Report of Second Order Runge-Kutta Method

Yitong (Tony) Zhao
Differential Equation, Period 1
Princeton International School of Mathematics and Science

Contents

1. Introduction

The Runge-Kutta (RK) method is a numerical method to solve ordinary differential equations. It is considered a generalization of Euler's method and its variations. And its accuracy is higher than that of ordinary Euler's method. In this report, the second-order Runge-Kutta (RK2) method is introduced and discussed.

1.1. RK2

Given a differential equation $y' = f(x, y)$ with initial condition $y(x_0) = y_0$. In a single step, the RK2 method can determine the value of y at $x_0 + h$ by the following formula [1]:

$$
\begin{aligned}
x_{n+1} &= x_n + h \\
y_{n+1} &= y(x_{n+1}) = y_n + \frac{h}{2}(w_1 k_1 + w_2 k_2) \\
k_1 &= f(x_n, y_n) \\
k_2 &= f(x_n + \alpha h, y_n + \beta h k_1)
\end{aligned} \tag{1}
$$

Here w_1, w_2, α, β are coefficients. These coefficients have to be selected according to the following conditions [1] in order to reach a local truncation error of $O(h^3)$ and a global truncation error of $O(h^2)$. The derivation of these two errors will be shown at Section 2:

$$
\begin{aligned}
w_1 + w_2 &= 1 \\
w_2 \alpha = w_2 \beta &= \frac{1}{2}
\end{aligned} \tag{2}
$$

It could be seen from Equation 1 that k_1 and k_2 are slopes of a function at some points. And from the first part of Equation 2 ($w_1 + w_2 = 1$) reveals that the RK2 method is a weighted average of the slopes.

1.2. General RK methods

More generally, RK methods (including other orders) can be expressed as:

$$
\begin{aligned}
y_{n+1} &= y_n + h \sum_{i=1}^{s} b_i k_i \\
k_1 &= f(x_n, y_n) \\
k_2 &= f(x_n + c_2 h, y_n + h(a_{21} k_1)) \\
k_3 &= f(x_n + c_3 h, y_n + h(a_{31} k_1 + a_{32} k_2)) \\
&\vdots \\
k_n &= f\big(x_n + c_n h, y_n + h\big(a_{n1} k_1 + a_{n2} k_2 + \ldots + a_{n,n-1} k_{n-1}\big)\big)
\end{aligned} \tag{3}
$$

This is, essentially, still a weighted average of the slopes. The coefficients a_{ij}, b_i, c_i are assigned to reach minimum truncation error.

The intuition behind this method lies behind the Mean Value Theorem, which states that for a function $f(x)$ that is continuous on the closed interval $[a, b]$ and differentiable on the open interval (a, b), there exists a point c in (a, b) such that:

$$
f'(c) = \frac{f(b) - f(a)}{b - a} \tag{4}
$$

Meaning that the slope of the function at some point is equal to the slope of the secant line between the two points. This can be illustrated as:

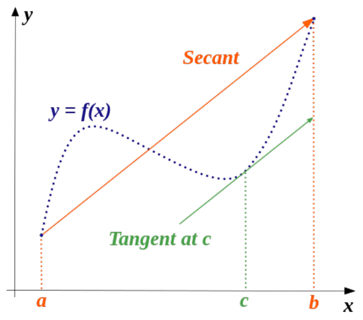


Figure 1: Mean Value Theorem [2]

Euler's method uses the slope at point a to approximate the secant line between a and b, which is often inaccurate. On the other hand, high order RK method examines the slope at multiple points in the interval $[a, b]$. Thus, the approximated slope is closer to that of point c's.

2. Derivation

The general approach to deriving the truncation error and the relationship mentioned in Equation 2 is to compare RK2 approximation in Taylor-expanded form with the exact solution, also in Taylor-expanded form.

Formally, given a function $f(x)$, its Taylor expansion at point x_0 is:

$$\begin{aligned} f(x) &= f(x_0) + f'(x_0)(x - x_0) + f''(x_0)\frac{(x - x_0)^2}{2} + f'''(x_0)\frac{(x - x_0)^3}{6} + \ldots \\ &= \sum_{n=0}^{\infty} f^n(x_0)\frac{(x - x_0)^n}{n!} \end{aligned} \tag{5}$$

Notice that the approximation will be more accurate if the point we are approximating is near x_0.

2.1. Expressing the Exact Solution in Taylor Series

Taking x_n as x_0 and $x_n + h$ as x in Equation 5, we can get the Taylor expansion of the exact solution $y(x_n + h)$:

$$\begin{aligned} y_{n+1} &= y(x_n + h) \\ &= y(x_n) + hy'(x) + \frac{1}{2}h^2 y''(x_n) + O(h^3) \end{aligned} \tag{6}$$

Notice that the $O(h^3)$ sign here is the asymptotic notation, which means that with the written part until the h^2 term, the error of this Taylor series approximation is bounded by Ch^3 for some constant C.

From Equation 6, the term $y'(x)$ can be written as $f(x, y)$, according to the definition in Section 1.1. However, the term $y''(x_n)$ is not yet expressed in the form of $f(x, y)$, which needs to be done in order to compare it with the RK2 approximation. The following steps show the transformation:

$$\begin{aligned} y''(x_n) &= \frac{\mathrm{d}}{\mathrm{d}x} f(x_n, y(x_n)) \\ &= \frac{\partial f}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}x} + \frac{\partial f}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}x} \\ &= f_x(x_n, y_n) + f_y(x_n, y_n) f(x_n, y_n) \end{aligned} \tag{7}$$

Here f_x and f_y are the partial derivatives of f with respect to x and y, respectively. Plug the result back into Equation 6, and we get:

$$y_{n+1} = y_n + hf(x_n, y_n) + \frac{1}{2}h^2\big(f_x(x_n, y_n) + f_y(x_n, y_n)f(x_n, y_n)\big) + O(h^3) \tag{8}$$

To simplify the notation, let $f_x(x_n, y_n)$ be f_x, $f(x_n, y_n)$ be f, and $f_y(x_n, y_n)$ be f_y. Then:

$$y_{n+1} = y_n + hf + \frac{1}{2}h^2(f_x + f_y f) + O(h^3) \tag{9}$$

2.2. Expressing the RK2 Approximation in Taylor Series

From Equation 1, we have:

$$\begin{aligned} y_{n+1\ \mathrm{RK2}} &= y_n + h(w_1 f + w_2 f(x_n + \alpha h, y_n + \beta h k_1)) \\ &= y_n + h(w_1 f + w_2 f(x_n + \alpha h, y_n + \beta h f)) \end{aligned} \tag{10}$$

To compare this with Equation 9, we need to express $f(x_n + \alpha h, \beta h k_1)$ in terms of f and its derivatives. This could be done using Taylor expansion as well:

$$\begin{aligned} f(x_n + \alpha h, \beta h k_1) &= f(x_n) + \mathrm{d}f\big|_{(x_n, y_n)} + \frac{1}{2}\mathrm{d}^2 f\big|_{(x_n, y_n)} + \frac{1}{6}\mathrm{d}^3 f\big|_{(x_n, y_n)} + \dots \\ &= f(x_n) + \mathrm{d}f\big|_0 + O(h^2) \end{aligned} \tag{11}$$

With the definition of total derivative, $\mathrm{d}f$ can be represented as $f_x \mathrm{d}x + f_y \mathrm{d}y$. $\mathrm{d}x$, which represent the change of x_n, is αh, and $\mathrm{d}y$ is $\beta h k_1$, according to Equation 1. Thus, we have:

$$f(x_n + \alpha h, \beta h k_1) = f(x_n) + \alpha h f_x + \beta h f_y k_1 + O(h^2) \tag{12}$$

Plug this back into Equation 10, we get:

$$\begin{aligned} y_{n+1\ \mathrm{RK2}} &= y_n + h\big(w_1 f + w_2(f + \alpha h f_x + \beta h f_y f)\big) \\ &= y_n + (w_1 + w_2)hf + w_2(f_x\alpha + f_y\beta f)h^2 \end{aligned} \tag{13}$$

To compare the approximation using RK2 and the actual solution, we perform a subtraction:

$$\begin{aligned} & y_{n+1} - y_{n+1\ \mathrm{RK2}} \\ &= \cancel{y_n} + hf + \frac{1}{2}h^2(f_x + f_y f) + O(h^3) - \cancel{y_n} - (w_1 + w_2)hf - w_2(f_x\alpha + f_y\beta f)h^2 \\ &= hf(1 - [w_1 + w_2]) + h^2\left(\frac{1}{2}f_x + \frac{1}{2}f_y f - w_2[f_x\alpha + f_y\beta f]\right) + O(h^3) \\ &= hf(1 - [w_1 + w_2]) + h^2\left(f_x\left[\frac{1}{2} - w_2\alpha\right] + f_y\left[\frac{1}{2} - w_2\beta\right]\right) + O(h^3) \end{aligned} \tag{14}$$

To minimize the difference between our approximation and the actual value, we want the coefficient for both h and h^2 to be zero. To do this, we need to ensure that $1 - (w_1 + w_2) = 0$ and $\frac{1}{2} - w_2\alpha = 0$ and $\frac{1}{2} - w_2\beta = 0$. These results are exactly the same as Equation 2.

When these conditions are met, the only thing left in Equation 14 is the $O(h^3)$ term. Thus, the local truncation error is $O(h^3)$.

2.3. Global Truncation Error

It could be derived that if the local truncation error of an RK method is $O(h^{p+1})$, then that of the global truncation error is $O(h^p)$. Thus, the global truncation error of RK2 is $O(h^2)$. The derivation is demonstrated in the following steps [3].

Assume we performed N steps of RK methods in an interval of $[t_0, t_1]$ and the error of each step is $e = O(h^{p+1})$. Then, the global error is $E = Ne$. Also, $h = \frac{t_1 - t_0}{N}$.

Thus:

$$\begin{aligned} E = Ne &= O(Nh^{p+1}) \\ &= O\left(\frac{(t_1 - t_0)^{p+1}}{N^p}\right) \\ &= O((t_1 - t_0)h^p) = O(h^p) \end{aligned} \tag{15}$$

3. Examples

As mentioned previously, a numerical method could be classified into RK2 if it satisfies the conditions mentioned in Equation 2. In this section, several specific examples of RK2 methods are introduced.

3.1. Improved Euler's Method

The method is defined as follows [1]:

$$\begin{aligned} y_{n+1} &= y_n + h\frac{f(x_n, y_n) + f(x_{n+1}, y^*_{n+1})}{2} \\ y^*_{n+1} &= y_n + hf(x_n, y_n) \\ y_{n+1} &= y_n + h\frac{f(x_n, y_n) + f(x_{n+1}, y_n + hf(x_n, y_n))}{2} \end{aligned} \tag{16}$$

If we re-write this in the form of RK2 from Equation 1, we have:

$$\begin{aligned} \alpha = \beta = 1 \\ w_1 = w_2 = \frac{1}{2} \end{aligned} \tag{17}$$

The method is well-illustrated by Figure 2.

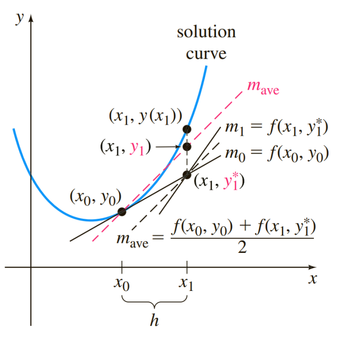


FIGURE 9.1.1 Slope of red dashed line is the average of m_0 and m_1

Figure 2: Improved Euler's Method [1]

It essentially takes the average of the slopes at the beginning and the end of the interval (if the slope does not change if y values changed).

3.2. Midpoint Method

The parameters for the midpoint method are described as follows. Which also met the condition mentioned in Equation 2:

$$\alpha = \beta = \frac{1}{2}$$
$$w_1 = 0, w_2 = 1 \tag{18}$$
$$y_{n+1} = y_n + hf\left(x_n + \frac{h}{2}, y_n + \frac{h}{2}f(x_n, y_n)\right)$$

Figure 3 shows the illustration of the method.

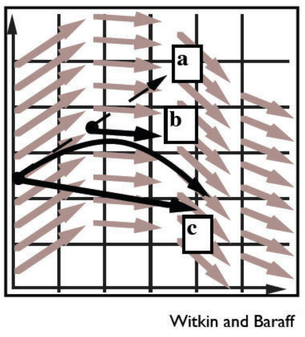


Figure 3: Midpoint Method [4]

And Figure 3 shows the steps of the midpoint method:

a. Compute a normal Euler step at (x_n, y_n), $\Delta y = hf(x, y)$
b. Evaluate the slope f_{mid} at $f\left(x_n + \frac{h}{2}, y_n + \frac{\Delta y}{2}\right)$
c. Take a step using f_{mid}

4. Problems

4.1. Given $y' = e^x - 3y, y(0) = 1$, and stpe size is 0.1, what is $y(0.1)$

4.1.1. Solving Problem

Here, we first translate the language of this problem into the mathematical language we used before to describe RK2 methods in Section 1.1.

1. $f(x, y) = e^x - 3y$
2. $x_0 = 0, y_0 = 1$
3. $h = 0.1$
4. $x_{\text{final}} = 0.1$

As discussed before, a single step of RK2 method is only able to calculate $y_{n+1} = y_n + h$ from y_n. To calculate points that are more far away, we need to repeat the algorithm multiple times. To start with, we determine how many iterations we will need to perform to solve this problem.

Since $x_{\text{final}} - x_0 = 0.1$, which is exactly our step size, we only need to perform one iteration.

As introduced in Section 3, there exist multiple types of RK2 methods, and we need to choose one of them to solve this problem. In this case, we choose the improved Euler's method. Its definition is listed in Equation 16 and Equation 17. The specific calculations are listed in the following steps:

$$\begin{gathered}
k_1 = f(x_0, y_0) = f(0.1) = e^0 - 3 = -2 \\
k_2 = f(x_0 + h, y_0 + hk_1) = f(0.1, 1 + 0.1 \times -2) = e^{0.1} - 3(0.8) = -1.295 \\
y_1 = y_0 + \frac{h}{2}(k_1 + k_2) = 1 + \frac{0.1}{2}(-2 + -1.295) = 0.853 \\
y(0.1) = 0.853
\end{gathered} \tag{19}$$

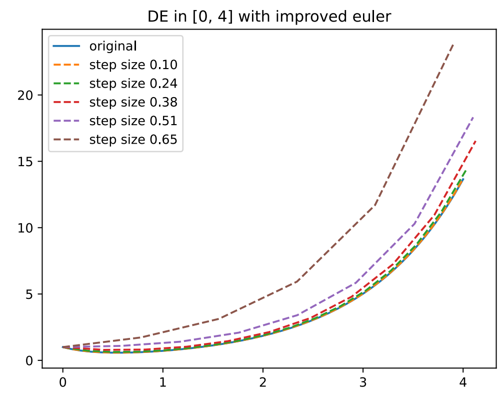
4.1.2. Error Analysis

Analytical techniques, specifically the integration factor, could solve this function. The specific processes are omitted, but the result is listed below:

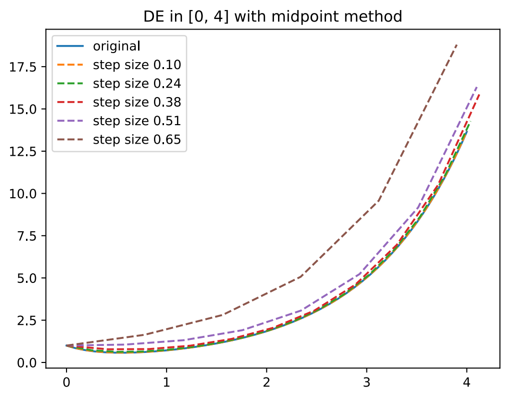
$$y = \frac{e^x}{4} + \frac{4}{4e^{3x}} \tag{20}$$

With this equation, we can calculate the error of the approximation. $y_{\text{actual}}(0.1) = 0.832$. And error $= |y_{\text{approx}}(0.1) - y_{\text{actual}}(0.1)| = |0.853 - 0.832| = 0.021$, which is relatively small.

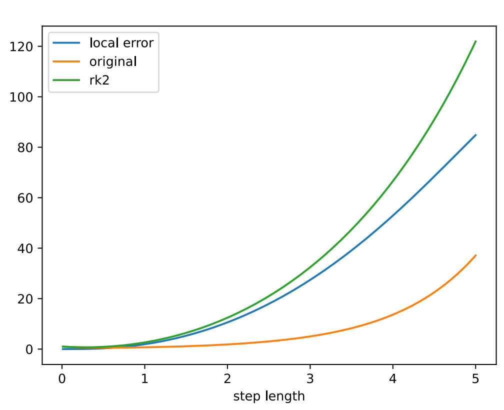
4.1.3. Graphs by Computers



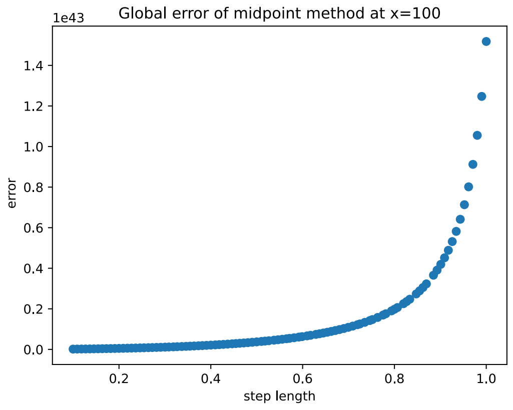
a) Improved Euler's Method, code in Section 5.2



b) Midpoint Method, code in Section 5.3



c) Local Error, code in Section 5.4



d) Global Error, code in Section 5.5

Figure 4: Graphs for Problem 1

From Figure 4(a, b), we can see that although different coefficients are used in the Improved Euler's method and midpoint method when the step size is similar, there are not too many differences in terms of precision when compared to the actual solution. This is because they are all classified as RK2 methods and share the same local truncation error of $O(h^3)$.

Figure 4(c, d) shows the local and global errors of RK2 methods (midpoint to be specific, but there are not too many differences as mentioned) in comparison with the actual solution of the problem. It can be observed that the blue line in Figure 4(c), which represents the local error, is growing much slower than the line in Figure 4(d), which represents the global error. This observation is compatible with the previous derivation of local error and global error in Section 2.2, in which we concluded that the global error is $O(h^2)$, higher than that of the local error of $O(h^3)$.

4.2. Modeling Question

A mass m is attached to the end of a spring whose constant is k. After the mass reaches equilibrium, its support begins to oscillate vertically about a horizontal line L according to a formula $h(t)$. The value of h represents the distance in feet measured from L.

a. Determine the differential equation of motion if the entire system moves through a medium offering a damping force that is numerically equal to $\beta\frac{\mathrm{d}x}{\mathrm{d}t}$

b. Solve for $x(0.4)$ in the differential equation in part (a) by RK2 methods with step size of 0.2. Given the spring is stretched 4 feet by a mass weighing 16 pounds and $\beta = 2, h(t) = 5\cos t, x(0) = x'(0) = 0$

— Modified from final review packet by Dr.Sood

4.2.1. Part (a)

Considering the general equation for damped spring motion $mx''(t) + \beta x'(t) + kx(t) = F(t)$. m is the mass of the object, β is the damping coefficient, k is the spring constant, and $F(t)$ is the external force. Here, the only external force presented in the system is caused by the vertical oscillation of the support. Since Hook's Law $F = kx$, we can convert the position function of the support into a force function: $F(t) = kh(t)$.

Overall, we have:

$$mx''(t) + \beta x'(t) + kx(t) = kh(t) \tag{21}$$

4.2.2. Part (b)

There are many coefficients in the differential equation constructed in the previous part. β is given, so we still need to figure out m and k.

- m: In the imperial unit system, the unit of mass is pound, and the unit of acceleration is feet per second squared. Thus, the unit of m is pound-second squared per foot (slug). To convert between pound and slug, we need to multiply the mass by the gravitational acceleration $g = 32$ feet per second squared. Thus $m = \frac{16}{32} = 0.5$ slug.
- k: The unit of k is pound per foot. Since the spring is stretched 4 feet by a mass weighing 16 pounds, we have $k = \frac{16}{4} = 4$ pound per foot (since $F = kx$).

Now, substitute these values. We have:

$$\frac{1}{2}x'' + 2x' + 4x = 20\cos t \tag{22}$$

Notice that this is a second-order differential equation. However, the RK2 method is only capable of solving first-order differential equations. Thus, we need to convert this second-order differential equation into a system of first-order differential equations.

Suppose we denote u as x'. Then, the following system of equations could be obtained:

$$\begin{cases} x' = u \\ u' = 2(20\cos t - 4x - 2u) = 40\cos t - 8x - 4u \end{cases} \tag{23}$$

Now, we converted a second-order ODE into a system of first-order ODEs. This could be more conveniently expressed in the form of matrices and vectors. Instead of using u, we use a vector $\boldsymbol{X} = [x_1, x_2]^T = [x, x']^T$:

$$f(\boldsymbol{X}, t) = \boldsymbol{X}' = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}' = \begin{bmatrix} 0 & 1 \\ -8 & -4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 40\cos t \end{bmatrix} \tag{24}$$

Since we're using this vector notation, the initial condition should also be re-written in this form: $\boldsymbol{X}_0 = [0, 0]^T$.

From the condition of step size = 0.2 and $t_{\text{final}} = 0.4$, we know that $0.4 \div 0.2 = 2$ iterations are needed. In this solution, the midpoint method will be used. Since the notations in Equation 24 is different from the notation in Equation 18, the definition of the midpoint method is re-written as follows:

$$\begin{aligned}
\alpha = \beta &= \frac{1}{2} \\
w_1 = 0, w_2 &= 1 \\
\boldsymbol{X}_{n+1} &= \boldsymbol{X}_n + hf\left(\boldsymbol{X}_n + \frac{h}{2}f(\boldsymbol{X}_n, t_n), t_n + \frac{h}{2}\right)
\end{aligned} \tag{25}$$

From Equation 24 and Equation 25, we can start to calculate. The calculation steps in the following are organized by iterations:

1. Since $t_1 = t_0 + h$, $t_1 = 0.2$.

$$f(\boldsymbol{X}_0, t_0) = f\left(\begin{bmatrix}0\\0\end{bmatrix}, 0\right) = \begin{bmatrix}0 & 1\\-8 & -4\end{bmatrix}\begin{bmatrix}0\\0\end{bmatrix} + \begin{bmatrix}0\\40\cos 0\end{bmatrix} = \begin{bmatrix}0\\40\end{bmatrix} \tag{26}$$

$$\begin{aligned}
\left(\boldsymbol{X}_0 + \frac{h}{2}f(\boldsymbol{X}_0, t_0), t_0 + \frac{h}{2}\right) &= \left(\begin{bmatrix}0\\0\end{bmatrix} + .1\begin{bmatrix}40\\-160\end{bmatrix}, 0 + .1\right) \\
&= \left(\begin{bmatrix}4\\-16\end{bmatrix}, .1\right)
\end{aligned} \tag{27}$$

$$\begin{aligned}
\boldsymbol{X}_1 &= \begin{bmatrix}0\\0\end{bmatrix} + .2 \cdot f\left(\begin{bmatrix}4\\-16\end{bmatrix}, .1\right) \\
&= .2\left(\begin{bmatrix}-16\\32\end{bmatrix} + \begin{bmatrix}0\\39.8\end{bmatrix}\right) \\
&= \begin{bmatrix}-3.2\\14.36\end{bmatrix}
\end{aligned} \tag{28}$$

2. $t_2 = t_1 + h, t_2 = 0.4$

$$f(\boldsymbol{X}_1, t_1) = f\left(\begin{bmatrix}-3.2\\14.36\end{bmatrix}, 0\right) = \begin{bmatrix}0 & 1\\-8 & -4\end{bmatrix}\begin{bmatrix}-3.2\\14.36\end{bmatrix} + \begin{bmatrix}0\\40\cos 0.2\end{bmatrix} = \begin{bmatrix}14.36\\-133.1\end{bmatrix} \tag{29}$$

$$\begin{aligned}
\left(\boldsymbol{X}_1 + \frac{h}{2}f(\boldsymbol{X}_1, t_1), t_1 + \frac{h}{2}\right) &= \left(\begin{bmatrix}-3.2\\14.36\end{bmatrix} + .1\begin{bmatrix}14.36\\-133.1\end{bmatrix}, .3\right) \\
&= \left(\begin{bmatrix}-1.764\\1.05\end{bmatrix}, .3\right)
\end{aligned} \tag{30}$$

$$\begin{aligned}
\boldsymbol{X}_2 &= \begin{bmatrix}-3.2\\14.36\end{bmatrix} + .2 \cdot f\left(\begin{bmatrix}-1.764\\1.05\end{bmatrix}, .3\right) \\
&= \begin{bmatrix}-3.2\\14.36\end{bmatrix} + .2\left(\begin{bmatrix}1.05\\9.912\end{bmatrix} + \begin{bmatrix}0\\38.21\end{bmatrix}\right) \\
&= \begin{bmatrix}-2.99\\23.98\end{bmatrix}
\end{aligned} \tag{31}$$

Recall that in this problem, we are asked to find $x(0.4)$. Since $x = x_1$, we have $x(0.4) = x_1(0.4) = -2.99$.

4.2.3. Error Analysis

This problem could also be solved using analytical techniques. Specifically, the complementary solution x_c can be obtained by solving the auxiliary equation:

$$\begin{gathered}\frac{1}{2}m^2 + 2m + 4 = 0 \\ m = -2 \pm 2i \\ x_c = e^{-2t}(c_1 \cos 2t + c_2 \sin 2t)\end{gathered} \tag{32}$$

For a particular solution, we can use the method of undetermined coefficients in the following form:

$$x_p = A\cos t + B\sin t \tag{33}$$

And it could be solved that $A = \frac{56}{13}, B = \frac{72}{13}$. After solving the two constants in Equation 32 by the initial conditions provided, we can obtain the following general solution:

$$x_g = e^{-t}\left(-\frac{56}{13}\cos 2t - \frac{72}{13}\sin 2t\right) + \frac{56}{13}\cos t + \frac{32}{13}\sin t \tag{34}$$

Using this equation, we are able to calculate the actual value of $x(0.4) = 0.25$. The error is $0.25 - (-2.99) = 3.24$. This is a relatively large error, which might be caused by the relatively large step size and the fast changing rate of the function. In Section 4.2.4, we can observe the relationship between the step size in this problem.

4.2.4. Graphs by Computers

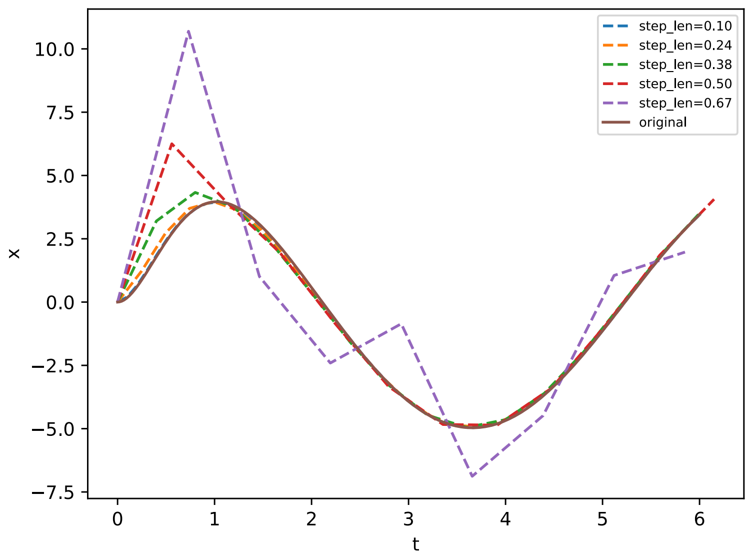


Figure 5: Effect of changing step size when approximating $x(0.4)$

Using the code listed in Section 5.1 and Section 5.6, the above graph is plotted. It can be seen that as the step size gets smaller and smaller, the approximation gets closer and closer to the actual value. Also, the error is the largest around $[0, 1]$, which is related to the large external force applied at the beginning (as $\cos 0 = 1$).

5. Appendix

5.1. Core RK2 Code

```
import numpy as np
import matplotlib.pyplot as plt
```

```python
from dataclasses import dataclass
@dataclass
class Rk2Config:
    alpha : float
    beta : float
    w1 : float
    w2 : float

IMP_EULER_CFG = Rk2Config(1, 1, .5, .5)
MIDPOINT_CFG = Rk2Config(.5, .5, 0, 1)


def rk2(dfunc : callable, init : np.ndarray, ts : np.ndarray, cfg : Rk2Config):
    """Runge-Kutta 2nd order method
    Args:
        dfunc (callable): function that returns the derivative of the function
        init (np.ndarray): initial values
        ts (np.ndarray): time steps
        cfg (Rk2Config): configuration of the method
    Returns:
        np.ndarray: array of values of the function at each time step
    """
    n = len(ts)
    ys = np.zeros((n, len(init)))
    ys[0] = init
    for i in range(1, n):
        t = ts[i-1]
        y = ys[i-1]
        h = ts[i] - t
        k1 = dfunc(t, y)
        k2 = dfunc(t + cfg.alpha * h, y + cfg.beta * h * k1)
        ys[i] = y + h * (cfg.w1 * k1 + cfg.w2 * k2)
    return ys
```

5.2. Plot for Improved Euler's Method in Problem 1

```python
def dfunc(t, y):
    return np.exp(t) - 3 * y
def func(t):
    return np.exp(t)/4 + np.exp(-3*t) * (3/4)
T_CNT = 100
T_MAX = 4
ts = np.linspace(0, T_MAX, T_CNT)
x_orig = func(ts)
plt.plot(ts, x_orig, label="original")
step_lens = np.linspace(.1, .65, 5)
for s in step_lens:
    cnt = round(T_MAX / s)
    near_final_t = cnt * s
    ts = np.linspace(0, near_final_t, cnt)
    x = rk2(dfunc, np.array([1]), ts, IMP_EULER_CFG)
    plt.plot(ts, x, label="step size {:.2f}".format(s), linestyle="--")
plt.legend()
plt.title("DE in [0, 4] with improved euler")
plt.savefig("./img/improved_euler_prob1.svg")
```

5.3. Plot for Midpoint Method in Problem 1

```
T_CNT = 100
T_MAX = 4
ts = np.linspace(0, T_MAX, T_CNT)
x_orig = func(ts)
plt.plot(ts, x_orig, label="original")
for s in step_lens:
    cnt = round(T_MAX / s)
    near_final_t = cnt * s
    ts = np.linspace(0, near_final_t, cnt)
    x = rk2(dfunc, np.array([1]), ts, MIDPOINT_CFG)
    plt.plot(ts, x, label="step size {:.2f}".format(s), linestyle="--")
plt.legend()
plt.title("DE in [0, 4] with midpoint method")
plt.savefig("./img/midpoint_prob1.svg")
```

5.4. Plot for Local Error in Problem 1

```
# evaluate local error
slens = np.linspace(.01, 5, 10000)
errs = []
yorigs = []
rk_ys = []
for s in slens:
    ts = [0, s]
    y = rk2(dfunc, np.array([1]), ts, MIDPOINT_CFG)
    x_orig = func(s)
    yorigs.append(x_orig)
    rk_ys.append(y[-1])
    err = abs((y[-1] - x_orig))
    errs.append(err)
plt.plot(slens, errs, label="local error")
plt.plot(slens, yorigs, label="original")
plt.plot(slens, rk_ys, label="rk2")
plt.xlabel("step length")
# plt.ylabel("error")
plt.legend()
plt.savefig("./img/local_err_prob1.svg")
```

5.5. Plot for Global Error in Problem 1

```
# evaluate how will the error change if we change step length

slens = np.linspace(.1, 1, 100)
errs = []
T_MAX = 100
actual_slens = []
err_deriv = []
for s in slens:
    ts = np.linspace(0, T_MAX, round(T_MAX / s))
    actual_slens.append(T_MAX / len(ts))
    y = rk2(dfunc, np.array([1]), ts, MIDPOINT_CFG)
    x_orig = func(ts)
    err = np.linalg.norm((y.flatten() - x_orig) )
    errs.append(err)
err_deriv = np.gradient(errs)
plt.scatter(actual_slens, errs)
# plt.scatter(actual_slens, err_deriv)
```

```
plt.xlabel("step length")
plt.ylabel(" error")
plt.title("Global error of midpoint method at x=100")
print("rg of deriv: ", max(err_deriv) - min(err_deriv))
plt.savefig("./img/global_err_prob1.svg")
```

5.6. Plotting Different Step Sizes for Problem 2

```
def dfunc(t, x):
    x, dx = x
    ddx = 40 * np.cos(t) - 8 * x - 4 * dx
    return np.array([dx, ddx])
def func(t):
    # x_g = e^(-t)(-56/13 cos 2t - 72/13 sin 2t) + 56/13 cos t + 32/13 sin t
    return np.exp(-2 * t) * (-56/13 * np.cos(2*t) - 72/13 * np.sin(2*t)) + 56/13 *
np.cos(t) + 32/13 * np.sin(t)
init_pts = np.array([0, 0])
FINAL_T = 6
ts = np.linspace(0, FINAL_T, 1000)
step_lens = np.linspace(.1, .7, 7)
for s in step_lens:
    cnt = round(FINAL_T / s)
    near_final_t = cnt * s
    ts = np.linspace(0, near_final_t, cnt)
    x_approx = rk2(dfunc, init_pts, ts, MIDPOINT_CFG)
    step_len = FINAL_T / cnt
    plt.plot(ts, x_approx[:, 0], label="step_len={:.2f}".format(step_len))
plt.legend(fontsize=7)
plt.xlabel("t")
plt.ylabel("x")
# plt.title("The effect of step length on the approximation of x(t)")
plt.savefig("img/problem2.svg")
```

Bibliography

[1] M. Braun and M. Golubitsky, *Differential equations and their applications*, vol. 2. Springer, 1983.

[2] Wikipedia contributors, “Mean value theorem.” [Online]. Available: https://en.wikipedia.org/wiki/Mean_value_theorem

[3] F. (https://math.stackexchange.com/users/850520/firavox), “Local truncation error vs Global truncation error.” [Online]. Available: https://math.stackexchange.com/q/4565373

[4] C. H. Séquin, “Spring Mass Systems: Midpoint Method for Solving ODEs,” *CS184: FOUNDATIONS OF COMPUTER GRAPHICS*. University of California, Berkeley, 2011.