WARSAW UNIVERSITY OF TECHNOLOGY

FACULTY OF POWER AND AERONAUTICAL ENGINEERING

COMPUTER SCIENCE 2 REPORT

Rössler attractor

Student: Nectarios Kisiigha

Album number: 288701

Date of submission: 30/04/2019

Abstract

This report deals with an explicit algorithm for the implementation of Runge-Kutta method of order 4 on the Rössler system.

1. Introduction

Systems of ordinary differential equations

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

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 modelling. Much study has been devoted to the solution of ordinary differential equations. In the case where 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. 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" but I will use the RK4 method in this report. 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.

2. The Rössler Attractor

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

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

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

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 says that the Rössler attractor was intended to behave similarly to the Lorenz attractor, but also be easier to analyse 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 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 modelling equilibrium in chemical reactions.

Rössler studied the chaotic attractor with $a = 0.2$, $b = 0.2$ and $c = 5.7$ and I've studied the chaotic attractor with the same values of parameters.

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 centre 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. In this paper, what is interesting about Rössler equations is the difficulty of using numerical methods to reach a given error bound.

3. The Runge-Kutta Method of Order 4

In this section, I present the numerical method for the solution of (1.1) where the *vector valued functions* are defined by (2.1). I will mainly concentrate on the case where (2.1) is solved numerically by the 4th order Runge-Kutta

method. The main purpose is to review the concepts and the structure of the 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).

4. Native code

The following text-file solves the Rössler attractor using Runge-Kutta method of order 4.

Runge-Kutta header file used in the code:

```
/* Runge-Kutta routine (rk4.h)
** This header file is for resolving a rank-1 ordinary differential
equations.
** The number of state variables of each equation is 'N'.
** This time, the Runge-Kutta method was used.
** Since using computer simulation cannot resolve analytically
** the differential equation 'dx(t) / dt = F(x(t), t),'
** the time 't' should be discretized.
** The order 4 Runge-Kutta method is as follows:
** d1 = hF(x(t), t),
** d2 = hF(x(t) + d1 / 2, t + h / 2),
** d3 = hF(x(t) + d2 / 2, t + h / 2),
** d4 = hF(x(t) + d3, t + h),
** x(t + h) = x(t) + d1 / 6 + d2 / 3 + d3 / 3 + d4 / 6,
** where 'h' is the size of discretized time, and is the Runge-Kutta
step.*/
#define _CRT_SECURE_NO_WARNINGS
#include <stdlib.h>

double *vector(int N) /* Allocating vector region */
{
  /*** Dynamically allocating an array using "malloc".
  ** The size of the array is 'N'.
  */
  return (double *)malloc(N * sizeof(double));
}

void free_vector(double *v) /* Releasing vector region */
{
  /* Releasing the region of the pointer 'v'. */
  free(v);
}

void copy_vector(int N, double a[], double b[]) /* Copying vectors */
{
  int i;
  /* Copying the vector 'a' to the vector 'b' */
  for(i = 0; i <= N - 1; i++) b[i] = a[i];
}
/*** Putting the Runge-Kutta step 'h,' the rank of the differential
equation N,
** the function 'dXdt' which describes the differential equation,
** and the vector 'X0' which contains the state variable at time 't'
```

```
** to get the state variable 'X1' at time 't + h'.
** 'dXdt,' which type is 'foo(double t, double X[], double dXdt[])' should
be
** defined by user.*/

 void rk4(double h, int N, void(*dXdt)(double t, double X[], double
dXdt[]),double t, double X0[], double X[]) /* Runge-Kutta */
{
  int i;
  /*** Dynamically allocating arrays for 'double d1[N],' 'double d2[N],'
  ** and 'double d3[N]'.  */
  double *d1 = vector(N),*d2 = vector(N),*d3 = vector(N);

   /** Dynamically allocating arrays for 'double Xa[N],' and 'double
X[N]'.*/
  double *Xa = vector(N),*dX = vector(N);
  /* d1 = hF(x(t), t) */
  dXdt(t, X0, dX);
  for(i = 0; i <= N - 1; i++)
    {
      d1[i] = h * dX[i];
      Xa[i] = X0[i] + 0.5 * d1[i];
    }
  /* d2 = hF(x(t) + d1 / 2, t + h / 2) */
  dXdt((t + (0.5*h)),Xa,dX);
  for(i = 0; i <= N - 1; i++)
    {
      d2[i] = h * dX[i];
      Xa[i] = X0[i] + 0.5 * d2[i];
    }
  /* d3 = hF(x(t) + d2 / 2, t + h / 2) */
  dXdt(t + 0.5 * h, Xa, dX);
  for(i = 0; i <= N-1; i++)
    {
      d3[i] = h * dX[i];
      Xa[i] = X0[i] + d3[i];
    }
  /* x(t + h) = x(t) + d1 / 6 + d2 / 3 + d3 / 3 */
  dXdt(t + h, Xa, dX);
  for(i = 0; i <= N - 1; i++)
    X[i] = X0[i] + (d1[i] + d2[i] * 2 + d3[i] * 2 + h * dX[i]) / 6.0;
  /* Releasing the regions of arrays */
  free_vector(d1); free_vector(d2); free_vector(d3);
  free_vector(Xa); free_vector(dX);
}
```

Below is the 'rossler.h' file used in running the code.

```
/*Rossler.h */
/* The Rossler equations (rossel.h) */
/*
** This header file describes function type of the Rossler equations
** for the Runge-Kutta routine (rk4.h).
** The Rossler equations derived by simplifying of convection rolls arising
** in the atmosphere, and described as ordinary differential equations
** that have 3 variables:
** dx / dt = -y - z,
** dy / dt = x + ay,
** dz / dt = b + zx - zc,
** ,where a, b, and c are parameters, and they are set to
** as follows:
** a = b = 0.2, and c = 5.7.
```

```
** The Rossler equations exhibit sensitive dependence on initial
conditions.*/


/* The notation of functions and variables are consistent with 'rk4.h'. */
#define _CRT_SECURE_NO_WARNINGS
#define dxdt dXdt[0]
#define dydt dXdt[1]
#define dzdt dXdt[2]
#define x X[0]
#define y X[1]
#define z X[2]

/* The Rossler equations */
void rossler(double t, double X[], double dXdt[])
{
  double a=0.2,b=0.2,c=5.7;
  dxdt = -y - z;
  dydt = x + a*y;
  dzdt = b + z*x - z*c;
}
```

Below is the source file with the above header files included:

```
/*NECTARIOS_KISIIGHA_CS2_PROJECT*/
/*STUDENT NO: 288701*/

/*The Rossler attractor*/
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include "rk4.h"
#include "rossler.h" /* rossler(double t, double X[], double dXdt[]) */
#define N 3 /*dimension of state variable*/
#define h 0.01 /*Runge-Kutta step*/
#define T 10000 /*Numbers of caculation of Runge-Kutta*/

int main()
{
  int t, i;
  /* Dynamically allocating arrays for 'double X0[N],' and 'double X1[N]'.
*/
  double *X0 = vector(N),*X1 = vector(N);
  /* Initial setting */
  X0[0] = 10.0;
  X0[1] = 20.0;
  X0[2] = 30.0;
  /*Main part*/
  for(t=0; t<= T-1; t++)
    {
      for(i=0; i<=N-1; i++)
      {
        printf("%f", X0[i]);
        if(i == N - 1) putchar('\n');
        else putchar(' ');
      }
      /*** Putting the function 'rossler' and the state variable 'X0' at
time
      ** 'h*t' to get the state variable 'X1' at time 'h*(t+1)'.*/
      rk4(h,N, rossler, h*t, X0, X1);
      copy_vector(N, X1, X0);
```

```
        }
    return 0;
}
/*releasing the definitions */
#undef dxdt
#undef dydt
#undef dzdt
#undef x
#undef y
#undef z
```

5. Graphs

Since the winbgi2 header and resource files provided for plotting do no plot 3D models, the plot below was obtained with matlab plotting and gnuplot.

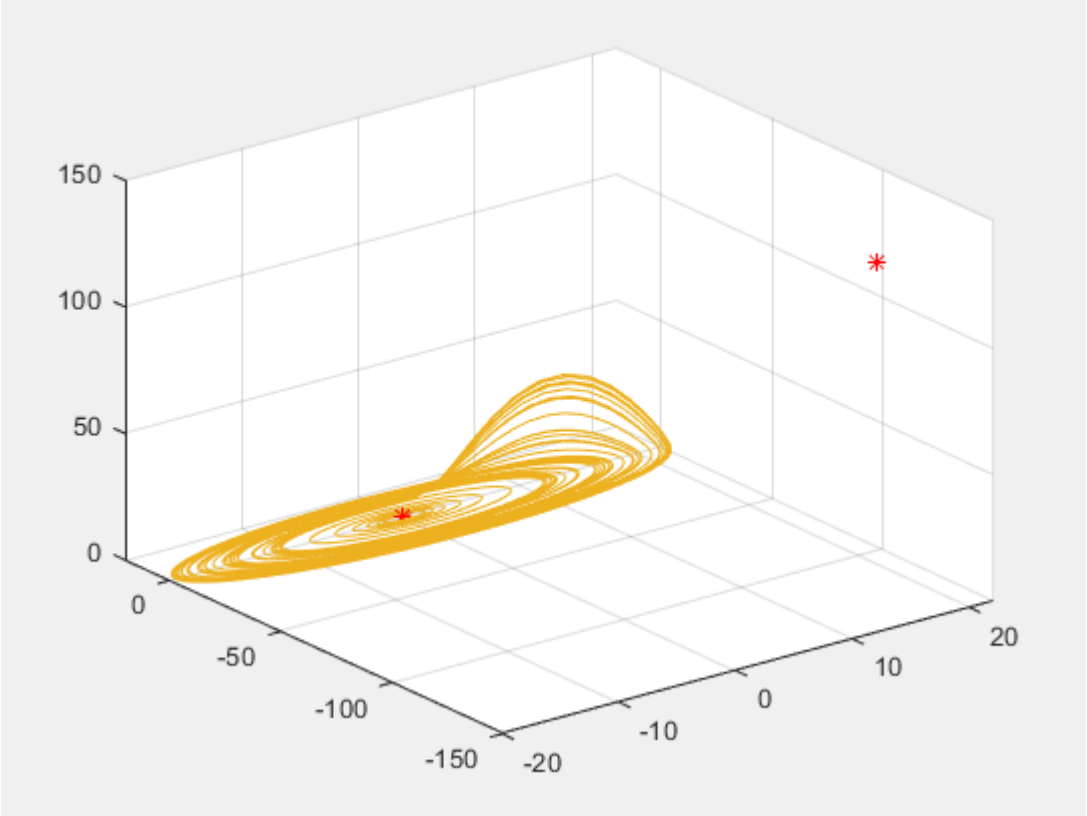

Figure1: Numerical simulation of the Rössler equations with a = b = 0.2 and c = 5.7 using RK4 method.

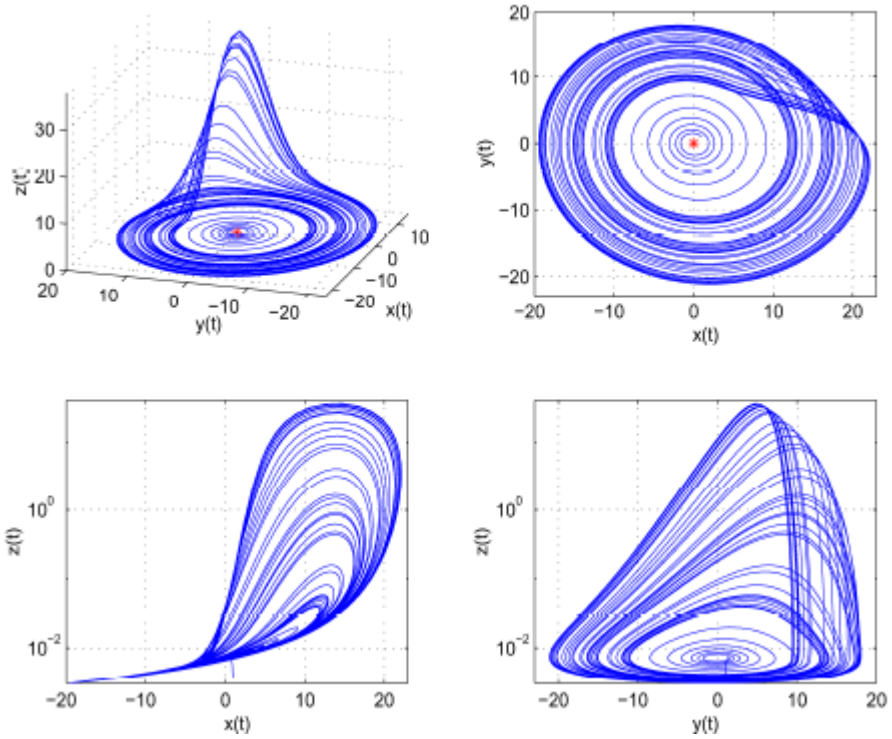

Figure 2: Numerical simulation of the Rössler equations with a = b = 0.2 and c = 5.7 using RK4 method.

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 are clear.
- Parameters like maximum running times, step size, etc. can be controlled.

7. References

- O. E. Rössler, Phys. Lett. 57A, 397-398 (1976)
- N. S. Christodoulou, Discrete Hopf bifurcation for Runge-Kutta methods, Appl. Math. Comput. 206 (2008), 346-356.
- 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.
- An Interactive Plotting Program Thomas Williams & Colin Kelley Version 5.2 organized by: Ethan A Merritt and many others