Numerical Methods in Differential Equations

Connor Lukan

April 2024

1 Introduction

Over the course of the semester, we have learned how to solve many types of ODEs. We learned how to solve homogenous and non-homogeneous equations and systems of any order, given that it is linear. However, we discussed the difficulties in trying to solve non-linear differential equations and systems. This project will illustrate the use numerical approximations in solving ODEs, and at the end I will demonstrate the use of numerical methods in my simulation software.

2 What we want

1. To be clear with the error associated with using the approximation
2. Scalable with the order of the equation or system
3. Want to be able to choose the amount of error at the possible expense of a bigger computation.
4. Works on any given differential equation or system of differential equations.

3 The most simple: Euler's Method

Consider the initial value problem $\frac{dy}{dt} = f(t, y), y(t_0) = y_0$. By the existence and uniqueness theorem, there exists a function $y(t)$ to satisfy the differential equation if f_y and f are both continuous on some rectangle R with the point (t_0, y_0).

4 Errors

Two types of errors:

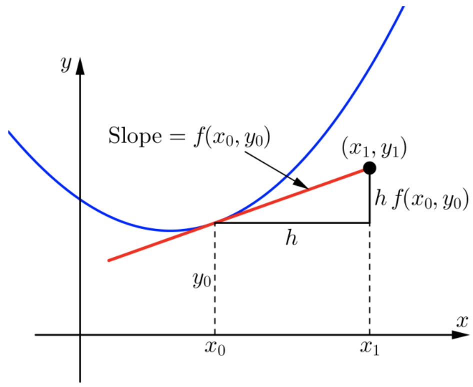

Figure 1: Caption for the image.

1. Truncation
   This error is caused by the use of an approximating formula. Consider the IVP $\frac{dy}{dt} = f(t, y), y(t_i) = y_i$. Suppose we use the function $\phi(t)$ to approximate about the initial condition.
   (a) Local
   Error caused by computing the next time step
   (b) Global
   Error caused by computing multiple steps
2. Round off
   This error is due to rounding to a certain decimal point. Computers do this all of the time due to the fact that floating point numbers must take up a finite amount of space.

5 Error in Euler's

Consider:

$\phi(t) = y(t_i) + (t - t_i)f(t_i, y(t_i))$

$y(t) = y(t_i) + (t - t_i)y'(t_i) + \sum_{i=2}^{\infty} \frac{y^{(i)}(t_i)(t-t_i)^i}{i!}$
$= y(t_i) + (t - t_i)f(t_i, y(t_i)) + \sum_{j=2}^{\infty} \frac{y^{(j)}(t_i)(t-t_i)^j}{j!}$
Notice that $\phi(t)$ is exactly the order 1 Taylor polynomial for y about $t = t_i$. Therefore, by Taylor's remainder theorem, there exists a c $\epsilon(t_i, t_i + h)$ such that:
$\phi(t_i + h) + \frac{y''(c)h^2}{2!} = y_{i+1}$
This implies local truncation is $O(h^2)$.

6 Taylor series

$T_n(t) = \sum_{i=0}^{n} \frac{y^{(i)}(t_0)}{i!}(t-t_0)^i$

One can compute the i^{th} derivative of y via the $(i-1)^{th}$ derivative of f and therefore use this for any first order differential equation. If one would like to approximate the solution to a k^{th} order differential equation, they can change it into a system of k differential equations, and do the multivariable Taylor series expansion from there when deriving the coefficients.

If we know that the initial point is $y(t_i) = y_i$, and the first k many derivatives of y about (t_i, y_i) exist, then $y_{i+1} = y(t_i + h)$ can be approximated with $T_k(t_i + h)$.

It follows that the local truncation error from the approximation is $O(h^{k+1})$.

Here we have it good. Taylor's remainder theorem gives great clarity in the error of our approximation, and one may easily change the complexity of the approximation given a certain requirement for error.

However, there are a few things to consider:

1. Functions with complicated higher order derivatives Consider the differential equation
   $y' = \sqrt{t^3y^2 + y^3t^2}$. In order to have error $O(h^5)$, one would need to compute the third derivative of y' with respect to t.

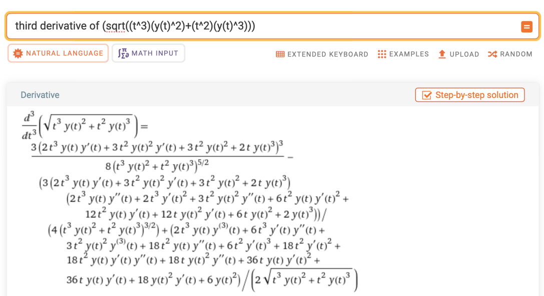

Figure 2: Undefined on the line $y = -t$

2. Functions whose higher order derivatives do not exist in the interval of interest.

7 Improved Euler's

Remember how the 2nd order taylor polynomial about t_i is:

$T_2(t_i + h) = y(t_i) + hf(t_i, y(t_i)) + \frac{h^2}{2!}(f_t + f_y f)$
Consider the approximating function:
$\phi(t) = y(t_i) + \frac{(t-t_i)}{2}(f(t_i, y_i) + f(t, y_i + (t - t_i)f(t_i, y_i)))$

We know that to compute $\phi(t_i + h)$, we must also compute
$f(t_i + h, y_i + hf(t_i, y_i)) = f(t_i, y_i) + \nabla f \cdot h, hf(\vec{t_i}, y_i)$
$= f(t_i, y_i) + hf_t(t_i, y_i) + hf(t_i, y_i)f_y(t_i, y_i)$

From this,
$\phi(t_i + h) = y(t_i) + \frac{h}{2}f(t_i, y_i) + \frac{h}{2}f(t_i, y_i) + \frac{h^2}{2}(f_t + f(t_i, y_i)f_y)$

$$= y(t_i) + hf(t_i, y_i) + \frac{h^2}{2!}(f_t + f(t_i, y_i)f_y)$$

Notice how this is in the form of an order 2 Taylor polynomial. By Taylor's remainder theorem, the local truncation error is $O(h^3)$.

8 Runge-Kutta for 1^{st} order differential equations

Unfortunately, I can only motivate that there exist coefficients for any Runge-Kutta order.

I shall instead demonstrate the procedure for finding RK-N coefficients by example with RK-3. Hopefully you can see enough of a pattern.

Time complexity of Runge-Kutta.
If n is the Runge-Kutta order, then each time step takes $\Theta(n^2)$ work.

Can we extend this idea into higher order differential equations and systems of differential equations?
Perhaps one should consider transforming the k^{th} order differential equation into a system of k many first order equations.

9 Runge-Kutta: k^{th} order systems

I claim that if I know the Runge-Kutta nth order coefficients for a single equation, I know them for an entire system. Like I said for first order systems, I can only motivate how this would work by doing examples.

Let $\vec{y'} = \vec{f}(t, \mathbf{y})$ with $\vec{y}(t_i) = \vec{y_i}$.

Let $\vec{\phi}(t_i + h) = \vec{y_i} + v_1 h\vec{f_1} + v_2 h\vec{f_2}$

Where

$$\vec{y} = \begin{pmatrix} y_1(t) \\ y_2(t) \\ ... \\ y_k(t) \end{pmatrix}$$
$$\vec{f}(t, \vec{y}) = \begin{pmatrix} g_1(t, \vec{y}) \\ g_2(t, \vec{y}) \\ ... \\ g_k(t, \vec{y}) \end{pmatrix}$$
$$\vec{f_1} = \vec{f}(t_i, \vec{y_i})$$
$$\vec{f_2} = \vec{f}(t_i + a_2 h, \vec{y_i} + b_0 h\vec{f_1})$$

In order to satisfy the taylor series 2 polynomial, we must match up phi with $T_2(t)$, the second order taylor polynomial.
For that to happen, one must compute $\vec{y''}$.

$$y''\vec{(t)} = \vec{f'}(t_i, \vec{y_i}) = \begin{pmatrix} \frac{\partial g_1}{t}\frac{dt}{dt} + \sum_{i=1}^{k}(\frac{\partial g_1}{\partial y_i}\frac{dy_i}{dt}) \\ \frac{\partial g_2}{\partial t}\frac{dt}{dt} + \sum_{i=1}^{k}(\frac{\partial g_2}{\partial y_i}\frac{dy_i}{dt}) \\ ... \\ \frac{\partial g_k}{t}\frac{dt}{dt} + \sum_{i=1}^{k}(\frac{\partial g_k}{\partial y_i}\frac{dy_i}{dt}) \end{pmatrix}$$

Therefore we must match:

$$\vec{y}(t_i + h) = y(t_i) + h\vec{f}(t_i, y_i) + \frac{h^2}{2!}\begin{pmatrix} \frac{\partial g_1}{\partial t}\frac{dt}{dt} + \sum_{i=1}^{k}(\frac{\partial g_1}{\partial y_i}\frac{dy_i}{dt}) \\ \frac{\partial g_2}{\partial t}\frac{dt}{dt} + \sum_{i=1}^{k}(\frac{\partial g_2}{\partial y_i}\frac{dy_i}{dt}) \\ ... \\ \frac{\partial g_k}{\partial t}\frac{dt}{dt} + \sum_{i=1}^{k}(\frac{\partial g_k}{\partial y_i}\frac{dy_i}{dt}) \end{pmatrix}$$
$$= \vec{y_i} + h\vec{f_1} + \frac{h^2}{2!}(\vec{f_t} + \vec{f_{y_1}}g_1 + \vec{f_{y_2}}g_2 + ... + \vec{f_{y_k}}g_k)$$

With:

$$\vec{\phi}(t_i + h) = \vec{y_i} + v_1 h\vec{f_1} + v_2 h\vec{f_2}$$

Consider using the linearization of $\vec{f}$ to replace $\vec{f_2}$.

Here is the linearization off of $(t_i, \vec{y_i})$

$$\vec{f}(t, \vec{y}) = \vec{f}(t_i, \vec{y_i}) + (t - t_i)\vec{f_t}(t_i, \vec{y_i}) + (\vec{y_1} - y_1\vec{(t_i)})\vec{f_{y_1}}(t_i, \vec{y_i}) + ... + (\vec{y_k} - y_k\vec{(t_i)})\vec{f_{y_k}}(t_i, \vec{y_i})$$

$\vec{f}(t_i + a_2 h, \vec{y_i} + hb_0\vec{f_1}) = \vec{f}(t_i, \vec{y_i}) + a_2 h\vec{f_t}(t_i, \vec{y_i}) + b_0 h g_1(t_i, \vec{y_i})\vec{f_{y_1}} + ... +$

$b_0 h g_k(t_i, \vec{y_i}) \vec{f_{y_k}}$

Which means that

$\vec{\phi}(t_i + h) = \vec{y}(t_i) + v_1 h \vec{f_1} v_2 h(\vec{f}(t_i, \vec{y_i}) + a_2 h \vec{f_t}(t_i, \vec{y_i}) + b_0 h g_1(t_i, \vec{y_i}) \vec{f_{y_1}} + ... +$
$b_0 h g_k(t_i, \vec{y_i}) \vec{f_{y_k}})$
$\vec{y_i} + (v_1 + v_2) h \vec{f_1} + v_2 h^2 (\vec{f_t} a_2 + b_0 \sum_{i=1}^{k} g_i \vec{f_{y_i}})$

And this results in the following order conditions:

$v_1 + v_2 = 1$
$v_2 a_2 = \frac{1}{2}$
$v_2 b_0 = \frac{1}{2}$

These are the same order conditions for the system of one equation.

10 Simulations

1. Particle simulation
2. Runge-Kutta on any system
3. Finding the coefficients to RK order N.

Link to github repo