

THE GLOBAL ERROR OF SINGLE-STEP
METHODS FOR INITIAL VALUE
PROBLEMS

By

ZU-LEI LI

Bachelor of Science

Shanghai Teacher University

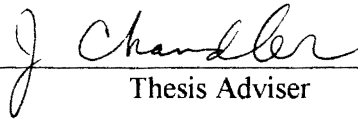
1982

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 1993

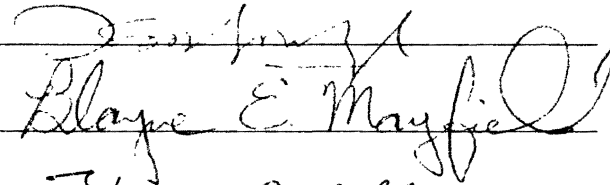
OKLAHOMA STATE UNIVERSITY

THE GLOBAL ERROR OF SINGLE-STEP
METHODS FOR INITIAL VALUE
PROBLEMS

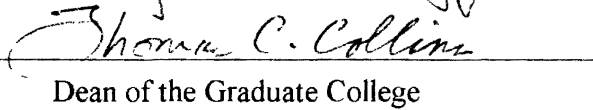
Thesis Approved:



Thesis Adviser



Blayne E. Mayfield



Dean of the Graduate College

PREFACE

The main objective of this study is to find the relationship between a polynomial initial value problem and the global error of single-step methods. Using multi-step methods we can solve the polynomial problem exactly while using single-step methods the global errors are constant after some iterations. We know both multi-step methods and single-step methods are based on Taylor Series and Taylor Series are polynomials. So we might think that we can solve the polynomial problems exactly just like the multi-step methods. Investigation this interesting problem I tested many different cases and used error estimate methods to verify the results obtained from the testing.

I would like to express my gratitude to my major advisor Dr. John Chandler for his motivation, guidance, and insightful instructions and suggestions. Appreciation is also expressed to the other committee members, Dr. K. M. George and Dr. Blayne E. Mayfield, for their cooperation and assistance.

Finally, special thanks are expressed to my parents and to my wife for their continued emotional support, moral encouragement and understanding throughout this study.

TABLE OF CONTENTS

| Chapter | Page |
|---|------|
| I. INTRODUCTION | 1 |
| II. BACKGROUND AND LITERATURE REVIEW | 5 |
| Taylor Series Methods | 5 |
| Euler Method | 6 |
| Runge-Kutta Methods | 8 |
| Some Formulas of Multi-step Methods | 18 |
| III. PROGRAM RESULTS | 20 |
| Stability | 20 |
| About the Program of This Paper | 23 |
| Test Problems Design and the Results | 23 |
| Multi-step Methods | 42 |
| Summary of the Program Results | 42 |
| IV. ANALYSES AND DISCUSSION | 44 |
| Roundoff Error and Truncation Error | 44 |
| Analytic Truncation Error | 45 |
| "Parallel" Parabolas Property | 59 |
| V. CONCLUSIONS AND SUGGESTIONS | 63 |
| SELECTED REFERENCES | 65 |
| APPENDIX - PROGRAMS FOR SINGLE-STEP METHODS AND MULTI-STEP METHODS | 69 |

LIST OF TABLES

| Table | Page |
|---|------|
| I. Coefficients For RK3 Error | 15 |
| II. Absolute Stability of RK Method | 22 |
| III. Error Versus Step size I..... | 24 |
| IV. Error Versus Step size II | 25 |
| V. Error Versus Step size III | 27 |
| VI. Error Versus Step size IV..... | 28 |
| VII. Error Versus Coefficient I | 33 |
| VIII. Error Versus Coefficient II..... | 35 |
| IX. Comparing I (Runge-Kutta 2)..... | 38 |
| X. Comparing I (Runge-Kutta 3)..... | 38 |
| XI. Comparing I (Runge-Kutta 4)..... | 39 |
| XII. Comparing II (Runge-Kutta 2)..... | 40 |
| XIII. Comparing II (Runge-Kutta 3)..... | 40 |
| XIV. Comparing II (Runge-Kutta 4)..... | 41 |

LIST OF FIGURES

| Figure | Page |
|--|------|
| 1. Euler Method Graphically Displayed | 7 |
| 2. Runge-Kutta Methods Graphically Displayed | 10 |
| 3. Absolute Stability Region In The Complex μ -plane | 22 |
| 4. Error Versus Stepsize I | 30 |
| 5. Error Versus Stepsize II | 30 |
| 6. Error Versus Stepsize III | 31 |
| 7. Error Versus Stepsize IV | 31 |
| 8. Error Versus Stepsize V | 32 |
| 9. Error Versus Stepsize VI | 32 |
| 10. Error Versus Coefficient I | 36 |
| 11. Error Versus Coefficient II | 37 |
| 12. Error Versus Coefficient III | 37 |
| 13. Different Methods Compared I | 41 |
| 14. Different Methods Compared II | 42 |
| 15. The Minimum Error as a Function of h | 45 |
| 16. "Parallel" Parabolas | 60 |

CHAPTER I

INTRODUCTION

Many important and significant problems in engineering, in the physical sciences, and the social sciences, when formulated in mathematical terms, require the determination of a function satisfying a differential equation or functions satisfying systems of differential equations containing one or more derivatives of the unknown function or functions. For instance, problems in mechanics such as the motion of projectiles or orbiting bodies, in population dynamics and in chemical kinetics may be modeled by ordinary differential equations. Many clever methods of finding analytical solutions of ordinary differential equations have been devised by mathematicians, but the majority of differential equations are not amenable to these methods, and unfortunately most of the differential equations which model practical problems fall into this category. We can solve linear constant coefficient ordinary equations by analytic methods, but using analytic methods to solve linear variable coefficient ordinary equations is very difficult, and it is even more difficult to solve nonlinear differential equations. For example,

$$\begin{aligned}\frac{dy}{dx} &= x^2 + 7y^{\ln x} - \frac{y}{x} - y^2 \\ y(1) &= -1\end{aligned}$$

In many circumstances we can only use numerical approximation to solve the problems of ordinary differential equations. There are two basic approaches to the numerical approximation of solutions of differential equations. One is approximate

analytic methods, for example series solution, and the other is discretization. In this thesis we shall deal with the latter method. The feature of this method is that no attempt is made to approximate the exact solution $y(x)$ over a continuous range of the independent variable. Approximate values are sought only on a set of discrete points $x_1, x_2, x_3, \dots, x_n$. Frequently, though not always, the points x_n are equidistant; if they are, we write $x_n = a + nh$, $n = 0, 1, 2, 3, \dots, n$, where h is a stepsize. Generally speaking, a discrete variable method for solving a differential equation consists of an algorithm which, corresponding to each lattice point x_n , furnishes a number y_n which is to be regarded as an approximation to the value $y(x_n)$ of the exact solution $y(x)$ at the point x_n . One of the main questions we shall be concerned with is the size of the discretization error $e_n = y_n - y(x_n)$, in particular as a function of the step h . In the last forty or so years, mathematicians have put the subject on a much sounder theoretical basis, particularly in areas such as the propagation of errors and stability. Much work has been done on the implementation of methods and on their comparative testing. This has produced some agreement on what is the "best" method for a given non-stiff ordinary differential equation. Recently the problem of stiffness has received a lot of attention. [27, pp. 7-8]

Although boundary value problems for ordinary differential equations are very important, only initial value problems have been considered here.

Contrary to some of the other methods, discrete variable methods enjoy the advantage of being almost universally applicable. In the case of the initial value problem, for example, the only requirement for the applicability of most discrete variable methods is the ability to calculate a good approximation to the value of $f(x,y)$ for a given x and y . It is true that the function f may have to be evaluated a large number of times in order to keep the discretization error sufficiently small. "Once a deterrent to the application of discrete variable methods, this fact causes no concern today, when large numerical calculations, especially if of a repetitive nature, can be performed efficiently and reliably on automatic digital computers." [27]

Among the discrete variable methods for the solution of initial value problems we can distinguish between one-step methods and multi-step methods. In a one-step method the value of y_{n+1} can be found if only y_n is known; no knowledge of any preceding values y_{n-1}, y_{n-2}, \dots is required. In a multi-step method, the calculation of y_{n+1} requires the explicit knowledge not only of y_n but also of a certain number of preceding values y_{n-1}, y_{n-2}, \dots . A method is called a k-step method if the values $y_n, y_{n-1}, y_{n-2}, \dots, y_{n-k}$ and $f_n, f_{n-1}, \dots, f_{n-k}$ are required for the calculation of y_{n+1} . [22, p. 4]

In one-step methods the starting point does not play a special role. In fact, every lattice point can be considered a new starting point. This makes it easy to change the stepsize h . "Stability" considerations also seem to favor the single-step methods. [16] Multi-step methods require a special starting procedure, since at the points $x_0, x_1, x_2, \dots, x_{k-1}$ some of the values y_{n-j} and f_{n-j} required for the computation are missing, and a special computation may also be necessary at points where the stepsize is changed. All this tends to increase the length and complexity of the required machine codes. But generally speaking, multi-step methods may be more accurate than one-step methods for a given time of computation.

Barbara Tracy, a former student of Dr. Chandler, found an interesting phenomenon that when solving the initial value problem

$$y' = 2x - 1000(y - x^2), \quad y(0) = 0$$

by using the Runge-Kutta method of order 4 with stepsize $h = 0.001$, after some iterations the global error is approximately a constant. The exact solution of this problem is $y(x) = x^2$, and Adams methods of order two or greater give the exact solution. Why does not a Runge-Kutta method of order 4 solve this polynomial problem exactly and its global error is a constant after some iterations? Like Adams methods, Runge-Kutta methods are based on a Taylor series, and Taylor series are polynomials. More general, for the initial value problem

$$y' = nx^{n-1} + c(y - x^n), \quad y(0) = 0 \quad (1)$$

where n is a positive integer, c is a negative real number, is the global error zero? If not, is it a constant after some iterations? The exact solution of the problem is the polynomial $y(x) = x^n$. We also want to know what will be happen if the initial value $y(0)=y_0$ where y_0 is not zero.

It is the purpose of this thesis to study these problems and we only use the Runge Kutta methods to do some research. Brief introductions of Runge-Kutta methods are presented in Chapter II. Meanwhile the different strategies for estimating the local or global error and the most useful error estimates are also described in Chapter II.

In Chapter III, according to the analytic theory we can calculate the domain of the convergence and the stability of the initial value problem (1). Within the domain I try different variable values of exponent n , constant coefficient c and stepsize h , so that we can get some interesting data. Using these data I plot the principle features of the graphs of $\log(\text{error})$ versus $\log(\text{stepsize } h)$ and global error versus constant coefficient c .

In Chapter IV we try to explain these phenomena using the posterior error estimate method.

The final conclusions of this thesis and suggestions for further study are given in Chapter V. All program listings are collected in the Appendices.

CHAPTER II

BACKGROUND AND LITERATURE REVIEW

Taylor Series Methods

The solution of the initial value problem

$$\begin{aligned}y' &= f(x, y) \\ y(x_0) &= y_0\end{aligned}\quad (2)$$

may be expressed as the Taylor series [22, p. 108]

$$\begin{aligned}y(x) = & y(x_0) + \frac{dy(x_0)}{dx}(x - x_0) + \frac{1}{2} \frac{d^2y(x_0)}{dx^2}(x - x_0)^2 + \frac{1}{3!} \frac{d^3y(x_0)}{dx^3}(x - x_0)^3 + \dots \\ & + \frac{1}{n!} \frac{d^ny(x_0)}{dx^n}(x - x_0)^n + \dots\end{aligned}\quad (3)$$

provided it is infinitely differentiable at $x = x_0$. The second and higher derivatives in (3) may be obtained (if they exist) by repeatedly differentiating the differential equation using the chain rule. Thus

$$\begin{aligned}y(x_0) &= y_0, \\ \frac{dy(x_0)}{dx} &= f(x_0, y_0), \\ \frac{d^2y(x_0)}{dx^2} &= \frac{d}{dx} f(x_0, y_0) = \frac{\partial f(x_0, y_0)}{\partial x} + \frac{\partial f(x_0, y_0)}{\partial y} \frac{dy(x_0)}{dx}, \\ \frac{d^3y(x_0)}{dx^3} &= \frac{d^2}{dx^2} f(x_0, y_0) = \dots, \\ &\dots \\ &etc.\end{aligned}$$

In practice it is usually computationally more efficient to calculate the derivatives recursively. This is the principal disadvantage of this method--to obtain accurate values of the solution well away from $x = x_0$ requires that a large number of terms of the Taylor series be used, which in turn requires that the solution must be many times differentiable at $x = x_0$, and that it is convergent for the particular value of x . The labor involved in calculating the derivatives may become prohibitive for more complicated problems unless computer software for symbolic manipulation including differentiation is available. However, these problems can be overcome to some extent by using the method in a stepwise manner. No matter how few terms of the Taylor series are used (at least two always exist), an estimate of $y(x_0 + h)$ can be obtained to any given accuracy simply by using a small enough h and a truncated Taylor series about $x_0 + h$ calculated and used to advance the solution further. By repeating this procedure, the solution can be approximated for any value of x . But as you can see later, if the stepsize h is too small the roundoff error will increase greatly. The Taylor series method has been developed most thoroughly by Chang and co-workers [11].

Euler Method

The Euler method, also called the method of the tangent, is the first method of approximate integration (1786).

The Formula of the Euler Method

Let the problem be (2)

We set

$$y_{n+1} = y_n + hf(x_n, y_n) \quad (4)$$

supposing, to shorten the notation, that $h = x_n - x_{n-1}$ is a constant. The Euler method is illustrated graphically in Figure 1.

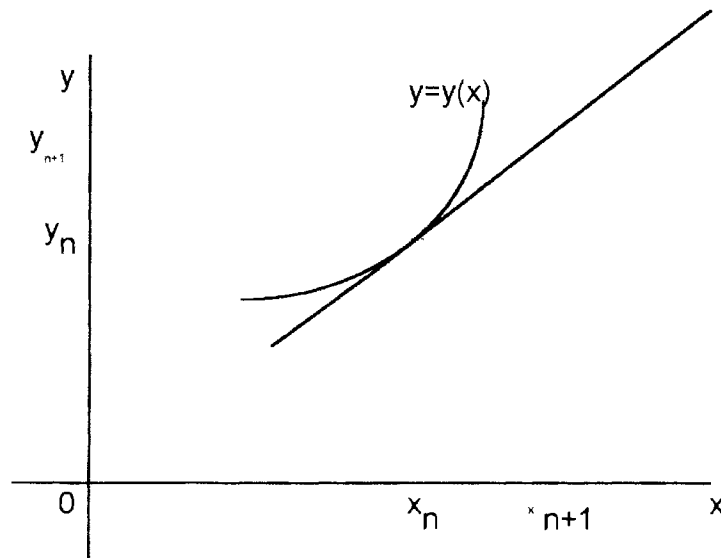


Figure 1 . Euler Method Graphically Displayed

Note that it uses the first two terms of the Taylor series to approximate the solution, so it can be treated as a special case of the Taylor series method.

Equation (4) simply replaces the function $y(x)$ at the point x_n by a straight line with the same slope as the tangent line at that point. If this approximation is not particularly good (for example, if $h = x_n - x_{n-1}$ is not small enough or if $y'(x)$ is rapidly changing), then the calculated value for y_{n+1} will not be very accurate. The subsequent point, y_{n+2} , which depends upon the values calculated for y_{n+1} , will be even worse. The error will accumulate for subsequent points. "Despite the obvious inadequacies of the Euler method, its compelling simplicity makes it useful as a starting point for the development of more sophisticated procedures for solving differential equations. Even though the Euler method is rarely used in a serious solution of a problem it is easy to analyze and is often the basis of constructive existence proofs." [7]

Local and Global Truncation Errors

"If an error estimate method depends only on a knowledge of the equation but not on the knowledge of the solution $y(x)$, it is called an a priori estimate method. If, on the other hand, a knowledge of the properties of the solution $y(x)$ is required, its error estimate is called a posterior estimate method." [16] Following are some error estimates for the Euler's method:

$$\text{a. } |e_n| \leq h \frac{K + LZ}{L} (e^{Lb} - 1) + e^{Lb} |e_0| \quad (5)$$

where h is the stepsize of the method, K is the Lipschitz constant for f as a function of x , L is the Lipschitz constant of f as a function of y , Z is the maximum of the value of $y'(x)$ in the domain of (a, b) . Formula (5) derived the error bound for the solution of (4). If we have some knowledge of the solution, and assume that its second derivative is continuous and bounded by a known quantity, say C , we can get a better bound. It is

$$|e_n| \leq \frac{hC}{2L} (e^{Lb} - 1) + e^{Lb} |e_0| \quad (6)$$

$$\text{b. } e_n = h\delta(x_n) + O(h^2) \quad (7)$$

where $\delta(x)$ is the solution of the ordinary differential equation

$$\begin{aligned} \frac{d\delta}{dx} &= g(x)\delta - \frac{1}{2} \frac{d^2y}{dx^2} \\ \delta(0) &= \frac{e_0}{h} \end{aligned} \quad (8)$$

and $g(x) = \frac{\partial f}{\partial y}$ if y is continuously differentiable three times. Theoretical details and justifications about these formulas may be found in Gear [14, pp. 13-15].

Runge-Kutta Methods

As mentioned previously, the Euler method is not very useful in practical problems

because it requires a very small stepsize for reasonable accuracy. Taylor's algorithm of higher order is inconvenient as a general-purpose procedure because of the need to obtain higher total derivatives of $y(x)$. The Runge-Kutta methods attempt to obtain greater accuracy by developing formulas equivalent to second, third, fourth, fifth, or even higher order Taylor series formulas, and at the same time avoid the need for higher derivatives, by obtaining an approximate numerical tabulation of the dependent variables at specified intervals of the independent variable.

Carl David Runge (1856-1927), German mathematician and physicist, worked for many years in spectroscopy. The analysis of data led him to consider problems in numerical computation, and the Runge-Kutta method originated in his paper on the numerical solution of differential equations in 1895. The method was extended to systems of equations in 1901 by M. Wilhelm Kutta (1867-1944). Kutta was a German mathematician and aerodynamicist who is also well-known for his important contributions to classical airfoil theory. [27, p. 12]

The elementary refinements of the basic Euler method, which is a first order Runge-Kutta method, were based on the idea that using the average slope of the tangent over an interval to extrapolate a function to the next point should result in greater accuracy. The other Runge-Kutta methods whose order are larger than one carries this a bit further. The slope that is used in the linear extrapolation is taken to be a weighted average of the slope at the left end of the interval and some intermediate point. Thus, the algorithm reads

$$y_{n+1} = y_n + f_{ave}h \quad (9)$$

where f_{ave} is determined by the equation

$$f_{ave} = af_n + bf_{n'} \quad (10)$$

where a, b are the "weights" and f_n is the function evaluated at the point x_n , and $f_{n'}$ is the function evaluated at some intermediate point defined as

$$f_{n'} = f(x_n + \alpha h, y_n + \beta f_n h) \quad (11)$$

where the two parameters α, β specify the position of the intermediate point. We can use the graph to explain these (Fig. 2).

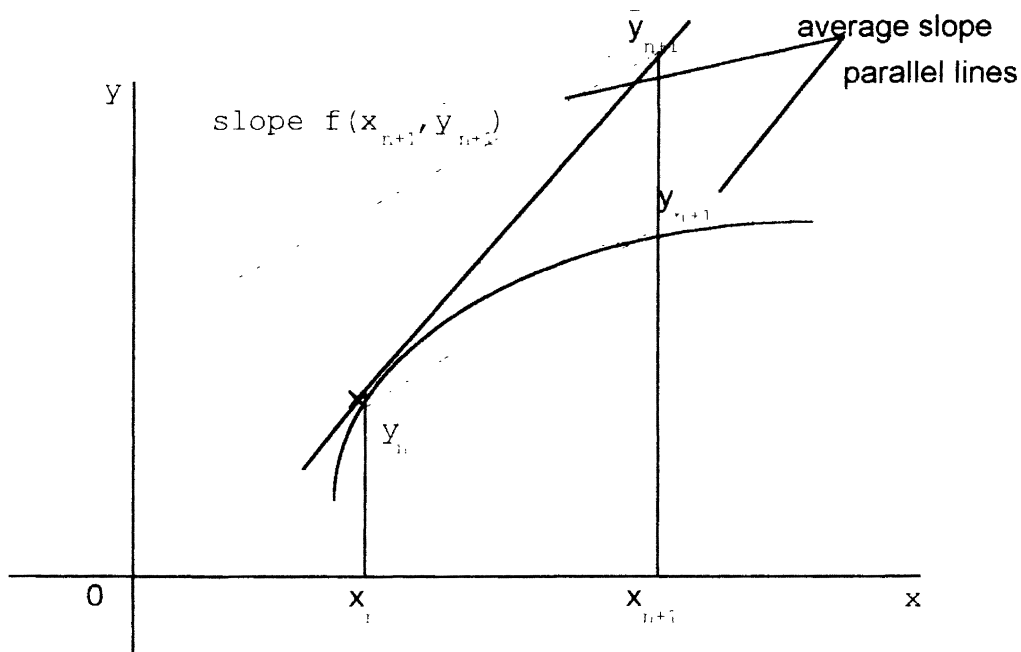


Figure 2 . Runge-Kutta Methods Graphically Displayed

General Runge-Kutta Formula

$$y_{n+1} = y_n + a_1 k_1 + a_2 k_2 + a_3 k_3 + \dots + a_n k_n$$

$$k_i = hf(x_n + \alpha_i h, y_n + \sum_{s=1}^i b_{is} k_s) \quad (i = 1, 2, 3, \dots, n) \quad (12)$$

where a_i, α_i, b_{is} ($i = 1, 2, 3, \dots, n$; $s = 1, 2, \dots, i$) are constant coefficients. For simplicity in

this thesis we only consider $i=1,2,3,4$.

Second Order Methods

$$\begin{aligned}
 y_{n+1} &= y_n + ak_1 + bk_2 \\
 k_1 &= hf(x_n, y_n) \\
 k_2 &= hf(x_n + hm, y_n + mk_1)
 \end{aligned} \tag{13}$$

here:

$$a = \frac{2m-1}{2m}$$

$$b = \frac{1}{2m}$$

This is one family of second order methods, where m is the independent parameter. If $m=1/2$ it is the so-called midpoint method; $m=1$ gives the well-known modified Euler method, and it is Heun's method of order two if $m=2/3$. [9, pp. 53-54]

Third Order Methods

$$\begin{aligned}
 y_{n+1} &= y_n + ak_1 + bk_2 + ck_3 \\
 k_1 &= hf(x_n, y_n) \\
 k_2 &= hf(x_n + mh, y_n + mk_1) \\
 k_3 &= hf(x_n + nh, y_n + (n-r)k_1 + rk_2)
 \end{aligned} \tag{14}$$

here:

$$a + b + c = 1$$

$$bm + cn = \frac{1}{2}$$

$$bm^2 + cn^2 = \frac{1}{3}$$

$$crm = \frac{1}{6}$$

Since there are four equations and six unknowns, there are two degrees of freedom in the system. Choosing different pairs of the parameters we can get various particular formulas.

Here are some special cases:

Classical Method . In formula (14), $m = 0.5, n = 1.$

Optimal Method. [27, p. 14] In formula (14), $m = \frac{1-2\sqrt{13}}{6}, n = \frac{1+\sqrt{13}}{6}.$

Heun Method of Order Three . In formula (14), $m = 2/3, n = 2/3, a = 0.25,$
 $b = 0, c = 3/4.$

Conte and Reeves method. [27, p.14] In formula (14), $m = 0.6265833,$
 $n = 0.0754259.$

Fourth Order Methods

$$y_{n-1} = y_n + ak_1 + bk_2 + ck_3 + dk_4$$

$$k_1 = hf(x_n, y_n)$$

$$k_2 = hf(x_n + mh, y_n + mk_1) \quad (15)$$

$$k_3 = hf(x_n + nh, y_n + (n-r)k_1 + rk_2)$$

$$k_4 = hf(x_n + ph, y_n + (p-s-t)k_1 + sk_2 + tk_3)$$

here:

$$a + b + c + d = 1$$

$$bm + cn + dp = \frac{1}{2}$$

$$bm^2 + cn^2 + dp^2 = \frac{1}{3}$$

$$crm + d(sm + tn) = \frac{1}{6}$$

$$bm^3 + cn^3 + dp^3 = \frac{1}{4}$$

$$crmn + dp(sm + tn) = \frac{1}{8}$$

$$crm^2 + d(sm^2 + tn^2) = \frac{1}{12}$$

$$dtrm = \frac{1}{24}$$

Since there are eight equations and ten unknowns, the equations above possess a two-parameter of solutions. The special cases are:

Merson's Method. [29] In formula (15), $m = 1/3$, $n = 0.5$.

Hull and Johnson Method. [25] In formula (15), $m = 0.35$, $n = 0.45$.

Boulton Method. [27, p. 16] In formula (15), $m = 1/3$, $n = 2m$.

Optimal Method. [27, p.16] In formula (15), $m = 2/5$, $n = 3/5$.

Classical method . In formula (15), $m = 0.5$, $n = 0.5$, $a = 1/6$, $b = 1/3$, $c = 1/3$,

$$d = 1/6, p = 1, r = 1/2, s = 0, t = 1.$$

Local And Global Truncation Errors

The Runge-Kutta methods may safely be applied to almost any problems of differential equations with initial value conditions, but present nevertheless certain inconveniences. One of these is the difficulty of obtaining any estimate of the truncation error involved. A brief summary of possible methods of error estimate is given by Scraton (1964). The procedures available for estimating truncation errors may be classified into two types: multi-stage methods, where the estimate is obtained only after the tabulation has proceeded for several steps; and single-stage methods, where each step provides its

own immediate error estimate. In general, it is possible to devise a multi-stage for any integration process, but a single-stage error estimate can only be obtained when the integration process is specifically designed to provide this feature.

Practical Evaluation of the Truncation Error of Rank Two

Ceschino and Kuntzmann [7, pp. 54-56] give the general multi-stage error estimate for the Runge-Kutta method of order two:

$$E = \frac{h}{12} \left[-5f_{n+1} + f_{n-1} + \frac{6}{m} f_{n,1} + \left(-\frac{6}{m} + 4 \right) f_n \right]$$

where

$$f_{n+1} = f(x_n + h, y_{n+1}) \quad y_{n+1} \text{ can be got formula (13)}$$

$$f_{n-1} = f(x_n - h, y_{n-1}) \quad y_{n-1} \text{ can be got formula (13) and}$$

only use -h to submit the h in the formula.

$$f_{n,1} = f(x_n + mh, y_n + mk_1) \quad k_1 \text{ and m is the same as (13)}$$

$$f_n = f(x_n, y_n) \quad h \text{ is stepsize.}$$

If m=1 it is Euler-Cauchy and

$$E = \frac{h}{12} [-5f_{n+1} + 6f_{n,1} - 2f_n + f_{n-1}]$$

$$\text{here } f_{n,1} = f(x_n + h, y_n + k_1)$$

If m = $\frac{1}{2}$ it is the improved Euler method and

$$E = \frac{h}{12} [-5f_{n+1} + 12f_{n,1} - 8f_n + f_{n-1}]$$

$$\text{where } f_{n,1} = f\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right)$$

Practical Evaluation of the Truncation Error of Rank Three

For the formula (14) the principal part of the error at one step is:

$$h^4 \left[c_3 \left(-\frac{1}{24} - \frac{mn}{12} + \frac{m+n}{18} \right) + j_1 c_2 \left(-\frac{1}{24} - \frac{m}{12} \right) \right. \\ \left. + k_1 c_1 \left(-\frac{1}{8} - \frac{n}{6} \right) - j_1^2 c_1 \frac{1}{24} + j_2 c_1 \left(-\frac{1}{24} - \frac{c}{2} rm \right) \right]$$

Here are the numerical values of the coefficients for the methods above:

TABLE I
COEFFICIENTS FOR RK3 ERROR

| | m | n | C_3 | $j_1 C_2$ | $k_1 C_1$ | $j_1^2 C_2$ |
|--------------------|---------------------------|-------------------------|------------------|-----------------|-----------------|-----------------|
| Nystrom [34] | $\frac{2}{3}$ | $\frac{2}{3}$ | $-\frac{1}{216}$ | $+\frac{1}{72}$ | $-\frac{1}{72}$ | $-\frac{1}{24}$ |
| RK3 Classic [44] | $\frac{1}{2}$ | 1 | 0 | 0 | $+\frac{1}{24}$ | $-\frac{1}{24}$ |
| Conte and Reeves | 0.6265833 | 0.0754259 | -0.0066 | +0.0105 | -0.113 | $-\frac{1}{24}$ |
| Optimum | $\frac{10-2\sqrt{13}}{6}$ | $\frac{1+\sqrt{13}}{6}$ | -0.0029 | -0.0029 | -0.0029 | $-\frac{1}{24}$ |

According to these formulas, we have to expect that :

- a. the Nystrom formula is often better than the classic formula.
- b. the optimal formula is often the best of all.

Practical Evaluation of the Truncation Error of Rank Four

Single Stage Estimate . Merson's process:

Merson (1957) has derived a process similar in form to the Runge-Kutta methods, which gives, in suitable cases, a single-stage error estimate.

$$30E = 2k_0 - 9k_2 + 8k_3 - k_4 + O(h^6)$$

The process is used for Runge-Kutta order four classical method and the valid use of the process is restricted to the differential equation which are linear in both x and y:

$$y' = f(x,y) = ax + by + c \quad (16)$$

or equations which closely approximate this form. It is natural to inquire whether it is possible to derive a process similar to Merson's method, but without any restrictions on its validity. A variety of such processes are in fact possible; following is an example for a general single-stage error estimate, and it is used for Runge-Kutta order four when

$$m = \frac{1}{3}, n = \frac{1}{3}, r = \frac{3}{4}, r = \frac{3}{4}, s = \frac{3 \times 90}{128}, t = \frac{3 \times 90}{128}, a = \frac{81}{170}, b = \frac{81}{170}, c = \frac{250}{1377},$$

$$d = \frac{250}{1377}:$$

$$E = -\frac{qr}{s} + O(h^6)$$

where

$$q = \frac{19}{24}k_1 - \frac{27}{8}k_2 + \frac{57}{20}k_3 - \frac{4}{15}k_4$$

$$r = \frac{19}{24}k_1 - \frac{27}{8}k_2 + \frac{57}{20}k_3 - \frac{4}{15}k_4$$

$$s = k_3 - k_0$$

The Extrapolation method [22, pp. 80-82] computes y_{2h}^* by two steps of length h and y_{2h} by one step of length $2h$. The average of the error accumulated in the two steps is then estimated by

$$E = \frac{y_{2h}^* - y_{2h}}{31}$$

for a fourth-order Runge-Kutta method. This method will be referred to as doubling and it seems fair to describe it as the standard estimate for Runge-Kutta methods.

It computes y_{2h}^* by two steps of length h and y_{2h} by one step of length $2h$. The average of the error accumulated in the two steps is then estimated by solving twice with different stepsizes and combining the results to estimate the error in one of the values.

Multi-Stage Estimate . The most accurate formula listed by Ceschino and Kuntzmann is of the form

$$E = \frac{1}{60} (11y_{n+1} + 27y_n - 27y_{n-1} - 11y_{n-2}) - \frac{h}{20} (f_{n+1} + 9f_n + 9f_{n-1} + f_{n-2}) \quad (17)$$

One can obtain such a formula via the Hermite interpolating polynomial. Theoretical justification of its use is rather complicated and is only sketched in [7, pp. 248-249].

A detailed derivation shows this formula to be more accurate (as applied to methods of order ≤ 4) when the estimated error is taken to be the error in going from x_{n+1} to x_n . If the error is incurred in going from $x = 0$ to $x = h$, formula (17) becomes

$$E = \frac{1}{60} (11y_{2h} + 27y_h - 11y_{-h}) - \frac{h}{20} (f_{2h} + 9f_h + 9f_0 + f_{-h}) \quad (18)$$

using the initial condition $y_0 = 0$.

Here y_{2h} , y_h and y_{-h} are taken to be the numerical solutions obtained from the formula (15) at points $x = 2h, h, -h$, respectively, where y_{-h} results from an integration proceeding backward one step from $x = 0$ and y_{2h} is the two-step solution as discussed

in the doubling procedure. This way of supplying the needed data for the error formula seems best for an unbiased comparison of the various estimates.

Alternatively, if we wish to anticipate the multistage error estimate (17) as representing the error over x_n to x_{n+1} , the formula for the comparison scheme becomes

$$E = \frac{1}{60} (11y_h - 27y_{-h} - 11y_{-2h}) - \frac{h}{20} (f_h + 9f_0 + 9f_{-h} + f_{-2h}) \quad (19)$$

As before, the values y_{-h} and y_{-2h} are computed solutions obtained via the Runge-Kutta formula (15) by backward integration of one and two steps, respectively.

There are two other multistage formulas:

$$E = \frac{1}{30} (y_{n+1} + 18y_n - 9y_{n-1} - 10y_{n-2}) - \frac{h}{30} (9f_{n+1} + 18f_{n-1} + 3f_{n-2}) \quad (20)$$

and

$$E = \frac{1}{30} (y_{n+1} + 18y_n - 9y_{n-1} - 10y_{n-2}) - \frac{h}{30} (9f_{n+1} + 18f_{n-1} + 3f_{n-2}) \quad (21)$$

These two formulas can be found in formulas V*(a) and V of [7, p. 250], respectively.

(Formula (21) was first given by Morel [47].)

Some Formulas For Multi-step Methods

Explicit Adams two-step method:

$$y_{n+1} = y_n + h \frac{3f(x_n, y_n) - f(x_{n-1}, y_{n-1})}{2}$$

Explicit Adams three-step method:

$$y_{n+1} = y_n + h \frac{23f(x_n, y_n) - 16f(x_{n-1}, y_{n-1}) + 5f(x_{n-2}, y_{n-2})}{12}$$

Explicit Adams four-step method:

$$y_{n+1} = y_n + h \frac{55f(x_n, y_n) - 59f(x_{n-1}, y_{n-1}) + 37f(x_{n-2}, y_{n-2}) - 9f(x_{n-3}, y_{n-3})}{24}$$

Explicit Adams five-step method:

$$y_{n+1} = y_n + h \left[\frac{1901f(x_n, y_n) - 2774f(x_{n-1}, y_{n-1}) + 2616f(x_{n-2}, y_{n-2}) - 1274f(x_{n-3}, y_{n-3}) + 251f(x_{n-4}, y_{n-4})}{720} \right]$$

For more details about these formulas, see [22, p. 192- 196].

CHAPTER III

PROGRAM RESULTS

Stability

In the Runge-Kutta method if there exists a $h_0 > 0$ for each differential equation such that a change in the starting values by a fixed amount produces a bounded change in the numerical solution for all $0 \leq h \leq h_0$, then the method is **stable**. Furthermore we can define **absolute stability** as follows: " A method is **absolutely stable** for a given stepsize h and a given differential equation if the change due to a perturbation of size δ in one of the mesh values y_n is no larger than δ in all subsequent values y_m , $m > n$."

[16]

Dahlquist [48] defined absolute stability using the test equation $y' = \lambda y$. We say that the region of absolute stability is that set of values of h (real nonnegative) and λ (possibly complex) for which a perturbation in a single value y_n will produce a change in subsequent values that does not increase from step to step. For second-order Runge-Kutta methods, we have

$$k_1 = hf(x_n, y_n) = h\lambda y_n$$

$$k_2 = hf(x_n + mh, y_n + mk_1) = h\lambda(y_n + mh\lambda y_n) = h\lambda(1 + mh\lambda)y_n$$

$$y_{n+1} = y_n + ak_1 + bk_2 = y_n + ah\lambda y_n + bh\lambda(1 + mh\lambda)y_n$$

$$= [1 + (a + b)\lambda h + bmh^2\lambda^2]y_n = (1 + \delta_1 h\lambda + \delta_2 h^2\lambda^2)y_n$$

where $\delta_1 = a + b$, $\delta_2 = bm$. [7] We know that $\delta_1 = 1$, $\delta_2 = \frac{1}{2}$; thus, the region of absolute stability is that area in the complex μ -plane in which

$$\left|1 + \mu + \frac{\mu^2}{2}\right| < 1 \quad (22)$$

where $\mu = h\lambda$.

Similarly for the Runge-Kutta order three methods we have

$$y_{n+1} = (1 + \delta_1 h\lambda + \delta_2 h^2 \lambda^2 + \delta_3 h^3 \lambda^3) y_n$$

where $\delta_1 = 1$, $\delta_2 = \frac{1}{2}$, $\delta_3 = \frac{1}{6}$; thus, the region of absolute stability is that area in which

$$\left|1 + \mu + \frac{\mu^2}{2} + \frac{\mu^3}{6}\right| < 1 \quad (23)$$

where $\mu = h\lambda$.

And for the Runge-Kutta order four methods we have

$$y_{n+1} = (1 + \delta_1 h\lambda + \delta_2 h^2 \lambda^2 + \delta_3 h^3 \lambda^3 + \delta_4 h^4 \lambda^4) y_n$$

where $\delta_1 = 1$, $\delta_2 = \frac{1}{2}$, $\delta_3 = \frac{1}{6}$, $\delta_4 = \frac{1}{24}$; thus, the region of absolute stability is that area

in which

$$\left|1 + \mu + \frac{\mu^2}{2} + \frac{\mu^3}{6} + \frac{\mu^4}{24}\right| < 1 \quad (24)$$

where $\mu = h\lambda$.

After solving the inequalities (22), (23) and (24) we get Table II, and it can be shown in Figure 3. [12]

Next we consider the problem

$$\begin{aligned} y' &= \lambda(y - F(x)) + F'(x) \\ y(0) &= F(0) + c_0 e^{\lambda x} \end{aligned} \quad (25)$$

with the solution

$$y = F(x) + c_0 e^{\lambda x}$$

When λ is small but negative, the equation (25) will lie in the region of absolute stability of the test equation $y' = \lambda y$. Details are given in [16, pp. 41-42].

TABLE II
ABSOLUTE STABILITY OF RK METHOD

| Order | Real Interval of absolute stability |
|-------|---|
| 1 | $\left(-\frac{2}{\lambda}, 0\right)$ |
| 2 | $\left(-\frac{2}{\lambda}, 0\right)$ |
| 3 | $\left(-\frac{2.51}{\lambda}, 0\right)$ |
| 4 | $\left(-\frac{2.78}{\lambda}, 0\right)$ |

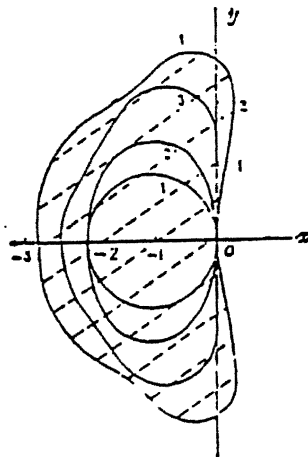


Figure 3. Absolute Stability Region In the Complex μ -plane [12]
($x = \text{Re}(\mu)$, $y = \text{Im}(\mu)$)

About the Program of This Paper

In this thesis, I developed a program for solving the ordinary differential equation:

$$y' = nx^{n-1} + c(y - x^n), \quad y(x_0) = y_0 \quad (26)$$

where n is an integer, the coefficient c , x_0 , y_0 are real numbers. The program includes the following main subroutines to solve equation (26):

1. Euler method (Runge-Kutta order one), using formula (4).
2. Runge-Kutta method of order two, using formula (13).
3. Runge-Kutta method of order three, using formula (14).
4. Runge-Kutta method of order four, using formula (15).
5. AUTO is a control subroutine for testing variables: the coefficient c of equation (26), the stepsize h of numerical methods and comparing some special cases of the same order, for same stepsize and the same coefficient c .

The rest of the program is for the purpose of some special needs that are not so important for this thesis, for example, the selection of the variable n , etc. All of the programs were run on an IBM-compatible PC and also on the VMS/VAX mainframe at Oklahoma State University using double precision computations.

The interesting part of the test is try to find the relationship between the global error and the parameters , such as c , n in (26), for a stepsize h within its absolute stability region for the Runge-Kutta methods. Here the global errors we got were using the exact solution minus the numerical solution and neglecting the roundoff error of the computer.

Test Problems Design and the Results

Single-Step Method

1. In equation (26), let $n = 1$, $c = -1000$, $x_0 = 0$, $y_0 = 0$. Then the equation will be:

$$y' = 1 - 1000(y - x), y(0) = 1$$

This gives information about the effect of the global error due to changing the stepsize h , listed in Table III and Table IV.

TABLE III
ERROR VERSUS STEPSIZE I

| # of iterations | Stepsize h | Error (Euler) | Error (RK2) |
|-----------------|--------------|-----------------|---------------|
| 500 | 0.002 | 1.0 | 1.0 |
| 527 | 0.0019005 | 0.0 | 0.11102E-14 |
| 556 | 0.0018001 | 0.11102E-15 | 0.11102E-15 |
| 588 | 0.001705 | 0.0 | 0.11102E-15 |
| 625 | 0.001602 | 0.0 | 0.0 |
| 666 | 0.0015025 | 0.0 | 0.0 |
| 713 | 0.001403 | 0.0 | 0.0 |
| 768 | 0.0013035 | 0.0 | 0.0 |
| 831 | 0.001204 | 0.0 | 0.0 |
| 906 | 0.0011045 | 0.0 | 0.0 |
| 996 | 0.001005 | 0.0 | 0.0 |
| 1105 | 0.0009055 | 0.0 | 0.0 |
| 1241 | 0.000806 | 0.0 | 0.0 |
| 1416 | 0.0007065 | 0.0 | 0.0 |
| 1648 | 0.000607 | 0.0 | 0.0 |
| 1971 | 0.0005075 | 0.0 | 0.0 |
| 2451 | 0.000408 | 0.0 | 0.0 |

TABLE III (Continued)

| # of iterations | Stepsize h | Error (Euler) | Error (RK2) |
|-----------------|------------|-----------------|---------------|
| 3242 | 0.0003085 | 0.0 | 0.0 |
| 4785 | 0.000209 | 0.0 | 0.0 |
| 9133 | 0.0001095 | 0.11102E-15 | 0.11102E-15 |

TABLE IV
ERROR VERSUS STEPSIZE II

| # of iter. | Stepsize h | Error (RK3) | # of iter. | Stepsize h | Error (RK4) |
|---------------|---------------|-----------------|---------------|---------------|----------------|
| 399 | 0.00251 | Increasing | 514 | 0.001949 | 0.0 |
| 420 | 0.002385 | 0.0 | 553 | 0.0018105 | 0.0 |
| 443 | 0.00226 | 0.0 | 559 | 0.001672 | 0.0 |
| 469 | 0.002135 | 0.0 | 653 | 0.0015335 | 0.0 |
| 498 | 0.00201 | 0.0 | 717 | 0.001395 | 0.0 |
| 531 | 0.001885 | 0.0 | 796 | 0.0012565 | 0.0 |
| 569 | 0.00176 | 0.0 | 895 | 0.00118 | 0.0 |
| 612 | 0.001635 | 0.0 | 1021 | 0.0009795 | 0.0 |
| 663 | 0.00151 | 0.0 | 1196 | 0.000841 | 0.0 |
| 723 | 0.001385 | 0.0 | 1424 | 0.0007025 | 0.0 |
| 794 | 0.00126 | 0.0 | 1774 | 0.000564 | 0.0 |
| 882 | 0.001135 | 0.0 | 2351 | 0.0004255 | 0.0 |
| 991 | 0.00101 | 0.0 | 3485 | 0.000287 | 0.0 |

TABLE IV (Continued)

| # of iter. | Stepsize h | Error (RK3) | # of iter. | Stepsize h | Error (RK4) |
|---------------|---------------|-----------------|---------------|---------------|----------------|
| 1130 | 0.000885 | 0.0 | 6735 | 0.0001485 | 0.0 |
| 1316 | 0.00076 | 0.0 | | | |
| 1575 | 0.000635 | 0.0 | | | |
| 1961 | 0.00051 | 0.0 | | | |
| 2598 | 0.000385 | 0.0 | | | |
| 3847 | 0.00026 | 0.0 | | | |
| 7408 | 0.000135 | 0.0 | | | |

Notes for all the Tables of this paper:

- a. Errors in the Tables of this paper are the global errors.
- b. Each number in the error column refers to the error after some iterations, the global error which is approximately constant until the end of the iterations.
- c. If the exponent part of the error is smaller than -13 it is caused by roundoff error so we can say the truncation error is zero.

Notes for Table III and Table IV:

- a. In Table III, RK2 refers to the Modified Euler method.
 - b. In Table IV, RK3 refers to the "Optimal" Runge-Kutta order 3 method.
 - c. In Table IV, RK4 refers to the "Optimal" Runge-Kutta order 4 method.
 - d. For $n=1$, the global errors are all zero for the initial value problem $y_0 = 0$.
2. In equation (26), let $n = 2$, $c = -1000$, $x_0 = 0$, $y_0 = 0$. Then the equation will be:

$$y' = 2x - 1000(y - x^2), \quad y(0) = 0$$

This gives information about the effect of the global error due to changing the stepsize h , listed in Table V and Table VI:

TABLE V
ERROR VERSUS STEPSIZE III

| # of iteration | Stepsize h | Error (Euler) | Error (RK2) |
|----------------|--------------|-----------------|---------------|
| 500 | 0.002 | Increasing | Increasing |
| 527 | 0.0019005 | 0.19005E -5 | 0.36301E-4 |
| 556 | 0.0018001 | 0.18010E-5 | 0.16300E-4 |
| 588 | 0.001705 | 0.17015E-5 | 0.96988E-5 |
| 625 | 0.001602 | 0.16020E-5 | 0.64483E-5 |
| 666 | 0.0015025 | 0.15025E-5 | 0.45377E-5 |
| 713 | 0.001403 | 0.14030E-5 | 0.32972E-5 |
| 768 | 0.0013035 | 0.13035E-5 | 0.24395E-5 |
| 831 | 0.001204 | 0.12040E-5 | 0.18211E-5 |
| 996 | 0.001005 | 0.10050E-5 | 0.10151E-5 |
| 1105 | 0.0009055 | 0.90550E-6 | 0.74914E-6 |
| 1241 | 0.000806 | 0.80600E-6 | 0.54408E-6 |
| 1416 | 0.0007065 | 0.70650E-6 | 0.38589E-6 |
| 1648 | 0.000607 | 0.60700E-6 | 0.26450E-6 |
| 1971 | 0.0005075 | 0.50750E-6 | 0.17257E-6 |
| 2451 | 0.000408 | 0.40800E-6 | 0.10456E-6 |

TABLE V (Continued)

| # of iteration | Stepsize h | Error (Euler) | Error (RK2) |
|----------------|------------|-----------------|---------------|
| 3242 | 0.0003085 | 0.30850E-6 | 0.56265E-7 |
| 4785 | 0.000209 | 0.20900E-6 | 0.24389E-7 |
| 9133 | 0.0001095 | 0.10950E-6 | 0.63424E-8 |

TABLE VI
ERROR VERSUS STEPSIZE IV

| # of iter. | Stepsize h | Error (RK3) | # of iter. | Stepsize h | Error (RK4) |
|---------------|---------------|------------------|---------------|---------------|------------------|
| 399 | 0.00251 | Increasing | 360 | 0.00278 | Increasing |
| 420 | 0.002385 | 0.13910E-5 | 379 | 0.0026415 | 0.13668E-4 |
| 443 | 0.00226 | 0.12398E-5 | 400 | 0.002503 | 0.58710E-5 |
| 469 | 0.002135 | 0.10892E-5 | 423 | 0.0023645 | 0.32766E-5 |
| 498 | 0.00201 | 0.94127E-6 | 450 | 0.00226 | 0.22617E-5 |
| 531 | 0.001885 | 0.79864E-6 | 480 | 0.0020875 | 0.13035E-5 |
| 569 | 0.00176 | 0.66379E-6 | 514 | 0.001949 | 0.85859E-6 |
| 612 | 0.001635 | 0.53914E-6 | 553 | 0.0018105 | 0.56844E-6 |
| 663 | 0.00151 | 0.42675E-6 | 559 | 0.001672 | 0.37415E-6 |
| 723 | 0.001385 | 0.32815E-6 | 653 | 0.0015335 | 0.24259E-6 |
| 794 | 0.00126 | 0.24420E-6 | 717 | 0.001395 | 0.13568E-6 |
| 882 | 0.001135 | 0.17502E-6 | 796 | 0.0012565 | 0.94036E-7 |
| 991 | 0.00101 | 0.12002E-6 | 895 | 0.00118 | 0.70416E-7 |

TABLE VI (Continued)

| # of iter. | Stepsize h | Error (RK3) | # of iter. | Stepsize h | Error (RK4) |
|---------------|---------------|-----------------|---------------|---------------|----------------|
| 1130 | 0.000885 | 0.78045E-7 | 1021 | 0.0009795 | 0.30391E-7 |
| 1316 | 0.00076 | 0.47478E-7 | 1196 | 0.000841 | 0.15495E-7 |
| 1575 | 0.000635 | 0.26458E-7 | 1424 | 0.0007025 | 0.70810E-8 |
| 1961 | 0.00051 | 0.13035E-7 | 1774 | 0.000564 | 0.27609E-8 |
| 2598 | 0.000385 | 0.53123E-8 | 2351 | 0.0004255 | 0.83872E-9 |
| 3847 | 0.00026 | 0.15451E-8 | 3485 | 0.000287 | 0.16261E-9 |
| 7408 | 0.000135 | 0.20374E-9 | 6735 | 0.0001485 | 0.10902E-10 |

Notes for Table V and Table VI:

- a. In Table V, RK2 refers to the Modified Euler method.
- b. In Table VI, RK3 refers to the "Optimal" Runge-Kutta order 3 method.
- c. In Table VI, RK4 refers to the Classical Runge-Kutta order 4 method.
- d. The results of the initial value $y_0 = 1$ are the same as that of $y_0 = 0$ which were shown in Table V and Table VI except the global errors of Euler method and Runge-Kutta order two method are 1.0 when the stepsize is 0.002 when $y_0 = 1$.

Figure 4 through Figure 9 give the graphs according to Table V and Table VI.

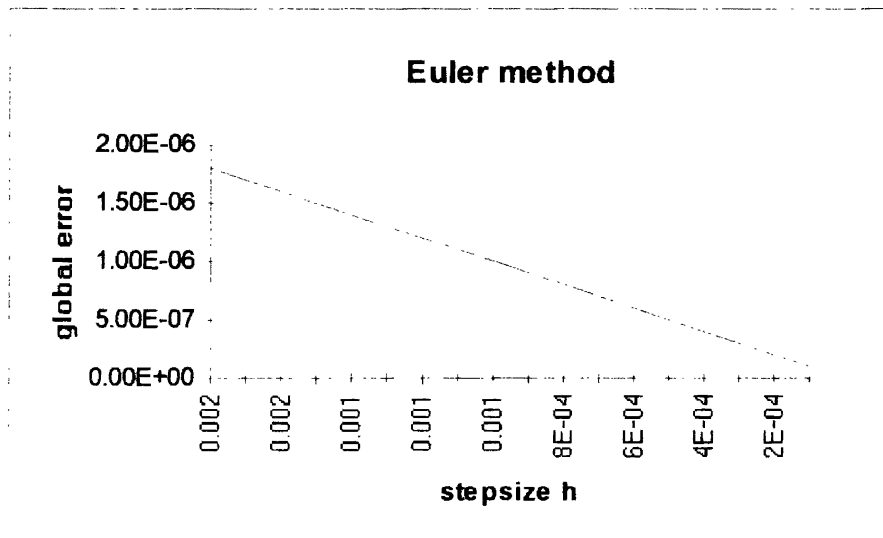


Figure 4. Error Versus Stepsize I

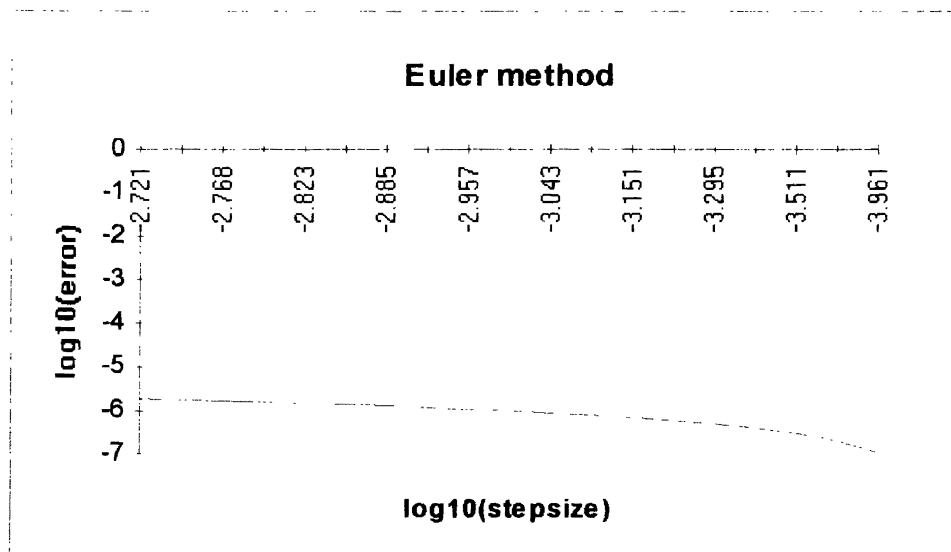


Figure 5. Error Versus Stepsize II

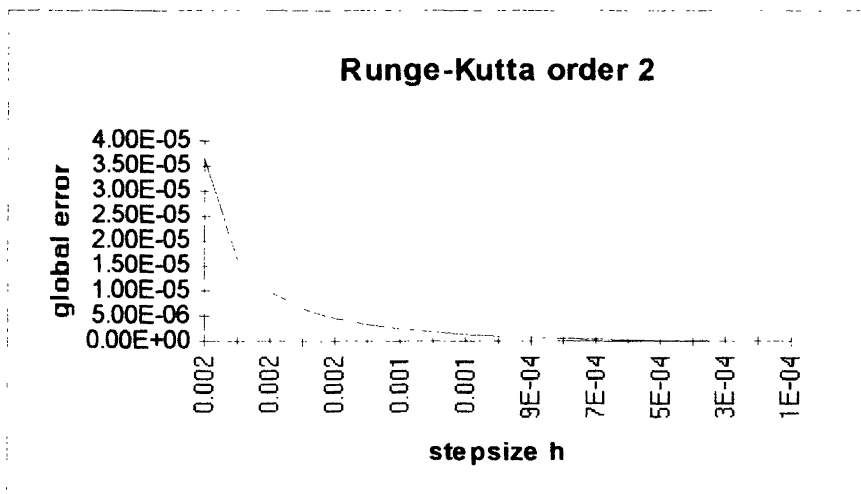


Figure 6. Error Versus Stepsize III

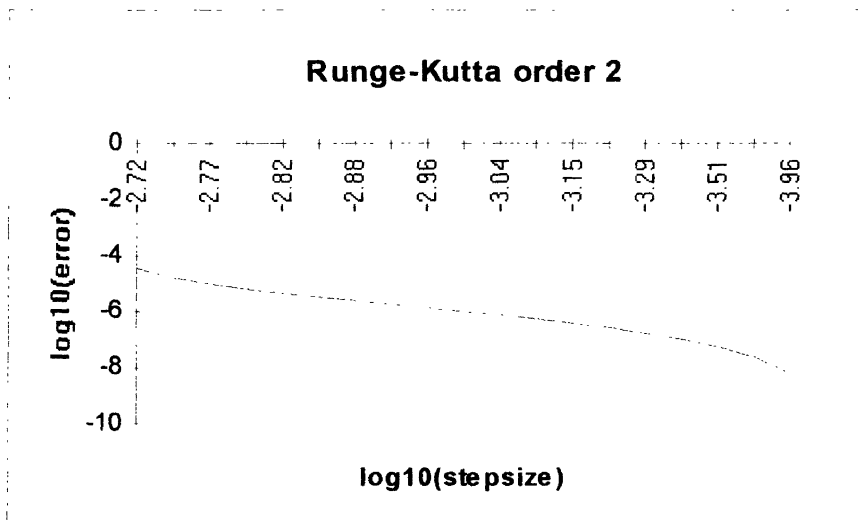


Figure 7. Error Versus Stepsize IV

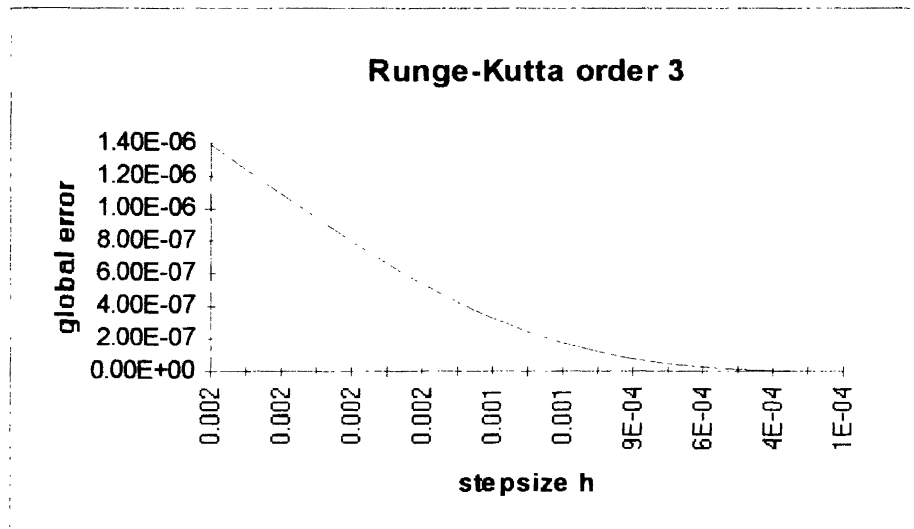


Figure 8. Error Versus Stepsize V

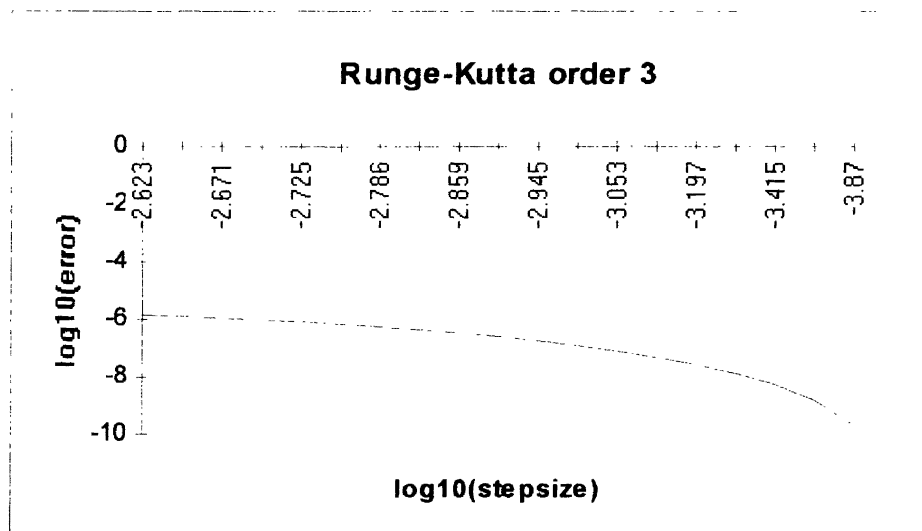


Figure 9. Error Versus Stepsize VI

3. In equation (26), let $n = 3$, $c = -1000$, $x_0 = 0$, $y_0 = 0$. Then the equation will be:

$$y' = 3x^2 - 1000(y - x^3), \quad y(0) = 0$$

The global error is not a constant so we just ignore the resulting data.

4. In equation (26), let $n = 1$, $h = 0.000125$, $x_0 = 0$, $y_0 = 1$. Then the equation will be:

$$y' = 1 - c(y - x), \quad y(0) = 1$$

This gives information about the effect of the global error due to changing the coefficient c , listed in Table VII:

TABLE VII
ERROR VERSUS COEFFICIENT I

| Coeff. c | Error (Euler) | Error (RK2) | Error (RK3) | Error (RK4) |
|----------|---------------|-------------|-------------|-------------|
| -1000 | 0.0 | 0.0 | 0.0 | 0.0 |
| -950 | 0.0 | 0.0 | 0.0 | 0.0 |
| -900 | 0.0 | 0.0 | 0.0 | 0.0 |
| -850 | 0.0 | 0.0 | 0.0 | 0.0 |
| -800 | 0.0 | 0.0 | 0.0 | 0.0 |
| -750 | 0.0 | 0.0 | 0.0 | 0.0 |
| -700 | 0.0 | 0.0 | 0.0 | 0.0 |
| -650 | 0.0 | 0.0 | 0.0 | 0.0 |
| -600 | 0.0 | 0.0 | 0.0 | 0.0 |
| -550 | 0.11102E-15 | 0.11102E-15 | 0.11102E-15 | 0.11102E-15 |
| -500 | 0.11102E-15 | 0.11102E-15 | 0.11102E-15 | 0.11102E-15 |
| -450 | 0.11102E-15 | 0.11102E-15 | 0.11102E-15 | 0.11102E-15 |

TABLE VII (Continued)

| Coeff. c | Error (Euler) | Error (RK2) | Error (RK3) | Error (RK4) |
|----------|---------------|-------------|-------------|-------------|
| -400 | 0.11102E-15 | 0.11102E-15 | 0.11102E-15 | 0.11102E-15 |
| -350 | 0.11102E-15 | 0.11102E-15 | 0.11102E-15 | 0.11102E-15 |
| -300 | 0.11102E-15 | 0.11102E-15 | 0.11102E-15 | 0.11102E-15 |
| -250 | 0.33307E-15 | 0.44409E-15 | 0.33307E-15 | 0.44409E-15 |
| -200 | 0.44409E-15 | 0.55511E-15 | 0.44409E-15 | 0.26895E-15 |
| -150 | 0.66613E-15 | 0.77716E-15 | 0.66613E-15 | 0.66613E-15 |
| -100 | 0.99920E-15 | 0.11102E-14 | 0.99920E-15 | 0.11102E-14 |
| -50 | 0.21094E-14 | 0.22204E-14 | 0.21094E-14 | 0.22204E-14 |

Notes of Table VII:

- a. In Table VII, RK2 refers to the Modified Euler method.
- b. In Table VII, RK3 refers to the "Optimal" Runge-Kutta order 3 method.
- c. In Table VII, RK4 refers to the Optimal Runge-Kutta order 4 method.
- d. For $n = 1$, the global errors for the initial value problem $y_0 = 0$ are all zero.
5. In equation (26), let $n = 2$, $h = 0.000125$, $x_0 = 0$, $y_0 = 0$. Then the equation

will be:

$$y' = 2x - c(y - x^2), \quad y(0) = 0$$

This gives information about the effect of the global error due to changing the coefficient c . The results are listed in Table VIII:

TABLE VIII
ERROR VERSUS COEFFICIENT II

| Coeff. c | Error (Euler) | Error (RK2) | Error (RK3) | Error (RK4) |
|----------|---------------|-------------|-------------|-------------|
| -1000 | 0.12500E-6 | 0.83333E-8 | 0.16095E-9 | 0.54101 -11 |
| -950 | 0.13158E-6 | 0.83056E-8 | 0.15243E-9 | 0.48677E-11 |
| -900 | 0.13889E-6 | 0.82781E-8 | 0.14397E-9 | 0.43554E-11 |
| -850 | 0.14706E-6 | 0.82508E-8 | 0.13556E-9 | 0.38729E-11 |
| -800 | 0.15625E-6 | 0.82237E-8 | 0.12719E-9 | 0.34199E-11 |
| -750 | 0.16667E-6 | 0.81967E-8 | 0.11888E-9 | 0.29965E-11 |
| -700 | 0.17857E-6 | 0.81699E-8 | 0.11061E-9 | 0.26021E-11 |
| -650 | 0.19231E-6 | 0.81433E-8 | 0.10240E-9 | 0.22364E-11 |
| -600 | 0.20833E-6 | 0.81169E-8 | 0.94230E-10 | 0.18996E-11 |
| -550 | 0.22727E-6 | 0.80906E-8 | 0.86111E-10 | 0.15909E-11 |
| -500 | 0.25000E-6 | 0.80645E-8 | 0.78042E-10 | 0.13105E-11 |
| -450 | 0.27778E-6 | 0.80338E-8 | 0.70022E-10 | 0.10578E-11 |
| -400 | 0.31250E-6 | 0.80128E-8 | 0.6205E-10 | 0.83267E-12 |
| -350 | 0.35714E-6 | 0.79872E-8 | 0.54126E-10 | 0.63505E-12 |
| -300 | 0.41667E-6 | 0.79618E-8 | 0.46251E-10 | 0.46430E-12 |
| -250 | 0.50000E-6 | 0.79365E-8 | 0.38423E-10 | 0.32041E-12 |
| -200 | 0.62500E-6 | 0.79114E-8 | 0.30644E-10 | 0.20273E-12 |
| -150 | 0.83333E-6 | 0.78864E-8 | 0.22914E-10 | 0.11124E-12 |
| -100 | 0.12500E-5 | 0.78616E-8 | 0.15232E-10 | decreasing |
| -50 | 0.25000E-5 | 0.78370E-8 | 0.76000E-11 | decreasing |

Notes for the Table VIII:

- a. In Table VIII, RK2 refers to the Modified Euler method.
- b. In Table VIII, RK3 refers to the "Optimal" Runge-Kutta order 3 method.
- c. In Table VIII, RK4 refers to the Classical Runge-Kutta order 4 method.
- d. The results of the initial problem $y_0 = 1$ is the same as that of $y_0 = 0$ which were shown in Table VIII.

Figure 10 through 12 gives the graphs according to Table VIII:

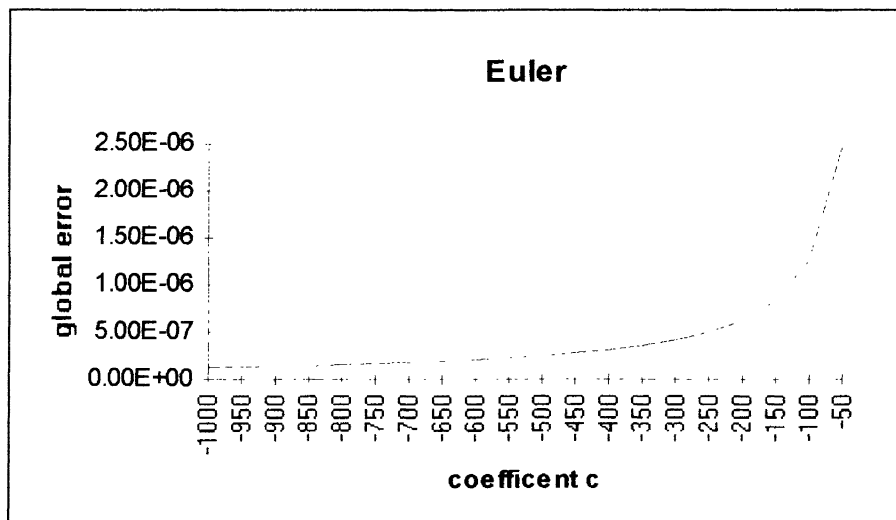


Figure 10. Error Versus Coefficient I

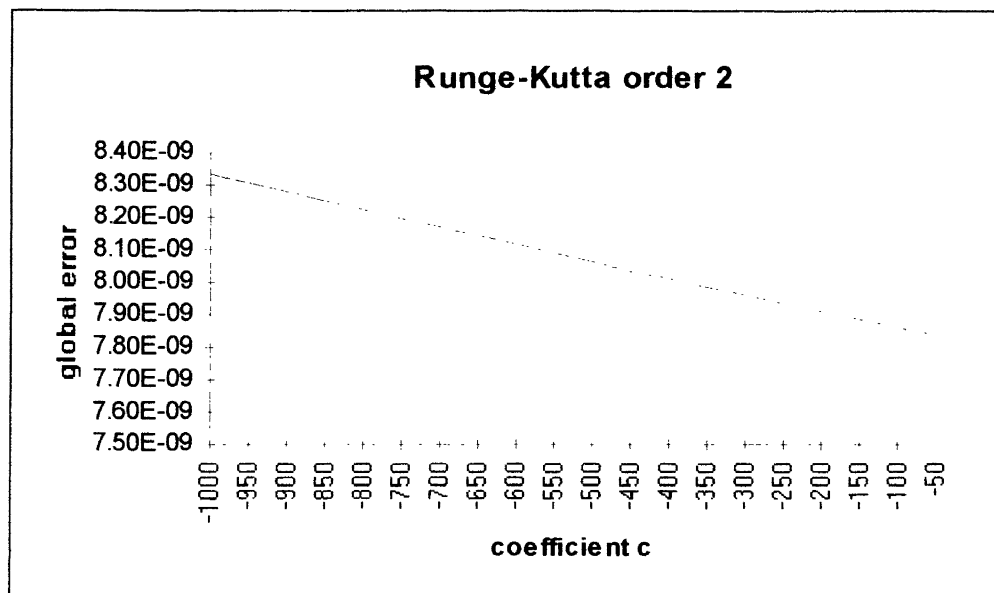


Figure 11. Error Versus Coefficient II

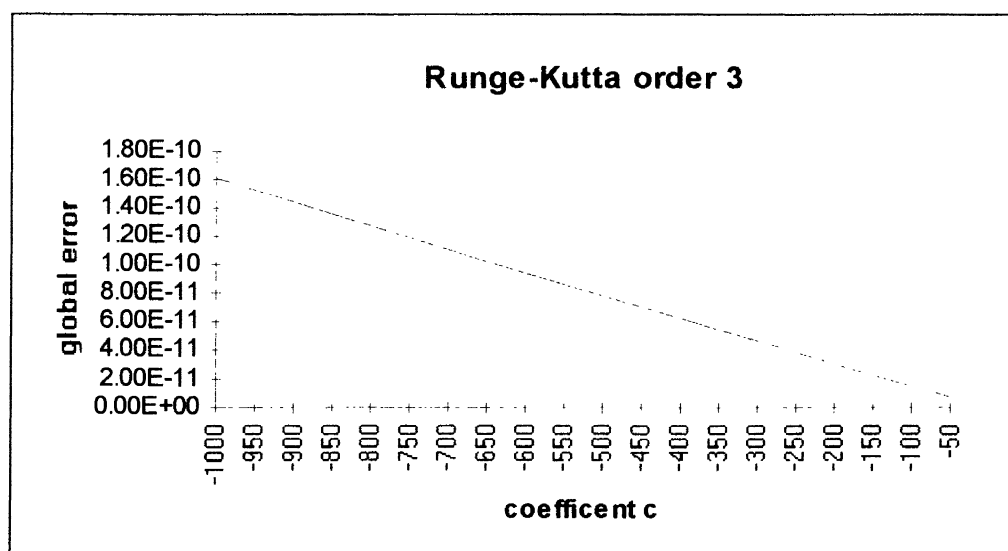


Figure 12. Error Versus Coefficient III

6. In equation (26), let $n = 1$, $h = 0.000125$, $c = -1000$, $x_0 = 0$, $y_0 = 0$. Then the equation is:

$$y' = 1 - 1000(y-x), \quad y(0) = 0$$

This gives information about the effect of the global error due to using different cases of the same order of Runge-Kutta methods. Results are listed in Tables IX , X and XI:

TABLE IX
COMPARING I. (RUNGE-KUTTA 2)

| Method | Global Error |
|---------------------------|--------------|
| Midpoint method | 0.0 |
| RK2 Heun method | 0.0 |
| RK2 modified Euler method | 0.0 |

TABLE X
COMPARING I. (RUNGE-KUTTA 3)

| Method | Global Error |
|-------------------------|--------------|
| RK3 Heun method | 0.0 |
| Classical method | 0.0 |
| Conte and Reeves method | 0.0 |
| Optimal method | 0.0 |

TABLE XI
COMPARING I. (RUNGE-KUTTA 4)

| Method | Global Error |
|-------------------------|--------------|
| Classical method | 0.0 |
| Gill method | 0.0 |
| Merson method | 0.0 |
| Hull and Johnson method | 0.0 |
| Boulton method | 0.0 |
| Optimal method | 0.0 |

Notes on Table IX through Table XI

The results of the initial problem $y_0 = 1$ are almost the same as that for $y_0 = 0$ which were shown in Table XI and Table XI.

7. In equation (26), let $n = 2$, $h = 0.000125$, $c = -1000$, $x_0 = 0$, $y_0 = 0$. Then the equation is:

$$y' = 2x - 1000(y - x^2), \quad y(0) = 0$$

This gives information about the effect of the global error due to using different cases of the same order of Runge-Kutta methods. The results are listed in Table XII through Table XIV.

Notes on Table XII through Table XIV

The results of the initial problem $y_0 = 1$ are the same as that for $y_0 = 0$ which were shown in Table XII and Table XIV.

TABLE XII
COMPARING II. (RUNGE-KUTTA 2)

| Method | Global Error |
|---------------------------|--------------|
| Midpoint method | 0.41667E-8 |
| RK2 Heun method | 0.55556E-8 |
| RK2 modified Euler method | 0.83333E-8 |

TABLE XIII
COMPARING II. (RUNGE-KUTTA 3)

| Method | Global Error |
|-------------------------|--------------|
| RK3 Heun method [27] | 0.23084E-9 |
| Classical method | 0.17313E-9 |
| Conte and Reeves method | 0.21695E-9 |
| Optimal method | 0.16095E-9 |

TABLE XIII
COMPARING II. (RUNGE-KUTTA 4)

| Method | Global Error |
|-------------------------|--------------|
| Classical method | 0.54109E-11 |
| Gill method | 0.54100E-11 |
| Merson method | Increasing |
| Hull and Johnson method | Increasing |
| Boulton method | Increasing |
| Optimal method | Increasing |

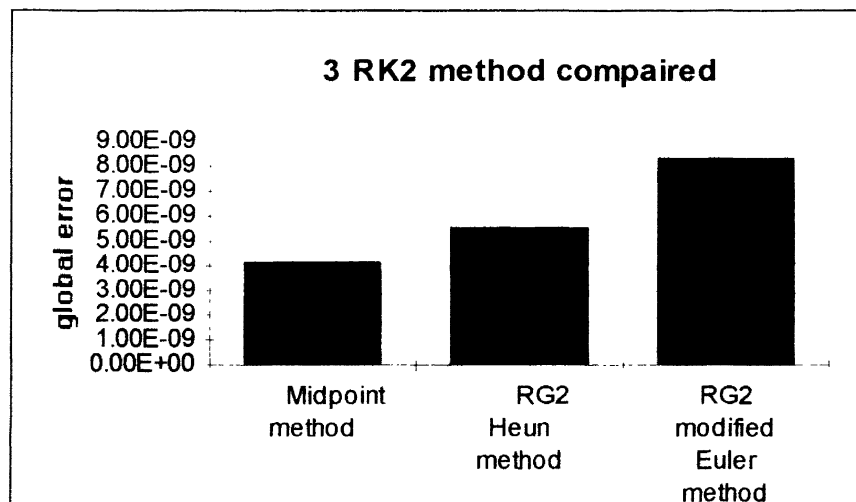


Figure 13. Different Methods Compared I

Figure 13 gives the graphs according to Table XII, and Figure 14 gives the graphs according to Table XIII.

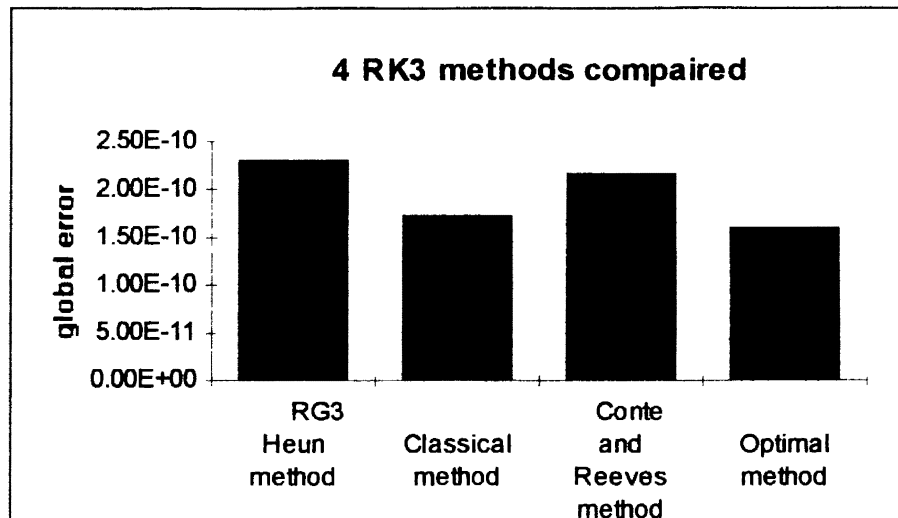


Figure 14. Different Methods Compared II

Multi-step Methods

For the initial problem with $y_0 = 0$, the global errors for all multi-step methods of order two or greater are all zero when we choose any configurations used above.

Summary of the Program Results

1. For the problem (26), where $n = 1, 2$, after some iterations the results are the same no matter the initial value $y_0 = 0$ or $y_0 = 1$.
2. Using the multi-step method to solve the differential equation (26), we can get the exact solution if the initial value is $y_0 = 0$.

3. For the differential equation (26), for a constant coefficient c , the smaller the stepsize h the smaller the global error.
4. For the differential equation (26), for a constant stepsize h , the smaller the coefficient c the smaller the global error.
5. For the differential equation (26), the midpoint method is the most accurate compared to the other two Runge-Kutta order 2 methods used in the program.
6. For the differential equation (26), the "optimal" Runge-Kutta method is the most accurate compared to the other three Runge-Kutta order 3 methods used in the program..
7. For the differential equation (26), the classical and Gill methods are the most accurate compared to the other five Runge-Kutta order 4 methods used in the program.
8. For the differential equation (26), if $n \geq 3$ the global errors of Runge-Kutta methods are neither zero nor constant after some iterations.

CHAPTER IV

ANALYSES AND DISCUSSION

Roundoff Error and Truncation Error

For the equation (2) if we use the discretization method

$$y_{n+1} = y_n + \Phi(x_n, y_n, h)$$

where $\Phi(x, y, h)$ satisfies the Lipschitz condition and is continuous (jointly as a function of its three arguments) in the region defined by x , y , and h , further more if the local roundoff error ε_n under the assumption that $|\varepsilon_n| \leq \varepsilon$, ε is a constant, then the accumulated roundoff error r_n satisfies: [7]

$$|r_n| \leq \frac{\varepsilon}{h} E_L(x_n - a), \quad a \leq x_n \leq b \quad (27)$$

Here L is the Lipschitz constant, $E_L(x)$ is the Lipschitz function which is defined as:

$$E_L(x) = \frac{e^{Lx} - 1}{L}, \quad \text{if } L > 0$$
$$E_L(x) = x, \quad \text{if } L = 0$$

We can see that the estimate (27) does not depend on the order of the discretization method and the total number of iterations N . This indicates that the propagation of roundoff error is not related to the discretization error of a method. As we know, for fixed x_n the maximum global truncation error decreases as the stepsize h decreases, and the estimate (27) also shows that for fixed x_n the maximum roundoff error increases as h decreases. The latter is reasonable since decreasing h increases the number of steps to reach x_n , so that the accumulated roundoff error is likely to increase. The overall effect is that the maximum error will decrease with h until at some value it reaches a minimum,

after which the maximum error increases as h is decreased (Figure 19). [16, p.19]

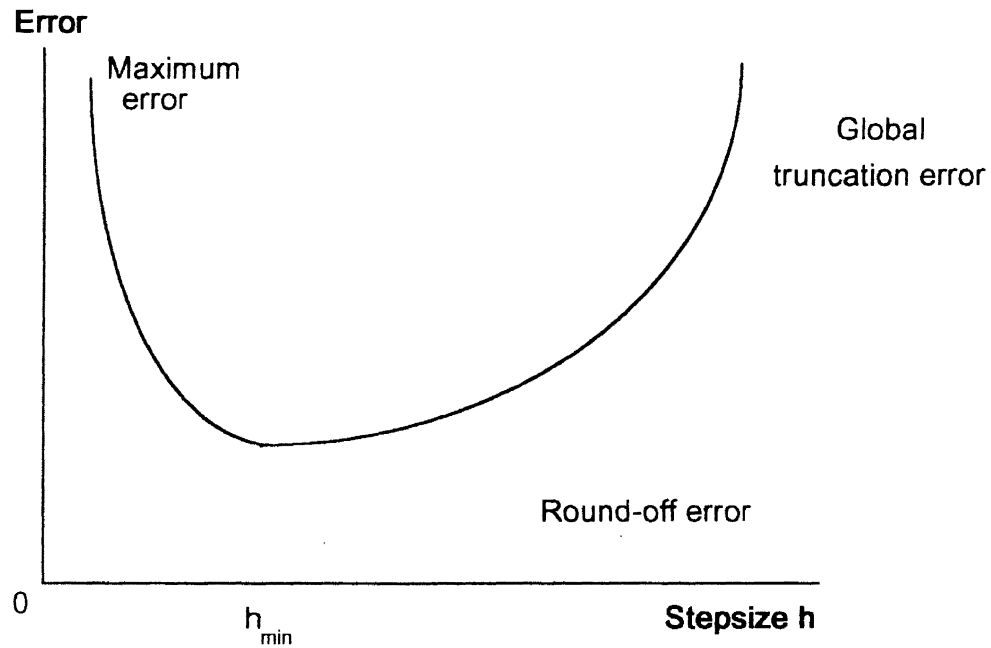


Figure 19 . the Minimum Error as a Function h

As mentioned above, in this paper we control the stepsize h and avoid it becoming too small, so only truncation error will be considered here.

Analytic Truncation Error

In Chapter III we were concerned about the equation (26)

$$y' = nx^{n-1} + c(y - x^n), \quad y(x_0) = y_0$$

For this particular equation, the analytic solution is

$$y = y_0 e^{cx} + x^n, \quad \text{if } x_0 = 0 \quad (28)$$

For reasons of accuracy, we use the posterior estimate method to analyze the truncation error of the Runge-Kutta methods of solving equation (26). In Chapter III, we know that the only interesting case of the equation (26) is for $n = 1$ and $n = 2$. Avoiding redundancy, we only study the case of $n = 2$, and the case of $n = 1$ is simpler than the case of $n = 2$ and the method of the justification is the same. In other words, we only consider the ordinary differential equation:

$$y' = 2x + c(y - x^2), \quad y(0) = y_0, \quad 0 \leq x \leq 1 \quad (29)$$

and the solution of the equation (29) is:

$$y = y_0 x^2 + e^{cx} \quad (30)$$

The solution of the difference equation

$$x_{n+1} = Ax_n + B, \quad x_0 = 0 \quad (31)$$

is

$$x_n = \frac{B(A^n - 1)}{A - 1} \quad (32)$$

proof:

From (31) we have:

$$\begin{aligned} x_{n+1} - x_n &= A(x_n - x_{n-1}) \\ &= A^2(x_{n-1} - x_{n-2}) \\ &= \dots \\ &= A^n(x_1 - x_0) \end{aligned}$$

and

$$x_1 = Ax_0 + B = A \times 0 + B = B$$

so

$$x_{n+1} - x_n = A^n B \quad (33)$$

(33) - (31)

$$0 = (1 - A)x_n + B(A^n - 1)$$

$$x_n = \frac{B(A^n - 1)}{A - 1}$$

end of proof.

Global Truncation Error e_n for the Euler Method (Runge-Kutta order one)

Theorem 1. The global truncation error e_n of the Euler method for problem (29) when

$y_0 = 1$ is

$$e_{n+1} = (1 + hc)e_n + (1 + hc - e^{ch})e^{cx_n} - h^2 \quad (34)$$

where h is the stepsize, if N is the total iteration of the method, then $hN = 1$ and $x_n = \frac{n}{N}$

= nh .

Proof:

$$\text{Let } k_1 = hf(x_n , y_n) = h[c(y_n - x_n^2) + 2x_n] = hc(y_n - x_n^2) + 2x_n h \quad (35)$$

$$\begin{aligned} e_{n+1} &= y_{n+1} - y(x_{n+1}) = [y_n + hf(x_n , y_n)] - [(x_n + h)^2 + e^{c(x_n + h)}] \\ &= y_n + hc(y_n - x_n^2) + 2x_n h - x_n^2 - 2x_n h - h^2 - e^{ch} e^{cx_n} \\ &= y_n - x_n^2 + hc(y_n - x_n^2) - h^2 - e^{ch} e^{cx_n} \\ &= (1 + hc)(y_n - x_n^2) - h^2 - e^{ch} e^{cx_n} \end{aligned}$$

because

$$\begin{aligned} e_n &= y_n - y(x_n) = y_n - (x_n^2 + e^{cx_n}) \\ y_n - x_n^2 &= e_n + e^{cx_n} \end{aligned} \quad (36)$$

Substitute, we can get

$$\begin{aligned} e_{n+1} &= (1 + hc)(e_n + e^{cx_n}) - h^2 - e^{ch} e^{cx_n} \\ &= (1 + hc)e_n + (1 + hc - e^{ch})e^{cx_n} - h^2 \end{aligned}$$

end of proof.

The local truncation error T_{n+1} of the Euler method for the problem (29) when

$y_0 = 1$ is

$$T_{n+1} = (1 + hc - e^{ch})e^{cx_n} - h^2 \quad (37)$$

The proof of (37) is the following:

$$\begin{aligned} T_{n+1} &= [y(x_n) + hf(x_n, y(x_n))] - y(x_{n+1}) \\ &= [x_n^2 + e^{cx_n} + hc(x_n^2 + e^{cx_n} - x_n^2) + 2x_n h] - [(x_n + h)^2 + e^{c(x_n + h)}] \\ &= (1 + hc - e^{ch})e^{cx_n} - h^2 \end{aligned}$$

So.

$$e_{n+1} = (1 + hc)e_n + T_{n+1}. \quad (38)$$

Note:

In problem (29), if the initial value $y(0) = 0$, the global truncation error formula (38) also holds for the Euler method except

$$T_{n+1} = -h^2 \quad (39)$$

Global Truncation Error e_n for Runge-Kutta Methods of Order Two

Theorem 2. The global truncation error e_n for a Runge-Kutta method of order two for problem (29) when $y_0 = 1$ is

$$e_{n+1} = \left(1 + hc + \frac{h^2 c^2}{2!}\right)e_n + \left(1 + hc + \frac{h^2 c^2}{2!} - e^{ch}\right)e^{cx_n} - \frac{mch^3}{2} \quad (40)$$

where h is the stepsize, if N is the total number of iterations of the method, then $hN = 1$

and $x_n = \frac{n}{N} = nh$.

Proof:

From (35) we have

$$\begin{aligned} k_1 &= hf(x_n, y_n) = hc(y_n - x_n^2) + 2x_n h \\ k_2 &= hf(x_n + mh, y_n + mk_1) = hf(x_n + mh, y_n + mhc(y_n - x_n^2) + m2x_n h) \\ &= hc[(y_n + mhc(y_n - x_n^2) + m2x_n h) - (x_n + mh)^2] + 2h(x_n + mh) \\ &= hc[y_n + mhc(y_n - x_n^2) + m2x_n h - x_n^2 - m2x_n h - m^2 h^2] + 2x_n h + 2mh^2 \\ &= hc(y_n - x_n^2) + mh^2 c^2 (y_n - x_n^2) - m^2 ch^3 + 2x_n h + 2mh^2 \\ &= (hc + mh^2 c^2)(y_n - x_n^2) - m^2 ch^3 + 2x_n h + 2mh^2 \end{aligned} \quad (41)$$

$$\begin{aligned}
e_{n+1} &= y_{n+1} - y(x_{n+1}) = [y_n + ak_1 + bk_2] - [(x_n + h)^2 + e^{c(x_n+h)}] \\
&= y_n + ahc(y_n - x_n^2) + a2x_n h + b(hc + mh^2c^2)(y_n - x_n^2) - bm^2ch^3 \\
&\quad + b2x_n h + b2mh^2 - x_n^2 - 2x_n h - h^2 - e^{ch}e^{cx_n} \\
&= [1 + (a + b)hc + bmh^2c^2](y_n - x_n^2) + [2(a + b) - 2]x_n h \\
&\quad + (2bm - 1)h^2 - bm^2ch^3 - e^{ch}e^{cx_n} \\
&= (1 + hc + \frac{h^2c^2}{2!})(y_n - x_n^2) - \frac{mch^3}{2} - e^{ch}e^{cx_n}
\end{aligned}$$

From (36) we have:

$$\begin{aligned}
e_{n+1} &= (1 + hc + \frac{h^2c^2}{2!})(e_n + e^{cx_n}) - \frac{mch^3}{2} - e^{ch}e^{cx_n} \\
&= (1 + hc + \frac{h^2c^2}{2!})e_n + (1 + hc + \frac{h^2c^2}{2!} - e^{ch})e^{cx_n} - \frac{mch^3}{2}
\end{aligned}$$

end of proof.

The local truncation error T_{n+1} for a Runge-Kutta method of order two for the problem (29) is

$$T_{n+1} = (1 + hc + \frac{h^2c^2}{2!} - e^{ch})e^{cx_n} - \frac{mch^3}{2} \quad (42)$$

The proof of (42) is the following:

$$k_1 = hf(x_n, y(x_n)) = h[c(x_n^2 + e^{cx_n} - x_n^2) + 2x_n] = hce^{cx_n} + 2x_n h \quad (43)$$

$$\begin{aligned}
k_2 &= hf(x_n + mh, y(x_n) + mk_1) \\
&= hf(x_n + mh, y(x_n) + mhce^{cx_n} + 2mx_n h) \\
&= hf(x_n + mh, x_n^2 + e^{cx_n} + mhce^{cx_n} + 2mx_n h) \\
&= hf(x_n + mh, x_n^2 + (1 + mhc)e^{cx_n} + 2mx_n h) \\
&= hc[x_n^2 + (1 + mhc)e^{cx_n} + 2mx_n h - (x_n + mh)^2] + 2h(x_n + mh) \\
&= hc(1 + mhc)e^{cx_n} - m^2ch^3 + 2x_n h + 2mh^2 \quad (44)
\end{aligned}$$

$$\begin{aligned}
T_{n+1} &= [y(x_n) + ak_1 + bk_2] - y(x_{n+1}) \\
&= [x_n^2 + e^{cx_n} + ahce^{cx_n} + a2x_n h + bhc(1 + mhc)e^{cx_n} \\
&\quad - bm^2ch^3 + b2x_n h + b2mh^2] - [(x_n + h)^2 + e^{c(x_n+h)}]
\end{aligned}$$

$$\begin{aligned}
&= x_n^2 + e^{cx_n} [1 + (a+b)hc + bmh^2c^2] + (a+b)2x_n h - bmmch^3 \\
&\quad + 2bmh^2 - x_n^2 - 2x_n h - h^2 - e^{ch} e^{cx_n} \\
&= \left(1 + hc + \frac{h^2c^2}{2!} - e^{ch}\right) e^{cx_n} - \frac{mch^3}{2}
\end{aligned}$$

So:

$$e_{n+1} = \left(1 + hc + \frac{h^2c^2}{2!}\right) e_n + T_{n+1}. \quad (45)$$

Note:

In the problem of (29), if the initial value $y(0) = 0$, the global truncation error formula (45) also holds for a Runge-Kutta order 2 method except

$$T_{n+1} = -\frac{mch^3}{2} \quad (46)$$

Global Truncation Error e_n for Runge-Kutta Methods of Order Three

Theorem 3. The global truncation error e_n for a Runge-Kutta method of order three for problem (29) when $y_0 = 1$ is

$$\begin{aligned}
e_{n+1} &= \left(1 + hc + \frac{h^2c^2}{2!} + \frac{h^3c^3}{3!}\right) e_n \\
&\quad + \left(1 + hc + \frac{h^2c^2}{2!} + \frac{h^3c^3}{3!} - e^{ch}\right) e^{cx_n} - \frac{mc^2h^4}{6}
\end{aligned} \quad (47)$$

where h is the stepsize, if N is the total number of iterations of the method, then $hN = 1$

and $x_n = \frac{n}{N} = nh$.

Proof:

From (35) we have

$$k_1 = hf(x_n, y_n) = hc(y_n - x_n^2) + 2x_n h$$

From (41) we have

$$k_2 = hf(x_n + mh, y_n + mk_1)$$

$$\begin{aligned}
&= (hc + mh^2c^2)(y_n - x_n^2) - m^2ch^3 + 2x_nh + 2mh^2 \\
k_3 &= hf(x_n + nh, y_n + (n-r)k_1 + rk_2) \\
&= hf(x_n + nh, y_n + (n-r)hc(y_n - x_n^2) + (n-r)2x_nh \\
&\quad + r(hc + mh^2c^2)(y_n - x_n^2) - rm^2ch^3 + r2x_nh + r2mh^2) \\
&= hf(x_n + nh, y_n + (y_n - x_n^2)(nhc + rmh^2c^2) + n2x_nh \\
&\quad - rm^2ch^3 + r2mh^2) \\
&= hc[y_n + (y_n - x_n^2)(nhc + rmh^2c^2) + n2x_nh - rm^2ch^3 + r2mh^2 \\
&\quad - (x_n + nh)^2] + 2h(x_n + nh) \\
&= hc(1 + nhc + rmh^2c^2)(y_n - x_n^2) - rm^2c^2h^4 \\
&\quad + h^3c(2rm - n^2) + 2x_nh + 2nh^2 \tag{48}
\end{aligned}$$

$$\begin{aligned}
e_{n+1} &= y_{n+1} - y(x_{n+1}) = [y_n + ak_1 + bk_2 + c_1k_3] - [(x_n + h)^2 + e^{cx_n+h}] \\
&= y_n + ahc(y_n - x_n^2) + a2x_nh + b(hc + mh^2c^2)(y_n - x_n^2) \\
&\quad - bm^2ch^3 + b2x_nh + b2mh^2 + c_1hc(1 + nhc + rmh^2c^2)(y_n - x_n^2) \\
&\quad - c_1rm^2c^2h^4 + c_1h^3c(2rm - n^2) + c_12x_nh + c_12nh^2 \\
&\quad - x_n^2 - 2x_nh - h^2 - e^{ch}e^{cx_n} \\
&= (y_n - x_n^2)[1 + (a + b + c_1)hc + (bm + c_1n)h^2c^2 + c_1rmh^3c^3] \\
&\quad + (a + b + c_1 - 1)2x_nh + (2bm + 2c_1n - 1)h^2 - c_1rm^2c^2h^4 \\
&\quad + h^3c(2c_1rm - c_1n^2 - bm^2) - e^{ch}e^{cx_n} \\
&= (1 + hc + \frac{1}{2}h^2c^2 + \frac{1}{6}h^3c^3)(y_n - x_n^2) - \frac{mc^2h^4}{6} - e^{ch}e^{cx_n}
\end{aligned}$$

From (36) we have:

$$\begin{aligned}
e_{n+1} &= (1 + hc + \frac{h^2c^2}{2!} + \frac{h^3c^3}{3!})(e_n + e^{cx_n}) - \frac{mc^2h^4}{6} - e^{ch}e^{cx_n} \\
&= (1 + hc + \frac{h^2c^2}{2!} + \frac{h^3c^3}{3!})e_n + (1 + hc + \frac{h^2c^2}{2!} + \frac{h^3c^3}{3!} - e^{ch})e^{cx_n} - \frac{mc^2h^4}{6}
\end{aligned}$$

end of proof.

The local truncation error T_{n+1} for a Runge-Kutta method of order three for the

problem (29) is

$$T_{n+1} = \left(1 + hc + \frac{h^2 c^2}{2!} + \frac{h^3 c^3}{3!} - e^{ch} \right) e^{cx_n} - \frac{mc^2 h^4}{6} \quad (49)$$

The proof of (47) is the following:

From (43) and (44) we have

$$k_1 = hf(x_n, y(x_n)) = hce^{cx_n} + 2x_n h$$

$$\begin{aligned} k_2 &= hf(x_n + mh, y(x_n) + mk_1) \\ &= hc(1 + mhc) e^{cx_n} - m^2 ch^3 + 2x_n h + 2mh^2 \end{aligned}$$

$$\begin{aligned} k_3 &= hf(x_n + nh, y(x_n) + (n-r)k_1 + rk_2) \\ &= hf(x_n + nh, x_n^2 + e^{cx_n} + (n-r)hce^{cx_n} + (n-r)2x_n h \\ &\quad + r(hc + mh^2 c^2) e^{cx_n} - rm^2 ch^3 + r2x_n h + r2mh^2) \\ &= hf(x_n + nh, x_n^2 + e^{cx_n} (1 + nhc + rmh^2 c^2) + n2x_n h \\ &\quad - rm^2 ch^3 + r2mh^2) \\ &= hc[x_n^2 + e^{cx_n} (nhc + rmh^2 c^2) + n2x_n h - rm^2 ch^3 + r2mh^2 \\ &\quad - (x_n + nh)^2] + 2h(x_n + nh) \\ &= hc(1 + nhc + rmh^2 c^2) e^{cx_n} - rm^2 c^2 h^4 \\ &\quad + h^3 c(2rm - n^2) + 2x_n h + 2nh^2 \end{aligned}$$

$$\begin{aligned} T_{n+1} &= [y(x_n) + ak_1 + bk_2 + c_1 k_3] - y(x_{n+1}) \\ &= x_n^2 + e^{cx_n} + ahce^{cx_n} + a2x_n h + b(hc + mh^2 c^2) e^{cx_n} \\ &\quad - bm^2 ch^3 + b2x_n h + b2mh^2 + c_1 hc(1 + nhc + rmh^2 c^2) e^{cx_n} \\ &\quad - c_1 rm^2 c^2 h^4 + c_1 h^3 c(2rm - n^2) + c_1 2x_n h + c_1 2nh^2 \\ &\quad - x_n^2 - 2x_n h - h^2 - e^{ch} e^{cx_n} \\ &= e^{cx_n} [1 + (a + b + c_1) hc + (bm + c_1 n) h^2 c^2 + c_1 rmh^3 c^3] \\ &\quad + (a + b + c_1 - 1) 2x_n h + (2bm + 2c_1 n - 1) h^2 - c_1 rm^2 c^2 h^4 \\ &\quad + h^3 c(2c_1 rm - c_1 n^2 - bm^2) - e^{ch} e^{cx_n} \\ &= \left(1 + hc + \frac{1}{2} h^2 c^2 + \frac{1}{6} h^3 c^3 \right) e^{cx_n} - \frac{mc^2 h^4}{6} - e^{ch} e^{cx_n} \end{aligned}$$

So:

$$e_{n+1} = \left(1 + hc + \frac{h^2 c^2}{2!} + \frac{h^3 c^3}{3!} \right) e_n + T_{n+1}. \quad (50)$$

Note:

In the problem of (29), if the initial value $y(0) = 0$, the global truncation error formula (45) also holds for a Runge-Kutta order 3 method except

$$T_{n+1} = - \frac{mc^2 h^4}{6} \quad (51)$$

Global Truncation Error e_n for Runge-Kutta Methods of Order Four

Theorem 4. The global truncation error e_n for a Runge-Kutta method of order four for problem (29) when $y(0) = 1$ is

$$e_{n+1} = \left(1 + hc + \frac{h^2 c^2}{2!} + \frac{h^3 c^3}{3!} + \frac{h^4 c^4}{4!} \right) e_n + \left(1 + hc + \frac{h^2 c^2}{2!} + \frac{h^3 c^3}{3!} + \frac{h^4 c^4}{4!} - e^{ch} \right) e^{cx_n} - \frac{mc^3 h^5}{4!} \quad (52)$$

where h is the stepsize, if N is the total number of iterations of the method, then $hN = 1$

and $x_n = \frac{n}{N} = nh$.

Proof:

From (35) we have

$$k_1 = hf(x_n, y_n) = hc(y_n - x_n^2) + 2x_n h$$

From (41) we have

$$\begin{aligned} k_2 &= hf(x_n + mh, y_n + mk_1) \\ &= (hc + mh^2 c^2)(y_n - x_n^2) - m^2 ch^3 + 2x_n h + 2mh^2 \end{aligned}$$

$$\begin{aligned} k_3 &= hf(x_n + nh, y_n + (n-r)k_1 + rk_2) \\ &= hc(1 + nhc + rmh^2 c^2)(y_n - x_n^2) - rm^2 c^2 h^4 \\ &\quad + h^3 c(2rm - n^2) + 2x_n h + 2nh^2 \end{aligned}$$

$$\begin{aligned}
k_4 &= hf(x_n + ph, y_n + (p - s - t)k_1 + sk_2 + tk_3) \\
&= hf(x_n + ph, y_n + (p - s - t)hc(y_n - x_n^2) + (p - s - t)2x_n h \\
&\quad + s(hc + mh^2c^2)(y_n - x_n^2) - sm^2ch^3 + s2x_n h + s2mh^2 \\
&\quad + thc(1 + nhc + rmh^2c^2)(y_n - x_n^2) - trm^2c^2h^4 + th^3c(2rm - n^2) \\
&\quad\quad\quad + t2x_n h + t2nh^2) \\
&= hf(x_n + ph, y_n + (y_n - x_n^2)[(p - s - t)hc + s(hc + mh^2c^2) + \\
&\quad thc(1 + nhc + rmh^2c^2)] + 2x_n h[p - s - t + s + t] + [2trm - tn^2 - \\
&\quad sm^2]h^3c + (t2n + 2sm)h^2 - trm^2c^2h^4) \\
&= hf(x_n + ph, y_n + (y_n - x_n^2)[phc + (sm + tn)h^2c^2 + trmh^3c^3] + 2x_n h p \\
&\quad + [2trm - tn^2 - sm^2]h^3c + (t2n + 2sm)h^2 - trm^2c^2h^4) \\
&= hc\{y_n + (y_n - x_n^2)[phc + (sm + tn)h^2c^2 + trmh^3c^3] + 2x_n h p \\
&\quad + [2trm - tn^2 - sm^2]h^3c + (t2n + 2sm)h^2 - trm^2c^2h^4 - (x_n + hp)^2\} \\
&\quad + 2h(x_n + ph) \\
&= hc[1 + phc + (sm + tn)h^2c^2 + trmh^3c^3](y_n - x_n^2) \\
&\quad + c^2h^4(2trm - tn^2 - sm^2 + h^3c(t2n + 2sm - p^2) \\
&\quad\quad\quad - trm^2c^3h^5 + 2x_n h + 2ph^2)
\end{aligned}$$

$$\begin{aligned}
e_{n+1} &= y_{n+1} - y(x_{n+1}) \\
&= [y_n + ak_1 + bk_2 + c_1k_3 + dk_4] - [(x_n + h)^2 + e^{c(x_n \cdot h)}] \\
&= y_n + ahc(y_n - x_n^2) + a2x_n h + b(hc + mh^2c^2)(y_n - x_n^2) - bm^2ch^3 \\
&\quad + b2x_n h + b2mh^2 + c_1hc(1 + nhc + rmh^2c^2)(y_n - x_n^2) - c_1rm^2c^2h^4 \\
&\quad + c_1h^3c(2rm - n^2) + c_12x_n h + c_12nh^2 + dhc[1 + phc \\
&\quad + (sm + tn)h^2c^2 + trmh^3c^3](y_n - x_n^2) + dc^2h^4(2trm - tn^2 - sm^2) \\
&\quad + dh^3c(t2n + 2sm - p^2) - dtrm^2c^3h^5 + d2x_n h + d2ph^2 \\
&\quad - x_n^2 - 2x_n h - h^2 - e^{ch}e^{cx_n} \\
&= (y_n - x_n^2)\{1 + (a + b + c_1 + d)hc + (bm + c_1n + dp)h^2c^2 + [c_1rm \\
&\quad + d(sm + tn)]h^3c^3 + dtrmh^4c^4\} + (a + b + c_1 + d - 1)2x_n h \\
&\quad + (2c_1rm - c_1n^2 - bm^2 + dt2n + d2sm - dp^2)ch^3
\end{aligned}$$

$$\begin{aligned}
& + (b2m + c_1 2n + d2p - 1)h^2 + c^2 h^4 (2dtrm - dt n^2 - dsm^2 - c_1 r m^2) \\
& - dtrm^2 c^3 h^5 - e^{ch} e^{cx_n} \\
& = (y_n - x_n^2) \left(1 + hc + \frac{1}{2} h^2 c^2 + \frac{1}{6} h^3 c^3 + \frac{h^4 c^4}{24} \right) - \frac{mc^3 h^5}{4!} - e^{ch} e^{cx_n}
\end{aligned}$$

From (36) we have:

$$\begin{aligned}
e_{n-1} & = \left(1 + hc + \frac{h^2 c^2}{2!} + \frac{h^3 c^3}{3!} + \frac{h^4 c^4}{24} \right) (e_n + e^{cx_n}) - \frac{mc^3 h^5}{4!} - e^{ch} e^{cx_n} \\
& = \left(1 + hc + \frac{h^2 c^2}{2!} + \frac{h^3 c^3}{3!} + \frac{h^4 c^4}{4!} \right) e_n + \\
& \quad \left(1 + hc + \frac{h^2 c^2}{2!} + \frac{h^3 c^3}{3!} + \frac{h^4 c^4}{4!} - e^{ch} \right) e^{cx_n} - \frac{mc^3 h^5}{4!}
\end{aligned}$$

end of proof.

The local truncation error T_{n-1} for a Runge-Kutta order 4 method for the problem (29) is

$$T_{n-1} = \left(1 + hc + \frac{h^2 c^2}{2!} + \frac{h^3 c^3}{3!} + \frac{h^4 c^4}{4!} - e^{ch} \right) e^{cx_n} - \frac{mc^3 h^5}{4!} \quad (53)$$

The proof of (53) is the following:

From (43) and (44) we have

$$k_1 = hf(x_n, y(x_n)) = hce^{cx_n} + 2x_n h$$

$$\begin{aligned}
k_2 & = hf(x_n + mh, y(x_n) + mk_1) \\
& = hc(1 + mhc) e^{cx_n} - m^2 ch^3 + 2x_n h + 2mh^2
\end{aligned}$$

$$\begin{aligned}
k_3 & = hf(x_n + nh, y(x_n) + (n-r)k_1 + rk_2) \\
& = hc(1 + nhc + rmh^2 c^2) e^{cx_n} - rm^2 c^2 h^4 \\
& \quad + h^3 c(2rm - n^2) + 2x_n h + 2nh^2
\end{aligned}$$

$$\begin{aligned}
k_4 & = hf(x_n + ph, y(x_n) + (p-s-t)k_1 + sk_2 + tk_3) \\
& = hf(x_n + ph, x_n^2 + e^{cx_n} + (p-s-t)hce^{cx_n} + (p-s-t)2x_n h \\
& \quad + s(hc + mh^2 c^2) e^{cx_n} - sm^2 ch^3 + s2x_n h + s2mh^2 \\
& \quad + thc(1 + nhc + rmh^2 c^2) e^{cx_n} - trm^2 c^2 h^4 + th^3 c(2rm - n^2)
\end{aligned}$$

$$\begin{aligned}
& + t2x_n h + t2nh^2) \\
= & hf(x_n + ph, x_n^2 + e^{\alpha x_n} [1 + (p - s - t)hc + s(hc + mh^2c^2) + \\
& thc(1 + nhc + rmh^2c^2)] + 2x_n h[p - s - t + s + t] + [2trm - tn^2 - \\
& sm^2]h^3c + (t2n + 2sm)h^2 - trm^2c^2h^4) \\
= & hf(x_n + ph, x_n + e^{\alpha x_n} [1 + phc + (sm + tn)h^2c^2 + trmh^3c^3] + 2x_n h p \\
& + [2trm - tn^2 - sm^2]h^3c + (t2n + 2sm)h^2 - trm^2c^2h^4) \\
= & hc\{ x_n + e^{\alpha x_n} [phc + (sm + tn)h^2c^2 + trmh^3c^3] + 2x_n h p \\
& + [2trm - tn^2 - sm^2]h^3c + (t2n + 2sm)h^2 - trm^2c^2h^4 - (x_n + hp)^2 \} \\
& + 2h(x_n + ph) \\
= & hc[1 + phc + (sm + tn)h^2c^2 + trmh^3c^3]e^{\alpha x_n} + c^2h^4(2trm - tn^2 - sm^2) \\
& + h^3c(t2n + 2sm - p^2) - trm^2c^3h^5 + 2x_n h + 2ph^2 \\
I_{n-1} = & [y(x_n) + ak_1 + bk_2 + c_1k_3 + dk_4] - [(x_n + h)^2 + e^{\alpha(x_n+h)}] \\
= & x_n + e^{\alpha x_n} + ahce^{\alpha x_n} + a2x_n h + b(hc + mh^2c^2)e^{\alpha x_n} - bm^2ch^3 \\
& + b2x_n h + b2mh^2 + c_1hc(1 + nhc + rmh^2c^2)e^{\alpha x_n} - c_1rm^2c^2h^4 \\
& + c_1h^3c(2rm - n^2) + c_12x_n h + c_12nh^2 + dhc[1 + phc \\
& + (sm + tn)h^2c^2 + trmh^3c^3]e^{\alpha x_n} + dc^2h^4(2trm - tn^2 - sm^2) \\
& + dh^3c(t2n + 2sm - p^2) - dtrm^2c^3h^5 + d2x_n h + d2ph^2 \\
& - x_n^2 - 2x_n h - h^2 - e^{ch}e^{\alpha x_n} \\
= & e^{\alpha x_n} \{ 1 + (a + b + c_1 + d)hc + (bm + c_1n + dp)h^2c^2 + [c_1rm \\
& + d(sm + tn)]h^3c^3 + dtrmh^4c^4 \} + (a + b + c_1 + d - 1)2x_n h \\
& + (2c_1rm - c_1n^2 - bm^2 + dt2n + d2sm - dp^2)ch^3 \\
& + (b2m + c_12n + d2p - 1)h^2 + c^2h^4(2dtrm - dtn^2 - dsm^2 - c_1rm^2) \\
& - dtrm^2c^3h^5 - e^{ch}e^{\alpha x_n} \\
= & e^{\alpha x_n} (1 + hc + \frac{1}{2}h^2c^2 + \frac{1}{6}h^3c^3 + \frac{h^4c^4}{24}) - \frac{mc^3h^5}{4!} - e^{ch}e^{\alpha x_n}
\end{aligned}$$

So:

$$e_{n+1} = \left(1 + hc + \frac{1}{2}h^2c^2 + \frac{1}{6}h^3c^3 + \frac{h^4c^4}{24} \right) e_n + T_{n+1}. \quad (54)$$

Note:

In the problem of (29), if the initial value $y(0) = 0$, the global truncation error for formula (45) also holds for a Runge-Kutta order 4 method except

$$T_{n+1} = - \frac{mc^3h^5}{4!} \quad (55)$$

Global Error Analysis for Chapter III

If $c < 0$ and $|c|$ is large enough, e^{cn} can be considered as almost zero after some iterations. So, from formulas (34), (40), (47), (52) we can deduce that the global error for problem (29) when $y_0 = 0$ is approximately the same as when $y_0 = 1$.

For simplicity we only consider the initial value problem (29) and assume $y_0 = 0$.

a. Euler method:

According to (32) and (38) the global error is

$$e_n = \frac{-h^2[(1+hc)^n - 1]}{hc} = \frac{-h[(1+hc)^n - 1]}{c}.$$

If $|1+hc| < 1$ and n is large enough we can say $(1+hc)^n$ is almost zero. So

$$e_n = \frac{h}{c} \quad (56)$$

b. Runge-Kutta method of order two:

According to (32) and (45) the global error is

$$e_n = \frac{-\frac{mch^3}{2} \left[\left(1 + hc + \frac{h^2c^2}{2} \right)^n - 1 \right]}{hc + \frac{h^2c^2}{2}} = \frac{-\frac{mh^2}{2} \left[\left(1 + hc + \frac{h^2c^2}{2} \right)^n - 1 \right]}{1 + \frac{hc}{2}}.$$

If $\left|1 + hc + \frac{h^2 c^2}{2}\right| < 1$ and n is large enough we can say $\left(1 + hc + \frac{h^2 c^2}{2}\right)^n$ is almost zero. So

$$e_n = \frac{mh^2}{2 + hc} \quad (57)$$

c. Runge-Kutta method of order three:

According to (32) and (50) the global error is

$$e_n = \frac{-\frac{mc^2 h^4}{6} \left[\left(1 + hc + \frac{h^2 c^2}{2} + \frac{h^3 c^3}{6}\right)^n - 1 \right]}{hc + \frac{h^2 c^2}{2} + \frac{h^3 c^3}{6}}$$

$$= \frac{-\frac{mch^3}{6} \left[\left(1 + hc + \frac{h^2 c^2}{2} + \frac{h^3 c^3}{6}\right)^n - 1 \right]}{1 + \frac{hc}{2} + \frac{h^2 c^2}{6}}$$

If $\left|1 + hc + \frac{h^2 c^2}{2} + \frac{h^3 c^3}{6}\right| < 1$ and n is large enough we can say $\left(1 + hc + \frac{h^2 c^2}{2} + \frac{h^3 c^3}{6}\right)^n$ is

almost zero. So

$$e_n = \frac{mch^3}{6 + 3hc + h^2 c^2} \quad (58)$$

d. Runge-Kutta method of order four:

According to (32) and (54) the global error is

$$e_n = \frac{-\frac{mc^3 h^5}{24} \left[\left(1 + hc + \frac{h^2 c^2}{2} + \frac{h^3 c^3}{6} + \frac{h^4 c^4}{24}\right)^n - 1 \right]}{hc + \frac{h^2 c^2}{2} + \frac{h^3 c^3}{6} + \frac{h^4 c^4}{24}}$$

$$= \frac{-\frac{mc^2 h^4}{24} \left[\left(1 + hc + \frac{h^2 c^2}{2} + \frac{h^3 c^3}{6} + \frac{h^4 c^4}{24} \right)^n - 1 \right]}{1 + \frac{hc}{2} + \frac{h^2 c^2}{6} + \frac{h^3 c^3}{24}}$$

If $\left| 1 + hc + \frac{h^2 c^2}{2} + \frac{h^3 c^3}{6} + \frac{h^4 c^4}{24} \right| < 1$ and n is large enough we can say

$\left(1 + hc + \frac{h^2 c^2}{2} + \frac{h^3 c^3}{6} + \frac{h^4 c^4}{24} \right)^n$ is almost zero. So

$$e_n = \frac{mc^2 h^4}{24 + 12hc + 4h^2 c^2 + h^3 c^3} \quad (59)$$

Using formula (56), (57), (58) and (59) we can get results which are listed in Table V, Table VI, and Table VIII of Chapter III.

"Parallel" Parabolas Property

In problem (29), using a Runge-Kutta method of order 4 if we require

$$y_1 = y_0 + x_1^2$$

we can get

$$y_n = y_0 + x_n^2, \quad y_0 = \frac{1}{\Delta} \frac{mc^3 h^5}{4!}$$

where $\Delta = hc + \frac{1}{2} h^2 c^2 + \frac{1}{3!} h^3 c^3 + \frac{1}{4!} h^4 c^4$ and $\frac{mc^3 h^5}{4!}$ is the local truncation error when

$y(0) = 0$. It can be shown in Figure 16.

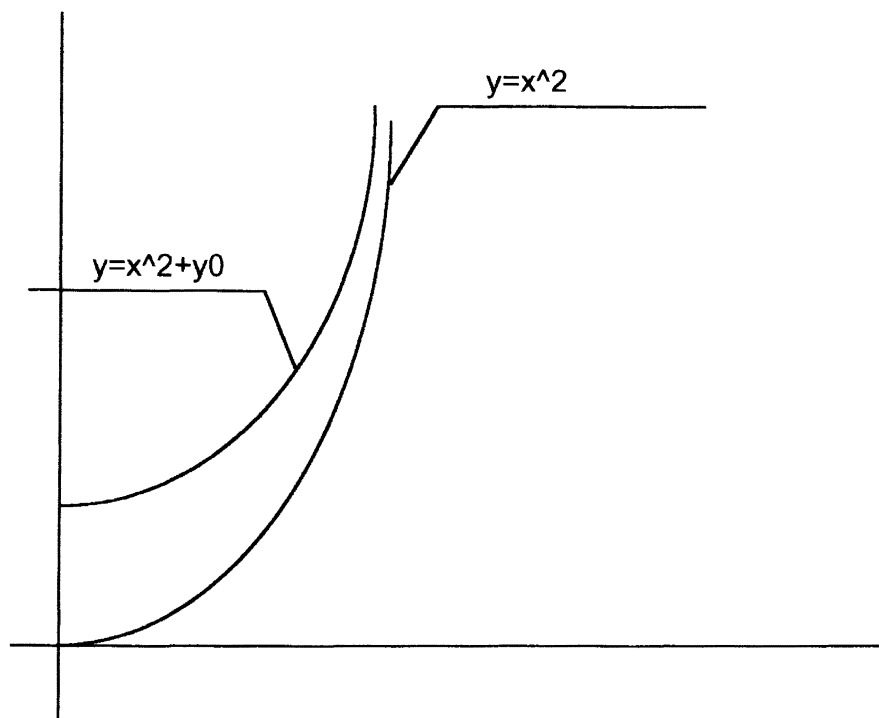


Figure 16. "Parallel" Parabolas

Proof:

The exact solution of problem (29) is

$$y = y_0 e^{cx} + x^2 \quad (60)$$

Let $x_n = nh$, $n = 0, 1, 2, 3, \dots$, then $x_1 = h$.

$$y(x_1) = y(h) = y_0 e^{ch} + h^2$$

from equation (53), we have

$$\begin{aligned} e_1 &= y_1 - y(x_1) \\ &= (y_0 - x_0^2) \left(1 + hc + \frac{1}{2} h^2 c^2 + \frac{1}{3!} h^3 c^3 + \frac{1}{4!} h^4 c^4 \right) - \frac{mc^3 h^5}{4!} - y_0 e^{ch} e^{cx_0} \\ &= y_0 (1 + \Delta) - \frac{mc^3 h^5}{4!} - y_0 e^{ch} \end{aligned}$$

So,

$$\begin{aligned}
y_1 &= y(x_1) + e_1 \\
&= y_0 e^{ch} + h^2 + y_0(1 + \Delta) - \frac{mc^3 h^2}{4!} - y_0 e^{ch}
\end{aligned} \tag{61}$$

Because

$$y_1 = y_0 + x_1^2 = y_0 + h^2$$

From (62) we have

$$\begin{aligned}
y_0 + h^2 &= h^2 + y_0(1 + \Delta) - \frac{mc^3 h^2}{4!} \\
y_0 &= \frac{1}{\Delta} \frac{mc^3 h^2}{4!}
\end{aligned} \tag{62}$$

If $n = 2$, we have:

$$\begin{aligned}
e_2 &= y_2 - y(x_2) \\
&= (y_1 - x_1^2)(1 + \Delta) - \frac{mc^3 h^3}{4!} - y_0 e^{ch} e^{ch} \\
&= (y_0 + x_1^2 - x_1^2)(1 + \Delta) - \frac{mc^3 h^3}{4!} - y_0 e^{ch} e^{ch} \\
&= y_0(1 + \Delta) - \frac{mc^3 h^3}{4!} - y_0 e^{2ch}
\end{aligned}$$

So,

$$\begin{aligned}
y_2 &= y(x_2) + e_2 \\
&= (y_0 e^{2ch} + 4h^2) + y_0(1 + \Delta) - \frac{mc^3 h^3}{4!} - y_0 e^{2ch} \\
&= 4h^2 + y_0 + y_0 \Delta - \frac{mc^3 h^3}{4!} \\
&= 4h^2 + y_0 = x_2^2 + y_0
\end{aligned}$$

It is true.

Assume $n = N$ is true.

For $n = N + 1$,

$$e_{N+1} = y_{N+1} - y(x_{N+1})$$

$$\begin{aligned}
&= (y_N - x_N^2)(1 + \Delta) - \frac{mc^3 h^5}{4!} - y_0 e^{ch} e^{c^2 v} \\
&= y_0(1 + \Delta) - \frac{mc^3 h^5}{4!} - y_0 e^{ch(N+1)}
\end{aligned}$$

So,

$$\begin{aligned}
y_{N+1} &= y(x_{N+1}) + e_{N+1} \\
&= (y_0 e^{c(N+1)h} + x_{n+1}^2) + y_0(1 + \Delta) - \frac{mc^3 h^5}{4!} - y_0 e^{ch(N+1)} \\
&= x_{n+1}^2 + y_0 + y_0 \Delta - \frac{mc^3 h^5}{4!} \\
&= x_{n+1}^2 + y_0
\end{aligned}$$

It is true.

End of proof.

CHAPTER V

CONCLUSIONS AND SUGGESTIONS

For the ordinary differential equation

$$y' = nx^{n-1} - C(y - x^n), \quad y(0) = 0, \quad (n = 1, 2) \quad (63)$$

using Runge-Kutta methods of order two or greater the global error is a constant after some iterations. The exact solution of this problem is $y(x) = x^n$, and the Adams methods of order two or greater give the exact solution. The purpose of this thesis is to try to find out the reasons why Runge-Kutta methods do not solve this polynomial problem exactly. Like Adams methods, Runge-Kutta methods are based on the Taylor series, and Taylor series are polynomials. Why are the global truncation errors of Runge-Kutta methods of solving this problem constant and nonzero after some iterations?

In the previous chapters, attempts were made to check and compare the results of problem (63) when we change some parameters: the coefficient c , the stepsize h , the initial value y_0 . The results showed that for problem (63), if $n = 2$, and $y_0 = 0$ whatever we choose for the different values of the stepsize h and the coefficient c we could not get the exact solutions by using the Runge-Kutta methods of order two or above although it is a polynomial problem $y(x) = x^2$. The results also showed that no matter whether the initial value y_0 is equal to 0 or 1, after some iterations the numerical solutions of Runge-Kutta problems were approximately the same. Furthermore the results showed that no matter what values of the parameters you choose, the global truncation error of Runge-Kutta methods is a constant.

Using the posterior estimate method we got the global error estimate formula

$$e_{n+1} = e_n \sum_{i=0}^{j-1} \frac{h^i c^i}{i!} + \left(\sum_{i=0}^{j-1} \frac{h^i c^i}{i!} - e^{ch} \right) e^{cx_n} - \frac{mc^{j-1} h^{j-1}}{(j-1)!}$$

where j is the order of Runge-Kutta methods and $0! = 1$. In Chapter IV, we only justified the formula for $j = 1, 2, 3, 4$; can we prove the above formula is also true for j greater than 4?

Another area of further work is can we find some inner relationship between the constant global error for some specific problems using Runge-Kutta methods and the problem itself, that is, for a specific problem do you know the global error of the problem is a constant or not after some iterations before you actually solve it?

The FORTRAN code in this thesis is used only to solve the problem (63). For further study on other problems rather than (63), the user can change some statements in the subroutine DISPLAY and functions G and FUNC.

SELECTED REFERENCES

1. Abramowitz, M. and Stegun, I. A., Handbook of Mathematical Functions, Dover Publications, 1965.
2. Barton, D., Willers, I. M. and Zahar, I. V. M. , "Taylor series methods for ordinary differential equations - an evaluation," Mathematical Software, J. R. Rice, Academic Press, 1971, pp. 369-390.
3. Bashforth, F. and Adams, J. C., Ordinary Differential Equations, Third ed., John Wiley and Sons. 1978.
4. Bulirsch, R. and Stoer, J., "Numerical treatment of ordinary differential equations by extrapolation methods," Numer. Math. 8, 1966, pp.1-13.
5. Butcher, J.C. "Coefficients for the study of Runge-Kutta integration processes," J. Austral. Math. Soc. Vol. 3, (1963) , pp. 185 - 201.
6. Butcher, J. C. , "On the attainable order of Runge-Kutta processes," Math.Comp. 19, 1965, pp.408-417.
7. Ceschino, F. , and Kuntzmann, J. Numerical solution of initial value problems, Prentice-Hall, Inc., Englewood Cliffs, N.Y. , 1966
8. Coddington, E. A.. and Levinson, N. Theory of Ordinary Differential Equations, McGraw-Hill, 1955.
9. Collatz, L., The Numerical Treatment of Differential Equations , Third ed., Spriger, Berlin, 1960.
10. Crane, R. L., and R. W. Klopfenstein, "A predictor corrector algorithm with increased range of absolute stability, " J. Assoc. Comput. Mach. 12, 1965 pp. 227--241.
11. Croliss, G. and Chang, Y. F. "Solving ordinary differential equations using Taylor Series," A.C.M. Trans. Math. Software 8, 1982, pp. 114-144.
12. Deng, Jian-Zhong, Computation Methods , Xi-An Jiao-Tong University Publish Press, P. R. of China., 1985.

13. England, R., "Error estimates for Runge-Kutta type solutions to systems of ordinary differential equations," Comput. J. 12, 1969, pp. 166-170.
14. Fox, L. and Mayers, D. F., "On the numerical solution of implicit ordinary differential equations," I. M. A. J. Numer. Anal. 1, 1981, pp. 377-401.
15. Fox, P. A., "Desub: integration of a First-Order System of ordinary differential equations," Math. Software, J. R. Rice, Academic Press, 1969, pp. 477-507.
16. Gear, C. W., Numerical Initial Value Problems in Ordinary Differential Equations, Prentice-Hall, Inc., Englewood Cliffs, N.Y., 1971.
17. Gear, C. W., "The automatic integration of ordinary differential equations," Comm. A. C. M. 14, 1971a, pp. 176-179.
18. Gear, C. W., "Algorithm 407, DIFSUB for solution of ordinary differential equations," Comm. A. C. M. 14, 1971b, pp. 185-196.
19. Gill, S., "A process for the step-by-step integration of differential equation in an automatic digital computing machine," Proc. Cambridge Philos. Soc. 47, 1951, pp. 95-108.
20. Gragg, W. B., "On extrapolation algorithms for ordinary initial value problems," SIAM J. Numer. Anal. 2, 1965, pp.384-403.
21. Hamming, R. W., "Stable predictor-corrector methods for ordinary differential equations," J. Assoc. Comp. Mach. 6, 1959, pp.37-47.
22. Henrici, P., Discrete Variable Methods in Ordinary Differential Equations, John and Wiley and Sons, 1962.
23. Hull, T. E. and Creemer, A. L., "Efficiency of predictor-corrector procedures," J. Assoc. Comp. Mach. 10, 1963, pp. 291-301.
24. Hull, T. E., Enright, W. H., Fellen, B. M., and Sedgwick, A. E., "Comparing numerical methods for ordinary differential equations," SIAM J. Numer. Anal. 9, 1972, pp. 603-637.
25. Hull, T. E. and Jonson, R. L., "Optimum Runge-Kutta methods," Math. Comp. 18, pp. 306-310, 1964.
26. Klopfenstein, R. W. and R. S. Millman, "Numerical stability of a one evaluation predictor-corrector algorithm for numerical solution of ordinary differential equations," Math. Comp. 22, 1968, pp. 557-564.

27. Lambert, J. D., Computational Methods in Ordinary Differential Equations, John Wiley and Sons, 1973.
28. Lotkin, M., "On the accuracy of Runge-Kutta's method," M. T. A. C. 5, 1951, pp. 128-132.
29. Merson, R. H., An operational method for the study of integration processes, Weapons Research Establishment, Salisbury, South Australia, 1957.
30. Miller, J. J. H., "On the location of zeros of certain classes of polynomials with applications to numerical analysis," J. Inst. Math. Appl. 8, 1971, pp. 397-406.
31. Murphy, C. P. and Evans, D. J., "A flexible variable order extrapolation technique for solving non-stiff ordinary differential equation," Int. J. Comp. Math. 10, 1981, pp. 63-75.
32. Murphy, G. M., Ordinary Differential Equations and Their Solutions, Van Nostrand-Reinhold, 1960.
33. Neville, E. H., "Iterative interpolation," J. Ind. Math. Soc. 20, 1934, pp. 7-120.
34. Nystrom, E. J., "Ueber die numerische integration von differentialgleichungen," Acta Soc. Sci. Fenn. 50,13, pp. 1-55, 1925.
35. Rosser, J. B., "A Runge-Kutta for all seasons," SIAM Rev. 9, 1967, pp. 417-452.
36. Scraton, R. E. "Estimation of the truncation error in Runge-Kutta and allied process," Comput. J. Vol. 7, (1964), p. 460.
37. Shampine, L. F. "What everyone solving differential equations should know," Computational Techniques for Ordinary Differential Equations, ed. I. Gladwell and D. K. Sayers, Academic Press, 1980, pp. 1-17.
38. Shampine, L. F. and Gordon, M. K., Computer Solution of Ordinary Differential Equations: The Initial Value Problem, W. H. Freeman and Company, 1975.
39. Shampine, L. F., Gordon, M. K. and Wisniewski, J. A., "Variable order Runge-Kutta codes," Computational Techniques for Ordinary Differential Equations, ed. I. Gladwell and D. K. Sayers, Academic Press, 1980, pp. 41- 46.
40. Shampine, L. F. and Watts, H. A., "Global error estimation for ordinary differential equations," A.C.M. Trans. Math. Software 2, 1976, pp. 172-186.

41. Shanks, E. B. "Solutions of differential equations by evaluations of functions," Notices Amer. Math. Soc. Vol. 10 (1963), p. 480.
42. Shintani, H., "Approximate computation of errors in numerical integration of ordinary differential equations by one-step methods," J. Sci. Hiroshima Univ. Ser. A-1 Math 29, 1965, pp. 97-120.
43. Shintani, H., "On a one-step method of order 4," J. Sci. Hiroshima Univ. Ser. A-1 Math. 30, 1966, pp.91-107.
44. Shintani, H., "Two-step processes by one-step methods of order 3 and order 4," J. Sci. Hiroshima Univ. Ser. A-1 Math. 30, 1966, pp.183-195.
45. Verner, J. H., "Explicit Runge-Kutta methods with estimates of the local truncation errors," SIAM J. Numer. Anal. 15, 1978, pp. 772-790.
46. Zonneveld, J. A., "Automatic Numerical Integration." Mathematics Centrum, Amsterdam , 1974.
47. Morel, H., "Evaluation de l'erreur sur un pas dans la methode de Runge-Kutta," C. R. Acad. Sci. Paris, 243, pp. 1999-2002.
48. Dahlquist, G., "Stability questions for some numerical methods for ordinary differential equations," Proc. Symposium for Applied Math. 15, pp. 147-158, 1963.

APPENDIX
PROGRAM FOR SINGLE-STEP METHODS
AND MULTI-STEP METHODS

PROGRAM ODESOLUTION

```

C--
C--
C-- This program is used to solve the differential equation of the
C-- form:
C--  $Y'=2X+C(2+X*X)-CY$ 
C--  $Y(11)=Y0$ 
C-- Where c is a constant and Y0 is a initial value of the ODE.
C--
C-- We use two numerical methods to solve the problem:
C-- 1. The single-step method.
C-- 2. The multi-step method.
C--
C-- Subroutines and Functions of the program:
C-- 1. AUTO
C-- This subroutine is used to control test data. It calls the
C-- subroutine ATSTEP to get the data when stepsize h changes.
C-- It calls the subroutine ATMETH to get the data when using
C-- different Runge-Kutta methods. The data are used for the
C-- thesis. All the parameters will be changed automatically.
C-- 2. MAINPA
C-- Unlike AUTO you can choose parameters by yourself using this
C-- subroutine. It calls the subroutine SELEF to let you select
C-- different initial value problem; the subroutine SELEC to
C-- let you select the coefficient C; the subroutine SELEY0 to
C-- let you select initial value Y0; the subroutine SELIni to
C-- let you select interval values; the subroutine SELEH to let
C-- you select the stepsize h. You must select all the parameters.
C-- 3. SELECT
C-- This subroutine like the subroutine MAINPA. It can call
C-- subroutine SELEF, SELEC, SELEY0, SELIni and SELEH.
C-- But it only changes one or none of above parameters.
C-- 4. SINGLE
C-- This subroutine is also a controlling routine. It calls one of
C-- subroutines EULER, RUNGE2, RUNGE3 and RUNGE4 to solve the
C-- initial value problem.
C-- 5. MULTY
C-- This subroutine is also a controlling routine. It calls one of
C-- subroutines ADAMS2, ADAMS3, ADAMS4, ADAMS5 and ADAMS6 to solve
C-- the initial value problem.
C-- 6. APPEAR
C-- It writes the title before a group of data. Such as numerical
C-- method used, the value of stepsize h and the initial value
C-- problem.
C-- 7. RK2PAR
C-- It is called by RUNGE2 for the purpose of selecting one
C-- specific method of Runge-Kutta method of order two.
C-- 8. RK3PAR
C-- It is called by RUNGE3 for the purpose of selecting one
C-- specific method of Runge-Kutta method of order three.
C-- 9. RK4PAR
C-- It is called by RUNGE4 for the purpose of selecting one
C-- specific method of Runge-Kutta method of order four.
C-- 10. RK2PP

```

C-- It is called by RUNGE2 for the purpose of selecting Runge-
 C-- Kutta method of order two automatically.
 C-- 11. RK3PP
 C-- It is called by RUNGE3 for the purpose of selecting Runge-
 C-- Kutta method of order three automatically.
 C-- 12. RK4PP
 C-- It is called by RUNGE4 for the purpose of selecting Runge-
 C-- Kutta method of order four automatically.
 C-- 13. Function G
 C-- It is the exact solution of the initial value problem.
 C-- 14. Function FUNC
 C-- It is the initial value problem.
 C--
 C--

```

    IMPLICIT REAL*8(A-H,O-Z)
    DOUBLE PRECISION I1,I2
    INTEGER NUMBER,II

    OPEN(UNIT=111,FILE='OUTPUT',STATUS='unknown')
    CALL AUTO
    CALL MAINPA(C,Y0,I1,I2,II,NN,H,ITER,NUM)
    IFLAG=0
    DO 200 I7=1,300
    PRINT *,'PLEASE CHOOSE THE FOLLOWING:'
    PRINT *,'1. SINGL-STEP METHOD'
    PRINT *,'2. MULTY-STEP METHOD'
    PRINT *,'3. QUIT'
    PRINT *,'Just press key 1 or 2 or 3 to selecte the menu'
    READ *,NUMBER
    GO TO (7770,7771,7772), NUMBER
7770 IF(IFLAG.EQ.1) THEN
    CALL SELECT(C,Y0,I1,I2,II,NN,H,ITER,NUM)
    ENDIF
    CALL SINGLE(C,Y0,I1,II,NN,H,ITER,NUM)
    IFLAG=1
    GOTO 200
7771 IF(IFLAG.EQ.1) THEN
    CALL SELECT(C,Y0,I1,I2,II,NN,H,ITER,NUM)
    ENDIF
    CALL MULTY(C,Y0,I1,II,NN,H,ITER,NUM)
    IFLAG=1
    GOTO 200
7772 PRINT *,'You have choose to quit the program'
    STOP
200  CONTINUE
    END
  
```

C--
 C*****
 C- Subroutine SINGLE
 C- It calls different single-step method to solve the
 C- initial value problem.
 C*****
 C--

```

SUBROUTINE SINGLE(C,Y0,I1,II,NN,H,ITER,NUM)
IMPLICIT REAL*8(A-H,O-Z)
DOUBLE PRECISION II
INTEGER NUMBER,II

DO 300 I=1,100
PRINT *
PRINT *,'Please choose from the following:'
PRINT *,'1. EULER method.'
PRINT *,'2. Runge Kutter second order method.'
PRINT *,'3. Runge Kutter third order method.'
PRINT *,'4. Runge Kutter fourth order method.'
PRINT *,'5. Return to the previous menu.'
PRINT *,'6. Quit'
PRINT *
PRINT *,'Which number do you choose?'
READ *,NUMBER
GO TO (7773,7774,7775,7776,7777,7778), NUMBER
7773 PRINT *,'Now you choose the first method.'
CALL EULER(C,Y0,II,II,NN,H,ITER,NUM)
GOTO 300
7774 PRINT *,'Now you choose the second method.'
CALL RUNGE2(C,Y0,II,II,NN,H,ITER,NUM,0,1)
GOTO 300
7775 PRINT *,'Now you choose the third method.'
CALL RUNGE3(C,Y0,II,II,NN,H,ITER,NUM,0,1)
GOTO 300
7776 PRINT *,'Now you choose the fourth method.'
CALL RUNGE4(C,Y0,II,II,NN,H,ITER,NUM,0,1)
GOTO 300
7777 RETURN
7778 PRINT *,'You choose quit.'
STOP
300 CONTINUE
END

C--
C*****
C-- Subroutine EULER
C*****
C--
SUBROUTINE EULER(C,Y0,II,II,NN,H,ITER,NUM)
C--
C-- EULER integrates the differential equation  $y'=f(x,y)$ . The formula
C-- of this numerical method is:
C--  $Y_{n+1} = Y_n + hf(X_n, Y_n)$ 
C-- Where  $Y_0=Y(II)$ . The interval is  $X=I1$  to  $X=I2$  and total  $ITER=$ 
C--  $(I2-I1)/h$  steps.
C--
IMPLICIT REAL*8(A-H,O-Z)
DOUBLE PRECISION II
INTEGER ITER,NUM,II

F(X,Y)=FUNC(C,X,Y,II,nn)
CALL APPEAR(1,H,NN,C,II,Y0)

```

```

X1=I1
Y1=Y0
DO 400 I=1,ITER
  Y2=Y1+h*F(X1,Y1)
  X1=X1+h
  IF(DABS(Y2-Y1).GT.99999999) THEN
    CALL MASSAG
    GO TO 411
  ENDIF
  IF(MOD(I.NUM).EQ.0. OR .I.LE.10. OR .I.EQ.ITER) THEN
    CALL PRINT(I,II,C,X1,Y2,NN,Y0)
  ENDIF
  Y1=Y2
400 CONTINUE
411 RETURN
END

C--
C*****
C-                               Subroutine RUNGE2
C*****
C--
  SUBROUTINE RUNGE2(C,Y0,I1,II,NN,H,ITER,NUM,IFLAG,IREP)
C--
C-- This is the second order Runge-Kutta method. The formula of this
C-- numerical method is:
C--  $Y_{n+1} = Y_n + aK_1 + bK_2$ 
C--  $K_1 = hf(X_n, Y_n)$ 
C--  $K_2 = hf(X_n + mh, Y_n + mK_1)$ 
C-- here  $a = (2m-1)/(2m)$ ,  $b = 1/(2m)$ 
C-- and  $Y_0 = Y(I1)$ . The interval is  $X = I1$  to  $X = I2$  and total ITER=
C--  $(I2-I1)/h$  steps.
C--
  IMPLICIT REAL*8(A-H,O-Z)
  DOUBLE PRECISION I1,M,K1,K2
  INTEGER ITER,NUM,II,I

  F(X,Y)=FUNC(C,X,Y,II,nn)
  IF(IFLAG.EQ.0) THEN
    CALL RK2PAR(A,B,M)
  ELSE
    CALL RK2PP(IFLAG,IREP,A,B,M)
  ENDIF
  CALL APPEAR(2,H,NN,C,II,Y0)
  X1=I1
  Y1=Y0
  DO 500 I=1,ITER
    K1=H*F(X1,Y1)
    K2=H*F(X1+M*H,Y1+M*K1)
    X1=X1+H
    Y2=Y1+A*K1+B*K2
    IF(DABS(Y2-Y1).GT.99999999) THEN
      CALL MASSAG
      GO TO 511
    ENDIF

```

```

        IF(MOD(I,NUM).EQ.0. OR .I.LE.5 OR.I.EQ.ITER) THEN
            CALL PRINT(I,II,C,X1,Y2,NN,Y0)
        ENDIF
        Y1=Y2
500    CONTINUE
511    RETURN
        END

C--
C*****
C--                               Subroutine RUNGE3
C*****
C--
        SUBROUTINE RUNGE3(C,Y0,I1,II,NN,H,ITER,NUM,IFLAG,IREF)
C--
C-- This is the third order Runge-Kutta method. The formula of this
C-- numerical method is:
C--    $Y_{n+1} = Y_n + aK_1 + bK_2 + cK_3$ 
C--    $K_1 = hf(X_n, Y_n)$ 
C--    $K_2 = hf(X_n + mh, Y_n + mK_1)$ 
C--    $K_3 = hf(X_n + nh, Y_n + (n-r)K_1 + rK_2)$ 
C-- here  $r = n(m-n)/m(3m-2)$ 
C--    $a = (6mn - 3m - 3n + 2)/(6mn)$ 
C--    $b = (3n - 2)/[6m(n - m)]$ 
C--    $c = (2 - 3m)/[6n(n - m)]$ 
C-- and  $Y_0 = Y(I1)$ . The interval is  $X = I1$  to  $X = I2$  and total ITER =
C--  $(I2 - I1)/h$  steps.
C--
        IMPLICIT REAL*8(A-H,O-Z)
        DOUBLE PRECISION I1,M,N,K1,K2,K3
        INTEGER ITER,NUM,II

        F(X,Y)=FUNC(C,X,Y,II,nn)
        IF(IFLAG.EQ.0) THEN
            CALL RK3PAR(A,B,C1,M,N,R)
        ELSE
            CALL RK3PP(IFLAG,IREF,A,B,C1,M,N,R)
        ENDIF
        CALL APPEAR(3,H,NN,C,II,Y0)
        P1=N-R
        X1=I1
        Y1=Y0
        DO 600 I=1,ITER
            K1=H*F(X1,Y1)
            K2=H*F(X1+M*H,Y1+M*K1)
            K3=H*F(X1+N*H,Y1+P1*K1+R*K2)
            X1=X1+H
            Y2=Y1+A*K1+B*K2+C1*K3
            IF(DABS(Y2-Y1).GT.999999999) THEN
                CALL MASSAG
                GO TO 611
            ENDIF
            IF(MOD(I,NUM).EQ.0. OR .I.LE.5 .OR. I.EQ.ITER) THEN
                CALL PRINT(I,II,C,X1,Y2,NN,Y0)
            
```



```

        ENDIF
        Y1=Y2
600  CONTINUE
611  RETURN
        END

C--
C*****
C-                                     Subroutine RUNGE4
C*****
C--
        SUBROUTINE RUNGE4(C,Y0,I1,I2,NN,H,ITER,NUM,IFLAG,IREP)
C--
C-- This is the third order Runge-Kutta method. The formula of this
C-- numerical method is:
C--  $Y_{n+1} = Y_n + aK_1 + bK_2 + cK_3 + dK_4$ 
C--  $K_1 = hf(X_n, Y_n)$ 
C--  $K_2 = hf(X_n + mh, Y_n + mK_1)$ 
C--  $K_3 = hf(X_n + nh, Y_n + (n-r)K_1 + rK_2)$ 
C--  $K_4 = hf(X_n + ph, Y_n + (p-s-t)K_1 + sK_2 + tK_3)$ 
C-- here  $r = n(m-n)/2m(2m-1)$ 
C--  $s = (1-m)(m-4n^*n+5n-2)/[2m(n-m)(6mn-4m-4n+3)]$ 
C--  $t = (1-2m)(1-m)(1-n)/[n(n-m)(6mn-4m-4n+3)]$ 
C--  $a = (6mn-2m-2n+1)/(12mn)$ 
C--  $b = (2n-1)/[12m(n-m)(1-m)]$ 
C--  $c = (2m-1)/[12n(n-m)(1-n)]$ 
C--  $d = (6mn-4m-4n+3)/[12(1-m)(1-n)]$ 
C-- and  $Y_0 = Y(I1)$ . The interval is  $X=I1$  to  $X=I2$  and total ITER=
C--  $(I2-I1)/h$  steps.
C--
        IMPLICIT REAL*8(A-H,O-Z)
        DOUBLE PRECISION I1,M,N,K1,K2,K3,K4
        INTEGER ITER

        F(X,Y)=FUNC(C,X,Y,I1,NN)
        IF(IFLAG.EQ.0) THEN
            CALL RK4PAR(A,B,C1,D,M,N,R,P,S,T)
        ELSE
            CALL RK4PP(IFLAG,IREP,A,B,C1,D,M,N,R,P,S,T)
        ENDIF
        CALL APPEAR(4,H,NN,C,I1,Y0)
        P1=N-R
        P2=P-S-T
        X1=I1
        Y1=Y0
        DO 700 I=1,ITER
            K1=H*F(X1,Y1)
            K2=H*F(X1+M*H,Y1+M*K1)
            K3=H*F(X1+N*H,Y1+P1*K1+R*K2)
            K4=H*F(X1+P*H,Y1+P2*K1+S*K2+T*K3)
            X1=X1+H
            Y2=Y1+A*K1+B*K2+C1*K3+D*K4
            IF(DABS(Y2-Y1).GT.999999999) THEN
                CALL MASSAG
                GO TO 711
            
```

```

        ENDIF
        IF(MOD(I,NUM).EQ.0 .OR. I.LE.5 .OR. I.EQ.ITER) THEN
            CALL PRINT(I,I.I.C.X1.Y2.NN.Y0)
        ENDIF
        Y1=Y2
700  CONTINUE
711  RETURN
    END

C--
C*****
C-          Subroutine MULTY
C-          It calls different multi-step method to solve the
C-          initial value problem.
C*****
C--
    SUBROUTINE MULTY(C.Y0.I1.II.NN.H.ITER.NUM)
    IMPLICIT REAL *8(A-H,O-Z)
    DOUBLE PRECISION I1
    INTEGER NUMBER,I1

    PRINT *,'We use Adams-Bashfourth formula to solve the problem.'
    PRINT *,'The formula is following:'
    PRINT *,'  Yn+1 = Yn + h(B0fn + B1fn-1 + ---- + Bkfn-k)'
    PRINT *,' Where k stand for the steps.'

    DO 750 I=1,100
    PRINT *,'Please choose from the following:'
    PRINT *,'1. Two steps method.'
    PRINT *,'2. Three steps method.'
    PRINT *,'3. Four steps method.'
    PRINT *,'4. Five steps method.'
    PRINT *,'5. Six steps method.'
    PRINT *,'6. Return to the previous menu.'
    PRINT *,'7. Quit'
    PRINT *
    PRINT *,'Enter number =====>'
    READ *,NUMBER
    GO TO (7779,7780,7781,7782,7783,7784,7785), NUMBER
7779 PRINT *,'Now you choose the two steps method.'
    CALL ADAMS2(C.Y0,I1,II,NN,H,ITER.NUM)
    GOTO 750
7780 PRINT *,'Now you choose the three steps method.'
    CALL ADAMS3(C.Y0,I1,II,NN,H,ITER.NUM)
    GOTO 750
7781 PRINT *,'Now you choose the four steps method.'
    CALL ADAMS4(C.Y0,I1,II,NN,H,ITER.NUM)
    GOTO 750
7782 PRINT *,'Now you choose the five steps method.'
    CALL ADAMS5(C.Y0,I1,II,NN,H,ITER.NUM)
    GOTO 750
7783 PRINT *,'Now you choose the six steps method.'
    CALL ADAMS(C.Y0,I1,II,NN,H,ITER.NUM)
    GOTO 750
7784 RETURN

```

```

7785 PRINT *,'You choose quit.'
      STOP
750  CONTINUE
      END

C--
C*****
C-
      Subroutine ADAMS2
C*****
C--
      SUBROUTINE ADAMS2(C,Y0,I1,II,NN,H,ITER,NUM)
C--
C-- It integrates the differential equation  $y'=f(x,y)$ . The formula
C-- of this numerical method is:
C--  $Y_{n+1} = Y_n + h(3f_n - f_{n-1})/2$ 
C-- Where  $Y_0=Y(I1)$ . The interval is  $X=I1$  to  $X=I2$  and total ITER=
C--  $(I2-I1)/h$  steps.
C--
      IMPLICIT REAL*8(A-H,O-Z)
      DOUBLE PRECISION II
      INTEGER ITER,NUM,II
      F(X,Y)=FUNC(C,X,Y,II,NN)

      PRINT *,' The formula of this method is:'
      PRINT *,'  $Y_{n+1} = Y_n + h(3f_n - f_{n-1})/2$ '
      CALL APPEAR(5,H,NN,C,II,Y0)
      X1=I1
      Y1=Y0
      X2=I1+H
      Y2=G(C,X2,II,NN,Y0)
      FF1=F(X1,Y1)
      DO 780 I=3,ITER
      FF2=F(X2,Y2)
      X2=X2+H
      Y3=Y2+h*(3.0D0*FF2-FF1)/2.0D0
      IF(DABS(Y3-Y2).GT.999999999) THEN
          CALL MASSAG
          GO TO 781
      ENDIF
      IF(MOD(I-1,NUM).EQ.0. OR .I.LE.8.or.I.EQ.ITER) THEN
          CALL PRINT(I-2,II,C,X2,Y3,NN,Y0)
      ENDIF
      FF1=FF2
      Y2=Y3
780  CONTINUE
781  RETURN
      END

C--
C*****
C-
      Subroutine ADAMS3
C*****
C--
      SUBROUTINE ADAMS3(C,Y0,I1,II,NN,H,ITER,NUM)
C--
C-- It integrates the differential equation  $y'=f(x,y)$ . The formula

```

```

C-- of this numerical method is:
C--  $Y_{n+1} = Y_n + h(23f_n - 16f_{n-1} + 5f_{n-2})/12$ 
C-- Where  $Y_0=Y(I1)$ . The interval is  $X=I1$  to  $X=I2$  and total ITER=
C--  $(I2-I1)/h$  steps.
C--
      IMPLICIT REAL*8(A-H,O-Z)
      DOUBLE PRECISION II,X(1:3),Y(1:4),FF(1:3)
      INTEGER ITER,NUM,II
      F(XX,YY)=FUNC(C,XX,YY,II,NN)

      PRINT *, ' The formula of this method is:'
      PRINT *, '  $Y_{n+1} = Y_n + h(23f_n - 16f_{n-1} + 5f_{n-2})/12$ '
      CALL APPEAR(6,H,NN,C,II,Y0)
      X(1)=I1
      Y(1)=Y0
      DO 5560 J=2,3
          X(J)=X(J-1)+H
          Y(J)=G(C,X(J),II,NN,Y0)
          FF(J-1)=F(X(J-1),Y(J-1))
5560  CONTINUE
      DO 800 I=4,ITER
          FF(3)=F(X(3),Y(3))
          X(3)=X(3)+H
          Y(4)=Y(3)+H*(23.0D0*FF(3)-16.0D0*FF(2)+5.0D0*FF(1))/12.0D0
          IF(DABS(Y(4)-Y(3)).GT.999999999) THEN
              CALL MASSAG
              GO TO 811
          ENDIF
          IF(MOD(I-1,NUM).EQ.0. OR .1.LE.9.OR.1.EQ.ITER) THEN
              CALL PRINT(I-3,II,C,X(3),Y(4),NN,Y0)
          ENDIF
          DO 5569 J=1,2
              FF(J)=FF(J+1)
5569  CONTINUE
          Y(3)=Y(4)
800  CONTINUE
811  RETURN
      END

C--
C*****
C-                               Subroutine ADAMS4
C*****
C--
      SUBROUTINE ADAMS4(C,Y0,II,II,NN,H,ITER,NUM)
C--
C-- It integrates the differential equation  $y'=f(x,y)$ . The formula
C-- of this numerical method is:
C--  $Y_{n+1} = Y_n + h(55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3})/24$ 
C-- Where  $Y_0=Y(I1)$ . The interval is  $X=I1$  to  $X=I2$  and total ITER=
C--  $(I2-I1)/h$  steps.
C--
      IMPLICIT REAL*8(A-H,O-Z)
      DOUBLE PRECISION II,X(1:4),Y(1:5),FF(1:4)
      INTEGER ITER,NUM,II

```

```

F(XX,YY)=FUNC(C,XX,YY,II,NN)

PRINT *, 'The formula of this method is:'
PRINT *, 'Yn+1 = Yn + h(55fn - 59fn-1 + 37fn-2 - 9fn-3)/24'
CALL APPEAR(7,H,NN,C,II,Y0)
X(1)=I1
Y(1)=Y0
DO 5561 J=2,4
  X(J)=X(J-1)+H
  Y(J)=G(C,X(J),II,NN,Y0)
  FF(J-1)=F(X(J-1),Y(J-1))
5561 CONTINUE
DO 850 I=5,ITER
  FF(4)=F(X(4),Y(4))
  X(4)=X(4)+H
  Y(5)=Y(4)+H*(55.0D0*FF(4)-59.0D0*FF(3)+37.0D0*FF(2)-9.0D0*
+ FF(1))/24.0D0
  IF(DABS(Y(5)-Y(4)).GT.999999999) THEN
    CALL MASSAG
    GO TO 851
  ENDIF
  IF(MOD(I-1,NUM).EQ.0. OR .I.LE.10.OR.I.EQ.ITER) THEN
    CALL PRINT(I-4,II,C,X(4),Y(5),NN,Y0)
  ENDIF
  DO 5568 J=1,3
    FF(J)=FF(J+1)
5568 CONTINUE
  Y(4)=Y(5)
850 CONTINUE
851 RETURN
END

C--
C*****
C--          Subroutine ADAMS5
C*****
C--
  SUBROUTINE ADAMS5(C,Y0,I1,II,NN,H,ITER,NUM)
C--
C-- It integrates the differential equation y'=f(x,y). The formula
C-- of this numerical method is:
C--   Yn+1 = Yn + h(1901fn - 2774fn-1 + 2616fn-2 - 1274fn-3
C--     +251fn-4)/720
C-- Where Y0=Y(I1). The interval is X=I1 to X=I2 and total ITER=
C-- (I2-I1)/h steps.
C--
  IMPLICIT REAL*8(A-H,O-Z)
  DOUBLE PRECISION II,X(1:5),Y(1:6),FF(1:5)
  INTEGER ITER,NUM,II
  F(XX,YY)=FUNC(C,XX,YY,II,NN)

  PRINT *, 'The formula of this method is:'
  PRINT *, 'Yn+1 = Yn + h(1901fn - 2774fn-1 + 2616fn-2 - 1274fn-3
+ +251fn-4)/720'
  CALL APPEAR(8,H,NN,C,II,Y0)

```

```

X(1)=I1
Y(1)=Y0
DO 5562 J=2,5
  X(J)=X(J-1)+H
  Y(J)=G(C,X(J),II,NN,Y0)
  FF(J-1)=F(X(J-1),Y(J-1))
5562 CONTINUE
DO 900 I=6,ITER
  FF(5)=F(X(5),Y(5))
  X(5)=X(5)+H
  Y(6)=Y(5)+H*(1901.0D0*FF(5)-2774.0D0*FF(4)+2616.0D0*FF(3)-
+ 1274.0D0*FF(2)+251.0D0*FF(1))/720.0D0
  IF(DABS(Y(6)-Y(5)).GT.999999999) THEN
    CALL MASSAG
    GO TO 911
  ENDIF
  IF(MOD(I-1,NUM).EQ.0. OR .I.LE.11.OR.I.EQ.ITER) THEN
    CALL PRINT(I-5,II,C,X(5),Y(6),NN,Y0)
  ENDIF
  DO 5565 J=1,4
    FF(J)=FF(J+1)
5565 CONTINUE
  Y(5)=Y(6)
900 CONTINUE
911 RETURN
END

C--
C*****
C- Subroutine ADAMS6
C*****
C--
SUBROUTINE ADAMS(C,Y0,II,II,NN,H,ITER,NUM)
C--
C-- It integrates the differential equation  $y'=f(x,y)$ . The formula
C-- of this numerical method is:
C--  $Y_{n+1} = Y_n + h(4277f_n - 7923f_{n-1} + 9982f_{n-2} - 7298f_{n-3}$ 
C--  $+ 2877f_{n-4} - 475f_{n-5})/1440$ 
C-- Where  $Y_0=Y(I1)$ . The interval is  $X=I1$  to  $X=I2$  and total ITER=
C--  $(I2-I1)/h$  steps.
C--
IMPLICIT REAL*8(A-H,O-Z)
DOUBLE PRECISION II,X(1:6),Y(1:7),FF(1:6)
INTEGER ITER,NUM,II

F(XX,YY)=FUNC(C,XX,YY,II,NN)

PRINT *, ' The formula of this method is:'
PRINT *, 'Yn+1 = Yn + h(4277fn - 7923fn-1 + 9982fn-2 - 7298fn-3
+ 2877fn-4 - 475fn-5)/1440'
CALL APPEAR(9,H,NN,C,II,Y0)
X(1)=I1
Y(1)=Y0
DO 5550 J=2,6
  X(J)=X(J-1)+H

```

```

        Y(J)=G(C,X(J),II,NN,Y0)
        FF(J-1)=F(X(J-1),Y(J-1))
5550  CONTINUE
        DO 999 I=7,ITER
            FF(6)=F(X(6),Y(6))
            X(6)=X(6)+H
            Y(7)=Y(6)+H*(4277.0D0*FF(6)-7923.0D0*FF(5)+9982.0D0*FF(4)
+       -7298.0D0*FF(3)+2877.0D0*FF(2)-475.0D0*FF(1))/1440.0D0
            IF(DABS(Y(7)-Y(6)).GT.999999999) THEN
                CALL MASSAG
                GO TO 919
            ENDIF
            IF(MOD(I-1,NUM).EQ.0. OR .I.LE.12.OR.I.EQ.ITER) THEN
                CALL PRINT(I,II,C,X(6),Y(7),NN,Y0)
            ENDIF
            DO 5551 J=1,5
                FF(J)=FF(J+1)
5551  CONTINUE
            Y(6)=Y(7)
999   CONTINUE
919   RETURN
      END

```

C--

C*****

C- Subroutine SELEH

C- It let you select stepsize h.

C*****

C--

```

SUBROUTINE SELEH(I1,I2,H,ITER,NUM)
DOUBLE PRECISION I1,I2,H
INTEGER ITER

```

PRINT *, 'Please input the stepsize h.'

READ *,H

ITER = (I2-I1)/H+1

NUM=0.1*ITER

PRINT *

PRINT *, ' The stepsize you choose is h=',h

PRINT *, ' The number of iterations is ITER=',ITER

PRINT *

RETURN

END

C--

C*****

C- Function G

C*****

C--

```

DOUBLE PRECISION FUNCTION G(C,X,II,NN,Y0)
IMPLICIT REAL*8(A-H,O-Z)
INTEGER II

```

IF(II.EQ.1) THEN

G=2.0D0+X*X

```

ELSEIF(II.EQ.2) THEN
  G=X
ELSEIF(II.EQ.3) THEN
  G=Y0*DEXP(C*X)+X**4
ELSEIF(II.EQ.4) THEN
  G=Y0*DEXP(C*X)+X**NN
ENDIF
RETURN
END

C--
C*****
C-                               Function FUNC
C*****
C--
  DOUBLE PRECISION FUNCTION FUNC(C,X,Y,II,NN)
  IMPLICIT REAL *8(A-H,O-Z)
  INTEGER II

  IF(II.EQ.1) THEN
    FUNC=2.0D0*X+C*(2.0D0+X*X)-C*Y
  ELSEIF(II.EQ.2) THEN
    FUNC=1.0D0+C*(Y-X)
  ELSEIF(II.EQ.3) THEN
    FUNC=4.0D0*X**3+C*(Y-X**4)
  ELSEIF(II.EQ.4) THEN
    FUNC=NN*X**(NN-1)+C*(Y-X**NN)
  ENDIF
  RETURN
  END

C--
C*****
C-                               Subroutine PRINT
C-  It writes data into the output file
C*****
C--
  SUBROUTINE PRINT(I,II,C,X,Y,NN,Y0)
  IMPLICIT REAL *8(A-H,O-Z)
  INTEGER I,II

  YEXACT=G(C,X,II,NN,Y0)
  ERROR=DABS(YEXACT-Y)
  WRITE(*,202)I,X,ERROR
202  FORMAT(I6,2X,F10.8,3X,D10.5)
  WRITE(111,222)I,X,Y,YEXACT,ERROR
222  FORMAT(I6,2X,F10.8,2X,D25.18,2X,D25.18,2X,D10.5)
123  RETURN
  END

C--
C*****
C-                               Subroutine MASSAG
C-  It writes not convergent message to the output file.
C*****
C--
  SUBROUTINE MASSAG

```



```

        WRITE(111,333)
333  FORMAT(2X,'ERRORS ARE GROWING EXPONERNIALLY -- OVERFLOW WILL
      + OCCUR - NUMERICAL SOLVING TERMINATED FOR CURRENT STEPSIZE')
      RETURN
      END

C--
C*****
C-
      Subroutine MAINPA
C- It let user choose all the parameters in the program.
C*****
C--
      SUBROUTINE MAINPA(C,Y0,I1,I2,II,NN,H,ITER,NUM)
      IMPLICIT REAL*8(A-H,O-Z)
      DOUBLE PRECISION I1,I2

      CALL SELEF(II,NN)
      CALL SELEC(C)
      CALL SELCY0(Y0)
      CALL SELINI(I1,I2)
      CALL SELEH(I1,I2,H,ITER,NUM)
      RETURN
      END

C--
C*****
C-
      Subroutine SELECT
C*****
C--
      SUBROUTINE SELECT(C,Y0,I1,I2,II,NN,H,ITER,NUM)
      IMPLICIT REAL*8(A-H,O-Z)
      DOUBLE PRECISION I1,I2
      CHARACTER CHOICE*2

      PRINT *
1240 PRINT *, 'Please select following:'
      PRINT *, ' 1. Changing function.'
      PRINT *, ' 2. Changing coefficient c.'
      PRINT *, ' 3. Changing initial value Y0'
      PRINT *, ' 4. Changing the interval I1 and I2.'
      PRINT *, ' 5. Changing stepsize H.'
      PRINT *, ' 6. Nothing to change.'
      READ *,NUMBER
      GO TO (1234,1235,1236,1237,1238,1242),NUMBER
1234 CALL SELEF(II,NN)
      GOTO 1239
1235 CALL SELEC(C)
      GOTO 1239
1236 CALL SELEY0(Y0)
      GOTO 1239
1237 CALL SELINI(I1,I2)
      GOTO 1239
1238 CALL SELEH(I1,I2,H,ITER,NUM)
1239 PRINT *
      PRINT *, 'Do you want to do more changes?'

```

```

        PRINT *,'If yes press Y, otherwise press N'
        READ(*,1241)CHOICE
1241  FORMAT(A2)
        IF(CHOICE.EQ.'Y') GOTO 1240
1242  RETURN
        END

C--
C*****
C-
        Subroutine SELEF
C*****
C--

        SUBROUTINE SELEF(II,NN)

        PRINT *
        PRINT *,'THE DIFFERENTIAL EQUATION IS OF THE FORM:'
        PRINT *,'1.  $dy/dx=2X+C(2+X*X)-CY$ '
        PRINT *,'2.  $dy/dx=1+C(Y-X)$ '
        PRINT *,'3.  $dy/dx=4X**3+C(Y-X**4)$ '
        PRINT *,'4.  $dy/dx=nX**(n-1)+C(Y-X**n)$ '
        PRINT *,' Y(0)=Y0'
        PRINT *,' WHICH FUNCTION DO YOU CHOOSE =====>'
        READ *.II
        PRINT *,' NOW YOU CHOOSE ' ,II
        IF(II.EQ.4) THEN
        PRINT *,'PLEASE INPUT THE VALUE OF THE CONSTANT N:'
        READ(*,*)NN
        ENDIF
        RETURN
        END

C--
C*****
C-
        Subroutine SELEC
C*****
C--

        SUBROUTINE SELEC(C)
        DOUBLE PRECISION C

        PRINT *
        PRINT *,'PLEASE INPUT THE VALUE OF THE CONSTANT C:'
        READ *.C
        PRINT *,'The coefficient C you choose is C=',C
        RETURN
        END

C--
C*****
C-
        Subroutine SELEY0
C*****
C--

        SUBROUTINE SELEY0(Y0)
        DOUBLE PRECISION Y0

```

```

PRINT *
PRINT *.'PLEASE INPUT THE VALUE OF THE INITIAL VALUE Y0:'
READ *,Y0
PRINT *.'The initial values Y0 you choose is Y0='.Y0
RETURN
END
C--
C*****
C-                               Subroutine SELINI
C*****
C--
SUBROUTINE SELINI(I1,I2)
DOUBLE PRECISION I1,I2

PRINT *
PRINT *.'PLEASE INPUT THE VALUE OF THE INTERVAL VALUE I1 AND I2:'
101 READ *,I1,I2
PRINT *
PRINT *.'The independent varibale x is between'.I1.' AND '.I2
IF(I1.GE.I2) THEN
PRINT *.'Left endpoint of the interval is great or equal to the right.'
PRINT *.'Try it again.'
GOTO 101
ENDIF
RETURN
END
C--
C*****
C-                               Subroutine APPEAR
C*****
C--
SUBROUTINE APPEAR(NUM,H,NN,C,II,Y0)
IMPLICIT REAL*8(A-H,O-Z)

WRITE(111,*)
WRITE(111,*)
IF(NUM.EQ.1) THEN
WRITE(111,*)'This is Euler method.'
ELSEIF(NUM.EQ.2) THEN
WRITE(111,*)'This is Runge-Kutta order 2 method.'
ELESIF(NUM.EQ.3) THEN
WRITE(111,*)'This is Runge-Kutta order 3 method.'
ELSEIF(NUM.EQ.4) THEN
WRITE(111,*)'This is Runge-Kutta order 4 method.'
ELESIF(NUM.EQ.5) THEN
WRITE(111,*)'This is Adams2 method.'
ELSEIF(NUM.EQ.6) THEN
WRITE(111,*)'This is Adams3 method.'
ELSEIF(NUM.EQ.7) THEN
WRITE(111,*)'This is Adams4 method.'
ELSEIF(NUM.EQ.8) THEN
WRITE(111,*)'This is Adams5 method.'
ELSEIF(NUM.EQ.9) THEN
WRITE(111,*)'This is Adams6 method.'

```


C--

```

SUBROUTINE RK3PAR(A,B,C1,M,N,R)
IMPLICIT REAL*8(A-H,O-Z)
DOUBLE PRECISION M,N

540 PRINT *,'The formula of the third order Runge-Kutta method is:'
PRINT *,' Yn+1 = Yn + aK1 + bK2 + c1K3'
PRINT *,' K1=hf(Xn,Yn)'
PRINT *,' K2=hf(Xn+mh,Yn+mK1)'
PRINT *,' K3=hf(Xn+nh,Yn+(n-r)K1+rK2)'
PRINT *,' here r=n(m-n)/m(3m-2)'
PRINT *,' a=(6mn-3m-3n+2)/(6mn)'
PRINT *,' b=(3n-2)/[6m(n-m)]'
PRINT *,' c1=(2-3m)/[6n(n-m)]'
PRINT *,' This is a two parameter family method.'
PRINT *,' 1. If m=n=2/3, then'
PRINT *,' a=1/4; b=(3r-1)/(4r) ; c1=1/(4r)'
PRINT *,' where if r=1/3 it is Heun third order formula'
PRINT *,' 2. Heun third order formula.'
PRINT *,' 3. If m=2/3 and n=0.0, then'
PRINT *,' a=(r-1)/(4r); b=3/4 ; c1=1/(4r)'
PRINT *,' 4. You choose any pair of m,n'
PRINT *,' 5. You choose m=1/2, n=1'
PRINT *,' Which do you choose?'
PRINT *
READ *.NUMBER
GO TO (2220,2221,2222,2223,2224).NUMBER
2220 PRINT *
PRINT *,'m=2/3; n=2/3; a=1/4; b=(3r-1)/(4r) ; c1=1/(4r)'
PRINT *,' If r=1/3, it is Heun third order formula'
PRINT *,' Now please choose the parameter r.'
2228 READ *.R
IF(R.EQ.0.0) THEN
PRINT *,' You could not choose r = 0. Please choose it again.'
GO TO 2228
ENDIF
2221 IF(NUMBER.EQ.2) THEN
R=1.0D0/3.0D0
ENDIF
B=(3.0D0*R-1.0D0)/(4.0D0*R)
C1=1.0D0/(4.0D0*R)
M=2.0D0/3.0D0
N=M
A=0.25D0
GOTO 2229
2222 PRINT *,'m=2/3; n=0; a=(r-1)/(4*r); b=3/4 ; c1=1/(4r)'
PRINT *,' so you will choose the parameter r.'
2227 READ *.R
IF(R.EQ.0.0) THEN
PRINT *,' You could not choose r = 0. Please choose it again.'
GO TO 2227
ENDIF
A=(R-1.0D0)/(4.0D0*R)
B=3.0D0/4.0D0

```

```

C1=1.0D0/(4.0D0*R)
M=2.0D0/3.0D0
N=0.0
GOTO 2229
2223 PRINT *
PRINT *,'Please choose the parameter m,n.'
READ *.M,N
2224 IF(NUMBER.EQ.5) THEN
M=.50D0
N=1.0D0
ENDIF
TEMP1=M*(3.0D0*M-2.0D0)
TEMP2=M*N
TEMP3=6.0D0*M*(N-M)
TEMP4=6.0D0*N*(N-M)
IF(TEMP1.EQ.0.0D0 .OR. TEMP2.EQ.0.0D0 .OR. TEMP3.EQ.0.0 .OR.
+ TEMP4.EQ.0.0) THEN
PRINT *,' You could not choose that pair. Please choose
+ it again.'
GO TO 2223
ENDIF
R=N*(M-N)/TEMP1
A=(6.0D0*TEMP2-3.0D0*M-3.0D0*N+2.0D0)/(6.0D0*TEMP2)
B=(3.0D0*N-2.0D0)/TEMP3
C1=(2.0D0-3.0D0*M)/TEMP4
2229 PRINT *,'Now you have chosen a='.A
PRINT *,'          b='.B
PRINT *,'          c1='.C1
PRINT *,'          m='.M
PRINT *,'          n='.N
PRINT *,'          r='.R

RETURN
END

C--
C*****
C-          Subroutine RK4PAR
C*****
C--

SUBROUTINE RK4PAR(A,B,C1,D,M,N,R,P,S,T)
IMPLICIT REAL*8(A-H,O-Z)
DOUBLE PRECISION M,N

640 PRINT *,'The formula of the fourth order Runge-Kutta method is:'
PRINT *,' Yn+1 = Yn + aK1 + bK2 + cK3 + dK4'
PRINT *,' K1=hf(Xn,Yn)'
PRINT *,' K2=hf(Xn+mh,Yn+mK1)'
PRINT *,' K3=hf(Xn+nh,Yn+(n-r)K1+rK2)'
PRINT *,' K4=hf(Xn+ph,Yn+(p-s-t)K1+sK2+tK3)'
PRINT *,' here r=n(m-n)/2m(2m-1)'
PRINT *,' s=(1-m)(m-4n*n+5n-2)/[2m(n-m)(6mn-4m-4n+3)]'
PRINT *,' t=(1-2m)(1-m)(1-n)/[n(n-m)(6mn-4m-4n+3)]'
PRINT *,' a=(6mn-2m-2n+1)/(12mn)'
PRINT *,' b=(2n-1)/[12m(n-m)(1-m)]'

```

```

PRINT *,' c=(2m-1)/[12n(n-m)(1-n)]'
PRINT *,' d=(6mn-4m-4n+3)/[12(1-m)(1-n)]; p=1'
PRINT *,' This is a two parameter family method.'
PRINT *,' Please choose following:'
PRINT *,' 1. m=n=0.5, then'
PRINT *,' a=1/6; b=(2-t)/3 ; c1=t/3; d=1/6; p=1:'
PRINT *,' r=1/(2t); s=1-t; '
PRINT *,' where t=1.0, is standard Runge-Kutta method.'
PRINT *,' where t=1.0+sqrt(0.5), is Gill formula'
PRINT *,' 2. Standard Runge-Kutta method.'
PRINT *,' 3. Gill formula'
PRINT *,' 4. m=1.0 and n=0.5, then'
PRINT *,' a=1/6; b=(t-2)/(6*t) ; c1=2/3; d=1/(3*t); p=1:'
PRINT *,' r=1/8; s=-t/4; '
PRINT *,' 5. m=0.5 and n=0.0, then'
PRINT *,' a=(1-t)/6; b=2/3 ; c1=t/6; d=1/6; p=1:'
PRINT *,' r=1/(2*t); s=3/2; '
PRINT *,' 6. You choose any pair of m,n'
PRINT *,' 7. Hull and Johnson method. (m=0.35, n=0.45)'
PRINT *,' 8. Merson method. ( m=1/3, n=1/2 )'
PRINT *,' 9. m=1/3, n=2/3'
READ *.NUMBER
GO TO (6991,6992,6992,6993,6994,6995,6996,6996,6996).NUMBER
6991 PRINT *,' a=1/6; b=(2-t)/3 ; c1=t/3; d=1/6; p=1:'
PRINT *,' r=1/(2t); s=1-t; m=0.5; n=0.5.'
PRINT *,' If t=1.0,it is standard Runge-Kutta method.'
PRINT *,' If t=1.0+sqrt(0.5),it is Gill formula'
PRINT *,' Now please choose the parameter t.'
6989 READ * T
IF(T.EQ.0.0) THEN
PRINT *,' You could not choose t = 0. Please choose it again.'
GO TO 6989
ENDIF
6992 IF(NUMBER.EQ.2) THEN
T=1.0D0
ELESIF(NUMBER.EQ.3) THEN
T=1.0D0+DSQRT(0.5D0)
ENDIF
A=1.0D0/6.0D0
B=(2.0D0-T)/3.0D0
C1=T/3.0D0
D=1.0D0/6.0D0
P=1.0D0
R=1.0D0/(2.0D0*T)
S=1.0D0-T
M=0.50D0
N=0.50D0
GO TO 6990
6993 PRINT *,' a=1/6; b=(t-2)/(6*t) ; c1=2/3; d=1/(3*t); p=1:'
PRINT *,' r=1/8; s=-t/4; m=1; n=0.5'
PRINT *,' so you will choose the parameter t.'
651 READ * T
IF(T.EQ.0.0) THEN
PRINT *,' You could not choose t = 0. Please choose it again.'

```

```

        GO TO 651
    ENDIF
    A=1.0D0/6.0D0
    B=(T-2.0D0)/(6.0D0*T)
    C1=2.0D0/3.0D0
    D=1.0D0/(3.0D0*T)
    P=1.0D0
    R=1.0D0/8.0D0
    S=-(T/4.0D0)
    M=1.0D0
    N=0.5D0
    GOTO 6990
6994 PRINT *, ' a=(1-t)/6; b=2/3 ; c1=t/6; d=1/6; p=1;'
    PRINT *, ' r=1/(2*t); s=1.5; m=0.5; n=0.'
    PRINT *, ' so you will choose the parameter t.'
652 READ *, T
    IF(T.EQ.0.0) THEN
        PRINT *, 'You could not choose t = 0. Please choose it again.'
        GO TO 652
    ENDIF
    A=(1.0D0-T)/6.0D0
    B=2.0D0/3.0D0
    C1=T/6.0D0
    D=1.0D0/6.0D0
    P=1.0D0
    R=1.0D0/(2.0D0*T)
    S=1.50D0
    M=0.50D0
    N=0.0D0
    GOTO 6990
6995 PRINT *
    PRINT *, 'Please choose the parameter m.n.'
653 READ *, M, N
6996 IF(NUMBER.EQ.7) THEN
    M=0.35D0
    N=0.45D0
    ELSEIF(NUMBER.EQ.8) THEN
    M=1.0D0/3.0D0
    N=.5D0
    ELSEIF(NUMBER.EQ.9) THEN
    M=1.0D0/3.0D0
    N=2.0D0/3.0D0
    ENDIF
    TEMP1=M*(2.0D0*M-1.0D0)
    TEMP2=M*N
    TEMP3=(N-M)*(6.0D0*TEMP2-4.0D0*M-4.0D0*N+3.0D0)
    TEMP4=M*(N-M)*(1.0D0-M)
    TEMP5=N*(N-M)*(1.0D0-N)
    TEMP6=(1.0D0-M)*(1.0D0-N)
    IF(TEMP1.EQ.0.0 .OR. TEMP2.EQ.0.0 .OR. TEMP3.EQ.0.0 .OR.
+ TEMP4.EQ.0.0 .OR. TEMP5.EQ.0.0 .OR. TEMP6.EQ.0.0) THEN
        PRINT *, 'You could not choose that pair. Please try again.'
        GO TO 653
    ENDIF

```



```

R=N*(M-N)/(2.0D0*TEMP1)
S=(1.0D0-M)*(M-4.0D0*N*N+5.0D0*N-2)/(2.0D0*M*TEMP3)
T=(1-2.0D0*M)*temp6/(N*TEMP3)
A=(6.0D0*TEMP2-2.0D0*M-2.0D0*N+1)/(12.0D0*TEMP2)
B=(2.0D0*N-1)/(12.0D0*TEMP4)
C=(2.0D0*M-1)/(12.0D0*TEMP5)
D=(6.0D0*TEMP2-4.0D0*M-4.0D0*N+3.0D0)/(12.0D0*TEMP6)
P=1.0D0
6990 PRINT *, 'Now you have chosen a=', A
      PRINT *, '          b=', B
      PRINT *, '          c1=', C1
      PRINT *, '          d=', D
      PRINT *, '          m=', M
      PRINT *, '          n=', N
      PRINT *, '          p=', P
      PRINT *, '          r=', R
      PRINT *, '          s=', S
      PRINT *, '          t=', T

      RETURN
      END

C--
C*****
C-          Subroutine AUTO
C*****
C--

      SUBROUTINE AUTO
      IMPLICIT REAL*8(A-H,O-Z)
      CHARACTER CHOICE*2

      PRINT *, 'DO YOU WANT TO RUN THE PROGRAM automatically?'
      PRINT *, 'If yes type Y, if not type in N'
      READ(*,1342)CHOICE
1342  FORMAT(A2)
      IF ( CHOICE.EQ.'N'.OR .CHOICE.EQ.'N') THEN
          GOTO 1345
      ENDIF
      C=-1000.0D0
      NN=2
      Y0=1
      OI1=0.0
      DO 6300 II=2,4,2
          DO 6099 ICONTRL=1,4
              GOTO (6095,6095,6096,6097) ICONTRL
6095          HH=-2.0D0/C
              H1=HH
              GO TO 6098
6096          HH=-2.51D0/C
              GO TO 6098
6097          HH=-2.78D0/C
6098          CALL ATSTEP(II,NN,ICONTRL,HH,C,Y0,OI1)
6099          CONTINUE

          H1=H1/16.0D0

```

```

        DO 6199 ICONTRL=1,4
          CALL AUTOC(IL,NN,ICONTRL,H1,Y0,OI1)
6199    CONTINUE

        CALL ATMETH(IL,NN,H1,Y0,OI1)
6300    CONTINUE
        PRINT *,'DO YOU WANT TO CONTINUE TO RUN THE PROGRAM ?'
        PRINT *,'If yes type Y, if not type in N'
        READ(*,1341)CHOICE
1341    FORMAT(A2)
        IF ( CHOICE.EQ.'N'.OR .CHOICE.EQ.'n') THEN
          STOP
        ENDIF

1345    RETURN
        END

C--
C*****
C-                               Subroutine ATSTEP
C*****
C--
SUBROUTINE ATSTEP(IL,NN,ICONTRL,H,C,Y0,OI1)
IMPLICIT REAL*8(A-H,O-Z)

        HH=(H-0.00001)/20.0D0
        H=H+HH
        DO 6306 I=1,20
          H=H-HH
          ITER=1.0D0/H+1
          NUM=0.1*ITER
          GOTO(6302,6303,6304,6305) ICONTRL
6302    CALL EULER(C,Y0,OI1,IL,NN,H,ITER,NUM)
          GOTO 6306
6303    CALL RUNGE2(C,Y0,OI1,IL,NN,H,ITER,NUM,1,1)
          GOTO 6306
6304    CALL RUNGE3(C,Y0,OI1,IL,NN,H,ITER,NUM,1,1)
          GOTO 6306
6305    CALL RUNGE4(C,Y0,OI1,IL,NN,H,ITER,NUM,1,1)
6306    CONTINUE
        RETURN
        END

C--
C*****
C-                               Subroutine AUTOC
C*****
C--
SUBROUTINE AUTOC(IL,NN,ICONTRL,H1,Y0,OI1)
IMPLICIT REAL*8(A-H,O-Z)

        ITER=1.0D0/H1+1
        NUM=0.1*ITER
        C=-1050.0D0
        DO 3606 I=1,20
          C=C+50.0D0

```

```

          GOTO(3602,3603,3604,3605) ICONTRL
3602     CALL EULER(C,Y0,OI1,II,NN,H1,ITER,NUM)
          GOTO 3606
3603     CALL RUNGE2(C,Y0,OI1,II,NN,H1,ITER,NUM,2,1)
          GOTO 3606
3604     CALL RUNGE3(C,Y0,OI1,II,NN,H1,ITER,NUM,2,1)
          GOTO 3606
3605     CALL RUNGE4(C,Y0,OI1,II,NN,H1,ITER,NUM,2,1)
3606     CONTINUE

      RETURN
      END

C--
C*****
C-                               Subroutine ATMETH
C*****
C--
      SUBROUTINE ATMETH(II,NN,H1,Y0,OI1)
      IMPLICIT REAL*8(A-H,O-Z)

      ITER=1.0D0/H1+1
      NUM=0.1*ITER
      C=-1000.0D0
      IREP=1
      DO 3610 I=1,3
          PRINT *, ' ON ATMETH BEFORE RK2 CALL I=' ,I
          CALL RUNGE2(C,Y0,OI1,II,NN,H1,ITER,NUM,3,IREP)
3610     CONTINUE
          IREP=1
          DO 3620 I=1,4
              CALL RUNGE3(C,Y0,OI1,II,NN,H1,ITER,NUM,3,IREP)
3620     CONTINUE
          IREP=1
          DO 3630 I=1,6
              CALL RUNGE4(C,Y0,OI1,II,NN,H1,ITER,NUM,3,IREP)
3630     CONTINUE
          RETURN
          END

C--
C*****
C-                               Subroutine RK2PP
C*****
C--
      SUBROUTINE RK2PP(IFLAG,IREP,A,B,M)
      IMPLICIT REAL*8(A-H,O-Z)
      DOUBLE PRECISION M
      PRINT *, 'irep=' ,IREP
      IF(IFLAG.EQ.3) THEN
          IF(IREP.EQ.1) THEN
              WRITE(*,*) ' RK2 Midpoint method m=0.5'
              WRITE(111,*) ' RK2 Midpoint method m=0.5'
              M=0.50D0
              IREP = IREP+1
          ELSEIF(IREP.EQ.2) THEN

```

```

        PRINT *, 'RK2 Heun method of order 2 m=2/3'
        WRITE(111,1010)
1010   FORMAT(' RK2 Heun method of order 2 m=2/3')
        M=2.0D0/3.0D0
        IREP = IREP+1
        ELSE
            WRITE(111,*)' RK2 Modified Euler method m=1'
            WRITE(*,*)' RK2 Modified Euler method m=1'
            M=1.0D0
        ENDIF
    ELSE
        M=1.0D0
    ENDIF
    B=1.0D0/(2.0D0*M)
    A=1.0D0-B

    RETURN
    END

C--
C*****
C-                               Subroutine RK3PP
C*****
C--
    SUBROUTINE RK3PP(IFLAG,IREP,A,B,C1,M,N,R)
    IMPLICIT REAL *8(A-H,O-Z)
    DOUBLE PRECISION M,N

    IF(IFLAG.EQ.3) THEN
        GOTO (6331,6332,6333,6334) IREP
C
C*** HENU METHOD **
C
6331  R=1.0D0/3.0D0
        B=(3.0D0*R-1.0D0)/(4.0D0*R)
        C1=1.0D0/(4.0D0*R)
        M=2.0D0/3.0D0
        N=M
        PRINT *, ' RK3 Heun method of order 3 m=2/3'
        WRITE(111,1011)
1011  FORMAT(' RK3 Heun method of order 3 m=2/3')
        A=0.25D0
        IREP = IREP+1
        GOTO 6337
C
c*** CLASSICAL METHOD **
C
6332  M=0.50D0
        N=1.0D0
        WRITE(111,1012)
1012  FORMAT(' RK3 classical method of order 3 m=0.5, n=1')
        PRINT *, ' RK3 classical method of order 3 m=0.5, n=1'
        GOTO 6336
C
C*** CONTE AND REEVES METHOD **

```

```

C
6333 M=0.6265383
      N=0.0754259
      PRINT *, ' RK3 Conte and Reeves method '
      WRITE(111,1013)
1013 FORMAT(' RK3 Conte and Reeves method ')
      GOTO 6336
C
c*** OPTIMAL METHOD **
C
6334 ENDIF
      M=(10.0D0-2.0D0*DSQRT(13.0D0))/6.0D0
      N=(1.0D0+DSQRT(13.0D0))/6.0D0
      PRINT *, ' RK3 optimal method '
      WRITE(111,1014)
1014 FORMAT(' RK3 optimal method ')
6336 TEMP1=M*(3.0D0*N-2.0D0)
      TEMP2=M*N
      TEMP3=6.0D0*M*(N-M)
      TEMP4=6.0D0*N*(N-M)
      R=N*(M-N)/TEMP1
      A=(6.0D0*TEMP2-3.0D0*M-3.0D0*N+2.0D0)/(6.0D0*TEMP2)
      B=(3.0D0*N-2.0D0)/TEMP3
      C1=(2.0D0-3.0D0*M)/TEMP4
      IF(IFLAG.EQ.3) THEN
        IREP = IREP+1
      ENDIF
6337 RETURN
      END
C--
C*****
C-                               Subroutine RK4PP
C*****
C--
      SUBROUTINE RK4PP(IFLAG,IREP,A,B,C1,D,M,N,R,P,S,T)
      IMPLICIT REAL *8(A-H,O-Z)
      DOUBLE PRECISION M,N
      IF(IFLAG.EQ.3) THEN
        GOTO (6341,6342,6343,6344,6345,6346) IREP
C
c*** CLASSICAL METHOD **
C
6341 T=1.0D0
      PRINT *, ' RK4 classical method '
      WRITE(111,1015)
1015 FORMAT(' RK4 classical method ')
      GOTO 6338
C
c*** GILL METHOD **
C
6342 T=1.0D0+DSQRT(.05D0)
      PRINT *, ' RK4 Gill method '
      WRITE(111,1016)
1016 FORMAT(' RK4 Gill method ')

```

```

6338 M=0.50D0
      N=M
      A=1.0D0/6.0D0
      B=(2.0D0-T)/3.0D0
      C1=T/3.0D0
      D=1.0D0/6.0D0
      P=1.0D0
      R=1.0D0/(2.0D0*T)
      S=1.0D0-T
      IREP = IREP+1
      GOTO 6349
C
c*** MERSON METHOD **
C
6343 M=1.0D0/3.0D0
      N=0.50D0
      PRINT *,' RK4 Merson method '
      WRITE(111,1017)
1017 FORMAT(' RK4 Merson method ')
      GOTO 6339
C
C*** HULL AND JOHNSON METHOD **
C
6344 M=0.35D0
      N=0.45D0
      PRINT *,' RK4 Hull and Johnson method '
      WRITE(111,1018)
1018 FORMAT(' RK4 Hull and Johnson method ')
      GOTO 6339
C
C*** BOULTON METHOD **
C
6345 M=1.0D0/3.0D0
      N=2.0D0*M
      PRINT *,' RK4 Boulton method '
      WRITE(111,1019)
1019 FORMAT(' RK4 Boulton method ')
      GOTO 6339
C
C*** OPTIMAL METHOD **
C
6346 ENDIF
      M=2.0D0/5.0D0
      N=3.0D0/5.0D0
      PRINT *,' RK4 optimal method '
      WRITE(111,1021)
1021 FORMAT(' RK4 optimal method ')
6339 TEMP1=M*(2.0D0*M-1.0D0)
      TEMP2=M*N
      TEMP3=(N-M)*(6.0D0*TEMP2-4.0D0*M-4.0D0*N+3.0D0)
      TEMP4=M*(N-M)*(1.0D0-M)
      TEMP5=N*(N-M)*(1.0D0-N)
      TEMP6=(1.0D0-M)*(1.0D0-N)
      R=N*(M-N)/(2.0D0*TEMP1)

```

```
S=(1.0D0-M)*(M-4.0D0*N*N+5.0D0*N-2)/(2.0D0*M*TEMP3)
T=(1-2.0D0*M)*TEMP6/(N*TEMP3)
A=(6.0D0*TEMP2-2.0D0*M-2.0D0*N+1)/(12.0D0*TEMP2)
B=(2.0D0*N-1)/(12.0D0*TEMP4)
C=(2.0D0*M-1)/(12.0D0*TEMP5)
D=(6.0D0*TEMP2-4.0D0*M-4.0D0*N+3.0D0)/(12.0D0*TEMP6)
P=1.0D0
IF(IFLAG.EQ.3) THEN
  IREP = IREP+1
ENDIF
6349 RETURN
END
```

VITA

Zu-Lei Li

Candidate for the Degree of
Master of Science

Thesis: THE GLOBAL ERROR OF SINGLE-STEP METHODS FOR INITIAL
VALUE PROBLEMS

Major Field: Computer Science

Biographical:

Personal Data: Born in Shanghai, People's Republic of China, August 3, 1958, the son of Song-nian Li and Xiu-zhen Zhao.

Education: Graduated from Shanghai Hong-Kou Senior High School, in January 1976; received Bachelor of Science Degree in Shanghai Teacher's University, Shanghai, People's Republic of China, in June 1982; completed requirements for the Master of Science degree at Oklahoma State University in July, 1993.

Professional Experience: Teacher, Shanghai Dong-Feng High School, September 1982, to December 1987; Manager of Business Department, Shanghai MingTai Chemical Industry Co. LTD.; Research Assistant, Department of Chemical Engineering, Oklahoma State University, January, 1993, to March, 1993.