Numerical Approximations in Differential Equations
The Runge-Kutta Method

by Ernest Ngaruiya

May 15 2007

Abstract

In this paper, I will discuss the Runge-Kutta method of solving simple linear and linearized non-linear differential equations. I start by stating why the Runge-Kutta method is ideal for solving simple linear differential equations numerically in comparison to more elementary methods. I will then proceed to explain what steps the method actually carries out in solving the differential equation along with the matlab code I used to write a simple Runge-Kutta solver and the output of the code, given some basic differential equations. I will then end by discussing the pitfalls of this way of solving differential equations.

1 Introduction

Any nth order linear ordinary differential equation can be expressed in the form of a system of first order linear ODEs. Numerical approximation can be applied to effectively analyse differential equations that may not be easy to solve by traditional methods. Once any ODE has been expressed as a linear system, any of the methods of linear algebra can be applied on the system. The Runge-Kutta method is a far better method to use than the Euler or Improved Euler method in terms of computational resources and accuracy. All these methods use a fixed step size, but there are other methods that use a variable step size (though not neccessarily better in all circumstances). While essentially the Euler methods are simple Runge-Kutta methods, I, like has now become common, refer to the fourth-order Runge-Kutta method as the Runge-Kutta method.

2 The Euler and Improved Euler methods

For an initial value problem

$$dy/dx = f(x, y), y(a) = y_0$$

that has a unique solution $y(x)$ on the closed interval $[a, b]$ and given that $y(x)$ has a continous second derivative on that interval, then there exists a constant C such that for the approximations y_i to the actual values $y(x_i)$

computed using the Euler's method with step size $h > 0$,

$|y_n - y(x_n)| \leq Ch$

For n = 1,2,3, . . . ,k

for the Euler method, and

$|y_n - y(x_n)| \leq Ch^2$

for the improved Euler method. Here, h is the step size. This implies that the error is of order h in the Euler method and order h^2 in the improved Euler method. The proof can be found in the book, *Ordinary Differential Equations* by G. Birkhoff and G.C. Rota. On the other hand, the Runge-Kutta method is a fourth-order method (Runge-Kutta methods can be modified into methods of other orders though). The Euler methods suffer from big local and cumulative errors. The improved Euler method and the Runge-Kutta method are predictor-corrector methods and are more accurate than the simple Euler method.

3 The Runge-Kutta Method

This method uses the simple fact that, for a given actual change in the output y, we can use the fundamental theorem of calculus to express the change in the form of an integral

$y(x_{n+1}) - y(x_n) = \int_{x_n}^{x_{n+1}} y'(x)\,dx = \int_{x_n}^{x_{n+h}} y'(x)\,dx$

We can then use Simpson's rule for numerical integration

$y(x_{n+1}) - y(x_n) \approx \frac{h}{6}\left(y'(x_n) + 4y'(x_n + \frac{h}{2}) + y'(x_{n+1})\right)$

$y_{n+1} \approx y_n + \frac{h}{6}\left(y'(x_n) + 2y'(x_n + \frac{h}{2}) + 2y'(x_n + \frac{h}{2}) + y'(x_{n+1})\right)$

I used matlab to write out some code that implements an algorithm that uses the Simpson's rule to approximate solutions for a given number of steps. Using the code, one can solve any nth order linear ODE as long as you specify the initial conditions. I have shown that by attempting to approximate the solutions to a first order ODE and a second order ODE. The code is shown below:

```
function [T,X] = hrungekutta(t,x,t1,n,f)
T = t;
X = x';
h = (t1 - t)/n;
for i = 1:n;
    k1 = f(t,x);
    k2 = f(t + h/2, x + k1*h/2);
    k3 = f(t+h/2, x + k2*h/2);
    k4 = f(t+h, x + k3*h);
    x = x + h*(k1 + 2*k2 + 2*k3 + k4)/6;
    t = t+h;
    T = [T;t];
    X = [X;x'];
end
```

This function takes the form $f(t_{initial}, [x_0; y_0], t_{final}, steps, @ode)$

I implemented ODEs as functions. So it can take in any ODE as long as you express it in the form of a linear system and you specify its initial values. Note that in Matlab, to input a function to another function, you must use the @ sign in front of the input function's name.

For example, the function ddxeqnegx is the ode $x'' = -x$.
In this case, you will input *@ddxeqnegx* .

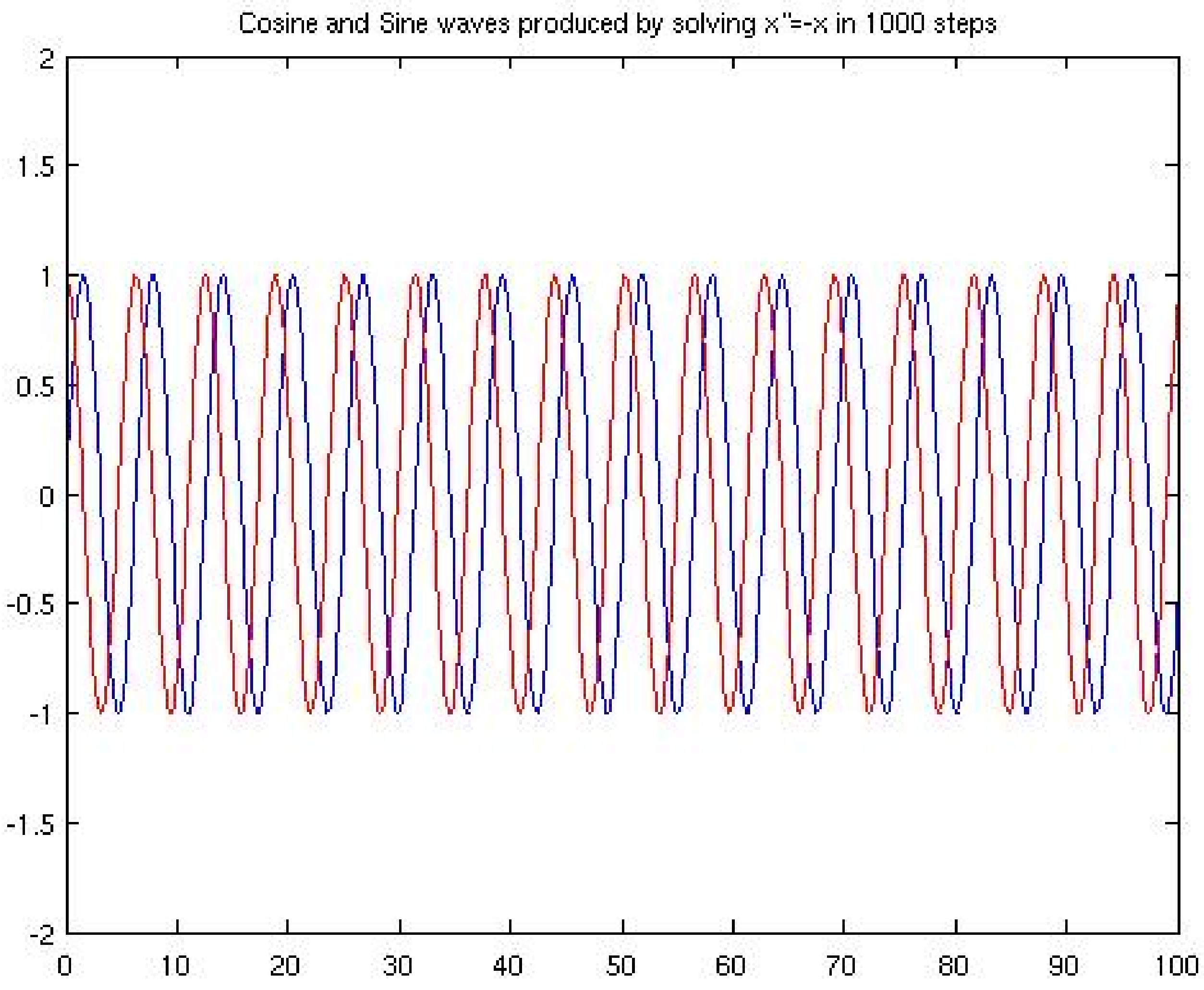
Example: *[T,X] = hrungekutta(0,[0;1],5,10,@ddxeqnegx)*

This solves $x'' = -x$ whose general solutions are of the form $[\sin(t), \cos(t)]$ for t running from 0 to 5 where the initial values are $x = 0, y = 1$, since you change $x'' = -x$ to the system of first order ODEs $x' = y, y' = -x$

The code for the input ODE is:

```
function X = ddxeqnegx(t,x)
X=x;
X(1) = x(2);
X(2) = -x(1);
```

A graph of the solution is shown in the next page:



An example of a first order ODE is *[x,y]=hrungekutta(0,[0],1,10,@pidiff)* which approximates the solution to $\frac{dy}{dx} = \frac{4}{(1+x^2)}$, with the initial value $y_0 = 0$. Note that the general solution to this differential equation is $y = \frac{4}{arctan(x)}$. Therefore, $y_1 \approx \pi$. Outputs for a ten step Runge-Kutta method are shown below.

```
x =

         0
    0.1000
    0.2000
    0.3000
    0.4000
    0.5000
    0.6000
    0.7000
    0.8000
    0.9000
    1.0000


y =

         0
    0.3987
    0.7896
    1.1658
    1.5220
    1.8546
    2.1617
    2.4429
    2.6990
    2.9313
    3.1416
```

With 1000 steps, I got $\pi = 3.141592653589791$

4 Pitfalls in the Runge-Kutta method and other numerical methods

There are a number of problems faced by the Runge-Kutta method. While I will not go into the details here, I will use an example equation to illustrate a problem that one could face.

Consider the equation

$\frac{dy}{dx} = 5y - 6e^{-x}, y(0) = 1$

Its general solution is

$$y(x) = e^{-x} + Ce^{5x}$$

With the initial condition $y(0) = 1$, then $C = 0$

A small error in the evaluation of the output leads to a non-zero value of C. Since the exponent is big, a very small error in approximation (which is almost inevitable) leads to a huge error in the output.

Acknowledgement

I would like to thank my instructor Vera Hur for great suggestions on how to write this paper and putting me in the same group as Lisa Danz, Marcelo Alvisio, Laura Evans and Oleg Golberg who are all interested in Numerical Methods of approximating differential equations. I would also like to thank Josh Nichols-Barrer, who showed us some really neat problems in math.

References

[1] Birkhoff, Garrett and Rota, Gian-Carlo. *Ordinary Differential Equations* 4th ed. (John Wiley & Sons, New Jersey, 1989)

[2] Edwards, Henry and Penney, David. *Elementary Differential Equations with Boundary Value Problems*4th ed. (Prentice Hall, 1999)

[3] http://en.wikipedia.org/wiki/Runge-kutta