An-Najah National University

Faculty of Graduate Studies

Analytical and Numerical Solutions of Volterra Integral Equation of the Second Kind

By

Feda' Abdel Aziz Mustafa Salameh

Supervisor

Prof. Naji Qatanani

This Thesis is Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Computational Mathematics, Faculty of Graduate Studies, An-Najah National University, Nablus, Palestine.

2014

Analytical and Numerical Solutions of Volterra Integral Equation of the Second Kind

By

Feda' Abdel Aziz Mustafa Salameh

This thesis was defended successfully on 21/12/2014 and approved by:

Defense Committee Members		Signature
– Prof. Naji Qatanani	(Supervisor)	.....
– Dr. Iyad Suwan	(External Examiner)	.....
– Dr. Anwar Saleh	(Internal Examiner)	.....

Dedication

I dedicate this thesis to my beloved Palestine, my parents, my love my husband Emad, my children Shahd,Qays and Karam, my brother Amjad, who helped me, stood by me and encouraged me.

Acknowledgement

First of all, I thank my God for all the blessing he bestowed on me and continues to bestow on me.

I would sincerely like to thank and deeply grateful to my supervisor Prof. Dr. Naji Qatanani who without his support, kind supervision, helpful suggestions and valuable remarks, my work would have been more difficult. My thanks also to my external examiner Dr. Iyad Suwan and to my internal examiner Dr. Anwar Saleh for their useful and valuable comments. Also, my great thanks are due to my family for their support, encouragement and great efforts for me.

Finally I would also like to acknowledge to all my teachers in An-Najah National University department of computational mathematics.

الإقرار

أنا الموقعة أدناه مقدم الرسالة التي تحمل العنوان:

Analytical and Numerical Solutions of Volterra Integral Equation of the Second Kind

أقر بأن ما اشتملت عليه هذه الرسالة إنما هي نتاج جهدي الخاص، باستثناء ما تمت الإشارة إليه حيثما ورد، و أن هذه الرسالة ككل، أو أي جزء منها لم يقدم من قبل لنيل أي درجة علمية أو بحثي لدى أية مِسسة تعليمية أو بحثية أخرى.

Declaration

The work provided in this thesis, unless otherwise referenced, is the researcher's own work, and has not been submitted elsewhere for any other degree or qualification.

Student's name: فداء عبد العزيز مصطفى سلاوة **إسم الطالبة**

Signature: فداء **التوقيع**

Date: 21/12/2014 **التاريخ**

Table of Contents

List of Figures

List of Tables

Analytical and Numerical Solutions of Volterra Integral Equation of the Second Kind

By
Feda' Abdel Aziz Mustafa Salameh
Supervisor
Prof. Naji Qatanani

Abstract

In this thesis we focus on the analytical and numerical aspects of the Volterra integral equation of the second kind. This equation has wide range of applications in physics and engineering such as potential theory, Dirichlet problems, electrostatics, the particle transport problems of astrophysics, reactor theory, contact problems, diffusion problems and heat transfer problems.

After introducing the types of integral equations, we will investigate some analytical and numerical methods for solving the Volterra integral equation of the second kind. These analytical methods include: the Adomian decomposition method, the modified decomposition method, the method of successive approximations, the series solution method and the conversion to initial value problem.

For the numerical treatment of the Volterra integral equation we will implement the following numerical methods: Quadrature methods (Trapezoidal rule, Runge-Kutta method of order two, the fourth order Runge-Kutta method), Projection methods including collocation method and Galerkin method and the Block method.

The mathematical framework of these numerical methods together with their convergence properties will be presented. These numerical methods

will be illustrated by some numerical examples. Comparisons between these methods will be drawn. Numerical results show that the Trapezoidal rule has proved to be the most efficient method in comparison to the other numerical methods.

Introduction

In recent years integral equations have attracted the attention of many scientists and researchers due to their wide range of applications in science and technology.

Many physical problems are modeled in the form of integral equations. These include potential theory, Dirichlet problems, electrostatics, contact problems, astrophysics problems and radiative heat transfer problems. (For more details see [3, 16]).

Some valid numerical methods, for solving Volterra integral equation have been developed by many researchers. Very recently, Mirzaee [25] studied a Simpson's quadrature method for solving linear Volterra integral equation of the second kind. Mustafa [27] and Campbell [11] used block methods to approximate the solution of Volterra integral equation with delay. Rahman, Hakim and Hasan [30] used Galerkin method with the Chebyshev polynomials for the numerical solution of Volterra integral equation of the second kind. Hermite polynomials were used by Rahman [29] and Shafiqul [36]. In [35] Saberi-Nadja and Heidari applied modified trapezoidal formula to solve linear integral equations of the second kind, and in [2] Aigo used repeated Simpson's and Trapezoidal quadrature rule to solve the linear Volterra integral equation of the second kind. Ahmad [1] has applied least-square technique to approximate the solution of Volterra-Fredholm integral equation of the second kind. Brunner, Hairer and Njersett [8] have used Runge-Kutta Theory for Volterra integral equation. Rahman and Islam in [31] solved Volterra integral equation of the first and the second

kind numerically by Galerkin method with Legendre polynomials. Marek and Arvet in [23] discussed the numerical solution of linear Volterra integral equation of the second kind with singularities by using collocation method. Bernstein's approximation were used in [22] by Maleknejad to find out the numerical solution of Volterra integral equation. In [37] Tahmasbi solved linear Volterra integral equation of the second kind based on the power series method. Maleknejad and Aghazadeh in [21] obtained a numerical solution of these equations with convolution kernel by using Taylor-series expansion method.

However many approaches for solving the linear and nonlinear kind of these equations may be found in [5], [10], [15], [32], [33] and [38].

In this work, some analytical methods have been used to solve the Volterra integral equation of the second kind. These methods are the Adomian decomposition method, the modified decomposition method, the series solutions, the method of successive approximations and the conversion to initial value problem.

For the numerical treatment of the Volterra integral equation of the second kind, we have implemented the following methods: Quadrature methods (Trapezoidal rule, Runge-Kutta method of order two, the fourth order Runge-Kutta method), Projection methods including collocation method and Galerkin method and the Block method.

This thesis is organized as follows: In chapter one, we introduce some basic concepts of integral equations. In chapter two, we investigate some analytical methods used to solve the Volterra integral equation. These

include: The Adomian decomposition method, the modified decomposition method, the method of successive approximations, the series solutions method and the conversion of the Volterra integral equation to ordinary differential equation. In chapter three, we implement some numerical methods for solving the Volterra integral equation. These are the Quadrature methods, Trapezoidal rule, Runge-Kutta methods, Blocks methods, the collocation method and the Galerkin method. Numerical examples and results are presented in chapter four and conclusions have been drawn.

Chapter One
Mathematical Preliminaries

Chapter One

Mathematical Preliminaries

An integral equation is an equation in which the unknown function appears under an integral sign. The most standard type of integral equation is given as

$$u(x) = f(x) + \lambda \int_{g(x)}^{h(x)} k(x,t)u(t)dt \qquad (1.1)$$

Here, $u(x)$ is the unknown function, $k(x,t)$ and $f(x)$ are known functions, λ is known constant parameter, and $g(x)$ and $h(x)$ are the limits of integration that may be both variables, constants, or mixed, and they may be in one dimension or more. The function $k(x,t)$ is known as the kernel of integral equation [39].

1.1 Classification of integral equations

1.1.1 Types of integral equations

1) Fredholm integral equations

The most standard form of a Fredholm integral equation is given by

$$\varphi(x)u(x) = f(x) + \lambda \int_a^b k(x,t)u(t)dt \qquad a \le x, t \le b, \qquad (1.2)$$

There are three kinds of Fredholm integral equations:

1. Fredholm integral equation of the second kind: when the function $\varphi(x) = 1$, then (1.2) becomes

$$u(x) = f(x) + \lambda \int_a^b k(x,t)u(t)dt \qquad a \le x, t \le b, \qquad (1.3)$$

2. Fredholm integral equation of the first kind: when the function $\varphi(x) = 0$, then (1.2) becomes

$$f(x) + \lambda \int_a^b k(x,t)u(t)dt = 0 \qquad a \le x, t \le b, \qquad (1.4)$$

3. Fredholm integral equation of the third kind: when $\varphi(x)$ is neither 0 nor 1.

2) **Volterra integral equations**

The most standard form of Volterra integral equation is given as

$$\varphi(x)u(x) = f(x) + \lambda \int_a^x k(x,t)u(t)dt, \qquad (1.5)$$

where the upper limit of integration is variable and the unknown function appears linearly or nonlinearly under the integral sign.

There are three kinds of Volterra integral equations:

1. Volterra integral equation of the second kind: when the function $\varphi(x) = 1$, then (1.5) becomes

$$u(x) = f(x) + \lambda \int_a^x k(x,t)u(t)dt\ , \qquad (1.6)$$

2. Volterra integral equation of the first kind: when the function $\varphi(x) = 0$, then (1.5) becomes

$$f(x) + \lambda \int_a^x k(x,t)u(t)dt = 0\ , \qquad (1.7)$$

3. Volterra integral equation of the third kind: when $\varphi(x)$ is neither 0 nor 1. (see [39],[40] and [9]).

3) Singular integral equations

A singular integral equation is an equation in which one or both limits of integration are infinite or when the kernel becomes infinite at one or more points within the range of integration. For example, the integral equation,

$$u(x) = f(x) + \lambda \int_{-\infty}^{\infty} u(t)dt \qquad (1.8)$$

is a singular integral equation of the second kind.

1. Weakly singular integral equation: The kernel is of the form

$$k(x,t) = \frac{H(x,t)}{|x-t|^{\alpha}}$$

or $$k(x,t) = H(x,t)\ln|x-t|$$

where $H(x,t)$ is bounded (that is, several times continuously differentiable) $a \le x \le b$ and $a \le t \le b$ with $H(x,t) \neq 0$ and α is a constant such that $0 < \alpha < 1$. For example, the equation of the form:

$$f(x) = \lambda \int_0^x \frac{1}{(x-t)^\alpha} u(t)dt,\ 0 < \alpha < 1 \qquad (1.9)$$

is called generalized Abel's integral equation. The equation of the second kind:

$$u(x) = f(x) + \lambda \int_0^x \frac{1}{(x-t)^\alpha} u(t)dt, \quad 0 < \alpha < 1 \qquad (1.10)$$

is called weakly singular integral equation.

2. Strongly singular integral equations: if the kernel $k(x,t)$ is of the form

$$k(x,t) = \frac{H(x,t)}{(x-t)^2}$$

and $H(x,t)$ is a differentiable function of (x,t) with $H(x,t) \neq 0$.

4) Integro-differential equations

In this type of equations, the unknown function appears as a combination of an ordinary derivative and under the integral sign,

For example:

Volterra-integro-differential equation

$$u^{(k)}(x) = f(x) + \lambda \int_a^x k(x,t)u(t)dt\,, \qquad (1.11)$$

where $u^{(k)}(x) = \frac{d^k u}{dx^k},\ k = 1,2,\dots,n$

Fredholm-integro-differential equation

$$u^{(k)}(x) = f(x) + \lambda \int_a^b k(x,t)u(t)dt\,, \qquad (1.12)$$

where $u^{(k)}(x) = \frac{d^k u}{dx^k},\ k = 1,2,\dots,n$

1.1.2 Linearity of integral equations

Definition (1.1): An integral equation is said to be linear if the unknown function $u(x)$ in the integral equation appears in a linear fashion (i.e. the exponent of the unknown function $u(x)$ inside the integral sign is one). Otherwise it called nonlinear, that is the exponent of the unknown function other than one, or if the equation contains nonlinear functions of $u(x)$.

For examples

$$u(x) = \frac{3}{2}x - \frac{1}{3} + \int_0^1 (x-t)u(t)dt. \quad (1.13)$$

is linear integral equation.

$$u(x) = 1 + \int_0^x (1+x-t)(u(t))^4 dt. \quad (1.14)$$

is nonlinear integral equation.

1.1.3 Homogeneity of integral equations

Definition (1.2): If the function $f(x)$ in the second kind of Volterra or Fredholm integral equations is identically zero, the equation is called homogeneous, otherwise it is called nonhomogeneous.

1.2 Kinds of kernels

1. Separable kernel

A kernel $k(x,t)$ is said to be separable or (degenerate) if it can be expressed in the form

$$k(x,t) = \sum_{i=1}^{n} a_i(x)b_i(t) \ , \quad (1.15)$$

where the functions $a_i(x)$ and the functions $b_i(t)$ are linearly independent. (see [17]).

2. Symmetric (or Hermitian) kernel

A complex-valued function $k(x,t)$ is called symmetric (or Hermitian) if

$$k(x,t) = k^*(x,t).$$

where the asterisk denotes the complex conjugate. For a real kernel, we have

$$k(x,t) = k(t,x).$$

1.3 The existence and uniqueness theorem

Some integral equations have a solution and some others have no solution or have an infinite number of solutions. The following theorems state the existence and uniqueness of the solution for the Voterra integral equation of the second kind.

Theorem (1.1) (Volterra's Theorem)

Assume that the kernel $k(x,t)$ of the linear Volterra integral equation

$$u(x) = f(x) + \int_0^x k(x,t)u(t)dt, \quad x \in I \coloneqq [0,T], \tag{1.16}$$

is continuous on $D \coloneqq \{(x,t): 0 \le t \le x \le T\}$. Then for any function $f(x)$ that is continuous on I (that is,$f \in C(I)$), the Volterra integral equation possesses a unique solution $u \in C(I)$. This solution can be written in the form

$$u(x) = f(x) + \int_0^x R(x,t)f(t)dt, \quad x \in I \tag{1.17}$$

for some $R \in C(D)$. The function $R = R(x,t)$ is called the resolvent kernel of the given kernel $k(x,t)$[6].

Theorem (1.2)

If we define the integral operator $K: C(I) \to C(I)$ by

$$(Kf)(x) \coloneqq \int_0^x R(x,t)f(t)dt\,, \quad x \in I \tag{1.18}$$

then the Volterra integral equation in operator form is given

$$u = f + Vu\,, \qquad \text{or} \qquad (I - V)u = f\,,$$

(where I denotes the identity operator, and the classical Volterra integral operator $V: C(I) \to C(I)$ is defined by

$(Vu)(x) \coloneqq \int_0^x k(x,t)u(t)dt, \ \ x \in I$, with $k \in C(D)$), then we have the following relationship:

$$(I - V)u = f \quad \Longrightarrow \quad u = (I + K)f.$$

By Theorem 1.1 this implies that the inverse operator $(I - V)^{-1}$always exists, and hence (by uniqueness of $R(x,t)$)

$$(I - V)^{-1} = I + K, \text{ see ([6]).}$$

Chapter Two
Analytical Methods for Solving Volterra Integral Equation of the Second Kind

Chapter Two

Analytical methods for solving Volterra integral equation of the second kind

There are many analytical methods available for solving Volterra integral equation of the second kind. In this chapter we will focus on the following methods: the Adomian decomposition method, the modified decomposition method, the method of successive approximations, the series solution method, converting Volterra integral equation to initial value problem.

2.1 The Adomian Decomposition Method

The Adomian decomposition method (ADM) was introduced and developed by George Adomian [39]. It consists of decomposing the unknown function $u(x)$ of any equation into a sum of an infinite number of components defined by the decomposition series

$$u(x) = \sum_{n=0}^{\infty} u_n(x) \ , \tag{2.1}$$

or equivalently

$$u(x) = u_0(x) + u_1(x) + u_2(x) + \cdots \ , \tag{2.2}$$

The decomposition method is concerned with finding the components $u_0, u_1, u_2, \ldots$ individually. To establish the recurrence relation, we substitute (2.1) into equation (1.6) to get

$$\sum_{n=0}^{\infty} u_n(x) = f(x) + \lambda \int_0^x k(x,t)\left(\sum_{n=0}^{\infty} u_n(t)\right)dt \tag{2.3}$$

or equivalently

$$u_0(x) + u_1(x) + u_2(x) + \cdots = f(x) + \lambda \int_0^x k(x,t)\,[u_0(t) + u_1(t) + u_2(t) + \cdots]dt \tag{2.4}$$

The components $u_j(x)$, $j \geq 1$ of the unknown function $u(x)$ are completely determined by setting the recurrence relation:

$$u_0(x) = f(x),$$
$$u_{n+1}(x) = \lambda \int_0^x k(x,t)\, u_n(t)dt\,, \qquad n \geq 0\,, \tag{2.5}$$

or equivalently

$$\begin{aligned} u_0(x) &= f(x), \\ u_1(x) &= \lambda \int_0^x k(x,t)\, u_0(t)dt, \\ u_2(x) &= \lambda \int_0^x k(x,t)\, u_1(t)dt, \\ u_3(x) &= \lambda \int_0^x k(x,t)\, u_2(t)dt, \end{aligned} \tag{2.6}$$

and so on for other components. As a result the components $u_1(x)$, $u_2(x), u_3(x), \ldots\ldots$ are completely determined, then the solution $u(x)$ of the Volterra integral equation (1.6) is readily obtained in a series form by using the series assumption in (2.1).

The decomposition method converts the integral equation into an elegant determination of computable components. If an exact solution exists for the problem, then the obtained series converges very rapidly to that exact solution. However, for concrete problems, where a closed form solution is not obtainable, a truncated number of terms is usually used for numerical purposes. The more components we use the higher accuracy we obtain [39].

Example 2.1

Consider the following Volterra integral equation of the second kind

$$u(x) = 6x - 3x^2 + \int_0^x u(t)dt\,, \tag{2.7}$$

We notice that $f(x) = 6x - 3x^2$, $\lambda = 1$, $k(x,t) = 1$. Recall that the solution $u(x)$ is assumed to have a series form given in (2.1). Substituting the decomposition series (2.1) into both sides of (2.7) gives

$$\sum_{n=0}^{\infty} u_n(x) = 6x - 3x^2 + \int_0^x \sum_{n=0}^{\infty} u_n(t)\, dt,$$

or equivalently

$$u_0(x) + u_1(x) + u_2(x) + \cdots = 6x - 3x^2 + \int_0^x [u_0(t) + u_1(t) + u_2(t) + \cdots]dt$$

We identify the zeroth component by all terms that are not included under the integral sign. Therefore, we obtain the following recurrence relation:

$$u_0(x) = 6x - 3x^2 \ ,$$
$$u_{n+1}(x) = \int_0^x u_n(t)\, dt \ , \qquad n \geq 0 \ ,$$

so that

$$u_0(x) = 6x - 3x^2,$$
$$u_1(x) = \int_0^x u_0(t)\, dt = \int_0^x 6t - 3t^2 dt = 3x^2 - x^3 \ ,$$
$$u_2(x) = \int_0^x u_1(t) dt = \int_0^x (3t^2 - t^3) dt = x^3 - \frac{x^4}{4} \ ,$$
$$u_3(x) = \int_0^x u_2(t)\, dt = \int_0^x (t^3 - \frac{t^4}{4})\, dt = \frac{x^4}{4} - \frac{x^5}{20} \ ,$$
$$u_4(x) = \int_0^x u_3(t)\, dt = \int_0^x (\frac{t^4}{4} - \frac{t^5}{20}) dt = \frac{x^5}{20} - \frac{x^6}{120},$$

The solution in a series form is given by

$$u(x) = 6x - 3x^2 + 3x^2 - x^3 + x^3 - \frac{x^4}{4} + \frac{x^4}{4} - \frac{x^5}{20} + \frac{x^5}{20} - \frac{x^6}{120} + \cdots$$

We can easily notice the appearance of identical terms with opposite signs. This phenomenon of such terms is called noise terms phenomenon. Canceling the identical terms with opposite terms gives the exact solution

$$u(x) = 6x \ .$$

2.2 The Modified Decomposition Method

As shown before, the Adomian decomposition method provides the solution in an infinite series of components. The components $u_j, j \geq 0$ are easily computed if the inhomogeneous term $f(x)$ in the Volterra integral equation:

$$u(x) = f(x) + \lambda \int_0^x k(x,t)u(t)dt\,, \tag{2.8}$$

consists of a polynomial. However, if the function $f(x)$ consists of a combination of two or more of polynomials, trigonometric functions, hyperbolic functions, and others, the evaluation of the components $u_j, j \geq 0$ require more work. A reliable modification of the Adomian decomposition method was developed by Wazwaz [39]. The modified decomposition method will facilitate the computational process and further accelerate the convergence of the series solution. This will be applied whenever it is appropriate to all integral equations and differential equations of any order. It is important to note that the modified decomposition method relies mainly on splitting the function $f(x)$ into two parts; therefore it can not be used if the function $f(x)$ consists of only one term. To explain this technique, we recall that the standard Adomian decomposition method admits the use of the recurrence relation:

$$u_0(x) = f(x),$$
$$u_{n+1}(x) = \lambda \int_0^x k(x,t)\, u_n(t)dt\,, \quad n \geq 0\,, \tag{2.9}$$

where the solution $u(x)$ is expressed by an infinite sum of components defined by

$$u(x) = \sum_{n=0}^{\infty} u_n(x)\ , \tag{2.10}$$

In virtue of (2.9), the components $u_n, n \geq 0$ can easily be evaluated. The modified decomposition method introduces a slight variation to the recurrence relation (2.9) that will lead to the determination of the components of $u(x)$ in an easier and faster manner. For many cases, the function $f(x)$ can be set as the sum of two partial functions, namely $f_1(x)$ and $f_2(x)$. In other words, we can set

$$f(x) = f_1(x) + f_2(x) \tag{2.11}$$

In virtue of (2.11), we introduce a qualitative change in the formation of the recurrence relation (2.9). To reduce the calculations, we will introduce of the modified decomposition method into recurrence relation:

$$\begin{aligned} u_0(x) &= f_1(x), \\ u_1(x) &= f_2(x) + \lambda \int_0^x k(x,t)\, u_0(t)dt\,, \\ u_{n+1}(x) &= \lambda \int_0^x k(x,t)\, u_n(t)dt\,, n \geq 1 \end{aligned} \tag{2.12}$$

This shows that the formation of the first two components $u_0(x)$ and $u_1(x)$ is only the difference between the standard recurrence relation (2.9) and the modified recurrence relation (2.12). The other components $u_j, j \geq 2$ remain the same in the two recurrence relations. This variation in the formation of $u_0(x)$ and $u_1(x)$ is important to accelerate the convergence of the solution and in minimizing the size of computational work [39].

Example 2.2

Consider the Volterra integral equation of the second kind

$$u(x) = e^x + xe^x - x - \int_0^x xu(t)dt.$$

Using the modified decomposition method, we first split $f(x)$

$$f(x) = e^x + xe^x - x$$

into two parts, namely

$$f_1(x) = e^x ,$$

$$f_2(x) = xe^x - x.$$

Next, use the modified recurrence formula (2.12) to obtain

$$u_0(x) = f_1(x) = e^x ,$$

$$u_1(x) = xe^x - x - \int_0^x xu_0(t)dt = 0,$$

$$u_{n+1}(x) = -\int_0^x k(x,t)u_n(t)dt = 0, \quad n \geq 1.$$

It is obvious that each component of $u_j, j \geq 1$ is zero. This in turn gives the exact solution by

$$u(x) = e^x.$$

2.3 The method of successive approximations

The successive approximations method provides a scheme that can be used for solving initial value problems or integral equations. This method solves any problem by finding successive approximations to the solution by starting with an initial guess as $u_0(x)$, called the zeroth approximation which can be any real-valued function $u_0(x)$, that will be used in a recurrence relation to determine the other approximations. There are two methods of successive approximations:

i) **The Picard's method:** In this method the n^{th} approximation for solving the Volterra integral equation (1.6) can be put in a recursive scheme defined by

$$u_n(x) = f(x) + \lambda \int_0^x k(x,t)u_{n-1}(t)dt, \quad n \geq 1 \qquad (2.13)$$

where the most commonly selected functions for $u_0(x)$ are 0 , 1 or x.

Accordingly, the first and the second approximation of the solution of $u(x)$ can be obtained as

$$u_1(x) = f(x) + \lambda \int_0^x k(x,t)u_0(t)dt \quad (2.14)$$

$$u_2(x) = f(x) + \lambda \int_0^x k(x,t)u_1(t)dt \quad (2.15)$$

It is obvious that $u_1(x)$ is continuous if $f(x)$, $k(x,t)$, and $u_0(x)$ are continuous. Notice that with the selection of $u_0(x) = 0$, the first approximation $u_1(x) = f(x)$. The final solution $u(x)$ is obtained by

$$u(x) = \lim_{n\to\infty} u_n(x) \quad (2.16)$$

so that the resulting solution $u(x)$ is independent of the choice of $u_0(x)$.

Example 2.3

Consider the Volterra integral equation of the second kind

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

Using the successive approximations method, we can select for the zeroth approximation $u_0(x)$

$$u_0(x) = 0. \quad (2.17)$$

The method of successive approximations admits the use of the iteration formula

$$u_{n+1}(x) = x + \int_0^x (x-t)u_n(t)dt, \qquad n \geq 0 \quad (2.18)$$

Substituting (2.17) into (2.18) we obtain

$$u_1(x) = x + \int_0^x (x-t)u_0(t)dt = x,$$

$$u_2(x) = x + \int_0^x (x-t)u_1(t)dt = x + \frac{1}{3!}x^3,$$

$$u_3(x) = x + \int_0^x (x-t)u_2(t)dt = x + \frac{1}{3!}x^3 + \frac{1}{5!}x^5,$$

$$u_4(x) = x + \int_0^x (x-t)u_3(t)dt = x + \frac{1}{3!}x^3 + \frac{1}{5!}x^5 + \frac{1}{7!}x^7,$$

$$\vdots$$

Consequently, we obtain

$$u_n(x) = \sum_{k=0}^{n} \frac{x^{2k+1}}{(2k+1)!} \ .$$

The solution $u(x)$ of (1.62) is

$$u(x) = \lim_{n\to\infty} u_n(x) \ = \sinh x \ .$$

(see [12] and [39]).

***ii*) The Neumann series method**

This method uses

$$u_0(x) = f(x).$$

Then we obtain the successive approximations:

$$u_1(x) = f(x) + \lambda \int_0^x k(x,t) f(t) dt,$$

$$u_2(x) = f(x) + \lambda \int_0^x k(x,t) u_1(x) dt,$$

$$\ldots\ldots\ldots$$

$$u_{n-1}(x) = f(x) + \lambda \int_0^x k(x,t) u_{n-2}(x) dt,$$

$$u_n(x) = f(x) + \lambda \int_0^x k(x,t) u_{n-1}(x) dt. \qquad (2.19)$$

Consider

$$\begin{aligned} u_2(x) - u_1(x) &= \lambda \int_0^x k(x,t) \left[f(t) + \lambda \int_0^t k(t,\tau) f(\tau) d\tau \right] dt \\ &\quad - \lambda \int_0^x k(x,t) f(t) dt \\ &= \lambda^2 \int_0^x k(x,t) \int_0^t k(t,\tau) f(\tau) d\tau \, dt \\ &= \lambda^2 \psi_2(x) \qquad (2.20) \end{aligned}$$

where

$$\psi_2(x) = \int_0^x k(x,t) \int_0^t k(t,\tau) f(\tau) d\tau \, dt \qquad (2.21)$$

Thus, it can easily be observed from equation (2.21) that

$$u_n(x) = \sum_{m=0}^{n} \lambda^m \psi_m(x) \qquad (2.22)$$

If $\psi_0(x) = f(x)$, and further that

$$\psi_m(x) = \int_0^x k(x,t)\psi_{m-1}(t)dt, \tag{2.23}$$

where $m = 1,2,3,\ldots$ and hence $\psi_1(x) = \int_0^x k(x,t)f(t)dt$.

The repeated integrals in equation (2.21) may be considered as a double integral over the triangular region; thus interchanging the order of integration, we obtain

$$\psi_2(x) = \int_0^x f(\tau)d\tau \int_\tau^x k(x,t)k(t,\tau)\,dt$$
$$= \int_0^x k_2(x,\tau)f(\tau)d\tau$$

where $k_2(x,\tau) = \int_\tau^x k(x,t)k(t,\tau)\,dt$. Similarly, we find in general

$$\psi_m(x) = \int_0^x k_m(x,\tau)f(\tau)d\tau\,, \quad m = 1,2,3,\ldots.. \tag{2.24}$$

where the iterative kernels $k_1(x,t) = k(x,t),\ k_2(x,t),\ k_3(x,t),$ .... are defined by the recurrence formula

$$k_{m+1}(x,t) = \int_t^x k(x,\tau)k_m(\tau,t)\,d\tau\,, \qquad m = 1,2,3,\ldots. \tag{2.25}$$

Thus, the solution for $u_n(x)$ can be written as

$$u_n(x) = f(x) + \sum_{m=1}^n \lambda^m \psi_m(x) \tag{2.26}$$

Upon using equation (2.24) we obtain

$$u_n(x) = f(x) + \sum_{m=1}^n \lambda^m \int_0^x k_m(x,\tau)f(\tau)d\tau$$
$$= f(x) + \int_0^x \{\sum_{m=1}^n \lambda^m k_m(x,\tau)\} f(\tau)d\tau, \tag{2.27}$$

Hence it is also clear that the solution of linear Volterra integral equation of the second kind will be given by

$$\lim_{n\to\infty} u_n(x) = u(x)$$
$$= f(x) + \int_0^x \{\sum_{m=1}^n \lambda^m k_m(x,\tau)\} f(\tau)d\tau,$$
$$= f(x) + \lambda \int_0^x H(x,\ \tau;\lambda)f(\tau)d\tau \tag{2.28}$$

where

$$H(x,\tau;\ \lambda) = \sum_{m=1}^n \lambda^m k_m(x,\tau) \tag{2.29}$$

is known as the resolvent kernel. (see [18],[19] and [28]) .

Example 2.4

Consider the Neumann series for the solution of the integral equation

$$u(x) = (1 + x) + \lambda \int_0^x (x - t)u(t)dt.$$

From the formula (2.25), we have

$$K_1(x,t) = (x - t),$$
$$K_2(x,t) = \int_t^x (x - \tau)(\tau - t)d\tau = \frac{(x-t)^3}{3!},$$
$$K_3(x,t) = \int_t^x \frac{(x-\tau)(\tau-t)^3}{3!}d\tau = \frac{(x-t)^5}{5!},$$

and so on .Thus,

$$\lim_{n\to\infty} u_n(x) = u(x)$$
$$= f(x) + \int_0^x \{\sum_{m=1}^n \lambda^m k_m(x,\tau)\} f(\tau)d\tau,$$
$$u(x) = f(x) + \int_0^x \lambda\, k(x,\tau)f(\tau)d\tau + \int_0^x \lambda^2\, k_2(x,\tau)f(\tau)d\tau +$$
$$\int_0^x \lambda^3\, k_3(x,\tau)f(\tau)d\tau + \cdots$$
$$u(x) = 1 + x + \lambda\left(\frac{x^2}{2!} + \frac{x^3}{3!}\right) + \lambda^2\left(\frac{x^4}{4!} + \frac{x^5}{5!}\right) + \cdots$$

for $\lambda = 1$, $u(x) = e^x$.

2.4 The series solution method

The series method is useful method that stems mainly from the Taylor series for analytic functions for solving integral equations.

Definition (2.1) A real function $u(x)$ is said to be analytic if it has derivatives of all orders such that the Taylor series at any point b in its domain

$$u(x) = \sum_{k=0}^n \frac{u^{(k)}(b)}{k!}(x - b)^k, \qquad (2.30)$$

converges to $u(x)$ in a neighborhood of b.

For simplicity, the generic form of Taylor series at $x = 0$ can be written as

$$u(x) = \sum_{n=0}^{\infty} a_n x^n. \tag{2.31}$$

we will assume that the solution $u(x)$ of the Volterra integral equation (1.6) is analytic, and therefore possesses a Taylor series of the form given in (2.31), where the coefficients a_n will be determined recurrently.

Substituting (2.31) into both sides of (1.6) gives

$$\sum_{n=0}^{\infty} a_n x^n = T\big(f(x)\big) + \lambda \int_0^x k(x,t)\left(\sum_{n=0}^{\infty} a_n t^n\right)dt\,, \tag{2.32}$$

or

$$a_0 + a_1 x^1 + a_2 x^2 + \cdots = T\big(f(x)\big) + \lambda \int_0^x k(x,t)(a_0 + a_1 t^1 + a_2 t^2 + \cdots)dt\,, \tag{2.33}$$

where $T\,(f(x))$ is the Taylor series for $f(x)$. the integral equation (2.32) will be converted to a traditional integral in (2.33) where instead of integrating the unknown function $u(x)$, the terms of the form $t^n, n \geq 0$ will be integrated. Notice that because we are seeking series solution, then if $f(x)$ includes elementary functions such as trigonometric functions, exponential functions, etc., then Taylor expansions for functions involved in $f(x)$ should be used. We will illustrate the series solution method by this example. (see [24], [28] and [39].

Example 2.5

Consider the solution of the Volterra integral equation of the second kind

$$u(x) = 1 + 2\sin x - \int_0^x u(t)dt\,,$$

using the series method. We assume the solution in the series form $(x) = \sum_{n=0}^{\infty} a_n x^n$. Hence substituting the series into the equation and the Taylor's series of $sin\ x$, we have

$$\sum_{n=0}^{\infty} a_n x^n = 1 + 2\sum_{n=0}^{\infty}(-1)^n \frac{x^{2n+1}}{(2n+1)!} - \int_0^x \sum_{n=0}^{\infty} a_n t^n dt$$

$$= 1 + 2\sum_{n=0}^{\infty}(-1)^n \frac{x^{2n+1}}{(2n+1)!} - \sum_{n=0}^{\infty} a_n \frac{x^{n+1}}{(n+1)}$$

Comparing the coefficients of the same power of x gives the following set of values:

$$a_0 = 1\,,$$
$$a_1 = 2 - a_0 = 1\,,$$
$$a_2 = -\frac{a_1}{2} = -\frac{1}{2}\,,$$
$$a_3 = -\frac{2}{3!} - \frac{a_2}{3} = -\frac{1}{3!}\,,$$
$$a_4 = -\frac{a_3}{4} = \frac{1}{4!}\,,$$

and so on. Hence the solution is given by

$$u(x) = \left(1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \cdots\right) + \left(x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots\right) = \cos x + \sin x\,.$$

2.5 Converting Volterra integral equation to ordinary differential equation

In this section we will present the technique that converts Volterra integral equations of the second kind to an equivalent differential equation. This may easily be achieved by applying the important Leibnitz Rule for differentiating an integral. It seems reasonable to review the basic outline of the rule.

To differentiate the integral

$$F(x) = \int_{v(x)}^{u(x)} f(x,t)dt. \tag{2.35}$$

with respect to x , we usually apply the useful Leibnitz rule given by :

$$F'(x) = \frac{dF}{dx} = f\big(x,u(x)\big)\frac{du(x)}{dx} - f\big(x,v(x)\big)\frac{dv(x)}{dx} + \int_{v(x)}^{u(x)} \frac{\partial f(x,t)}{\partial x}dt. \tag{2.36}$$

where $f(x,t)$ and $\frac{\partial f(x,t)}{\partial x}$ are continuous functions in the domain D in the xt-plane that contains the rectangular region R, $a \le x \le b\,, t_0 \le t \le t_1$,

and the limits of integration $v(x)$ and $u(x)$ are defined functions having continuous derivatives for $a \leq x \leq b$.

Thus the Leibnitz rule converts the Volterra integral equation or the Volterra integro-differential equations into an equivalent initial value problem. The initial conditions can be obtained by substituting $x = 0$ into $u(x)$ and its derivatives. The resulting initial value problem can be solved easily by using ODEs methods. The conversion process will be illustrated by the following example.

Example 2.6

We find the initial value problem equivalent to the Volterra integral equation of the second kind

$$u(x) = 1 - \cos(x) + 2\int_0^x (x-t)^2 u(t)dt. \qquad (2.37)$$

Differentiating both sides of (2.37) and using Leibnitz rule three times to get rid of the integral sign, we find

$$u'(x) = \sin(x) + 4\int_0^x (x-t)u(t)dt. \qquad (2.38)$$

$$u''(x) = \cos(x) + 4\int_0^x u(t)dt. \qquad (2.39)$$

$$u'''(x) = -\sin(x) + 4u(x)$$

To determine the initial conditions, we substitute $x = 0$ into both sides of (2.37), (2.38) and (2.39) to find $u(0) = 0$, $u'(0) = 0$ and $u''(0) = 1$.

This in turn gives the initial value problem

$$u'''(x) - 4u(x) = -\sin(x),$$

$$u(0) = 0\ ,\ u'(0) = 0 \text{ and } u''(0) = 1\ .$$

This resulting ODE is a third order inhomogeneous equation.

Chapter Three
Numerical Methods for Solving Volterra Integral Equation of the Second Kind

Chapter Three

Numerical Techniques for Solving Volterra Integral Equation of the Second Kind

There are many numerical techniques available for solving Volterra integral equation of the second kind. These techniques are based on the following methods: Quadrature methods (Trapezoidal rule, Runge-Kutta method of order two, the fourth order Runge-Kutta method), Blocks methods, the collocation method and the Galerkin method.

3.1 Quadrature methods for Volterra equation of the second kind

We consider the numerical solution of the Volterra integral equation of the second kind

$$u(x) = f(x) + \int_a^x k\big(x,t,u(t)\big)dt, \qquad a \le x \le b \qquad (3.1)$$

We assume that the solution is required over a finite interval $[a,b]$, that $f(x)$ is continuous in $[a,b]$, k is continuous in $a \le t \le x \le b$ and satisfies a uniform Lipschitz conditions in u. These conditions will ensure that a unique continuous solution to the problem (3.1) exists. If the kernel is linear in its third argument, that is, there exists a function k such that

$$k\big(x,t,u(t)\big) = k(x,t)u(t) + k_0(x,t) \qquad (3.2)$$

for all $a \le t \le x \le b$, then equation (3.1) is said to be linear and reduces to

$$u(x) = \overline{f(x)} + \int_a^x k(x,t)u(t)dt, \qquad a \le x \le b \qquad (3.3)$$

where

$$\overline{f(x)} = f(x) + \int_a^x k_0(x,t)u(t)dt, \qquad (3.4)$$

We shall take (3.3) as the canonical form for a linear Volterra equation and we will not distinguish notationally between $f(x)$ and $\overline{f(x)}$.

3.1.1 Quadrature methods for linear equations

An obvious numerical procedure is to approximate the integral term in (3.3) via a quadrature rule which integrates over the variable t for a fixed value of x. It is natural to choose a regular mesh in x and t ; thus setting $x = x_i = a + ih$, where $h = (b - a)/N$ is the fixed step length. We approximate in an obvious notation the integral term in the linear equation (3.3) by

$$\int_a^{x_i} k(x_i,t)u(t)dt \approx h\sum_{j=0}^{i} w_{ij}k(x_i,t_j)u(t_j) = h\sum_{j=0}^{i} w_{ij}k_{ij}u(t_j) \qquad (3.5)$$

where $x_i = t_i$, $i = 0,1,\dots,N$. This quadrature rule leads to the following set of equations:

$$\begin{aligned} u(x_0) &= f(x_0)\,, \\ u(x_1) &= f(x_1) + h[\, w_{10}k_{10}u(t_0) + w_{11}k_{11}u(t_1)\,] \\ &\quad + E_{1,t}\big(k(x_1,t)u(t)\big), \\ u(x_i) &= f(x_i) + h\sum_{j=0}^{i} w_{ij}k_{ij}u(t_j) \\ &\quad + E_{i,t}\big(k(x_i,t)u(t)\big), \quad i = 1,2,\dots,N\,, \end{aligned} \qquad (3.6)$$

where $E_{i,t}\big(k(x_i,t)u(t)\big)$ represents the error term in the quadrature rule. If the $E_{i,t}$ are assumed negligible and $(1 - hw_{ii}k_{ii}) \neq 0$ for any i we can clearly solve this set of equations for u_i , $i = 1,2,\dots,N$, where u_i is an approximation to $u(x_i)$, by direct forward substitution.

This procedure is obviously numerically very straightforward; however, there remains the problem of choosing suitable weights w_{ij}. We note that,

for each i, the set $\{w_{ij}, j = 0,1, \dots, i\}$ represents the weights for an $(i+1)$ point quadrature rule of Newton-Cotes type (equally spaced points) for the interval $[0, ih]$. For large i there are many possible choices of rule, for small $i = l, 2, \dots$, the choice is rather limited, yet there seems (and is) little point in choosing an accurate rule for large i if we cannot choose an equally accurate rule for small i. Let us start by considering the simplest possible rule, the repeated (Trapezoid rule). (see [4] and [13]).

3.1.2 Trapezoidal rule

Let $a < b \in \mathbb{R}$. We divide the interval (a, b) into subintervals with equal length $h = \frac{b-a}{N}$. We denote $x_i = a + (i-1)h$, $1 \le i \le N+1$, then the Trapezoidal method reads :

$$\int_a^b f(x)dx = h\left[\frac{f(a)+f(b)}{2} + \sum_{i=2}^{N-1} f(x_i)\right] \qquad (3.7)$$

Using the Trapezoidal approximation to solve the Volterra integral equation:

$$u(x) - \lambda \int_a^x k(x,t)u(t)dt = f(x) \qquad (3.8)$$

We substitute (3.7) into (3.8) with $= x_i$, we get

$$u(x_i) - h\left[\frac{k(x_i,\mathrm{a})\mathrm{u}(\mathrm{a})+\mathrm{k}(x_i,x_i)\mathrm{u}(x_i)}{2} + \sum_{j=2}^{i-1} k(x_i, x_j)\mathrm{u}(x_j)\right] = f(x_i) \qquad (3.9)$$

$1 \le i \le N+1,\ x_1 = a, x_2, \dots, x_{N+1} = b$

$$-h\frac{k(x_i,\mathrm{a})}{2}u(a) - h\sum_{j=2}^{i-1} k(x_i, x_j)\mathrm{u}(x_j) + \left(1 - h\frac{\mathrm{k}(x_i,x_i)}{2}\right)\mathrm{u}(x_i) = f(x_i)$$

For $i = 1, x_1 = a$, the Volterra integral equation (3.8) is reduced to

$$u(a) = f(a)$$

For $i = 2$, we get

$$-h\frac{k(x_2,x_1)}{2}u(x_1) + \left(1 - h\frac{\mathrm{k}(x_2,x_2)}{2}\right)\mathrm{u}(x_2) = f(x_2)$$

For $i = 3$, we obtain

$$-h\frac{k(x_3,x_1)}{2}u(x_1) - h\mathrm{k}(x_3\,,x_2\,)u(x_2) + \left(1 - h\frac{\mathrm{k}(x_3\,,x_3\,)}{2}\right)\mathrm{u}(x_3\,) \;\; = f(x_3)$$

To this end, we obtain the linear system

$$A\bar{u} = B$$

where the matrix $A = (a_{ij})$, $1 \le i, j \le N+1$ with:

$$\begin{cases} a_{ij} = 0, & \forall\ j \le i+1 \\ a_{ij} = -h\mathrm{K}\big(x_i\,, x_j\,\big), & 2 \le j \le i \le n+1 \\ a_{ii} = 1 - \dfrac{h}{2}\mathrm{K}(x_i\,, x_i\,) & \\ a_{11} = 1 & \\ a_{i1} = -\dfrac{h}{2}\mathrm{K}(x_i\,, x_1\,), & 1 \le i \le n+1 \end{cases}$$

$$A = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ a_{21} & a_{22} & 0 \ \ldots & 0 \\ a_{31} & a_{32} & a_{33} \ldots & 0 \\ \vdots & & \ddots & \vdots \\ a_{N+1,1} & a_{N+1,2} & \ldots\ldots & a_{N+1,N+1} \end{bmatrix}$$

$B = [f(x_1) = f(a), f(x_2), \ldots, f(x_{N+1}) = f(b)]^T$,

$\bar{u} = [u(a), u(x_2), \ldots, u(x_{N+1})]^T$.

(See [2], [3],[20],[25] and [26]).

3.1.3 Runge-Kutta methods

Runge-Kutta methods for the solution of (3.1) are self-starting methods which determine approximations to the solution at the points $x_i = a + ih\,, i = 1,2, \ldots, N$, by generating approximations at some intermediate points in $[x_i, x_{i+1}]\,, i = 1,2, \ldots, N$:

$$x_i + \theta_r h\,, i = 1,2, \ldots, N-1\,, r = 1,2, \ldots, p-1\,,$$

where

$$0 = \theta_0 \le \theta_1 \le \cdots \le \theta_{p-1} \le 1\,. \qquad (3.10)$$

We recall the general p-stage Runge-Kutta method for the initial value Problem $u'(x) = g\big(x, u(x)\big)$,

$$u(a) = u_0\,, \tag{3.11}$$

given by

$$u_{i+1} = u_i + h\sum_{i=0}^{p-1} A_{pi}{k_i}^i \tag{3.12}$$

where

$${k_0}^i = g(a + ih, u_i)$$

$${k_r}^i = g\big(a + (i+\theta_r)h, u_i + h\sum_{i=0}^{r-1} A_{ri}{k_i}^i\big), r = 1,2,\dots,p-1. \tag{3.13}$$

$$\sum_{i=0}^{r-1} A_{ri} = \begin{cases}\theta_r\,, & r = 1,2,\dots,p-1,\\ 1\,, & r = p\,,\end{cases} \tag{3.14}$$

with u_r is an approximation to the solution at $x = x_r = a + rh$. The second argument of ${k_r}^i$ may be regarded as an approximation to $u(a + (i+\theta_r)h)$ and we rewrite equation (3.12) as

$$u_{i+1} = u_i + h\sum_{i=0}^{p-1} A_{pi}g\big(x_i + \theta_i h, u_{i+\theta_i}\big). \tag{3.15}$$

The parameters A_{pi}, θ_i 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.

Example 3.1

Suppose we choose $p = 2$ in (3.15) .Then it follows that

$$u_{i+1} = u_i + hA_{20}g(x_i, u_i) + hA_{21}g\big(x_i + \theta_1 h, u_i + hA_{10}g(x_i, u_i)\big).$$

We use Taylor's theorem for a function of two variables to obtain

$$u_{i+1} = u_i + h(A_{20} + A_{21})g + h^2A_{21}(\theta_1 g_x + A_{10}gg_u) + O(h^3).$$

where we have introduced the notations

$$g = g(x_i, u_i)\,,\quad g_x = \frac{\partial g(x_i,u_i)}{\partial x}\,,\ g_u = \frac{\partial g(x_i,u_i)}{\partial u}.$$

Now if we compare this expression term by term with

$$u_{i+1} = u_i + hg + \frac{1}{2}h^2(g_x + gg_u) + \varphi(h^3).$$

then we have the following set of three equations

$$A_{20} + A_{21} = 1,$$
$$A_{21}\theta_1 = \frac{1}{2},$$
$$A_{21}A_{10} = \frac{1}{2},$$

Clearly there exist an infinite number of solutions of these equations corresponding to an infinite number of two-stage Runge-Kutta method of order two. We consider two particular solutions which are popular in practice:

(i) When $A_{20} = A_{21} = \frac{1}{2}$ the resulting method is

$$u_{i+1} = u_i + \frac{1}{2}\, h[g(x_i, u_i) + g\big(x_{i+1}, u_i + hg(x_i, u_i)\big)] . \qquad (3.16)$$

or

$$u_{i+1} = u_i + \frac{1}{2}\, h[g(x_i, u_i) + g(x_{i+1}, \hat{u}_{i+1})] . \qquad (3.17)$$

where

$$\hat{u}_{i+1} = u_i + hg(x_i, u_i). \qquad (3.18)$$

This is the improved Euler method.

(ii) when $A_{20} = 0\,, A_{21} = 1$ the resulting method is the modified Euler method given by

$$u_{i+1} = u_i + hg\left(x_i + \frac{1}{2}h, u_i + \frac{1}{2}hg(x_i, u_i)\right). \qquad (3.19)$$

when $p = q = 4$, we obtain in a similar way the classical fourth order Runge Kutta method given by the following choice of parameters:

$$\theta_0 = 0\,,\ \theta_1 = \theta_2 = \frac{1}{2}\,,\ \theta_3 = 1\,,$$
$$A_{10} = \frac{1}{2}\,,\ A_{20} = 0,\ A_{21} = \frac{1}{2}\,,$$
$$A_{30} = A_{31} = 0\,,\quad A_{32} = 1\,,$$
$$A_{40} = A_{43} = \frac{1}{6}\,,\quad A_{41} = A_{42} = \frac{1}{3}\,.$$

The method defined in equation (3.15) can be extended to give a class of Runge-Kutta method for the solution of

$$u(x) = f(x) + \int_a^x k\big(x, t, u(t)\big)dt, \qquad a \le x \le b \qquad (3.20)$$

Setting $x = x_i$ in (3.20) we have

$$u(x_i) = f(x_i) + \int_a^{a+ih} k\big(a + ih, t, u(t)\big)dt, \qquad i = 1, \dots, N,$$
$$= f(x_i) + \sum_{j=0}^{i-1} \int_{a+jh}^{a+(j+1)h} k\big(a + ih, t, u(t)\big)dt, \quad i = 1, \dots, N, \qquad (3.21)$$

and we can determine an approximation x_i to $u(x_i)$ from the following equation

$$u(x_i) = f(x_i) + h \sum_{j=0}^{i-1} \sum_{i=0}^{p-1} A_{pi} k(a + ih, a + (j + \theta_i)h, u_{j+\theta_i}) \qquad (3.22)$$

Now for $x\epsilon(x_i, x_{i+1})$ we may write equation (3.20) in the following form

$$u(x) = f(x) + \sum_{j=0}^{i-1} \int_{x_j}^{x_{j+1}} k\big(x, t, u(t)\big)dt + \int_{x_j}^{x} k\big(x, t, u(t)\big)dt \qquad (3.23)$$

Then setting $x = x_i + \theta_v h$, $v = 1,2, \dots, p-1$, and approximating the final integral term in (3.23) by

$$\int_{x_i}^{x_i+\theta_v h} k\big(x_i + \theta_v h, t, u(t)\big)dt \approx$$
$$h \sum_{i=0}^{v-1} A_{vi} k(x_i + \theta_v h,\ x_i + \theta_i h, u_{i+\theta_i})$$

we see that the Runge-Kutta method for (3.20) may be expressed as

$$u_{i+\theta_i} = f(x_i + \theta_v h) + h \sum_{j=0}^{i-1} \sum_{i=0}^{p-1} A_{pi} k(x_i + \theta_v h, x_j + \theta_i h, u_{j+\theta_i})$$
$$+ h \sum_{i=0}^{v-1} A_{vi} k(x_i + \theta_v h,\ x_i + \theta_i h, u_{i+\theta_i}),$$
$$i = 0,1, \dots, N-1, \qquad v = 1,2, \dots, p-1, \qquad (3.24)$$

where $u(a) = f(a)$ and the parameters $A_{rj},\ \theta_j, r = 1,2, \dots, p$,

$j = 0,1, \dots, p-1$, define the particular method. (see [13]).

3.2 The Block Methods

A Block method is essentially an extrapolation procedure which has advantage of being self-starting and produces a block of values at a time.

They give up any attempt to solve the problem by marching one step at a time and instead introduce a rule over a small region which uses points over a larger region. Consider the solution of (3.3) in the range $a < x < b$ with $b - a = Nph$; that is, we divide the interval $[a, b]$ into n equal intervals, each of which is then divided into p subintervals of length h. Now assume that approximate solution values have been calculated for the first $(r - 1)$ blocks; then a typical block method produces at the r^{th} stage the following set of approximations:

$$u_{p(r-1)+1}, u_{p(r-1)+2}, \dots, u_{pr}.$$

For $p(r-1) < i < pr$, $r = 1,2, \dots, N$, we may rewrite (3.3) in the form

$$u(a+ih) = f(a+ih) + \int_a^{a+p(r-1)h} k(a+ih, t)u(t)dt + \int_{a+p(r-1)h}^{a+ih} k(a+ih, t)u(t)dt \quad (3.25)$$

Using the following quadrature rules to approximate the integral terms in (3.25)

$$\int_a^{a+p(r-1)h} k(a+ih, t)u(t)dt \approx h\sum_{j=0}^{p(r-1)} w_{ij}\, k(a+ih, a+jh)u(a+jh) \quad (3.26)$$

$$\int_{a+p(r-1)h}^{a+ih} k(a+ih, t)u(t)dt \approx h\sum_{j=p(r-1)}^{pr} \bar{w}_{ij}\, k(a+ih, a+jh)u(a+jh) \quad (3.27)$$

we obtain the set of approximating equations

$$\begin{aligned} u_0 &= f(a) \\ u_i &= f(a+ih) + h\sum_{j=0}^{p(r-1)} w_{ij}\, k(a+ih, a+jh)u(a+jh) \\ &\quad + h\sum_{j=p(r-1)}^{pr} \bar{w}_{ij}\, k(a+ih, a+jh)u(a+jh) \end{aligned} \quad (3.28)$$

where $i = p(r-1)+1, p(r-1)+2, \dots, pr$, and $r = 1,2, \dots, n$.

Linze [20], described two block methods and uses these methods to solve Volterra integral equation of the second kind. In this work this method has been used to solve Volterra integral equations of the second kind, in which a block of two and three values are produced at each stage and the values of the involved integrals are obtained using the quadrature formula.

3.2.1 Method of two Blocks:

Applying equation (3.3) with $x = x_{2n+1} = x_0 + (2n+1)h$,

and $x = x_{2n+2} = x_0 + (2n+2)h$, where $x_0 = a$, to get:

$$u_{2n+1} = f_{2n+1} + \int_{x_0}^{x_{2n}} k(x_{2n+1}, t)u(t)dt + \int_{x_{2n}}^{x_{2n+1}} k(x_{2n+1}, t)u(t)dt \quad (3.29)$$

$$u_{2n+2} = f_{2n+2} + \int_{x_0}^{x_{2n}} k(x_{2n+2}, t)u(t)dt + \int_{x_{2n}}^{x_{2n+2}} k(x_{2n+2}, t)u(t)dt \quad (3.30)$$

This technique depends on the use of a quadrature formula. This is

Simpson's 1/3 rule [20]

$$\int_{x_0}^{x_1} f(x)dx = \frac{h}{2}[5f_0 + 8f_1 - f_2] \quad (3.31)$$

with $x_0 = x_{2n}$and $x_1 = x_{2n+1}$ where $n \geq 0$ .therefore we obtain:

$$u_{2n+1} = f_{2n+1} + \frac{h}{3}\sum_{j=0}^{2n}[w_j\, k(x_{2n+1}, x_j)u(x_j)] + \frac{h}{12}[5k(x_{2n+1}, x_{2n})u(x_{2n}) + 8k(x_{2n+1}, x_{2n+1})u(x_{2n+1}) - k(x_{2n+1}, x_{2n+2})u(x_{2n+2})] \quad (3.32)$$

$$u_{2n+2} = f_{2n+2} + \frac{h}{3}\sum_{j=0}^{2n+2}[\bar{w}_j\, k(x_{2n+2}, x_j)u(x_j)] \quad (3.33)$$

Where $w_0 = w_{2n} = 1$, $w_j = 3 - (-1)^j$, $1 \leq j \leq 2n-1$

and $\bar{w}_0 = \bar{w}_{2n} = 1$, $\bar{w}_j = 3 - (-1)^j$, $1 \leq j \leq 2n+1$

Thus we have a pair of equations to solve for u_{2n+1} and u_{2n+2}.

3.2.2 Method of Three Blocks:

Applying equation (3.3) with $x = x_{3n+1} = x_0 + (3n+1)h$,

$$x = x_{3n+2} = x_0 + (3n+2)h \ \ and \ x = x_{3n+3} = x_0 + (3n+3)h$$

Where $x_0 = a$, to get:

$$u_{3n+1} = f_{3n+1} + \int_{x_0}^{x_{3n}} k(x_{3n+1}, t)u(t)dt + \int_{x_{3n}}^{x_{3n+1}} k(x_{3n+1}, t)u(t)dt \quad (3.34)$$

$$u_{3n+2} = f_{3n+2} + \int_{x_0}^{x_{3n}} k(x_{3n+2}, t)u(t)dt + \int_{x_{3n}}^{x_{3n+2}} k(x_{3n+2}, t)u(t)dt \quad (3.35)$$

$$u_{3n+3} = f_{3n+3} + \int_{x_0}^{x_{3n}} k(x_{3n+3}, t)u(t)dt + \int_{x_{3n}}^{x_{3n+3}} k(x_{3n+3}, t)u(t)dt \quad (3.36)$$

This technique depends on the use of three quadrature formulas. These are Simpson's 3/8 rule and Simpson's 1/3 rule. Therefore:

$$u_{3n+1} = f_{3n+1} + \frac{3h}{8}\sum_{j=0}^{3n}[w_j\, k(x_{3n+1}, x_j)u(x_j)] + \frac{h}{12}[5k(x_{3n+1}, x_{3n})u(x_{3n}) + 8k(x_{3n+1}, x_{3n+1})u(x_{3n+1}) - k(x_{3n+1}, x_{3n+2})u(x_{3n+2})] \quad (3.37)$$

$$u_{3n+2} = f_{2n+2} + \frac{3h}{8}\sum_{j=0}^{3n}[\overline{w}_j\, k(x_{3n+2}, x_j)u(x_j)] + \frac{h}{3}[k(x_{3n+2}, x_{3n})u(x_{3n}) + 4k(x_{3n+2}, x_{3n+1})u(x_{3n+1}) + k(x_{3n+2}, x_{3n+2})u(x_{3n+2})] \quad (3.38)$$

$$u_{3n+3} = f_{2n+3} + \frac{3h}{8}\sum_{j=0}^{3n+3}[\overline{\overline{w}}_j\, k(x_{3n+3}, x_j)u(x_j)] \quad (3.39)$$

where

$$w_0 = w_{3n} = 1 \ \ , w_j = \begin{cases} 2 & , if \ \frac{j}{3} \ integer \\ 3 & , otherwise \end{cases}$$

$$\overline{w}_0 = \overline{w}_{3n} = 1 \ \ , \overline{w}_j = \begin{cases} 2 & , if \ \frac{j}{3} \ integer \\ 3 & , otherwise \end{cases}$$

and

$$\bar{\bar{w}}_0 = \bar{\bar{w}}_{3n+3} = 1 \quad , \bar{\bar{w}}_j = \begin{cases} 2 & , if \ \frac{j}{3} \ integer \\ 3 & , otherwise \end{cases}$$

Thus, we have a system of three equations to solve for u_{3n+1} , u_{3n+2} and u_{3n+3} . (see[11], [13] and [27]).

3.3 The Collocation method

3.3.1 Meshes and piecewise polynomial spaces:

We wish to solve the Volterra integral equation (3.3) on the interval $I \coloneqq [0,T]$. Let

$I_h \coloneqq \{ x_n : 0 = x_0 < x_1 < x_2 < \cdots x_N = T \}$ be a mesh, and define

$e_n \coloneqq (x_n, x_{n+1}]$, $h_n \coloneqq x_{n+1} - x_n$ $(0 \le n \le N-1)$, and

$h \coloneqq \max\{ h_n : 0 \le n \le N-1 \}$ (mesh diameter).

Remark: Different types of meshes on $I \coloneqq [0,T]$

$$I_h \coloneqq \{ x_n : 0 = x_0 < x_1 < x_2 < \cdots x_N = T \} \ (N \in \mathbb{N}).$$

- Quasi-uniform mesh I_h : there exists a constant $\gamma < \infty$ (independent of N) so that

$$\frac{max_{(n)} h_n}{min_{(n)} h_n} \le \gamma \qquad \text{for all } N \ge 1 . \quad (\rightarrow Nh \le \gamma T)$$

- Graded mesh I_h :

$x_n = \left(\frac{n}{N}\right)^r T \quad (n = 0,1,2, \dots, N)$, with grading exponent $r > 1$.

If $r = 1$ then the mesh I_h is a uniform mesh.

- Geometric mesh I_h:

$$x_n = q^{N-n} T \quad (n = 0,1,2, \dots, N).$$

Where $q \in (0,1)$.

Definition 3.3.1: For a given mesh I_h the piecewise polynomial space $S_r^{(d)}(I_h)$, with $r \ge 0$, $-1 \le d < r$ is given by

$$S_r^{(d)}(I_h) \coloneqq \{v \in C^d(I) : v|_{e_n} \in P_r \ \ (0 \le n \le N-1)\}. \tag{3.40}$$

Here, P_r denotes the space of (real) polynomials of degree not exceeding r. It is readily verified that $S_r^{(d)}(I_h)$ is a (real) linear vector space whose dimension is given by

$$dim\, S_r^{(d)}(I_h) = N(r-d) + d + 1.$$

If $r = m + d$ with $m \ge 1$ and $d \ge -1$, then the piecewise polynomial space is $S_{m+d}^{(d)}(I_h)$ and the dimension of this linear space is given by

$$dim\, S_{m+d}^{(d)}(I_h) = Nm + d + 1.$$

For Volterra integral equation of the second kind we choose $d = -1$, hence, the natural collocation space will be $S_{m-1}^{(-1)}(I_h)$. Its dimension is given by

$$dim\, S_{m-1}^{(-1)}(I_h) = Nm. \tag{3.41}$$

To find: 'good' approximation $u_h(x)$ to the solution $u(x)$ of (3.3) so that

- $u_h(x)$ is definded for all $x \in I$;
- $u_h(x)$ can be easily computed on non-uniform meshes I_h;
- The approximation error satisfies

$$\max\{\, |u(x) - u_h(x)| : x \in I \,\} \le Ch^p$$

where p (the order of the numerical method) is as large as possible. We will use piecewise polynomial collocation methods in $S_{m-1}^{(-1)}(I_h)$.

3.3.2 Piecewise polynomial collocation methods in $S_{m-1}^{(-1)}(I_h)$

Let the linear Volterra integral operator $V: C(I) \to C(I)$ be given by

$$(Vu)(x) \coloneqq \int_0^x k(x,t)u(t)dt, \quad x \in I \coloneqq [0,T], \tag{3.42}$$

where $k \in C(D)$ ($D \coloneqq \{(x,t): 0 \le t \le x \le T\}$), and let $f \in C(I)$ be a given function. The solution of the Volterra integral equation

$$u(x) = f(x) + (Vu)(x), \qquad x \in I, \tag{3.43}$$

will be approximated by collocation in the piecewise polynomial space

$$S_{m-1}^{(-1)}(I_h) := \{v : v|_{e_n} \in P_{m-1} \ (0 \le n \le N-1)\}. \tag{3.44}$$

where $P_{m-1} = P_{m-1}(e_n)$ is the set of (real) polynomials on $e_n = (x_n, x_{n+1}]$, of degree $\le m-1$

$S_{m-1}^{(-1)}(I_h)$ is called the space of piecewise polynomials of degree less than or equal to $m-1$.

- If $m = 1$ then $S_0^{(-1)}(I_h)$ is piecewise constant functions.

  (such a function contains N unknown coefficients).

- If $m = 2$ then $S_1^{(-1)}(I_h)$ is piecewise linear functions.

  (such a function contains $2N$ unknown coefficients).

In general: By (3.41) an element $u_h(x) \in S_{m-1}^{(-1)}(I_h)$ contains Nm unknown coefficients. We choose Nm distinct points in the interval $[0, T]$ to determine these coefficients at which the approximate solution $u_h(x)$ must satisfy the given Volterra integral equation. These points are called the collocation points.

3.3.3 Collocation points and collocation equation

Let $0 < c_1 < \cdots < c_m \le 1$ be given numbers (collocation parameters). The set

$$X_h := \{x_n + c_i h_n \ : i = 1,2,\dots,m \ (0 \le n \le N-1)\}$$

is called the set of collocation points. In each subinterval $(x_n, x_{n+1}]$, there are m such points, and so we have $|X_h| = Nm$.

Consider $u_h(x) \in S_{m-1}^{(-1)}(I_h)$ so that it satisfies the given Volterra integral equation at the points X_h :

$$u_h(x) = f(x) + \int_0^x k(x,t)\, u_h(t)dt\,, \qquad x \in X_h. \qquad (3.45)$$

This function $u_h(x)$ is called the collocation solution for the Volterra integral equation (3.3).

(see[7], [33] and [34]).

3.4 The Galerkin Method

Definition (3.1) L^p-space:

The set of L^p- functions (where ≥ 1) generalizes L^2-space. Instead of square integrable, the measurable function f must be p-integrable, for f to be in L^p.

On a measure space X, the L^p norm of a function f is

$$\|f\|_{L^p} = \left(\int_X |f(x)|^p\, dx\right)^{\frac{1}{p}}$$

The L^p-functions are the functions for which this integral converges.

For $p = 2$, the space of L^p-functions is a Hilbert space. For $p \neq 2$, the space of L^p-functions is a Banach space.

In the case where $p = \infty$, we have $L^\infty(D)$ defined as

$$\{f : \text{measurable in } D \text{ and } \|f\|_\infty < \infty\},$$

where

$$\|f\|_\infty = \inf\{\sup\{|f(x)| : x \epsilon S\}, S \subset D\}$$

with Lebesgue measure of the set S equals zero.

Let $X = L^2(I)$ or some other Hilbert function space, and let $\langle .,. \rangle$ denote the inner product for X. Require the residual r_n to satisfy

$$\langle r_n, \emptyset_i \rangle = 0, \qquad i = 1,2, \dots, d_n \qquad (3.46)$$

The left side is the Fourier coefficient of r_n associated with $\emptyset_i$.If $\{\emptyset_1, \dots, \emptyset_d, \dots\}$ consists of the leading members of an orthonormal family

$\emptyset \equiv \{\emptyset_1, \dots, \emptyset_d, \dots\}$ which spans X, then (3.46) requires the leading terms to be zero in the Fourier expansion of r_n with respect to $\emptyset$.

To find u_n, apply (3.46) to (3.3) written as $(\lambda - k)u = f$.

This yields the linear system

$$\sum_{j=1}^{d_n} c_j\{\langle \emptyset_j, \emptyset_i\rangle - \lambda\langle k\emptyset_j, \emptyset_i\rangle\} = \langle f, \emptyset_i\rangle, \qquad i = 1, \dots, d_n \tag{3.47}$$

This is Galerkin's method for obtaining an approximate solution to (3.3).

Note that the above formulation contains double integrals $\langle k\emptyset_j, \emptyset_i\rangle$. These must often be computed numerically.

As a part of writing (3.47) in a more abstract form, we introduce a projection operator $P_n$that maps X onto X_n. For general $x \in X$, define $P_n x$ to be the solution of the following minimization problem:

$$\|x - P_n x\| = \min_{z \in X_n} \|x - z\| \tag{3.48}$$

Since X_n is finite dimensional, it can be shown that this problem has a solution; and by X_n being an inner product space, the solution can be shown to be unique. To obtain a better understanding of P_n, we give an explicit formula for $P_n x$.

Introduce a new basis $\{\psi_1, \dots, \psi_d\}$ for X_n by using the Gram-Schmidt process to create an orthonormal basis from $\{\emptyset_1, \dots, \emptyset_d\}$. The element ψ_i is a linear combination of $\{\emptyset_1, \dots, \emptyset_d\}$, and moreover

$$\langle \psi_i, \psi_j\rangle = \delta_{ij}\,, \qquad i, j = 1, \dots, d_n$$

With this new basis, it is straightforward to show that

$$P_n x = \sum_{i=1}^{d_n} \langle x, \psi_i\rangle\, \psi_i \tag{3.49}$$

This shows immediately that P_n is a linear operator.

With this formula, we can show the following results.

$$\|x\|^2 = \|\, P_n x\|^2 + \|x - \, P_n x\|^2 \tag{3.50}$$

$$\| P_n x\|^2 = \sum_{i=1}^{d_n} |\langle x, \psi_i \rangle|^2$$

$$\langle P_n x, y\rangle = \langle x, P_n y\rangle, \qquad x, y \in X \tag{3.51}$$

$$\langle\langle 1 - P_n\rangle x,\ P_n y\rangle = 0, \qquad x, y \in X \tag{3.52}$$

Because of the latter, $P_n x$ is called the orthogonal projection of x onto X_n. The operator P_n is called an orthogonal projection operator. The result (3.50) leads to

$$\| P_n\| = 1 \tag{3.53}$$

Using (3.52), we can show

$$\|x - z\|^2 = \| x - P_n x\|^2 + \| P_n x - z\|^2 , \qquad z \in X_n \tag{3.54}$$

This shows $P_n x$ is the unique solution to (3.48).

We note that

$$P_n z = 0 \text{ if and only if } \langle z, \emptyset_i\rangle = 0, \qquad i = 1, \ldots d_n \tag{3.55}$$

Using the orthogonal projection P_n , we can write as

$$P_n\, r_n = 0$$

or equivalently,

$$P_n(I - \lambda k)\, u_n = P_n f , \qquad u_n \in X_n \tag{3.56}$$

However, in this work, we provide a numerical approach for the Volterra integral equation based on Chebyshev piecewise polynomials basis by the technique of Galerkin. Firstly, we give an introduction of Chebyshev piecewise polynomials. Then, we drive a matrix formulation for general linear problems by the technique of Galerkin method. (See [7]).

3.4.1 Chebyshev polynomials

The Chebyshev polynomials, named after Pafnuty Chebyshev, are a sequence of orthogonal polynomials which are related to de Moivre's formula and which can be defined recursively. The general form of the Chebyshev polynomials of nth degree is defined by

$$T_n(x) = \sum_{m=0}^{[n/2]} (-1)^m \frac{n!}{(2m)!(n-2m)!} (1-x^2)^m x^{n-2m} \quad (3.57)$$

where, $[n/2] = \begin{cases} \frac{n}{2} & if\ n\ is\ even \\ \frac{n+1}{2} & if\ n\ is\ odd \end{cases}$

The first few Chebyshev polynomials are given as:

$T_0(x) = 1\ , T_1(x) = x$

$T_2(x) = 2x^2 - 1\ ,$

$T_3(x) = 4x^3 - 3x\ ,$

$T_4(x) = 8x^4 - 8x^2 + 1$

$T_5(x) = 16x^5 - 20x^3 + 5x\ , T_6(x) = 32x^6 - 48x^4 + 18x^2 - 1$

3.4.2 Formulation of Integral Equation in Matrix Form

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

$$u(x) - \lambda \int_a^x K(x,t)u(t)dt = f(x)\ , \qquad a \le x \le b \quad (3.58)$$

Now we use the technique of Galerkin method [3], to find an approximate solution $u(x)$ of (3.58). For this, we assume that

$$\overline{u}(x) = \sum_{i=0}^{n} a_i N_i(x) \quad (3.59)$$

where $N_i(x)$ are Chebyshev polynomials of degree i defined in equation (3.57) and a_i are unknown parameters, to be determined. Substituting (3.59) into (3.58), we get

$$\sum_{i=0}^{n} a_i \left[N_i(x) - \lambda \int_a^x k(x,t)N_i(t)dt\right] = f(x), \quad a \le x \le b \quad (3.60)$$

Then the Galerkin equations are obtained by multiplying both sides of (3.60) by $N_j(x)$ and then integrating with respect to x from $a\ to\ b$. We obtain

$$\sum_{i=0}^{n} a_i \int_a^b \left[N_i(x) - \lambda \int_a^x k(x,t)N_i(t)dt\right] N_j(x)dx = \int_a^b f(x)N_j(x)dx\ , \quad j = 0,1,2,\dots,n \quad (3.61)$$

The inner integrand of the left side is a function of x, and t, and is integrated with respect to t from $a\ to\ x$. As a result the outer integrand becomes a function of x only and integration with respect to x from $a\ to\ b$ yields a constant.

Thus for each, $j = 0,1,2,\dots,n$ we have a linear equation with $n+1$ unknowns a_i, $i = 0,1,2,\dots,n$. Finally (3.61) represents the system of $n+1$ linear equations in $n+1$ unknowns, a given by

$$\sum_{i=0}^{n} a_i k_{i,j} = F_j, \quad i,j = 0,1,2,\dots,n \tag{3.62}$$

where

$$k_{i,j} = \int_a^b \left[N_i(x) - \lambda \int_a^x k(x,t) N_i(t) dt\right] N_j(x) dx\,, \quad i,j = 0,1,2,\dots,n$$

$$F_j = \int_a^b f(x) N_j(x) dx\,, \qquad j = 0,1,2,\dots,n$$

Now the unknown parameters a_i are determined by solving the system of equations (3.62) and substituting these values of parameters in (3.59). We get the approximate solution $\overline{u}(x)$ of the integral equation (3.3).

The maximum absolute error for this formulation is defined by

$$\text{Maximum absolute error} = Max\ |u(x) - \overline{u}(x)|.$$

(see [14], [29], [30], [36] and [41]).

Chapter Four
Numerical Examples and Results

Chapter Four

Numerical Examples and Results

To test the efficiency of the numerical methods represented in chapter three we will consider the following numerical examples.

Example 4.1

Consider the Volterra integral equation of the second kind

$$u(x) = 2e^x - x - 2 + \int_0^x (x-t)u(t)dt. \qquad (4.1)$$

Equation (4.1) has the exact solution

$$u(x) = xe^x.$$

We will find an approximate solution to equation (4.1) by the following numerical methods:

4.1 The numerical realization of equation (4.1) using Trapezoidal rule

The following algorithm implements the Trapezoidal rule using the Matlab software.

Algorithm 4.1

1. Input n : The number of subdivisions of $[a, b]$

   a, b : $[a, b]$ is the interval for the solution function

   fcn_f : The handle of the driver function $f(x)$

   and fcn_k : The handle of the kernel function $k(x, t)$

2. $loop = 10$ This is much more than is usually needed.
3. Calculate $h = (b - a)/n$
4. Calculate $x = linspace(a, b, n + 1)$
5. Calculate $f\_vec = fcn\_f(x)$

6. Set $u\_vec = zeros(size(x))$
7. Set $u\_vec(1) = f\_vec(1)$
8. $for\ i = 1{:}n$

   $u\_vec(i+1) = u\_vec(i)$, The initial estimate for the iteration.

   $k\_vec = fcn\_k(x(i+1), x(1{:}n+1)) * u\_vec(1{:}i+1)$

   $for\ j = 1{:}loop$

   Applying trapezoid rule

$$u\_vec(i+1) = f\_vec(i+1) + h * (sum(k\_vec(2{:}i)) + \ldots + (k\_vec(1) + k\_vec(i+1))/2)$$

$$k\_vec(i+1) = fcn\_k(x(i+1), x(i+1)) * u\_vec(i+1)$$

end

end

9. Set $u = u\_vec$
10. Output: the numerical solution $u(x)$, and the grid points x at which the solution $u(x)$ is approximated.

Thus we can solve the Volterra integral equation of the second kind (4.1) by using algorithm 4.1. Table 4.1 shows the exact and numerical results when n=20, and showing the error resulting of using the numerical solution.

Table 4.1: The exact and numerical solutions for Algorithm 4.1

x	Analytical solution $u(x) = xe^x$	Approximate solution $u_h(x)$	Error= $\lvert u - u_h \rvert$
0	0	0	0
0.05	0.052563555	0.052563554	0.02136×10^{-3}
0.1	0.110517092	0.110517091	0.04390×10^{-3}
0.15	0.174275136	0.174275136	0.06775×10^{-3}
0.2	0.244280552	0.244280551	0.09308×10^{-3}
0.25	0.321006354	0.321006354	0.12004×10^{-3}
0.3	0.404957642	0.404957642	0.14880×10^{-3}
0.35	0.496673642	0.496673642	0.17956×10^{-3}
0.4	0.596729879	0.596729879	0.212503×10^{-3}
0.45	0.705740483	0.705740483	0.247840×10^{-3}
0.5	0.824360635	0.824360635	0.285798×10^{-3}
0.55	0.95328916	0.953289159	0.326617×10^{-3}
0.6	1.09327128	1.093271280	0.370555×10^{-3}
0.65	1.245101539	1.245101538	0.417888×10^{-3}
0.7	1.409626895	1.409626895	0.468909×10^{-3}
0.75	1.587750012	1.587750012	0.523935×10^{-3}
0.8	1.780432743	1.780432742	0.583304×10^{-3}
0.85	1.988699824	1.988699824	0.647376×10^{-3}
0.9	2.2136428	2.213642800	0.716541×10^{-3}
0.95	2.456424176	2.456424176	0.791212×10^{-3}
1	2.718281828	2.7182818284	0.871835×10^{-3}

Figure 4.1 shows both the exact and the numerical solutions with $n = 20$.

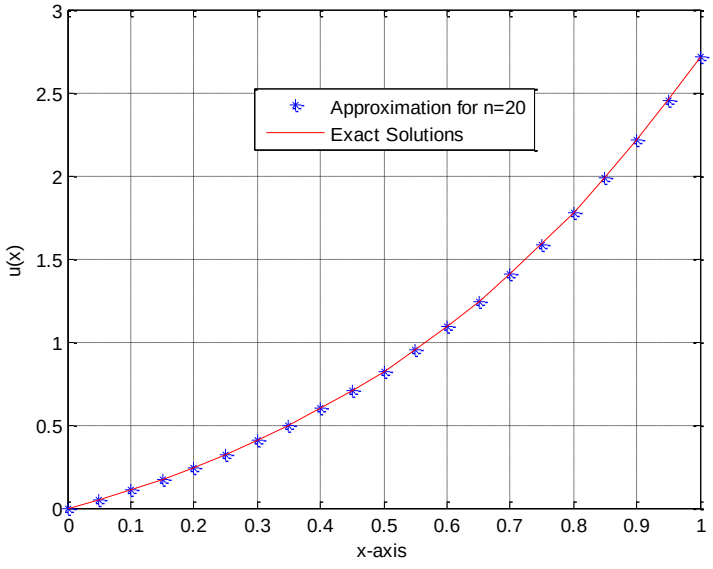

Figure 4.1: The exact and numerical solution of applying Algorithm 4.1 for equation (4.1).

The CPU time is 0.018776 seconds. Figure 4.2 shows the absolute error resulting of applying algorithm 4.1 for equation (4.1).

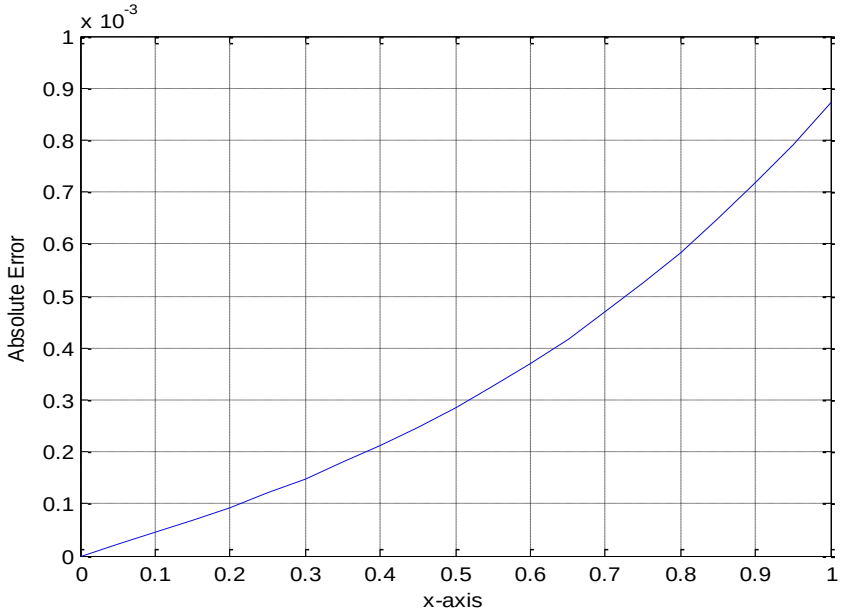

Figure 4.2: the error resulting of applying algorithm 4.1 on equation (4.1).

4.2 The numerical realization of equation (4.1) using the Runge-Kutta method

4.2.1 The Runge-Kutta method of order 2 or (the Improved Euler Method)

The following algorithm implements the Improved Euler Method using the Matlab software.

Algorithm 4.2

1. Input : 1) h - step-size

   2) a, b - endpoints of interval of integration

   3) kernel- Matlab function of the kernel

   4) f- Matlab function of $f(x)$

2. Outputs:

1. nodes- node values
2. $u(x)$- solution values at nodes
3. Specifying weights

$$theta = [0 \;\; 1]$$
$$A = \begin{bmatrix} 1 & 0 \\ 0.5 & 0.5 \end{bmatrix}$$

4. Nodes$= a\ to\ b\ \ step\ h$
5. Number of intermediate points $= (length(nodes) - 1) * 1$
6. x =Vector of nodes and intermediate points
7. Placing node values into x

$$for\ i = 1\ to\ length(nodes)$$
$$x\big(i + (i-1)\big) = nodes(i)$$
$$end$$

8. Placing intermediate points into x

$$for\ i = 1\ to\ length(nodes) - 1$$
$$for\ j = 2{:}\,2$$
$$x(i + (i-1) + \ j - 1) = x(i + (i-1)) + h * theta(j);$$
$$end$$
$$end$$

9. Keeps track of which intermediate points are associated with which node

$$Index = \text{Vector of length of } x$$
$$for\ i = 1\ to\ length(nodes) - 1$$
$$index(i + (i-1)\ to\ i + (i-1) + 1) = i$$
$$end$$
$$index\big(length(index)\big) = length(nodes)$$

10. Let $u(x)$ =Vector of solution values has the same length of x

11. Set order of method $p = 2$

12. $for\ i = 1\ to\ length\ of\ x$

$u(i) = f(x(i))$

$m = mod(i, 2)$

$k = index(i)$

$if\ m == 0$

$v = 1$

$elseif\ m == 1$

$v = 0$

end

$if\ i\sim = 1 || i\sim = 2$

$for\ j = 1\ to\ k - 1$

$for\ l = 1\ to\ p$

$ind1 = find(j == index)$

$ind1 = ind1(1)$

13. Applying Runge-Kutta formula

$$u(i) = u(i) + h * A(2, l) * kernel\left(x(i), x\left(ind1 + (l - 1)\right)\right) * u(ind1 + (l - 1))$$

end

end

end

$if\ v\sim = 0$

$for\ l = 1\ to\ v \quad (depends\ on\ mod)$

$ind1 = find(index(i) == index)$

$ind1 = ind1(1)$

14. Applying Runge-Kutta formula

$$u(i) = u(i) + h * \ A(v,l) * kernel\left(x(i), x\left(ind1 + (l-1)\right)\right)$$

$$* u(ind1 + (l-1))$$

end

end

end

15. Obtaining node values

Now let $u(x) = Vectorof\ length\ of\ nodes$

$for\ i = 1\ to\ length(nodes)$

$u(i) = u(i + (i-1))$

end

Table 4.2 shows the exact and numerical results when step size $h = 0.1$, and showing the error resulting of using the numerical solution.

Table 4.2: The exact and numerical solutions of applying Algorithm 4.2 for equation (4.1).

x	Analytical solution $u(x) = xe^x$	Approximate solution $u_h(x)$	Error= $\lvert u - u_h \rvert$
0	0	0	0
0.1	0.110517092	0.110341836	0.000175256
0.2	0.244280552	0.243908935	0.000371617
0.3	0.404957642	0.404365783	0.00059186
0.4	0.596729879	0.595889872	0.000840007
0.5	0.824360635	0.823239117	0.001121518
0.6	1.09327128	1.091827769	0.001443511
0.7	1.409626895	1.407811873	0.001815022
0.8	1.780432743	1.778185427	0.002247315
0.9	2.2136428	2.210888564	0.002754236
1	2.718281828	2.714929201	0.003352627

These results show the accuracy of the Runge-Kutta method of order 2 to solve equation (4.1) since the $max\ error = 0.003352627$.

Figure 4.3 compares the exact solution $u(x) = xe^x$ with the approximate solution with step size $h = 0.1$.

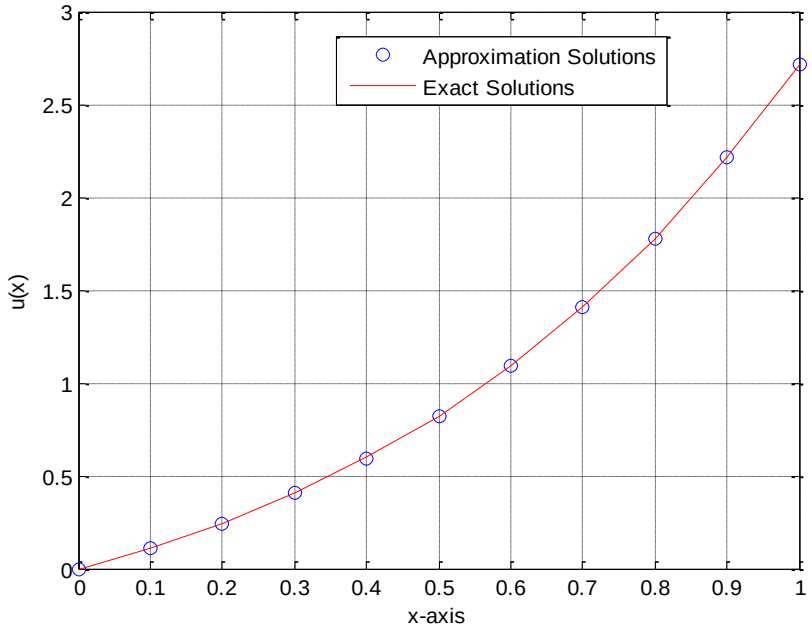

Figure 4.3: The exact and numerical solutions of applying Algorithm 4.2 for equation (4.1).

The CPU time is 0.027644 seconds. Figure 4.4 shows the absolute error resulting of applying algorithm 4.2 on equation (4.1).

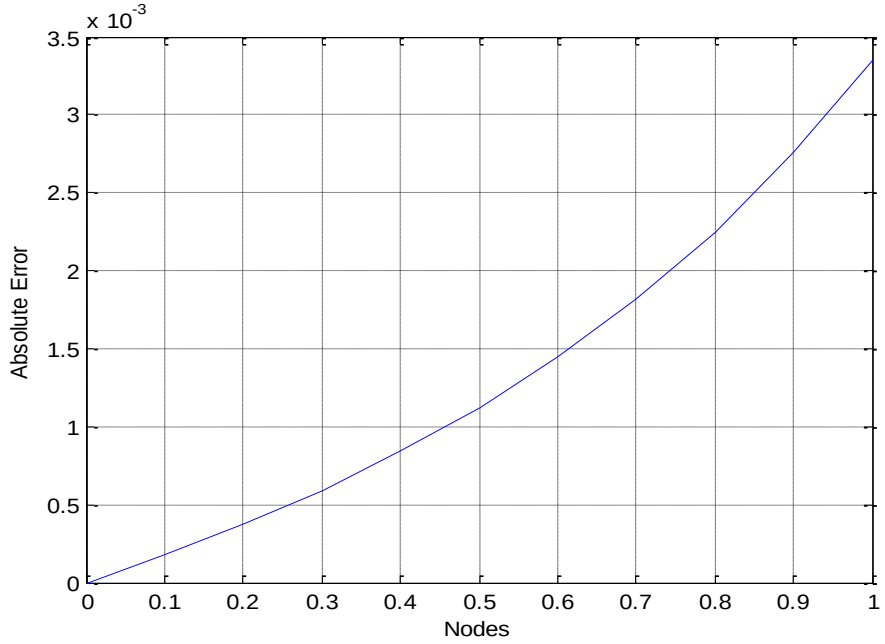

Figure 4.4: the error resulting of applying algorithm 4.2 on equation (4.1)

4.2.2 The fourth order Runge-Kutta method

The following algorithm implements the fourth order Runge-Kutta method using the Matlab software.

Algorithm 4.3

1. Input h , a , b , λ , kernel, $f(x)$

2. Outputs: 1) nodes- node values

   2) $u(x)$- solution values at nodes

3. Specifying weights

$$theta = [0\ ,0.5\ ,0.5\ ,1]$$

$$A = \begin{bmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{bmatrix}$$

4. Nodes$= a\ to\ b\ \ step\ h$

5. Number of intermediate points $= (length(nodes) - 1) * 3$

6. x =Vector of nodes and intermediate points

7. Placing node values into x

$$for\ i = 1\ to\ length(nodes)$$

$$x\big(i + 3 * (i - 1)\big) = nodes(i)$$

$$end$$

8. Placing intermediate points into x

$$for\ i = 1\ to\ length(nodes) - 1$$

$$for\ j = 2{:}\ 4$$

$$x(i + 3 * (i - 1) + \ j - 1) = x(i + 3 * (i - 1)) + h * theta(j);$$

end

end

9. Keeps track of which intermediate points are associated with which node

$Index =$ Vector of length of x

$for\ i = 1\ to\ length(nodes) - 1$

$index(i + 3 * (i - 1)\ to\ i + 3 * (i - 1) + 3) = i$

end

$index\big(length(index)\big) = length(nodes)$

10. Let $u(x)$ =Vector of solution values has the same length of x

11. Set order of method $p = 4$

12. $for\ i = 1\ to\ length\ of\ x$

$u(i) = f(x(i))$

$m = mod(i, 4)$

$k = index(i)$

$if\ m == 2$

$v = 1$

$elseif\ m == 3$

$v = 2$

$elseif\ m == 0$

$v = 3$

$elseif\ m == 1$

$v = 0$

end

$if\ i\sim=1||\ i\sim=2||i\sim=3||\ i\sim=4$

$for\ j=1\ to\ k-1$

$for\ l=1\ to\ p$

$ind1=\ find(j==index)$

$ind1=ind1(1)$

13. Applying Runge-Kutta formula

$$u(i)=u(i)+h*A(4,l)*kernel\left(x(i),x\left(ind1+(l-1)\right)\right)$$
$$*u(ind1+(l-1))$$

end

end

end

$if\ v\sim=0$

$for\ l=1\ to\ v\quad(depends\ on\ mod)$

$ind1=\ find(index(i)==index)$

$ind1=ind1(1)$

14. Applying Runge-Kutta formula

$$u(i)=u(i)+h*\ A(v,l)*kernel\left(x(i),x\left(ind1+(l-1)\right)\right)$$
$$*u(ind1+(l-1))$$

end

end

end

14. Obtaining node values

Now let $u(x) = Vectorof\ length\ of\ nodes$

$for\ i = 1\ to\ length(nodes)$

$u(i) = u(i + 3 * (i - 1))$

end

Table 4.3 shows the exact and numerical results when step size $h = 0.1$, and showing the error resulting of using the numerical solution.

Table 4.3: The exact and numerical solutions of applying Algorithm 4.3 for equation (4.1).

x	Analytical solution $u(x) = xe^x$	Approximate solution $u_h(x)$	Error= $\|u - u_h\|$
0	0	0	0×10^{-5}
0.1	0.11051709	0.110516977	$0.011501376 \times 10^{-5}$
0.2	0.24428055	0.244280304	$0.024723293\times 10^{-5}$
0.3	0.40495764	0.404957241	$0.040154546\times 10^{-5}$
0.4	0.59672988	0.596729295	$0.058363265\times 10^{-5}$
0.5	0.82436064	0.824359835	$0.080010692\times 10^{-5}$
0.6	1.09327128	1.093270222	$0.105866965\times 10^{-5}$
0.7	1.4096269	1.409625527	$0.136829191\times 10^{-5}$
0.8	1.78043274	1.780431003	$0.173942149\times 10^{-5}$
0.9	2.2136428	2.213640616	$0.218422001\times 10^{-5}$
1	2.71828183	2.718279112	$0.271683433\times 10^{-5}$

These results show the accuracy of the fourth order Runge-Kutta method to solve equation (4.1) since the $max\ error = 0.271683433 \times 10^{-5}$.

Figure 4.5 compares the exact solution $u(x) = xe^x$ with the approximate solution with step size $h = 0.1$.

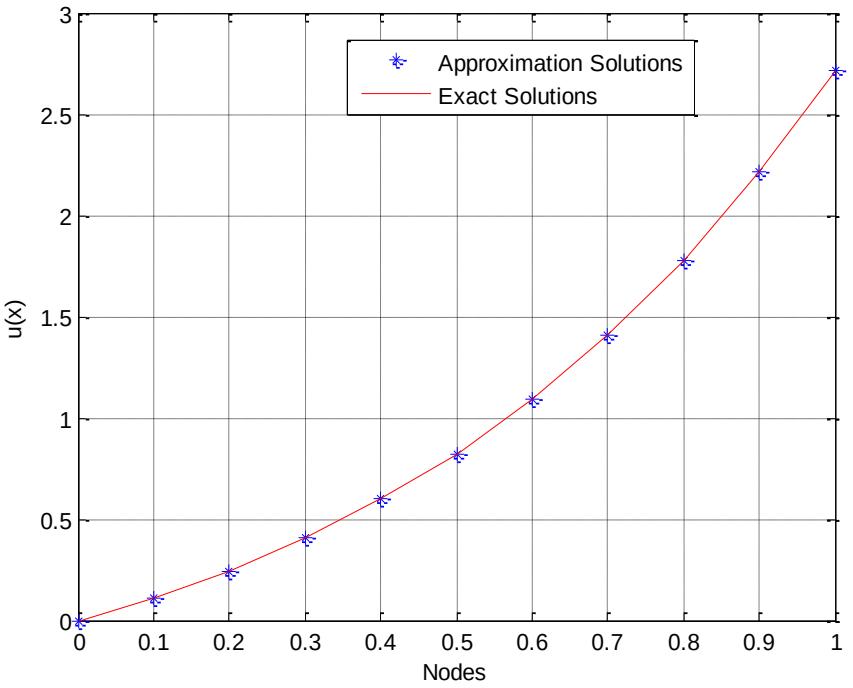

Figure 4.5: The exact and numerical solutions of applying Algorithm 4.3 for equation (4.1).

The CPU time is 0.034696 seconds. Figure 4.6 shows the absolute error resulting of applying algorithm 4.3 on equation (4.1).

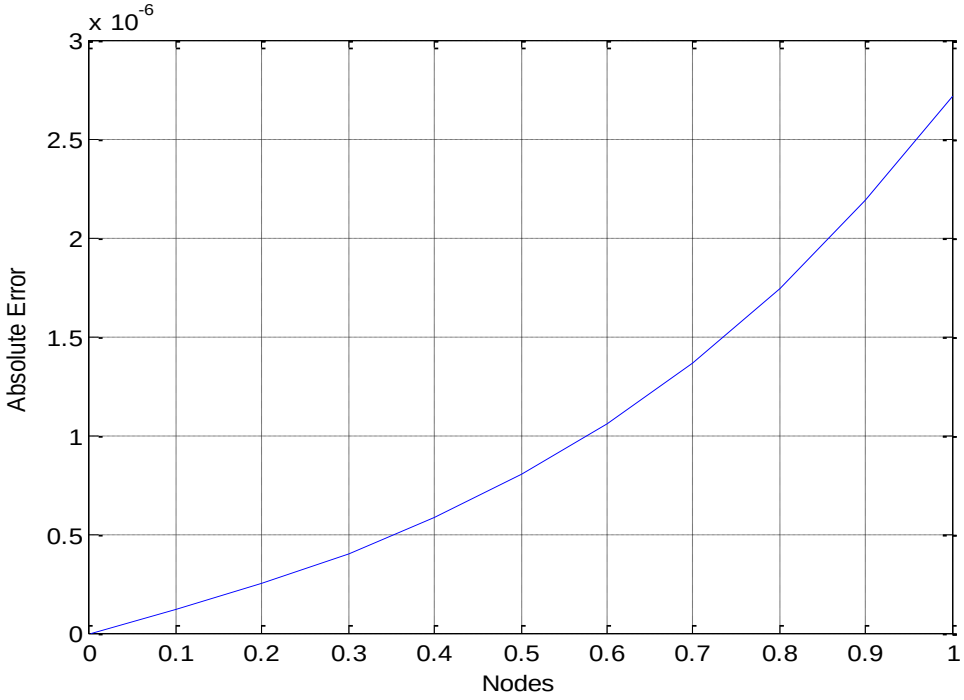

Figure 4.6: the error resulting of applying algorithm 4.3 on equation (4.1).

4.3 The numerical realization of equation (4.1) using the Block method

The following algorithms (Block 2 and Block 3) for solving Volterra integral equation of the second kind (4.1) using the two Block method and three Block method respectively:

Algorithm 4.4 (Block 2)

Step (1):

1. Put $h = (b-a)/n \, ; n \in N$
2. Set $u_0 = f_0 = f(a)$

Step (2):

$for\ m = 1\ to\ n-1$

Calculate $u_m\ and\ \ u_{m+1}$ using equations (3.49), (3.50) which are shown in chapter three section two, and use Gauss elimination procedure to solve the resulting system.

Algorithm 4.5 (Block 3)

Step (1):

1. Put $h = (b-a)/n \, ; n \in N$
2. Set $u_0 = f_0 = f(a)$

Step (2):

$for\ m = 1\ to\ n-2$

Calculate $u_m\ , u_{m+1} and\ u_{m+2}$ using equations (3.54), (3.55) and (3.56) which are shown in chapter three section two, and use Gauss elimination procedure to solve the resulting system.

Table 4.4 shows the exact and numerical results when applying algorithm 4.4 (Block 2), and showing the error resulting of using the numerical solution.

Table 4.4: The exact and numerical solutions of applying Algorithm 4.4 for equation (4.1).

x	Analytical solution $u = xe^x$	Approximate solution $u_h(x)$	Error= $\lvert u - u_h \rvert$
0	0	0	0
0.1	0.110517092	0.110517092	0.000175256
0.2	0.244280552	0.244280552	0.001015278
0.3	0.404957642	0.404957642	0.001283973
0.4	0.596729879	0.596729879	0.0037413
0.5	0.824360635	0.824360635	0.002932474
0.6	1.09327128	1.09327128	0.007714405
0.7	1.409626895	1.409626895	0.005368352
0.8	1.780432743	1.780432743	0.013414801
0.9	2.2136428	2.2136428	0.008924114
1	2.718281828	2.718281828	0.007691095

Figure 4.7 shows both the exact and the numerical solutions with $n = 10$

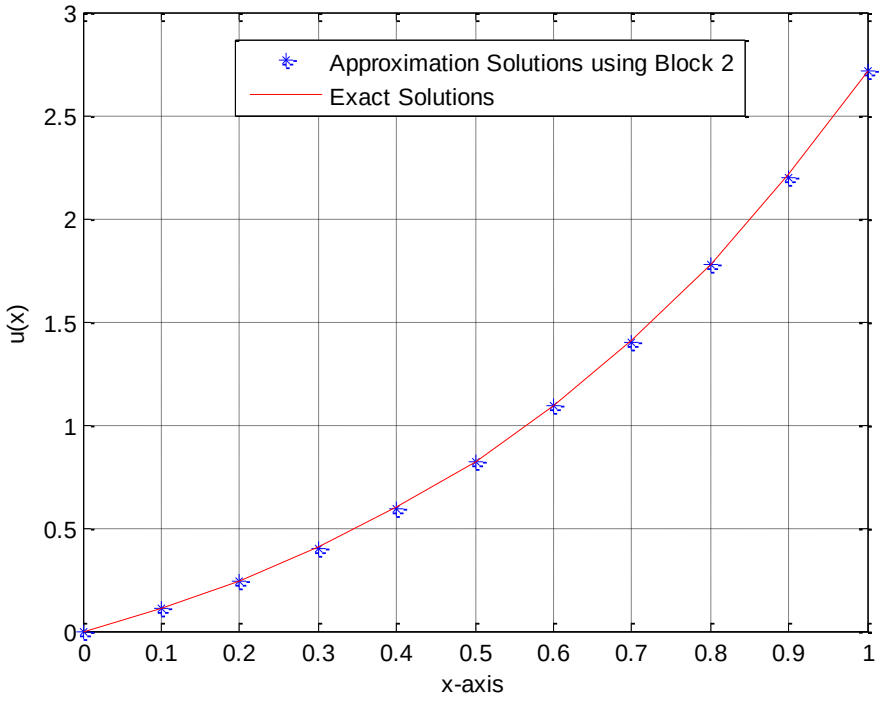

Figure 4.7: The exact and numerical solutions of applying Algorithm 4.4 for equation (4.1).

The CPU time is 0.024423seconds. Figure 4.8 shows the absolute error resulting of applying algorithm 4.4 on equation (4.1).

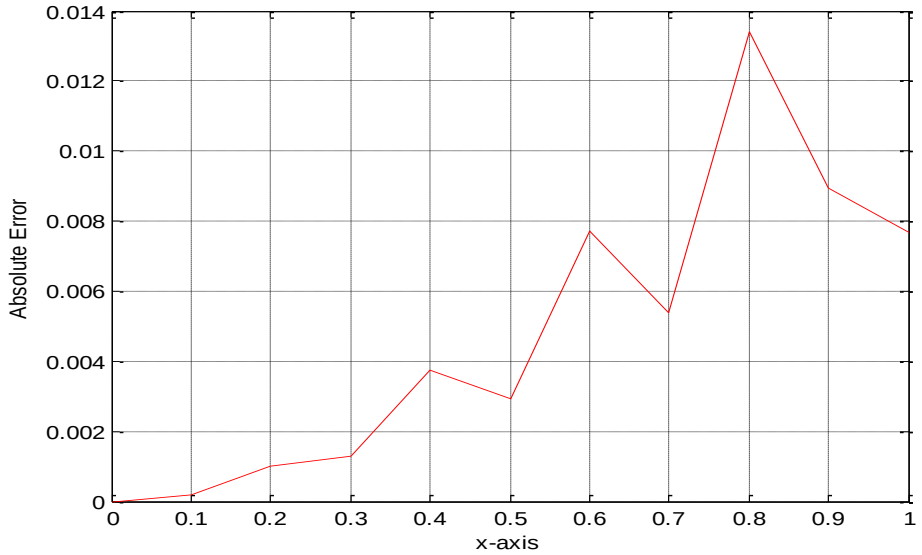

Figure 4.8: the error resulting of applying algorithm 4.4 on equation (4.1).

Table 4.5 shows the exact and numerical results when applying algorithm 4.5 (Block 3), and showing the error resulting of using the numerical solution.

Table 4.5: The exact and numerical solution of applying Algorithm 4.5 for equation (4.1).

x	Analytical solution $u(x) = xe^x$	Approximate solution $u_h(x)$	Error= $\lvert u - u_h \rvert$
0	0	0	0
0.05	0.052563555	0.052542193	2.13621E-05
0.1	0.110517092	0.110580628	6.35364E-05
0.15	0.174275136	0.174337428	6.22911E-05
0.2	0.244280552	0.244402305	0.000121753
0.25	0.321006354	0.321270884	0.00026453
0.3	0.404957642	0.405172042	0.0002144
0.35	0.496673642	0.496972541	0.000298899
0.4	0.596729879	0.597277609	0.00054773
0.45	0.705740483	0.706174156	0.000433672
0.5	0.824360635	0.824911163	0.000550527
0.55	0.95328916	0.954225985	0.000936826
0.6	1.09327128	1.094011611	0.000740331
0.65	1.245101539	1.246000269	0.00089873
0.7	1.409626895	1.411088363	0.001461467
0.75	1.587750012	1.588909892	0.001159879
0.8	1.780432743	1.781804061	0.001371318
0.85	1.988699824	1.990858493	0.002158669
0.9	2.2136428	2.215367152	0.001724352
0.95	2.456424176	2.460025314	0.003601138
1	2.718281828	2.718175056	0.000106772

Figure 4.9 shows the exact solution $u(x) = xe^x$ with the approximate solution when $n = 20$

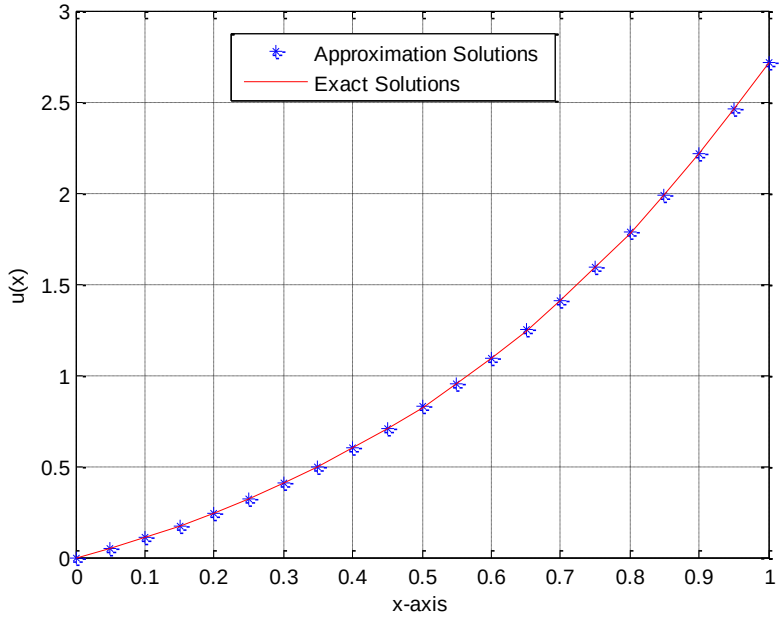

Figure 4.9: The exact and numerical solutions of applying Algorithm 4.5 for equation (4.1).

The CPU time is 0.021548 seconds. Figure 4.10 shows the absolute error resulting of applying algorithm 4.5 on equation (4.1).

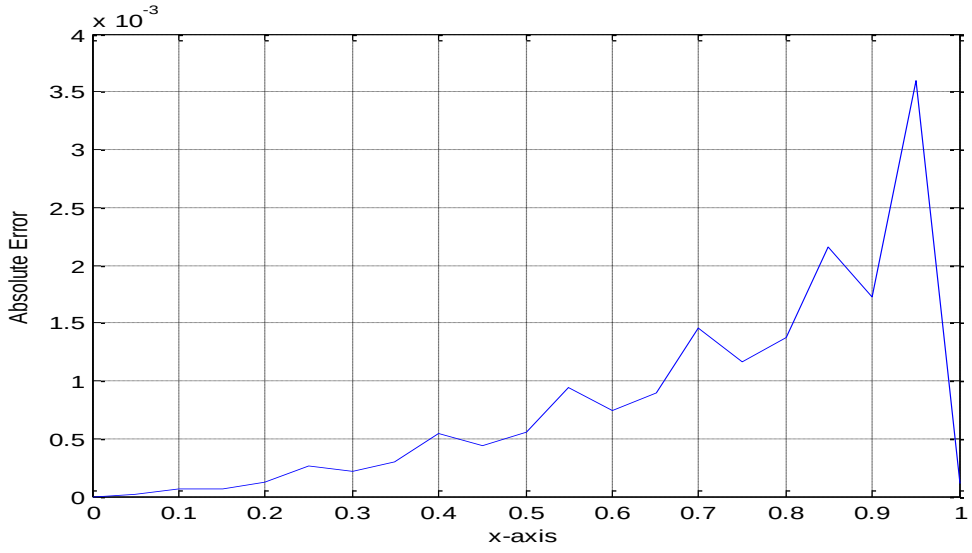

Figure 4.10: the error resulting of applying algorithm 4.5 on equation (4.1)

Example 4.2

Consider the Volterra integral equation of the second kind

$$u(x) = 1 - xsin(x) + xcos(x) + \int_0^x tu(t)dt \ . \qquad (4.2)$$

Equation (4.2) has the exact solution

$$u(x) = \sin(x) + \cos(x).$$

We will use all the numerical methods that we used in the example 4.1.

4.4 The numerical realization of equation (4.2) using Trapezoidal rule

Table 4.6 shows the exact and numerical results when n=20, and showing the error resulting of using the numerical solution.

Table 4.6: shows the exact and numerical results when n=20, and showing the error resulting of using the numerical solution.

x	Analytical solution $u(x) = \sin(x) + \cos(x)$	Approximate solution $u_h(x)$	Error= $\lvert u - u_h \rvert$
0	1	1	0
0.05	1.04872943	1.0487495	0.02006×10^{-3}
0.1	1.094837582	1.0948761	0.03856×10^{-3}
0.15	1.13820921	1.1382647	0.05548×10^{-3}
0.2	1.178735909	1.1788067	0.07081×10^{-3}
0.25	1.216316381	1.2164009	0.08455×10^{-3}
0.3	1.250856696	1.2509534	0.09666×10^{-3}
0.35	1.28227052	1.2823776	0.10712×10^{-3}
0.4	1.310479336	1.3105952	0.1159×10^{-3}
0.45	1.335412636	1.3355356	0.12298×10^{-3}
0.5	1.3570081	1.3571364	0.1283×10^{-3}
0.55	1.375211751	1.3753436	0.13183×10^{-3}
0.6	1.389978088	1.3901116	0.13351×10^{-3}
0.65	1.401270204	1.4014035	0.1333×10^{-3}
0.7	1.409059875	1.409191	0.13112×10^{-3}
0.75	1.413327629	1.4134545	0.12691×10^{-3}
0.8	1.4140628	1.4141834	0.1206×10^{-3}
0.85	1.411263551	1.4113756	0.11208×10^{-3}
0.9	1.404936878	1.4050381	0.10127×10^{-3}
0.95	1.395098594	1.3951866	0.08804×10^{-3}
1	1.381773291	1.3818456	0.07229×10^{-3}

Figure 4.11 shows both the exact and the numerical solutions with $n = 20$.

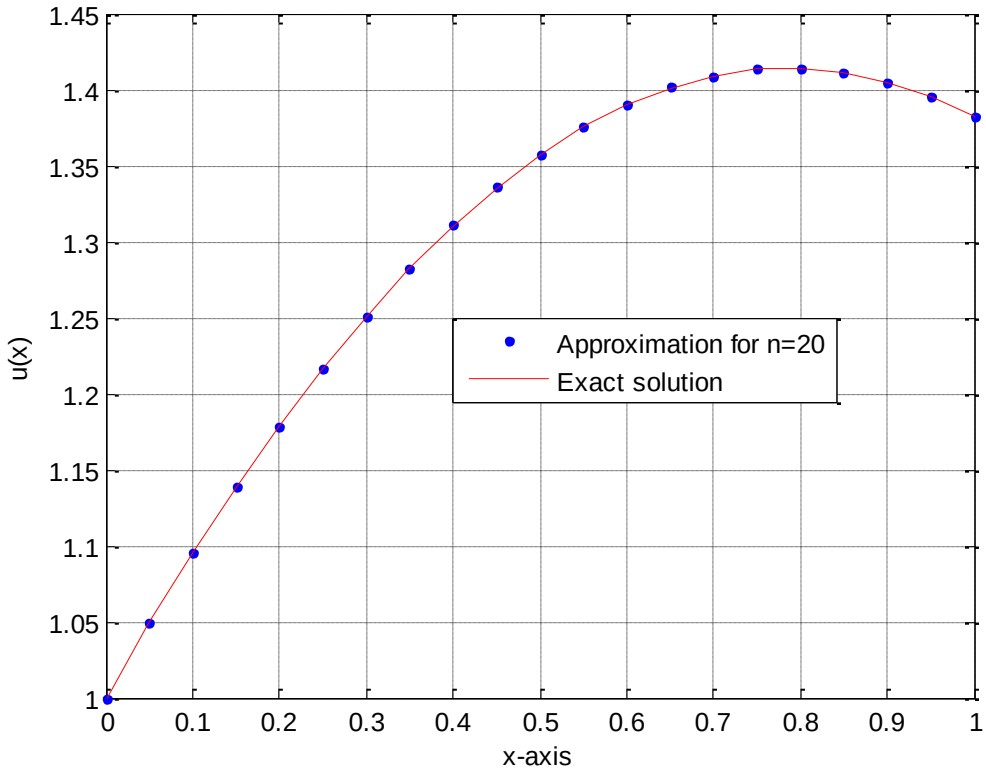

Figure 4.11: The exact and numerical solutions of applying Algorithm 4.1 for equation (4.2).

The CPU time is 0.031057 seconds. Figure 4.12 shows the absolute error resulting of applying algorithm 4.1 on equation (4.2).

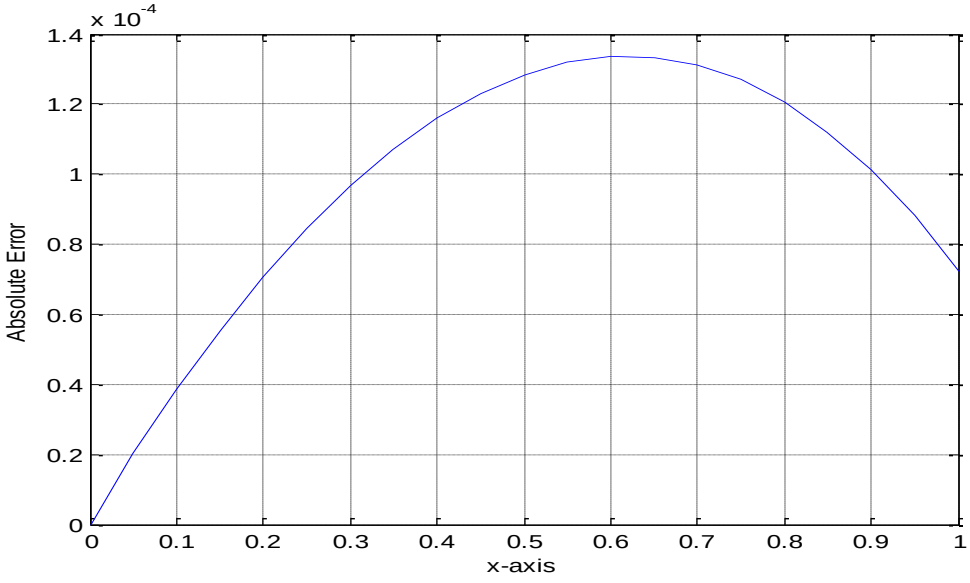

Figure 4.12: the error resulting of applying algorithm 4.1 on equation (4.2).

4.5 The numerical realization of equation (4.2) using the Runge-Kutta method

4.5.1 The Runge-Kutta method of order 2

Table 4.7 shows the exact and numerical solutions when applying Algorithm 4.2 on equation (4.2).

Table 4.7: The exact and numerical solution of applying Algorithm 4.2 for equation (4.2).

x	Analytical solution $u(x) = \sin(x) + \cos(x)$	Approximate solution u_h	Error= $\lvert u - u_h \rvert$
0	1	1	0
0.1	1.0948375819	1.094964660237	0.0001270783
0.2	1.1787359086	1.178829128432	0.0000932197
0.3	1.2508566957	1.250650273503	0.00020642228
0.4	1.3104793363	1.309609452533	0.00086988377
0.5	1.3570081004	1.355023865136	0.00198423535
0.6	1.3899780883	1.386357672182	0.00362041612
0.7	1.4090598745	1.403232930278	0.00582694424
0.8	1.4140628002	1.405440478563	0.00862232168
0.9	1.4049368778	1.392951018872	0.01198585902
1	1.3817732906	1.365926762361	0.01584652831

Figure 4.13 shows both the exact and the numerical solutions when step size $h = 0.1$. Figure 4.14 shows the absolute error resulting of applying algorithm 4.2 on equation (4.2). The CPU time is 0.030646 seconds.

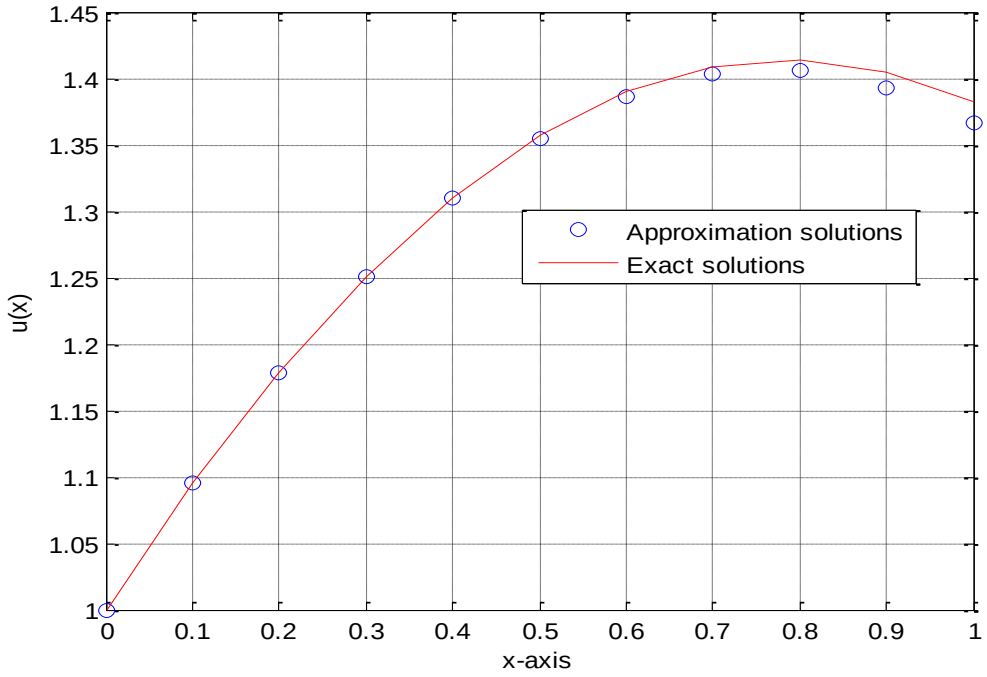

Figure 4.13: The exact and numerical solutions of applying Algorithm 4.2 for equation (4.2).

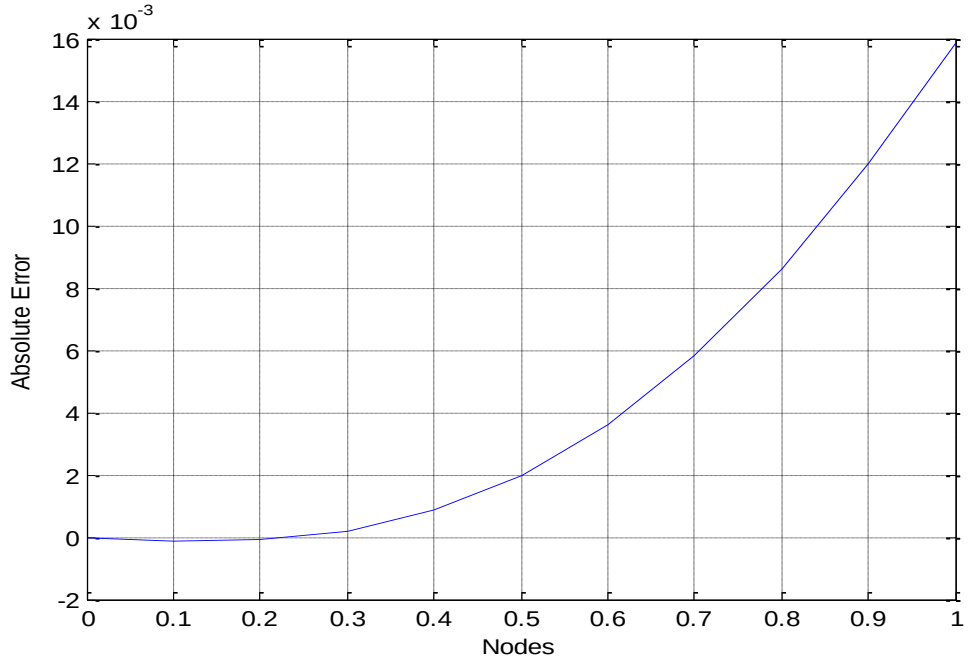

Figure 4.14: the error resulting of applying algorithm 4.2 on equation (4.2).

4.5.2 The fourth order Runge-Kutta method

Table 4.8 shows the exact and numerical solutions when applying Algorithm 4.3 on equation (4.2).

Table 4.8: The exact and numerical solutions of applying Algorithm 4.3 for equation (4.2).

x	Analytical solution $u(x) = \sin(x) + \cos(x)$	Approximate solution $u_h(x)$	Error= $\lvert u - u_h \rvert$
0	1	1	0
0.1	1.0948375819	1.0948375136	0.068618×10^{-6}
0.2	1.1787359086	1.17873580473	0.103902×10^{-6}
0.3	1.2508566957	1.2508565934	0.102642×10^{-6}
0.4	1.3104793363	1.31047927197	0.064334×10^{-6}
0.5	1.3570081004	1.35700810922	0.008726×10^{-6}
0.6	1.3899780883	1.38997819988	0.111581×10^{-6}
0.7	1.4090598745	1.40906011150	0.236987×10^{-6}
0.8	1.4140628002	1.41406317628	0.376041410^{-6}
0.9	1.4049368778	1.40493739697	0.519081×10^{-6}
1	1.3817732906	1.38177394754	0.656871×10^{-6}

These results show the accuracy of the fourth order Runge-Kutta method to solve equation (4.2) since the $max\ error = 0.656871 \times 10^{-6}$.

Figure 4.15 compares the exact solution $u(x) = \sin(x) + \cos(x)$ with the approximate solution with step size $h = 0.1$. Figure 4.16 shows the absolute error resulting of applying algorithm 4.3 on equation (4.2). The CPU time is 0.042564 seconds.

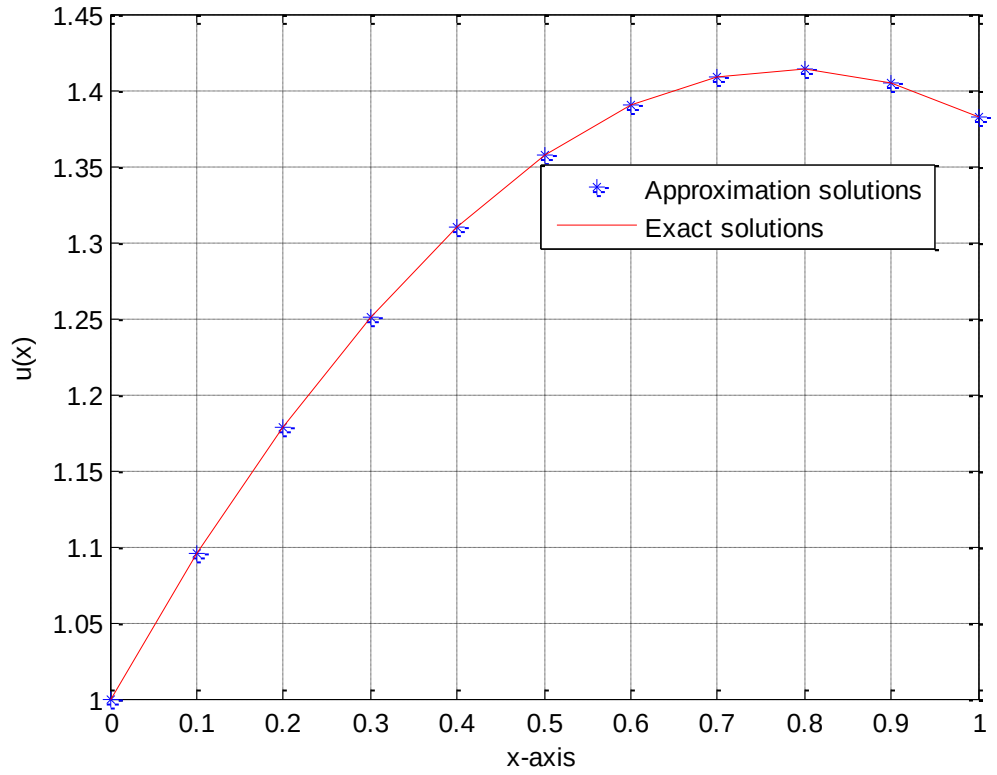

Figure 4.15: The exact and numerical solutions of applying Algorithm 4.3 for equation (4.2).

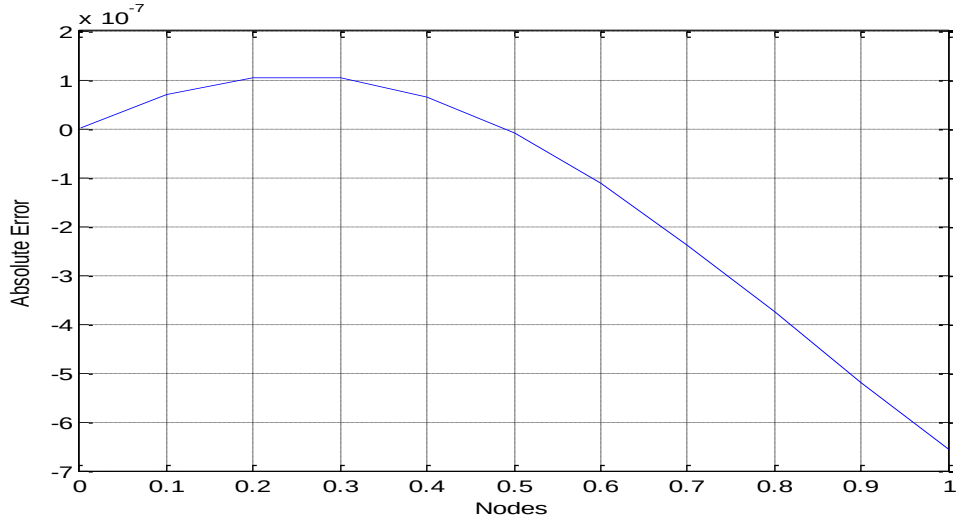

Figure 4.16: the error resulting of applying algorithm 4.3 on equation (4.2).

4.6 The numerical realization of equation (4.2) using the Block method

Table 4.9 shows the exact and numerical results when applying algorithm 4.4 (Block 2) on equation (4.2), and showing the error resulting of using the numerical solutions.

Table 4.9: The exact and numerical solutions of applying Algorithm 4.4 for equation (4.2).

x	Analytical solution $u(x) = \sin(x) + \cos(x)$	Approximate solution u_h	Error= $\lvert u - u_h \rvert$
0	1	1	0
0.1	1.0948375819	1.089517075	0.005320507
0.2	1.1787359086	1.176429855	0.002306053
0.3	1.2508566957	1.246838511	0.004018184
0.4	1.3104793363	1.292928321	0.017551016
0.5	1.3570081004	1.344501575	0.012506525
0.6	1.3899780883	1.353415132	0.036562956
0.7	1.4090598745	1.385132889	0.023926985
0.8	1.4140628002	1.355339508	0.058723292
0.9	1.4049368778	1.366107278	0.0388296
1	1.3817732906	1.322662336	0.059110954

Figure 4.17 shows both the exact and the numerical solutions with $n = 10$

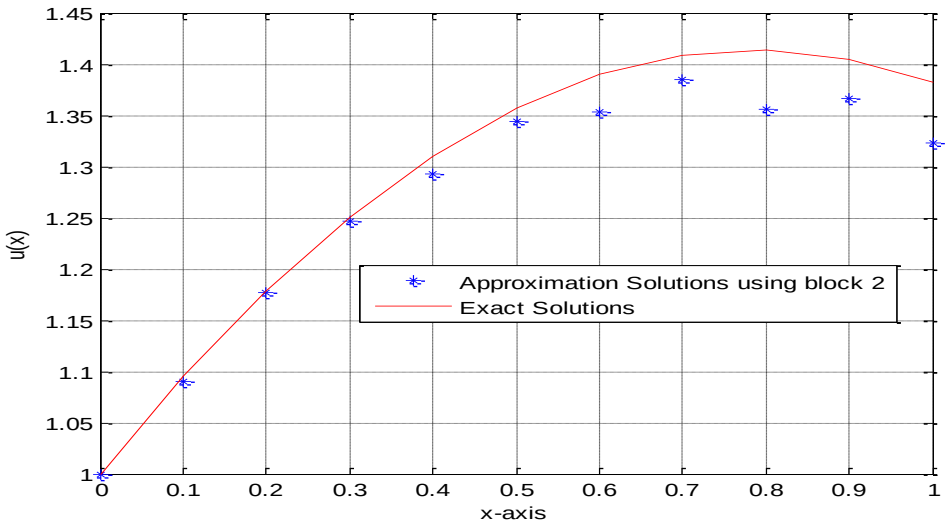

Figure 4.17: The exact and numerical solutions of applying Algorithm 4.4 for equation (4.2).

The CPU time is 0.020419 seconds. Figure 4.18 shows the absolute error resulting of applying algorithm 4.4 on equation (4.2).

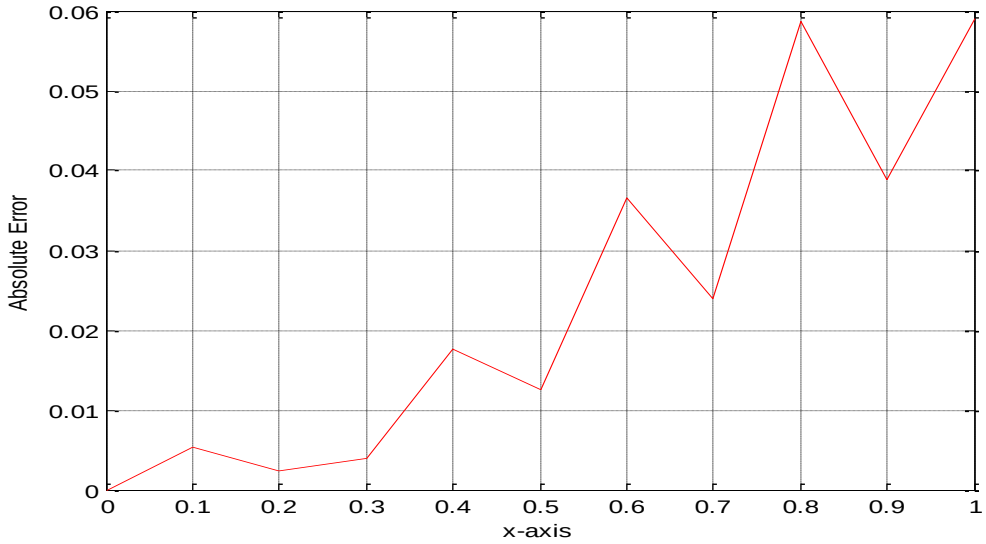

Figure 4.18: the error resulting of applying algorithm 4.4 on equation (4.2)

Table 4.10 shows the exact and numerical results when applying algorithm 4.5 (Block 3) on equation (4.2), and showing the error resulting of using the numerical solution.

Table 4.10: The exact and numerical solutions of applying Algorithm 4.5 for equation (4.2).

x	Analytical solution $u(x) = \sin(x) + \cos(x)$	Approximate solution u_h	Error= $\lvert u - u_h \rvert$
0	1	1	0
0.05	1.0487294296	1.0474385545	0.0012908751
0.1	1.0948375819	1.0964893490	0.0016517670
0.15	1.1382092104	1.1399638933	0.0017546829
0.2	1.1787359086	1.1831271015	0.0043911929
0.25	1.2163163809	1.2239166512	0.0076002703
0.3	1.2508566957	1.2565105379	0.0056538421
0.35	1.2822705203	1.2921410205	0.0098705002
0.4	1.3104793363	1.3253177493	0.0148384130
0.45	1.3354126364	1.3459547179	0.0105420814
0.5	1.3570081004	1.3734191281	0.0164110276
0.55	1.3752117509	1.3984991021	0.0232873511
0.6	1.3899780883	1.4064927889	0.0165147006
0.65	1.4012702042	1.4253096225	0.0240394183
0.7	1.4090598745	1.4419669659	0.0329070913
0.75	1.4133276288	1.4370526147	0.0237249858
0.8	1.4140628002	1.4469159900	0.0328531897
0.85	1.4112635510	1.4549955444	0.0437319934
0.9	1.4049368778	1.4373453783	0.0324085004
0.95	1.3950985942	1.4275080107	0.0324094165
1	1.3817732906	1.3669518409	0.0148214497

Figure 4.19 compares the exact solution $u(x) = \sin(x) + \cos(x)$ with the approximate solution when $n = 20$.

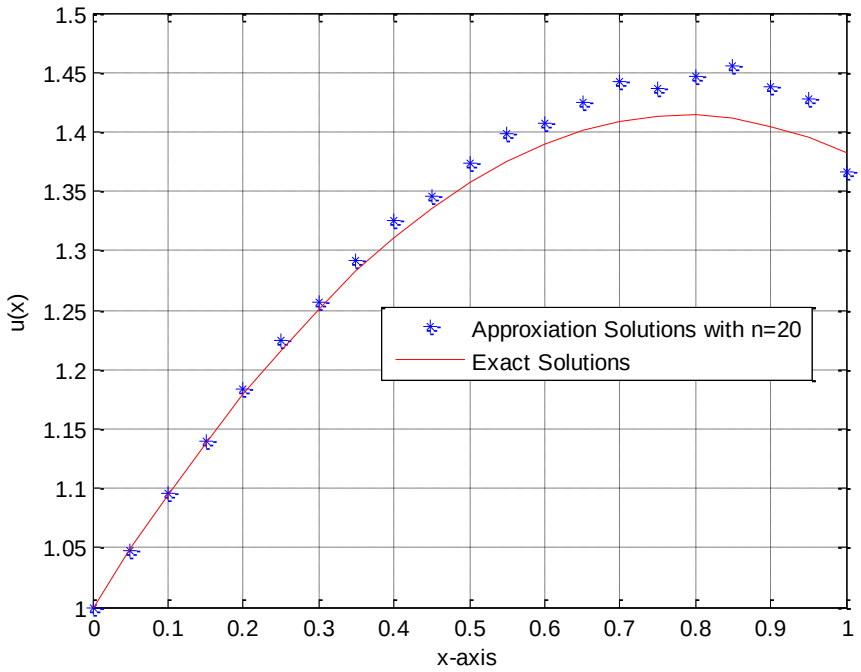

Figure 4.19: The exact and numerical solutions of applying Algorithm 4.5 for equation (4.2).

The CPU time is 0.028830 seconds. Figure 4.20 shows the absolute error resulting of applying algorithm 4.5 on equation (4.2).

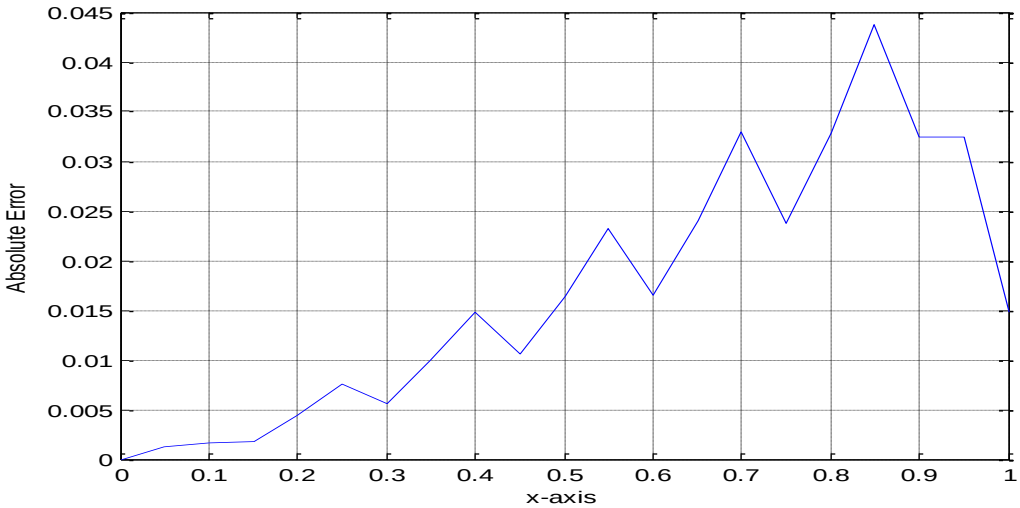

Figure 4.20: the error resulting of applying algorithm 4.5 on equation (4.2).

4.7 The numerical realization of equation (4.2) using the Collocation method.

The approximate solution of Volterra integral equation of the second kind calculated at the 7^{th} iteration $n = 10$, the following algorithm implements the Collocation method using the Matlab software.

Algorithm 4.6

1. $Input\ a, b, \lambda, f(x), k(x,t), n$
2. $x_0 = a, x_n = b$
3. Calculate $h = (b-a)/n$
4. Calculate $x = linspace(a, b, n+1)$
5. Let ${u_h}^{(0)}(\mathrm{x}) = f(x)$
6. Compute the collocation solution $u_h(x)$ by the iterated collocation solution

$$ {u_h}^{(i)}(\mathrm{x}) = f(x) + \int_a^x k(x,t)\ {u_h}^{(i-1)}(t) $$

7. Maximum absolute error= $Max|u(x) - u_h(x)|$
8. $Plot\ (u(x), u_h)$

So we obtain the following results:

Table 4.11shows the exact and numerical solutions when applying Algorithm 4.6 on equation (4.2), and showing the error resulting of using the numerical solution.

Table 4.11: The exact and numerical solutions of applying Algorithm 4.6 for equation (4.2)

x	Analytical solution $u(x) = \sin(x) + \cos(x)$	Approximate solution u_h	Error= $\lvert u - u_h \rvert$
0	1	1	0
0.1	1.0948375819	1.0948375820	0.000105×10^{-6}
0.2	1.1787359086	1.17873590870	0.000066×10^{-6}
0.3	1.2508566957	1.25085669581	0.000026×10^{-6}
0.4	1.3104793363	1.31047933606	0.000242×10^{-6}
0.5	1.3570081004	1.35700810048	0.000007×10^{-6}
0.6	1.3899780883	1.38997808832	0.000017×10^{-6}
0.7	1.4090598745	1.40905987401	0.000507×10^{-6}
0.8	1.4140628002	1.41406279732	0.002921×10^{-6}
0.9	1.4049368778	1.40493685659	0.021307×10^{-6}
1	1.3817732906	1.38177317148	0.119195×10^{-6}

These results show the accuracy of Collocation method to solve equation (4.2) since the $max\ error = 0.119195 \times 10^{-6}$.

Figure 4.21 compares the exact solution $u(x) = \sin(x) + \cos(x)$ with the approximate solution when $n = 10$.

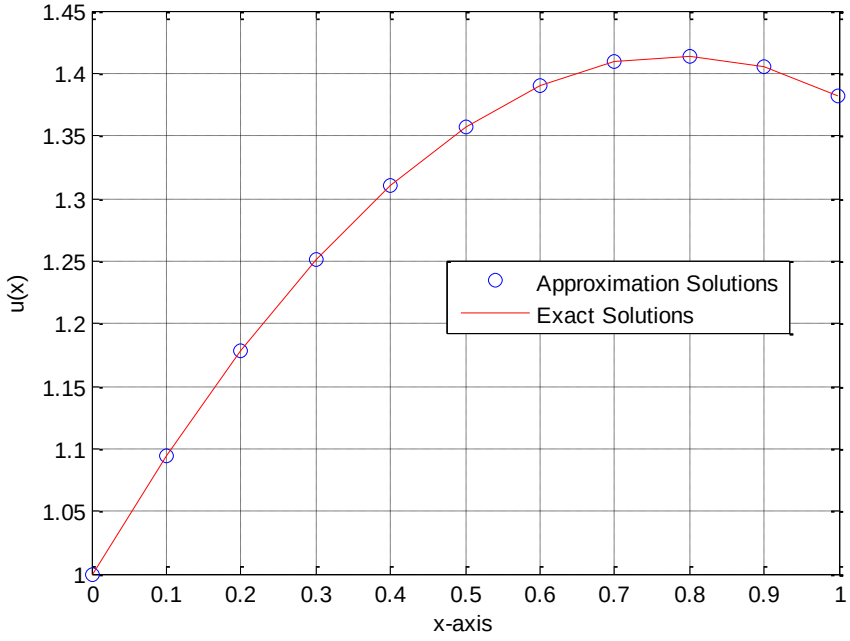

Figure 4.21: The exact and numerical solutions of applying Algorithm 4.6 for equation (4.2).

The CPU time is 0.379443 seconds. Figure 4.22 shows the absolute error resulting of applying algorithm 4.6 on equation (4.2).

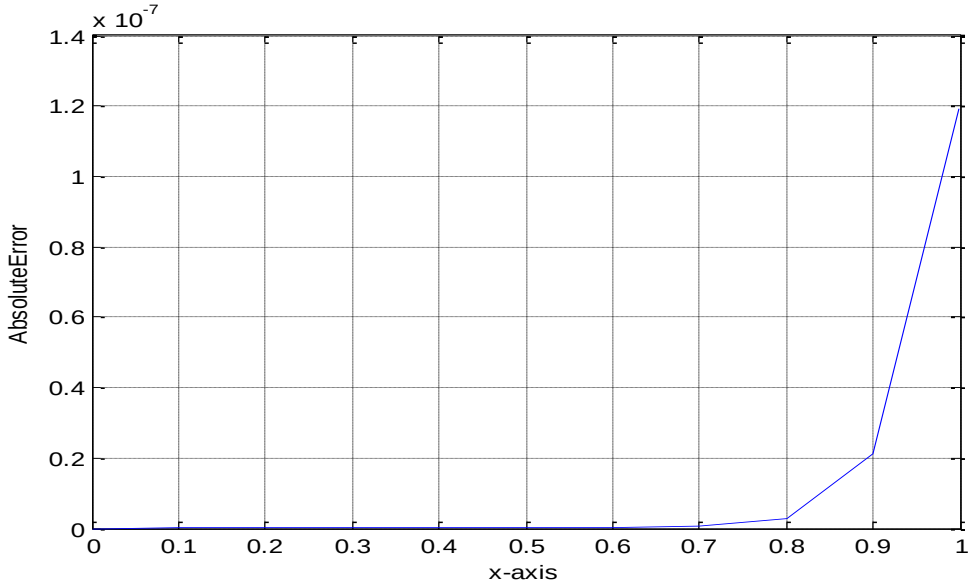

Figure 4.22: the error resulting of applying algorithm 4.6 on equation (4.2).

Example 4.3

The following Volterra integral equation of the second kind

$$u(x) = 1 + x - x^2 + \int_0^x u(t)dt \ . \qquad (4.3)$$

Equation (4.3) has the exact solution

$$u(x) = 1 + 2x.$$

4.8 The numerical realization of equation (4.3) using the Galerkin method with Chebyshev polynomial

The following algorithm for solving Volterra integral equation of the second kind (4.3) using the Galerkin method with Chebyshev polynomial.

Algorithm 4.7

1) Let $F = zeros(n + 1,1)$

$A = zeros(n + 1,1)$

$K = zeros(n + 1, n + 1)$

$T_i(x)$ is Chebyshev polynomial of degree i

2) Calculate all entries of vector F such that

$$F_i = \int_0^1 f(x)T_i(x)dx\,, \qquad i = 0,1,2,\dots,n$$

3) Calculate all entries of matrix K such that

$$K_{i,j} = \left[\int_0^1 \left[T_i(x) + \lambda \int_0^x K(x,t)T_i(t)\,dt\right] T_j(x)dx\right], i,j = 0,1,2,\dots,n$$

4) Find the unknown vector A by solve this system $AK = F$

5) Substituting the entries of vector A at the technique of Galerkin method [4], to find an approximate solution $u(x)$ of (4.2). For this, we assume that

$$\bar{u}(x) = \sum_{i=0}^{n} A_i T_i(x)$$

6) Maximum absolute error $= Max|u(x) - \bar{u}(x)|$.

Table 4.12 for $n = 10$ shows the exact and numerical results when applying algorithm 4.7 on equation (4.3), and showing the error resulting of using the numerical solution.

Table 4.12: The exact and numerical solutions of applying Algorithm 4.7 for equation (4.3).

x	Analytical solution $u(x) = 1 + 2x$	Approximate solution $u_h(x)$	Error $= \|u - u_h\|$
0	1	0.999798920	0.2010794×10^{-3}
0.1	1.2	1.199948050	0.051949×10^{-3}
0.2	1.4	1.400031461	0.031461×10^{-3}
0.3	1.6	1.600017341	0.017341×10^{-3}
0.4	1.8	1.799943469	0.056530×10^{-3}
0.5	2	2.000030734	0.030734×10^{-3}
0.6	2.2	2.200037998	0.037998×10^{-3}
0.7	2.4	2.399935471	0.064528×10^{-3}
0.8	2.6	2.6000252227	0.025222×10^{-3}
0.9	2.8	2.8000178425	0.017842×10^{-3}
1	3	3.0002797874	0.279787×10^{-3}

These results show the accuracy of Galerkin method with Chebyshev polynomial to solve equation (4.3) since the $max\ error = 0.279787 \times 10^{-3}$.

Figure 4.23 compares the exact solution $u(x) = 1 + 2x$ with the approximate solution when $n = 10$.

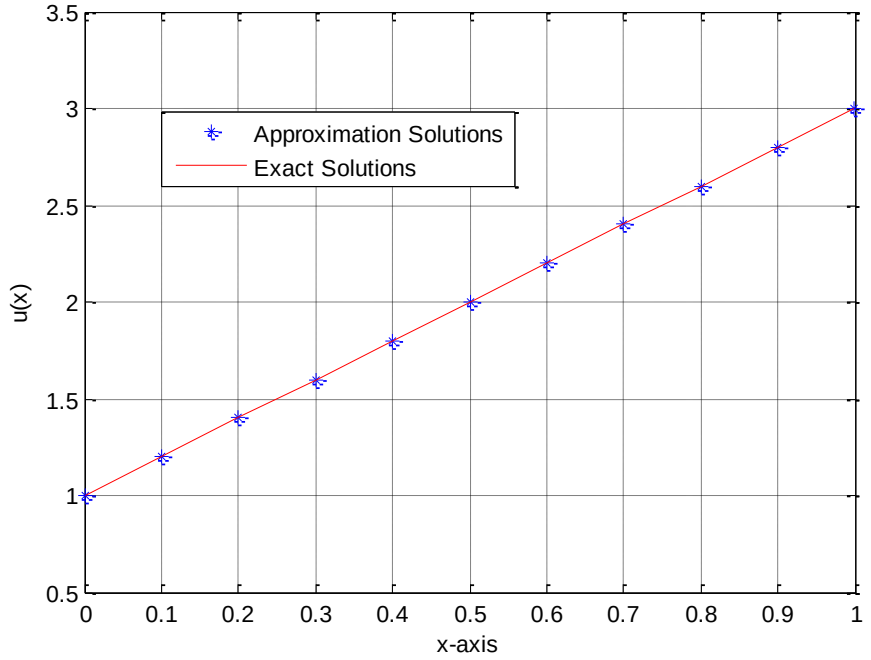

Figure 4.23 The exact and numerical solutions of applying Algorithm 4.7 for equation (4.3).

The CPU time is 0.64554 seconds. Figure 4.24 shows the absolute error resulting of applying algorithm 4.7on equation (4.3).

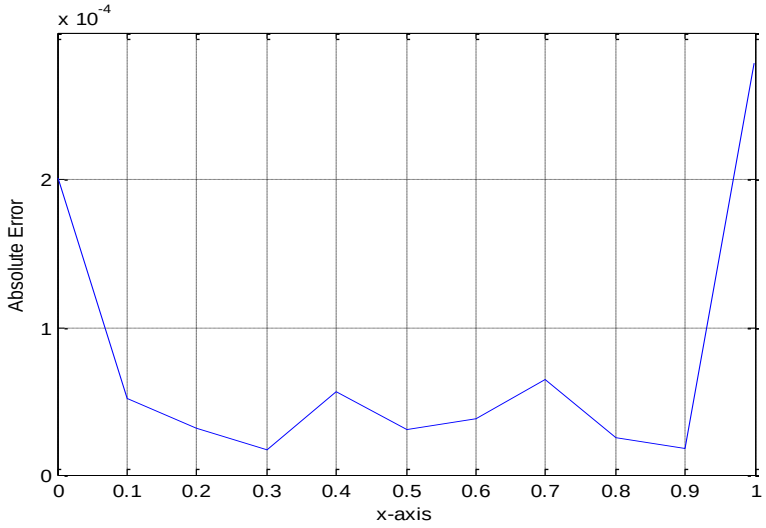

Figure 4.24: the error resulting of applying algorithm 4.7 on equation (4.3).

4.9 The numerical realization of equation (4.3) using the Collocation method

Table 4.13 for $n = 10$ shows the exact and numerical results when applying algorithm 4.6 on equation (4.3), and showing the error resulting of using the numerical solution.

Table 4.13: The exact and numerical solutions of applying Algorithm 4.6 for equation (4.3).

x	Analytical solution $u(x) = 1 + 2x$	Approximate solution $u_h(x)$	Error $= \lvert u - u_h \rvert$
0	1	1	0
0.1	1.2	1.2	0
0.2	1.4	1.399999999	0.00000002×10^{-6}
0.3	1.6	1.599999999	0.0000017×10^{-6}
0.4	1.8	1.799999999	0.0000309×10^{-6}
0.5	2	1.999999999	0.0002935×10^{-6}
0.6	2.2	2.199999998	0.0018480×10^{-6}
0.7	2.4	2.399999991	0.0087749×10^{-6}
0.8	2.6	2.599999966	0.0338933×10^{-6}
0.9	2.8	2.799999888	0.1118096×10^{-6}
1	3	2.999999674	0.32567740×10^{-6}

These results show the accuracy of Collocation method to solve equation (4.3) since the $max\ error = 0.32567740 \times 10^{-6}$.

Figure 4.25 shows the exact solution $u(x) = 1 + 2x$ with the approximate solution when $n = 10$.

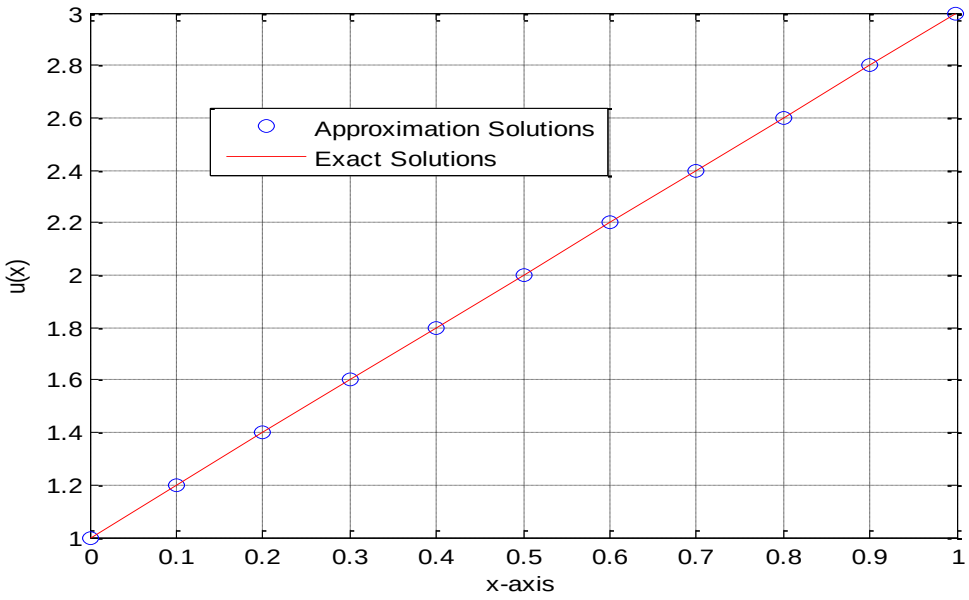

Figure 4.25 The exact and numerical solutions of applying Algorithm 4.6 for equation (4.3).

The CPU time is 0.777991 seconds. Figure 4.26 shows the absolute error resulting of applying algorithm 4.6 on equation (4.3).

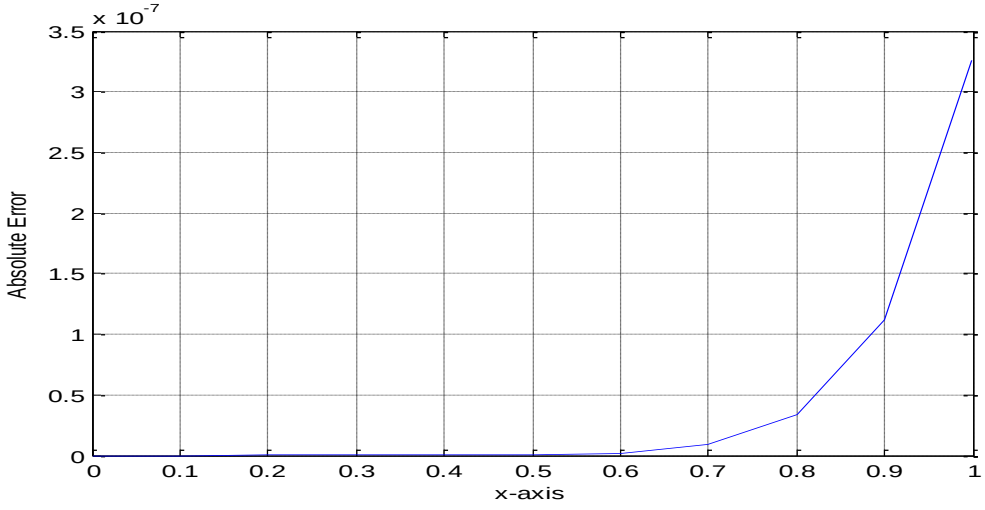

Figure 4.26: the error resulting of applying algorithm 4.7 on equation (4.3).

We conclde from our numerical test cases in this example 4.3 that the Collocation method is more efficient than the Galerkin method with Chebyshev polynomial.

4.10 The numerical realization of equation (4.3) using Trapezoidal rule

Table 4.14 for $n = 10$ shows the exact and numerical results when applying algorithm 4.1 on equation (4.3), and showing the error resulting of using the numerical solution.

Table 4.14: The exact and numerical solutions of applying Algorithm 4.1 for equation (4.3).

x	Analytical solution $u(x) = 1 + 2x$	Approximate solution $u_h(x)$	Error $= \|u - u_h\|$
0	1	1	0
0.1	1.2	1.2	0.193179×10^{-13}
0.2	1.4	1.4	0.215383×10^{-13}
0.3	1.6	1.6	0.239808×10^{-13}
0.4	1.8	1.8	0.264233×10^{-13}
0.5	2	2	0.288658×10^{-13}
0.6	2.2	2.2	0.319744×10^{-13}
0.7	2.4	2.4	0.35083×10^{-13}
0.8	2.6	2.6	0.390799×10^{-13}
0.9	2.8	2.8	0.430767×10^{-13}
1	3	3	0.475175×10^{-13}

These results show the accuracy of Trapezoidal rule to solve equation (4.3) since the $max\ error = 0.475175\times 10^{-13}$.

Figure 4.27 shows the exact solution $u(x) = 1 + 2x$ with the approximate solution when $n = 10$.

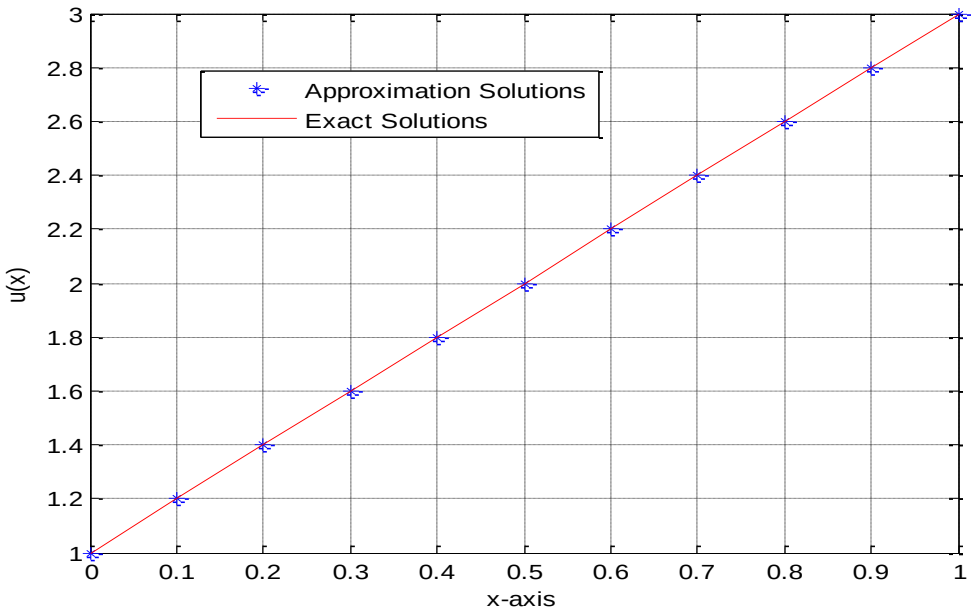

Figure 4.27: The exact and numerical solutions of applying Algorithm 4.1 for equation (4.3).

The CPU time is 0.029789 seconds. Figure 4.28 shows the absolute error resulting of applying algorithm 4.1 on equation (4.3).

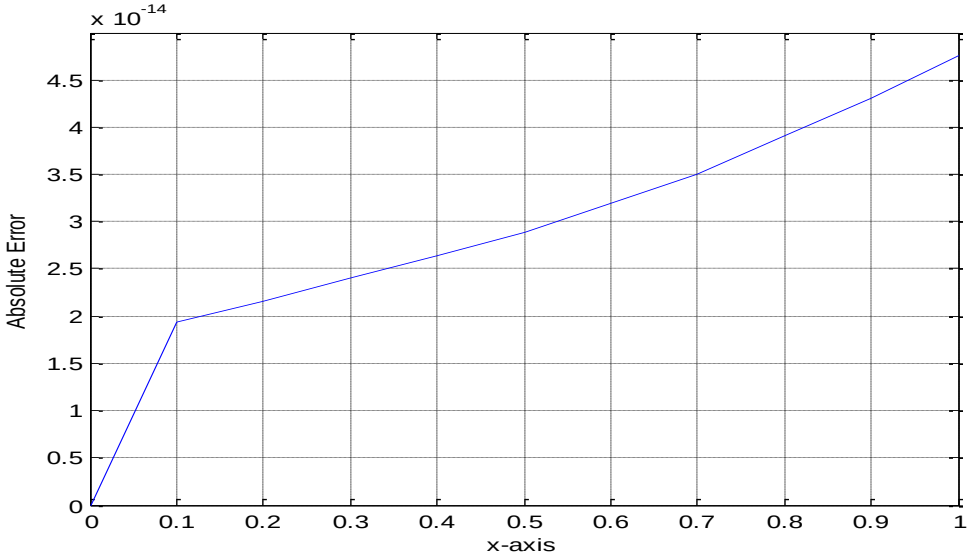

Figure 4.28: the error resulting of applying algorithm 4.1 on equation (4.3).

4.11 The numerical realization of equation (4.3) using the Runge-Kutta method

4.11.1 The Runge-Kutta method of order 2

Table 4.15 shows the exact and numerical solutions when applying Algorithm 4.2 on equation (4.3).

Table 4.15: The exact and numerical solution of applying Algorithm 4.2 for equation (4.3).

x	Analytical solution $u(x) = 1 + 2x$	Approximate solution u_h	Error= $\lvert u - u_h \rvert$
0	1	1	0
0.1	1.2	1.1945	0.0055
0.2	1.4	1.38795	0.01205
0.3	1.6	1.580245	0.019755
0.4	1.8	1.7712695	0.0287305
0.5	2	1.96089645	0.03910355
0.6	2.2	2.148986095	0.051013905
0.7	2.4	2.335384705	0.064615296
0.8	2.6	2.519923175	0.080076825
0.9	2.8	2.702415492	0.097584508
1	3	2.882657042	0.117342958

These results show the accuracy of the Runge-Kutta method of order 2 to solve equation (4.3) since the $max\ error$ =0.117342958.

Figure 4.29 shows the exact solution $u(x) = 1 + 2$x with the approximate solution when $n = 10$.

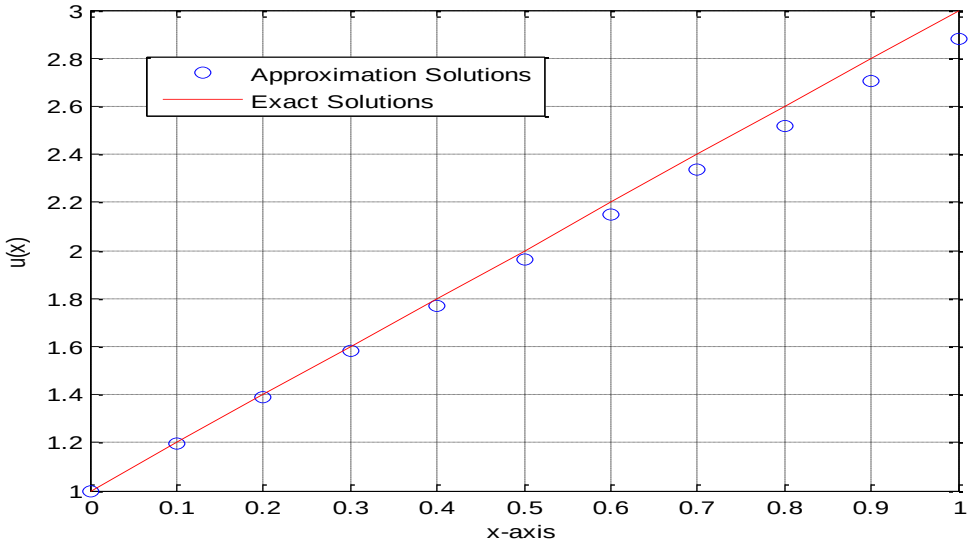

Figure 4.29: The exact and numerical solutions of applying Algorithm 4.2 for equation (4.3).

The CPU time is 0.032915 seconds. Figure 4.30 shows the absolute error resulting of applying algorithm 4.2 on equation (4.3).

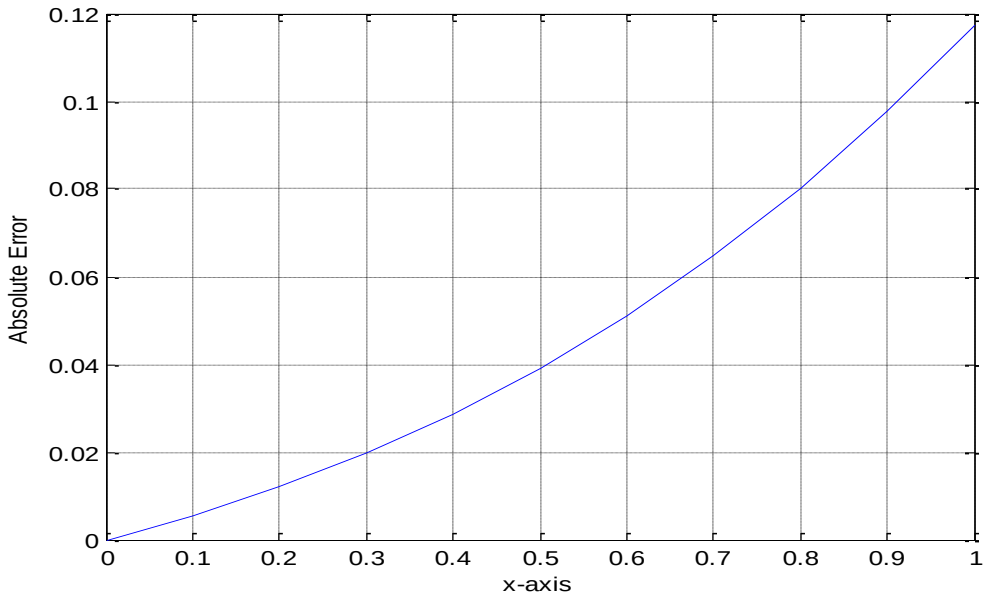

Figure 4.30: the error resulting of applying algorithm 4.2 on equation (4.3).

4.11.2 The fourth order Runge-Kutta method

Table 4.16 shows the exact and numerical solutions when applying Algorithm 4.3 on equation (4.3).

Table 4.16: The exact and numerical solution of applying Algorithm 4.3 for equation (4.3).

x	Analytical solution $u(x) = 1 + 2x$	Approximate solution u_h	Error= $\lvert u - u_h \rvert$
0	1	1	0
0.1	1.2	1.199999792	0.02083×10^{-5}
0.2	1.4	1.399999561	0.04385×10^{-5}
0.3	1.6	1.599999307	0.0693×10^{-5}
0.4	1.8	1.799999026	0.0974×10^{-5}
0.5	2	1.999998715	0.1285×10^{-5}
0.6	2.2	2.199998371	0.1628×10^{-5}
0.7	2.4	2.399997992	0.2008×10^{-5}
0.8	2.6	2.599997572	0.2427×10^{-5}
0.9	2.8	2.799997109	0.2891×10^{-5}
1	3	2.999996596	0.3403×10^{-5}

These results show the accuracy of the fourth order Runge-Kutta method to solve equation (4.3) since the $max\ error$ =0.3403×10^{-5}.

Figure 4.31 shows the exact solution $u(x) = 1 + 2x$ with the approximate solution when $n = 10$.

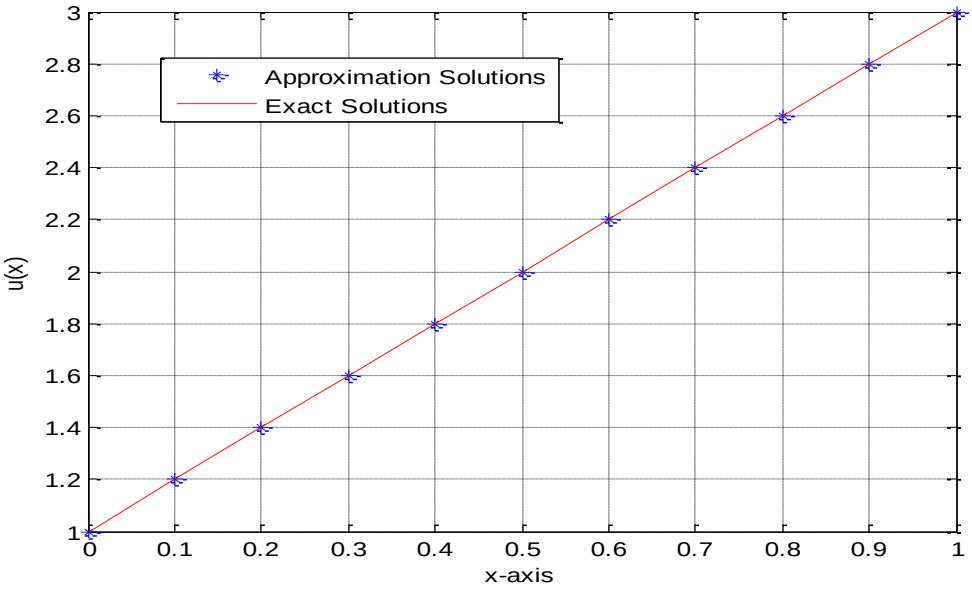

Figure 4.31: The exact and numerical solutions of applying Algorithm 4.3 for equation (4.3).

The CPU time is 0.03540 seconds. Figure 4.32 shows the absolute error resulting of applying algorithm 4.3 on equation (4.3).

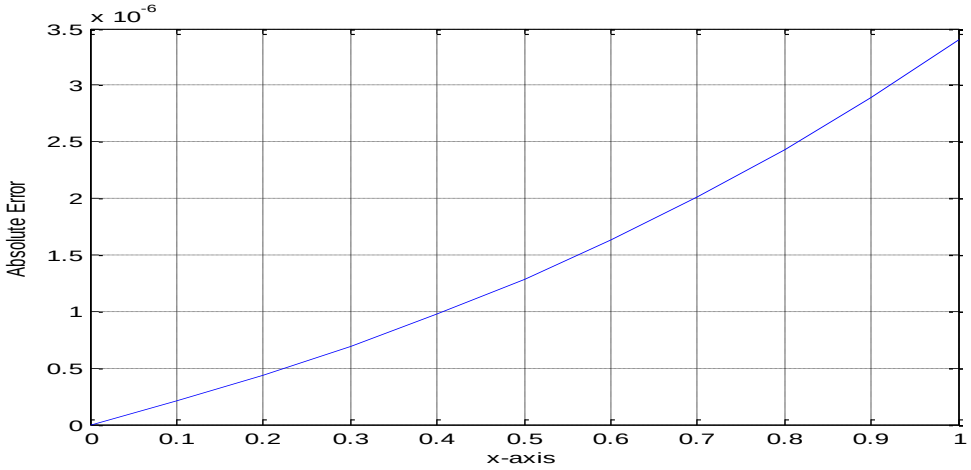

Figure 4.32: the error resulting of applying algorithm 4.3 on equation (4.3).

4.12 The numerical realization of equation (4.3) using the Block method

Table 4.17 shows the exact and numerical results when applying algorithm 4.4 (Block 2) on equation (4.3), and showing the error resulting of using the numerical solutions.

Table 4.17: The exact and numerical solution of applying Algorithm 4.4 for equation (4.3).

x	Analytical solution $u(x) = 1 + 2x$	Approximate solution u_h	Error= $\lvert u - u_h \rvert$
0	1	1	0
0.1	1.2	1.140666667	0.059333333
0.2	1.4	1.364689259	0.035310741
0.3	1.6	1.618577708	0.018577708
0.4	1.8	1.801723035	0.001723035
0.5	2	2.07477211	0.07477211
0.6	2.2	2.234306442	0.034306442
0.7	2.4	2.498720238	0.098720238
0.8	2.6	2.607302089	0.007302089
0.9	2.8	2.83177704	0.03177704
1	3	2.92246989	0.07753011

These results show the accuracy of the method of two block to solve equation (4.3) since the $max\ error$ =0.098720238.

Figure 4.33 shows the exact solution $u(x) = 1 + 2x$ with the approximate solution when $n = 10$.

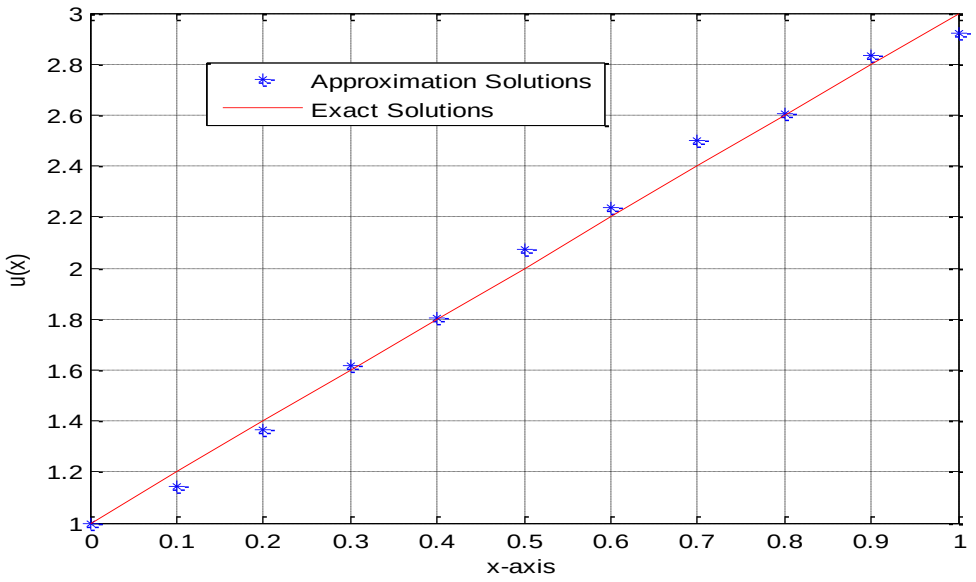

Figure 4.33: The exact and numerical solutions of applying Algorithm 4.4 for equation (4.3).

The CPU time is 0.022529 seconds. Figure 4.34 shows the absolute error resulting of applying algorithm 4.4 on equation (4.3).

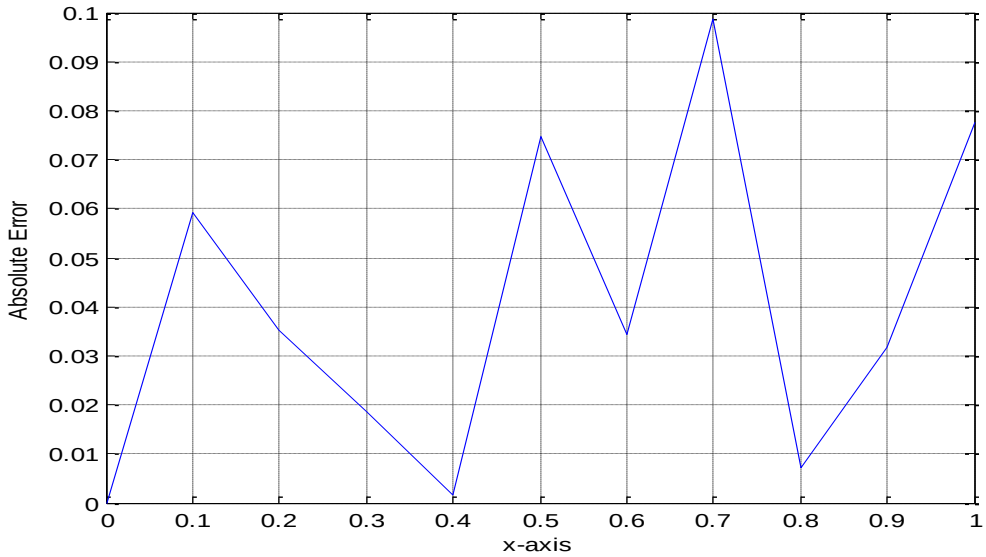

Figure 4.34: the error resulting of applying algorithm 4.4 on equation (4.3).

Table 4.18 shows the exact and numerical results when applying algorithm 4.5 (Block 3) on equation (4.3), and showing the error resulting of using the numerical solutions.

Table 4.18: The exact and numerical solution of applying Algorithm 4.5 for equation (4.3).

x	Analytical solution $u(x) = 1 + 2x$	Approximate solution u_h	Error= $\lvert u - u_h \rvert$
0	1	1	0
0.1	1.2	1.169166667	0.030833333
0.2	1.4	1.385198779	0.014801221
0.3	1.6	1.548159227	0.051840773
0.4	1.8	1.770048273	0.029951727
0.5	2	2.002123835	0.002123835
0.6	2.2	2.15170046	0.04829954
0.7	2.4	2.38525662	0.01474338
0.8	2.6	2.631867325	0.031867325
0.9	2.8	2.794195798	0.005804202
1	3	3.067162265	0.067162265

These results show the accuracy of the method of three block to solve equation (4.3) since the $max\ error$ =0.067162265.

Figure 4.35 shows the exact solution $u(x) = 1 + 2x$ with the approximate solution when $n = 10$.

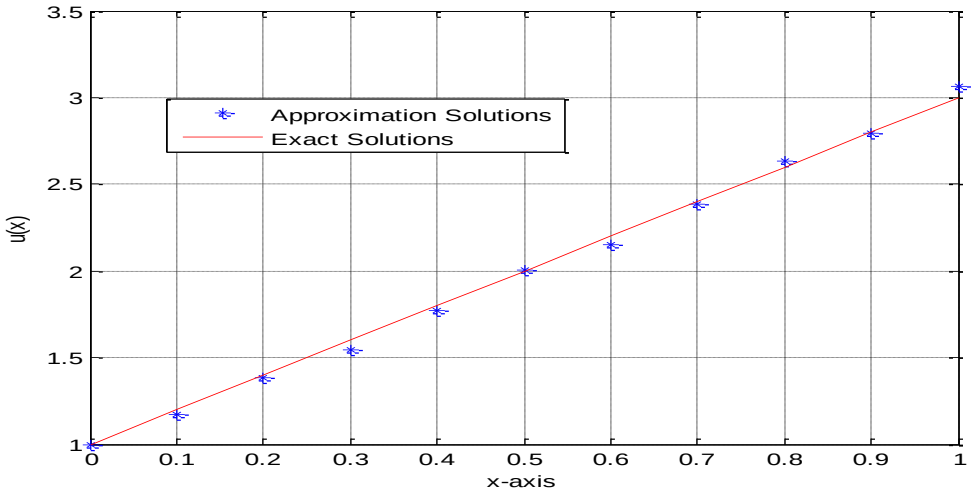

Figure 4.35: The exact and numerical solutions of applying Algorithm 4.5 for equation (4.3).

The CPU time is 0.027388 seconds. Figure 4.36 shows the absolute error resulting of applying algorithm 4.5 on equation (4.3).

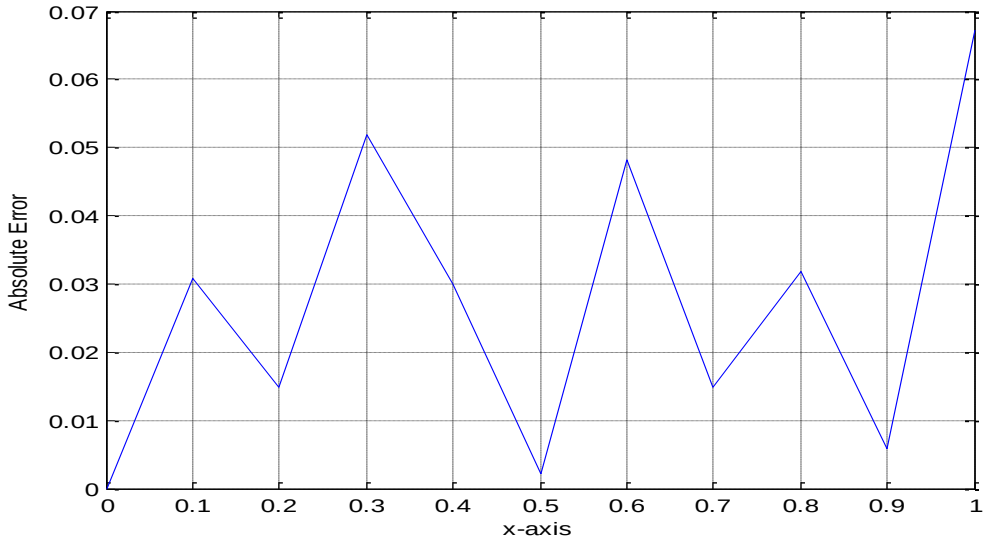

Figure 4.36: the error resulting of applying algorithm 4.5 on equation (4.3).

Conclusions

The numerical results show the following conclusions:

In example 4.3, we have applied the following algorithms: Trapezoidal rule, the second order and fourth order Runge-Kutta method, the two block and three block methods and the collocation and Galerkin methods. We have obtained the following results:

Numerical method	Maximum error	The CPU time
Trapezoidal rule	0.475175×10^{-13}	0.029789 seconds
The Improved Euler	0.117342958	0.032915 seconds
The fourth order Runge-Kutta	0.3403×10^{-5}	0.03540 seconds
The method of two block	0.098720238	0.022529 seconds
The method of three block	0.067162265	0.027388 seconds
The collocation method	0.325677×10^{-6}	0.777991 seconds
The Galerkin method	0.279787×10^{-3}	0.64554 seconds

From the above table we see clearly that Trapezoidal rule is the most efficient technique for solving the integral equation 4.3.

References

[1] S. S. Ahmed, **Numerical solution for Volterra-Fredholm integral equations of the second kind by using least-square technique**, Iraqi Journal of Science, Vol.52, No.4, 2011, PP.504-512.

[2] M. Aigo, **On The Numerical Approximation of Volterra Integral Equations of Second kind Using Quadrature Rules,** International Journal of Advanced Scientific and Technical Research Issue 3 Vol. 1, (2013).

[3] K. Atkinson, **The Numerical Solution of Integral Equations of the Second Kind,** The press Syndicate of the University of Cambridge, United Kingdom, (1997).

[4] C. Baker, **The Numerical Treatment of Integral Equations**, Oxford Univ. Press, (1977).

[5] J. Biazar and M. Pourabd, **A Maple Program for Solving Systems of Linear and Nonlinear Integral Equations by Adomian Decomposition Method**, Int. J. Contemp. Math. Sciences, Vol. 2, 2007, no. 29, 1425 – 1432.

[6] H. Brunner, **Theory and numerical solution of Volterra functional integral equations,** Hong Kong Baptist University, (2010).

[7] H. Brunner, **Collocation Methods for Volterra Integral and Related Functional Differential Equations**, Cambridge University Press, New York,(2004) .

[8] H. Brunner, E. Hairer and S. P. Njersett, **Runge-Kutta Theory for Volterra Integral Equations of the Second Kind**, Mathematics of computation Vol. 39, No. 159 JULY 1982, 147-163.

[9] T.A. Burton, **Volterra Integral and Differential Equations,** 2nd Edition, Elsevier, (2005).

[10] B. Cahlon and L. Nachman, **Numerical Solutions of Volterra Integral Equations with a Solution Dependent Delay**, Journal of mathematical analysis and application 112, 541-562 (1985).

[11] G. M. Campbell and J. T. Day, **A block-by-block method for the numerical solution of Volterra integral equations**, BIT 11 (1971), 120-124.

[12] P. Collins, **Differential and Integral Equations**, Oxford University Press Inc, New York, (2006).

[13] L. Delves and J. Mohammad, **Computational Methods for Integral Equations**, Cambridge University press, (1988).

[14] W. Hackbusch, **Integral Equations: Theory and Numerical Treatment,** Birkhäuser Verlag, Basel, (1995).

[15] A. Isaacson and M. Kirby, **Numerical solution of linear Volterra integral equations of the second kind with sharp gradients**, Journal of Computational and Applied Mathematics 235 (2011) 4283–4301.

[16] A. J. Jerri, **Introduction to Integral Equations with Applications**, John Wiley and Sons, INC, (1999).

[17] R. Kanwal, **Linear Integral Equations, Theory and Technique,** Academic press, INC, New York (1971).

[18] D. Keffer, **Advanced Analytical Techniques for the Solution of Single- and Multi-Dimensional Integral Equations**, University of Tennessee, August, 1999.

[19] R. Kress, **Linear Integral Equations**, 2nd Edition, Springer-Verlag, (1999).

[20] P. Linz, **Analytical and Numerical Methods for Volterra Equations**, Society for Industrial and Applied Mathematics, Philadelphia, (1985).

[21] Maleknejad K, Aghazadeh N, **Numerical solution of Volterra integral equations of the second kind with convolution kernel by using Taylor-series expansion method**, Appl Math Comput 2005;161(3):915–22.

[22] K. Maleknejad , E. Hashemizadeh and R. Ezzati, **A new approach to the numerical solution of Volterra integral equations by using Bernstein's approximation**, Commun Nonlinear Sci Numer Simulat 16 (2011) 647–655.

[23] Marek and Arvet, **Numerical solution of Volterra integral equations with singularities,** Front. Math. China 2013, 8(2): 239–259 DOI 10.1007/s11464-013-0292-z.

[24] Michael.A.Goldberg, **Solution Methods for Integral Equations Theory and Applications,** Plenum Press, New York and London,(1978).

[25] F. Mirzaee, **A computational method for solving linear Volterra integral equations** , Appl. Math. Sci., Vol. 6, 2012, no. 17-20, 807-814.

[26] F. Mirzaee, **Numerical Solution for Volterra Integral Equations of the First Kind via Quadrature Rule**, Applied Mathematical Sciences, Vol. 6, 2012, no. 20, 969 – 974.

[27] M. Mustafa, **Numerical Solution of volterra Integral Equations with Delay Using Block Methods**, AL-Fatih Journal . No . 36, October (2008).

[28] M. Rahman, **Integral Equations and their Applications**, WIT, (2007).

[29] M.M.Rahman, **Numerical Solutions of Volterra Integral Equations Using Galerkin method with Hermite Polynomials,** Pure and Appl. Mathe,(2013).

[30] M.M. Rahman, M.A. Hakim, and M. Kamrul, **Numerical Solutions of Volterra Integral Equations of Second kind with the help of Chebyshev Polynomials**, Pure and Appl. Mathe., Vol. 1,No. 2, 2012, 158-167.

[31] A.Rahman and Sh.Islam, **Numerical Solutions of Volterra Integral Equations Using Legendre polynomials,** GANIT J.Bangladesh Math.Soc.(ISSN 1606-3694) 32 (2012) 29-35.

[32] E. Rakotch, **Numerical Solution of Volterra Integral Equations,** Springer-Verlag, Numer. Math. 20, 271-279, (1973).

[33] A. Ramm, **A Collocation Method for Solving Integral Equations,** Int. J.Computing Science and Mathematics, Vol. 48, No. 10, (2008).

[34] J. Rashidinia, E. Najafi and A. Arzhang, **An iterative scheme for numerical solution of Volterra integral equations using collocation**

method and Chebyshev polynomials, Rashidinia et al. Mathematical Sciences 2012.

[35] J. Saberi-Nadja and M. Heidari, **Solving linear integral equations of the second kind with repeated modified trapezoid quadrature method**. Appl. Math. Comput., 189 (2007), 980- 985.

[36] M. Shafiqul, M. Rahman, **Solutions of Linear and Nonlinear Volterra Integral Equations Using Hermite and Chebyshev Polynomials,** ISSN 2277-3061.

[37] Tahmasbi A**, A new approach to the numerical solution of linear Volterra integral equations of the second kind**, Int J Contemp Math Sci 2008;3(32):1607–10.

[38] W. Wang, **A mechanical algorithm for solving the Volterra integral equation**, Applied Mathematics and Computation 172 (2006) 1323–1341.

[39] A. Wazwaz, **Linear and Nonlinear Integral Equations: Methods and Applications**, Springer Heidelberg, Dordrecht London, (2011).

[40] A. Wazwaz, **A First Course in Integral Equations**, World Scientific Publishing Co. Pte. Ltd., (1997).

[41] S. Zhang, Y.Lin and M. Rao, **Numerical solutions for second-kind Volterra integral equations by Galerkin methods**, Applications of Mathematics, Vol. 45 (2000), No. 1, 19-39.

Appendix

Appendix

Matlab Code for Trapezoidal rule:

```
tic
clc
clear
format long
%Composite trapezoid rule for volterra integral
equations of the
%second kind
%Taken from Atkinson, K.E. "Numerical solution of
ordinary differential
%equations", Wiley (2009)
% The problem is u(x)=2*exp(x)-2-x+int(0,x)(x-
t)u(t)dt
tic
clc
clear
format long
loop = 30;% This is much more than is usually
needed.
b=1;
n=20;
h = b/n;
x = linspace(0,b,n+1);
```

```
fcnf=@(x)(2*exp(x)-2-x);
fvec = fcnf(x);
uvec = zeros(1,n+1);
uvec(1) = fvec(1);
 for i=1:n;
    uvec(i+1) = uvec(i);% The initial estimate for
the iteration.
    kvec = fcnk(x(i+1),x(1:i+1)).*uvec(1:i+1);
    for j=1:loop
        %applying trapezoid rule
        uvec(i+1) = fvec(i+1) + h*(sum(kvec(2:i)
+(kvec(1)+ kvec(i+1))/2);
      kvec(i+1)= fcnk(x(i+1),x(i+1)).*uvec(i+1);
    end
 end
u = uvec;
x = linspace(0,b,n+1);
ue=x.*exp(x);
y=(abs(ue-u));
m=[x',u',ue',y']
plot(x,u,'*',x,ue,'r')
grid on
plot(x,y)
grid on
```

```
toc
```

Matlab Code for Runge-Kutta method of order 2

```
tic
clc
clear
% Runge_Kutta_Method_of_order 2
% The problem is u(x)=2*exp(x)-2-x+int(0,x)(x-t)u(t)dt
%specifyig weights
theta=[0,1] ;
A=[1, 0 ; 0.5, 0.5];
h=.1;
nodes=0:.1:1;
num_inter_pts=(length(nodes)-1)*1; %number of
intermediate points

x=zeros(1, num_inter_pts+length(nodes)); %vector
of nodes and intermediate points

for i=1:length(nodes) %placing node values into x
    x(i+(i-1))=nodes(i);
end

for i=1:length(nodes)-1 % placing intermediate
points into x
```

```
    for j=2:2
        x(i+(i-1)+ j-1)=x(i+(i-1))+h*theta(j);
    end
end
index=zeros(1,length(x)); % keeps track of which
intermediate points are associated with which node
for i=1:length(nodes)-1
    index(i+(i-1):i+(i-1)+1)=i;
end
index(length(index))=length(nodes);
u=zeros(1,length(x)); %vector of solution values
p=2; % order of method
f=@(x)(2*exp(x)-2-x);
for i=1:length(u)
    u(i)=f(x(i));
    m=mod(i, 2);
    k=index(i);
    if m==0
       v=1;
    elseif m==1
        v=0;
    end
    if i~=1||i~=2
        for j=1:k-1
```

```
            for l=1:p
                ind1= find(j==index);
                ind1=ind1(1);

                %applying RK formula
                u(i)=u(i)+h*A(2,l)*fcnk(x(i),
x(ind1+(l-1))).* u(ind1+(l-1));
            end
        end
    end
 if v~=0
        for l=1:v %depends on mod
            ind1= find(index(i)==index);
            ind1=ind1(1);

            %applying RK formula
            u(i)=u(i)+h* A(v,l)*fcnk(x(i),
x(ind1+(l-1))).*x(ind1+(l-1));

        end
    end
end
%obtaining node values
u2=zeros(1,length(nodes));
```

```
for i=1:length(nodes)
    u2(i)=u(i+(i-1));
end
ue=zeros(1,length(nodes));
for i=1:length(nodes)
ue(i)=x(i+(i-1)).*exp(x(i+(i-1)));
end
y=ue-u2;
m=[nodes',u2',ue',y'];
plot(nodes,u2,'o',nodes,ue,'r')
grid on
plot(nodes,y)
grid on
toc
```

Matlab Code for Runge-Kutta method of order 4

```
tic
clc
clear
format long
%Calculates an approximation to a Volterra
Integral Equation of the Second Kind using the
fourth order Runge-Kutta Method
% The problem is u(x) =2*exp(x)-2-x+int(0,x)(x-t)u(t)dt
```

```
%specifying weights:
theta=[0,0.5,0.5,1] ;
A=[0.5, 0, 0, 0; 0, 0.5, 0, 0; 0, 0, 1, 0; (1/6),
(1/3), (1/3), (1/6)];
a=0;
b=1;
h=.1;
nodes=a:h:b ;
num_inter_pts=(length(nodes)-1)*3; % number of
intermediate points

x=zeros(1, num_inter_pts+length(nodes)); %vector
of nodes and intermediate points

for i=1:length(nodes) % placing node values into x
    x(i+3*(i-1))=nodes(i);
end

for i=1:length(nodes)-1 % placing intermediate
points into x
    for j=2:4
        x(i+3*(i-1)+ j-1)=x(i+3*(i-1))+h*theta(j);
    end
end
```

```
index=zeros(1,length(x)); % keeps track of which
intermediate points are % associated with which
node
for i=1:length(nodes)-1
    index(i+3*(i-1):i+3*(i-1)+3)=i;
end
index(length(index))=length(nodes);
u=zeros(1,length(x)); % vector of solution values
p=4; % order of method
 f=@(x)(2*exp(x)-2-x);
for i=1:length(x)
    u(i)=f(x(i));
    m=mod(i, 4);
    k=index(i);
    if m==2
       v=1;
    elseif m==3
        v=2;
    elseif m==0
        v=3;
    elseif m==1
        v=0;
    end
```

```
    if i~=1||i~=2||i~=3||i~=4
        for j=1:k-1
            for l=1:p
                ind1= find(j==index);
                ind1=ind1(1);

                %applying RK formula
                u(i)=u(i)+h*A(4,l)*fcnk(x(i),
x(ind1+(l-1))).*u(ind1+(l-1));
            end
        end
    end
    if v~=0
        for l=1:v %depends on mod
            ind1= find(index(i)==index);
            ind1=ind1(1);

            %applying RK formula
            u(i)=u(i)+h* A(v,l)*fcnk(x(i),
x(ind1+(l-1))).*u(ind1+(l-1));
        end
    end
end
%obtaining node values
```

```
u2=zeros(1,length(nodes));
for i=1:length(nodes)
    u2(i)=u(i+3*(i-1));
end
ue=zeros(1,length(nodes));
for i=1:length(nodes)
ue(i)=x(i+3*(i-1)).*exp(x(i+3*(i-1)));
end
y=ue-u2;

m=[nodes',u2',ue',y']
plot(nodes,u2,'bl*',nodes,ue,'r')
grid on
plot(nodes,y)
grid on
toc
```

Matlab Code for Method of two Blocks

```
tic
clc
clear
format long
% Method of two Blocks to solve Volterra integral
equation of the second kind
```

```
% The problem is u(x)=2*exp(x)-2-x+int(0,x)(x-
t)u(t)dt
b=1;
n=10;
p=2;
h = b/n;
x=linspace(0,1,n+1);
u=zeros(1,n+1);
fcnf=@(x)(2*exp(x)-2-x);
fvec = fcnf(x);
u(1) = fvec(1);
w=zeros(1,n);
w(1)=1;
w(n)=1;
for j=2:n-1
    w(j)=3-(-1)^j;
end
wp=zeros(1,n+1);
wp(1)=1;
wp(n)=1;
for j=2:n+1
    wp(j)=3-(-1)^j;
end
for m=2:n
```

```
    u(m)=fcnf(x(m))+(h/3)*sum(w(1:m-2)
.*fcnk(x(m),x(1:m-2)).*u(1:m-2))
+(h/12)*(5*fcnk(x(m),x(m-1)).*u(m-1)
+8*fcnk(x(m),x(m)).*u(m)-fcnk(x(m),x(m+1))
.*u(m+1));
    u(m+1)=fcnf(x(m+1))+(h/3)*sum(wp(1:m+1)
          .*fcnk(x(m+1),x(1:m+1)).*u(1:m+1));
end
ue=zeros(1,n+1);
for i=1:n+1
ue(i)=x(i).*exp(x(i));
end
y=abs(ue-u);
m=[u',ue',y']
plot(x,u,'*',x,ue,'r')
grid on
plot(x,y,'r')
grid on
toc
```

Matlab Code for Method of three Blocks

```
tic
clc
clear
```

```
format long
% Method of three Blocks to solve Volterra
integral equation of the second kind
% The problem is u(x)=2*exp(x)-2-x+int(0,x)(x-
t)u(t)dt
b=1;
n=20;
p=2;
h = b/n;
x=linspace(0,1,n+1);
u=zeros(1,n+1);
fcnf=@(x)(2*exp(x)-2-x);
fvec = fcnf(x);
u(1) = fvec(1);
w=zeros(1,n+1);
w(1)=1;
w(n-1)=1;
for j=2:n
    m=mod(j,3);
    if m==0
    w(j)=2;
    else
        w(j)=3;
    end
```

```
end

for m=2:n-1
    u(m)=fcnf(x(m))+(3*h/8)*sum(w(1:m-1)
.*fcnk(x(m),x(1:m-1)).*u(1:m-1))
+(h/12)*(5*fcnk(x(m),x(m-1)).*u(m-1)
+8*fcnk(x(m),x(m)).*u(m)-fcnk(x(m),x(m+1))
.*u(m+1));
    u(m+1)=fcnf(x(m+1))+(3*h/8)*sum(w(1:m-1)
.*fcnk(x(m+1),x(1:m-1)).*u(1:m-1))
+(h/3)*(fcnk(x(m+1),x(m-1)).*u(m-1)
+4*fcnk(x(m+1),x(m)).*u(m)+fcnk(x(m+1),x(m+1))
.*u(m+1));
    u(m+2)=fcnf(x(m+2))+(3*h/8)*sum(w(1:m+2)
.*fcnk(x(m+2),x(1:m+2)).*u(1:m+2));
end

ue=zeros(1,n+1);
for i=1:n+1
ue(i)=x(i).*exp(x(i));
end
y=abs(ue-u);
m=[u',ue',y']
plot(x,u,'*',x,ue,'r')
```

```
grid on
plot(x,y)
grid on
toc
```

Matlab Code for Collocation method

```
clc
clear
n=10;
% u(x)=1+x-x^2+int(u(t),0,x)
% collocation method with nine iterations
syms x t
u0=1+x-x^2;
u1=u0+int(1+t-t^2,0,x);
u2=u0+int(1+2*t-1/2*t^2-1/3*t^3,0,x);
u3=u0+int(1+2*t-1/6*t^3-1/12*t^4,0,x);
u4=u0+int(1+2*t-1/24*t^4-1/60*t^5,0,x);
u5=u0+int(1+2*t-1/120*t^5-1/360*t^6,0,x);
u6=u0+int(1+2*t-1/720*t^6-1/2520*t^7,0,x);
u7=u0+int(1+2*t-1/5040*t^7-1/20160*t^8,0,x);
u8=u0+int(1+2*t-1/40320*t^8-1/181440*t^9,0,x);
u9=u0+int(1+2*t-1/362880*t^9-
1626697008263629/29514790517935282585 60*t^10,0,x)
```

```
ua=@(x)(1+2*x-
1626697008263629/5902958103587056517120*x^10-
5205430426443613/10389206262313219470l312*x^11)
ue=@(x)(1+2*x);
uaa=zeros(11,1);
uee=zeros(11,1);
xx=zeros(11,1);
c=0;
for i=1:11
    uaa(i)=ua(c);
    uee(i)=ue(c);
    xx(i)=c;
    c=c+.1;
end
y=(abs(uaa-uee));
[xx uee uaa y];
uee
uaa
y
plot(xx,uaa,'o',xx,uee,'r')
grid on
plot(xx,y)
grid on
toc
```

Matlab Code for Galerkin method with Chebyshev polynomials

```
tic
clc
clear
format long
% degree of polynomial is n
n=10;
F=zeros(n+1,1);
a=zeros(n+1,1);
k=zeros(n+1,n+1);
syms x t
F(1)=int((1+x-x^2),0,1);
F(2)=int((1+x-x^2)*x,0,1);
for i=3:n+1
F(i)=int((1+x-x^2)*Tch(i-1),0,1);
end
F;
k(1,1)=int(1-int(1,0,x),0,1);
k(1,2)=int((1-int(1,0,x))*x,0,1);
for j=3:n+1
    k(1,j)=int((1-int(1,0,x))*Tch(i-1),0,1);
end
k(2,1)=int(x-int(t,0,x),0,1);
```

```
k(2,2)=int((x-int(t,0,x))*x,0,1);
for j=3:n+1
    k(2,j)=int(((x-int(t,0,x))*Tch(j-1)),0,1);
end
for i=3:n+1
k(i,1)=int((Tch(i-1)-(int(Tcht(i-1),0,x))),0,1);
k(i,2)=int((Tch(i-1)-(int(Tcht(i-1),0,x))*x),0,1);
end
for i=3:n+1
    for j=3:n+1
        k(i,j)=int(((Tch(i-1)-(int(Tcht(i-
1),0,x))).*Tch(j-1)),0,1);
    end
end
k
a=inv(k)*F
u=sum(a(1)+a(2).*x+a(3).*Tch(2)+a(4).*Tch(3)+a(5).
*Tch(4)+a(6).*Tch(5)+a(7).*Tch(6)+a(8).*Tch(7)+a(9
).*Tch(8)+a(10).*Tch(9)+a(11).*Tch(10))

ua=@(x)(3378014733232755/1125899906842624-
336084258588602870 91/18446744073709551616*x-
780646173785663/562949953421312*x^2+25641194402273
5/703687441776 64*x*(2*x^2-1)-
```

```
758816794818963/140737488355328*x*(2*x*(2*x^2-1)-
x)+385490043195975/70368744177664*x*(2*x*(2*x*(2*x
^2-1)-x)-2*x^2+1)-
576129132906897/140737488355328*x*(2*x*(2*x*(2*x*(
2*x^2-1)-x)-2*x^2+1)-2*x*(2*x^2-
1)+x)+160094110262051/70368744177664*x*(2*x*(2*x*(
2*x*(2*x*(2*x^2-1)-x)-2*x^2+1)-2*x*(2*x^2-1)+x)-
2*x*(2*x*(2*x^2-1)-x)+2*x^2-1)-
128373776867859/140737488355328*x*(2*x*(2*x*(2*x*(
2*x*(2*x*(2*x^2-1)-x)-2*x^2+1)-2*x*(2*x^2-1)+x)-
2*x*(2*x*(2*x^2-1)-x)+2*x^2-1)-
2*x*(2*x*(2*x*(2*x^2-1)-x)-2*x^2+1)+2*x*(2*x^2-1)-
x)+16954400822283/70368744177664*x*(2*x*(2*x*(2*x*
(2*x*(2*x*(2*x*(2*x^2-1)-x)-2*x^2+1)-2*x*(2*x^2-
1)+x)-2*x*(2*x*(2*x^2-1)-x)+2*x^2-1)-
2*x*(2*x*(2*x*(2*x^2-1)-x)-2*x^2+1)+2*x*(2*x^2-1)-
x)-2*x*(2*x*(2*x*(2*x*(2*x^2-1)-x)-2*x^2+1)-
2*x*(2*x^2-1)+x)+2*x*(2*x*(2*x^2-1)-x)-2*x^2+1)-
4550460827751/140737488355328*x*(2*x*(2*x*(2*x*(2*
x*(2*x*(2*x*(2*x*(2*x^2-1)-x)-2*x^2+1)-2*x*(2*x^2-
1)+x)-2*x*(2*x*(2*x^2-1)-x)+2*x^2-1)-
2*x*(2*x*(2*x*(2*x^2-1)-x)-2*x^2+1)+2*x*(2*x^2-1)-
x)-2*x*(2*x*(2*x*(2*x*(2*x^2-1)-x)-2*x^2+1)-
2*x*(2*x^2-1)+x)+2*x*(2*x*(2*x^2-1)-x)-2*x^2+1)-
```

```
2*x*(2*x*(2*x*(2*x*(2*x*(2*x^2-1)-x)-2*x^2+1)-
2*x*(2*x^2-1)+x)-2*x*(2*x*(2*x^2-1)-x)+2*x^2-
1)+2*x*(2*x*(2*x*(2*x^2-1)-x)-2*x^2+1)-2*x*(2*x^2-
1)+x));
c=0;
xx=zeros(n+1,1);
uap=zeros(n+1,1);
uex=zeros(n+1,1);
for j=1:n+1
    uap(j)=(ua(1-c));
    uex(j)=(1+2*(c));
    xx(j)=c;
    c=c+.1;
end
y=(abs(uap-uex));
[xx uex uap y]
uex
uap
y
plot(xx,uap,'*',xx,uex,'r')
grid on
plot(xx,y)
grid on
toc
```

جامعة النجاح الوطنية
كلية الدراسات العليا

الحلول التحليلية والعددية لمعادلة فولتيرا التكاملية من النوع الثاني

إعداد
فداء عبد العزيز مصطفى سلامة

إشراف
أ.د. ناجي قطناني

قدمت هذه الأطروحة استكمالا لمتطلبات الحصول على درجة الماجستير في الرياضيات المحوسبة بكلية الدراسات العليا في جامعة النجاح الوطنية، نابلس – فلسطين.
2014

الحلول التحليلية والعددية لمعادلة فولتيرا التكاملية من النوع الثاني

إعداد

فداء عبد العزيز مصطفى سلامة

إشراف

أ.د. ناجي قطناني

الملخص

في هذه الأطروحة نحن نركز على الحلول التحليلية والعددية لمعادلة فولتيرا التكاملية من النوع الثاني نظرا لمداها الواسع في الفيزياء والهندسة مثل نظرية المحتملة والمشاكل ديريتشليت، والكهرباء الساكنة، ومشاكل النقل الجسيمات من الفيزياء الفلكية والنظرية المفاعل.

بعد تصنيف هذه المعادلات التكاملية قمنا باستقصاء بعض الطرق التحليلية والعددية لمعادلة فولتيرا التكاملية من النوع الثاني. هذه الطرق التحليلية شملت: طريقة أدومين التحليلية، وطريقة أدومين التحليلية المعدلة، وطريقة التقريبات المتتالية، وطريقة حل السلسلة، وطريقة تحويل معادلة فولتيرا التكاملية إلى معادلة تفاضلية عادية .

الطرق العددية التي نتناولها هي: الطرق التربيعية (طريقة نيسترون) بنوعيها :شبه المنحرف وسمبسون ، طريقة المساقط العمودية بنوعيها: طريقة التجميع وطريقة جاليركين، وطريقة البلوك.

بعض الأمثلة نفذت باستخدام هذه الطرق العددية لحل معادلة فولتيرا التكاملية من النوع الثاني. وأجرينا مقارنة بين النتائج التحليلية والنتائج التقريبية. النتائج التقريبية أظهرت دقتها وقربها من النتائج التحليلية.