On the Runge-Kutta-Method

N. Bernstein, A. Thompson

Math 340 - Fall 2018.

December 14, 2018

Abstract

This report discusses the Runge-Kutta-Fehlberg method of solving initial-value problems. It is compared other approximation methods of solving these problems both numerically and graphically.

Introduction

Many ordinary differential equations (ODE) found outside of the classroom are often too complicated to solve explicitly, however, the solution to initial-value problems (IVP) offer valuable information necessary to create an accurate mathematical model. To combat this, methods have been developed to acquire a reasonable approximation of the solution. Consider the following IVP for some function $y(t)$:

$$y' = f(t, y) \qquad a \leq t \leq b \qquad y(a) = y_0$$

To solve this equation, we make approximations of $y(t_i) \approx y_i$ for $t_i \in [a, b]$ in order to create a set of values called mesh points that are used to approximate the solution at other points with the use of interpolation. We will use N equidistant mesh points with $h = \frac{b-a}{N} = t_{i+1} - t_i$ as the distance between them. Methods of making these approximations include:

Euler's Method

$y_{i+1} = y_i + hf(t_i, y_i)$

Taylor's Method of order n[1]

$y_{i+1} = y_i + hT^{(n)}(t_i, y_i)$

where $T^{(n)}(t_i, y_i) = f(t_i, y_i) + \frac{h}{2}f'(t_i, y_i) + \cdots + \frac{h^{n-1}}{n!}f^{(n-1)}(t_i, y_i)$

Midpoint Method[2]

$y_{i+1} = y_i + hf\Big(t_i + \frac{h}{2}, y_i + \frac{h}{2}f(t_i, y_i)\Big)$

Runge-Kutta of Order 4

$k_1 = hf(t_i, y_i)$

$k_2 = hf(t_i + \frac{h}{2}, y_i + \frac{1}{2}k_1)$

$k_3 = hf(t_i + \frac{h}{2}, y_i + \frac{1}{2}k_2)$

$k_4 = hf(t_i, y_i + k_3)$

$y_{i+1} = y_i + \frac{1}{6}\big(k_1 + 2k_2 + 2k_3 + k_4\big)$

Runge-Kutta Methods have an advantage over other methods in that they keep the high-order truncation error estimation but eliminate the need to compute the derivatives for $f(t, y)$. Methods such as these are considered to be adaptive. There are adaptive methods for intial-value problems that are efficient and allow the control of error. That is the control of the truncation error under some bounds which also help to control the global error. It does this by adjusting the step size of the procedure. In the next section, we will discuss the Runge-Kutta-Fehlberg method and how it can be more effective than the methods above.

1 Derivation of the method

For any one step method for solving the IVP, the error is $|y(t_i) - y|$. We can take measures to prevent this error from exceeding some $\varepsilon > 0$. We take approximation technique of order n and $n+1$ represented by the functions $\phi(t_i, y_i, h_i)$ and $\tilde{\phi}(t_i, y_i, h_i)$ respectively, so that we have:

$y(t_{i+1}) \approx y_{i+1} = y_i + h\phi(t_i, y_i, h)$ with a local error of $\tau_{i+1}(h) = O(h^n)$

$y(t_{i+1}) \approx \tilde{y}_{i+1} = y_i + h\tilde{\phi}(t_i, y_i, h)$ with a local error of $\tilde{\tau}_{i+1}(h) = O(h^{n+1})$

[1]Notice that Euler's Method is Taylor's Method of order 1.
[2]Runge-Kutta of order 2.

When we assume that $y_i \approx y(t_i) \approx \tilde{y}_i$ and choose a fixed step size h to generate the approximations we have:

$$\begin{aligned}\tau_{i+1}(h) &= \frac{y(t_{i+1}) - y(t_i)}{h} - \phi(t_i, y(t_i), h) = \frac{y(t_{i+1}) - y_i}{h} - \phi(t_i, y(t_i), h) \\ &= \frac{y(t_{i+1}) - y_i - h\phi(t_i, y(t_i), h)}{h} = \frac{1}{h}\Big(y(t_{i+1}) - y_{i+1}\Big)\end{aligned}$$

and similarly

$$\tilde{\tau}_{i+1}(h) = \frac{1}{h}\Big(y(t_{i+1}) - \tilde{y}_{i+1}\Big)$$

So we obtain:

$$\begin{aligned}\tau_{i+1}(h) &= \frac{1}{h}\Big(y(t_{i+1}) - y_{i+1}\Big) = \frac{1}{h}\Big(y(t_{i+1}) - \tilde{y}_{i+1} + \tilde{y}_{i+1} - y_{i+1}\Big) \\ &= \tilde{\tau}_{i+1}(h) + \frac{1}{h}\Big(\tilde{y}_{i+1} - y_{i+1}\Big)\end{aligned}$$

Since $\tau_{i+1}(h)$ is $O(h^n)$ and $\tilde{\tau}_{i+1}(h)$ is $O(h^{n+1})$, the significant portion of $\tau_{i+1}(h)$ comes from $\frac{1}{h}\Big(\tilde{y}_{i+1} - y_{i+1}\Big)$. Thus $\tau_{i+1}(h) \approx \frac{1}{h}\Big(\tilde{y}_{i+1} - y_{i+1}\Big) \approx Kh^n$ for some number K independent of h. Now to keep up with a specific bound for the local truncation error we can modify h by some factor q so that we have:

$$\tau_{i+1}(qh) \approx K(qh)^n = q^n(Kh^n) \approx q^n \tau_{i+1}(h) \approx \frac{q^n}{h}\Big(\tilde{y}_{i+1} - y_{i+1}\Big)$$

To bound $\tau_{i+1}(qh)$ to our error tolerance , we choose q such that:

$$\frac{q^n}{h}\Big(\tilde{y}_{i+1} - y_{i+1}\Big) \approx |\tau_{i+1}(qh)| \leq \varepsilon$$

We obtain the following inequality:

$$q \leq \left(\frac{\varepsilon h}{|\tilde{y}_{i+1} - y_{i+1}|}\right)^{\frac{1}{n}} = \left(\frac{\epsilon}{R}\right)^{\frac{1}{n}} \quad \text{where} R = |\tilde{y}_{i+1} - y_{i+1}| \quad \text{and} \quad \epsilon = \varepsilon h \tag{1}$$

There is a specific method for error control that uses (1) called the Runge-Kutta Fehlberg (RKF45) method and has a truncation error $O(h^5)$. The advantage of the Runge-Kutta Fehlberg is that it combines both fourth and fifth order Runge-Kutta, but only uses 6 k_i constants, instead of the 10 needed for fifth order Runge-Kutta, allowing for the same residual order with 40% less calculation per step. The particular derivation and determination of the relevant constants is left as an exercise for the reader, however the end result is as follows.

$k_1 = hf(t_i, y_i)$

$k_2 = hf\left(t_i + \frac{h}{4}, y_i + \frac{1}{4}k_1\right)$

$k_3 = hf\left(t_i + \frac{3h}{8}, y_i + \frac{3}{32}k_1 + \frac{9}{32}k_2\right)$

$k_4 = hf\left(t_i + \frac{12h}{13}, y_i + \frac{1932}{2197}k_1 - \frac{7200}{2197}k_2 + \frac{7296}{2197}k_3\right)$

$k_5 = hf\left(t_i + h, y_i + \frac{439}{216}k_1 - 8k_2 + \frac{3680}{513}k_3 - \frac{845}{4104}k_4\right)$

$k_6 = hf\left(t_i + \frac{h}{2}, y_i - \frac{8}{27}k_1 + 2k_2 - \frac{3544}{2565}k_3 + \frac{1859}{4104}k_4 - \frac{11}{40}k_5\right)$

<u>Truncation error order 4</u>

$y_{i+1} = y_i + \frac{25}{216}k_1 + \frac{1408}{2565}k_3 + \frac{2197}{4104}k_4 - \frac{1}{5}k_5$

<u>Truncation error order 5</u>

$\tilde{y}_{i+1} = y_i + \frac{16}{135}k_1 + \frac{6656}{12825}k_3 + \frac{28561}{56430}k_4 - \frac{9}{50}k_5 + \frac{2}{55}k_6$

To perform this method we use an initial value for h to calculate the first values for $\tilde{y}_{i+1}$ and y_{i+1} and then proceed to choose q for subsequent steps. When $R > \epsilon$, we reject h for those values of $\tilde{y}_{i+1}$ and y_{i+1} and perform the calculations again using qh. If $R \leq \epsilon$ then we accept the values and use qh for the next step.

2 Simulations

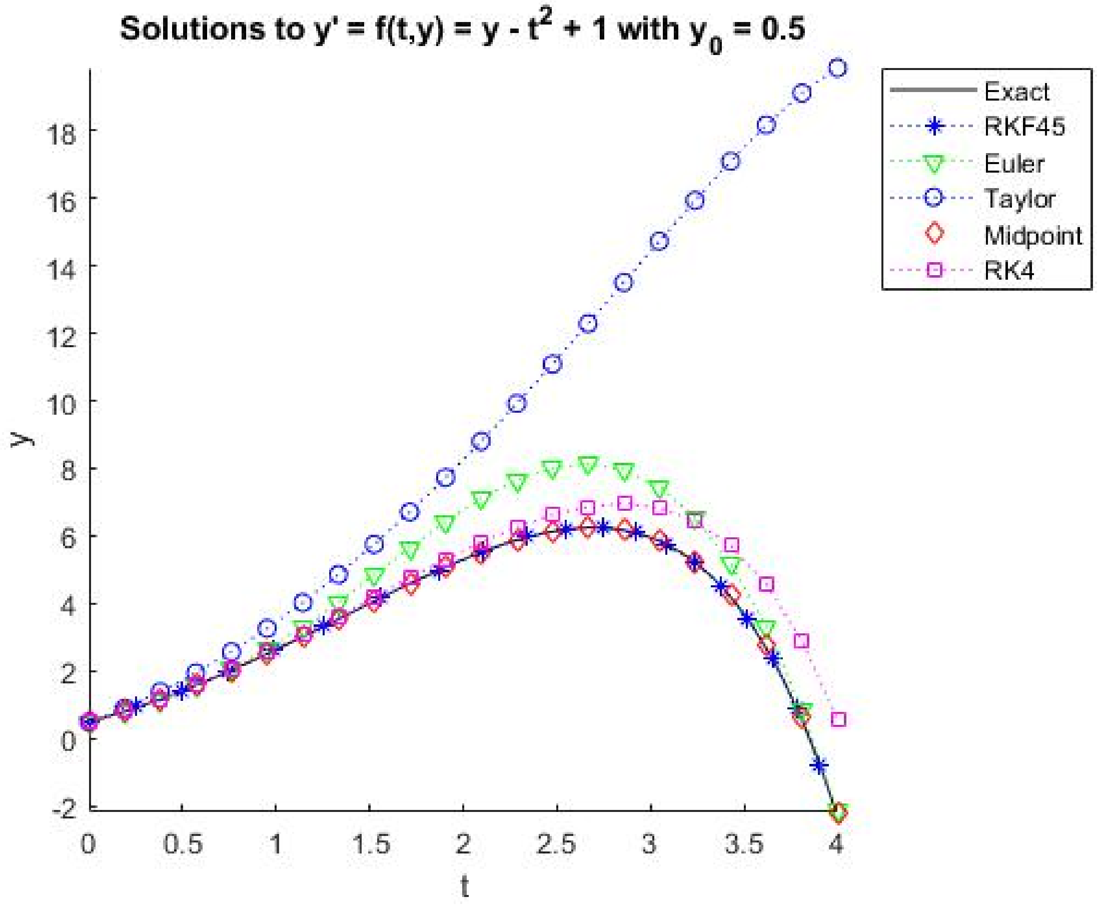
Now let's use all of these methods to approximate solutions for the following ODE and compare them to the correct answer $y(t) = (t+1)^2 - 5e^t$:

$y' = y - t^2 + 1 \quad 0 \leq t \leq 4 \quad y(0) = 0.5$

The RKF45 algorithm, see appendix, gives us the following values for y with the corresponding error.

t	RKF45	Error	t	RKF45	Error
0.256126	0.931897	0.000001	2.741529	6.243732	0.000034
0.493082	1.410619	0.000003	2.915772	6.101780	0.000040
0.736328	1.970712	0.000004	3.078945	5.770109	0.000046
0.989180	2.612328	0.000007	3.233070	5.240242	0.000053
1.258053	3.339529	0.000009	3.379521	4.501909	0.000060
1.559679	4.173323	0.000013	3.519305	3.543711	0.000069
1.869869	4.992443	0.000017	3.653192	2.353438	0.000078
2.107761	5.543301	0.000021	3.781798	0.918232	0.000088
2.343986	5.970917	0.000025	3.905626	-0.775318	0.000098
2.552901	6.200979	0.000029	4.000000	-2.298967	0.000108

After calculating Euler's Method, Taylor's Method of Order 3, Midpoint Method, and Runge-Kutta of Order 4, we have the below plot. For these we used the same amount of mesh points as in the RKF45, $n = 21$, which resulted in an $h = 2/21 \approx 0.0952$. The table for the y_i and error values for these methods can be found in the appendix.



..

Figure 1: Plot of all calculated methods.

Upon looking more closely at length 1 subintervals, we can see that the best solutions are the RK4 and RK45

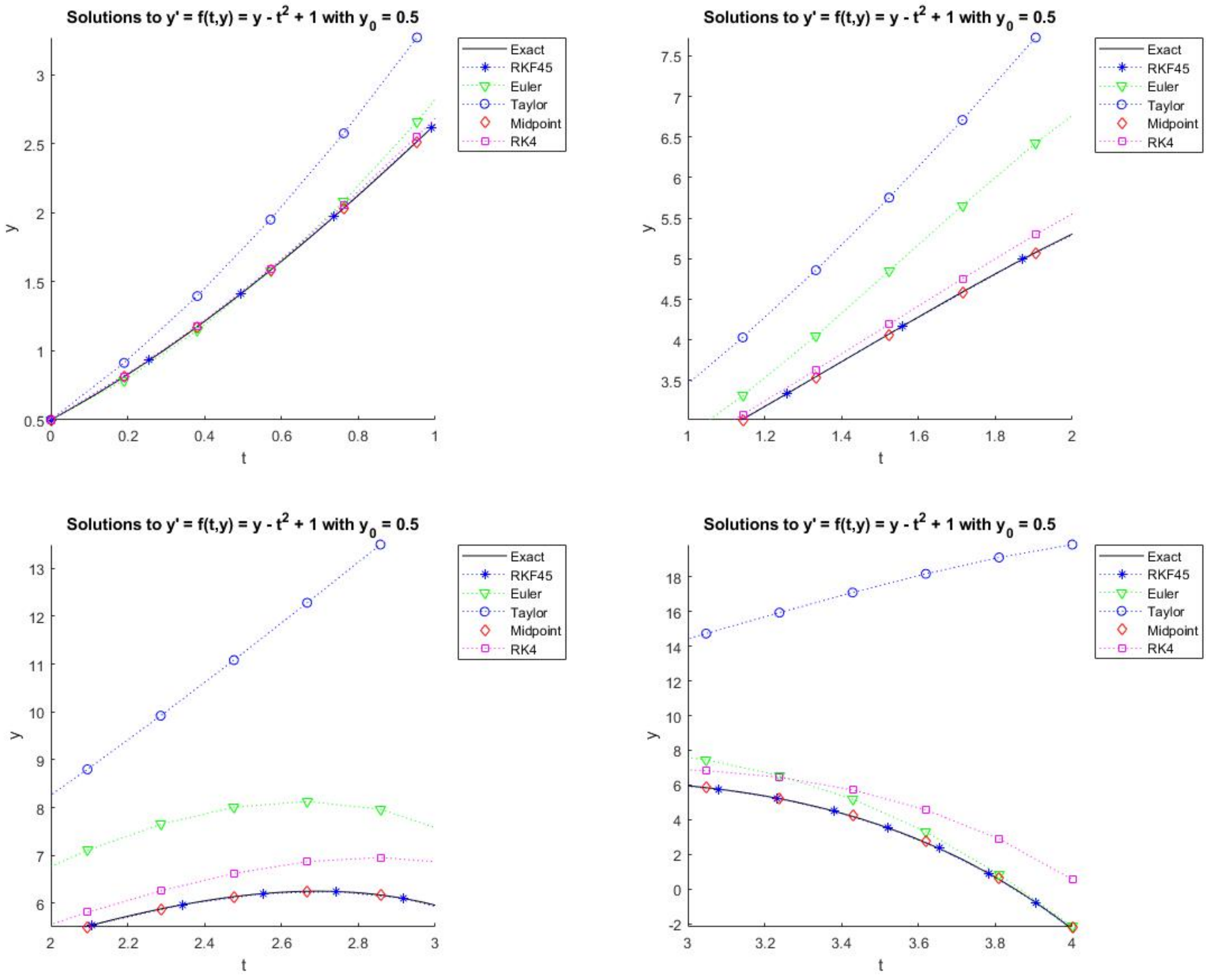


Table 1:

3 Conclusion and discussions

The advantage of the method is that it reduces the amount of computation necessary if order 5 is used as the truncation error for an order 4 computation. The RKF45 method uses only 6 coefficient equations instead of the total 10 (4 from order 4, 6 from order 5). As a method that uses Runge-Kutta, it is able to bypass the need to calculate any of the higher order derivatives needed in Taylor method, making it a more favorable method to use. In addition, just as order 5 provides a more accurate estimation than order 4, RKF45 provides a more accurate approximation over order 4 because it utilizes a combination of 4 and 5 as its truncation error and constantly adjusts during the steps of the procedure.

The disadvantage is if there is not a major concern for the truncation error to a small amount, this method holds a lot of excessive computation that can be done with Euler method which can still run under a satisfactory error. This can be seen when comparing the RK4 and RKF45 plots in Figure 1.

A Matlab Code

The following gives the Matlab code for the Runge-Kutta-Fehlberg method.

```
f = @(T,Y) Y - T.^2 + 1;
s = @(T) (T+1).^2 - 0.5*exp(T);
a = 0; b = 4;
y_0 = 0.5;

% RKF45
n = 4; q = 1/2; h_max = (b-a)/n; h_min = h_max*10^(-4); tol = ...
    10^(-5);
d_max = 2; d_min = d_max*10^(-2);
bool_flag = 1; t(1) = a; y(1) = y_0; i = 1; h = h_max;
while bool_flag
     %set k values
     k_1 = h*f(t(i),y(i));
     k_2 = h*f(t(i) + (h/4), y(i) + k_1/4);
     k_3 = h*f(t(i) + 3*h/8, y(i) + 3*k_1/32 + 9*k_2/32);
     k_4 = h*f(t(i) + 12*h/13, y(i) + 1932*k_1/2197 - ...
         7200*k_2/2197 ...
          + 7296*k_3/2197);
     k_5 = h*f(t(i)+h, y(i) + 439*k_1/216 - 8*k_2 + 3680*k_3/513 ...
          - 845*k_4/4104);
     k_6 = h*f(t(i) + h/2, y(i) - 8*k_1/27 + 2*k_2 - ...
         3544*k_3/2565 ...
          + 1859*k_4/4104 - 11*k_5/40);

     R = abs(k_1/360 - 128*k_3/4275 - 2197*k_4/75240 + k_5/50 + ...
         2*k_6/55)/h;
     if R ≤ tol
          t(i+1) = t(i) + h;
          y(i+1) = y(i) + 25*k_1/216 + 1408*k_3/2565 + ...
              2197*k_4/4104 - k_5/5;
          i = i+1;
     end
     d = nthroot(q, n) * nthroot((tol/R), n);
     if d ≤ d_min
          h = d_min*h;
     elseif d ≥ d_max
          h = d_max*h;
     else
          h = d*h;
     end
     if h > h_max
          h = h_max;
     end
     if t(i) ≥ b
          bool_flag = 0;
     elseif t(i)+h>b
          h = b-t(i);
     elseif h < h_min
          bool_flag = 0;
          error("Mininmum h exceeded.");
     end
end
```

B Table of Results from Other Methods

t	Euler	Error	Taylor	Error	Midpoint	Error	RK4	Error
0.000000	0.500000	0.000000	0.500000	0.000000	0.500000	0.000000	0.500000	0.000000
0.190476	0.785714	0.026606	0.912698	0.100378	0.811197	0.001123	0.813468	0.001148
0.380952	1.146784	0.028407	1.396519	0.221329	1.172836	0.002354	1.180030	0.004840
0.571429	1.578307	0.005684	1.951006	0.367016	1.580302	0.003688	1.595601	0.011611
0.761905	2.081960	0.048828	2.575556	0.542424	2.028017	0.005115	2.055238	0.022106
0.952381	2.659452	0.143597	3.269376	0.753522	2.509240	0.006615	2.552962	0.037108
1.142857	3.313261	0.289282	4.031421	1.007442	3.015823	0.008156	3.081540	0.057561
1.333333	4.049871	0.502260	4.860314	1.312704	3.537920	0.009690	3.632220	0.084610
1.523810	4.852664	0.777888	5.754241	1.679465	4.063630	0.011146	4.194415	0.119638
1.714286	5.656722	1.065729	6.710812	2.119819	4.578567	0.012426	4.755314	0.164321
1.904762	6.424749	1.346012	7.726878	2.648141	5.065348	0.013389	5.299422	0.220684
2.095238	7.105294	1.588483	8.798291	3.281480	5.502961	0.013850	5.807990	0.291179
2.285714	7.648858	1.769293	9.919586	4.040021	5.866010	0.013555	6.258334	0.378769
2.476190	8.006437	1.870467	11.083562	4.947592	6.123802	0.012168	6.623014	0.487044
2.666667	8.128073	1.879587	12.280733	6.032247	6.239243	0.009244	6.868829	0.620343
2.857143	7.962193	1.790496	13.498603	7.326906	6.167495	0.004202	6.955615	0.783918
3.047619	7.455268	1.604613	14.720706	8.870051	5.854373	0.003718	6.834776	0.984121
3.238095	6.551598	1.332713	15.925350	10.706465	5.234382	0.015497	6.447524	1.228639
3.428571	5.193155	0.997193	17.083957	12.887994	4.228368	0.032406	5.722733	1.526771
3.619048	3.319466	0.634878	18.158877	15.474289	2.740662	0.056073	4.574355	1.889767
3.809524	0.867499	0.300452	19.100506	18.533459	0.655636	0.088589	2.898286	2.331239
4.000000	-2.144203	0.154872	19.843496	22.142571	-2.166457	0.132618	0.568579	2.867654

References

[1] R. Burden, J. Faires, and A. M. Burden, *Numerical Analysis*, Cengage, 10 ed., 2015.