International Journal of Numerical Methods and Applications
Volume 2, Number 1, 2009, Pages 47-57
This paper is available online at http://www.pphmj.com

AN ALGORITHM USING RUNGE-KUTTA METHODS OF ORDERS 4 AND 5 FOR SYSTEMS OF ODEs

NIKOLAOS S. CHRISTODOULOU

TEI of Chalkis
School of Technological Applications (STEF)
Gen. Department of Applied Sciences
Psahna Gr-34400, Greece
e-mail: nikolaos_christodoulou@yahoo.gr

Abstract

This paper deals with an explicit MATLAB algorithm for the implementation of Runge-Kutta method of orders 4 and 5. The running time and maximum errors for the two methods are compared on Rössler system.

1. Introduction

Systems of ordinary differential equations

$$F(x, y, y', ..., y^{(n-1)}) = y^{(n)} \tag{1.1}$$

for vector valued functions,

$$y : \mathbb{R} \to \mathbb{R}^m,$$

in x with $y^{(n)}$ the nth derivative of y, arise in many different contexts including geometry, mechanics, astronomy, engineering and population modeling. Much study has been devoted to the solution of ordinary differential equations. In the case where

2000 Mathematics Subject Classification: 65P20, 65P30, 65L99.

Keywords and phrases: Runge-Kutta, Rössler, numerical solution, system of ODE.

Received June 17, 2009

the equation is linear, it can be solved by analytical methods. Unfortunately, most of the interesting differential equations are nonlinear and, with a few exceptions, cannot be solved exactly. Approximate solutions are arrived at using computer approximations. The method of Runge-Kutta is one of the well-known numerical methods for differential equations. Two members of the family of Runge-Kutta methods are so commonly used that it is often referred to as "RK4" and "RK5". The RK4 method is a fourth-order method, meaning that the error per step is on the order of h^5, while the total accumulated error has order h^4. The RK5 method is a fifth-order method, meaning that the error per step is on the order of h^6, while the total accumulated error has order h^4.

In this paper, we have written a MATLAB code that applies RK4 and RK5 methods to Rössler system. However this code applies to any system of differential equation, in any dimensions, and finds solutions satisfying a given error bound. The advantage of using such an explicit *m*-file rather than MATLAB's built-in solvers like ode45 [6] is that we have better control in the inner workings of the program. We can determine the step size, maximum error level, time limit, etc. We can start with more than one initial point and plot the results. Also, the algorithm is more pedagogical and may be used to illustrate the method of Runge-Kutta for numerical analysis students.

2. The Rössler Attractor

The so called "Rössler" system is credited to Rössler in 1976 [5] and arose from work in chemical kinetics. The system is described with three coupled nonlinear differential equations

$$\begin{cases} \dot{x} = -(y + z), \\ \dot{y} = x + ay, \\ \dot{z} = b + z(x - c). \end{cases} \tag{2.1}$$

While the equations look simple enough, they lead to wonderful trajectories, some examples of which are illustrated in Section 5.

These differential equations in (2.1) define a *continuous-time dynamical system* that exhibits chaotic dynamics associated with the fractal properties of the attractor. Some properties of the Rössler system can be deduced via linear methods such as eigenvectors, but the main features of the system require nonlinear methods such as

Poincarè maps and bifurcation diagrams. The original Rössler paper [5] says that the Rössler attractor was intended to behave similarly to the Lorenz attractor, but also be easier to analyze qualitatively. An orbit within the attractor follows an outward spiral close to the *x*-*y* plane around an unstable fixed point. Once the graph spirals out enough, a second fixed point influences the graph, causing a rise and twist in the *z*-dimension. In the time domain, it becomes apparent that although each variable is oscillating within a fixed range of values, the oscillations are chaotic. This attractor has some similarities to the Lorenz attractor (see [2, 3, 4] and the references therein), but is simpler and has only one manifold. Rössler designed the Rössler attractor in 1976, but the originally theoretical equations were later found to be useful in modeling equilibrium in chemical reactions.

Rössler studied the chaotic attractor with $a = 0.2$, $b = 0.2$ and $c = 5.7$, though properties of $a = 0.1$, $b = 0.1$ and $c = 14$ have been more commonly used since.

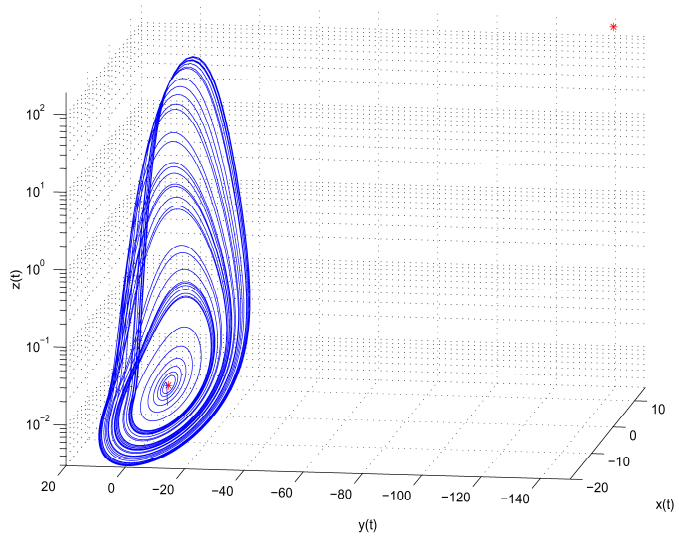

Figure 1. Numerical simulation of the Rössler equations with $a = b = 0.1$ and $c = 14$ using RK5 method.

In order to find the fixed points, the three Rössler equations (2.1) are set to zero and the (x, y, z) coordinates of each fixed point were determined by solving the resulting equations. This yields for a given set of parameter values the general equations of each of the fixed point coordinates:

$$\begin{cases} FP_{+} = \left(\dfrac{c + \sqrt{c^2 - 4ab}}{2}, \dfrac{-c - \sqrt{c^2 - 4ab}}{2a}, \dfrac{c + \sqrt{c^2 - 4ab}}{2a} \right), \\ FP_{-} = \left(\dfrac{c - \sqrt{c^2 - 4ab}}{2}, \dfrac{-c + \sqrt{c^2 - 4ab}}{2a}, \dfrac{c - \sqrt{c^2 - 4ab}}{2a} \right). \end{cases} \tag{2.2}$$

As shown in Figure 1, one of these fixed points resides in the center of the attractor loop and the other lies comparatively removed from the attractor. Theoretical properties of Rössler system, is the subject of dynamical systems and chaos theory [1]. In this paper, what is interesting about Rössler equations is the difficulty of using numerical methods to reach a given error bound.

3. Runge-Kutta Methods of Orders 4 and 5

In this section, we present the numerical methods for the solution of (1.1) where the *vector valued functions* are defined by (2.1). We will mainly concentrate on the case where (2.1) is solved numerically by the 4th and 5th order Runge-Kutta methods. The main purpose is to review the concepts and the structure of these methods.

One member of the family of Runge-Kutta methods is so commonly used that it is often referred to as “RK4” or simply as “the Runge-Kutta method”.

Let an initial value problem be specified as follows:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Then, the RK4 method for this problem is given by the following equations:

$$y_{n+1} = y_n + \frac{1}{6} h(k_1 + 2k_2 + 2k_3 + k_4),$$

$$t_{n+1} = t_n + h,$$

where y_{n+1} is the RK4 approximation of $y(t_{n+1})$, and

$$k_1 = f(t_n, y_n),$$

$$k_2 = f\left(t_n + \frac{1}{2} h, y_n + \frac{1}{2} hk_1\right),$$

$$k_3 = f\left(t_n + \frac{1}{2}h,\ y_n + \frac{1}{2}hk_2\right),$$

$$k_4 = f(t_n + h,\ y_n + hk_3).$$

Thus, the next value (y_{n+1}) is determined by the present value (y_n) plus the product of the size of the interval (h) and an estimated slope. The slope is a weighted average of slopes:

- k_1 is the slope at the beginning of the interval;
- k_2 is the slope at the midpoint of the interval, using slope k_1 to determine the value of *y* at the point $t_n + h/2$ using Euler's method;
- k_3 is again the slope at the midpoint, but now using the slope k_2 to determine the *y*-value;
- k_4 is the slope at the end of the interval, with its *y*-value determined using k_3.

In averaging the four slopes, greater weight is given to the slopes at the midpoint:

$$\text{slope} = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4).$$

The RK4 method is a fourth-order method, meaning that the error per step is on the order of h^5, while the total accumulated error has order h^4.

Note that the above formulae are valid for both scalar- and vector-valued functions, (i.e., *y* can be a vector and *f* an operator).

Similarly the 5th order Runge-Kutta method (RK5) for the same problem is given by the following equations:

$$k_1 = f(t_n,\ y_n),$$

$$k_2 = hf\left(t_n + \frac{h}{2},\ y_n + \frac{k_1}{2}\right),$$

$$k_3 = hf\left(t_n + \frac{h}{4},\ y_n + \frac{3k_1 + k_2}{16}\right),$$

$$k_4 = hf\left(t_n + \frac{h}{2},\ y_n + \frac{k_3}{2}\right),$$

$$k_5 = hf\left(t_n + \frac{3h}{4},\ y_n + \frac{-3k_2 + 6k_3 + 9k_4}{16}\right),$$

$$k_6 = hf\left(t_n + h,\ y_n + \frac{k_1 + 4k_2 + 6k_3 - 12k_4 + 8k_5}{7}\right),$$

$$y_{n+1} = y_n + \frac{7k_1 + 32k_3 + 12k_4 + 32k_5 + 7k_6}{90}.$$

4. MATLAB Code

The following MATLAB *m*-file solves the Rössler attractor using Runge-Kutta method of order 4 or 5. By changing the first few lines, we can give the equation, initial point(s), step size, maximum running time, maximum error and the order of the method. The program will halt if the error limit is satisfied or time limit exceeded, whichever comes first. The given system must be an autonomous system of equations.

```
% Solution of an autonomous ODE y' = f(y) (Rössler attractor)
% using RK4 and RK5 methods where y(y1, y2, ..., yn)
% prints the solution to matrix y.

tic; clear;
f = inline ( '[-y(2) - y(3); y(1) + 0.1 * y(2); 0.1 + y(3) * (y(1) - 14)]', 'y' ); % equation
order = 5;              % The order of the Runge-Kutta method must be 4 or 5
y0 = [1 1 0]';          % This is the initial point
k = 5000;               % k+1 is the number of columns of the solution matrix y
dt = 0.05;              % The step size of t for the numerical method
error = 10^(-4);        % This is the maximum error of y
plotrange = [-20  15  -150  20  0  150]; % The region to be plotted
timelimit = 3000;    % The maximum time allowed for the program to run.
dim = size(y0, 1);    % The dimension of the system
m = size(y0,2);        % The number of initial points
tmax = timelimit/4;

fprintf  ('This code uses Runge-Kutta method of order  %-2.0f\n',  order)
fprintf  ('starting with the initial point  (%2.2f   %2.2f   %2.2f\n',  y0'))
```

```
fprintf  ('from time t=0  to  t = %-3.0f', k * dt)
fprintf  ('The stepsize is Δ  t = %2.12g\n', dt)
fprintf  ('It will continue until the error falls below %2.12g', error)
fprintf  ('or time limit %3.0f',  timelimit)
fprintf  ('seconds is exceeded\n\n\n')
for q = 1 : m
    y = zeros (dim, k + 1);
    y(:, 1) = y0 (:, q);
    yprevious = repmat (inf, dim, k + 1);
    ttt = 0;
    e = inf;
    p = 0;
   while ttt<tmax & e>error
        n = 2^p;
        h = dt/n;
         ta = 0;
         ya = y0 (:, q);
         t0 = clock;
         for j = 1: n * k
             if order = 4              % This is the 4th order RK method
             k1 = h * f  (ya);
             k2 = h * f  (ya + k1/2);
             k3 = h * f  (ya + k2/2);
             k4 = h * f  (ya + k3);
             ya = ya + (k1 + 2 * k2 + 2 * k3 + k4)/6;
             ta = j * h;
          else if   order = 5        % This is the 5th order RK method
             k1 = h * f  (ya);
             k2 = h * f  (ya + k1/2);
             k3 = h * f  (ya + (3 * k1 + k2)/16);
             k4 = h * f  (ya + k3/2);
             k5 = h * f  (ya+ (-3 * k2 + 6 * k3 + 9 * k4)/16);
             k6 = h * f  (ya + (k1 + 4 * k2 + 6 * k3-12 * k4 + 8 * k5)/7);
             ya = ya + (7 * k1 + 32 * k3 + 12 * k4 + 32 * k5 + 7 * k6)/90;
             ta = j * h;
          else fprintf  ('The order of the method must be 4 or 5\n')
             break
          end
          if mod  (j, n) = 0;
```

```
59              i = j/n;
60              y  (:, i+1) = ya;
61            end
62        end
63        e = max (max (abs (y-yprevious)));
64        yprevious = y;
65        fprintf  ('The step size is dt/%-2.0f',  2^p)
66        fprintf  (' = %2.12g\n', h)
67        fprintf  ('The estimated error < %2.12g\n', e)
68        ttt = etime (clock, t0);
69        fprintf  ('time in seconds = %2.1f\n\n', ttt)
70        if e<error
71             fprintf  ('The error limit is satisfied\n\n')
72        elseif  ttt>tmax
73            fprintf  ('Time limit exceeded\n\n')
74        end
75        p = p + 1;
76     end
77     figure (1)
78     plot 3 (y(1, :), y(2, :), y(3, :));
79     hold on
80     a = 0.1; b = 0.1; c = 14;
81     A1 = c + sqrt (c^2-4 * a * b); A2 = -c-sqrt (c^2-4 * a * b);
82     A3 = c-sqrt (c^2-4 * a * b); A4 = -c + sqrt (c^2-4 * a * b);
83     plot 3 (A1/2, A2/(2 * a), A1/(2 * a),  'r*')     % plots fixed points
84     hold on
85     plot 3 (A3/2, A4/(2 * a), A3/(2 * a), 'r*')      % plots fixed points
86     axis (plotrange);
87     grid on;
88  end
89  fprintf  ('The total time in seconds is %2.1f\n', toc)
```

5. Numerical Results of the Two Methods

We study the Rössler chaotic attractor (2.1) with $a = 0.1$, $b = 0.1$ and $c = 14$. The program in Section 4, has been run separately for orders 4 and 5 with initial condition $y_0 = (1\ 1\ 0)$ and step size 0, 05. The resulting errors and program runtime is given below:

RK4 Method			RK5 Method		
Step Size	Error	Time	Step Size	Error	Time
0.025	35.7235940083	25.0	0.025	4.32464747225	10.6
0.0125	21.9453774473	44.9	0.0125	0.114162427377	21.3
0.00625	1.07721196989	108.7	0.00625	0.00373384751883	41.7
0.003125	0.0658815896626	115.8	0.003125	0.000118970270334	82.7
0.0015625	0.00418440282466	113.6	0.0015625	3.91436761227e-006	158.2
0.00078125	0.000263350832215	226.5			
0.000390625	1.68170355241e-005	443.7			

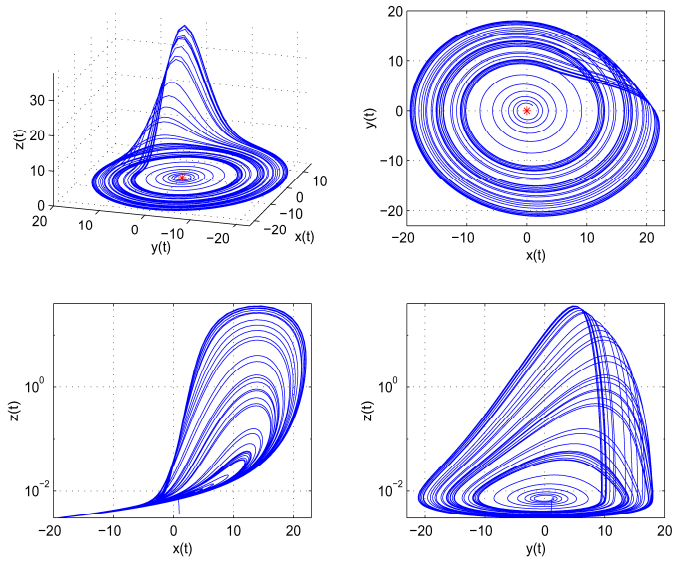

Figure 2. Numerical simulation of the Rössler equations with $a = b = 0.1$ and $c = 14$ using RK4 method.

The graph of the resulting solutions is given in Figure 2. The graphs for the two methods are identical. We see that the fixed point FP_- given by (2.2) is located in the middle of the attractor and is a saddle-focus with an unstable 2D manifold (an unstable spiral mainly in the *x*-*y* plane) when the trajectory settles down onto a chaotic attractor. The FP_+ given by (2.2) is outside of the region of the attractor.

The nonlinearity $z(x-c)$ becomes active when the trajectory leaves the *x*-*y* plane. The trajectory thus visits the neighborhood of the FP_+ - also a saddle-focus - whose 1D unstable manifold sends the trajectory along the 1D stable manifold of the $FP\_$. A new cycle can then occur. With these parameter values of *a*, *b* and *c*, the trajectory thus describes a chaotic attractor.

As shown in the general plots of the Rössler attractor above, one of these fixed points resides in the center of the attractor loop and the other lies comparatively removed from the attractor. This attractor has some similarities to the Lorenz attractor, but is simpler and has only one manifold.

6. Conclusions

The nonlinear systems of differential equations require using computer algebra systems.

Although package programs exist to solve such systems, an explicit routine has certain advantages:

- Inner workings of the numerical methods will be clear, especially for students.
- Parameters like maximum running times, step size, etc. can be controlled.
- Different methods can be contrasted and compared

In this paper, we implemented Runge-Kutta methods of orders 4 and 5 by a MATLAB *m*-file, and compared them on the Rössler equation. For smaller step sizes, Runge-Kutta method of order 5 clearly gives smaller errors for a given running time. In future, similar programs can be written to compare 1-step and multi-step methods.

References

[1] K. T. Alligood, T. D. Sauer and J. A. Yorke, Chaos: An Introduction to Dynamical Systems, Springer-Verlag, 1996.

[2] N. S. Christodoulou, Discrete Hopf bifurcation for Runge-Kutta methods, Appl. Math. Comput. 206 (2008), 346-356.

[3] N. S. Christodoulou, Derivation of a necessary condition to stop merging or crossing of trajectories in numerical simulations, Advances in Differential Equations and Control Processes 2 (2008), 113-133.

[4] N. S. Christodoulou, Bifurcation and spurious solutions using numerical methods, Advances in Differential Equations and Control Processes 2 (2008), 99-112.

[5] O. E. Rössler, An equation for continuous chaos, Physics Letters A 57(5) (1976), 397-398.

[6] L. F. Shampine and M. W. Reichelt, The MATLAB ode suite, SIAM J. Sci. Comput. 18 (1997), 1-22.