Universiti Malaysia Terengganu Journal of Undergraduate Research *eISSN: 2637-1138*
Volume 3 Number 1, January 2021: 25-36 *© Penerbit UMT*

ADAPTIVE TIME-STEPPING FOR RUNGE-KUTTA METHODS FOR ORDINARY DIFFERENTIAL EQUATIONS

CHUA KAH WAI, LOY KAK CHOON* AND RUWAIDIAH IDRIS

Faculty of Ocean Engineering Technology and Informatics, Universiti Malaysia Terengganu, 21030 Kuala Nerus, Terengganu, Malaysia.

**Corresponding author: kakchoon@umt.edu.my* *http://doi.org/10.46754/umtjur.2021.01.004*

Abstract: Ordinary Differential Equations (ODEs) are usually used in numerous fields especially in solving the modelling problem. Numerical methods are one of the vital mathematical tools to solve the ODEs that appear in various modelling problems by determining the approximation solution close to the in exact solution if it exists. Runge-Kutta methods (RK) are the numerical methods used to integrate the ODEs by applying multistage methods at the midpoint of an interval which can efficiently produce a more accurate result or small magnitude of error. We proposed Runge-Kutta methods (RK) to solve the 1st– order nonlinear stiff ODEs. The RK methods used in this research are known as the RK-2, RK-4, and RK-5 methods. We proved the existence and uniqueness of the ODEs before we solved it numerically. We also proved the absolute-stability of the RK methods to determine the overall stability of these methods. We found two suitable test cases which are the standard test problem and manufactured solution. We proved that by combining the adaptive step size with RK methods can result in more efficient computation. We implemented the 2nd–, 4th– and 5th– order of RK methods with step size adaptively algorithm to solve the test problem and manufactured solution via Octave programming language. The resulting numerical error and the stability of each method can be studied. We compared our results using several error plots versus the Central Processing Unit (CPU) time required to compute a given nonlinear 1st– order stiff ODE problem. In a conclusion, RK methods which combine with the adaptive step size can result in more efficient computation and accuracy compare with the fixed step size RK methods.

Keywords: Ordinary Differential Equation; Runge-Kutta; Multistage method; Adaptive step size; non-linear stiff equation.

Introduction

Ordinary Differential Equations

A differential equation is an equation which contains the function and its derivatives. A differential equation is said to be linear if it can be written as a linear combination of the derivatives. In general, the differential equation can be divided into two categories which are the ordinary differential equations (ODEs) and partial differential equations (PDEs).

Ordinary differential equations (ODEs) is a function that involves one or higher order derivatives with only one independent variable. Normally, solving the first order linear ODEs is easy by using various type of methods such as the exact method, separation of variables, integrating factors, variation of parameters, and method of undetermined coefficients. In general, nonlinear ODEs are hard to be solved analytically because there are many functions that we can construct using a finite combination of finite number of elementary functions. ODEs can be grouped into two types which are linear and nonlinear differential equations. For the first-order linear ODEs, the function $f(t,y)$ can be expressed in a linear form

$$f(t,y) = a(x)y+b(x), \qquad (1.1)$$

where $a(x)$ and $b(x)$ are arbitrary functions of x only. The first-order nonlinear ODEs can be simply written as

$$y' = f(x,y) \text{ or } \frac{dy}{dx}(x) = f(x,y(x)) \quad (1.2)$$

In our daily life, ODEs are used in a wide variety of disciplines, from biology, physics, chemistry, economics, and engineering. Some examples of real-world application by using ODEs are such as in medicine for modelling cancer growth or the spread of a disease, in engineering for describing the movement of electricity, in physics to describe the motion of waves, pendulum or chaotic systems and in chemistry for modelling chemical reactions and half-life for decaying radioactive isotopes.

Runge-Kutta Methods

There are many known methods to solve ODE problems. Runge-Kutta methods are the most common ODE solvers. Runge-Kutta methods are the generalization of the concept used in Modified Euler's method. Runge-Kutta methods are numerical methods that can integrate the ordinary differential equation by using multistep trial at the midpoint of an interval. Runge-Kutta has the advantage of requiring only explicit evaluation of the derivatives of *f*(*t,y*).The step-size can be modified any time. In this project, I can use Runge-Kutta methods to solve the first-order nonlinear stiff ODEs with adaptive step size.

Adaptive Step Size

During the progress, good ODE solvers may have some adaptive control that making repeated changes in its step size which can make the computation more efficient. Normally the objective of adaptive step size control is to obtain some predetermined accuracy in the solution with minimum computational effort. Sometimes numerical accuracy is not directly demanded in the solution itself, but in some related conserved quantity that can be monitored. Implementation of adaptive step size control requires that the stepping algorithm signal information about its performance and the important condition is an estimation of its truncation error.

Methodology

Runge-Kutta Methods For Stiff ODEs

In this section, we will use Runge-Kutta methods to solve the 1st– order nonlinear stiff ordinary differential equation (ODE). Runge-Kutta methods have been chosen in this project is because this method is a multistage method which has better stability even though they are categorized as a fully explicit method. RK method does not require initialization and it is efficient since high order RK methods are already available. Hence, we will introduce the general formula of 2nd–, 4th– and 5th– order Runge-Kutta methods to handle stiff ordinary differential equations (ODEs).

The general equation of Runge-Kutta method can be expressed below:

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i k_i, \tag{2.1}$$

is called an -stage RK method.

where

$$k_1 = f(t_n, y_n) \tag{2.2}$$
$$k_2 = f\big(t_n + c_2 h, y_n + h(a_{21}k_1)\big) \tag{2.3}$$
$$k_3 = f\big(t_n + c_3 h, y_n + h(a_{31}k_1 + a_{32}k_2)\big) \tag{2.4}$$
$$k_s = f\Big(t_n + c_s h, y_n + h\big(a_{s1}k_1 + a_{s2}k_2 + \cdots + a_{s,s-1}k_{s-1}\big)\Big) \tag{2.5}$$

where the Butcher tableau with coefficients are written as

$$\begin{array}{c|ccccc}
0 & & & & & \\
c_2 & a_{21} & & & & \\
c_3 & a_{31} & a_{32} & & & \\
\vdots & \vdots & & \ddots & & \\
c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} & \\
\hline
 & b_1 & b_2 & \cdots & b_{s-1} & b_s
\end{array}$$

A general explicit Euler method is given by

$$y_{n+1} = y_n + hf(y_n, t_n)\,, \tag{2.6}$$

with Butcher tableau

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}\;,$$

while the implicit Euler method is given by

$$y_{n+1} = y_n + hf(y_{n+1}, t_n + h)\,, \tag{2.7}$$

with its Butcher tableau

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array}\;.$$

Test Problem and Manufactured Solution

Test Problem of 1st-Order Nonlinear Stiff ODE

In this section, we provide a 1st- order nonlinear stiff ODE with an exact solution. By using this test problem is to demonstrate the numerical illustration and we will use the exact solution to validate the programming code. The test case for 1st-order order nonlinear ODEs has the following condition:

$$y' = -15y,\ y = 1 \text{ whem } t = 0, \tag{2.8}$$

where the exact solution is given

$$y(t) = e^{-15t} \tag{2.9}$$

Manufacture a 1st-Order Nonlinear Stiff ODE

In this section, we will discuss about the manufacturing of an ODE equation. Hence, we let a 1st- order nonlinear stiff ODE to have the following expression:

$$y' - 4y^2 + y + 3t = g(t), \text{ with } y(0) = y_0 \tag{2.10}$$

where $g(t)$ is a function of t alone.

Furthermore, we assume a stiff solution to be

$$y(t) = \sin(t)\, e^{20t}, \tag{2.11}$$

where the initial condition $y(0) = 0$ (by direct substitution)

Now, we need to find $g(t)$:

$$y' = 20\sin(t)\, e^{20t}, + \cos(t)\, e^{20t}$$
(by product rule) (2.12)

Then by putting y and y' in the ODE equation (2.10), we have

$$y' - 4y^2 + y + 3t = 20\sin(t)e^{20t} + \cos(t)e^{20t}$$
$$-4sin^2(t)e^{40t} + \sin(t)e^{20t} + 3t \tag{2.13}$$

Finally, we have manufactured an ODE

$$y' - 4y^2 + y = 20\sin(t)e^{20t} + \cos(t)e^{20t}$$
$$-4sin^2(t)\, e^{40t} + \sin(t)e^{20t} \tag{2.14}$$

where $y(0) = 0$ with an exact solution $y(t) = \sin(t)e^{20t}$.

The Existence and Uniqueness of nonlinear ODEs

Before we solve the 1st-order nonlinear ODE, we need to find their existence and uniqueness by using the concept of Lipchitz continuity. The stiffness on ODEs can be regarded as a nonlinear ODE problem with a large Lipschitz constant. The theorem of the existence and uniqueness states that if f is uniformly Lipschitz continuous over some time period which is $0 \le t \le T$, then the unique solution will be obtained through any value α from initial value α. For the explicit methods, it is important to take a reasonably small step size to obtain accurate numerical results.

The theory for the existence of a solution to the initial value problem (IVP) of the 1st-order ODE is in the form:

$$y'(t) = f(t,y) \qquad y(0) = \alpha \tag{2.15}$$

The function (y') is said to be Lipschitz continuous in y over some range of t and y, such that there exists some constant $L>0$ such that

$$|f(t,y) - f(t,y^*)| \leq L|\, y-y'| \tag{2.16}$$

for all y and y' in this range. Lipschitz continuity slightly stronger than continuity which requires that $|f(t,y)-f(t,y^*)| \to 0$ as $y \to y^*$ whereas Lipschitz continuity only requires:

$$|f(t,y) - f(t,y^*)| \leq 0 \text{as } y \to y^* \tag{2.17}$$

For example: Using Taylor expansion, we have

$$f(t,y^*) = f(t,y) + \frac{(y^*-y)f_y(t,y)}{1!} + \frac{(y^*-y)^2 f_{yy}(t,y)}{2!} + O((y^*-y)^3) \tag{2.18}$$

or

$$f(t,y^*) - f(t,y) = (y^*-y)f_y(t,y), \qquad y < \bar{y} < y^* \tag{2.19}$$

by Mean Value Theorem.

Taking bound on the both side

$$|f(t,y^*) - f(t,y)| = |(y^*-y)f_y(t,\bar{y})| \tag{2.20}$$

$$|f(t,y^*) - f(t,y)| = |f_y(t,\bar{y})||y^*-y|, \qquad 0 < L < 1, \tag{2.21}$$

$$|f(t,y^*) - f(t,y)| = L|y^*-y| \tag{2.22}$$

Therefore, we had proved that the Lipschitz constant, $L=|f_y(t,y^*)|$.

Zero stability of Runge-Kutta methods

Linear Multistep Method (LMM) is known to be zero stable and we will proof in this section. Usually, a method is said to have a special type of numerical stability if the global errors can be bounded in term of sum of all the one-step error and has the same asymptotic behaviour as the local truncation error approaches 0. In the ODE theory, it is called as the zero-stability, a notion used to differentiate this from other type of stability that produces the same significance.

LeVeque (2004) mentioned that an r-step linear multistep method (LMM) is said to be zero-stable if the roots of the polynomial $P(\theta)$ satisfy the following conditions:

$|\theta j| \leq 1$ for j = 1,2,3,...

If θj is a repeated root, then $|\theta j| 1$.

In this section, we can conclude that RK-2, RK-4 and RK-5 are all zero-stable methods.

Absolute stability

Basically, the local approximations for many problems can be nullified in the long term are due to the unstable behaviour. A way to exploring what is needed for a method to solve these problems in stable manner is to check the condition of simple linear equation $y' \lambda y =$ to be solved over a single time step. λ is the constant that can be a complex number, however we just focus our attention on this quantity scaled in terms of the step size, which is the complex number $z = h\lambda$.

We defined the stability function as the rational function $R(z) = 1 + zb^T (I-zA)^{-1}e$, where $e \in R$ has every component equal to 1 for a given a Runge-Kutta method. The stability region is a set on the complex plane such that $|R(z)| \leq 1$. When solving the problem $y'=\lambda y$, we have to choose step size, h small enough to ensure that $z=h\lambda$ is in the stability region, and otherwise the sequence of approximations produced in many steps will be unbounded. Then we will interpret it as an unstable behaviour. The stability region for explicit method with $s=p\leq 6$ is shown in figure below.

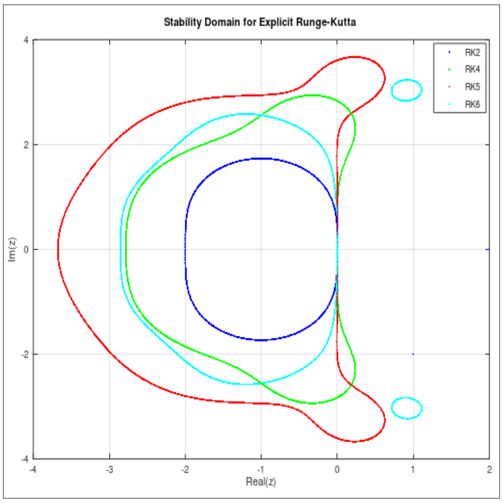

Figure 3.1: Stability domain for explicit runge-kutta methods

In the case of a stiff problem, even though it may not have a linear form but there will be growth factors corresponding to values of *z* which may lie anywhere in the left half-plane. In this particular case, we are also interested to have methods which are "A-stable", in which all complex numbers with negative real part lie in the stability region.

Analysis of Error of the Test Problem

Results and Discussion

Analysis of Error

In this section, we will analyze the error between three different orders of Runge-Kutta methods and exact solution for manufactured solution.

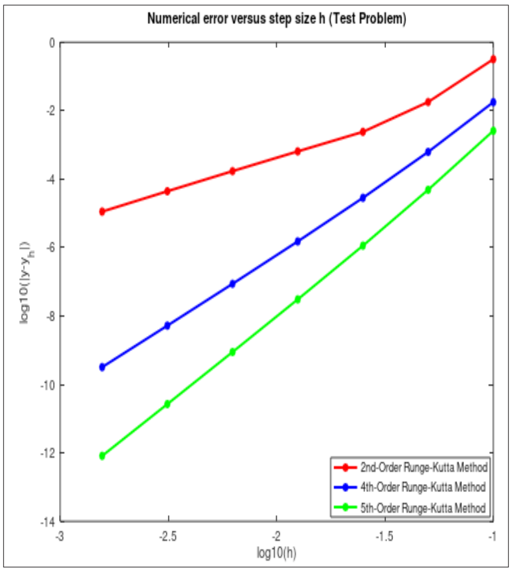

Figure 3.1: Numerical error versus step size h for test problem

In Figure 3.1, the graph was plotted using the logarithm value of base 10 for the numerical error versus the similar logarithm value of base 10 for step size *h*. The graph consists of three different methods used which is 2nd-, 4th-, and 5th-order of Runge-Kutta methods. The red line represents 2nd-order of Runge-Kutta method, the blue line represents 4th-order of Runge-Kutta method and the green line represents 5th-order of Runge-Kutta method. From Figure 3.1, we analyze the global error of the numerical approximations with Runge-Kutta methods along *t*[*a,b*]. By observing the graph, we can clearly see that all the numerical error for the manufactured solution is getting smaller as the step size *h* decreases. When we set up our *h* = 0.0125, the 2nd-, 4th-, and 5th-order of Runge-Kutta methods produce the numerical error of $6.47\text{x}10^{-4}$, $1.51\text{x}10^{-6}$, $3.06\text{x}10^{-8}$, respectively. At *h* = 0.0015625, the numerical error for 2nd-, 4th-, and 5th-order of Runge-Kutta methods which are $1.13\text{x}10^{-5}$, $3.213\text{x}10^{-10}$, $8.18\text{x}10^{-13}$, respectively. From these results, we can conclude that 5th-order of Runge-Kutta methods produce smallest numerical error with same value of step size compared to the 2nd- and 4th-order of Runge-Kutta methods.

Analysis of Error of the Manufactured Solution

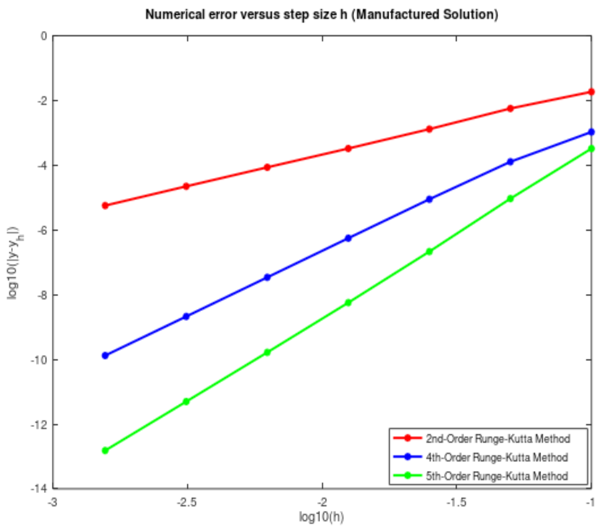

Figure 3.2: Numerical error versus step size h for manufactured solution

In Figure 3.2, the graph was plotted using the logarithm value of base 10 for the numerical error versus the similar logarithm value of base 10 for step size *h*. The graph consists of three different methods used which is 2nd-, 4th-, and 5th-order of Runge-Kutta methods. The red line stands for 2nd-order of Runge-Kutta method, the blue line stands for 4th-order of Runge-Kutta method and the green line stands for 5th-order of Runge-Kutta method. From Figure 3.2, we analyze the global error of the numerical approximations with Runge-Kutta methods along *t*[*a,b*]. By observing the graph, we can clearly see that all the numerical error for the manufactured solution is getting smaller as the step size *h* decreases. When we set up our, *h* = 0.0125, the 2nd-, 4th, and 5th-order of Runge-Kutta methods produce the numerical error of $3.27\text{x}10^{-4}$, $5.54\text{x}10^{-7}$,$5.65\text{x}10^{-9}$, respectively. At *h*=0.025, the numerical error is $1.30\text{x}10^{-3}$, $8.84\text{x}10^{-6}$

and 2.14x10^{-7} for 2nd-, 4th-, and 5th-order of Runge-Kutta methods, respectively. From these results, we can conclude that 5th-order of Runge-Kutta methods produces smallest numerical error with same value of step size compared to the 2nd- and 4th-order of Runge-Kutta methods.

Efficiency Analysis

In this section, we are going to analyze the efficiency of 2nd-, 4th- and 5th-order Runge-Kutta methods using the plot of numerical error versus the CPU time. We obtained the CPU time by recording the total time of calculation in second starting from (n =1) until the end of stopping algorithm (n =N).

Efficiency Analysis of Test Problem

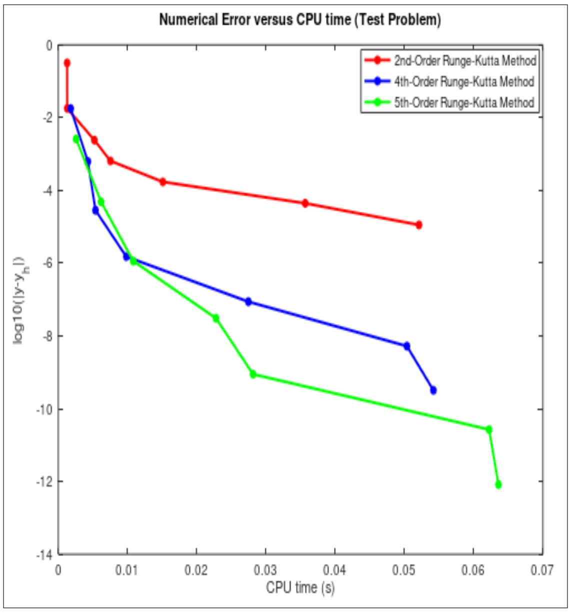

Figure 3.3: Numerical error versus CPU Time for test problem

In Figure 3.3, the graph was plotted using the logarithm value of base 10 for numerical error versus the (central processor unit) CPU time required to complete the computation in seconds (s). The graph consists of three different methods used which is 2nd-, 4th-, and 5th-order of Runge-Kutta methods. The red line is for 2nd-order of Runge-Kutta method, the blue line is for 4th-order of Runge-Kutta method and the green line is for 5th-order of Runge-Kutta method. From the graph above, we clearly can see that the CPU time for each method increase in nonlinear pattern when the numerical error is getting decreases. CPU time is recorded before (n=1) and after (n=N) the loop of time-stepping algorithm. Based on the efficiency result, at step size h=0.05, the required CPU time for 2nd-, 4th-, and 5th-order of Runge-Kutta methods are 1.28x10^{-3}s,4.27x10^{-3}s, and 6.17x10^{-3}s, respectively. At h=0.0125, the required CPU time for 2nd-, 4th-, and 5th-order of Runge-Kutta methods are 7.53x10^{-3}s, 9.82x10^{-3}s, and 2.28x10^{-2}s, respectively. As a conclusion, the CPU time increases when the step size h decreases for all three Runge-Kutta methods.

Efficiency Analysis of Manufactured Solution

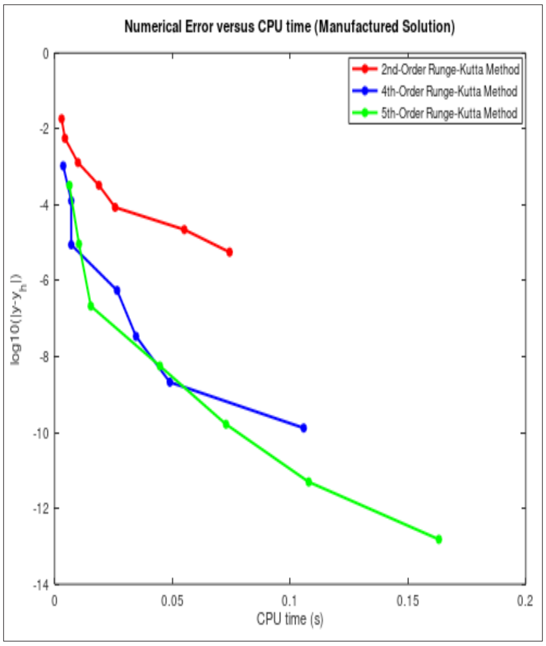

Figure 3.4: Numerical error versus CPU Time for manufactured solution

In Figure 3.4, the graph was plotted using the logarithm value of base 10 for numerical error versus the (central processor unit) CPU time required to complete the computation in seconds (*s*). The graph consists of three different methods used which is 2nd-, 4th-, and 5th-order of Runge-Kutta methods. The red line stands for 2nd-order of Runge-Kutta method, the blue line stands for 4th-order of Runge-Kutta method and the green line stands for 5th-order of Runge-Kutta method. From the graph above, we can clearly see that the CPU time for each method increases in a nonlinear fashion when the numerical error decreases. CPU time is recorded before(n=1) and after (n=N) within the loop of time-stepping algorithm. Based on the efficiency results, at step size h=0.05, the required CPU time for 2nd-, 4th-, and 5th-order of Runge-Kutta methods are 4.41x$10^{-3}s$, 7.08x$10^{-3}s$, and, 1.04x$10^{-2}s$ respectively. As we setting h = 0.0015625, the CPU time for 2nd-, 4th-, and 5th-order of Runge-Kutta methods are 7.43x$10^{-2}s$, 1.06x$10^{-1}s$, and 1.631x$10^{-1}s$, respectively. As a conclusion, the CPU time is increasing when the step size h is getting smaller for all three Runge-Kutta methods.

Convergence Analysis

The setting of the step sizes is also important to verify the order of the convergence for the numerical solution. We consider the y the exact value and we denote y the numerical approximation where it depends the step size h. From here, we can compute the order of convergence k of any numerical methods for ODE is mathematically expressed as:

$$k = \frac{\log \frac{||y - y_h||}{||y - y_{\frac{h}{2}}||}}{log2} \qquad (3.31)$$

where k is the order of convergence, y is the exact solution and y_h is the numerical approximation which depends on the step size h, $\frac{||y-y_h||}{||y-y_{\frac{h}{2}}||}$ is the norm of relative error which depends on the value of h, the step size and the half of, $\frac{h}{2}$ the chosen step size.

Comparison On Efficency Error Between Fixed and Adaptive Step Size

In this section, we will compare the efficiency error between the fixed and adaptive step size of 5th-order Runge-Kutta method for test problem and manufactured solution.

Efficiency Error Between Fixed and Adaptive Step Size for Test Problem

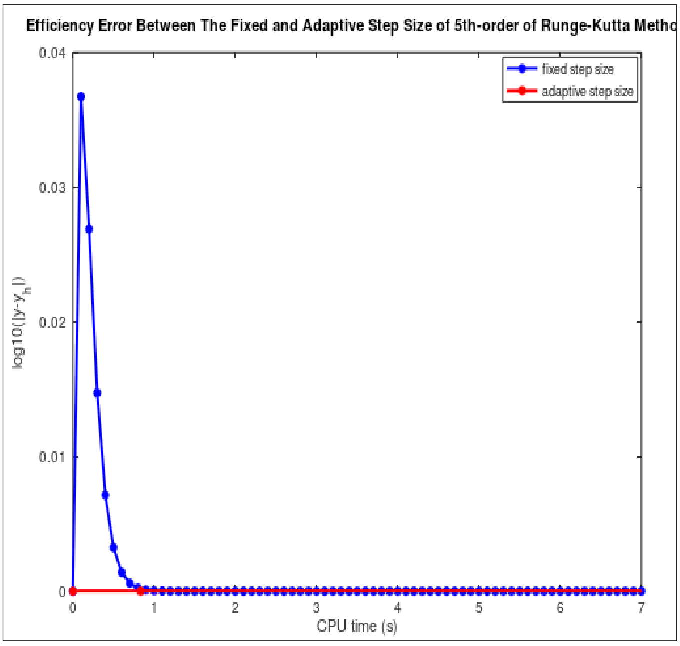

Figure 3.6: Efficiency error between the fixed and adaptive step size of 5th-order of Runge-Kutta method. (Test Problem)

In Figure 3.6, the graph was plotted using the logarithm value of base 10 for numerical error versus the (central processor unit) CPU time required to complete the computation in seconds (s). The graph consists of two different methods used which is 5th-order of Runge-Kutta method with adaptive step size and 5th-order of Runge-Kutta method with fixed step size. The red line is for 5th-order of Runge-Kutta method with adaptive step size, and the blue line is for 5th-order of Runge-Kutta method with fixed step size. From the graph above, we clearly can see that the CPU time for the 5th-order of Runge-Kutta method with adaptive step size is more efficient than 5th-order of Runge-Kutta method with fixed step size. We also clearly see that the numerical error for adaptive step size of 5th-order of Runge-Kutta method is smaller than the fixed step size. Based on the result, at CPU time s=0.0086773, the computed error for fixed and adaptive step size of 5th-order of Runge-Kutta methods are 3.68x10^{-2}, and 8.66x10^{-5}, respectively. As a conclusion, the adaptive step size will result in more efficient and produce smaller numerical error than fixed step size for 5th-order Runge-Kutta method.

Efficiency Error Between Fixed and Adaptive Step Size for Manufactured Solution

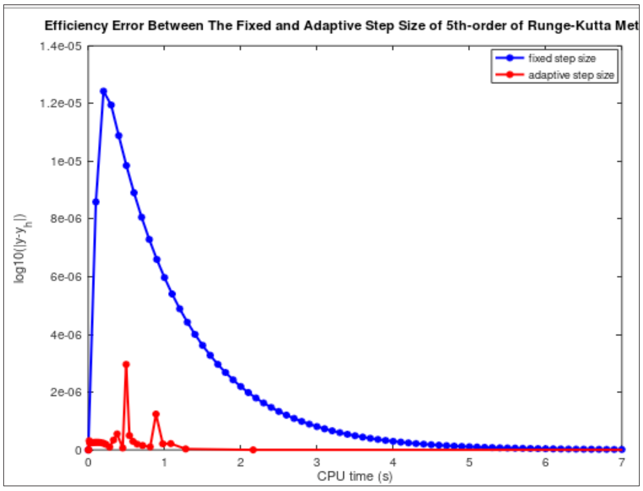

Figure 3.7: Efficiency error between the fixed and adaptive step size of 5th-order of Runge-Kutta method. (Manufactured solution)

In Figure 3.7, the graph was plotted using the logarithm value of base 10 for numerical error versus the (central processor unit) CPU time required to complete the computation in seconds (s). The graph consists of two different methods used which is 5th-order of Runge-Kutta method with adaptive step size and 5th-order of Runge-Kutta method with fixed step size. The red line is for 5th-order of Runge-Kutta method with adaptive step size, and the blue line is for 5th-order of Runge-Kutta method with fixed step size. From the graph above, we clearly can see that the CPU time for the 5th-order of Runge-Kutta method with adaptive step size is more efficient than 5th-order of Runge-Kutta method with fixed step size. We can observe that the numerical error for adaptive step size of 5th-order of Runge-Kutta method is smaller than the fixed step size. Based on the result, at CPU time s=0.11613, the computed error for fixed and adaptive step size of 5th-order of Runge-Kutta methods are 9.60x10^{-6}, and 2.79x10^{-7}, respectively. As a conclusion, the adaptive step size will result in more efficient and produces smaller numerical error than fixed step size for 5th-order Runge-Kutta method.

Conclusion

In this research, the purpose is to study the RK methods when solving 1st-order nonlinear stiff ODEs. We have used two test cases which are the test problem and manufactured solution to determine the accuracy and efficiency of the numerical method. We provided the general formula of 2nd-, 4th-, and 5th-order RK methods with the Butcher tableau. Then, we proposed a test problem and constructed the manufactured solution with certain assumptions for the purpose of numerical illustration. Before that, we verified the existence and uniqueness of the test case by using Lipschitz continuity.

By using RK methods, the numerical schemes of 2nd-, 4th-, and 5th-order are constructed for both two test cases. Then, we verified the RK methods in term of numerical error and the order of convergence for all accuracy by implementing the algorithm using Octave programming language. We also compared the efficiency of the RK methods in term of the plot of numerical error versus the CPU time for both two test cases. Then, we also compared the fixed and adaptive step size for 5th-order RK method in term of the plot on efficiency error versus CPU time graph for both two test cases.

Based on the obtained results, we can conclude that 5th-order Runge-Kutta method is the most efficient for solving the first-order nonlinear stiff ODEs problem. We also can conclude that the adaptive step size is more efficient and produce smaller computed error than the fixed step size of 5th-order Runge-Kutta method.

Acknowledgements

At the end of my thesis, I would like to thank all people who made this thesis possible and conducting this final year project is such an enjoyable experience.

First and foremost, I wish to express my appreciation to Dr Loy Kak Choon, supervisor and Dr Ruwaidiah Binti Idris, co-supervisor of this thesis for giving me many useful advices and guidance with patients. Thank you for inspiring me to have a better understanding my research and giving me many useful information, which helped me to complete this research. I really appreciate it.

Next, my sincere gratitude goes to Dr. Mohamed Saifullah Bin Hussin and Dr. Zabidin Bin Salleh as my coordinators for Final Year Project. All the information of writing a formal thesis was given to us through workshops. These workshops had helped me to prepare my project in a proper way.

Most importantly, I would like to express my deepest gratitude for a constant support, emotional understanding and love that I received from my family especially my mother, Liew Siew Foong. Thank you for their motivational help, support and encouragement for the success of my research. Without their support, this research cannot be completed with a great success.

Finally, I would also like to thank my friends who are under the same supervision with my both supervisors for their encouragement and willingness to share their valuable ideas and knowledge which are very helpful to me for completing this research.

References

Aliyu, B. K., Osheku C. A.,Funmilayo, A. A., & Musa, J. I. (2014). Identifying Stiff Ordinary Differential Equations and Problem Solving Environmetns (PSEs). *Journal of Scientific Research & Reports, 3*(11), 1420-1448.

Balac, S., & Mahe, F. (2013). Embedded Runge-Kutta scheme for step-size control in the interaction picture method.*Computer Physics Communications*, *184*(4), 1211-1219.

Butcher, J. C. (1996). A history of Runge-Kutta methods. *Applied Numerical Mathematics*, *20*(3), 247-260.

Butcher, J. C. (2005). *Numerical Methods for Ordinary Differential Equations Second Edition*. England: John Wiley & Sons Ltd.

Butcher, J. C. (2007). *Runge-Kutta methods*. Scholarpedia, *2*(9), 3147.http://www.scholarpedia.org/article/Runge-Kutta_methods [21 October 2011]

LeVeque, R. J. (2004). *Finite Difference Methods for Differential Equation*. Washington: University of Washington.

Oleg G. (2007). Adaptive Stepsize Numerical Methods for Solving Ordinary Differential Equations.

Richard, L. B., & Douglas, F. J. (2011). Numerical Analysis, Ninth Edition. *Error Control and the Runge-Kutta-Fehlberg Method,* pp. 294-302.

Salau, O., & Ajide, O. (2017). Adaptive Time Steps Runge-Kutta Methods: Comparative Analysis of Simulation Time in Nonlinear and Harmonically

Excited Pendulum and Duffing Oscillators.*British Journal of Applied Science & Technology, 19*(5), 1-13.

Serdar I. (2012). A Runge-Kutta Model Based Adaptive Predictive Controller for Nonlinear Systems. *IFAC Proceedings Volumes*, *45*(17), 151-156.

Stephane B. (2013). High order embedded Runge-Kutta scheme for adaptive step-size control in Interaction Picture method. *Journal of the Korean Society for Industrial and Applied Mathematics*, *17*(4), 238-266.

Syed Hyder, A. M, & Arnold Willem, H. (2013). Adaptive time stepping algorithm for Lagrangian transport models: Theory and idealised test cases, *68*, 9-21.

Thohura, S., & Rahman, A. (2010). Comparison of Numerical Methods for Solving Initial Value Problems for Stiff Differential Equations. *Journal of Bangladesh Mathematical Society*, *30*, 122-132.

Wang Z., Wang Q., & Klinke D.Z. (20160. Simulation Study on Effects of Order and Step Size of Runge-Kutta Methods that Solve Contagious Disease and Tumor Models. *Journal of Computer Science & System Biology, 9*(5), 163-172.

Wen S. (2015). An Introduction to Numerical Computation. *Penn State University, USA*, pp. 173-216.

Zhao, F., & Utkin, V. (1996). Adaptive Simulation and control of variable-structure control systems in sliding regimes. *Automatica*, *32*(7), 1037-1042.