An-Najah National University
Faculty of Graduate Studies

Numerical Methods for Solving Differential Algebraic Equations

By
Samer Amin Kamel Abu Sa'

Supervisor
Dr. Samir Matar

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.
2010

Numerical Methods for Solving Differential Algebraic Equations

By
Samer Amin Kamel Abu Sa'

This Thesis was defended successfully on 22/2/2010 and approved by

Defense Committee Members	**Signature**
1. **Dr. Samir Matar** (Supervisor)	.....
2. **Dr. Sa'ed Mallak** (External Examiner)	.....
3. **Dr. Mohammed N. Ass'ad** (Internal Examiner)	.....

Numerical Methods for Solving Differential Algebraic Equations

By
Samer Amin Kamel Abu Sa'

This Thesis was defended successfully on 22/2/2010 and approved by

Defense Committee Members	**Signature**
1. Dr. Samir Matar (Supervisor)	**……..**
2. Dr. Sa'ed Mallak (External Examiner	**……...**
3. Dr. Mohammed N. Ass'ad (Internal Examiner)	**……...**

Dedication

To the memory of my mother

To my lovely daughters: Sama and Sahar

To my family and friends

To department of math

Acknowledgements

First and foremost, alhamdulilah, my gratitude to Allah (swt) for giving me the strength and inspiration to complete this study.

I should like particularly to thank my supervisor, Dr. Sameer Matar, for his insight, invaluable advice, guidance and assistance.

Special word of thanks go to my brother Nasser and his family for their support.

Thanks for Dr, Mohammed Najeeb Ass'ad for his invaluable advice and encouragement.

Thanks also go my father, to his wife Amal, to my brothers Hussam and Salam and their families, to my sisters Ilham, Sawsan, Amal and their families for their support.

Special word of thanks go to my wife Sana' for her kindness and patient and loving support and encouragement.

A word of thanks goes also to my father-in -law Mr. Mahmoud Qadourah for his attention.

Thanks for Mr. Faris Raby'ah and Mr. Mohammed Mohnna for their encouragement.

إقرار

أنا الموقع أدناه، مقدم الرسالة التي تحمل عنوان:

Numerical Methods for Solving Differential Algebraic Equations

طرق عددية لحل المعادلات الجبرية التفاضلية

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

Declaration

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

Student's Name: Samer Amin Abu Sa' **اسم الطالب :سامر أمين كامل ابو صاع**

Signature : **التوقيع:**

Date **التاريخ :**

Table of contents

List of Tables

List of Figures

List of Appendices

Numerical Methods for Solving
Differential algebraic Equations

By
Samer Amin Kamel Abu Sa'
Supervisor
Dr. Samir Matar

Abstract

This study involves the implementation of two numerical methods for solving linear Differential-algebraic equations (DAEs): Power series and Backward Difference Formula (BDF). It aims to facilitate dealing with DAEs by analyzing these two numerical methods, designing simple algorithms, and using MATLAB programs to code the two algorithms. Some numerical examples were implemented by the two methods, tables and figures were also presented in order to make comparisons and conclusions.

This study concluded that Power Series is easier to use, time and effort saving. Further more, it is direct in tackling the problems. Finally, some difficult problems were solved by using this method. Other methods were unable to deal with.

Chapter 1

Introduction

1.1 Introduction

Some problematic applications in science and engineering have been modeled mathematically as a system of Algebraic or Differential equations which is treated numerically and analytically. One kind of these systems is called Differential Algebraic Equations (DAEs).

Differential Algebraic Equations (DAEs) are distinguished by theoretical and numerical difficulties which don't occur in Ordinary Differential Equations (ODE). These difficulties are due to the structure and nature of the problem which is represented by additional algebraic constraints according to elements (variables) of the problem.

Differential Algebraic Equations (DAEs) can be used to describe the evolution of many interesting and important systems in variety of disciplines such as simple pendulum as an example of multibody systems, electrical network as RLC circuit, chemical reactions such as Akzo Nobel problem, and discretization of PDF's such as heat equation. These examples on these areas will be presented later.

Purposes of the study, definition, main types, applications and methodology will be presented in chapter one. Chapter Two will present main definitions and theorems of DAEs. Chapter Three will discuss the numerical methods. The last chapter will discuss some examples, their numerical results and conclusions.

1.2 Purpose of the study

The rationale behind conducting such a study is the scarcity of research in numerical methods for DAEs. So, this study will discuss some numerical methods for solving initial value problems in DAEs. The study will also use algorithms that simplify the complexity of DAEs. In addition, the study will produce a most friendly user MATLAB code that can help any one who is interested in such applications in mathematics, science and technology .

1.3 Definition

Differential Algebraic Equations (DAEs) are systems of differential equations with algebraic constraints that can be expressed in terms of initial value problem such as

$$F(t,x(t),x'(t)) = 0$$

$$G(t,x(t)) = 0 \quad , \quad (1.1)$$

$$x(t_0) = x_0 \, , \; x'(t_0) = x_1$$

or can be expressed in terms of boundary value problem such as

$$F(t,x(t),x'(t)) = 0$$

$$G(t,x(t)) = 0 \quad (1.2)$$

$$B_a x(a) + B_b x(b) = \beta . \; [1]$$

where $t \in R,\ x \in R^n,\ F : R \times R^n \times R^n \to R^n,\ G : R \times R^n \to R^n$

F **represents differential equations which must contain differential terms.** ***G*** **represents algebraic constraints which are equations without differential terms so they may be considered as initial or boundary conditions in ODEs.**

1.4 Main types of DAEs :

None of the currently available numerical techniques work for all DAEs. Some additional conditions, either on the structure of the DAEs and/or the numerical methods, need to be satisfied.

The structural classification of the DAEs can be:

1) Linear constant coefficient DAEs are in the form :

$$A_{n\times n}.\mathbf{x}'_{n\times 1}(t)+B_{n\times n}.\mathbf{x}_{n\times 1}(t)=\mathbf{C}_{n\times 1}(t) \qquad \textbf{(1.3)}$$

where ***A, B*** **are two square matrices of real or complex numbers and** ***t*** **is a real variable. We shall usually assume vectors are real for notational convenience but the results are the same for the complex case.**

2) Linear time varying DAEs

$$A_{n\times n}(t).\mathbf{x}'_{n\times 1}(t)+B_{n\times n}(t).\mathbf{x}_{n\times 1}(t)=\mathbf{C}_{n\times 1}(t) \qquad \textbf{(1.4)}$$

With ***A(t)*** **singular for all** ***t.*** **This system is the general, or fully-implicit linear time varying DAEs.**

3) A special case of (1.4) is the semi-explicit linear DAEs

$$x_1' + B_{11}(t)x_1(t) + B_{12}(t)x_2(t) = f_1(t)$$
$$B_{21}(t)x_1(t) + B_{22}(t)x_2(t) = f_2(t) \qquad \textbf{(1.5)}$$

4) Non-linear constant coefficient DAEs,

$$\boldsymbol{F}(t, x(t), x'(t)) = \boldsymbol{0} \qquad \textbf{(1.6)}$$

5) The general (or *fully-implicit*) non-linear time varying DAEs

$$\boldsymbol{F}(t, x(t), x'(t)) = \boldsymbol{0} \qquad \textbf{(1.7)}$$

6) The linearly implicit or linear in the derivative DAEs

$$A(t, \boldsymbol{x}(t)).\boldsymbol{x}'(t) + f(t, \boldsymbol{x}(t)) = \boldsymbol{0} \qquad \textbf{(1.8)}$$

7) A special case of (1.6) is the semi-explicit non-linear DAE

$$x_1'(t) = f_1(t, x_1(t), x_2(t))$$
$$0 = f_2(t, x_1(t), x_2(t)) \qquad \textbf{(1.9)}$$

Depending on the application, sometimes it can be referred to a system as semi-explicit if it is in the form

$$\boldsymbol{F}(t, x'(t), x(t), y(t)) = \boldsymbol{0}$$
$$\boldsymbol{G}(t, x(t), y(t)) = \boldsymbol{0} \qquad \textbf{(1.10)}$$

where $F_{x'}$ is nonsingular. [8]

1.5 Applications:

Modeling on DAEs contributes a lot of different areas such as systems of rigid bodies, network, electrical circuits, chemical reactions

and discretization of PDEs. Some of widely used applications can be summarized as follows:

1) Simple pendulum:

It can be represented by the system of

$$x'' = -\lambda x$$

$$y'' = -\lambda y - g$$

$$0 = x^2 + y^2 - L^2$$

where (x, y) is the cartesian coordinates of the infinitesimal bar of mass one at the end, L is a length of the bar, and g is the gravitational constant and λ is the force in the bar.[6]

2) RLC Circuit which formulated as $Hx' + Gx = f$ where

$$x = \begin{pmatrix} v_B \\ i_B \end{pmatrix}, f = \begin{pmatrix} e(t)^T & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & i(t)^T & 0 \end{pmatrix}^T$$

$$i_B = \begin{pmatrix} i_E \\ i_C \\ i_R \\ i_L \\ i_S \end{pmatrix}, \quad \mathrm{v}_B = \begin{pmatrix} v_E \\ v_C \\ v_R \\ v_L \\ v_S \end{pmatrix}$$

Where *E*: voltage , *S*: current, *R*: resistor, *C*: capacitor, *L*: inductor

$$H = diag\,(0, C, 0, 0, 0, 0, 0, 0, L, 0, 0)$$

$$G = \begin{pmatrix} I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -I & 0 & 0 & 0 & 0 \\ 0 & 0 & I & 0 & 0 & 0 & 0 & -R & 0 & 0 & 0 \\ -I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & A_E \\ 0 & -I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & A_C \\ 0 & 0 & -I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & A_R \\ 0 & 0 & 0 & -I & 0 & 0 & 0 & 0 & 0 & 0 & A_L \\ 0 & 0 & 0 & 0 & -I & 0 & 0 & 0 & 0 & 0 & A_S \\ 0 & 0 & 0 & -I & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 & 0 & A_E^T & A_C^T & A_R^T & A_L^T & A_S^T & 0 \end{pmatrix}$$

[6]

3) Method of lines (MOL) which replaces the derivatives by discrete difference approximations for solving PDE's , e.g. consider the heat equations $\frac{\partial y}{\partial t} = \frac{\partial^2 y}{\partial x^2}$ defined on the region $0 \le t, 0 \le x \le 1,$ with boundary conditions given for $y(t, 0)$ and $y(t, 1),$ and initial conditions givien for $y(0, x)$.

Taking uniform spatial mesh of Δx, and mesh points

$$x_j = (j+1)\Delta x, 1 \le j \le (\frac{1}{\Delta x}) - 1 = N,$$

and using centered differences, we obtain the semi-explicit DAE in the variables

$$y_i(t) = y(t, x_i)$$

$$y'_j - \frac{y_{j-1} - 2y_j + y_{j+1}}{(\Delta x)^2} = 0, \quad j = 2, ...N - 1 \qquad \textbf{.....[6]}$$

$$y_1 - y(t,0) = 0, \quad y_N - y(t,1) = 0.$$

4)Chemical Akzo Nobel problem:

This problem is of the form

$$M\frac{dy}{dt} = \boldsymbol{f}(y), \quad = y\,\mathbf{(0)} = \mathbf{y}_0, \quad \mathbf{y'(0)} = \mathbf{y}'_0,$$

with $y \in \mathbb{R}^6$, $0 \le t \le 180.$

$$\mathbf{M} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \qquad \boldsymbol{f}(y) = \begin{pmatrix} -2r_1 + r_2 - r_3 - r_4 \\ -\frac{1}{2}r - r_4 - \frac{1}{2}r_5 + F_{in} \\ r_1 - r_2 + r_3 \\ -r_2 + r_3 - 2r_4 \\ r_2 - r_3 + r_5 \\ K_s.y_1.y_4 - y_6 \end{pmatrix}$$

where the r_i and F_{in} are auxiliary variables, given by

$$r_1 = k_1.y_1^4.y_2^{\frac{1}{2}}, \qquad r_2 = k_2.y_3.y_4, \qquad r_3 = \frac{k_2}{k}.y_1.y_5,$$

$$r_4 = k_3.y_1.y_4^2, \qquad r_5 = k_4.y_6^2.y_2^{\frac{1}{2}}, \qquad F_{in} = klA.(\frac{p(CO_2)}{H} - y_2).$$

The values of the parameters $k_1, \cdots k_4, K, klA, p(CO_2)$ and H are

k_1=18.7, k_2 = 0.58, k_3=0.09, k_4=0.42 k_s =115.83,

K =34.4, $p(CO_2)$=0.9 klA=3.3, H=737. **…[9]**

1.6 Methodology

Having discussed the development of BDF, its technique, and the presentation of the most widely code (DASSL), this resulted in designing and developing simple algorithm, and using MATLAB for coding this algorithm (see Appendix B: *BDF.m*). This was manifested in replacing Euler's method with Runge-Kutta fourth order method for system, which computes the starting values. One advantages of our program is that it replaced some complicated loops with more than 370 GOTO statements with a simple easy code.

The second method used is Power Series which considered a modern method. The technique of this method was discussed in details. Power Series utilized Pade' approximation in order to get more accurate results. The emphasis of this study was on designing algorithm and code it by using MATLAB. (see Appendix B: *powerpade.m*)

The two algorithms and programs were applied to different examples. The results will be presented in tables and figures in order to give a significant comparison. The comparison will be demonstrated in one figure and three tables for each variable. The main table contains the numerical solutions and the absolute error for each method. Then, the numerical solution will be followed by a plot. The second table will give the maximum error for each variable. The third

table will give the order of convergence for each variable of each method. Order of convergence will be estimated by

$\lim_{n\to\infty} \frac{|\, p_{n+1} - p \,|}{|\, p_n - p \,|^{\alpha}} = \lambda,$ where λ and α are positive constants. We say that p_n

converges to p of order α, with asymptotic constant error; λ.

Chapter 2

Theory of DAEs

2.1 Introduction

Theory of DAEs has been developed to meet the needs of certain solution to the systems discussed in section 1.4. In this chapter, main definitions and theorems are stated in order to facilitate dealing with DAEs, and to understand the differences between DAEs and ODEs.

2.2 Solvability

One of the important definitions is *solvability* which means, in simple words, the existence and uniqueness of solution on the interval of interest.

Definition

Let I be an open subinterval of $\mathbf{R}$, Ω is a connected open subset of $\mathbf{R}^{2m+1}$, and F is a differentiable function from Ω to $\mathbf{R}^m$. Then the DAE (1.1) is solvable on I in Ω if there is an r - dimensional family of solutions $\phi(t,c)$ defined on a connected open set $I \times \tilde{\Omega}, \tilde{\Omega} \subset \mathbf{R}^r$, such that :

1) $\phi(t,c)$ is defined on all of I for each $c \in \tilde{\Omega}$

2) $(t, \phi(t,c), \phi'(t,c)) \subset \Omega$ for $(t,c) \subset I \times \tilde{\Omega}$

3) If $\psi(t)$ is any other solution with $(t, \psi(t,c), \psi'(t,c)) \in \Omega$,

then $\psi(t) = \phi(t,c)$

4) The graph of ϕ as a function of (t,c) is an $r+1$ - dimensional

manifold.

The definition says that locally there is an *r*-dimensional family of solutions. At any time $t_0 \in I$, the initial conditions form an r-dimensional manifold $\phi(t_0, c)$ and *r* is independent of t_0. The solutions are continuous function of the initial conditions on this manifold (or equivalently of *c*). Since the solutions exist for all $t \in I$ and (3) holds, there are no bifurcations of these solutions. [6]

2.3 Index

As mentioned earlier in Chapter One, DAEs are distinguished by difficulty. One way to simplify dealing with DAEs is to convert it as a system of ODEs, so the measure that will play a key role in the classification and behavior of DAEs is called an index. In order to motivate the definition of index, let us consider the special case of a semi-explicit non-linear DAE (1.9)

$$x_1'(t) = f_1(t, x_1(t), x_2(t))$$

$$0 = f_2(t, x_1(t), x_2(t))$$

If we differentiate the constraint equation $f_2(t, x_1(t), x_2(t)) = 0$ with respect to *t*, we get an implicit ODE of the form

$$f_{2_{x1}}(t,x_1(t),x_2(t))x_1' + f_{2_{x2}}(t,x_1(t),x_2(t))x_2' = -f_{2_t}(t,x_1(t),x_2(t)). \quad \textbf{(2.1)}$$

Substitute $x_1'(t) = f_1(t,x_1(t),x_2(t))$ **in (2.1), we have**

$$f_{2_{x1}}(t,x_1(t),x_2(t))f_1(t,x_1(t),x_2(t)) + f_{2_{x2}}(t,x_1(t),x_2(t))x_2' = -f_{2_t}(t,x_1(t),x_2(t)).$$

that is

$$f_{2_{x2}}(t,x_1(t),x_2(t))x_2' = -f_{2_t}(t,x_1(t),x_2(t)) - f_{2_{x1}}(t,x_1(t),x_2(t))f_1(t,x_1(t),x_2(t)).$$

If $f_{2_{x_2}}$ **is nonsingular, then the following system**

$$x_1'(t) = f_1(t,x_1(t),x_2(t))$$

$$f_{2_{x2}}(t,x_1(t),x_2(t))x_2' = -f_{2_t}(t,x_1(t),x_2(t)) - f_{2_{x1}}(t,x_1(t),x_2(t))f_1(t,x_1(t),x_2(t)).$$

is an implicit ODEs and we say that it has index one. If this is not the case, suppose that with algebraic manipulation and coordinate changes we can rewrite it in the form of (1.9) but with different x's **. Again, we differentiate the constraint equation. If an implicit ODE results, we say that the original problem has index two. If the new system is not an implicit ODE, we repeat the process. The number of differentiation steps required in this procedure is the index.[6]**

Definition: The minimum number of times that all or part of (1.1) must be differentiated with respect to *t* in order to determine x' **as a continuous function of *x, t,* is the *index* of DAE.**

***Example* 1 : take the following DAEs**

$y'_2 = y_1 + f_1(t)$

$0 = y_2 + f_2(t)$

The first equation is ODE, but the seond is a constraint, so differentiate

it will yield

$y'_2 = -f'_2(t)$

using ODE we have

$\Rightarrow y_1 + f_1(t) = -f'_2(t)$

$\Rightarrow y_1 = -f'_2(t) - f_1(t)$

differentiate this equation will give

$y'_1 = -f''_2(t) - f'_1(t)$

so, after two times of differentiation we have the following ODE system

$y'_2 = -f'_2(t)$

$y'_1 = -f''_2(t) - f'_1(t)$.

so, the original DAEs is of index two.

$$\left.\begin{aligned} y_2' &= y_1 + f_1(t) \\ 0 &= y_2 + f_2(t) \end{aligned}\right\} \quad \Rightarrow \quad \begin{cases} y_2' = y_1 + f_1(t) \\ \\ y_1' = -f_2''(t) - f_1'(t) \end{cases} \qquad \textbf{[7]}$$

DAEs ODEs

***Example* 2 *:* Let us discuss the simple pendulum problem:**

$$x'' = -\lambda x$$

$$y'' = -\lambda y - g$$

$$0 = x^2 + y^2 = L^2$$

This system can be transformed as a system of first order as follows:

$x_1 = x$,
$x_2 = x'$,
$x_3 = y$,
$x_4 = y'$,

$x_1' = x_2$,

$x_2' = -\lambda x_1$,

$x_3' = x_4$,

$x_4' = -\lambda x_3 - g$,

$x_1^2 + x_3^2 - L^2 = 0.$

If we want to determine the index for this system, we must differentiate the constraint until we have system of ODEs by taking the following steps:

Step 1

Take

$x_1^2 + x_3^2 - L^2 = 0,$

differentiate implicitly

$2x_1x_1' + 2x_3x_3' = 0,$

that is

$x_1x_2 + x_3x_4 = 0,$ (1)

Step 2

Differentiate equation (1) implicitly, yields

$x_1'x_2 + x_1x_2' + x_3'x_4 + x_3x_4' = 0,$

substitute $x_1' = x_2,\ x_2' = -\lambda x_1,\ x_3' = x_4,\ x_4' = -\lambda x_3 - g$, yield

$x_2^2 - \lambda x_1^2 + x_4^2 - \lambda x_3^2 - gx_3 = 0,$

rearrange

$x_2^2 + x_4^2 - \lambda(x_1^2 + x_3^2) - gx_3 = 0,$ (2)

Step 3

Differentiate equation (2)

$$2x_2x_2' + 2x_4x_4' - \lambda(2x_1x_1' + 2x_3x_3') - \lambda'(x_1^2 + x_3^2) - gx_3' = 0,$$

substitute

$$2\lambda x_2x_1 - 2\lambda x_4x_3 - 2gx_4 - 2\lambda x_1x_2 - 2\lambda x_3x_4 - \lambda' L - gx_4 = 0,$$

rearrange

$$\lambda' = \frac{-3gx_4 - 4\lambda x_3x_4}{L}. \quad [7]$$

We have transform DAEs for simple pendulum problem of the form

$$x'' = -\lambda x$$

$$y'' = -\lambda y - g$$

$$0 = x^2 + y^2 - L^2$$

into the following ODE system

$$x_1' = x_2$$
$$x_2' = -\lambda x_1$$
$$x_3' = x_4,$$
$$x_4' = -\lambda x_3 - g,$$
$$\lambda' = \frac{-3gx_4 - 4\lambda x_3x_4}{L}$$

This ODE system resulted after three differentiation times, so DAEs is of index three. The original system consists of three equations, but the new ODE system consists of five equations with two new variables.

Although, we can sometimes transform DAEs into ODEs, but there are difficulties that may appear as disadvantages for this process:

1) **The exact solution for ODEs is not necessarily a solution for the original DAEs.**

2) **It is not trivial to choose the right initial values that satisfy the original system. [2]**

The concept of index has been introduced in order to quantify the level of difficulty that is involved in solving a given DAEs. So, higher index, greater than one, means that we have more difficulty to solve DAEs. By this concept, ODEs can be classified as 0- index, which agrees with the simplicity of numerical methods that deal directly with ODEs.

2.4 Theory of linear DAEs

Linear DAEs in both types, constant or varying, are best understood class of DAEs systems. In this section, main definitions and theorems are needed to understand the numerical methods in the following chapter.

2.4.1 Matrix pencil

Matrix pencil is magnitude coupled with linear DAEs of the form $A_{n\times n}.\boldsymbol{x}'_{n\times 1} + B_{n\times n}.\boldsymbol{x}_{n\times 1} = \boldsymbol{C}_{n\times 1}$**.**

Definition:

Matrix pencil is the magnitude $\lambda A + B$, λ be a complex parameter. **If determinant of this magnitude is**

not equal to zero then the pencil is said to be regular. The importance of matrix pencil comes from the existence of solution for linear DAEs, this is justified in the following derivation:

Take the linear DAE $A_{n\times n}.\boldsymbol{x}'_{n\times 1} + B_{n\times n}.\boldsymbol{x}_{n\times 1} = \boldsymbol{C}_{n\times 1}$

let $\boldsymbol{x}'$ replaced by $\frac{\boldsymbol{x}_{n+1} - \boldsymbol{x}_n}{h}$, then we have

$$A.\frac{\boldsymbol{x}_{n+1} - \boldsymbol{x}_n}{h} + B\ .\boldsymbol{x}_{n+1} = C$$

multiply by h to get

$$A.\boldsymbol{x}_{n+1} + hB.\boldsymbol{x}_{n+1} - A.\boldsymbol{x}_n = hC$$

factorize $\boldsymbol{x}_{n+1}$

$$(A + hB)\boldsymbol{x}_{n+1} = hC + A.\boldsymbol{x}_n$$

multibly by $(A + hB)^{-1}_{n\times n}$

$$\boldsymbol{x}_{n+1} = (A + hB)^{-1}_{n\times n}.(hC + A\boldsymbol{x}_n)_{n\times 1}$$

This means that $A + hB$ must be invertible, i.e. must be nonsingular.

The following theorem will relate solvability with matrix pencil:

Theorem: The linear DAE is solvable if and only if $\lambda A + B$ is a regular pencil.

Examples :1)Let

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \boldsymbol{x}'(t) + \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \boldsymbol{x}(t) = \boldsymbol{f}(t)$$

This system has a regular matrix pencil because det($\lambda A + B$)=1.

3) let

$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \boldsymbol{x}'(t) + \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \boldsymbol{x}(t) = \boldsymbol{f}(t)$$

has singular matrix pencil because $\det(\lambda A + B) = 0$.

2.4.2 Nilpotency :

Other measure of DAE's difficulty which is nilpotency.

Definition:Let M be a square matrix, k $\in \square$, if M, M ^{2},M ^{3},…, $M^{k-1} \neq 0$ **,**

and M ^{k} =0 then M is said to be nilpotent, k is said to be the nilpotency.

Example:

Let M=$\begin{bmatrix} 0 & 0 \\ -1 & 0 \end{bmatrix}$,

$M^2 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$.

So we can say that M is nilpotent with nilpotency k =2

Theorem: Suppose that $\lambda A + B$ is a regular pencil, *A* and *B* are square

matrices of order n. Then there exists a nonsingular matrices P, Q such that

$$\text{PAQ}=\begin{pmatrix} I & 0 \\ 0 & N \end{pmatrix}, \quad \text{PBQ}=\begin{pmatrix} C & 0 \\ 0 & I \end{pmatrix}$$

where N is a matrix of nilpotency k and I is the identity matrix.

The degree of nilpotency of N in this theorem, namely the integer k is

the index of pencil $\lambda A + B$. It is also the index of the DAE. [5]

2.4.3 Special DAE forms

Definition :The DAEs (1.1) is in Hessenberg form of size r if it can

be written as

$$\begin{bmatrix} I & 0 & \cdots & 0 \\ 0 & I & & \vdots \\ \vdots & & I & \vdots \\ 0 & \cdots & \cdots & 0 \end{bmatrix} \begin{bmatrix} x_1' \\ \vdots \\ \vdots \\ x_r' \end{bmatrix} + \begin{bmatrix} B_{11} & \cdots & B_{1,r-1} & B_{1r} \\ B_{21} & \cdots & B_{2,r-1} & 0 \\ 0 & \cdots & \vdots & \vdots \\ \vdots & \cdots & B_{r,r-1} & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ \vdots \\ x_r \end{bmatrix} = \begin{bmatrix} f_1 \\ \vdots \\ \vdots \\ f_r \end{bmatrix}$$

where x_i are vectors, B_{ij} are matrices, and the product $B_{r,r-1}B_{r-1,r-2}\cdots B_{1r}$ are nonsingular.

The Hessenberg form of size two and three are the most common. The Hessenberg form of size two is

$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1' \\ x_2' \end{bmatrix} + \begin{bmatrix} B_{11} & B_{12} \\ 0 & B_{21} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$$

$\det(B) \neq 0.$

The Hessenberg form of size three is

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} x_1' \\ x_2' \\ x_3' \end{bmatrix} + \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & 0 \\ 0 & B_{32} & 0 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix}$$

$\det(B) \neq 0.$

2.5 Non-linear Hessenberg form

The general DAE system (1.1) can include problems which are not well-defined in mathematical sense, as well as problems which will result in failure for any direct discretization method. Fortunately, many of the higher-index problems can be expressed as a combination of more restrictive structures of ODEs coupled with contraints. One of the more important classes of systems are the *Hessenberg forms*.

Definition The DAEs (1.1) is in non-linear Hessenberg form of size *r* if it is written

$$x_1' = F_1(x_1, x_2, ..., x_r, t)$$

$$x_2' = F_2(x_1, x_2, ..., x_{r-1}, t)$$
$$\vdots$$
$$x_i' = F_i(x_{i-1}, x_i, ..., x_{r-1}, t),\ 3 \leq \text{ i } \leq \text{r-1}$$
$$\vdots$$
$$0 = F_r(x_{r-1}, t)$$

and $(\partial F_r / \partial x_{r-1})(\partial F_{r-1} / \partial x_{r-2}) \cdots (\partial F_2 / \partial x_1)(\partial F_1 / \partial x_r)$ *is nonsingular.*

The Hessenberg form of size two is

$$x_1' = F_1(x_1, x_2, t)$$

$$0 = F_2(x_1, t)$$

with $(\partial F_2/\partial x_1)(\partial F_1/\partial x_2)$ *nonsingular.*

The Hessenberg form of size 3 is

$$x_1' = F_1(x_1, x_2, x_3, t)$$

$$x_2' = F_2(x_1, x_2, t)$$

$$0 = F_3(x_2, t)$$

with $(\partial F_3/\partial x_2)\ (\partial F_2/\partial x_1)(\partial F_1/\partial x_3)$ *nonsingular.*

2.5.1 Main Properties for these forms

1) **Assuming the *Fi* is sufficiently differentiable, the Hessenberg form of size *r* is solvable and has index *r* .**

2) **Many applications appear as versions of the Hessenberg forms such as trajectory control problem of size two and three and some beam deflection problems are of size four[6].**

3) **They are easy to manipulate numerically and analytically.**

2.6 The importance of DAEs

There are several reasons to consider DAEs directly, these reasons can be summarized as follows:

1) **Each variable in DAEs is significant that represents an element of physical problem simulated. So, changing the original model to ODEs may produce less meaningful variables.**

2) **Sometimes it is impossible or time consuming to obtain an explicit or normal ODEs..**

3) **It is easier for the scientist and engineer to explore the effect of modeling changes and parameter variation from original DAEs. [6]**

These reasons were manifested in the simple pendulum problem from the previous chapter. There were two new variables added to the problem in order to get ODE system. Beside this, the original DAEs consists of three equations, whereas, the derived ODE system consists of five equations. In addition, the algebraic manipulations used in the pendulum problem, are stuffy and complicated.

2.7 Review of literature

Several numerical methods have been developed for solving DAEs. However the pioneering numerical methods in the field of DAEs could be presented here:

The code DASSL (Differential Algebraic System SoLver) in Fortran designed by Petzold[6] based on backward difference formula (BDF) to solve general index zero and one DAEs. It will be discussed in detail in the next chapter.

Radau collocation which based on implicit Runge-Kutta method. It solves DAEs of the form $Mx' = f(t, x)$**, where** M **is a constant,**

square matrix which may be singular. The code is applicable to problems of index 1,2,3. The higher-index variables must be identified by the user. [4]

DAETS (Differential Algebraic Equations by Taylor Series) that solves initial value problems for (DAEs) using Taylor series expansion. DAETS designed by Nedialkov and Pryce.[10]

Power series method, which was suggested by Bayram[3], it deals with the DAEs directly by constructing a polynomial to approximate the solution. It will be discussed in detailed in the next chapter.

There are instant solvers for specified DAEs problems which can be available on Mathwork website. For example, ode15s.m, hb1dae.m and dae4.m.

Chapter 3

Numerical Methods

In this chapter, we want to use and explore some numerical methods for solving differential algebraic equations (DAEs).

3.1 Backward difference formula (BDF) Method:

The first general technique for the numerical solution of DAEs , proposed in 1971 by C. W. Gear. This method was initially defined for systems of differential equations of the form

$$\boldsymbol{F}(t, x(t), x'(t)) = \boldsymbol{0}$$

$$\boldsymbol{G}(t, x(t)) = \boldsymbol{0}$$

$$\boldsymbol{x}(t_0) = \boldsymbol{x}_0\,,\ \boldsymbol{x}'(t_0) = \boldsymbol{x}_1\,.$$

This method was soon extended to apply to any fully-implicit DAEs $F(t, x, x') = 0$. **It is considered the most popular multistep method.**

The simplest first order BDF method is the implicit Euler method, which consists of replacing the derivative x' **by a backward difference**

$$F\left(t_n, \boldsymbol{x}_n, \frac{\boldsymbol{x}_n - \boldsymbol{x}_{n-1}}{h}\right) = 0$$

where $h = t_n - t_{n-1}$.

The resulting system of nonlinear equations for x_n **at each time step is then usually solved by Newton's method. The *k*-step (constant**

stepsize) BDF consists of replacing x' by the derivative of the polynomial which interpolates the computed solution at *k*+1 times t_n, t_{n-1},..., t_{n-k}, evaluated at t_n. This yields

$$F(t_n, x_n, \frac{\rho y_n}{h}) = 0,$$

where $\rho y_n = \sum_{i=0}^{k} \alpha_i y_{n-i}$ and α_i, $i = 1,...,k$ are the coefficients of the BDF method.[6]

3.1.1 Software for BDF

The most widely used production code for DAEs that based on BDF is the code DASSL (Differential Algebraic System Solver) which designed by Petzold. It can be used for the solution of DAEs of index zero and one.

DASSL uses a variable stepsize variable order fixed leading coefficient implementation of BDF formula to advance the solution from one time step to the next. The fixed leading coefficient implementation is one way of extending the fixed stepsize BDF methods to variable stepsize. There are three main approaches to extending fixed stepsize multi step method to variable stepsize. These formulations are called fixed coefficient, variable coefficient and fixed leading coefficient. The fixed coefficient method have the property that they can be implemented very efficiently for smooth problems, but suffer from inefficiency, or possible instability, for problems which require stepsize adjustments. The variable coefficient methods are the most stable implementation, but have the disadvantage that they have tend to require more

evaluation of the Jacobian matrix in intervals when the stepsize is changing, and hence are usually considered to be less efficient than a fixed coefficient implementation for most problems. The fixed leading coefficient formulation is a compromise between the fixed coefficient and variable coefficient approaches, offering somewhat less stability, along with usually fewer Jacobian evaluation, than the variable coefficient formulation. [6]

3.1.2 Basic formula used in DASSL

Suppose we have approximation y_{n-i} to the true solution $y(t_{n-i})$ **for** $i = 0,1,...,k$ **where *k* is the order of the BDF method that we are currently planning to use. We would like to find an approximation to the solution at time** t_{n+1}**. First, an initial guess for the solution and its derivative at** t_{n+1} **is formed by evaluating the predictor polynomial and the derivative of the predictor polynomial at** t_{n+1}**. The predictor polynomial** $\omega_{n+1}^{P}(t)$ **is the polynomial which interpolates** y_{n-i} **at the last *k*+1 times,** $\omega_{n+1}^{P}(t_{n-i}) = y_{n-i}$, $i = 0,1,...,k$.

The predicted values for *y* and *y′* at t_{n+1} **are obtained by evaluating** $\omega_{n+1}^{P}(t)$ **and** $\omega_{n+1}^{\prime P}(t)$ **at** t_{n+1}**, so that**

$$y_{n+1}^{(0)} = \omega_{n+1}^{P}(t_{n+1})$$

$$y_{n+1}^{\prime(0)} = \omega_{n+1}^{\prime P}(t_{n+1}).$$

The approximation y_{n+1} **to the solution at** t_{n+1} **which is finally accepted by DASSL is the solution to the corrector formula. The**

formula used is the fixed leading coefficient form of the k^{th} order BDF method. The solution to the corrector formula is the vector y_{n+1} such that the corrector polynomial $\omega_{n+1}^{C}(t)$ and its derivative satisfy the DAE at t_{n+1}, and the corrector polynomial interpolates the predictor polynomial at *k* equally spaced points behind t_{n+1},

$$\omega_{n+1}^{C}(t_{n+1}) = y_{n+1}$$

$$\omega_{n+1}^{C}(t_{n+1} - ih_{n+1}) = \omega_{n+1}^{C}(t_{n+1} - ih_{n+1}),\ 1 \le i \le k\,,$$

$$F(t_{n+1}, \omega_{n+1}^{C}(t_{n+1}), \omega_{n+1}^{\prime C}(t_{n+1})) = 0.$$

The values of the predictor $y_{n+1}^{(0)}, y_{n+1}^{\prime(0)}$ and the corrector y_{n+1} at t_{n+1} are defined in terms of polynomials which interpolate the solution at the previous time steps. These polynomials are represented by in DASSL in terms of modified divided differences of *y*.

The divided difference are defined by the recurrence

$$[y_n] = y_n$$

$$[y_n, y_{n-1}, \ldots, y_{n-k}] = \frac{[y_n, y_{n-1}, \ldots, y_{n-k-1}] - [y_{n-1}, y_{n-2}, \ldots, y_{n-k}]}{t_n - t_{n-k}}.$$

The predictor polynomial is given in terms of the divided difference by

$$\omega_{n+1}^{P}(t) = y_n + (t - t_n)[y_n, y_{n-1}] + (t - t_n)(t - t_{n-1})[y_n, y_{n-1}, y_{n-2}] + \ldots$$

$$+ (t - t_n)(t - t_{n-1}) \cdots (t - t_{n-k-1})[y_n, y_{n-1}, \cdots, y_{n-k}].$$

The corrector polynomial can be formulated as follows

$$\omega_{n+1}^{C}(t)-\omega_{n+1}^{P}(t)=b(t)(y_{n+1}-y_{n+1}^{(0)}),$$

where

$$b(t_{n+1}-ih_{n+1})=0,\quad i=1,2,\cdots,k$$

$$b(t_{n+1})=1.$$

by differentiating at t_{n+1} gives $\alpha_s(y_{n+1}-y_{n+1}^{(0)})+h_{n+1}(y'_{n+1}-y'^{(0)}_{n+1})=0$

Solving for y'_{n+1}, we find that the corrector iteration must solve

$$F(t_{n+1},y_{n+1},y'^{(0)}_{n+1}-\frac{\alpha_s}{h_{n+1}}(y_{n+1}-y_{n+1}^{(0)}))=0.$$

This equation must be solved at each time step using modified Newton's iteration by simplifying the equation as

$F(t, y, \alpha y + \beta) = 0.$

where $\alpha = -\alpha_s / h_{n+1}$

$\beta = y'^{(0)}_{n+1} - \alpha y^{(0)}_{n+1}.$

and $\alpha_s = -\sum_{j=1}^{k} \frac{1}{j}$

so the iterative formula will be

$y^{(m+1)} = y^{(m)} - G^{-1} F(t, y^{(m)}, \alpha y^{(m)} + \beta)$,

with $y^{(0)}$ can be taken from pridector polynomial,

and $G = \alpha \frac{\partial F}{\partial y'} + \frac{\partial F}{\partial y}$.[6]

3.1.3 Algorithm

This section presents a new algorithm that facilitates dealing with linear DAEs, this algorithm based on BDF with fixed leading coefficient, fixed step size, variable order polynomial.

BDF method for solving linear DAEs

***Input* : system of ODEs with initial conditions**

***Output* : approximated values**

Step 1 **evaluate starting values using Runge Kutta 4th order method for system**

Step 2 **repeat step 3-5 until end of interval**

Step 3 **interpolate the previous *k* points for each *y***

Step 4 **substitute t_{n+1} in the polynomial of degree *k-1*,** $2 \le k \le 6$

Step 5 **check consistency**

Step 6 **save new y_{n+1}**

See flowchart at Appendix A

3.2 Power Series Method

Assume that differential algebraic equations (DAEs) has the form (1.1) with initial values $y(x_0) = y_0, \quad y'(x_0) = y_1$ **which are consistent , i.e.**

$$F(y_0, y_1, x_0) = 0$$

$$G(y_0, x_0) = 0$$

The solution of (1.1) can be assumed that $v = y_0 + y_1 x + e x^2$ **(3.1)**

where *e* is a vector function which is the same size as y_0 and y_1**.**

Substitute (3.1) into (1.1) will yield vector function which is equal to zero, then choose appropriate terms in order to produce linear equations of *e* in which can be written in the form S*e* =T , where S is constant square matrix and T is constant vector, solve this system for *e*, return to (3.1) with new value of *e*. Add new term to *y* and repeat the above steps until achieve specified number of iterations. The resulting power series can be transformed into pade' series, this

transformation will be considered as an elective step, in order to get better solution than Power series. The last step will substitute step size *h* into pade' approximant function and compare it with exact solution.

3.2.1 Construction of Power Series

Let $f(x) = f_0 + f_1 x + f_2 x^2 + \cdots + (f_n + p_1 e_1 + \cdots + p_m e_m) x^n$

where $p_1, p_2, \cdots, p_m$ are constants. $e_1, e_2, \cdots, e_m$ are bases of vector e, m is the size of vector e, y is a vector with m elements, each element can be represented by the power series of the form $y_i = y_{i,0} + y_{i,1} x + y_{i,2} x^2 + \cdots + e_i x^n$,

where y_i is the ith element of y. Substituting y_i into (1.1) will get the following:

$$y_i = (f_{i,n} + p_{i,1} e_1 + \cdots + p_{i,m} e_m) x^{n-j} + Q(x^{n-j+1}),$$

where f_i is the ith element of (1.1). The linear equation can be determined as follows:

$$S_{i,j} = p_{i,j}, \quad T_{i,j} = -f_{i,n}.$$

Solving this linear equation, we have e_i *(i=1,…,m)*. Substituting e_i into y_i *(i=1,…,m)* which are polynomials of degree n. Repeating this procedure will produce the arbitrary order power series that represents the solution for DAEs (1.1). The numerical solution for DAEs could be achieved by substituting values of x with specified step size h such that $x = x_0 + h$. [3]

Some rules that followed in order to gain good value of $\underline{e}$ are :

1) **If the system of size *m,* then vector *f* resulted from substitution has *m* elements, each element must equal zero, so from this equation we can extract linear equation that enables us to compute one e_i . In other words, it is supposed that the best situation that every e_i estimated from equation $f_i=0$ where $i=1,\ldots,m$.**

2) **Assume that the new term will be added to the power series has a degree that is equal to *n,* then the higher order terms can be neglected except the terms of degree *n* or *n-1*.**

3) **The linear equation of *e* result from factorization x^n.**

4) **If it is impossible that the system achieve rule 1, so it is allowed that two equations, say, f_i, f_j can be embedded into one equation to extract linear equation in order to evaluate e_i *or* e_j**

3.2.2 Algorithm

Power series method for solving linear DAEs

Input **square matrices A and B, vector function C, initial values y_0 and y_0', number of iterations, exact solutions.**

Output **power series.**

***Step* 1 assume** $y_i = y_{i,0} + y_{i,1}x + y_{i,2}x^2 + \cdots + e_i x^n$**, *i*=1…size**

***Step* 2 repeat the 3-6 steps until number of iteration**

Step* 3 substitute y in *Ay'+By=C

Step 4 **construct linear system S*e* =T**

Step 5 **solve the system for *e***

Step 6 **add new term for each y_i , *i*=1…size**

Step 7 **transform every v_i ,(*i*=1…size), into Pade' approximant**

Step 8 **estimate numerical solution**

Step 9 **compare numerical solution with the exact solution.**

***See flowchart at Appendix* A**

3.2.3 Rational function approximation

The class of algebraic polynomials has some distinct advantages for use in approximation:

1. **there are sufficient number of polynomials to approximate any continuous function on a closed interval to within an arbitrary tolerance;**
2. **polynomials are easily evaluated at arbitrary values; and**
3. **the derivatives and integrals of polynomial exist and are easily determined.**

The disadvantages of using polynomial for approximation is their tendency for oscillation. This often causes error bounds in polynomial approximation to significantly exceed the average approximation error, since error bounds are determined by the maximum approximation error. We now consider methods that spread

the approximation error more evenly over the approximation interval. These techniques involve rational functions.

A rational function *r* of degree *N* has the form $r(x) = \frac{p(x)}{q(x)}$ **,**

where *p(x)* and *q(x)* are polynomials whose degrees sum to *N* .

Since every polynomial is rational function (simply let *q(x)*=1), approximation by rational functions gives results that are no worse than approximation by polynomials. However, rational functions whose numerator and denominator have the same or nearly the same degree generally produce approximation results superior to polynomial methods for the same amount of computational effort. (This statement is based on the assumption that the amount of computation effort required for division is approximately the same as for multiplication). Rational functions have the added advantage of permitting efficient approximation of functions with infinite discontinuities near, but out side, the interval of approximation. Polynomial approximation is generally unacceptable in this situation.

Suppose r is a rational function of degree *N=n+m* of the form

$$r(x) = \frac{p(x)}{q(x)} = \frac{p_0 + p_1 x + \cdots + p_n x^n}{q_0 + q_1 x + \cdots + q_m x^m},$$

that is used to approximate function *f* on a closed interval *I* containing zero. For *r* to be defined at zero requires that $q_0 \neq 0$. In fact, we can assume that $q_0=1$, for if this is not the case we simply replace *p(x)* by $p(x)/q_0$ and *q(x)* by $q(x)/q_0$. Consequently, there are *N*+1 parameters $q_1, q_2, \ldots, q_m, p_0, p_1, \ldots, p_n$ available for the approximation of *f* by *r*.

Pade' approximation technique

The pade' approximation technique is the extension of Taylor polynomial approximation to rational functions, choose the *N*+1 parameters so that $f^{(k)}(0) = r^{(k)}(0),$ **for each *k=0,1,…,N*. When *n=N* and m=0, the Pade' approximation is just the *N*th Maclaurin polynomial.**

Consider the difference

$$f(x) - r(x) = f(x) - \frac{p(x)}{q(x)} = \frac{f(x)q(x) - p(x)}{q(x)} = \frac{f(x)\sum_{i=0}^{m} q_i x^i - \sum_{i=0}^{n} p_i x^i}{q(x)},$$

and suppose *f* has the Maclaurin series expansion

$f(x) = \sum_{i=0}^{\infty} a_i x^i$. Then

$$f(x) - r(x) = \frac{\sum_{i=0}^{\infty} a_i x^i \sum_{i=0}^{m} q_i x^i - \sum_{i=0}^{n} p_i x^i}{q(x)}.$$

the object is to choose the constants

$q_1, q_2, \cdots, q_m$ and $p_0, p_1, \cdots, p_n$ so that $f^{(k)}(0) - r^{(k)}(0) = 0,$ for each $k = 0, 1, \cdots, N$.

Using cross-multiplying yields

$$(a_0 + a_1 x + \cdots)(1 + q_1 x + \cdots + q_m x^m) - (p_0 + p_1 x + \cdots + p_n x^n) = 0,$$

has no terms of degree less than or equal *N*. By expansion (2.3) and equating the coefficient will produce *N*+1 linear equations which can be summarized by $\sum_{i=0}^{k} a_i q_{k-i} = p_k$, $k = 0, 1, \cdots, N$.

Using Gauss elemination and backward substitution method for solvng this system and find the values of $q_0, q_1,\cdots,q_m$, $p_0, p_1,\cdots, p_n$.[11]

Example:

Let $f(x) = 1 - x + 1.5x^2 - \frac{1}{6}x^3 - \frac{1}{8}x^4$

By applying pade' approximation we get the following rational functions

$(2,2)$Pade' is $r_1(x) = \frac{p_1(x)}{q_1(x)} = \frac{1 - \frac{41}{50}x + \frac{427}{300}x^2}{1 + \frac{9}{50}x + \frac{31}{300}x^2}$

and (3,1) Pade' is $r_2(x) = \frac{1 - \frac{7}{4}x + \frac{9}{4}x^2 - \frac{31}{24}x^3}{1 - \frac{3}{4}x}$.

By substitution in r_1 and r_2 over [0,1] we have:

Table (3.1): Pade' approximations

X	***f(x)***	**(2,2)Pade' approx. $r_1(x)$**	**(3,1)Pade' approx. $r_2(x)$**
0.0	**1.0000**	**1.0000**	**1.0000**
0.1	**0.9148**	**0.9148**	**0.9148**
0.2	**0.8585**	**0.8585**	**0.8584**
0.3	**0.8295**	**0.8296**	**0.8292**
0.4	**0.8261**	**0.8266**	**0.8248**
0.5	**0.8464**	**0.8476**	**0.8417**
0.6	**0.8878**	**0.8910**	**0.8745**
0.7	**0.9478**	**0.9548**	**0.9146**
0.8	**1.0235**	**1.0370**	**0.9467**
0.9	**1.1115**	**1.1358**	**0.9412**
1.0	**1.2083**	**1.2494**	**0.8333**

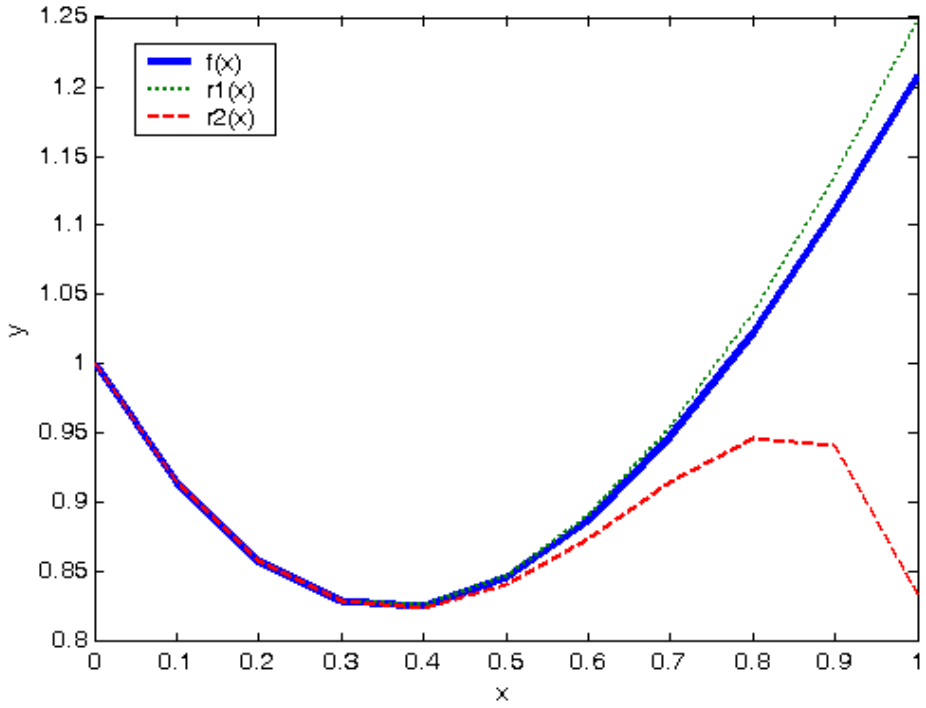


Fig. (3.1): Two Pade' approximations for f(x).

Either approximations r1 or r2 may give better solution in comparison with the exact solution than f. For this reason, Pade' approximation is considered an elective step.

Chapter 4

Examples and Results

4.1 Execution some examples

This section will apply the above algorithms on different linear DAEs as test problems. For each example we display tables and illustrative graphs resulted by Matlab code. The detailed execution will be shown in Appendix C, and m-files are given in Appendix B

Example (1): **Consider the following ODE index-0 system**

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} v_1' \\ v_2' \end{bmatrix} + \begin{bmatrix} 4 & 2 \\ -3 & -1 \end{bmatrix}\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} \cos(t) + 4\sin(t) \\ -3\sin(t) \end{bmatrix} \tag{4.1}$$

with initial values

$$\begin{bmatrix} v_1'(0) \\ v_2'(0) \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \quad \begin{bmatrix} v_1(0) \\ v_2(0) \end{bmatrix} = \begin{bmatrix} 3 \\ -1 \end{bmatrix}$$

with exact solution $\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 2e^{-t} - 2e^{-2t} + \sin(t) \\ -3e^{-t} + 2e^{-2t} \end{bmatrix}$ on the intreval [0,1]

By running powerpade2.m and BDF.m with h=0.1, we have numerical solutions which can be presented as follow:

Table (4.1.1) :Numerical results for y_1

t	Exact	Power series by (1,3)Pade	Error= |Power-exact|	BDF	Error= |BDF –exact|
0	0	0	0	0	0
0.1	0.2720	0.2722	1.3067e-004	0.2720	5.3756e-006
0.2	0.4955	0.4963	8.3278e-004	0.4955	9.0564e-006
0.3	0.6795	0.6818	2.2848e-003	0.6795	1.1513e-005
0.4	0.8314	0.8359	4.5128e-003	0.8314	1.3099e-005
0.5	0.9567	0.9643	7.5577e-003	0.9567	1.4076e-005
0.6	1.0599	1.0714	1.1551e-002	1.0599	5.5123e-005
0.7	1.1442	1.1609	1.6739e-002	1.1443	7.5302e-005
0.8	1.2122	1.2357	2.3477e-002	1.2117	4.7326e-004
0.9	1.2659	1.2981	3.2208e-002	1.2644	1.4797e-003
1.0	1.3066	1.3500	4.3441e-002	1.3054	1.1399e-003

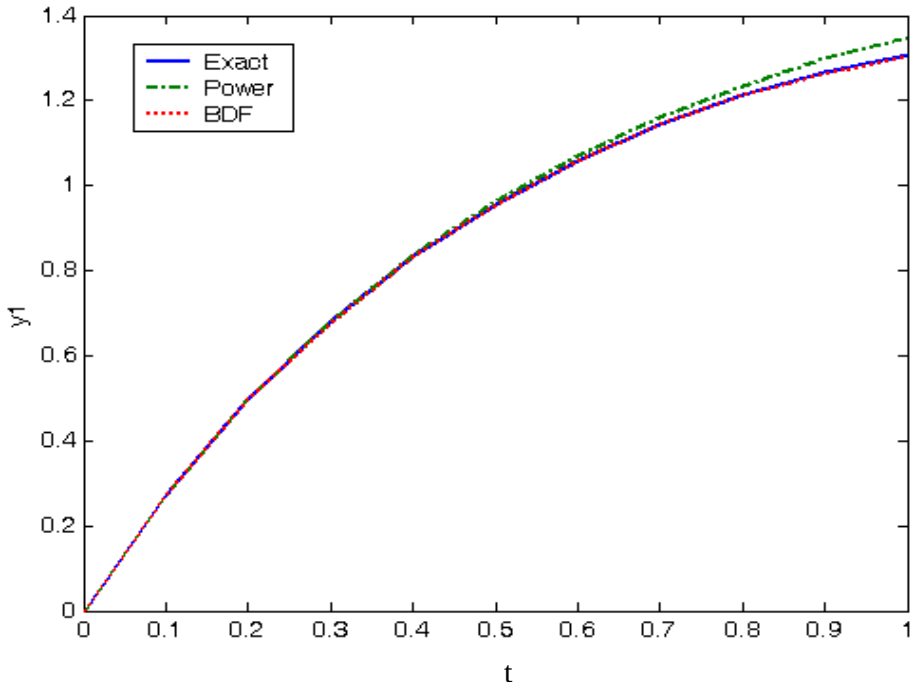


Fig. (4.1.1) : Plot for y_1

Table (4.1.2): Numerical results for y_2

t	Exact	Power series by (1,7)Pade	Error= |Power-exact|	BDF	Error= |BDF –exact|
0.0	-1.0000	-1.0000	0	-1.0000	0
0.1	-1.0771	-1.0771	1.1906e-010	-1.0770	5.2614e-006
0.2	-1.1156	-1.1156	1.3740e-008	-1.1155	8.8381e-006
0.3	-1.1248	-1.1248	2.0670e-007	-1.1248	1.1200e-005
0.4	-1.1123	-1.1123	1.3402e-006	-1.1123	1.2699e-005
0.5	-1.0838	-1.0838	5.4607e-006	-1.0838	1.3595e-005
0.6	-1.0440	-1.0441	1.6555e-005	-1.0441	5.4640e-005
0.7	-0.9966	-0.9966	4.0891e-005	-0.9966	8.6111e-005
0.8	-0.9442	-0.9443	8.6895e-005	-0.9438	3.9697e-004
0.9	-0.8891	-0.8893	1.6458e-004	-0.8879	1.2272e-003
1.0	-0.8330	-0.8333	2.8464e-004	-0.8324	5.4471e-004

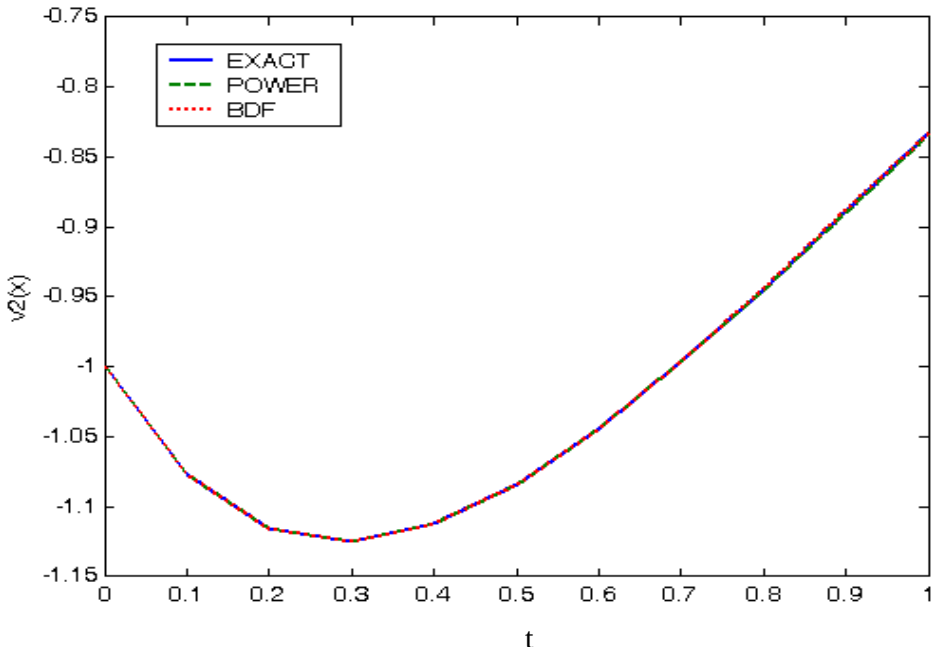


Fig. (4.1.2):Plot for y_2

Table(4.1.3): Table of maximum error

Method / Variable	Power Series	BDF
y_1	4.3441e-002	1.4797e-003
y_2	2.8464e-004	1.2272e-003

Table (4.1.3) shows that the BDF is more accurate for y_1, but Power series is more accurate for y_2.

The following table will display the rate of convergence. We apply the formula of convergence, $\lim_{n\to\infty}\frac{|p_{n+1}-p|}{|p_n-p|^{\alpha}}=\lambda,$ for two trials of α=1 and α= 2, and then we estimate the range of λ = max λ- min λ.

Table (4.1.4): Order of convergence

Method	λ for y_1									Range of λ
Power	1.8233	1.3738	1.2260	1.1536	1.1111	1.0835	1.0644	1.0505	1.0400	0.7833
BDF	1.8217	1.3713	1.2235	1.1507	1.1079	.0796	1.0589	1.0435	1.0324	0.7893
λ for y_2										
Power	1.4994	1.0796	0.8998	0.7462	0.5263	0.0771	16.3824	1.9874	1.5059	16.3053
BDF	1.8233	1.3738	1.2260	1.1536	1.1111	1.0835	1.0644	1.0505	1.0400	16.4523

The best rate of convergence (Table 4.1.4) is achieved by taking α=1.

Since the range of λ for Power series is slightly less than BDF, so we can say that Power series has better convergence than BDF.

Example (2) :

$$\begin{bmatrix} 1 & -t \\ 0 & 0 \end{bmatrix}\begin{bmatrix} v_1' \\ v_2' \end{bmatrix} + \begin{bmatrix} 1 & -(1+t) \\ 0 & 1 \end{bmatrix}\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0 \\ \sin t \end{bmatrix} \qquad (4.2)$$

with initial values

$$\begin{bmatrix} v_1(0) \\ v_2(0) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} v_1'(0) \\ v_2'(0) \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \quad \text{where } 0 \le t \le 1.$$

The exact solution is

$$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} e^{-t} + t\sin t \\ \sin t \end{bmatrix}.$$

This system can be reformulated into ODEs as follows:

$$y_1' = t\cos(t) - y_1 + (1+t)y_2$$

$$y_2' = \cos(t)$$

So, system(4.2) can be considered as index-1 DAEs.

By running powerpade2.m and BDF.m with h=0.1, we have numerical solutions which can be presented as follow:

Table (4.2.1): Numerical solution of y_1

t	Exact	Power series by (3,3)Pade'	Error= |Power-exact|	BDF	Error= |BDF –exact|
0	1.0000	1.0000	0	1.0000	0
0.1	0.9148	0.9148	2.7016e-006	0.9148	2.1843e-007
0.2	0.8585	0.8585	2.0447e-005	0.8585	3.9469e-007
0.3	0.8295	0.8295	6.6964e-005	0.8295	5.3065e-007
0.4	0.8261	0.8262	1.5842e-004	0.8261	6.2820e-007
0.5	0.8462	0.8466	3.1812e-004	0.8462	6.8928e-007
0.6	0.8876	0.8882	5.8196e-004	0.8876	1.2598e-005
0.7	0.9475	0.9485	1.0049e-003	0.9475	8.5201e-005
0.8	1.0232	1.0249	1.6682e-003	1.0232	2.2568e-005
0.9	1.1116	1.1143	2.6874e-003	1.1121	5.8026e-004
1.0	1.2094	1.2136	4.2199e-003	1.2117	2.3039e-003

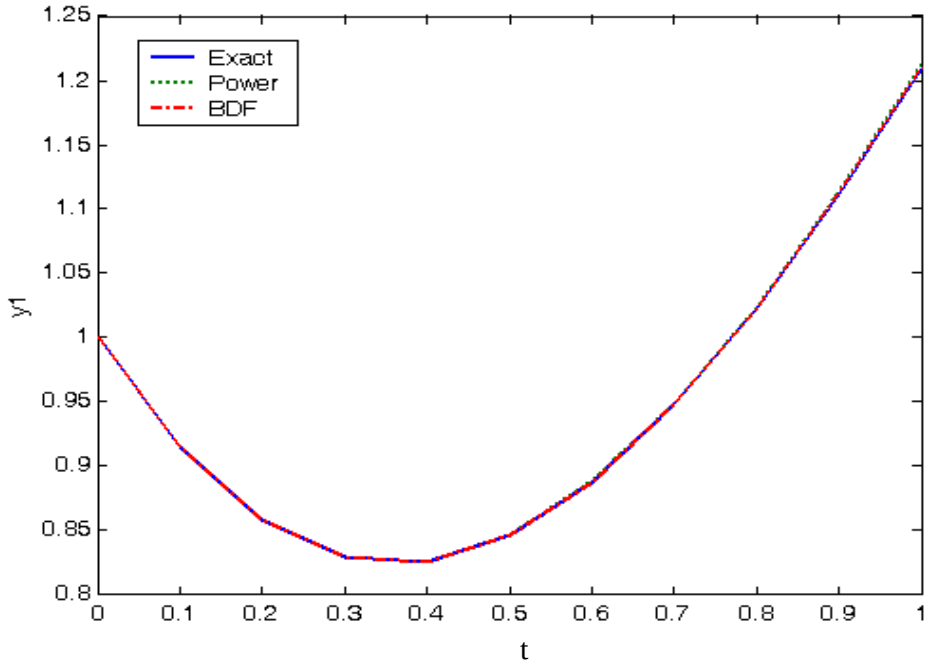


Fig. (4.2.1) : Graph of y_1

Table(4.2.2): Numerical solution for y_2

t	Exact	Power series by (5,1)Pade'	Error= |Power-exact|	BDF	Error= |BDF –exact|
0	0	0	0	0	0
0.1	0.0998	0.0998	1.9839e-011	0.0998	3.4675e-009
0.2	0.1987	0.1987	2.5383e-009	0.1987	6.9003e-009
0.3	0.2955	0.2955	4.3339e-008	0.2955	1.0264e-008
0.4	0.3894	0.3894	3.2436e-007	0.3894	1.3525e-008
0.5	0.4794	0.4794	1.5447e-006	0.4794	1.6652e-008
0.6	0.5646	0.5646	5.5266e-006	0.5646	3.8857e-005
0.7	0.6442	0.6442	1.6229e-005	0.6440	2.0335e-004
0.8	0.7174	0.7174	4.1242e-005	0.7177	3.0163e-004
0.9	0.7833	0.7834	9.3840e-005	0.7855	2.2069e-003
1.0	0.8415	0.8417	1.9568e-004	0.8467	5.2333e-003

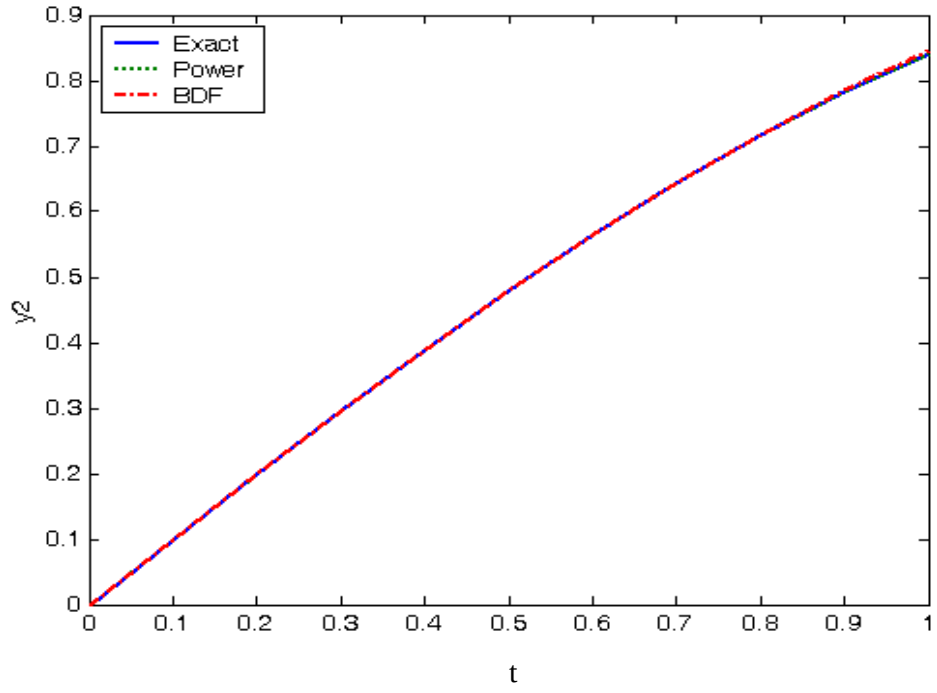


Fig.(4.2.2): Plot of y_2

Table(4.2.3): Maximum error

Method / Variable	Power Series	BDF
y_1	4.2199e-003	2.3039e-003
y_2	1.9568e-004	5.2333e-003

BDF is more accurate for y_1, but Power series is more accurate for y_2.

Table (4.2.4): Order of Convergence with α=1.

Method	λ for y_1									Range of λ
Power	0.9706	0.9844	0.9982	1.0112	1.0225	1.0319	1.0392	1.0442	1.0470	0.0764
BDF	0.9706	0.9844	0.9981	1.0110	1.0224	1.0317	1.0389	1.0439	1.0472	0.0766
λ for y_2										
Power	1.0899	1.0808	1.0725	1.0648	1.0576	1.0509	1.0445	1.0384	1.0327	0.0572
BDF	1.0899	1.0808	1.0725	1.0648	1.0576	1.0507	1.0448	1.0395	1.0343	0.0556

We can deduce that the two methods have the same rate of convergence.

Example (3) **: Let us consider the following DAEs;**

$$\begin{bmatrix} 1 & -t & t^2 \\ 0 & 1 & -t \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_1' \\ v_2' \\ v_3' \end{bmatrix} + \begin{bmatrix} 1 & -(t+1) & t^2+2t \\ 0 & -1 & t-1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \sin(t) \end{bmatrix} \quad (4.3)$$

with initial value $v(0) = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$. where $0 \le t \le 1$

The exact solution is

$v_1(t) = e^{-t} + te^t$,

$v_2(t) = e^t + t\sin(t)$,

$v_3(t) = \sin(t)$.

This system can be reformulated into ODEs as follows:

$y_1' = (2t+1)y_2 - y_1 - (2t^2+t)\sin(t)$

$y_2' = y_2 + t\cos(t) - (t-1)\sin(t)$

$y_3' = \cos(t)$

with $y(0) = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$.

This formulation needs one differentiation time in order to produce ODEs, so (4.3) is index-1.

Numerical solution for this system using Power series method and BDF method can be presented in the following tables and figures.

Table (4.3.1): Numerical solution for y_1

t	Exact	Power series by (4,4)Pade'	Error= |Power-exact|	BDF	Error= |BDF –exact|
0.0	1.0000	1.0000	0	1.0000	0
0.1	1.0154	1.0154	5.0082e-008	1.0154	7.4657e-007
0.2	1.0630	1.0630	8.5244e-007	1.0630	1.4138e-006
0.3	1.1458	1.1458	4.7306e-006	1.1458	1.9942e-006
0.4	1.2670	1.2671	1.6827e-005	1.2671	2.4792e-006
0.5	1.4309	1.4309	4.7246e-005	1.4309	2.8601e-006
0.6	1.6421	1.6422	1.1453e-004	1.6420	6.5949e-005
0.7	1.9062	1.9065	2.5095e-004	1.9058	4.2144e-004
0.8	2.2298	2.2303	5.1017e-004	2.2282	1.5202e-003
0.9	2.6202	2.6212	9.7809e-004	2.6161	4.1612e-003
1.0	3.0862	3.0879	1.7878e-003	3.0766	9.6031e-003

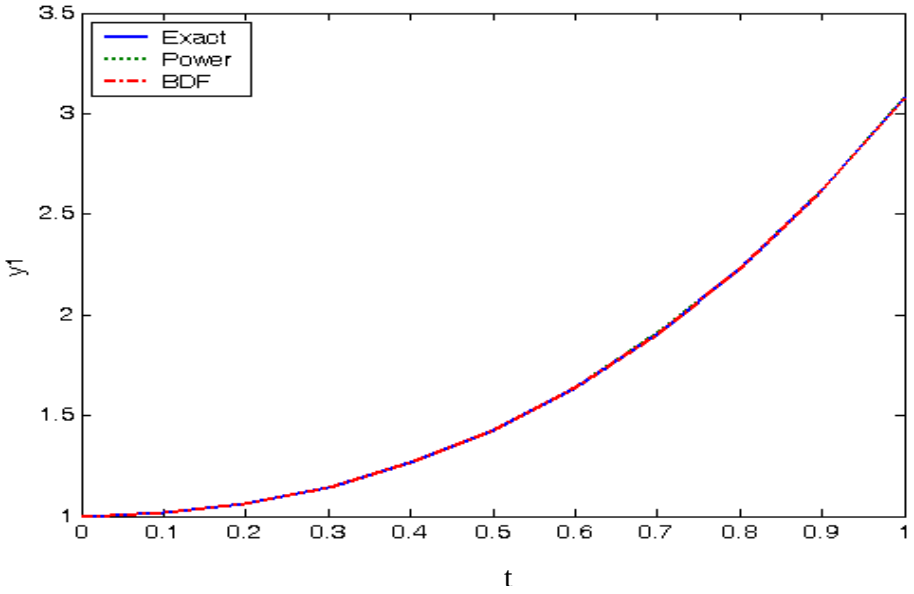


Fig.(4.3.1):Plot for y_1

Table(4.3.2): Numerical solution for y_2

t	Exact	Power series by(4,4)Pade'	Error= |Power-exact|	BDF	Error= |BDF –exact|
0.0	1.0000	1.0000	0	1.0000	0
0.1	1.1152	1.1152	3.0780e-007	1.1152	2.2607e-007
0.2	1.2611	1.2611	5.7972e-006	1.2611	4.9474e-007
0.3	1.4385	1.4385	3.4915e-005	1.4385	8.0813e-007
0.4	1.6476	1.6475	1.3208e-004	1.6476	1.1686e-006
0.5	1.8884	1.8880	3.8698e-004	1.8884	1.5787e-006
0.6	2.1609	2.1599	9.6337e-004	2.1609	3.6638e-005
0.7	2.4647	2.4626	2.1405e-003	2.4647	2.0133e-005
0.8	2.7994	2.7951	4.3727e-003	2.7998	4.0818e-004
0.9	3.1646	3.1562	8.3749e-003	3.1660	1.4254e-003
1.0	3.55982	3.5445	1.5252e-002	3.5631	3.3675e-003

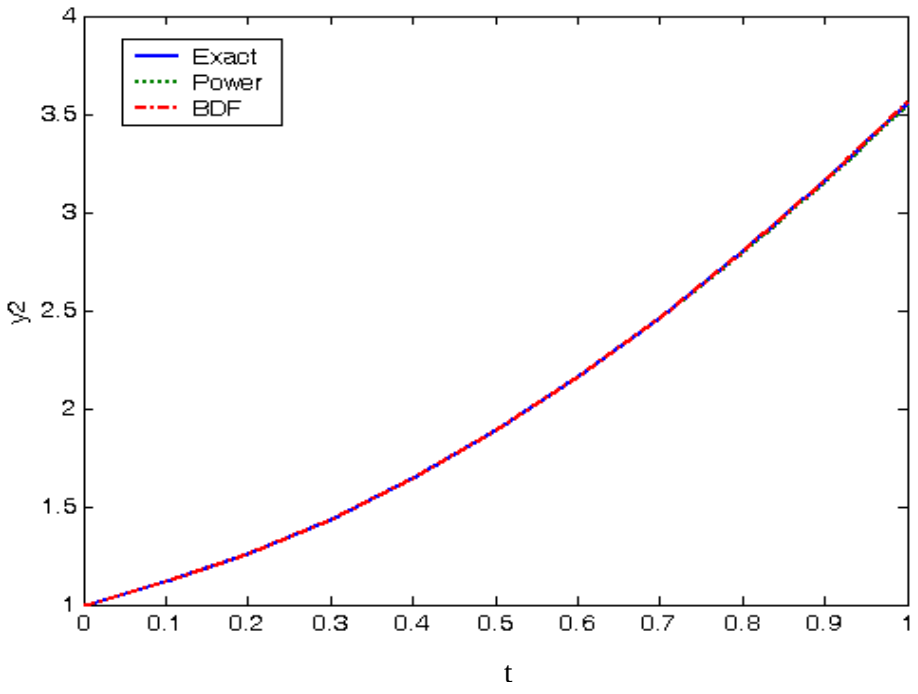


Fig.(4.3.2): Plot for y_2

Table (4.3.3): Numerical solution for y_3:

t	Exact	Power series by(4,4)Pade'	Error= |Power-exact|	BDF	Error= |BDF –exact|
0.0	0	0	0	0	0
0.1	0.0998	0.0998	5.0985e-008	0.0998	3.4675e-009
0.2	0.1987	0.1987	1.5739e-006	0.1987	6.9003e-009
0.3	0.2955	0.2955	1.1499e-005	0.2955	1.0264e-008
0.4	0.3894	0.3895	4.6499e-005	0.3894	1.3525e-008
0.5	0.4794	0.4796	1.3579e-004	0.4794	1.6652e-008
0.6	0.5646	0.5650	3.2243e-004	0.5646	3.8857e-005
0.7	0.6442	0.6449	6.6304e-004	0.6440	2.0335e-004
0.8	0.7174	0.7186	1.2261e-003	0.7177	3.0163e-004
0.9	0.7833	0.7854	2.0891e-003	0.7855	2.2069e-003
1.0	0.8415	0.8448	3.3340e-003	0.8467	5.2333e-003

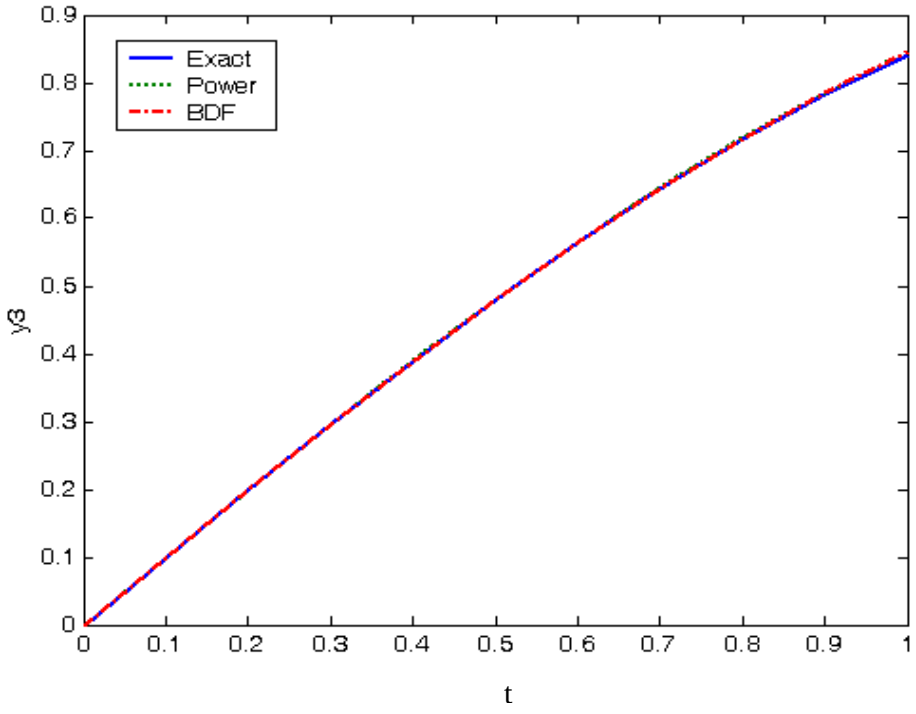


Table (4.3.3): Plot y_3

Table(4.3.4) :Maximum error

Method / Variable	Power Series	BDF
y_1	1.7878e-003	9.6031e-003
y_2	1.5252e-002	3.3675e-003
y_3	3.3340e-003	5.2333e-003

We can say that the two methods have approximately the same accuracy.

Table (4.3.5): Order of convergence with α=1.

Method	λ for y_1									Range of λ
Power	1.0236	1.0401	1.0565	1.0723	1.0869	1.1000	1.1114	1.1210	1.1289	0.1053
BDF	1.0236	1.0401	1.0565	1.0723	1.0868	1.0998	1.1110	1.1202	1.1273	0.1037
	λ for y_2									
Power	1.0690	1.0785	1.0857	1.0908	1.0941	1.0958	1.0960	1.0951	1.0934	0.0270
BDF	1.0690	1.0785	1.0857	1.0910	1.0943	1.0961	1.0967	1.0964	1.0953	0.0277
	λ for y_3									
Power	1.0899	1.0808	1.0726	1.0648	1.0577	1.0511	1.0448	1.0389	1.0333	0.0567
BDF	1.0899 1	1.0808	1.0725	1.0648	1.0576	1.0507	1.0448	1.0395	.0343	0.0556

It can be seen from Table(4.3.5) that the two methods have approximately the same rate of convergence.

***Example (4)*: Consider the following DAE which depends on a parameter η,**

this system could be named *η-system*.

$$\begin{bmatrix} 0 & 0 \\ 1 & \eta t \end{bmatrix} \begin{bmatrix} v_1' \\ v_2' \end{bmatrix} + \begin{bmatrix} 1 & \eta t \\ 0 & 1+\eta \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} q(t) \\ 0 \end{bmatrix},$$

The exact solution is $v_1(t) = q(t) + \eta t q'(t), \quad v_2(t) = -q'(t).$

This system has two properties, firstly , the parameter η plays a main role for existence and stability of the solution, if η< -0.5 then the system will be unstable, if η=-1 then the system doesn't have a regular pencil.[12]

Secondly, the direct differentiation is not enough to produce a system of ODEs. So, η-system requires discretization by implicit Euler method. The difference equations will be derived as follows:

Let us write the system as

$$v_1 + \eta t v_2 = q(t)$$

$$v_1' + \eta t v_2' + (1+\eta) v_2 = 0$$

which can be written as

$$v_2 = \frac{-1}{\eta + 1}(v_1 + \eta t v_2')$$

$$v_1 = q(t) - \eta t v_2$$

and differentiate $v_1' = q'(t) - \eta t v_2' - \eta v_1$

and finally

$$v_2 = \frac{\eta}{1+\eta} v_2 - \frac{1}{1+\eta} q'(t)$$

$$v_1 = q(t) - \eta t v_2.$$

So, the difference equations will be

$$v_{2,n} = \frac{\eta}{1+\eta} v_{2,n-1} - \frac{q(t_n) - q(t_{n-1})}{(1+\eta)h}$$

$$v_{1,n} = q(t_n) - \eta t_n v_{2,n}. \quad [8]$$

Assume that $q(t)=e^t$, and η=1, so we have the system

$$\begin{bmatrix} 0 & 0 \\ 1 & t \end{bmatrix} \begin{bmatrix} v_1' \\ v_2' \end{bmatrix} + \begin{bmatrix} 1 & t \\ 0 & 2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} e^t \\ 0 \end{bmatrix}, \qquad (4.4)$$

with initial values at starting point x=0: $v_0 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$ and $v_0' = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$.

and exact solution: $v_1(t) = e^t + te^t$, $v_2(t) = -e^t$.

Applying BDF method and Power series over the interval [0,1]. The following tables will try to present the results and comparisons.

Table(4.4.1): Numerical solution for y_1

t	Exact	Power series By (3,2)Pade	Error= |Power-exact|	BDF	Error= |BDF –exact|
0.0	1.0000	1.0000	0	1.0000	0
0.1	1.2157	1.2107	5.0156e-003	1.2078	7.9316e-003
0.2	1.4657	1.4441	2.1540e-002	1.4402	2.5463e-002
0.3	1.7548	1.7028	5.1983e-002	1.7067	4.8161e-002
0.4	2.0886	1.9895	9.9016e-002	2.0136	7.4933e-002
0.5	2.4731	2.3075	1.6558e-001	2.3671	1.0600e-001
0.6	2.9154	2.6605	2.5489e-001	2.7745	1.4084e-001
0.7	3.4234	3.0530	3.7040e-001	3.2472	1.7621e-001
0.8	4.0060	3.4902	5.1579e-001	3.8019	2.0412e-001
0.9	4.6732	3.9784	6.9489e-001	4.4634	2.0981e-001
1.0	5.4366	4.5250	9.1156e-001	5.2667	1.6984e-001

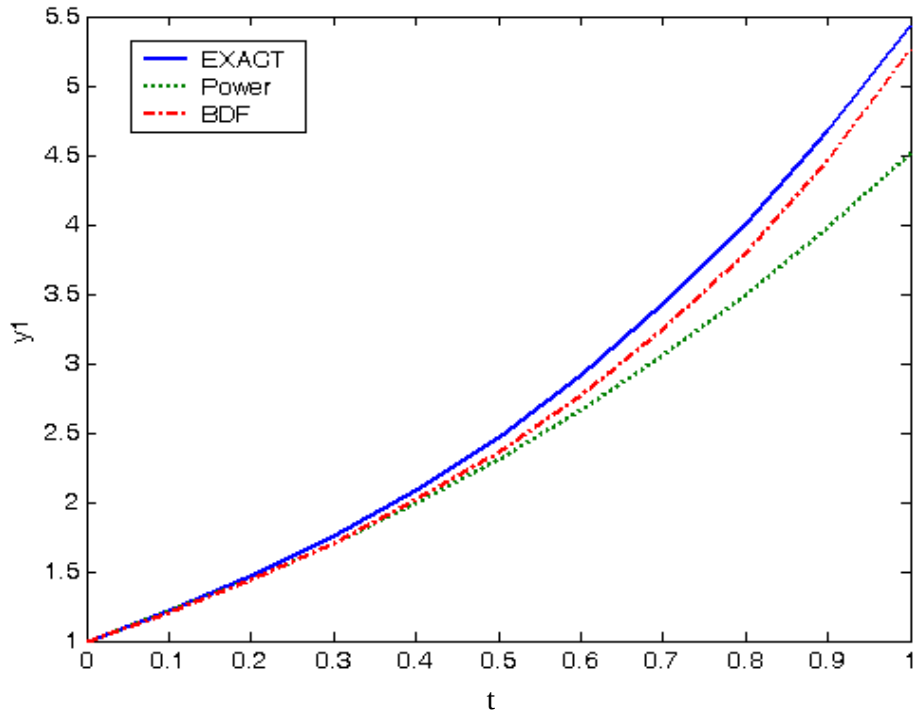


Fig.(4.4.1) :Plot of y_1

Table(4.4.2): Numerical solution for y_2:

t	Exact	Power series By (3,2)Pade	Error= |Power-exact|	BDF	Error= |BDF –exact|
0.0	-1.0000	-1.0000	0	-1.0000	0
0.1	-1.1052	-1.1028	2.3498e-003	-1.0259	7.9316e-002
0.2	-1.2214	-1.2116	9.7664e-003	-1.0941	1.2732e-001
0.3	-1.3499	-1.3270	2.2835e-002	-1.1893	1.6054e-001
0.4	-1.4918	-1.4496	4.2190e-002	-1.3045	1.8733e-001
0.5	-1.6487	-1.5802	6.8513e-002	-1.4367	2.1199e-001
0.6	-1.8221	-1.7196	1.0253e-001	-1.5874	2.3473e-001
0.7	-2.0138	-1.8687	1.4502e-001	-1.7620	2.5172e-001
0.8	-2.2255	-2.0287	1.9680e-001	-1.9704	2.5510e-001
0.9	-2.4596	-2.2009	2.5871e-001	-2.2266	2.3299e-001
1.0	-2.7183	-2.3867	3.3162e-001	-2.5487	1.6954e-001

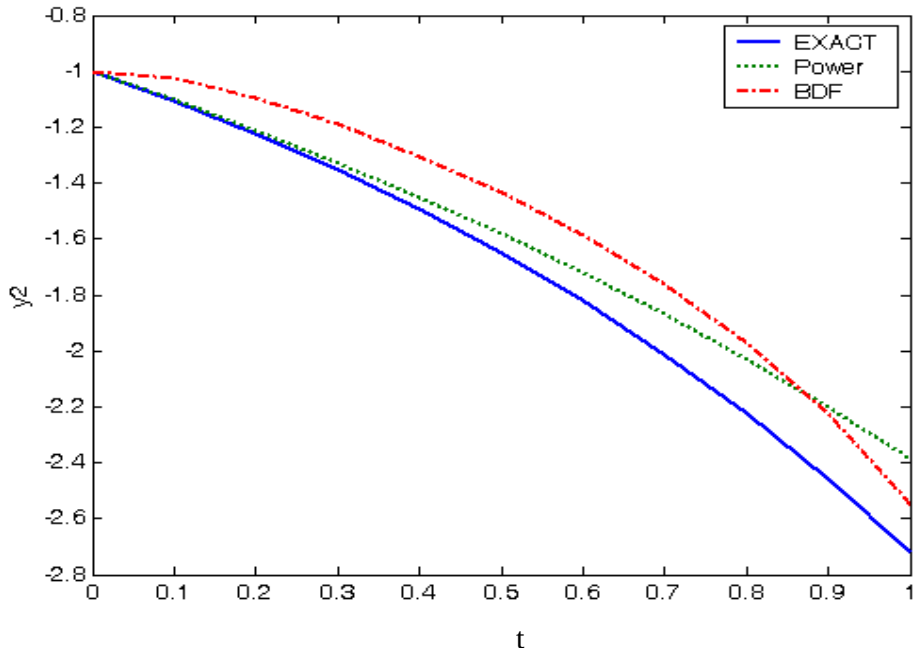


Fig.(4.4.2): Plot for y_2

Table(4.4.3): Maximum error

Method / Variable	Power Series	BDF
y_1	9.1156e-001	2.0981e-001
y_2	3.3162e-001	2.5510e-001

Table (4.4.4): Order of convergence with α=1.

Method	λ for y_1									Range of λ
Power	1.1056	1.1058	1.1061	1.1064	1.1067	1.1072	1.1079	1.1087	1.1098	0.0042
BDF	1.1053	1.1092	1.1134	1.1173	1.1210	1.1252	1.1306	1.1378	1.1470	0.0418
λ for y_2										
Power	2.0584	1.5454	1.3749	1.2905	1.2403	1.2072	1.1842	1.1674	1.1547	0.9036
BDF	3.6332	2.0117	1.6086	1.4342	1.3451	1.2972	1.2735	1.2640	1.2626	2.3706

In example (4), it can be noticed that BDF is more accurate than Power series, but Power series has better rate of convergence than BDF.

Example (5):

Consider The previous system by taking η=-1, we have

$$\begin{bmatrix} 0 & 0 \\ 1 & -t \end{bmatrix}\begin{bmatrix} v_1' \\ v_2' \end{bmatrix}+\begin{bmatrix} 1 & -t \\ 0 & 0 \end{bmatrix}\begin{bmatrix} v_1 \\ v_2 \end{bmatrix}=\begin{bmatrix} e^t \\ 0 \end{bmatrix}, \qquad (4.5)$$

with initial values at starting point t=0: $v_0=\begin{bmatrix} 1 \\ -1 \end{bmatrix}$ and $v_0'=\begin{bmatrix} 0 \\ -1 \end{bmatrix}$.

and exact solution: $v_1(t)=e^t-te^t$, $v_2(t)=-e^t$.

This system has no regular pencil, so BDF will fail to solve this system. So we can apply power series algorithm for solving the system

numerically over $t \in [0,1]$, and comparison with exact solution will be included in the following tables:

Table(4.5.1): Numerical solution for y_1

t	Exact	Power series by (3,3)Pade	Error= |Power-exact
0.0	1.0000	1.0000	0
0.1	0.9947	0.9903	4.3263e-003
0.2	0.9771	0.9625	1.4666e-002
0.3	0.9449	0.9177	2.7159e-002
0.4	0.8951	0.8570	3.8060e-002
0.5	0.8244	0.7806	4.3767e-002
0.6	0.7288	0.6879	4.0969e-002
0.7	0.6041	0.5771	2.7025e-002
0.8	0.4451	0.4442	8.8376e-004
0.9	0.2460	0.2805	3.4588e-002
1.0	0	0.0659	6.5891e-002

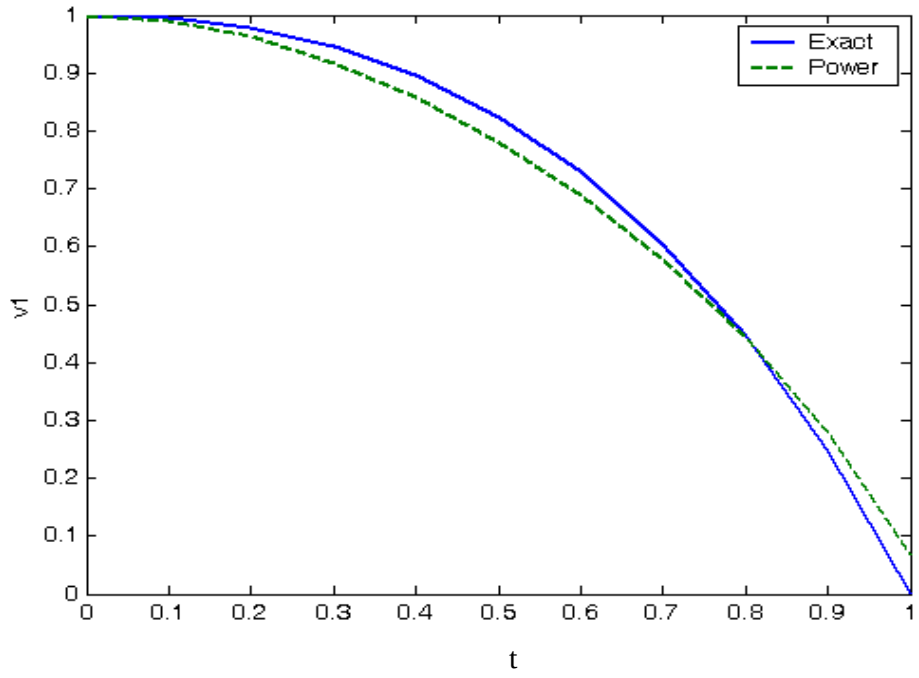


Fig.(4.5.1) :Plot of y_1

Table(4.5.2): Numerical solution for y_2

t	Exact	Power series by (3,2)Pade	Error= |Power-exact|
0.0	**-1.0000**	**-1.0000**	**0**
0.1	**-1.1052**	**-1.1091**	**3.9526e-003**
0.2	**-1.2214**	**-1.2354**	**1.3951e-002**
0.3	**-1.3499**	**-1.3775**	**2.7675e-002**
0.4	**-1.4918**	**-1.5350**	**4.3135e-002**
0.5	**-1.6487**	**-1.7072**	**5.8455e-002**
0.6	**-1.8221**	**-1.8939**	**7.1759e-002**
0.7	**-2.0138**	**-2.0949**	**8.1098e-002**
0.8	**-2.2255**	**-2.3099**	**8.4401e-002**
0.9	**-2.4596**	**-2.5390**	**7.9434e-002**
1.0	**-2.7183**	**-2.7821**	**6.3769e-002**

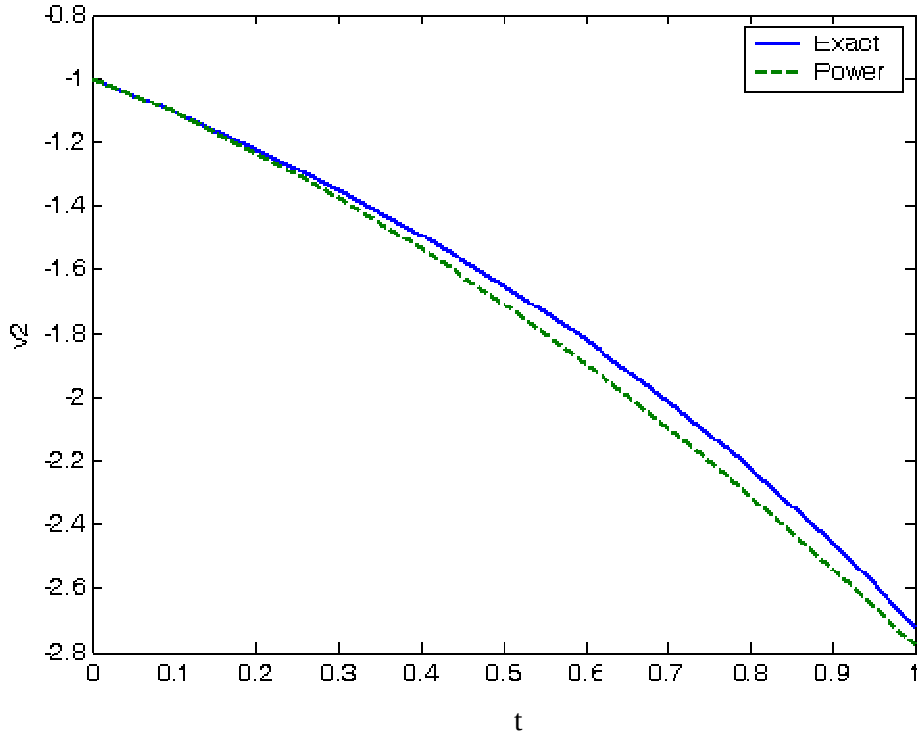


Fig.(4.5.2): Plot of y_2

Table(4.5.3): Maximum error and order of convergence for Power series

Method / Variable	Max. error	Rate of convergence
y_1	6.5891e-002	$\alpha = 1$ $\lambda \in [1.2983, 3.866]$
y_2	8.4401e-002	$\alpha = 1$ $\lambda \in [1.1580, 2.1577]$

4.2 Results and conclusions

To simplify dealing with DAEs, power series, as a numerical method, tries to get rid of the restrictions such as step size adaptation which occurs in the old algorithm such as, BDF. In addition, power series can avoid the main difficulties which attach DAEs such as converting into ODEs.

In this section we will discuss the advantages of power series depending on the numerical examples in the previous section, and try to make a comparison with BDF method.

1) Simplicity: It is shown in two aspects:

a) There is no need to reformulate the system into ODEs. This algorithm implements the system directly. While in BDF, the reformulation is considered as an introductory step, but no more than 1-index.

b) The algorithms are coded by MATLAB, which is designed in order to facilitate usages of matrices and vectors. Power series algorithm is obvious, direct, concise, and has no complicated loops and it is easy to be followed. BDF may produce a good results, but it exerts more effort and

storage space. In addition, no one can build on previous execution when step size is changed.

2) **Consistency: It is automatically achieved by step 3 of Power series algorithm. However in BDF, consistency must be checked at each step.**

3) **Accuracy: It is seen that the two method have, in general, almost the same accuracy.**

4) **Convergence: From the tables in the previous section, we see that Power series method and BDF method have the same order of convergence $\alpha=1$, i.e. linearly convergent.**

5) **Power series solved η-system in the last example (5), but BDF failed completely to solve it when $\eta = -1$.**

For the above mentioned aspects, we can deduce that Power series will be the most widely used production code for solving DAEs.

It can also be used to solve ODE systems.

Finally, this study opens new horizons in exploring numerical methods for solving other types of DAEs.

References

Clark, Kenneth D. , Petzold, Linda R. September 1989.Numerical solution of Boundary Value Problems in differential algebraic systems. SIAM. J. STAT. COMPUT. Vol. 10, No. 5, pp. 915-936.

Claus Bendtsen, Per Grove Thomsen,1999. Numerical Solution of Differential Algebraic Equations

Ercan Celik, Mustafa Bayram,2003. Arbitrary order numerical method for solving differential-algebraic equation by Pade series,AMC

Ernst Hairer, Christian Lubich, Michel Roche,1989. The Numerical Solution of Differential-Algebraic Systems by Runge-Kutta Methods

Gantmacher, F. R., 1989. The theory of matrices, AMC

K.E. Brenan, S.L. Campbell, L.R. Petzold,1989. Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations, Elsevier

Publishing Co.,Inc.

L. T. Biegler, 2000. Differential-Algebraic Equations (DAEs), ***dynopt.cheme.cmu.edu***

Matar, Sameer, 1998. Lectures on Differential Algebraic Equations, Graduate studies at Brunel University.

Mazzia Francisca, Magherini Cecilia, 2008. Test Set for Initial Value Problem Solvers

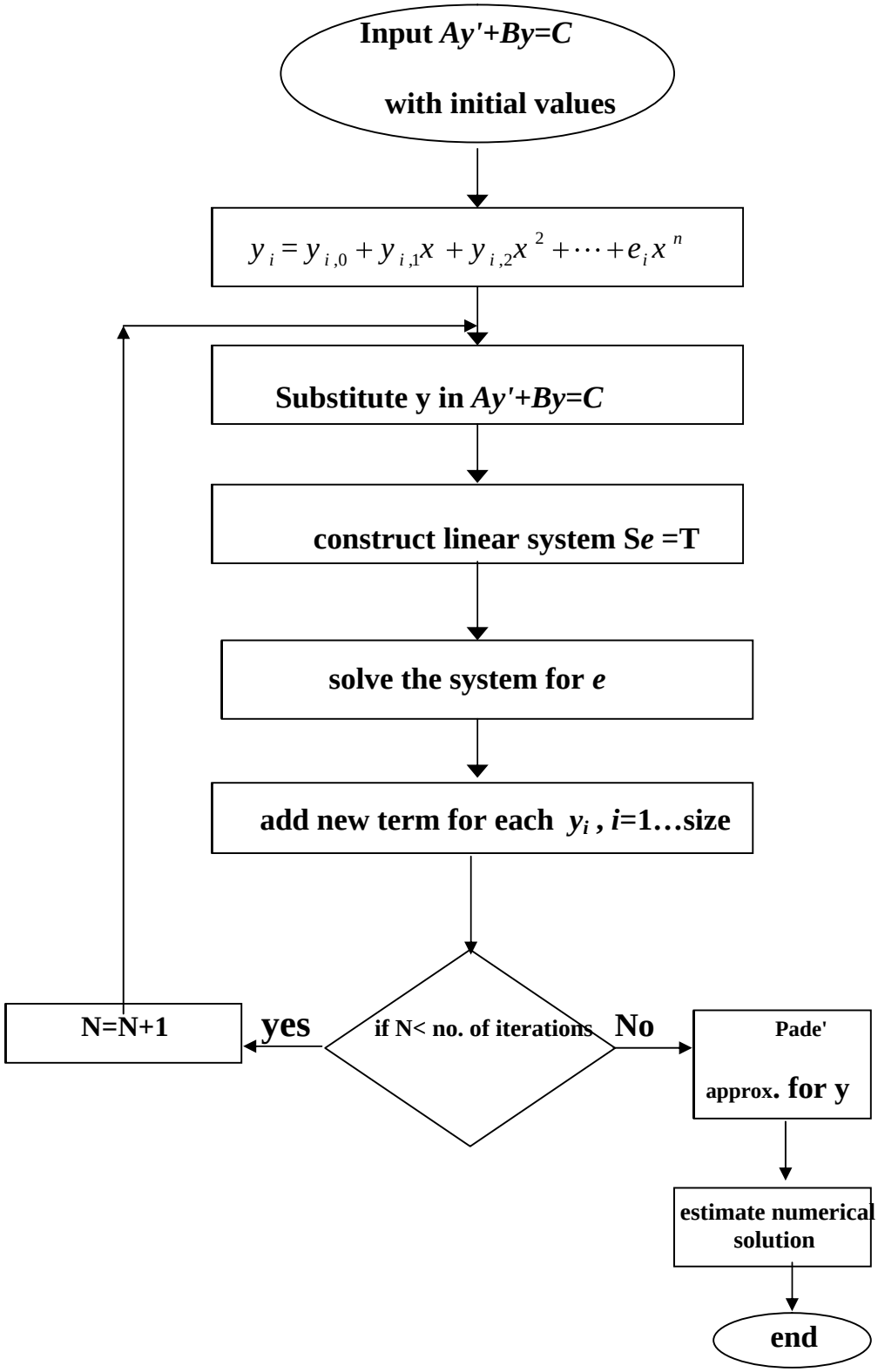
Nedialko S. Nedialkov , John D. Pryce,2005. SOLVING DIFFERENTIAL-ALGEBRAIC EQUATIONS BY TAYLOR SERIES (I): COMPUTING TAYLOR COEFFICIENTS

Richard L. Burden, J. Douglas Faires,2001. Numerical Analysis, Brooks/Cole.

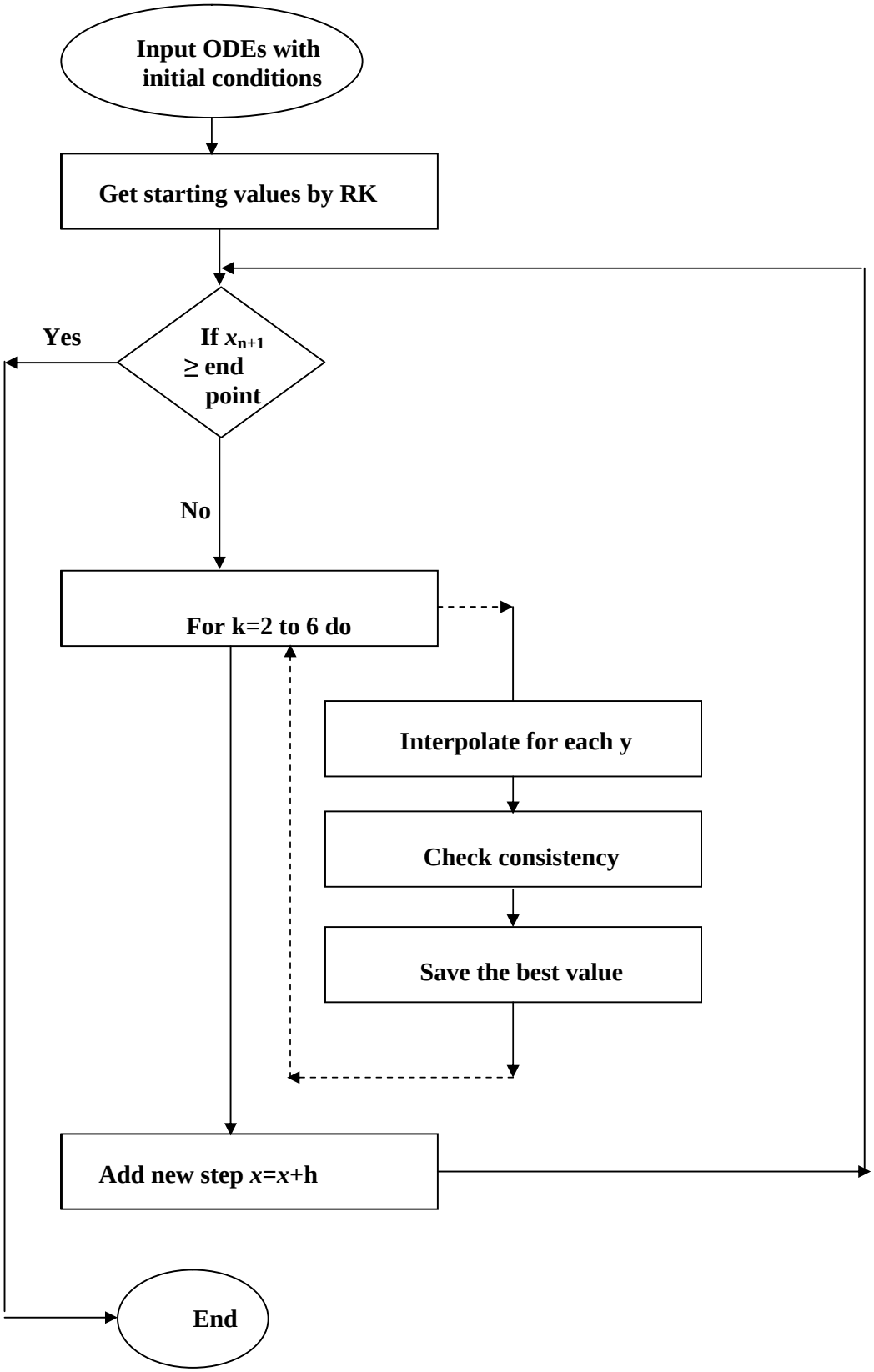
Uri M. Ascher , Linda R. Petzold, 1997. Computer Methods for Ordinary Differential Equations and Differential Algebraic Equations

Appendix A

Flow chart for Power series algorithm :



Flow chart for BDF algorithm



Appendix B

Matlab code for Power series method (powerpade.m)

```
%============================================================
=
%|
%|  THIS PROGRAM PREPARED BY : SAMER AMIN ABU SA'
%|  PURPOSE : TO SOLVE SYSTEM OF DAEs IN THE FORM Ax'+Bx=C
%|  WHERE A AND B ARE TWO SQUARE MATRICES AND C IS VECTOR
FUNCTION
%|  METHOD USED : USING POWER SERIES AND PADE' APPROXIMANT
%|  INPUT DATA : A,B,C,INITIAL VALUES X0,X'0 AND DEGREE OF PADE
APPROXIMANT
%|  OUTPUT DATA : POWER SERIES, AND PADE' APPROXIMANT
%|  COMPARE THE RESULT WITH THE EXACT SOLUTION
%|
%|============================================================

clear
clc
syms x e1 e2 e3 v1 v2 v3 OK

TRUE=1;
FALSE=0;

% Ax'+Bx=C

size=3;
tabcoef=zeros(size,10)
A=[1 -x x^2;0 1 -x;0 0 0];
B=[1 -(x+1) x^2+2*x;0 -1 x-1 ; 0 0 1];
C=[0;0;sin(x)];

%Initial values

v1=1;
v2=1;
v3=0;

v1=1+e1*x;
v2=1+e2*x;
v3=e3*x;

esv1=1;
esv2=1;
esv3=0;

tabcoef(1,1)=1;
tabcoef(2,1)=1;
```

```
tabcoef(3,1)=0;

power=1
% Main loop
for iteration=1:8

%Substitute

f=A*[diff(v1,x);diff(v2,x);diff(v3,x)]+B*[v1;v2;v3]-C
for i=1:size
  simplify(f(i))
end

%ADD NEW TERM
%  HERE ARE ANALYTIC MANIPULATIONS
display('here are analytic manipulations for getting linear system')
display(' ')
power
OK = FALSE;
 while OK == FALSE
 fprintf(1,'Add new vector e : \n');
 fprintf(1,'Choice of addition method:\n');
 fprintf(1,'1. From keyboard\n');
 fprintf(1,'2. Input system and solve it :\n');
 fprintf(1,'Choose 1 or 2 please\n');
 FLAG = input(' ');
 if FLAG == 1 | FLAG == 2
 OK = TRUE;
 end;
 end;
 if FLAG == 1
%FROM KEYBOARD
for i=1:size
newcoef(i)=input('coeffecient :')
end
 end %if flag=1
 if FLAG ==2

    dim=input('Input size of linear system :')
    AA=zeros(dim,dim);
    BB=zeros(dim,1);
    display('Input  coeffecients matrix :')

    for i=1:dim
       for j=1:size
          AA(i,j)=input(' ');
       end
    end
    display('Input constant vector :')
```

```
    for i=1:dim
       BB(i)=input(' ')
    end

    % Solve the system
    scoef=inv(AA)*BB
    if dim==size
       newcoef=scoef
    end
    if dim< size
       display('Input new coeffecient from keyboard ....  size of linear system is
small')
       for i=1:size
          newcoef(i)=input(' ');
       end
    end
 end% flag =2


 %update v
 esv1=esv1+newcoef(1)*x^power;
 esv2=esv2+newcoef(2)*x^power;
 esv3=esv3+newcoef(3)*x^power;

power=power+1;

v1=esv1+e1*x^power;
v2=esv2+e2*x^power;
v3=esv3+e3*x^power;

tabcoef(1,power)=newcoef(1);
tabcoef(2,power)=newcoef(2);
tabcoef(3,power)=newcoef(3);

esv1
esv2
esv3

end %end  for  loop


tabcoef
for k=1:size

run pade
if k==1
p1=0;
for i=1:LN+1
p1=p1+ P(i)*x^(i-1);
```

```
end
q1=0;
for i=1:LM
   q1=q1+Q(i)*x^(i-1);
end
 format short
p1
q1
end
 if k==2
  p2=0;
for i=1:LN+1
p2=p2+ P(i)*x^(i-1);
end
q2=0;
for i=1:LM
   q2=q2+Q(i)*x^(i-1);
end
real format
p2
q2
 end
if k==3
   p3=0;
for i=1:LN+1
p3=p3+ P(i)*x^(i-1);
end
q3=0;
for i=1:LM
   q3=q3+Q(i)*x^(i-1);
end
real format
p3
q3
end
end %for k
display('  x   exact1   v1(x)   pade1   exact2   v2(x)   pade2   exact3   v3(x)
pade3  ')

tabl=zeros(11,7);
exav1=exp(-x)+x*exp(x);
exav2=exp(x)+x*sin(x);
exav3=sin(x);
for i=1:11
  t=(i-1)/10;
tabl(i,:)=[t subs(exav1,t)  subs(p1,t)/subs(q1,t) subs(exav2,t) subs(p2,t)/subs(q2,t)
subs(exav3,t) subs(p3,t)/subs(q3,t)];
end
tabl
```

```
%plot(tabl(:,1),tabl(:,2),tabl(:,3),tabl(:,4))
```

powerpade2eta.m

```
%|============================================================
%|
%|  THIS PROGRAM PREPARED BY : SAMER AMIN ABU SA'
%|  PURPOSE : TO SOLVE SYSTEM OF DAEs IN THE FORM Ax'+Bx=C
%|  WHERE A AND B ARE TWO SQUARE MATRICES AND C IS VECTOR
FUNCTION
%|  METHOD USED : USING POWER SERIES AND PADE' APPROXIMANT
%|  INPUT DATA : A,B,C,INITIAL VALUES X0,X'0 AND DEGREE OF PADE
APPROXIMANT
%|  OUTPUT DATA : POWER SERIES, AND PADE' APPROXIMANT
%|  COMPARE THE RESULT WITH THE EXACT SOLUTION
%|
%|============================================================

clear
clc
syms x e1 e2  v1 v2  OK eta

TRUE=1;
FALSE=0;

% Ax'+Bx=C

size=2;
tabcoef=zeros(size,10)
eta=1;
A=[0 0;1 eta*x];
B=[1 eta*x;0 1+eta];
C=[exp(x); 0];

%Initial values

v1=1;
v2=-1;


v1=1+2*x+e1*x^2
v2=-1-x+e2*x^2


esv1=1+2*x;
esv2=-1-x;


tabcoef(1,1)=1;
tabcoef(2,1)=-1;
```

```
tabcoef(1,2)=1+eta;
tabcoef(2,2)=-1;
power=2

% Main loop

for iteration=1:4

%Substitute

f=A*[diff(v1,x);diff(v2,x)]+B*[v1;v2]-C
for i=1:size
  simplify(f(i))
end

%ADD NEW TERM
%  HERE ARE ANALYTIC MANIPULATIONS

display('here are analytic manipulations for determinig vector e by  ')
display('consisting linear system and solve it or input e directly from key board')
display(' ')
power
OK = FALSE;
 while OK == FALSE
 fprintf(1,'Add new vector e : \n');
 fprintf(1,'Choice of addition method:\n');
 fprintf(1,'1. From keyboard\n');
 fprintf(1,'2. Input system and solve it :\n');
 fprintf(1,'Choose 1 or 2 please\n');
 FLAG = input(' ');
 if FLAG == 1 | FLAG == 2
 OK = TRUE;
 end;
 end;
 if FLAG == 1
%FROM KEYBOARD
for i=1:size
newcoef(i)=input('coeffecient :')
end
 end %if flag=1
 if FLAG ==2

    dim=input('Input size of linear system :')
    AA=zeros(dim,dim);
    BB=zeros(dim,1);
    display('Input  coeffecients matrix :')

    for i=1:dim
       for j=1:size
```

```
        AA(i,j)=input(' ');
      end
    end
    display('Input constant vector :')
    for i=1:dim
      BB(i)=input(' ')
    end

    % Solve the system
    scoef=inv(AA)*BB
    if dim==size
      newcoef=scoef
    end
    if dim< size
      display('Input new coeffecient from keyboard ....  size of linear system is
small')
      for i=1:size
        newcoef(i)=input(' ');
      end
    end
 end% flag =2


 %update v
 esv1=esv1+newcoef(1)*x^power;
 esv2=esv2+newcoef(2)*x^power;

power=power+1;

v1=esv1+e1*x^power;
v2=esv2+e2*x^power;


tabcoef(1,power)=newcoef(1);
tabcoef(2,power)=newcoef(2);

esv1
esv2

end %end main loop

esv1;
esv2;

tabcoef
for trial=1:16

for k=1:size
```

```
run pade
if k==1
p1=0;
for i=1:LN+1
p1=p1+ P(i)*x^(i-1);
end
q1=0;
for i=1:LM
   q1=q1+Q(i)*x^(i-1);
end
real format
p1
q1
end
 if k==2
  p2=0;
for i=1:LN+1
p2=p2+ P(i)*x^(i-1);
end
q2=0;
for i=1:LM
   q2=q2+Q(i)*x^(i-1);
end
real format
p2
q2
 end

end %for k
display('    x    exact1    pade1    exact2    pade2    ')
tabl=zeros(11,5);
exav1=exp(x)+eta*x*exp(x);
exav2=-exp(x);
for i=1:11
  t=(i-1)/10;
tabl(i,:)=[t subs(exav1,t)  subs(p1,t)/subs(q1,t) subs(exav2,t)  subs(p2,t)/subs(q2,t)];
end
tabl
end%trial
%plot(tabl(:,1),tabl(:,2),tabl(:,3),tabl(:,4))
```

pade.m

```
% PADE RATIONAL APPROXIMATION ALGORITHM
%
% To obtain the rational approximation
%
%     r(x) = p(x) / q(x)
%          = (p0 + p1*x + ... + Pn*x^n) / (q0 + q1*x + ... + qm*x^m)
%
% for a given function f(x):
%
% INPUT  nonnegative integers m and n.
%
% OUTPUT  coefficients q0, q1, ... , qm, p0, p1, ... , pn.
%
% The coefficients of the Maclaurin polynomial a0, a1,  ... could
% be calculated instead of input as is assumed in this program.
 syms('OK', 'LM', 'LN', 'BN', 'FLAG', 'I', 'AA', 'AAA');
 syms('NAME', 'INP', 'N', 'M', 'NROW', 'NN', 'Q', 'P', 'J');
 syms('A', 'IMAX', 'AMAX', 'JJ', 'IP', 'JP', 'NCOPY', 'I1');
 syms('J1', 'XM', 'K', 'N1', 'PP', 'N2', 'SUM', 'KK', 'LL', 'OUP');
 TRUE = 1;
 FALSE = 0;
 fprintf(1,'This is Pade Approximation.\n\n');
 OK = FALSE;
 while OK == FALSE
 fprintf(1,'m+n must equal %d \n',power-1);
 fprintf(1,'Input m and n on separate lines.\n');
 LN = input('order of numerator ');
 LM = input('order of denominator  ');
 BN = LM+LN;
 if LM >= 0 & LN >= 0
 OK = TRUE;
 else
 fprintf(1,'m and n must both be nonnegative.\n');
 end;
 if LM == 0 & LN == 0
 OK = FALSE;
 fprintf(1,'Not both m and n can be zero\n');
 end;
 %if BN~= power-1
  %   OK=FALSE;
   %  fprintf(1,'m+n must equal %d \n',power-1);
   %end
 end;
```

```
AA = zeros(1,BN+1);
%if FLAG == 1
%fprintf(1,'Input in order a(0) to a(N)\n');
for I = 0 : BN
%fprintf(1,'Input A( %d ) \n',I);
AA(I+1) = tabcoef(k,I+1) %input(' ');
%end;
end;

if OK == TRUE
% STEP 1
N = BN;
M = N+1;
% STEP 2 - performed in input
NROW = zeros(1,N);
for I = 1 : N
NROW(I) = I;
end;
% initialize row pointer for linear system
NN = N-1;
% STEP 3
Q = zeros(1, LM + 1);
P = zeros(1, LN + 1);
A = zeros(N,N+1);
Q(1) = 1;
P(1) = AA(1);
% STEP 4
% Set up a linear system, but use A(i,j) instead of B(i,j).
for I = 1 : N
% STEP 5
for J = 1 : I-1
if J <= LN
A(I,J) = 0;
end;
end;
% STEP 6
if I <= LN
A(I,I) = 1;
end;
% STEP 7
for J = I+1 : LN
A(I,J) = 0;
end;
% STEP 8
for J = 1 : I
if  J <= LM
A(I,LN+J) = -AA(I-J+1);
end;
```

```
 end;
% STEP 9
 for J = LN+I+1 : N
 A(I,J) = 0;
 end;
% STEP 10
 A(I,N+1) = AA(I+1);
 end;
% Solve the linear system using partial pivoting.
 I = LN+1;
% STEP 11
 while OK == TRUE & I <= NN
% STEP 12
 IMAX = NROW(I);
 AMAX = abs(A(IMAX,I));
 IMAX = I;
 JJ = I+1;
 for IP = JJ : N
 JP = NROW(IP);
 if abs(A(JP,I)) > AMAX
 AMAX = abs(A(JP,I));
 IMAX = IP;
 end;
 end;
% STEP 13
 if AMAX <= 1.0e-20
 OK = false;
 else
% STEP 14
% simulate row interchange
 if NROW(I) ~= NROW(IMAX)
 NCOPY = NROW(I);
 NROW(I) = NROW(IMAX);
 NROW(IMAX) = NCOPY;
 end;
 I1 = NROW(I);
% STEP 15
% Perform elimination.
 for J = JJ : N
 J1 = NROW(J);
% STEP 16
 XM = A(J1,I)/A(I1,I);
% STEP 17
 for K = JJ : M
 A(J1,K) = A(J1,K)-XM * A(I1,K);
 end;
% STEP 18
 A(J1,I) = 0;
 end;
```

```
end;
I = I+1;
end;
if OK == TRUE
% STEP 19
N1 = NROW(N);
if abs(A(N1,N)) <= 1.0e-20
OK = FALSE;
% system has no unique solution
else
% STEP 20
% Start backward substitution.
if LM > 0
Q(LM+1) = A(N1,M)/A(N1,N);
A(N1,M) = Q(LM+1);
end;
PP = 1;
% STEP 21
for K = LN+1 : NN
I = NN-K+LN+1;
JJ = I+1;
N2 = NROW(I);
SUM = A(N2,N+1);
for KK = JJ : N
LL = NROW(KK);
SUM = SUM-A(N2,KK)*A(LL,M);
end;
A(N2,M) = SUM/A(N2,I);
Q(LM-PP+1) = A(N2,M);
PP = PP+1;
end;
% STEP 22
for K = 1 : LN
I = LN-K+1;
N2 = NROW(I);
SUM = A(N2,N+1);
for KK = LN+1 : N
LL = NROW(KK);
SUM = SUM-A(N2,KK)*A(LL,M);
end;
A(N2,M) = SUM;
P(LN-K+2) = A(N2,M);
end;

OUP = 1;
%end;
fprintf(OUP, 'PADE RATIONAL APPROXIMATION\n\n');
fprintf(OUP, 'Denominator Coefficients Q(0), ..., Q(M) \n');
for I = 0 : LM
```

```
fprintf(OUP, ' %11.8f', Q(I+1));
end;
fprintf(OUP, '\n');
fprintf(OUP, 'Numerator Coefficients P(0), ..., P(N)\n');
for I = 0 : LN
fprintf(OUP, ' %11.8f', P(I+1));
end;
fprintf(OUP, '\n');
if OUP ~= 1
fclose(OUP);
fprintf(1,'Output file %s created successfully \n',NAME);
end;
end;
end;
if OK == FALSE
fprintf(1,'System has no unique solution\n');
end;
end;
```

Matlab code for BDF method (BDF.m)

```
%|============================================================
%|
%|  THIS PROGRAM PREPARED BY : SAMER AMIN ABU SA'
%|  PURPOSE : TO SOLVE SYSTEM OF DAEs IN THE FORM Ax'+Bx=C
%|  WHERE A AND B ARE TWO SQUARE MATRICES AND C IS VECTOR
FUNCTION
%|  METHOD USED : USING BACKWARD DIFFERENCE FORMULA WITH
FIXED LEADING COEFFECIENT
%|  INPUT DATA : A,B,C,INITIAL VALUES X0,X'0 AND DEGREE OF PADE
APPROXIMANT
%|  OUTPUT DATA : TABLE CONTAINS NUMERICAL RESULTS,EXACT
SOLUTION
%|  COMPARE THE RESULT WITH THE EXACT SOLUTION
%|
%|============================================================
clear
clc
format short
syms('s1','s2','s3','t','y1','y2','y3','h','f1','f2','f3','a','exact1','exact2','exact3','x','k')

%run BDFeta;
%ys1=tabl(:,3)
%ys2=tabl(:,5)



% EXACT SOLUTION

exact1=exp(-x)+x*exp(x);
exact2=exp(x)+x*sin(x);
exact3=sin(x);


%Initial values
h=0.1;
a=0;
b=1;
alpha1=1;
alpha2=-1;
alpha3=0;
Iteration=0;

ys1(1)=alpha1;
ys2(1)=alpha2;
```

```
ys3(1)=alpha3;

ts(1)=0;
siz=(b-a)/h;
tss=a:h:b

%STARTING VALUES
display('FIRST DIVISION : INPUT THE SYSTEM, INITIAL CONDITIONS
AND STEP SIZE h')

run RK4SYS;

display('SECOND DIVISION: SAVE STARTING VALUES USING RUNGE-
KUTTA FOR SYSTEM ')
ys1=tabl(:,1);
ys2=tabl(:,2);
ys3=tabl(:,3)
%%%%%%%%%%%%%%%%%%%%%
k=6;

ks=k;
yss1=ys1;
yss2=ys2;
yss3=ys3;

err1=0.3;
err2=0.3;
err3=0.3;
%MAIN LOOP
display('THIRD DIVISION : MAIN LOOP ... APPLYING BDF ')
while tss(ks)< b
   Iteration=Iteration+1
   if Iteration ==7
      input(' ')
   end
   curt=tss(ks)+h
   p1=0;
   p2=0;
   p3=0;
   kk=2;
   errp=20;
   display('EVALUATE NEW y1(n+1)')
   while (kk<7)&(err1>0.0001)

      ys1=zeros(1,kk+1);
      ts=zeros(1,kk);
      for i=ks-kk+1:ks
         ys1(i-ks+kk)=yss1(i);
         ts(i-ks+kk)=tss(i);
```

```
   end
   p1=0;
   k=kk
   k1=kk;
   run divdif1;
   p1;
   tc=tss(ks)+h;
   cury1=subs(p1,tss(ks)+h)
   err1=abs(cury1-subs(exact1,tss(ks)+h));
   if err1 < errp
      yss1(ks+1)=cury1;
      errp=err1;
   end
   kk=kk+1;
end

kk=2;
errp=20;
display('EVALUATE NEW y2(n+1)')
while (kk<7)&(err2>0.0001)
   ys2=zeros(1,kk);
   ts=zeros(1,kk);
   for i=ks-kk+1:ks
      ys2(i-ks+kk)=yss2(i);
      ts(i-ks+kk)=tss(i);
   end
   p2=0;
   k=kk
   k2=kk;
   run divdif2;
   p2;
   tc=tss(ks)+h;
   cury2=subs(p2,tss(ks)+h)
   err2=abs(cury2-subs(exact2,tss(ks)+h));
   if err2 < errp
      yss2(ks+1)=cury2;
      errp=err2;
   end
   kk=kk+1;
end

kk=2;
errp=20;
display('EVALUATE NEW y3(n+1)')
while (kk<7)&(err3>0.0001)
   ys3=zeros(1,kk);
   ts=zeros(1,kk);
   for i=ks-kk+1:ks
      ys3(i-ks+kk)=yss3(i);
```

```
        ts(i-ks+kk)=tss(i);
      end
      p3=0;
      k=kk
      k3=kk;
      run divdif3;
      p3;
      tc=tss(ks)+h;
      cury3=subs(p3,tss(ks)+h)
      err3=abs(cury3-subs(exact3,tss(ks)+h));
      if err3 < errp
        yss3(ks+1)=cury3;
        errp=err3;
      end
      kk=kk+1;
   end

   ks=ks+1;
   err1=0.3;
   err2=0.3;
   err3=0.3;
end% END MAIN LOOP

display('FOURTH DIVISION : NUMERICAL RESULTS WITH EXACT
SOLUTION ')

table=zeros(11,7);
for i=1:11
  t=(i-1)/10;
table(i,:)=[t subs(exact1,t)  yss1(i) subs(exact2,t) yss2(i) subs(exact3,t) yss3(i)];
end
```

RK4SYS.m

```
% RUNGE-KUTTA FOR SYSTEMS OF DIFFERENTIAL EQUATIONS
ALGORITHM 5.7
%
% TO APPROXIMATE THE SOLUTION OF THE MTH-ORDER SYSTEM OF
FIRST-
% ORDER INITIAL-VALUE PROBLEMS
%          UJ' = FJ( T, U1, U2, ..., UM ), J = 1, 2, ..., M
%          A <= T <= B, UJ(A) = ALPHAJ, J = 1, 2, ..., M
% AT (N+1) EQUALLY SPACED NUMBERS IN THE INTERVAL (A,B).
%
% INPUT:   ENDPOINTS A,B; NUMBER OF EQUATIONS M; INITIAL
%        CONDITIONS ALPHA1, ..., ALPHAM; INTEGER N.
%
% OUTPUT:  APPROXIMATION WJ TO UJ(T) AT THE (N+1) VALUES OF T.
 syms('OK', 'M', 'I', 'A', 'B', 'ALPHA', 'N', 'FLAG');
 syms('NAME', 'OUP', 'H', 'T', 'J', 'W', 'L', 'K','ss');
 syms('K1','K2','K3','K4','Z','kk');
 TRUE = 1;
 FALSE = 0;
 fprintf(1,'This is the Runge-Kutta Method for Systems of m equations\n');
 fprintf(1,'This program uses the file F.m.  If the number of equations\n');
 fprintf(1,'exceeds 7, then F.m must be changed.\n');
 OK = FALSE;
 while OK == FALSE
 fprintf(1,'Input the number of equations\n');
 M = input(' ');
 if M <= 0 | M > 7
 fprintf(1,'Number must be a positive integer < 8\n');
 else
 OK = TRUE;
 end;
 end;
 ss = cell(M,1);
 for I = 1:M
 fprintf(1,'Input the function F_(%d) in terms of t and y1 ... y%d\n', I,M);
 fprintf(1,'For example: y1-t^2+1 \n');
 kk = input(' ');
 ss{I} = kk;
 end;
 OK = FALSE;
 while OK == FALSE
 fprintf(1,'Input left and right endpoints on separate lines.\n');
 A = input(' ');
 B = input(' ');
```

```
if A >= B
fprintf(1,'Left endpoint must be less than right endpoint\n');
else
OK = TRUE;
end;
end;
tabl=zeros(6,3);
ALPHA = zeros(1,M);
for I = 1:M
fprintf(1,'Input the initial condition alpha(%d)\n', I);
ALPHA(I) = input(' ');
tabl(1,I)=ALPHA(I);
end;

OK = FALSE;
while OK == FALSE
fprintf(1,'Input a positive integer for the number of subintervals\n');
N = input(' ');
if N <= 0
fprintf(1,'Number must be a positive integer\n');
else
OK = TRUE;
end;
end;
H=input('INPUT step size h : ')
h=H;
if OK == TRUE
fprintf(1,'Choice of output method:\n');
fprintf(1,'1. Output to screen\n');
fprintf(1,'2. Output to text file\n');
fprintf(1,'Please enter 1 or 2\n');
FLAG = input(' ');
if FLAG == 2
fprintf(1,'Input the file name in the form - drive:\\name.ext\n');
fprintf(1,'For example   A:\\OUTPUT.DTA\n');
NAME = input(' ','s');
OUP = fopen(NAME,'wt');
else
OUP = 1;
end;
fprintf(OUP,'RUNGE-KUTTA METHOD FOR SYSTEMS OF DIFFERENTIAL
EQUATIONS\n\n');
fprintf(OUP, '    T');
for I = 1:M
fprintf(OUP, '         W%d', I);
end;
% STEP 1

W = zeros(1,M);
```

```
V = zeros(1,M+1);
K1 = zeros(1,M);
K2 = zeros(1,M);
K3 = zeros(1,M);
K4 = zeros(1,M);
%H = (B-A)/N;

T = A;
% STEP 2
for J = 1:M
W(J) = ALPHA(J);
end;
% STEP 3
fprintf(OUP, '\n%5.3f', T);
for I = 1:M
fprintf(OUP, ' %11.8f', W(I));
end;
fprintf(OUP, '\n');
% STEP 4
for L = 1:N
% STEP 5
V(1) = T;
for J = 2:M+1
V(J) = W(J-1);
end;
for J = 1:M
Z = H*F(J,M,V,ss);
K1(J) = Z;
end;
% STEP 6
V(1) = T+H/2;
for J = 2:M+1
V(J) = W(J-1)+K1(J-1)/2;
end;
for J = 1:M
Z = H*F(J,M,V,ss);
K2(J) = Z;
end;
% STEP 7
for J = 2:M+1
V(J) = W(J-1)+K2(J-1)/2;
end;
for J = 1:M
Z = H*F(J,M,V,ss);
K3(J) = Z;
end;
% STEP 8
V(1) = T + H;
for J = 2:M+1
```

```
V(J) = W(J-1)+K3(J-1);
end;
for J = 1:M
Z = H*F(J,M,V,ss);
K4(J) = Z;
end;
% STEP 9
for J = 1:M
W(J) = W(J)+(K1(J)+2.0*K2(J)+2.0*K3(J)+K4(J))/6.0;
tabl(L+1,J)=W(J);
end;
% STEP 10
T = A+L*H;
% STEP 11
fprintf(OUP, '%5.3f', T);
for I = 1:M
fprintf(OUP, ' %11.8f', W(I));
end;
fprintf(OUP, '\n');
end;
% STEP 12
if OUP ~= 1
fclose(OUP);
fprintf(1,'Output file %s created successfully \n',NAME);
end;
end;
```

divdif.m

```
% NEWTONS INTERPOLATORY DIVIDED-DIFFERENCE FORMULA
ALGORITHM
 % To obtain the divided-difference coefficients of the
 % interpolatory polynomial P on the (n+1) distinct numbers x(0),
 % x(1), ..., x(n) for the function f:
 % INPUT:   numbers x(0), x(1), ..., x(n); values f(x(0)), f(x(1)),
 %          ..., f(x(n)) as the first column Q(0,0), Q(1,0), ...,
 %    Q(N,0) of Q, or may be computed if function f is supplied.
 % OUTPUT:  the numbers Q(0,0), Q(1,1), ..., Q(N,N) where
 %          P(x) = Q(0,0) + Q(1,1)*(x-x(0)) + Q(2,2)*(x-x(0))*
 %          (x-x(1)) +... + Q(N,N)*(x-x(0))*(x-x(1))*...*(x-x(N-1)).
 syms('OK', 'FLAG', 'N', 'I', 'X', 'Q', 'A', 'NAME', 'INP','OUP', 'J');
 syms('s','x','p1','sum','mult');
 TRUE = 1;
 FALSE = 0;
 fprintf(1,'Newtons form of the interpolation polynomial\n');
 N = k-1;
 %if N > 0
 %OK = TRUE;
 X = zeros(N+1);
 Q = zeros(N+1,N+1);
 for I = 0:N
 %fprintf(1,'Input X(%d) and F(X(%d)) ', I, I);
 %fprintf(1,'on separate lines\n');
 X(I+1,1) =ts(I+1) ;
 Q(I+1,1) = ys1(I+1);
 end
 %else

% STEP 1
 for I = 1:N
 for J = 1:I
 Q(I+1,J+1) = (Q(I+1,J) - Q(I,J)) / (X(I+1) - X(I-J+1));
 end
 end
% STEP 2
  %construct polynomial
sum=0;
mult=1;
for i=1:N
   mult=1;
   for j=1:i
      mult=mult*(x-X(j,1));
   end
```

```
   sum=sum+mult*Q(i+1,i+1);
end
p1=sum+Q(1,1);p1;
```

BDFeta.m

```
%This program prepared by SAMER AMIN ABU SA'
%It used to estimate the starting values for BDF method for eta- system
% It is based on difference formulas.

clear
clc
format short
syms('s1','s2','t','h','q','f2','a','exact1','exact2','x','k','eta')
display('Input q(t) :')
s1 = input(' ');
q = inline(s1,'t');
h=0.1;
tabl=zeros(6,5);
a=0;
eta=1;
tabl(1,1)=a;
alpha1=1;
tabl(1,3)=alpha1;
alpha2=-1;
tabl(1,5)=alpha2;
y1(1)=1;
y2(1)=-1;
tabl(1,2)=q(a)+eta*a*q(a);
tabl(1,4)=-q(a);
format short
for i=1:5
   x=(i-1)/10;tabl(i+1,1)=x+h;
   y2(i+1)=eta*y2(i)/(eta+1)-(q(x+h)-q(x))/((1+eta)*h);
   tabl(i+1,5)=y2(i+1);
   tabl(i+1,2)=q(x+h)+eta*(x+h)*q(x+h);
   y1(i+1)=q(x+h)-eta*(x+h)*y2(i+1);
   tabl(i+1,3)=y1(i+1);
   tabl(i+1,4)=-q(x+h);
end
format short

y1
y2
```

Execution Appendix C

Execution for example 1 by Power series:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} v_1' \\ v_2' \end{bmatrix}+\begin{bmatrix} 4 & 2 \\ -3 & -1 \end{bmatrix}\begin{bmatrix} v_1 \\ v_2 \end{bmatrix}=\begin{bmatrix} \cos(x)+4\sin(x) \\ -3\sin(x) \end{bmatrix} \qquad (4.1)$$

with initial values

$$\begin{bmatrix} v_1'(0) \\ v_2'(0) \end{bmatrix}=\begin{bmatrix} 0 \\ -1 \end{bmatrix}, \quad \begin{bmatrix} v_1(0) \\ v_2(0) \end{bmatrix}=\begin{bmatrix} 3 \\ -1 \end{bmatrix}$$

with exact solution $\begin{bmatrix} v_1 \\ v_2 \end{bmatrix}=\begin{bmatrix} 2e^{-x}-2e^{-2x}+\sin(x) \\ -3e^{-x}+2e^{-2x} \end{bmatrix}$ on the intreval [0,1]

Assume $v_1(x)=3x+e_1x^2$ and $v_2(x)=-1-x+e_2x^2$.

***Iteration* 1 by substitution v_1 ,v_2 in the system (4.1) will produce**

$1+\underline{2e_1x}+\underline{10x}+4e_1x^2+2e_2x^2-\cos(x)-\underline{4\sin(x)}=0$

$\underline{2e_2x-8x}-3e_1x^2-e_2x^2+\underline{3\sin(x)}=0$

Solving these two equations will give

$2e_1+6=0$ → $e_1=-3$

$2e_2-5=0$ → $e_2=2.5$

Append v_1,v_2 :

$$v_1(x) = 3x - 3x^2 + e_1 x^3,$$

$$v_2(x) = -1 - x + \frac{5}{2}x^2 + e_2 x^3.$$

***Iteration* 2 substitute v_1, v_2 in (4.1) will get**

$$1+4x+\underline{3e_1x^2}-7x^2+4e_1x^3+2e_2x^3-\cos(x)-4\sin(x)=0,$$

$$-3x+\underline{3e_2x^2+\frac{13}{2}x^2}-3e_1x^3-e_2x^3+3\sin(x)=0.$$

$$3e_1x^2-7x^2=0 \Rightarrow e_1=\frac{7}{3}$$

$$3e_2x^2+\frac{13}{2}x^2=0 \Rightarrow e_2=\frac{-13}{6}$$

Append v_1, v_2 :

$$v_1(x) = 3x - 3x^2 + \frac{7}{3}x^3 + e_1 x^4,$$

$$v_2(x) = -1 - x + \frac{5}{2}x^2 - \frac{13}{6}x^3 + e_2 x^4.$$

***Iteration* 3 substitute v_1, v_2**

$1+4x+\underline{4e_1x^3+5x^3}+4e_1x^4+2e_2x^4-\cos(x)-4\sin(x)=0,$

$-3x+\underline{4e_2x^3-\frac{29}{6}x^3}-3e_1x^4-e_2x^4+3\sin(x)=0.$

$4e_1x^3+5x^3=0\Rightarrow e_1=\frac{-5}{4}$

$4e_2x^3-\frac{29}{6}x^3=0\Rightarrow e_2=\frac{29}{24}.$

Append v_1, v_2 :

$v_1(x)=3x-3x^2+\frac{7}{3}x^3-\frac{5}{4}x^4+e_1x^5,$

$v_2(x)=-1-x+\frac{5}{2}x^2-\frac{13}{6}x^3-\frac{29}{24}x^4+e_2x^5.$

***Iteration* 4 substitute v_1, v_2**

$$1+4x+\underline{5e_1x^4-\frac{31}{12}x^4}+4e_1x^5+2e_2x^5-\cos(x)-4\sin(x)=0,$$

$$-3x+\underline{5e_2x^4+\frac{61}{24}x^4}-3e_1x^5-e_2x^5+3\sin(x)=0.$$

$$5e_1x^4-\frac{31}{12}x^4=0\Rightarrow e_1=\frac{31}{60},$$

$$5e_2x^4+\frac{61}{24}x^4=0\Rightarrow e_2=\frac{-61}{120}.$$

Append v_1, v_2 :

$$v_1(x)=3x-3x^2+\frac{7}{3}x^3-\frac{5}{4}x^4+\frac{31}{60}x^5+e_1x^6,$$

$$v_2(x)=-1-x+\frac{5}{2}x^2-\frac{13}{6}x^3-\frac{29}{24}x^4-\frac{61}{120}x^5+e_2x^6.$$

***Iteration* 5 substitute v_1, v_2**

$$1+4x+\underline{6e_1x^5+\frac{21}{20}x^5}+4e_1x^6+2e_2x^6-\cos(x)-4\sin(x)=0,$$

$$-3x+\underline{6e_2x^5-\frac{25}{24}x^5}-3e_1x^6-e_2x^6+3\sin(x)=0.$$

$$6e_1x^5+\frac{21}{20}x^5=0\Rightarrow e_1=\frac{-21}{120},$$

$$6e_2x^5-\frac{25}{24}x^5=0\Rightarrow e_2=\frac{25}{144}.$$

Append v_1 ,v_2:

$$v_1(x) = 3x - 3x^2 + \frac{7}{3}x^3 - \frac{5}{4}x^4 + \frac{31}{60}x^5 - \frac{21}{120}x^6 + e_1x^7,$$

$$v_2(x) = -1 - x + \frac{5}{2}x^2 - \frac{13}{6}x^3 - \frac{29}{24}x^4 - \frac{61}{120}x^5 + \frac{25}{144}x^6 + e_2x^7.$$

***Iteration* 6 substitute v_1 ,v_2**

$$1+4x+ \underline{7e_1x^6 - \frac{127}{360}x^6} + 4e_1x^7 + 2e_2x^7 - \cos(x) - 4\sin(x) = 0$$

$$-3x+ \underline{\underline{7e_2x^6 + \frac{253}{720}x^6}} - 3e_1x_7 - e_2x^7 + 3\sin(x) = 0$$

$$7e_1x^6 - \frac{127}{360}x^6 = 0 \Rightarrow e_1 = \frac{127}{2520}$$

$$7e_2x^6 + \frac{253}{720}x^6 = 0 \Rightarrow e_2 = \frac{-253}{5040}$$

Append v_1 ,v_2:

$$v_1(x) = 3x - 3x^2 + \frac{7}{3}x^3 - \frac{5}{4}x^4 + \frac{31}{60}x^5 - \frac{21}{120}x^6 + \frac{127}{2520}x^7 + e_1x^8,$$

$$v_2(x) = -1 - x + \frac{5}{2}x^2 - \frac{13}{6}x^3 - \frac{29}{24}x^4 - \frac{61}{120}x^5 + \frac{25}{144}x^6 - \frac{253}{5040}x^7 + e_2x^8.$$

***Iteration* 7 substitute v_1 ,v_2**

$$1+4x+\underline{8e_1x^7+\frac{17}{168}x^7}+4e_1x^8+2e_2x^8-\cos(x)-4\sin(x)=0$$

$$-3x+\underline{8e_2x^7-\frac{509}{5040}x^7}-3e_1x^8-e_2x^8+3\sin(x)=0$$

$$8e_1x^7+\frac{17}{168}x^7=0\Rightarrow e_1=\frac{-17}{1344}$$

$$8e_2x^7-\frac{509}{5040}x^7=0\Rightarrow e_2=\frac{509}{40320}$$

Append v_1 ,v_2:

$$v_1(x)=3x-3x^2+\frac{7}{3}x^3-\frac{5}{4}x^4+\frac{31}{60}x^5-\frac{21}{120}x^6+\frac{127}{2520}x^7-\frac{17}{1344}x^8,$$

$$v_2(x)=-1-x+\frac{5}{2}x^2-\frac{13}{6}x^3-\frac{29}{24}x^4-\frac{61}{120}x^5+\frac{25}{144}x^6-\frac{253}{5040}x^7+\frac{509}{40320}x^8.$$

It is noticed that the best polynomial for approximating y1 will be of degree 4, and for y2 will be of degree 8:

$$v_1(x)=3x-3x^2+\frac{7}{3}x^3-\frac{5}{4}x^4$$

$$v_2(x)=-1-x+\frac{5}{2}x^2-\frac{13}{6}x^3-\frac{29}{24}x^4-\frac{61}{120}x^5+\frac{25}{144}x^6-\frac{253}{5040}x^7+\frac{509}{40320}x^8.$$

These two power series v_1 ,v_2 can be transformed into Pade' series [1/3] ,[1/7] respectively:

$$v_1(x)=\frac{p_1(x)}{q_1(x)}=\frac{3x}{1+x+\frac{2}{9}x^2}$$

$$v_2(x)=\frac{p_2(x)}{q_2(x)}=\frac{-1-2.4663x}{1+1.4663x+1.0337x^2+0.4654x^3+0.1502x^4+0.0371x^5+0.0073x^6}$$

0	**0**	**0**	**-1.0000**	**-1.0000**
0.1000	**0.2720**	**0.2722**	**-1.0771**	**-1.0771**
0.2000	**0.4955**	**0.4963**	**-1.1156**	**-1.1156**
0.3000	**0.6795**	**0.6818**	**-1.1248**	**-1.1248**
0.4000	**0.8314**	**0.8359**	**-1.1123**	**-1.1123**
0.5000	**0.9567**	**0.9643**	**-1.0838**	**-1.0838**
0.6000	**1.0599**	**1.0714**	**-1.0440**	**-1.0441**
0.7000	**1.1442**	**1.1609**	**-0.9966**	**-0.9966**
0.8000	**1.2122**	**1.2357**	**-0.9442**	**-0.9443**
0.9000	**1.2659**	**1.2981**	**-0.8891**	**-0.8893**
1.0000	**1.3066**	**1.3500**	**-0.8330**	**-0.8333**

Execution for example (1) by BDF execution:

Let us solve this system using BDF method by running algorithm BDF.m, execution will be

FIRST DIVISION : INPUT THE SYSTEM, INITIAL CONDITIONS AND STEP SIZE h

Input the function F_1 in terms of t and y_1 ... y_2

'$-4y_1-2y_2+\cos(t)+4\sin(t)$'

Input the function F2 in terms of t and y_1 ... y2

'3*y1+y2-3*sin(t)'

Input left and right endpoints on separate lines.

0

1

Input the initial condition alpha(1)

0

Input the initial condition alpha(2)

-1

Input a positive integer for the number of subintervals

5

INPUT step size h : 0.1

h = 0.1000

SECOND DIVISION: SAVE STARTING VALUES USING RUNGE-KUTTA FOR SYSTEM

x	y1	y2
0.000	0.00000000	-1.00000000
0.100	0.27204137	-1.07704549
0.200	0.49548169	-1.11554333
0.300	0.67952186	-1.12482019
0.400	0.83138741	-1.11228951
0.500	0.95671390	-1.08381950

THIRD DIVISION : MAIN LOOP ... APPLYING BDF

Iteration = 1

x = 0.6000

EVALUATE NEW y1

k = 2

Newtons form of the interpolation polynomial

$$p_1(x) = 1.2533x + 0.3301$$

y_1 = 1.0820

k = 3

Newtons form of the interpolation polynomial

$p_1(x)$=1.5187x+ 0.2239- 1.3270(x-0.3)(x-0.4)

y1 = 1.0555

k = 4

Newtons form of the interpolation polynomial

$$p_1(x) = 0.1274 + 1.8404x - 1.6087(x-0.2)(x-0.3) + 0.9393(x-0.2)(x-0.3)(x-0.4)$$

y1 = 1.0611

k = 5

Newtons form of the interpolation polynomial

$$p_1(x) = 0.0486 + 2.2344x - 1.97(x-0.1)(x-0.2) + 1.2043(x-0.1)(x-0.2)(x-0.3)\ldots$$
$$-0.6625(x-0.1)(x-0.2)(x-0.3)(x-0.4)$$

y1 = 1.0595

k = 6

Newtons form of the interpolation polynomial

$p_1(x)$=2.7204x-2.4301x(x-0.1)+1.5335 x(x-0.1)(x-0.2)-0.8231 x(x-0.1)(x-0.2)(x-0.3)+0.3212 x(x-0.1)(x-0.2)(x-0.3)(x-0.4)

y1 = 1.0599

EVALUATE NEW y2

k = 2

Newtons form of the interpolation polynomial

$p_2(x)$= 0.2847x- 1.2262

y2 =-1.0553

k = 3

Newtons form of the interpolation polynomial

$p_2(x)$= - 1.1624+0.1253x+ 0.7970(x-0.3)(x-0.4)

y2 = -1.0394

k = 4

Newtons form of the interpolation polynomial

$p_2(x)$=-1.0970-0.0928x+1.0904(x-0.2)(x-0.3)-0.9780(x-0.2)(x-0.3)(x-0.4)

y2 = -1.0453

k = 5

Newtons form of the interpolation polynomial

$p_2(x)$=-1.0385-0.3850x+1.4610(x-0.1)(x-0.2)-1.2356(x-0.1)(x-0.2)(x-0.3)+

0.6438(x-0.1)(x-0.2)(x-0.3)(x-0.4)

y2 = -1.0437

k = 6

Newtons form of the interpolation polynomial

p_2(x)=-1+0.7705x+1.9274x(x-0.1)-1.5544 x(x-0.1)(x-0.2)+0.7972x(x-0.1)(x-0.2)(x-0.3)- 0.3067x(x-0.1)(x-0.2)(x-0.3)(x-0.4)

y2 = -1.0441

Iteration = 2

curt = 0.7000

EVALUATE NEW y1(n+1)

k = 2

Newtons form of the interpolation polynomial

p1= 1.0322x+ 0.4406

cury1 = 1.1632

k = 3

Newtons form of the interpolation polynomial

p1=1.2533x+ 0.3301- 1.1054(x-0.4)(x-0.5)

cury1 = 1.1410

k = 4

Newtons form of the interpolation polynomial

p1=1.5187x+0.2239-1.3270(x-0.3)(x-0.4)+ 0.7385(x-0.3)*(x-0.4)*(x-0.5)

cury1 = 1.1455

k = 5

Newtons form of the interpolation polynomial

p1=1.8404x+0.1274-1.6087(x-0.2)(x-0.3)+0.9393(x-0.2)(x-0.3)(x-0.4)-

0.5019(x-0.2)(x-0.3)(x-0.4)(x-0.5)

cury1 = 1.1443

EVALUATE NEW y2(n+1)

k = 2

Newtons form of the interpolation polynomial

p2 =0.3972x-1.2824

cury2 = -1.0044

k = 3

Newtons form of the interpolation polynomial

p2= 0.2847x- 1.2262+0.5624(x-0.4)(x-0.5)

cury2 = -0.9931

k = 4

Newtons form of the interpolation polynomial

p2=0.1253x-1.1624+0.7970(x-0.3)(x-0.4)-0.7818(x-0.3)(x-0.4)(x-0.5)

cury2 = -0.9978

k = 5

Newtons form of the interpolation polynomial

p2 = -0.0928x-1.0970+1.0904(x-0.2)(x-0.3)-0.9780(x-0.2)(x-0.3)(x-2/5)+

0.4905(x-0.2)(x-0.3)(x-0.4)(x-0.5)

cury2 = -0.9966

Iteration = 3

curt = 0.8000

EVALUATE NEW y1(n+1)

k = 2

Newtons form of the interpolation polynomial

p1 =0.8434x+0.5539

cury1 = 1.2286

k = 3

Newtons form of the interpolation polynomial

p1 =

1.0322x+ 0.4406-0.9441(x-0.5)(x-0.6)

cury1 = 1.2097

k = 4

Newtons form of the interpolation polynomial

p1=1.2533x+0.3301-1.1054(x-0.4)(x-0.5)+ 0.5378(x-0.4)(x-0.5)(x-0.6)

cury1 = 1.2130

k = 5

Newtons form of the interpolation polynomial

p1 =

1.5187x+0.2239-1.3270(x-0.3)(x-2/5)+0.7385(x-0.3)(x-0.4)(x-0.5)-

0.5019(x-0.3)(x-0.4)(x-0.5)(x-0.6)

cury1 = 1.2117

k = 6

Newtons form of the interpolation polynomial

p1 =

1.8404x+ 0.1274- 1.6087*(x-1/5)*(x-3/10)+ 0.9393*(x-1/5)*(x-3/10)*(x-

2/5)-0.5019*(x-1/5)*(x-3/10)*(x-2/5)*(x-1/2)+8.7486 e-014*(x-1/5)*(x-

3/10)*(x-2/5)*(x-1/2)*(x-3/5)

cury1 = 1.2117

EVALUATE NEW y2(n+1)

k = 2

Newtons form of the interpolation polynomial

p2 = 0.4745*x-1.3288

cury2 = -0.9492

k = 3

Newtons form of the interpolation polynomial

p2=3577513407339501/9007199254740992*x-

2310186982270700 1/18014398509481984+3483377481602101/9007199254740992*

(x-1/2)*(x-3/5)

cury2 = -0.9415

k = 4

Newtons form of the interpolation polynomial

p2=5128700957163681/18014398509481984*x-

5522176723487335 1/4503599627370496+316613415236769/562949953421312*(x-

2/5)*(x-1/2)-2637395270310339/4503599627370496*(x-2/5)*(x-1/2)*(x-3/5)

cury2 = -0.9450

k = 5

Newtons form of the interpolation polynomial

p2=4514652808665441/36028797018963968*x-

4188031414563391 23/36028797018963680+7178436382077403/9007199254740992*(x-

3/10)*(x-2/5)-7042072460963665/9007199254740992*(x-3/10)*(x-

2/5)*(x-1/2)+8836409601714937/18014398509481984*(x-3/10)*(x-

2/5)*(x-1/2)*(x-3/5)

cury2 = -0.9438

k = 6

Newtons form of the interpolation polynomial

p2=-3342341348510079/36028797018963968*x-

19761607949240628 1/18014398509481984+1227655337058675/11258 9906842624*(x-

1/5)*(x-3/10)-8809354381306657/9007199254740992*(x-1/5)*(x-

3/10)*(x-2/5)+8836409601714961/18014398509481984*(x-1/5)*(x-

3/10)*(x-2/5)*(x-1/2)-

6755399441055745/2535301200456458802993406410752*(x-1/5)*(x-

3/10)*(x-2/5)*(x-1/2)*(x-3/5)

cury2 = -0.9438

Iteration = 4

curt = 0.9000

EVALUATE NEW y1(n+1)

k = 2

Newtons form of the interpolation polynomial

p1=6077882871116695/9007199254740992*x+12104293790969151/1801 4398509481984

cury1 = 1.2792

k = 3

Newtons form of the interpolation polynomial

p1=3798210898727491/4503599627370496*x+12472924117619437/225179981368524 80-949086828961429/1125899906842624*(x-3/5)*(x-7/10)

cury1 = 1.2624

k = 4

Newtons form of the interpolation polynomial

p1=4648549995135151/4503599627370496*x+3968762731358583/900719925474099 2-4251695482038301/4503599627370496*(x-1/2)*(x-3/5)+6071308882567799/18014398509481984*(x-1/2)*(x-3/5)*(x-7/10)

cury1 = 1.2644

k = 5

Newtons form of the interpolation polynomial

p1=2822101737686233/2251799813685248*x+14865546255606341/450359962737049 60-311141712574161/281474976710656*(x-2/5)*(x-1/2)+2421906397160917/4503599627370496*(x-2/5)*(x-1/2)*(x-3/5)-1130098970648709/2251799813685248*(x-2/5)*(x-1/2)*(x-3/5)*(x-7/10)

cury1 = 1.2632

k = 6

Newtons form of the interpolation polynomial

p1=6839416090030577/4503599627370496*x+10084695796973899/450359962737049 60-1494015768322639/1125899906842624*(x-3/10)*(x-2/5)+1662992786839967/2251799813685248*(x-3/10)*(x-2/5)*(x-1/2)-

2260197941297543/4503599627370496*(x-3/10)*(x-2/5)*(x-1/2)*(x-3/5)+125/2251799813685248*(x-3/10)*(x-2/5)*(x-1/2)*(x-3/5)*(x-7/10)

cury1 = 1.2632

EVALUATE NEW y2(n+1)

k = 2

Newtons form of the interpolation polynomial

p2=1190103470690919/2251799813685248*x-3077324423629595 3/22517998136852480

cury2 = -0.8909

k = 3

Newtons form of the interpolation polynomial

p2=4274188903659921/9007199254740992*x-5984470104572876 3/4503599627370496*(x-3/5)*(x-7/10)

cury2 = -0.8855

k = 4

Newtons form of the interpolation polynomial

p2=3577513407339501/9007199254740992*x-2310186982270700 1/18014398509481984+3483377481602101/9007199254740992*(x-

1/2)*(x-3/5)-876877155069439/2251799813685248*(x-1/2)*(x-3/5)*(x-7/10)

cury2 = -0.8879

k = 5

Newtons form of the interpolation polynomial

p2=5128700957163681/18014398509481984*x-5522176723487335 1/45035996273704960+316613415236769/562949953421312*(x-2/5)*(x-1/2)-2637395270310339/4503599627370496*(x-2/5)*(x-1/2)*(x-3/5)+4418204800857305/9007199254740992*(x-2/5)*(x-1/2)*(x-3/5)*(x-7/10)

cury2 = -0.8867

k = 6

Newtons form of the interpolation polynomial

p2=4514652808665441/36028797018963968*x-41880314145633912 3/360287970189639680+717843638207740 3/9007199254740992*(x-3/10)*(x-2/5)-7042072460963665/9007199254740992*(x-3/10)*(x-2/5)*(x-1/2)+8836409601714937/18014398509481984*(x-3/10)*(x-2/5)*(x-1/2)*(x-3/5)-327/9007199254740992*(x-3/10)*(x-2/5)*(x-1/2)*(x-3/5)*(x-7/10)

cury2 = -0.8867

Iteration = 5

curt = 1

EVALUATE NEW y1(n+1)

k = 2

Newtons form of the interpolation polynomial

p1=4741483211255461/9007199254740992*x+4450791639608477/5629499534213120

cury1 = 1.3170

k = 3

Newtons form of the interpolation polynomial

p1=6077882871116695/9007199254740992*x+12104293790969151/18014398509481984-835249787413271/1125899906842624*(x-7/10)*(x-4/5)

cury1 = 1.3022

k = 4

Newtons form of the interpolation polynomial

p1=3798210898727491/4503599627370496*x+12472924117619437/22517998136852480-949086828961429/1125899906842624*(x-3/5)*(x-7/10)+3035654441284213/9007199254740992*(x-3/5)*(x-7/10)*(x-4/5)

cury1 = 1.3042

k = 5

Newtons form of the interpolation polynomial

p1=4648549995135151/4503599627370496*x+3968762731358583/900719925474099 2-

4251695482038301/4503599627370496*(x-1/2)*(x-

3/5)+6071308882567799/18014398509481984*(x-1/2)*(x-3/5)*(x-

7/10)+3135/36028797018963968*(x-1/2)*(x-3/5)*(x-7/10)*(x-4/5)

cury1 = 1.3042

k = 6

Newtons form of the interpolation polynomial

p1=2822101737686233/2251799813685248*x+14865546255606341/450359962737049 60-

311141712574161/281474976710656*(x-2/5)*(x-

1/2)+2421906397160917/4503599627370496*(x-2/5)*(x-1/2)*(x-3/5)-

1130098970648709/2251799813685248*(x-2/5)*(x-1/2)*(x-3/5)*(x-

7/10)+1130098970648905/1125899906842624*(x-2/5)*(x-1/2)*(x-

3/5)*(x-7/10)*(x-4/5)

cury1 = 1.3054

EVALUATE NEW y2(n+1)

k = 2

Newtons form of the interpolation polynomial

p2=5036188344650771/9007199254740992*x-

15662396580035071/11258999068426240

cury2 = -0.8320

k = 3

Newtons form of the interpolation polynomial

p2=1190103470690919/2251799813685248*x-

3077324423629595 3/2251799813685248 0+2757744618870949/18014398509481984*(x-

7/10)*(x-4/5)

cury2 = -0.8289

k = 4

Newtons form of the interpolation polynomial

p2=4274188903659921/9007199254740992*x-

5984470104572876 3/4503599627370496+1215562447759387/4503599627370496*(x-

3/5)*(x-7/10)-7015017240555329/18014398509481984*(x-3/5)*(x-

7/10)*(x-4/5)

cury2 = -0.8312

k = 5

Newtons form of the interpolation polynomial

p2=3577513407339501/9007199254740992*x-

2310186982270700 1/18014398509481984+3483377481602101/9007199254740992*(x-

1/2)*(x-3/5)-876877155069439/2251799813685248*(x-1/2)*(x-3/5)*(x-

7/10)+915/36028797018963968*(x-1/2)*(x-3/5)*(x-7/10)*(x-4/5)

cury2 = -0.8312

k = 6

Newtons form of the interpolation polynomial

p2=5128700957163681/18014398509481984*x-

5522176723487335 1/4503599627370496+316613415236769/562949953421312*(x-

2/5)*(x-1/2)-2637395270310339/4503599627370496*(x-2/5)*(x-1/2)*(x-

3/5)+4418204800857305/9007199254740992*(x-2/5)*(x-1/2)*(x-3/5)*(x-

7/10)-1104551200214269/1125899906842624*(x-2/5)*(x-1/2)*(x-3/5)*(x-

7/10)*(x-4/5)

cury2 = -0.8324

FOURTH DIVISION : NUMERICAL RESULTS WITH EXACT SOLUTION

>> table

table =

0	**0**	**0**	**-1.0000**	**-1.0000**
0.1000	**0.2720**	**0.2720**	**-1.0771**	**-1.0770**
0.2000	**0.4955**	**0.4955**	**-1.1156**	**-1.1155**
0.3000	**0.6795**	**0.6795**	**-1.1248**	**-1.1248**
0.4000	**0.8314**	**0.8314**	**-1.1123**	**-1.1123**
0.5000	**0.9567**	**0.9567**	**-1.0838**	**-1.0838**
0.6000	**1.0599**	**1.0599**	**-1.0440**	**-1.0441**
0.7000	**1.1442**	**1.1443**	**-0.9966**	**-0.9966**
0.8000	**1.2122**	**1.2117**	**-0.9442**	**-0.9438**
0.9000	**1.2659**	**1.2644**	**-0.8891**	**-0.8879**
1.0000	**1.3066**	**1.3054**	**-0.8330**	**-0.8324**

Execution for example (2)

Power series execution:

```
tabcoef =

    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0
power =     2

f =
[ 2*e1*x-x*(1+2*e2*x)-x+e1*x^2+(-1-x)*(x+e2*x^2)]
[                          x+e2*x^2-sin(x)]
 ans =

2*e1*x-3*x-3*e2*x^2+e1*x^2-x^2-e2*x^3
 ans =

x+e2*x^2-sin(x)

here are analytic manipulations for determinig vector e by
consisting linear system and solve it or input e directly from key board

power =     2
Add new vector e :
Choice of addition method:
1. From keyboard
2. Input system and solve it :
Choose 1 or 2 please
 1
coeffecient :1.5

newcoef =    1.5000

coeffecient :0

newcoef =    1.5000         0

 esv1 =1-x+3/2*x^2
 esv2 =x

f =
[ 2*x+3*e1*x^2-x*(1+3*e2*x^2)+3/2*x^2+e1*x^3+(-1-x)*(x+e2*x^3)]
[                                    x+e2*x^3-sin(x)]

 ans =
3*e1*x^2-4*e2*x^3+1/2*x^2+e1*x^3-e2*x^4
 ans =
x+e2*x^3-sin(x)
 ans =
here are analytic manipulations for determinig vector e by
consisting linear system and solve it or input e directly from key board

power =     3

Add new vector e :
Choice of addition method:
1. From keyboard
```

```
2. Input system and solve it :
Choose 1 or 2 please
 1
coeffecient :-1/6

newcoef =

   -0.1667         0

coeffecient :-1/6

newcoef =

   -0.1667   -0.1667

 esv1 =1-x+3/2*x^2-1/6*x^3
 esv2 =x-1/6*x^3

f =
[ 2*x+x^2+4*e1*x^3-x*(1-1/2*x^2+4*e2*x^3)-1/6*x^3+e1*x^4+(-1-x)*(x-1/6*x^3+e2*x^4)]
[                                          x-1/6*x^3+e2*x^4-sin(x)]

 ans =
 4*e1*x^3+1/2*x^3-5*e2*x^4+e1*x^4+1/6*x^4-e2*x^5
 ans =
 x-1/6*x^3+e2*x^4-sin(x)
 here are analytic manipulations for determinig vector e by
consisting linear system and solve it or input e directly from key board
ans =
power =
     4

Add new vector e :
Choice of addition method:
1. From keyboard
2. Input system and solve it :
Choose 1 or 2 please
 1
coeffecient :-1/8

newcoef =
   -0.1250   -0.1667

coeffecient :0

newcoef =
   -0.1250         0
esv1 =1-x+3/2*x^2-1/6*x^3-1/8*x^4
 esv2 =x-1/6*x^3
 f =
 [ 2*x+x^2-2/3*x^3+5*e1*x^4-x*(1-1/2*x^2+5*e2*x^4)-1/8*x^4+e1*x^5+(-1-x)*(x-
1/6*x^3+e2*x^5)]
[                                              x-1/6*x^3+e2*x^5-sin(x)]
 ans =
 5*e1*x^4-6*e2*x^5+1/24*x^4+e1*x^5-e2*x^6
 ans =
 x-1/6*x^3+e2*x^5-sin(x)

here are analytic manipulations for determining vector e by
```

consisting linear system and solve it or input e directly from key board
ans =
power = 5
Add new vector e :
Choice of addition method:
1. From keyboard
2. Input system and solve it :
Choose 1 or 2 please
1
coeffecient :-1/120

newcoef =

-0.0083 0

coeffecient :1/120

newcoef =

-0.0083 0.0083

esv1 =
1-x+3/2*x^2-1/6*x^3-1/8*x^4-1/120*x^5
esv2 =
x-1/6*x^3+1/120*x^5
f =
[2*x+x^2-2/3*x^3-1/6*x^4+6*e1*x^5-x*(1-1/2*x^2+1/24*x^4+6*e2*x^5)-1/120*x^5+e1*x^6+(-1-x)*(x-1/6*x^3+1/120*x^5+e2*x^6)]
[x-1/6*x^3+1/120*x^5+e2*x^6-sin(x)]

ans =
6*e1*x^5-7/120*x^5-7*e2*x^6+e1*x^6-1/120*x^6-e2*x^7
ans =
x-1/6*x^3+1/120*x^5+e2*x^6-sin(x)

here are analytic manipulations for determining vector e by
consisting linear system and solve it or input e directly from key board

ans =
power = 6

Add new vector e :
Choice of addition method:
1. From keyboard
2. Input system and solve it :
Choose 1 or 2 please
1
coeffecient :1/120

newcoef =

0.0083 0.0083

coeffecient :0

newcoef =

0.0083 0

```
esv1 =
1-x+3/2*x^2-1/6*x^3-1/8*x^4-1/120*x^5+1/120*x^6
 esv2 =
 x-1/6*x^3+1/120*x^5

tabcoef =

   1.0000   -1.0000    1.5000   -0.1667   -0.1250   -0.0083    0.0083         0         0         0
        0    1.0000         0   -0.1667         0    0.0083         0         0         0         0

This is Pade Approximation.

m+n must equal 6
Input m and n on separate lines.
order of numerator 3
order of denominator  3

AA =

    1     0     0     0     0     0     0

AA =

    1    -1     0     0     0     0     0

AA =

   1.0000   -1.0000    1.5000         0         0         0         0


AA =

   1.0000   -1.0000    1.5000   -0.1667         0         0         0


AA =

   1.0000   -1.0000    1.5000   -0.1667   -0.1250         0         0


AA =

   1.0000   -1.0000    1.5000   -0.1667   -0.1250   -0.0083         0


AA =

   1.0000   -1.0000    1.5000   -0.1667   -0.1250   -0.0083    0.0083

PADE RATIONAL APPROXIMATION

Denominator Coefficients Q(0), ..., Q(M)
  1.00000000 -0.12658960  0.07121387  0.00291908
Numerator Coefficients P(0), ..., P(N)
  1.00000000 -1.12658960  1.69780347 -0.42484586

ans =
```

102 111 114 109 97 116

p1 =

1-1949/1730*x+7343/4325*x^2-7653342580632429/18014398509481984*x^3

q1 =

1-219/1730*x+308/4325*x^2

This is Pade Approximation.

m+n must equal 6
Input m and n on separate lines.
order of numerator 5
order of denominator 1

AA =

0 0 0 0 0 0 0

AA =

0 1 0 0 0 0 0

AA =

0 1 0 0 0 0 0

AA =

0 1.0000 0 -0.1667 0 0 0

AA =

0 1.0000 0 -0.1667 0 0 0

AA =

0 1.0000 0 -0.1667 0 0.0083 0

AA =

0 1.0000 0 -0.1667 0 0.0083 0

PADE RATIONAL APPROXIMATION

Denominator Coefficients Q(0), ..., Q(M)
1.00000000 0.00000000
Numerator Coefficients P(0), ..., P(N)
0.00000000 1.00000000 0.00000000 -0.16666667 0.00000000 0.00833333

ans =

102 111 114 109 97 116

p2 =

x-1/6*x^3+1/120*x^5

q2 =

1

ans =

x exact1 pade1 exact2 pade2

tabl =

0	**1.0000**	**1.0000**	**0**	**0**
0.1000	**0.9148**	**0.9148**	**0.0998**	**0.0998**
0.2000	**0.8585**	**0.8585**	**0.1987**	**0.1987**
0.3000	**0.8295**	**0.8295**	**0.2955**	**0.2955**
0.4000	**0.8261**	**0.8262**	**0.3894**	**0.3894**
0.5000	**0.8462**	**0.8466**	**0.4794**	**0.4794**
0.6000	**0.8876**	**0.8882**	**0.5646**	**0.5646**
0.7000	**0.9475**	**0.9485**	**0.6442**	**0.6442**
0.8000	**1.0232**	**1.0249**	**0.7174**	**0.7174**
0.9000	**1.1116**	**1.1143**	**0.7833**	**0.7834**
1.0000	**1.2094**	**1.2136**	**0.8415**	**0.8417**

FIRST DIVISION : INPUT THE SYSTEM, INITIAL CONDITIONS AND STEP SIZE h

This is the Runge-Kutta Method for Systems of m equations
This program uses the file F.m. If the number of equations
exceeds 7, then F.m must be changed.
Input the number of equations
2
Input the function F_(1) in terms of t and y1 ... y2
For example: y1-t^2+1
't*cos(t)-y1+(1+t)*sin(t)'
Input the function F_(2) in terms of t and y1 ... y2
For example: y1-t^2+1
'cos(t)'
Input left and right endpoints on separate lines.
0
1
Input the initial condition alpha(1)
1
Input the initial condition alpha(2)
0
Input a positive integer for the number of subintervals
5
INPUT step size h : 0.1

H =

0.1000

Choice of output method:
1. Output to screen
2. Output to text file
Please enter 1 or 2
1
RUNGE-KUTTA METHOD FOR SYSTEMS OF DIFFERENTIAL EQUATIONS

T	W1	W2
0.000	1.00000000	0.00000000
0.100	0.91482098	0.09983342
0.200	0.85846501	0.19866934
0.300	0.82947481	0.29552022
0.400	0.82608801	0.38941836
0.500	0.84624412	0.47942556

SECOND DIVISION: SAVE STARTING VALUES USING RUNGE-KUTTA FOR SYSTEM

THIRD DIVISION : MAIN LOOP ... APPLYING BDF
Iteration = 1
curt =
0.6000

EVALUATE NEW y1(n+1)
k = 2

Newtons form of the interpolation polynomial

p1 =3631001463713361/18014398509481984*x+3357269512852609 9/45035996273704960
cury1 = 0.8664
k = 3

Newtons form of the interpolation polynomial

p1 =
-
4880896315753601/144115188075855872*x+1210041876214677123/1441151880758558720+2650695
939489101/2251799813685248*(x-3/10)*(x-2/5)
cury1 = 0.8899

k = 4
Newtons form of the interpolation polynomial
p1 =
-
2611205132073509/9007199254740992*x+644891129690891/703687441776640+720671597605909/56
2949953421312*(x-1/5)*(x-3/10)-3093206012460467/9007199254740992*(x-1/5)*(x-3/10)*(x-2/5)
cury1 =

0.8879

k = 5

Newtons form of the interpolation polynomial
p1 =
-
2538046994107455/4503599627370496*x+8747584231271685/9007199254740992+616222214035350

1/4503599627370496*(x-1/10)*(x-1/5)-2645662396708193/9007199254740992*(x-1/10)*(x-1/5)*(x-3/10)-1118859039380685/9007199254740992*(x-1/10)*(x-1/5)*(x-3/10)*(x-2/5)

cury1 =

0.8876

ans =

EVALUATE NEW y2(n+1)

k = 2

Newtons form of the interpolation polynomial

p2 =

63336935902559/70368744177664*x+2647176062407617/90071992547409920

cury2 = 0.5694

k = 3

Newtons form of the interpolation polynomial

p2 =

8457592467994097/9007199254740992*x+1245317372541439/90071992547409920-3504646724665451/18014398509481984*(x-3/10)*(x-2/5)

cury2 =

0.5655

k = 4

Newtons form of the interpolation polynomial

p2 =

8723551672228807/9007199254740992*x+223719879918653/45035996273704960-664898010586775/4503599627370496*(x-1/5)*(x-3/10)-1408424470530585/9007199254740992*(x-1/5)*(x-3/10)*(x-2/5)

cury2 =

0.5646

Iteration = 2

curt =

0.7000

ans =

EVALUATE NEW y1(n+1)

k =

2

Newtons form of the interpolation polynomial

p1 =

3723612505755191/9007199254740992*x+11520966277511929/18014398509481984

cury1 =

0.9289

k =

3

Newtons form of the interpolation polynomial

p1 =

3631001463713361/18014398509481984*x+33572695128526099/45035996273704960+4770279434746277/4503599627370496*(x-2/5)*(x-1/2)

cury1 =

0.9501

k = 4

Newtons form of the interpolation polynomial
p1 =
-
4880896315753601/144115188075855872*x+1210041876214677123/1441151880758558720+2650695939489101/2251799813685248*(x-3/10)*(x-2/5)-1770374814106417/4503599627370496*(x-3/10)*(x-2/5)*(x-1/2)
cury1 = 0.9478
k = 5
Newtons form of the interpolation polynomial

p1 =
-
2611205132073509/9007199254740992*x+644891129690891/703687441776640+720671597605909/562949953421312*(x-1/5)*(x-3/10)-3093206012460467/9007199254740992*(x-1/5)*(x-3/10)*(x-2/5)-8950872315047341/72057594037927936*(x-1/5)*(x-3/10)*(x-2/5)*(x-1/2)

cury1 =
0.9475
ans =
EVALUATE NEW y2(n+1)

k = 2
Newtons form of the interpolation polynomial
p2 =
7672157654829177/9007199254740992*x+482202676589949/9007199254740992

cury2 = 0.6498

k = 3

Newtons form of the interpolation polynomial

p2 =
63336935902559/70368744177664*x+2647176062407617/90071992547409920-4349701406983751/18014398509481984*(x-2/5)*(x-1/2)

cury2 = 0.6450
k = 4

Newtons form of the interpolation polynomial

p2 =

8457592467994097/9007199254740992*x+1245317372541439/90071992547409920-3504646724665451/18014398509481984*(x-3/10)*(x-2/5)-5633697882122001/36028797018963968*(x-3/10)*(x-2/5)*(x-1/2)

cury2 = 0.6440

k =
5
Newtons form of the interpolation polynomial

p2 =
8723551672228807/9007199254740992*x+223719879918653/45035996273704960-66489801058675/4503599627370496*(x-1/5)*(x-3/10)-1408424470530585/9007199254740992*(x-

1/5)*(x-3/10)*(x-2/5)+7454688836321281/3169126500570573503741758013 44*(x-1/5)*(x-3/10)*(x-2/5)*(x-1/2)

cury2 =
0.6440
k = 6

Newtons form of the interpolation polynomial

p2 =
4451174015888705/4503599627370496*x+4492352037005/4503599627370496-7151854381944119/72057594037927936*(x-1/10)*(x-1/5)-2905428156203567/18014398509481984*(x-1/10)*(x-1/5)*(x-3/10)+442896075711985/36028797018963968*(x-1/10)*(x-1/5)*(x-3/10)*(x-2/5)-885792151422275/36028797018963968*(x-1/10)*(x-1/5)*(x-3/10)*(x-2/5)*(x-1/2)

cury2 = 0.6440

Iteration = 3
curt =
0.8000
ans =

EVALUATE NEW y1(n+1)
k = 2
Newtons form of the interpolation polynomial

p1 =
5392426685015791/9007199254740992*x+11897986577999011/22517998136852480

cury1 = 1.0073

k = 3

Newtons form of the interpolation polynomial

p1 =
3723612505755191/9007199254740992*x+11520966277511929/18014398509481984+4172035448151501/4503599627370496*(x-1/2)*(x-3/5)

cury1 = 1.0258
k = 4

Newtons form of the interpolation polynomial

p1 =
3631001463713361/18014398509481984*x+33572695128526099/45035996273704960+4770279434746277/4503599627370496*(x-2/5)*(x-1/2)-1994146621982587/4503599627370496*(x-2/5)*(x-1/2)*(x-3/5)

cury1 = 1.0232

ans =EVALUATE NEW y2(n+1)

k = 2

Newtons form of the interpolation polynomial

p2 =
447042627868687/562949953421312*x+3969440209740299/45035996273704960
cury2 = 0.7234
k = 3

Newtons form of the interpolation polynomial

p2 =
7672157654829177/9007199254740992*x+482202676589949/9007199254740992-5194756089301851/18014398509481984*(x-1/2)*(x-3/5)
cury2 =
0.7177
k =
4
Newtons form of the interpolation polynomial

p2 =
63336935902559/70368744177664*x+2647176062407617/90071992547409920-4349701406983751/18014398509481984*(x-2/5)*(x-1/2)-1408424470530167/9007199254740992*(x-2/5)*(x-1/2)*(x-3/5)

cury2 = 0.7167

k = 5

Newtons form of the interpolation polynomial

p2 =
8457592467994097/9007199254740992*x+1245317372541439/90071992547409920-3504646724665451/18014398509481984*(x-3/10)*(x-2/5)-5633697882122001/36028797018963968*(x-3/10)*(x-2/5)*(x-1/2)+3664122499563521/396140812571321687967719751 68*(x-3/10)*(x-2/5)*(x-1/2)*(x-3/5)

cury2 = 0.7167

k = 6

Newtons form of the interpolation polynomial

p2 =
8723551672228807/9007199254740992*x+223719879918653/45035996273704960-66489801058677 5/4503599627370496*(x-1/5)*(x-3/10)-1408424470530585/9007199254740992*(x-1/5)*(x-3/10)*(x-2/5)+7454688836321281/31691265005705735037417580 1344*(x-1/5)*(x-3/10)*(x-2/5)*(x-1/2)+5464572790046723/396140812571321687967719751 68*(x-1/5)*(x-3/10)*(x-2/5)*(x-1/2)*(x-3/5)

cury2 = 0.7167

Iteration = 4

curt = 0.9000

EVALUATE NEW y1(n+1)

k = 2

Newtons form of the interpolation polynomial

p1 =

3410971634819237/4503599627370496*x+1879266510981863 1/45035996273704960
cury1 = 1.0989
k = 3

Newtons form of the interpolation polynomial

p1 =
5392426685015791/9007199254740992*x+11897986577999011/22517998136852480+7147582923113413/9007199254740992*(x-3/5)*(x-7/10)
cury1 = 1.1148

k = 4

Newtons form of the interpolation polynomial

p1 =
3723612505755191/9007199254740992*x+11520966277511929/18014398509481984+4172035448151501/4503599627370496*(x-1/2)*(x-3/5)-249268327747831/562949953421312*(x-1/2)*(x-3/5)*(x-7/10)

cury1 = 1.1121

k = 5

Newtons form of the interpolation polynomial

p1 =
3631001463713361/18014398509481984*x+33572695128526099/45035996273704960+4770279434746277/4503599627370496*(x-2/5)*(x-1/2)-1994146621982587/4503599627370496*(x-2/5)*(x-1/2)*(x-3/5)-305/9007199254740992*(x-2/5)*(x-1/2)*(x-3/5)*(x-7/10)

cury1 = 1.1121

k = 6

Newtons form of the interpolation polynomial

p1 =-
4880896315753601/144115188075855872*x+121004187621467712 3/144115188075855720+2650695939489101/2251799813685248*(x-3/10)*(x-2/5)-1770374814106417/4503599627370496*(x-3/10)*(x-2/5)*(x-1/2)-4475436157523401/36028797018963968*(x-3/10)*(x-2/5)*(x-1/2)*(x-3/5)+4475436157522181/18014398509481984*(x-3/10)*(x-2/5)*(x-1/2)*(x-3/5)*(x-7/10)

cury1 = 1.1124

EVALUATE NEW y2(n+1)

k = 2

Newtons form of the interpolation polynomial

p2 =
1658301609242201/2251799813685248*x+1446901210248989/11258999068426240

cury2 = 0.7913

k = 3
Newtons form of the interpolation polynomial

p2 =
447042627868687/562949953421312*x+3969440209740299/45035996273704960-2597378044650939/9007199254740992*(x-3/5)*(x-7/10)
cury2 =
0.7855
k = 4

Newtons form of the interpolation polynomial

p2 =
7672157654829177/9007199254740992*x+482202676589949/9007199254740992-5194756089301851/18014398509481984*(x-1/2)*(x-3/5)-6333186975989759/1267650600228229401496703205376*(x-1/2)*(x-3/5)*(x-7/10)

cury2 = 0.7855
k = 5

Newtons form of the interpolation polynomial

p2 =
63336935902559/70368744177664*x+2647176062407617/90071992547409920-4349701406983751/18014398509481984*(x-2/5)*(x-1/2)-1408424470530167/9007199254740992*(x-2/5)*(x-1/2)*(x-3/5)+3521061176325305/9007199254740992*(x-2/5)*(x-1/2)*(x-3/5)*(x-7/10)

cury2 = 0.7865

k = 6

Newtons form of the interpolation polynomial

p2 =
8457592467994097/9007199254740992*x+1245317372541439/90071992547409920-3504646724665451/18014398509481984*(x-3/10)*(x-2/5)-5633697882122001/36028797018963968*(x-3/10)*(x-2/5)*(x-1/2)+3664122499563521/39614081257132168796771975168*(x-3/10)*(x-2/5)*(x-1/2)*(x-3/5)+440132647040559/562949953421312*(x-3/10)*(x-2/5)*(x-1/2)*(x-3/5)*(x-7/10)

cury2 = 0.7874

Iteration = 5

curt = 1

EVALUATE NEW y1(n+1)

k = 2

Newtons form of the interpolation polynomial

p1 =
4006081129811611/4503599627370496*x+7015894574939821/22517998136852480

cury1 = 1.2011

k = 3

Newtons form of the interpolation polynomial

p1 =
3410971634819237/4503599627370496*x+18792665109818631/45035996273704960+2975547474961869/4503599627370496*(x-7/10)*(x-4/5)

cury1 = 1.2143

k = 4

Newtons form of the interpolation polynomial

p1 =
5392426685015791/9007199254740992*x+11897986577999011/22517998136852480+7147582923113413/9007199254740992*(x-3/5)*(x-7/10)-7976586487931165/18014398509481984*(x-3/5)*(x-7/10)*(x-4/5)

cury1 = 1.2117

k = 5

Newtons form of the interpolation polynomial

p1 =
3723612505755191/9007199254740992*x+11520966277511929/18014398509481984+4172035448151501/4503599627370496*(x-1/2)*(x-3/5)-249268327747831/562949953421312*(x-1/2)*(x-3/5)*(x-7/10)-2865/36028797018963968*(x-1/2)*(x-3/5)*(x-7/10)*(x-4/5)

cury1 = 1.2117

k = 6

Newtons form of the interpolation polynomial

p1 =
3631001463713361/18014398509481984*x+33572695128526099/45035996273704960+4770279434746277/4503599627370496*(x-2/5)*(x-1/2)-1994146621982587/4503599627370496*(x-2/5)*(x-1/2)*(x-3/5)-305/9007199254740992*(x-2/5)*(x-1/2)*(x-3/5)*(x-7/10)-1645/18014398509481984*(x-2/5)*(x-1/2)*(x-3/5)*(x-7/10)*(x-4/5)

cury1 = 1.2117

EVALUATE NEW y2(n+1)

k = 2

Newtons form of the interpolation polynomial

p2 =
6113730828038611/9007199254740992*x+7865507276716731/45035996273704960

cury2 = 0.8534

k = 3
Newtons form of the interpolation polynomial

p2 =
1658301609242201/2251799813685248*x+1446901210248989/11258999068426240-
649344511162741/2251799813685248*(x-7/10)*(x-4/5)

cury2 = 0.8476

k = 4

Newtons form of the interpolation polynomial

p2 =
447042627868687/562949953421312*x+3969440209740299/45035996273704960-
2597378044650939/9007199254740992*(x-3/5)*(x-7/10)-
1466015503701333/158456325028528675187087900672*(x-3/5)*(x-7/10)*(x-4/5)

cury2 = 0.8476

k = 5

Newtons form of the interpolation polynomial

p2 =
7672157654829177/9007199254740992*x+482202676589949/9007199254740992-
5194756089301851/18014398509481984*(x-1/2)*(x-3/5)-
6333186975989759/1267650600228229401496703205376*(x-1/2)*(x-3/5)*(x-7/10)-
6743671317026131/6338253001141147007483516026 88*(x-1/2)*(x-3/5)*(x-7/10)*(x-4/5)

cury2 = 0.8476

k = 6
Newtons form of the interpolation polynomial

p2 =63336935902559/70368744177664*x+2647176062407617/90071992547409920-
4349701406983751/18014398509481984*(x-2/5)*(x-1/2)-1408424470530167/9007199254740992*(x-
2/5)*(x-1/2)*(x-3/5)+3521061176325305/9007199254740992*(x-2/5)*(x-1/2)*(x-3/5)*(x-7/10)-
3521061176325401/4503599627370496*(x-2/5)*(x-1/2)*(x-3/5)*(x-7/10)*(x-4/5)

cury2 = 0.8467

ans =

FOURTH DIVISION : NUMERICAL RESULTS WITH EXACT SOLUTION

>> table
table =

0	1.0000	1.0000	0	0
0.1000	0.9148	0.9148	0.0998	0.0998
0.2000	0.8585	0.8585	0.1987	0.1987
0.3000	0.8295	0.8295	0.2955	0.2955
0.4000	0.8261	0.8261	0.3894	0.3894
0.5000	0.8462	0.8462	0.4794	0.4794
0.6000	0.8876	0.8876	0.5646	0.5646
0.7000	0.9475	0.9475	0.6442	0.6440
0.8000	1.0232	1.0232	0.7174	0.7177

0.9000 1.1116 1.1121 0.7833 0.7855
1.0000 1.2094 1.2117 0.8415 0.8467

Execution for example 3

Using power series

From initial condition, the solutions of the system can be supposed as

$v_1(x) = y_{0,1} + e_1 x \Rightarrow v_1(x) = 1 + e_1 x$,

$v_2(x) = y_{0,2} + e_2 x \Rightarrow v_2(x) = 1 + e_2 x$,

$v_3(x) = y_{0,3} + e_3 x \Rightarrow v_3(x) = e_3 x$.

The following are the execution for Matlab code using 8 iterations with power series of degree 8:

***Iteration* 1:**

By substitution v_1, v_2, v_3 in ( ) will get

e1-2*e2*x+3*x^2*e3+e1*x-x-e2*x^2+e3*x^3=0,

e2-2*e3*x-1-e2*x+x^2*e3=0,

e3*x-sin(x)=0, so we have the vector

$$e = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

v1 = 1

v2 =1+x

v3 =x

***Iteration* 2:**

2*e1*x-3*x-3*e2*x^2+2*x^2+4*e3*x^3+e1*x^2-e2*x^3+x^3+x^4*e3=0

2*e2*x-3*x-3*x^2*e3-e2*x^2+x^2+e3*x^3=0

x+x^2*e3-sin(x)=0

$$e = \begin{bmatrix} 1.5 \\ 1.5 \\ 0 \end{bmatrix}$$

v1 =1+3/2*x^2

v2 =1+x+3/2*x^2

v3 =x

***Iteration* 3:**

3*e1*x^2-x^2-4*e2*x^3+5*x^4*e3+e1*x^3-1/2*x^3-e2*x^4+x^5*e3=0

3*e2*x^2-4*e3*x^3-1/2*x^2-e2*x^3+x^4*e3=0

x+e3*x^3-sin(x)=0

$$e=\begin{bmatrix} \frac{1}{3} \\ \frac{1}{6} \\ \frac{-1}{6} \end{bmatrix}$$

v1 =1+3/2*x^2+1/3*x^3

v2 =1+x+3/2*x^2+1/6*x^3

v3 =x-1/6*x^3

***Iteration* 4:**

-5/6*x^3+4*e1*x^3-5*e2*x^4+6*x^5*e3-x^4+e1*x^4-e2*x^5+x^6*e3-1/6*x^5=0

4*e2*x^3+1/2*x^3-5*x^4*e3-e2*x^4-1/6*x^4+x^5*e3=0

x-1/6*x^3+x^4*e3-sin(x)=0

$$e=\begin{bmatrix} \frac{5}{24} \\ \frac{-1}{8} \\ 0 \\ \end{bmatrix}$$

v1 =1+3/2*x^2+1/3*x^3+5/24*x^4

v2 =1+x+3/2*x^2+1/6*x^3-1/8*x^4

v3 =x-1/6*x^3

***Iteration* 5:**

-1/6*x^4+5*e1*x^4-6*e2*x^5+7*x^6*e3-1/24*x^5+e1*x^5-e2*x^6+x^7*e3=0

5*e2*x^4-6*x^5*e3-1/24*x^4-e2*x^5+x^6*e3=0

x-1/6*x^3+x^5*e3-sin(x)=0

$$e=\begin{bmatrix}\frac{1}{30}\\ \frac{1}{120}\\ \frac{1}{120}\end{bmatrix}$$

v1 =1+3/2*x^2+1/3*x^3+5/24*x^4+1/30*x^5

v2 =1+x+3/2*x^2+1/6*x^3-1/8*x^4+1/120*x^5

v3 =x-1/6*x^3+1/120*x^5

***Iteration* 6 :**

e1*x^6-7/120*x^5+6*e1*x^5-7*e2*x^6+8*x^7*e3+…

+1/20*x^6- e2*x^7+x^8*e3+1/120*x^7=0

6*e2*x^5-7/120*x^5-7*x^6*e3-e2*x^6+1/120*x^6+x^7*e3=0

x-1/6*x^3+1/120*x^5+x^6*e3-sin(x)=0

$$e=\begin{bmatrix}7/720\\ 7/720\\ 0\end{bmatrix}$$

v1 =1+3/2*x^2+1/3*x^3+5/24*x^4+1/30*x^5+7/720*x^6

v2 =1+x+3/2*x^2+1/6*x^3-1/8*x^4+1/120*x^5+7/720*x^6

v3 =x-1/6*x^3+1/120*x^5.

***Iteration* 7**

7*e1*x^6-1/120*x^6-e2*x^8+e1*x^7-1/720*x^7+x^9*e3-8*e2*x^7+9*x^8*e3=0

7*e2*x^6-8*x^7*e3-1/720*x^6-e2*x^7+x^8*e3=0

x-1/6*x^3+1/120*x^5+x^7*e3-sin(x) =0

$$e=\begin{bmatrix}\frac{1}{840}\\ \frac{1}{5040}\\ \frac{1}{5040}\end{bmatrix}$$

v1(x) =1+3/2*x^2+1/3*x^3+5/24*x^4+1/30*x^5+7/720*x^6+1/840*x^7

v2(x) =x+1+3/2*x^2+1/6*x^3-1/8*x^4+1/120*x^5+7/720*x^6+1/5040*x^7

v3(x) =x-1/6*x^3+1/120*x^5+1/5040*x^7

***Iteration* 8**

x^10*e3+1/630*x^8+e1*x^8-9*e2*x^8+1/5040*x^9-e2*x^9+8*e1*x^7-

1/560*x^7+10*x^9*e3=0

8*e2*x^7-1/560*x^7-9*x^8*e3-e2*x^8+1/5040*x^8+x^9*e3=0

x-1/6*x^3+1/120*x^5+1/5040*x^7+x^8*e3-sin(x)=0

$$e = \begin{bmatrix} \frac{1}{4480} \\ \frac{1}{4480} \\ \frac{-1}{5040} \end{bmatrix}$$

v1(x)=1+3/2*x^2+1/3*x^3+5/24*x^4+1/30*x^5+7/720*x^6+1/840*x^7+1/4480*x^8

v2(x) =x+1+3/2*x^2+1/6*x^3-1/8*x^4+1/120*x^5+7/720*x^6+1/5040*x^7+1/4480*x^8

v3(x) =x-1/6*x^3+1/120*x^5+1/5040*x^7-1/5040*x^8

This is Pade Approximation.

m+n must equal 8
Input m and n on separate lines.
order of numerator 4
order of denominator 4

AA =

1 0 0 0 0 0 0 0 0

AA =

1 0 0 0 0 0 0 0 0

AA =

1.0000 0 1.5000 0 0 0 0 0 0

AA =

1.0000 0 1.5000 0.3333 0 0 0 0 0

AA =

1.0000 0 1.5000 0.3333 0.2083 0 0 0 0

AA =

1.0000 0 1.5000 0.3333 0.2083 0.0333 0 0 0

AA =

1.0000 0 1.5000 0.3333 0.2083 0.0333 0.0097 0 0

AA =

1.0000 0 1.5000 0.3333 0.2083 0.0333 0.0097 0.0012 0

AA =

1.0000 0 1.5000 0.3333 0.2083 0.0333 0.0097 0.0012 0.0002

PADE RATIONAL APPROXIMATION

Denominator Coefficients Q(0), ..., Q(M)
1.00000000 -0.14578018 -0.03674512 0.00619061 0.00048588
Numerator Coefficients P(0), ..., P(N)
1.00000000 -0.14578018 1.46325488 0.12085367 0.10510814

p1 =

1-
5252284416630351/36028797018963968*x+6589914150013469/4503599627370496*x^2+8708424934
678601/72057594037927936*x^3+3786919947090701/36028797018963968*x^4

q1 =

1-5252284416630351/36028797018963968*x-
2647764656676393/72057594037927936*x^2+1784320687708035/288230376151711744*x^3

This is Pade Approximation.

m+n must equal 8
Input m and n on separate lines.
order of numerator 4
order of denominator 4

AA =

1 0 0 0 0 0 0 0 0

AA =

1 1 0 0 0 0 0 0 0

AA =

1.0000 1.0000 1.5000 0 0 0 0 0 0

AA =

1.0000 1.0000 1.5000 0.1667 0 0 0 0 0

AA =

1.0000 1.0000 1.5000 0.1667 -0.1250 0 0 0 0

AA =

1.0000 1.0000 1.5000 0.1667 -0.1250 0.0083 0 0 0

AA =

1.0000 1.0000 1.5000 0.1667 -0.1250 0.0083 0.0097 0 0

AA =

1.0000 1.0000 1.5000 0.1667 -0.1250 0.0083 0.0097 0.0002 0

AA =

1.0000 1.0000 1.5000 0.1667 -0.1250 0.0083 0.0097 0.0002 0.0002

PADE RATIONAL APPROXIMATION

Denominator Coefficients Q(0), ..., Q(M)
1.00000000 -0.37109032 -0.02188273 -0.03227693 -0.00265710
Numerator Coefficients P(0), ..., P(N)
1.00000000 0.62890968 1.10702695 -0.44412848 -0.25460652

ans =

102 111 114 109 97 116

p2 =

1+2832357386366157/4503599627370496*x+1246401537090769/1125899906842624*x^2-2000176856153425/4503599627370496*x^3-1146645815766603/4503599627370496*x^4

q2 =

1-6684968964017357/18014398509481984*x-6307267308053071/2882303761517117 44*x^2-4651596187317417/144115188075855872*x^3

This is Pade Approximation.

m+n must equal 8
Input m and n on separate lines.
order of numerator 4
order of denominator 4

AA =

0 0 0 0 0 0 0 0 0

AA =

0 1 0 0 0 0 0 0 0

AA =

0 1 0 0 0 0 0 0 0

AA =

0 1.0000 0 -0.1667 0 0 0 0 0

AA =

0 1.0000 0 -0.1667 0 0 0 0 0

AA =

0 1.0000 0 -0.1667 0 0.0083 0 0 0

AA =

0 1.0000 0 -0.1667 0 0.0083 0 0 0

AA =

0 1.0000 0 -0.1667 0 0.0083 0 0.0002 0

AA =

0 1.0000 0 -0.1667 0 0.0083 0 0.0002 -0.0002

PADE RATIONAL APPROXIMATION

Denominator Coefficients Q(0), ..., Q(M)

```
  1.00000000  0.32258065  0.08163265  0.01612903  0.00527211
Numerator Coefficients P(0), ..., P(N)
  0.00000000  1.00000000  0.32258065 -0.08503401 -0.03763441

ans =

   102   111   114   109    97   116


p3 =

x+10/31*x^2-25/294*x^3-7/186*x^4


q3 =

1+10/31*x+4/49*x^2+1/62*x^3


ans =

    x    exact1    v1(x)    pade1    exact2    v2(x)    pade2    exact3    v3(x)    pade3


tabl =

       0    1.0000    1.0000    1.0000    1.0000         0         0
  0.1000    1.0154    1.0154    1.1152    1.1152    0.0998    0.0998
  0.2000    1.0630    1.0630    1.2611    1.2611    0.1987    0.1987
  0.3000    1.1458    1.1458    1.4385    1.4385    0.2955    0.2955
  0.4000    1.2670    1.2671    1.6476    1.6475    0.3894    0.3895
  0.5000    1.4309    1.4309    1.8884    1.8880    0.4794    0.4796
  0.6000    1.6421    1.6422    2.1609    2.1599    0.5646    0.5650
  0.7000    1.9062    1.9065    2.4647    2.4626    0.6442    0.6449
  0.8000    2.2298    2.2303    2.7994    2.7951    0.7174    0.7186
  0.9000    2.6202    2.6212    3.1646    3.1562    0.7833    0.7854
  1.0000    3.0862    3.0879    3.5598    3.5445    0.8415    0.8448




Using  BDF


FIRST DIVISION : INPUT THE SYSTEM, INITIAL CONDITIONS AND STEP SIZE h


This is the Runge-Kutta Method for Systems of m equations

This program uses the file F.m.  If the number of equations

exceeds 7, then F.m must be changed.

Input the number of equations

 3
```

Input the function F_(1) in terms of t and y1 ... y3

For example: y1-t^2+1

'(2*t+1)*y2-y1-(2*t^2+t)*sin(t)'

Input the function F_(2) in terms of t and y1 ... y3

For example: y1-t^2+1

'y2+t*cos(t)-(t-1)*sin(t)'

Input the function F_(3) in terms of t and y1 ... y3

For example: y1-t^2+1

'cos(t)'

Input left and right endpoints on separate lines.

0

1

Input the initial condition alpha(1)

1

Input the initial condition alpha(2)

1

Input the initial condition alpha(3)

0

Input a positive integer for the number of subintervals

5

INPUT step size h : 0.1

H =

0.1000

Choice of output method:

1. Output to screen

2. Output to text file

Please enter 1 or 2

1

RUNGE-KUTTA METHOD FOR SYSTEMS OF DIFFERENTIAL EQUATIONS

T W1 W2 W3

0.000 1.00000000 1.00000000 0.00000000

0.100 1.01535526 1.11515403 0.09983342

0.200 1.06301272 1.26113613 0.19866934

0.300 1.14577786 1.43851406 0.29552022

0.400 1.26705240 1.64759087 0.38941836

0.500 1.43089416 1.88843246 0.47942556

SECOND DIVISION: SAVE STARTING VALUES USING RUNGE-KUTTA FOR SYSTEM

ys3 = 0

0.0998

0.1987

0.2955

0.3894

0.4794

THIRD DIVISION : MAIN LOOP ... APPLYING BDF

Iteration = 1

curt = 0.6000

EVALUATE NEW y1(n+1)

k = 2

Newtons form of the interpolation polynomial

cury1 = 1.5947

k = 3

Newtons form of the interpolation polynomial

cury1 = 1.6373

k = 4

Newtons form of the interpolation polynomial

cury1 = 1.6414

k = 5

Newtons form of the interpolation polynomial

cury1 = 1.6420

EVALUATE NEW y2(n+1)

k = 2

Newtons form of the interpolation polynomial

cury2 = 2.1293

k = 3

Newtons form of the interpolation polynomial

cury2 = 2.1610

k = 4

Newtons form of the interpolation polynomial

cury2 = 2.1611

k = 5

Newtons form of the interpolation polynomial

cury2 = 2.1609

EVALUATE NEW y3(n+1)

k = 2

Newtons form of the interpolation polynomial

cury3 = 0.5694

k = 3

Newtons form of the interpolation polynomial

cury3 = 0.5655

k = 4

Newtons form of the interpolation polynomial

cury3 = 0.5646

Iteration = 2

curt = 0.7000

EVALUATE NEW y1(n+1)

k = 2

Newtons form of the interpolation polynomial

cury1 = 1.8531

k = 3

Newtons form of the interpolation polynomial

cury1 = 1.9004

k = 4

Newtons form of the interpolation polynomial

cury1 = 1.9051

k = 5

Newtons form of the interpolation polynomial

cury1 = 1.9058

k = 6

Newtons form of the interpolation polynomial

cury1 = 1.9058

EVALUATE NEW y2(n+1)

k = 2

Newtons form of the interpolation polynomial

cury2 = 2.4333

k = 3

Newtons form of the interpolation polynomial

cury2 = 2.4649

k = 4

Newtons form of the interpolation polynomial

cury2 = 2.4647

EVALUATE NEW y3(n+1)

k = 2

Newtons form of the interpolation polynomial

cury3 = 0.6498

k = 3

Newtons form of the interpolation polynomial

cury3 = 0.6450

k = 4

Newtons form of the interpolation polynomial

cury3 = 0.6440

k = 5

Newtons form of the interpolation polynomial

cury3 = 0.6440

k = 6

Newtons form of the interpolation polynomial

cury3 = 0.6440

Iteration = 3

curt = 0.8000

EVALUATE NEW y1(n+1)

k = 2

Newtons form of the interpolation polynomial

cury1 = 2.1696

k = 3

Newtons form of the interpolation polynomial

cury1 = 2.2222

k = 4

Newtons form of the interpolation polynomial

cury1 = 2.2276

k = 5

Newtons form of the interpolation polynomial

cury1 = 2.2282

k = 6

Newtons form of the interpolation polynomial

cury1 = 2.2282

EVALUATE NEW y2(n+1)

k = 2

Newtons form of the interpolation polynomial

cury2 = 2.7686

k = 3

Newtons form of the interpolation polynomial

cury2 = 2.8000

k = 4

Newtons form of the interpolation polynomial

cury2 = 2.7998

k = 5

Newtons form of the interpolation polynomial

cury2 = 2.7998

k = 6

Newtons form of the interpolation polynomial

cury2 = 2.8001

EVALUATE NEW y3(n+1)

k = 2

Newtons form of the interpolation polynomial

cury3 = 0.7234

k = 3

Newtons form of the interpolation polynomial

cury3 = 0.7177

k = 4

Newtons form of the interpolation polynomial

cury3 = 0.7167

k = 5

Newtons form of the interpolation polynomial

cury3 = 0.7167

k = 6

Newtons form of the interpolation polynomial

cury3 = 0.7167

Iteration = 4

curt = 0.9000

EVALUATE NEW y1(n+1)

k = 2

Newtons form of the interpolation polynomial

cury1 = 2.5507

k = 3

Newtons form of the interpolation polynomial

cury1 = 2.6094

k = 4

Newtons form of the interpolation polynomial

cury1 = 2.6154

k = 5

Newtons form of the interpolation polynomial

cury1 = 2.6161

k = 6

Newtons form of the interpolation polynomial

cury1 = 2.6161

EVALUATE NEW y2(n+1)

k = 2

Newtons form of the interpolation polynomial

cury2 = 3.1349

k = 3

Newtons form of the interpolation polynomial

cury2 = 3.1662

k = 4

Newtons form of the interpolation polynomial

cury2 = 3.1660

k = 5

Newtons form of the interpolation polynomial

cury2 = 3.1660

k = 6

Newtons form of the interpolation polynomial

cury2 = 3.1660

EVALUATE NEW y3(n+1)

k = 2

Newtons form of the interpolation polynomial

cury3 = 0.7913

k = 3

Newtons form of the interpolation polynomial

cury3 = 0.7855

k = 4

Newtons form of the interpolation polynomial

cury3 = 0.7855

k = 5

Newtons form of the interpolation polynomial

cury3 = 0.7865

k = 6

Newtons form of the interpolation polynomial

cury3 = 0.7874

Iteration = 5

curt = 1

EVALUATE NEW y1(n+1)

k = 2

Newtons form of the interpolation polynomial

cury1 = 3.0039

k = 3

Newtons form of the interpolation polynomial

cury1 = 3.0692

k = 4

Newtons form of the interpolation polynomial

cury1 = 3.0759

k = 5

Newtons form of the interpolation polynomial

cury1 = 3.0766

k = 6

Newtons form of the interpolation polynomial

cury1 = 3.0766

EVALUATE NEW y2(n+1)

k = 2

Newtons form of the interpolation polynomial

cury2 = 3.5322

k = 3

Newtons form of the interpolation polynomial

cury2 = 3.5633

k = 4

Newtons form of the interpolation polynomial

cury2 = 3.5631

k = 5

Newtons form of the interpolation polynomial

cury2 = 3.5631

k = 6

Newtons form of the interpolation polynomial

cury2 = 3.5631

EVALUATE NEW y3(n+1)

k = 2

Newtons form of the interpolation polynomial

cury3 = 0.8534

k = 3

Newtons form of the interpolation polynomial

cury3 = 0.8476

k = 4

Newtons form of the interpolation polynomial

cury3 = 0.8476

k = 5

Newtons form of the interpolation polynomial

cury3 = 0.8476

k = 6

Newtons form of the interpolation polynomial

cury3 = 0.8467

FOURTH DIVISION : NUMERICAL RESULTS WITH EXACT SOLUTION

>> table

table =

0	1.0000	1.0000	1.0000	1.0000	0	0
0.1000	1.0154	1.0154	1.1152	1.1152	0.0998	0.0998
0.2000	1.0630	1.0630	1.2611	1.2611	0.1987	0.1987
0.3000	1.1458	1.1458	1.4385	1.4385	0.2955	0.2955
0.4000	1.2670	1.2671	1.6476	1.6476	0.3894	0.3894
0.5000	1.4309	1.4309	1.8884	1.8884	0.4794	0.4794
0.6000	1.6421	1.6420	2.1609	2.1609	0.5646	0.5646
0.7000	1.9062	1.9058	2.4647	2.4647	0.6442	0.6440
0.8000	2.2298	2.2282	2.7994	2.7998	0.7174	0.7177
0.9000	2.6202	2.6161	3.1646	3.1660	0.7833	0.7855
1.0000	3.0862	3.0766	3.5598	3.5631	0.8415	0.8467

Execution for example 4

Using Power series

tabcoef =

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

v1 =

1+2*x+e1*x^2

v2 =

-1-x+e2*x^2

power = 2

f =[1+2*x+e1*x^2+x*(-1-x+e2*x^2)-exp(x)]

[2*e1*x+x*(-1+2*e2*x)-2*x+2*e2*x^2]

ans = 1+x+e1*x^2-x^2+e2*x^3-exp(x)

ans =2*e1*x-3*x+4*e2*x^2

here are analytic manipulations for determinig vector e by

consisting linear system and solve it or input e directly from key board

power = 2

Add new vector e :

Choice of addition method:

1. From keyboard

2. Input system and solve it :

Choose 1 or 2 please

1

coeffecient :1

newcoef =

1

coeffecient :-1/4

newcoef = 1.0000e+000 -2.5000e-001

esv1 =1+2*x+x^2

esv2 =-1-x-1/4*x^2

f =[1+2*x+x^2+e1*x^3+x*(-1-x-1/4*x^2+e2*x^3)-exp(x)]

[3*e1*x^2+x*(-1-1/2*x+3*e2*x^2)-1/2*x^2+2*e2*x^3]

ans =

1+x+e1*x^3-1/4*x^3+e2*x^4-exp(x)

ans =

3*e1*x^2-x-x^2+5*e2*x^3

here are analytic manipulations for determinig vector e by

consisting linear system and solve it or input e directly from key board

power = 3

Add new vector e :

Choice of addition method:

1. From keyboard

2. Input system and solve it :

Choose 1 or 2 please

1

coeffecient :1/4

newcoef =

2.5000e-001 -2.5000e-001

coeffecient :-1/20

newcoef =

2.5000e-001 -5.0000e-002

esv1 =1+2*x+x^2+1/4*x^3

esv2 =-1-x-1/4*x^2-1/20*x^3

f =

[1+2*x+x^2+1/4*x^3+e1*x^4+x*(-1-x-1/4*x^2-1/20*x^3+e2*x^4)-exp(x)]

[1/4*x^2+4*e1*x^3+x*(-1-1/2*x-3/20*x^2+4*e2*x^3)-1/10*x^3+2*e2*x^4]

ans

1+x+e1*x^4-1/20*x^4+e2*x^5-exp(x)

ans =

-1/4*x^2+4*e1*x^3-x-1/4*x^3+6*e2*x^4

here are analytic manipulations for determinig vector e by

consisting linear system and solve it or input e directly from key board

ans =

power =

4

Add new vector e :

Choice of addition method:

1. From keyboard

2. Input system and solve it :

Choose 1 or 2 please

1

coeffecient :1/20

newcoef =

5.0000e-002 -5.0000e-002

coeffecient :-1/120

newcoef =

5.0000e-002 -8.3333e-003

esv1 =

1+2*x+x^2+1/4*x^3+1/20*x^4

esv2 =

-1-x-1/4*x^2-1/20*x^3-1/120*x^4

f =

[1+2*x+x^2+1/4*x^3+1/20*x^4+e1*x^5+x*(-1-x-1/4*x^2-1/20*x^3-1/120*x^4+e2*x^5)-exp(x)]

[1/4*x^2+1/10*x^3+5*e1*x^4+x*(-1-1/2*x-3/20*x^2-1/30*x^3+5*e2*x^4)-1/60*x^4+2*e2*x^5]

ans =

1+x+e1*x^5-1/120*x^5+e2*x^6-exp(x)

ans =

-1/4*x^2-1/20*x^3+5*e1*x^4-x-1/20*x^4+7*e2*x^5

ans =

here are analytic manipulations for determinig vector e by

ans =

consisting linear system and solve it or input e directly from key board

ans =

power =

5

Add new vector e :

Choice of addition method:

1. From keyboard

2. Input system and solve it :

Choose 1 or 2 please

1

coeffecient :1/120

newcoef =

8.3333e-003 -8.3333e-003

coeffecient :-1/840

newcoef =

8.3333e-003 -1.1905e-003

esv1 =

1+2*x+x^2+1/4*x^3+1/20*x^4+1/120*x^5

esv2 =

-1-x-1/4*x^2-1/20*x^3-1/120*x^4-1/840*x^5

tabcoef =

Columns 1 through 8

1.0000e+000 2.0000e+000 1.0000e+000 2.5000e-001 5.0000e-002 8.3333e-003 0 0

-1.0000e+000 -1.0000e+000 -2.5000e-001 -5.0000e-002 -8.3333e-003 -1.1905e-003 0 0

Columns 9 through 10

0 0

0 0

This is Pade Approximation.

m+n must equal 5

Input m and n on separate lines.

order of numerator 3

order of denominator 2

AA =

1 0 0 0 0 0

AA =

1 2 0 0 0 0

AA =

1 2 1 0 0 0

AA =

1.0000e+000 2.0000e+000 1.0000e+000 2.5000e-001 0 0

AA =

1.0000e+000 2.0000e+000 1.0000e+000 2.5000e-001 5.0000e-002 0

AA =

1.0000e+000 2.0000e+000 1.0000e+000 2.5000e-001 5.0000e-002 8.3333e-003

PADE RATIONAL APPROXIMATION

Denominator Coefficients Q(0), ..., Q(M)

1.00000000 -0.33333333 0.03333333

Numerator Coefficients P(0), ..., P(N)

1.00000000 1.66666667 0.36666667 -0.01666667

ans =

102 111 114 109 97 116

p1 =

1+5/3*x+11/30*x^2-300239975158035/18014398509481984*x^3

q1 =

1-1/3*x

This is Pade Approximation.

m+n must equal 5

Input m and n on separate lines.

order of numerator 3

order of denominator 2

AA =

-1 0 0 0 0 0

AA =

-1 -1 0 0 0 0

AA =

-1.0000e+000 -1.0000e+000 -2.5000e-001 0 0 0

AA =

-1.0000e+000 -1.0000e+000 -2.5000e-001 -5.0000e-002 0 0

AA =

-1.0000e+000 -1.0000e+000 -2.5000e-001 -5.0000e-002 -8.3333e-003 0

AA =

-1.0000e+000 -1.0000e+000 -2.5000e-001 -5.0000e-002 -8.3333e-003 -1.1905e-003

PADE RATIONAL APPROXIMATION

Denominator Coefficients Q(0), ..., Q(M)

1.00000000 -0.28571429 0.02380952

Numerator Coefficients P(0), ..., P(N)

-1.00000000 -0.71428571 0.01190476 -0.00238095

ans =

102 111 114 109 97 116

p2 =

-1-5/7*x+3431314001806021/288230376151711744*x^2-

686262800361227/288230376151711744*x^3

q2 =

1-2/7*x

tabl =

0 1.0000 1.0000 -1.0000 -1.0000

0.1000 1.2157 1.2107 -1.1052 -1.1028

0.2000 1.4657 1.4441 -1.2214 -1.2116

0.3000 1.7548 1.7028 -1.3499 -1.3270

0.4000 2.0886 1.9895 -1.4918 -1.4496

0.5000 2.4731 2.3075 -1.6487 -1.5802

0.6000 2.9154 2.6605 -1.8221 -1.7196

0.7000 3.4234 3.0530 -2.0138 -1.8687

0.8000 4.0060 3.4902 -2.2255 -2.0287

0.9000 4.6732 3.9784 -2.4596 -2.2009

1.0000 5.4366 4.5250 -2.7183 -2.3867

Using BDF

ans =

Input q(t) :

'exp(t)'

y1 =

1.0000 1.2078 1.4402 1.7067 2.0136 2.3671

y2 =

-1.0000 -1.0259 -1.0941 -1.1893 -1.3045 -1.4367

ys1 =

1.0000

1.2078

1.4402

1.7067

2.0136

2.3671

ys2 =

-1.0000

-1.0259

-1.0941

-1.1893

-1.3045

-1.4367

tss =

Columns 1 through 10

0 0.1000 0.2000 0.3000 0.4000 0.5000 0.6000 0.7000 0.8000 0.9000

Column 11

1.0000

ans =

FIRST DIVISION : INPUT THE SYSTEM, INITIAL CONDITIONS AND STEP SIZE h

ans =

SECOND DIVISION: SAVE STARTING VALUES USING RUNGE-KUTTA FOR SYSTEM

ans =

THIRD DIVISION : MAIN LOOP ... APPLYING BDF

Iteration = 1

curt = 0.6000

EVALUATE NEW y1(n+1)

k = 2

Newtons form of the interpolation polynomial

p1 =3979654517613871/1125899906842624*x+3376370441361363/5629499534213120

cury1 = 2.7205

k = 3

Newtons form of the interpolation polynomial

p1

=3456122259882631/1125899906842624*x+8846869913647687/11258999068426240+5235322577312

401/2251799813685248*(x-3/10)*(x-2/5)

cury1 = 2.7670

k = 4

Newtons form of the interpolation polynomial

p1 =

5999600820045893/2251799813685248*x+20431670926453479/22517998136852480+4563218498596845/2251799813685248*(x-1/5)*(x-3/10)+4480693858103707/4503599627370496*(x-1/5)*(x-3/10)*(x-2/5)

cury1 = 2.7730

k = 5

Newtons form of the interpolation polynomial

p1

=2617308358466205/1125899906842624*x+4392327826536089/4503599627370496+7649841031134829/4503599627370496*(x-1/10)*(x-1/5)+76906039898899/70368744177664*(x-1/10)*(x-1/5)*(x-3/10)-2206463477129145/9007199254740992*(x-1/10)*(x-1/5)*(x-3/10)*(x-2/5)

cury1 = 2.7724

k = 6

Newtons form of the interpolation polynomial

p1 =

4678257712760375/2251799813685248*x+2781795020860175/2251799813685248*x*(x-1/10)+6954169964714929/4503599627370496*x*(x-1/10)*(x-1/5)-2540229263981741/2251799813685248*x*(x-1/10)*(x-1/5)*(x-3/10)+7954453578797819/4503599627370496*x*(x-1/10)*(x-1/5)*(x-3/10)*(x-2/5)+1

cury1 = 2.7745

EVALUATE NEW y2(n+1)

k = 2

Newtons form of the interpolation polynomial

p2 =-5955437037279541/4503599627370496*x-8731828143170189/11258999068426240

cury2 = -1.5690

k = 3

Newtons form of the interpolation polynomial

p2 =-5186692237926881/4503599627370496*x-38002291770091397/45035996273704960-3843723996763301/4503599627370496*(x-3/10)*(x-2/5)

cury2 =

-1.5860

k = 4

Newtons form of the interpolation polynomial

p2 =-8578186288782497/9007199254740992*x-40695089050698293/45035996273704960-8975990935356325/9007199254740992*(x-1/5)*(x-3/10)+2147571569716205/4503599627370496*(x-1/5)*(x-3/10)*(x-2/5)

cury2 = -1.5832

k = 5

Newtons form of the interpolation polynomial

p2 =-1536445914921655/2251799813685248*x-4312749167979417/4503599627370496-6081006572739691/4503599627370496*(x-1/10)*(x-1/5)+5310037016871761/4503599627370496*(x-1/10)*(x-1/5)*(x-3/10)-3953081808944445/2251799813685248*(x-1/10)*(x-1/5)*(x-3/10)*(x-2/5)

cury2 = -1.5874

k = 6

Newtons form of the interpolation polynomial

p2 =

-145548404491565/562949953421312*x-4771261484776975/2251799813685248*x*(x-1/10)+5769193994690431/2251799813685248*x*(x-1/10)*(x-1/5)-973179839454547/281474976710656*x*(x-1/10)*(x-1/5)*(x-3/10)+3832356906691931/1125899906842624*x*(x-1/10)*(x-1/5)*(x-3/10)*(x-2/5)-1

cury2 = -1.5833

Iteration = 2

curt = 0.7000

EVALUATE NEW y1(n+1)

k = 2

Newtons form of the interpolation polynomial

p1 =4587641153521671/1125899906842624*x+742561540636745/2251799813685248

cury1 = 3.1820

k = 3

Newtons form of the interpolation polynomial

p1

=3979654517613871/1125899906842624*x+3376370441361363/5629499534213120+6079866359078001/2251799813685248*(x-2/5)*(x-1/2)

cury1 = 3.2360

k = 4

Newtons form of the interpolation polynomial

p1

=3456122259882631/1125899906842624*x+8846869913647687/11258999068426240+5235322577312401/2251799813685248*(x-3/10)*(x-2/5)+5630291878437335/4503599627370496*(x-3/10)*(x-2/5)*(x-1/2)

cury1 = 3.2435

k = 5

Newtons form of the interpolation polynomial

p1 =

5999600820045893/2251799813685248*x+20431670926453479/22517998136852480+4563218498596845/2251799813685248*(x-1/5)*(x-3/10)+4480693858103707/4503599627370496*(x-1/5)*(x-3/10)*(x-2/5)+1436997525417035/2251799813685248*(x-1/5)*(x-3/10)*(x-2/5)*(x-1/2)

cury1 = 3.2450

k = 6

Newtons form of the interpolation polynomial

p1 =

2617308358466205/1125899906842624*x+4392327826536089/4503599627370496+7649841031134829/4503599627370496*(x-1/10)*(x-1/5)+7690603989899/70368744177664*(x-1/10)*(x-1/5)*(x-3/10)-2206463477129145/9007199254740992*(x-1/10)*(x-1/5)*(x-3/10)*(x-2/5)+7954453578797285/4503599627370496*(x-1/10)*(x-1/5)*(x-3/10)*(x-2/5)*(x-1/2)

cury1 = 3.2472

EVALUATE NEW y2(n+1)

k = 2

Newtons form of the interpolation polynomial

p2 =-3392537734639281/2251799813685248*x-3077912041268565/4503599627370496

cury2 = -1.7380

k = 3

Newtons form of the interpolation polynomial

p2 =

-5955437037279541/4503599627370496*x-8731828143170189/11258999068426240-

2074096079997553/2251799813685248*(x-2/5)*(x-1/2)

cury2 = -1.7565

k = 4

Newtons form of the interpolation polynomial

p2 =-5186692237926881/4503599627370496*x-38002291770091397/45035996273704960-

3843723996763301/4503599627370496*(x-3/10)*(x-2/5)-4059575509757401/18014398509481984*(x-

3/10)*(x-2/5)*(x-1/2)

cury2 = -1.7578

k = 5

Newtons form of the interpolation polynomial

p2 =

-8578186288782497/9007199254740992*x-40695089050698293/45035996273704960-

8975990935356325/9007199254740992*(x-1/5)*(x-3/10)+2147571569716205/4503599627370496*(x-

1/5)*(x-3/10)*(x-2/5)-988270452236111/562949953421312*(x-1/5)*(x-3/10)*(x-2/5)*(x-1/2)

cury2 = -1.7620

k = 6

Newtons form of the interpolation polynomial

p2 =

-1536445914921655/2251799813685248*x-4312749167979417/4503599627370496-

6081006572739691/4503599627370496*(x-1/10)*(x-1/5)+5310037016871761/4503599627370496*(x-

1/10)*(x-1/5)*(x-3/10)-3953081808944445/2251799813685248*(x-1/10)*(x-1/5)*(x-3/10)*(x-

2/5)+1/1125899906842624*(x-1/10)*(x-1/5)*(x-3/10)*(x-2/5)*(x-1/2)

cury2 = -1.7620

Iteration = 3

curt = 0.8000

EVALUATE NEW y1(n+1)

k = 2

Newtons form of the interpolation polynomial

p1 =

5321189498647431/1125899906842624*x-172120591892709/2814749767106560

cury1 = 3.7198

k = 3

Newtons form of the interpolation polynomial

p1 =

4587641153521671/1125899906842624*x+742561540636745/2251799813685248+3667741725628801/1125899906842624*(x-1/2)*(x-3/5)

cury1 = 3.7849

k = 4

Newtons form of the interpolation polynomial

p1 =

3979654517613871/1125899906842624*x+3376370441361363/5629499534213120+6079866359078001/2251799813685248*(x-2/5)*(x-1/2)+8370780614530675/4503599627370496*(x-2/5)*(x-1/2)*(x-3/5)

cury1 = 3.7961

k = 5

Newtons form of the interpolation polynomial

p1 =

3456122259882631/1125899906842624*x+8846869913647687/11258999068426240+5235322577312401/2251799813685248*(x-3/10)*(x-2/5)+5630291878437335/4503599627370496*(x-3/10)*(x-2/5)*(x-1/2)+856402730029169/562949953421312*(x-3/10)*(x-2/5)*(x-1/2)*(x-3/5)

cury1 = 3.7997

k = 6

Newtons form of the interpolation polynomial

p1 =

5999600820045893/2251799813685248*x+20431670926453479/22517998136852480+4563218498596845/2251799813685248*(x-1/5)*(x-3/10)+4480693858103707/4503599627370496*(x-1/5)*(x-3/10)*(x-

2/5)+1436997525417035/2251799813685248*(x-1/5)*(x-3/10)*(x-2/5)*(x-

1/2)+7954453578798565/4503599627370496*(x-1/5)*(x-3/10)*(x-2/5)*(x-1/2)*(x-3/5)

cury1 = 3.8019

EVALUATE NEW y2(n+1)

k = 2

Newtons form of the interpolation polynomial

p2 =

-1966338865188323/1125899906842624*x-6074360115959317/11258999068426240

cury2 = -1.9367

k = 3

Newtons form of the interpolation polynomial

p2 =

-3392537734639281/2251799813685248*x-3077912041268565/4503599627370496-

5401399957373651/4503599627370496*(x-1/2)*(x-3/5)

cury2 = -1.9607

k = 4

Newtons form of the interpolation polynomial

p2 =-5955437037279541/4503599627370496*x-8731828143170189/11258999068426240-

2074096079997553/2251799813685248*(x-2/5)*(x-1/2)-4177359324595151/4503599627370496*(x-

2/5)*(x-1/2)*(x-3/5)

cury2 = -1.9662

k = 5

Newtons form of the interpolation polynomial

p2 =-5186692237926881/4503599627370496*x-38002291770091397/45035996273704960-

3843723996763301/4503599627370496*(x-3/10)*(x-2/5)-4059575509757401/18014398509481984*(x-

3/10)*(x-2/5)*(x-1/2)-247067613059047/140737488355328*(x-3/10)*(x-2/5)*(x-1/2)*(x-3/5)

cury2 = -1.9704

k = 6

Newtons form of the interpolation polynomial

p2 =

-8578186288782497/9007199254740992*x-40695089050698293/45035996273704960-

8975990935356325/9007199254740992*(x-1/5)*(x-3/10)+2147571569716205/4503599627370496*(x-

1/5)*(x-3/10)*(x-2/5)-988270452236111/562949953421312*(x-1/5)*(x-3/10)*(x-2/5)*(x-1/2)-

5418393301680129/19807040628566084398385987584*(x-1/5)*(x-3/10)*(x-2/5)*(x-1/2)*(x-3/5)

cury2 = -1.9704

Iteration = 4

curt = 0.9000

EVALUATE NEW y1(n+1)

k = 2

Newtons form of the interpolation polynomial

p1 =

6245270244768939/1125899906842624*x-7157047590421393/11258999068426240

cury1 = 4.3565

k = 3

Newtons form of the interpolation polynomial

p1 =

5321189498647431/1125899906842624*x-

172120591892709/2814749767106560+2310201865303769/562949953421312*(x-3/5)*(x-7/10)

cury1 = 4.4386

k = 4

Newtons form of the interpolation polynomial

p1 =

4587641153521671/1125899906842624*x+742561540636745/2251799813685248+3667741725628801/

1125899906842624*(x-1/2)*(x-3/5)+6351080033191579/2251799813685248*(x-1/2)*(x-3/5)*(x-7/10)

cury1 = 4.4555

k = 5

Newtons form of the interpolation polynomial

p1 =

3979654517613871/1125899906842624*x+3376370441361363/5629499534213120+607986635907800

1/2251799813685248*(x-2/5)*(x-1/2)+8370780614530675/4503599627370496*(x-2/5)*(x-1/2)*(x-

3/5)+5414224314815603/2251799813685248*(x-2/5)*(x-1/2)*(x-3/5)*(x-7/10)

cury1 = 4.4613

k = 6

Newtons form of the interpolation polynomial

p1 =

3456122259882631/1125899906842624*x+8846869913647687/11258999068426240+52353225773124

01/2251799813685248*(x-3/10)*(x-2/5)+5630291878437335/4503599627370496*(x-3/10)*(x-2/5)*(x-

1/2)+856402730029169/562949953421312*(x-3/10)*(x-2/5)*(x-1/2)*(x-

3/5)+1988613394698927/1125899906842624*(x-3/10)*(x-2/5)*(x-1/2)*(x-3/5)*(x-7/10)

cury1 = 4.4634

EVALUATE NEW y2(n+1)

k = 2

Newtons form of the interpolation polynomial

p2 =

-2346506234633273/1125899906842624*x-6826377059689333/22517998136852480

cury2 = -2.1789

k = 3

Newtons form of the interpolation polynomial

p2 =

-1966338865188323/1125899906842624*x-6074360115959317/11258999068426240-

7603347388898997/4503599627370496*(x-3/5)*(x-7/10)

cury2 = -2.2126

k = 4

Newtons form of the interpolation polynomial

p2 =

-3392537734639281/2251799813685248*x-3077912041268565/4503599627370496-

5401399957373651/4503599627370496*(x-1/2)*(x-3/5)-458739048234447/281474976710656*(x-

1/2)*(x-3/5)*(x-7/10)

cury2 = -2.2224

k = 5

Newtons form of the interpolation polynomial

p2 =

-5955437037279541/4503599627370496*x-8731828143170189/11258999068426240-2074096079997553/2251799813685248*(x-2/5)*(x-1/2)-4177359324595151/4503599627370496*(x-2/5)*(x-1/2)*(x-3/5)-395308180894500l/2251799813685248*(x-2/5)*(x-1/2)*(x-3/5)*(x-7/10)

cury2 = -2.2266

k = 6

Newtons form of the interpolation polynomial

p2 =

-5186692237926881/4503599627370496*x-38002291770091397/45035996273704960-3843723996763301/4503599627370496*(x-3/10)*(x-2/5)-4059575509757401/18014398509481984*(x-3/10)*(x-2/5)*(x-1/2)-247067613059047/140737488355328*(x-3/10)*(x-2/5)*(x-1/2)*(x-3/5)-249/1125899906842624*(x-3/10)*(x-2/5)*(x-1/2)*(x-3/5)*(x-7/10)

cury2 = -2.2266

Iteration = 5

curt = 1

EVALUATE NEW y1(n+1)

k = 2

Newtons form of the interpolation polynomial

p1 =

7448717444400367/1125899906842624*x-16784625187472811/11258999068426240

cury1 = 5.1250

k = 3

Newtons form of the interpolation polynomial

p1 =

6245270244768939/1125899906842624*x-7157047590421393/11258999068426240+3008617999078569/562949953421312*(x-7/10)*(x-4/5)

cury1 = 5.2319

k = 4

Newtons form of the interpolation polynomial

p1 =5321189498647431/1125899906842624*x-172120591892709/2814749767106560+2310201865303769/562949953421312*(x-3/5)*(x-7/10)+2328053779249333/562949953421312*(x-3/5)*(x-7/10)*(x-4/5)

cury1 = 5.2567

k = 5

Newtons form of the interpolation polynomial

p1 = 4587641153521671/1125899906842624*x+742561540636745/2251799813685248+3667741725628801/1125899906842624*(x-1/2)*(x-3/5)+6351080033191579/2251799813685248*(x-1/2)*(x-3/5)*(x-7/10)+3701418854757191/1125899906842624*(x-1/2)*(x-3/5)*(x-7/10)*(x-4/5)

cury1 = 5.2646

k = 6

Newtons form of the interpolation polynomial

p1 = 3979654517613871/1125899906842624*x+3376370441361363/5629499534213120+6079866359078001/2251799813685248*(x-2/5)*(x-1/2)+8370780614530675/4503599627370496*(x-2/5)*(x-1/2)*(x-3/5)+5414224314815603/2251799813685248*(x-2/5)*(x-1/2)*(x-3/5)*(x-7/10)+1988613394698779/1125899906842624*(x-2/5)*(x-1/2)*(x-3/5)*(x-7/10)*(x-4/5)

cury1 = 5.2667

EVALUATE NEW y2(n+1)

k = 2

Newtons form of the interpolation polynomial

p2 = -1442103978680919/562949953421312*x+1776850503967703/22517998136852480

cury2 = -2.4828

k = 3

Newtons form of the interpolation polynomial

p2 =

-2346506234633273/1125899906842624*x-6826377059689333/22517998136852480-

336063576705353/140737488355328*(x-7/10)*(x-4/5)

cury2 = -2.5305

k = 4

Newtons form of the interpolation polynomial

p2 =

-1966338865188323/1125899906842624*x-6074360115959317/11258999068426240-

7603347388898997/4503599627370496*(x-3/5)*(x-7/10)-5251145109453831/2251799813685248*(x-

3/5)*(x-7/10)*(x-4/5)

cury2 = -2.5445

k = 5

Newtons form of the interpolation polynomial

p2 =

-3392537734639281/2251799813685248*x-3077912041268565/4503599627370496-

5401399957373651/4503599627370496*(x-1/2)*(x-3/5)-458739048234447/281474976710656*(x-

1/2)*(x-3/5)*(x-7/10)-7906163617891275/4503599627370496*(x-1/2)*(x-3/5)*(x-7/10)*(x-4/5)

cury2 = -2.5487

k = 6

Newtons form of the interpolation polynomial

p2 =

-5955437037279541/4503599627370496*x-8731828143170189/11258999068426240-

2074096079997553/2251799813685248*(x-2/5)*(x-1/2)-4177359324595151/4503599627370496*(x-

2/5)*(x-1/2)*(x-3/5)-3953081808945001/2251799813685248*(x-2/5)*(x-1/2)*(x-3/5)*(x-7/10)-

1273/2251799813685248*(x-2/5)*(x-1/2)*(x-3/5)*(x-7/10)*(x-4/5)

cury2 = -2.5487

FOURTH DIVISION : NUMERICAL RESULTS WITH EXACT SOLUTION

>> table

table =

0	**1.0000**	**1.0000**	**-1.0000**	**-1.0000**
0.1000	**1.2157**	**1.2078**	**-1.1052**	**-1.0259**

0.2000 1.4657 1.4402 -1.2214 -1.0941

0.3000 1.7548 1.7067 -1.3499 -1.1893

0.4000 2.0886 2.0136 -1.4918 -1.3045

0.5000 2.4731 2.3671 -1.6487 -1.4367

0.6000 2.9154 2.7745 -1.8221 -1.5874

0.7000 3.4234 3.2472 -2.0138 -1.7620

0.8000 4.0060 3.8019 -2.2255 -1.9704

0.9000 4.6732 4.4634 -2.4596 -2.2266

1.0000 5.4366 5.2667 -2.7183 -2.5487

>>

Execution for example 5

Using Power series

tabcoef =

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

v1 =1+e1*x^2

v2 =-1-x+e2*x^2

power = 2

f =

[1+e1*x^2-x*(-1-x+e2*x^2)-exp(x)]

[2*e1*x-x*(-1+2*e2*x)]

ans =

1+e1*x^2+x+x^2-e2*x^3-exp(x)

ans =

2*e1*x+x-2*e2*x^2

ans =

here are analytic manipulations for determinig vector e by

ans =

consisting linear system and solve it or input e directly from key board

ans =

power = 2

Add new vector e :

Choice of addition method:

1. From keyboard

2. Input system and solve it :

Choose 1 or 2 please

1

coeffecient :-1

newcoef =

-1

coeffecient :-0.5

newcoef =

-1.0000 -0.5000

esv1 =1-x^2

esv2 =-1-x-1/2*x^2

f =

[1-x^2+e1*x^3-x*(-1-x-1/2*x^2+e2*x^3)-exp(x)]

[-2*x+3*e1*x^2-x*(-1-x+3*e2*x^2)]

ans =

1+e1*x^3+x+1/2*x^3-e2*x^4-exp(x)

ans =

-x+3*e1*x^2+x^2-3*e2*x^3

ans =

here are analytic manipulations for determinig vector e by

ans =

consisting linear system and solve it or input e directly from key board

ans =

power =

3

Add new vector e :

Choice of addition method:

1. From keyboard

2. Input system and solve it :

Choose 1 or 2 please

1

coeffecient :0.5

newcoef =

0.5000 -0.5000

coeffecient :1/6

newcoef =

0.5000 0.1667

esv1 =

1-x^2+1/2*x^3

esv2 =

-1-x-1/2*x^2+1/6*x^3

f =

[1-x^2+1/2*x^3+e1*x^4-x*(-1-x-1/2*x^2+1/6*x^3+e2*x^4)-exp(x)]

[-2*x+3/2*x^2+4*e1*x^3-x*(-1-x+1/2*x^2+4*e2*x^3)]

ans =

1+x^3+e1*x^4+x-1/6*x^4-e2*x^5-exp(x)

ans =

-x+5/2*x^2+4*e1*x^3-1/2*x^3-4*e2*x^4

ans =

here are analytic manipulations for determinig vector e by

ans =

consisting linear system and solve it or input e directly from key board

ans =

power =

4

Add new vector e :

Choice of addition method:

1. From keyboard

2. Input system and solve it :

Choose 1 or 2 please

1

coeffecient :-1/6

newcoef =

-0.1667 0.1667

coeffecient :-1/24

newcoef =

-0.1667 -0.0417

esv1 =

1-x^2+1/2*x^3-1/6*x^4

esv2 =

-1-x-1/2*x^2+1/6*x^3-1/24*x^4

f =

[1-x^2+1/2*x^3-1/6*x^4+e1*x^5-x*(-1-x-1/2*x^2+1/6*x^3-1/24*x^4+e2*x^5)-exp(x)]

[-2*x+3/2*x^2-2/3*x^3+5*e1*x^4-x*(-1-x+1/2*x^2-1/6*x^3+5*e2*x^4)]

ans =

1+x^3-1/3*x^4+e1*x^5+x+1/24*x^5-e2*x^6-exp(x)

ans =

-x+5/2*x^2-7/6*x^3+5*e1*x^4+1/6*x^4-5*e2*x^5

ans =

here are analytic manipulations for determinig vector e by

ans =

consisting linear system and solve it or input e directly from key board

ans =

power =

5

Add new vector e :

Choice of addition method:

1. From keyboard

2. Input system and solve it :

Choose 1 or 2 please

1

coeffecient :-1/24

newcoef =

-0.0417 -0.0417

coeffecient :-1/120

newcoef =

-0.0417 -0.0083

esv1 =

1-x^2+1/2*x^3-1/6*x^4-1/24*x^5

esv2 =

-1-x-1/2*x^2+1/6*x^3-1/24*x^4-1/120*x^5

f =

[1-x^2+1/2*x^3-1/6*x^4-1/24*x^5+e1*x^6-x*(-1-x-1/2*x^2+1/6*x^3-1/24*x^4-1/120*x^5+e2*x^6)-exp(x)]

[-2*x+3/2*x^2-2/3*x^3-5/24*x^4+6*e1*x^5-x*(-1-x+1/2*x^2-1/6*x^3-1/24*x^4+6*e2*x^5)]

ans =

1+x^3-1/3*x^4+e1*x^6+x+1/120*x^6-e2*x^7-exp(x)

ans =

-x+5/2*x^2-7/6*x^3-1/24*x^4+6*e1*x^5+1/24*x^5-6*e2*x^6

ans =

here are analytic manipulations for determinig vector e by

ans =

consisting linear system and solve it or input e directly from key board

ans =

power =

6

Add new vector e :

Choice of addition method:

1. From keyboard

2. Input system and solve it :

Choose 1 or 2 please

1

coeffecient :1/120

newcoef =

0.0083 -0.0083

coeffecient :-1/720

newcoef =

0.0083 -0.0014

esv1 =

1-x^2+1/2*x^3-1/6*x^4-1/24*x^5+1/120*x^6

esv2 =

-1-x-1/2*x^2+1/6*x^3-1/24*x^4-1/120*x^5-1/720*x^6

tabcoef =

Columns 1 through 9

1.0000 0 -1.0000 0.5000 -0.1667 -0.0417 0.0083 0 0

-1.0000 -1.0000 -0.5000 0.1667 -0.0417 -0.0083 -0.0014 0 0

Column 10

0

0

This is Pade Approximation.

m+n must equal 6

Input m and n on separate lines.

order of numerator 3

order of denominator 3

AA =

1 0 0 0 0 0 0

AA =

1 0 0 0 0 0 0

AA =

1 0 -1 0 0 0 0

AA =

1.0000 0 -1.0000 0.5000 0 0 0

AA =

1.0000 0 -1.0000 0.5000 -0.1667 0 0

AA =

1.0000 0 -1.0000 0.5000 -0.1667 -0.0417 0

AA =

1.0000 0 -1.0000 0.5000 -0.1667 -0.0417 0.0083

PADE RATIONAL APPROXIMATION

Denominator Coefficients Q(0), ..., Q(M)

1.00000000 -0.31666667 -0.32500000 -0.15138889

Numerator Coefficients P(0), ..., P(N)

1.00000000 -0.31666667 -1.32500000 0.66527778

ans =

102 111 114 109 97 116

p1 =

1-19/60*x-53/40*x^2+479/720*x^3

q1 =

1-19/60*x-13/40*x^2

This is Pade Approximation.

m+n must equal 6

Input m and n on separate lines.

order of numerator 3

order of denominator 2

AA =

-1 0 0 0 0 0

AA =

-1 -1 0 0 0 0

AA =

-1.0000 -1.0000 -0.5000 0 0 0

AA = -1.0000 -1.0000 -0.5000 0.1667 0 0

AA = -1.0000 -1.0000 -0.5000 0.1667 -0.0417 0

AA = -1.0000 -1.0000 -0.5000 0.1667 -0.0417 -0.0083

PADE RATIONAL APPROXIMATION

Denominator Coefficients Q(0), ..., Q(M)

1.00000000 1.60000000 0.45000000

Numerator Coefficients P(0), ..., P(N)

-1.00000000 -2.60000000 -2.55000000 -1.08333333

ans = 102 111 114 109 97 116

p2 =-1-13/5*x-51/20*x^2-13/12*x^3

q2 =1+8/5*x

x exact1 pade1 exact2 pade2

tabl =

0 1.0000 1.0000 -1.0000 -1.0000

0.1000 0.9947 0.9903 -1.1052 -1.1091

0.2000 0.9771 0.9625 -1.2214 -1.2354

0.3000 0.9449 0.9177 -1.3499 -1.3775

0.4000 0.8951 0.8570 -1.4918 -1.5350

0.5000 0.8244 0.7806 -1.6487 -1.7072

0.6000 0.7288 0.6879 -1.8221 -1.8939

0.7000 0.6041 0.5771 -2.0138 -2.0949

0.8000 0.4451 0.4442 -2.2255 -2.3099

0.9000 0.2460 0.2805 -2.4596 -2.5390

1.0000 0 0.0659 -2.7183 -2.7821

جامعة النجاح الوطنية
كلية الدراسات العليا

طرق عددية لحل معادلات تفاضلية جبرية

إعداد
سامر أمين كامل أبو صاع

إشراف
د. سمير مطر

قدمت هذه الأطروحة استكمالاً لمتطلبات درجة الماجستير في الرياضيات المحوسبة بكلية الدراسات العليا في جامعة النجاح الوطنية في نابلس،فلسطين.
2010

طرق عدية لحل المعادلات الجبرية التفاضلية

إعداد

سامر أمين كامل ابو صاع

إشراف

د. سمير مطر

الملخص

تتضمن هذه الدراسة تطبيقاً لطريقتين عدديتين لحل المعادلات الجبرية التفاضلية من النوع الخطي و هما طريقة متسلسلة القوة و طريقة الفروق العكسية . و تهدف هذه الدراسة على تبسيط التعامل مع مثل هذه الأنظمة و ذلك بتحليل الطريقتين و تصميم برامج باستخدام لغة الحاسوب MATLAB. ثم تطبيق هاتين الطريقتين على مسائل مختلفة و من ثم عقد مقارنة بينهما.

وقد خلصت الدراسة إلى أنّ طريقة متسلسلة القوة يمكنها تجنب كثير من الصعوبات التي تواجه الطرق الأخرى و باستطاعتها حل مسائل صعبة ليس بمقدور الطرق الأخرى حلها.