7/2) +
      4/9 Sqrt[h] i^(3/2) - 8/15 Sqrt[h] i^(5/2) +
      16/105 Sqrt[h] i^(7/2) - 2/3 Sqrt[h] (i - 5)^(3/2) -
      8/15 Sqrt[h] (i - 5)^(5/2) /; (j == 3 && OddQ[i] && i > 3);
   (* The remaining computations are as they were in the i even case *)

      w[i_, j_] :=
    8/3 Sqrt[h] (i - j - 1)^(3/2) + 16/15 Sqrt[h] (i - j - 1)^(5/2) +
      8/3 Sqrt[h] (i - j + 1)^(3/2) -
      16/15 Sqrt[h] (i - j + 1)^(5/2) /; (EvenQ[j] && OddQ[i] &&
       j > 3 );
   w[i_, j_] := -2/3 Sqrt[h] (i - j - 2)^(3/2) -
      8/15 Sqrt[h] (i - j - 2)^(5/2) - 4 Sqrt[h] (i - j)^(3/2) -
      2/3 Sqrt[h] (i - j + 2)^(3/2) +
      8/15 Sqrt[h] (i - j + 2)^(5/2) /; (OddQ[j] && OddQ[i] &&
       3 < j < i );
   (* If the last point of the integral is not the last point of the \
4 point integration then it is the last point of a 3 point Simpson's \
```

```
step and is dealt with here *)

   w[i_, j_] := 4/5 Sqrt[2 h] /; (j == i && OddQ[i] && i > 3);
   (* First x value does not involve an integral *)

   xcoord[[1]] = a[t0];
   (* For each step compute the new value of x by using the \
trapezoidal approximation to the integral *)

   For[i = 1, i <= steps, i++,
    xcoord[[
       i + 1]] = (a[tcoord[[i + 1]]] +
         Sum[w[i, j] xcoord[[j + 1]], {j, 0, i - 1}])/(1 -
         w[i, i]);];
   (* Print[xcoord,tcoord];(* Debugging Code *) *)

   Return[Transpose[{tcoord, xcoord}]]];
```

Bibliography

[DD01] K. R. Davidson and A. P. Donsig. *Real Analysis with Real Applications*. Prenitice Hall, 2001.

[DM85] L. M. Delves and J. L. Mohamed. *Computational methods for integral equations*. Cambridge University Press, Cambridge, 1985.

[Jer85] A. J. Jerri. *Introduction to Integral Equations with Applications*. Marcel Dekker, Inc., 1985.

[MB07] B. N. Mandal and Subhra Bhattacharya. Numerical solution of some classes of integral equations using Bernstein polynomials. *Appl. Math. Comput.*, 190(2):1707–1716, 2007.