PHYS 222 - Ordinary Differential Equations

Fahed Abu Shaer

September 2023

1 Euler's Method

Consider the following differential equation,

$$y^{'} = \frac{dy}{dt} = f(y)$$

The discrete definition of a derivative is,

$$\frac{dy}{dt} \approx \frac{y(t+h) - y(t)}{h}$$

$$\Rightarrow y(t+h) = y(t) + h\frac{dy}{dt}$$

but $\frac{dy}{dt} = f(y)$, therefore,

$$y(t+h) = y(t) - hf(y)$$

This is called the Euler's method, and we can use it to solve a differential equation numerically. I used this method to solve the following problem,
Consider a harmonic oscillator with frequency $\omega^2 = 10\ rad/sec$, initial position $x_0 = 5\ m$, and initial velocity $v_0 = 0\ m/s$, the differential equation that governs this system is,

$$\ddot{x} + w^2 x = 0$$

this equation is second-order, but it can be reduced to a system of two first-order differential equations,

$$\begin{cases} \dot{x} = v \\ \dot{v} = -w^2 x \end{cases}$$

We can apply Euler's method to solve these two equations, and obtain x the solution we are interested in.
So, I constructed a function where with a given initial conditions, parameters, number of points, and the distance between each point, it solves the equations using Euler's method and return the time, position, and velocity of the system,

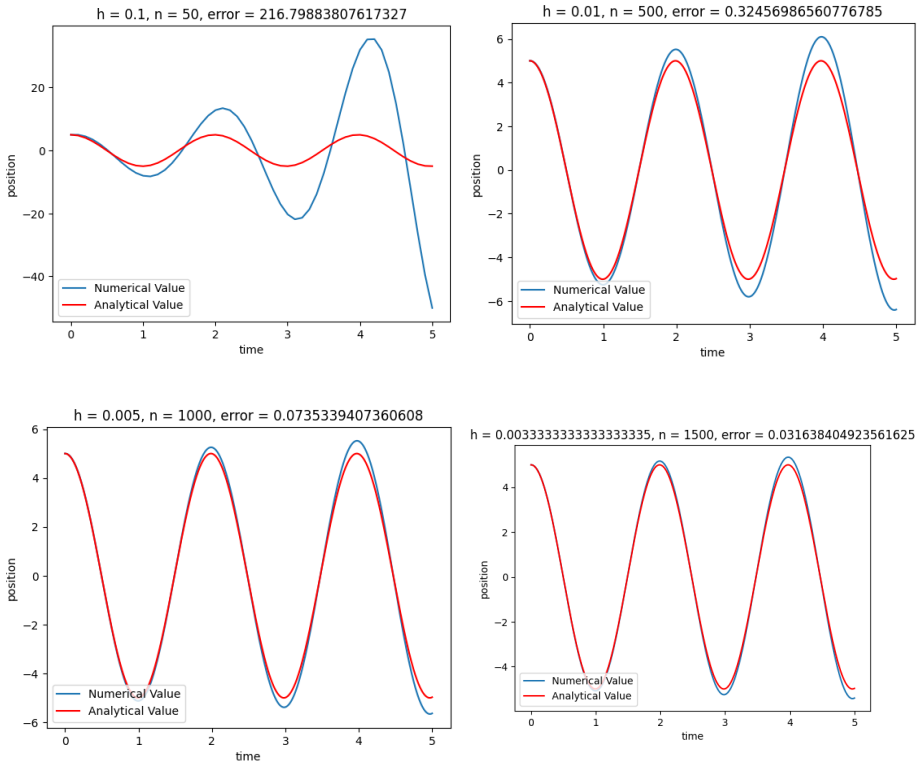
```
def Euler(h, n, x_0, v_0, w2):
    t = np.zeros(n+1)
    x = np.zeros(n+1)
    v = np.zeros(n+1)
    x[0] = x_0
    v[0] = v_0
    for i in range(n):
        x[i+1] = x[i] + h * v[i]
        v[i+1] = v[i] - h * w2 * x[i]
        t[i+1] = t[i] + h
    return t, x, v
```

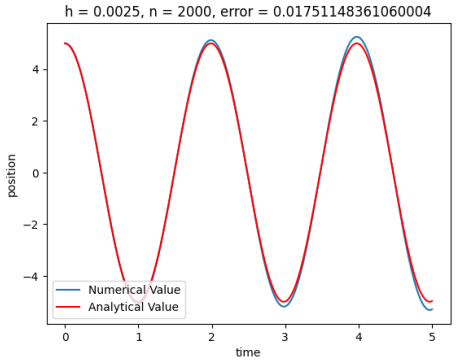
and I applied it with the initial conditions I had, varied the number of points and the distance between them, compared them to the analytic solution $x = x_0 cos(wt)$, and calculated the root mean square error of the method $error = \frac{1}{N}\sum_1^N (x_{analytic} - x_{numerical})^2$,

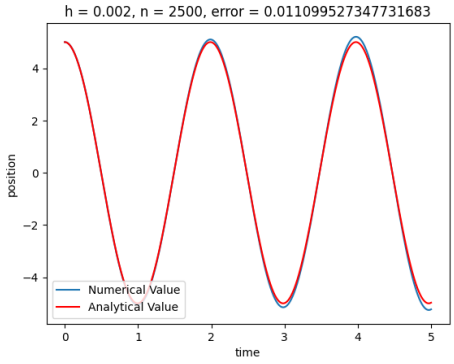
```
hs = []
errors = []
for j in range(0, 101, 10):
    if j == 0:
        h = 0.1
        n = 50
    else:
        h = 0.1/j
        n = 50 * j
    x_0, v_0, w2 = 5, 0, 10
    t, x, v = Euler(h, n, x_0, v_0, w2)
    x2 = 5 * np.cos(np.sqrt(w2)*t)
    error= np.sum((x2-x) ** 2)/n
    errors.append(error)
    hs.append(h)
    plt.plot(t, x, label = "Numerical Value")
    plt.plot(t, x2, color = 'r', label = "Analytical Value")
    plt.legend(loc = 'lower left')
    plt.xlabel("time")
    plt.ylabel("position")
    plt.title(f'h = {h}, n = {n}, error = {error}')
    print("h = ", h, " n =  ", n, " error = ", error)
    plt.show()
print("hs = ", hs, "\nerrors = ", errors)
plt.plot(hs, errors)
plt.scatter(hs, errors)
plt.xlabel("distance between the points")
plt.ylabel("error")
plt.show()
```

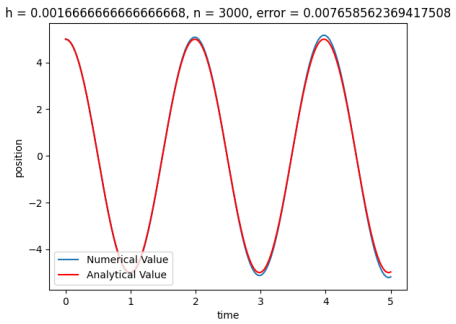
and this is the results I got,

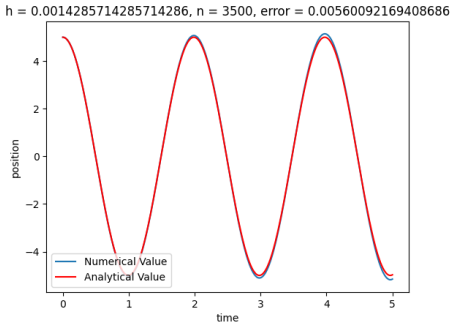
h	number of points	error
0.1	50	216.8
0.01	500	0.325
0.005	1000	0.074
0.003	1500	0.032
0.0025	2000	0.018
0.002	2500	0.011
0.001	3000	0.008
0.001	3500	0.006
0.00125	4000	0.004
0.00111	4500	0.003
0.001	5000	0.002

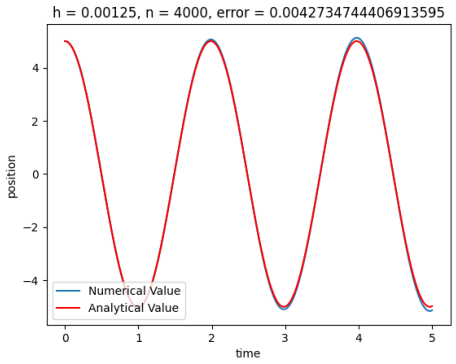


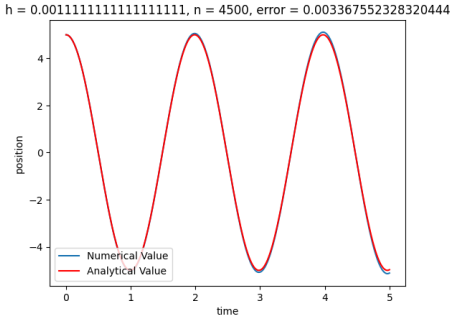

h = 0.0025, n = 2000, error = 0.01751148361060004
position
time
Numerical Value
Analytical Value

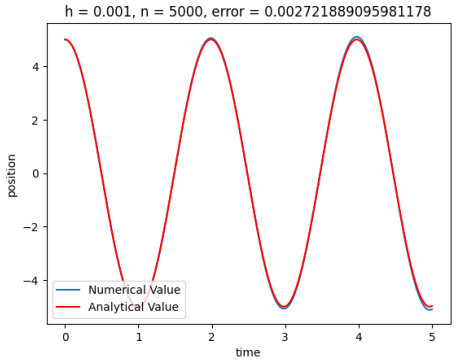

h = 0.002, n = 2500, error = 0.011099527347731683
position
time
Numerical Value
Analytical Value


h = 0.0016666666666666668, n = 3000, error = 0.007658562369417508
position
time
Numerical Value
Analytical Value


h = 0.0014285714285714286, n = 3500, error = 0.00560092169408686
position
time
Numerical Value
Analytical Value


h = 0.00125, n = 4000, error = 0.0042734744406913595
position
time
Numerical Value
Analytical Value


h = 0.0011111111111111111, n = 4500, error = 0.003367552328320444
position
time
Numerical Value
Analytical Value



The error as a function of the distance between the points is,

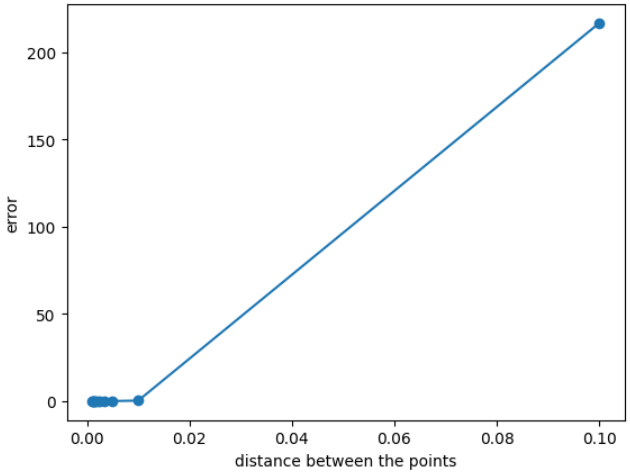


Figure 1: The Error of the Method as a Function of h, the distance between each point

As expected, as we decrease the distance between the consecutive points, the solution becomes more and more accurate to the actual solution of the system.

2 The Runge-Kutta Method

2.1 The Runge-Kutta Method Order 2 (RK2)

Consider the following differential equation,

$$\dot{y} = f(y,t) \tag{1}$$

using Taylor expansion, we expand the function $y(t+h)$,

$$y(t+h) = y(t) + h\dot{y}(t) + \frac{h^2}{2}\ddot{y}(t) + O(h^3) \tag{2}$$

since $\dot{y} = f(t,y)$,

$$\ddot{y} = \frac{d\dot{y}}{dt} = \frac{df}{dt} = \frac{df}{dt}\frac{dt}{dt} + \frac{df}{dy}\frac{dy}{dt} = f_t(y,t) + f_y(y,t)\dot{y} \tag{3}$$

we substitute equations (1) and (3) into (2), and we get,

$$y(t+h) = y(t) + hf(y,t) + \frac{h^2}{2}\left[f_t(y,t) + f_y(y,t)f(y,t)\right] + O(h^3) \tag{4}$$

using Taylor expansion again, we expand the function $f(y+\frac{k}{2}, t+\frac{h}{2}$,

$$f(y+\frac{k}{2}, t+\frac{t}{2}) = f(y,t) + \frac{h}{2}f_t + \frac{k}{2}f_y + O(h^2) \tag{5}$$

now, let $k = hf(y,t)$ and substitute equation (5) in equation (4),

$$y(t+h) = y(t) + hf(y+\frac{k}{2}, t+\frac{h}{2}) + O(h^2) \tag{6}$$

This is the Runge-Kutta Method of order 2 (RK2) where it follows an intermediary point, and look at what is happening next to correct the current point, so it doesn't overshoot with the solution.
I used this method to solve the same harmonic oscillator problem that was considered in section 1, with initial conditions $x_0 = 5\ m$, $v_0 = 0\ m/s$, and $w^2 = 10\ rad/sec$, and I constructed a function to solve using RK2 method that take the number of points, the distance between the points, and the initial conditions,

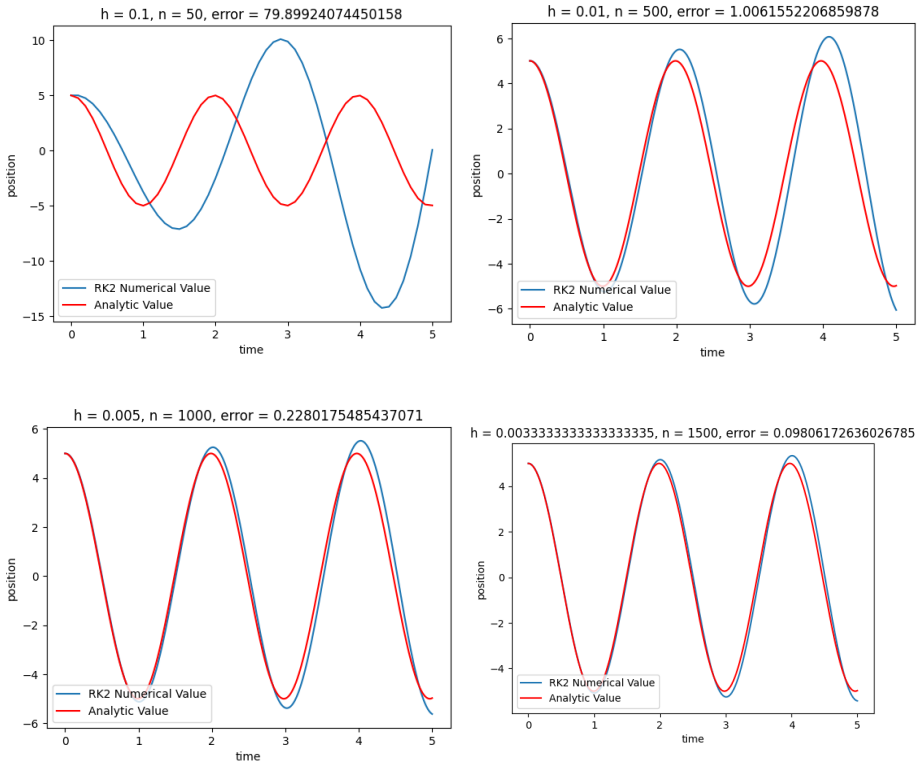
```
def RK2(h, n, x_0, v_0, w2):
    t = np.zeros(n+1)
    x = np.zeros(n+1)
    v = np.zeros(n+1)
    x[0] = x_0
    v[0] = v_0
    for i in range(1, n+1):
        k = h * (-w2 * x[i-1])
        x[i] = x[i - 1] + h * v[i - 1]
        v[i] = v[i - 1] - h * w2 * (x[i - 1] + (k/2))
        t[i] = t[i - 1] + h
    return t, x
```

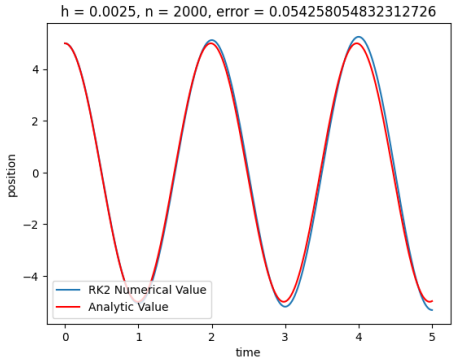
and I applied it to my initial conditions and and calculated the root mean square error of the method, $error = \frac{1}{N}\sum_1^N (x_{analytic} - x_{numerical})^2$,

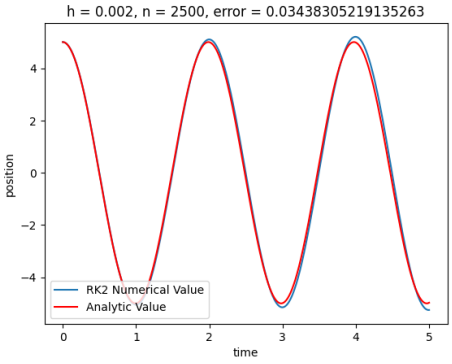
```
hs = []
errors = []
for j in range(0, 101, 10):
    if j == 0:
        h = 0.1
        n = 50
    else:
        h = 0.1/j
        n = 50 * j
    x_0, v_0, w2 = 5, 0, 10
    t, x = RK2(h, n, x_0, v_0, w2)
    plt.plot(t, x, label = "RK2 Numerical Value")
    x2 = 5 * np.cos(np.sqrt(w2)*t)
    error= np.sum((x2-x) ** 2)/n
    errors.append(error)
    hs.append(h)
    plt.plot(t, x2, color = 'r', label = "Analytic Value")
    print("h = ", h, " n =  ", n, " error = ", error)
    plt.legend(loc = 'lower left')
    plt.xlabel("time")
    plt.ylabel("position")
    plt.title(f'h = {h}, n = {n}, error = {error}')
    plt.show()
print("hs = ", hs, "\nerrors = ", errors)
plt.plot(hs, errors)
plt.scatter(hs, errors)
plt.xlabel("distance between the points")
plt.ylabel("error")
plt.show()
```

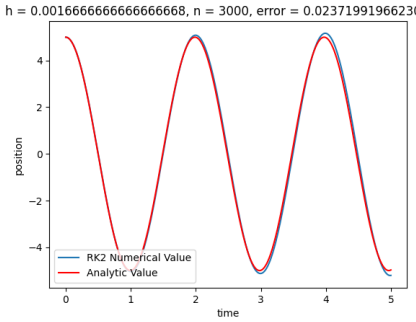
and these are the results I got,

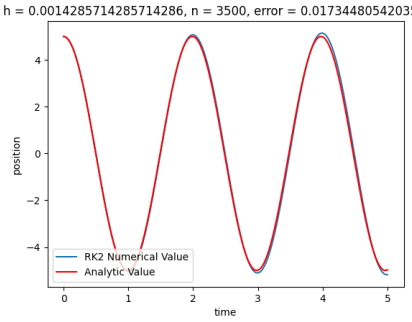
h	number of points	error
0.1	50	79.899
0.01	500	1.006
0.005	1000	0.228
0.003	1500	0.098
0.0025	2000	0.054
0.002	2500	0.034
0.001	3000	0.023
0.001	3500	0.017
0.00125	4000	0.013
0.00111	4500	0.010
0.001	5000	0.008

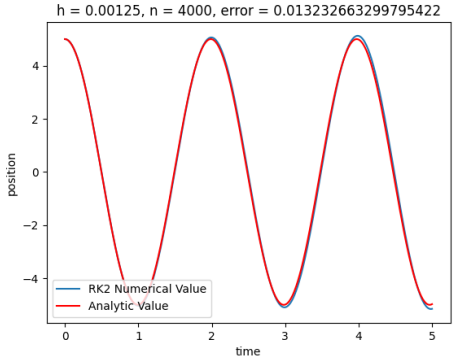


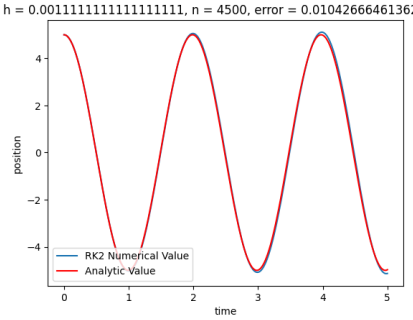

h = 0.0025, n = 2000, error = 0.054258054832312726
position
time
RK2 Numerical Value
Analytic Value

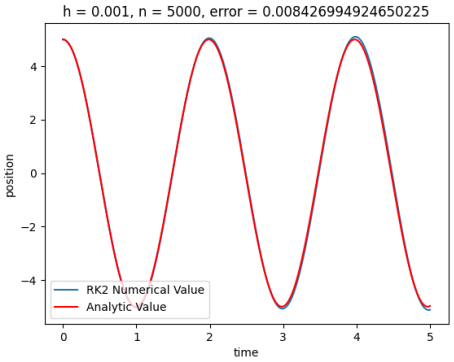

h = 0.002, n = 2500, error = 0.03438305219135263
position
time
RK2 Numerical Value
Analytic Value


h = 0.0016666666666666668, n = 3000, error = 0.023719919662306424
position
time
RK2 Numerical Value
Analytic Value


h = 0.0014285714285714286, n = 3500, error = 0.017344805420357646
position
time
RK2 Numerical Value
Analytic Value


h = 0.00125, n = 4000, error = 0.013232663299795422
position
time
RK2 Numerical Value
Analytic Value


h = 0.0011111111111111111, n = 4500, error = 0.010426664613626638
position
time
RK2 Numerical Value
Analytic Value



The error as a function of the distance between the points is,

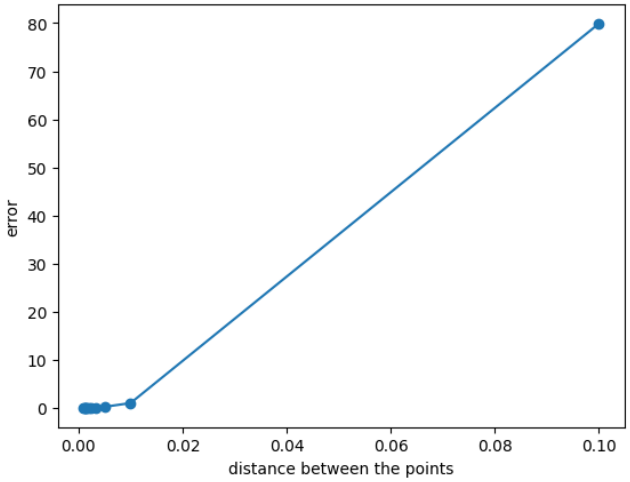


Figure 2: The Error of the Method as a Function of h, the distance between each point

We can conclude that the method can conclude accurate solution with small region of errors. As we expected, the method also gets more accurate as we decrease the distance between the points.

2.2 The Runge-Kutta Method Order 4 (RK4)

For the Runge-Kutta Method of order 4 (RK4), we do the same thing as RK2, but we truncate the expansion at the fifth order,

$$y(t+h) = y(t) + h\frac{dy}{dt} + \frac{h^2}{2!}\frac{d^2y}{dt^2} + \frac{h^3}{3!}\frac{d^3y}{dt^3} + \frac{h^4}{4!}\frac{d^4y}{dt^4} + O(h^5)$$

$$\Rightarrow y(t+h) = y(t) + h\dot{f}(y,t) + \frac{h^2}{2!}\ddot{f}(y,t) + \frac{h^3}{3!}\dddot{f}(y,t) + \frac{h^4}{4!}\ddddot{f}(y,t) + O(h^5)$$

after some development and substitution, the above equation becomes,

$$y(t+h) = y(t) + \frac{1}{6}[k_1 + 2k_2 + 2k_3 + k_4] + O(h^4)$$

where

$$k_1 = hf(t,y)$$

$$k_2 = hf\left(t + \frac{h}{2}, y + \frac{k_1}{2}\right)$$

$$k_3 = hf\left(t + \frac{h}{2}, y + \frac{k_2}{2}\right)$$

$$k_4 = hf\left(t + \frac{h}{2}, y + k_3\right)$$

Again, I used this method to solve a harmonic oscillator system, like sections 1 and 2, with initial conditions $x_0 = 5\ m$, $v_0 = 0\ m/s$, and $w^2 = 10\ rad/sec$, and I constructed a function to solve using RK2 method that take the number of points, the distance between the points, and the initial conditions,

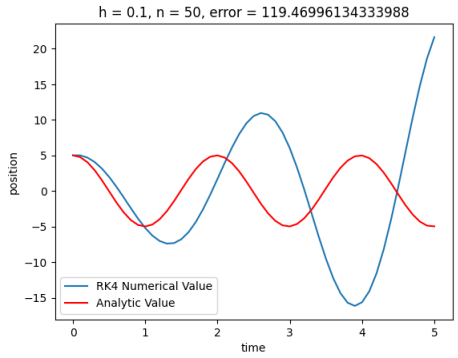
```
def RK4(h, n, x_0, v_0, w2):
    t = np.zeros(n+1)
    x = np.zeros(n+1)
    v = np.zeros(n+1)
    x[0] = x_0
    v[0] = v_0
    for i in range(1, n + 1):
        k1 = h * (-w2 * x[i-1])
        k2 = h * (-w2 * (x[i-1] + (k1/2)))
        k3 = h * (-w2 * (x[i-1] + (k2/2)))
        k4 = h * (-w2 * (x[i-1] + k3))
        x[i] = x[i - 1] + h * (v[i - 1])
        v[i] = v[i - 1] + (k1 + 2 * k2 + 2 * k3 + k4)/6
        t[i] = t[i - 1] + h
    return t, x
```

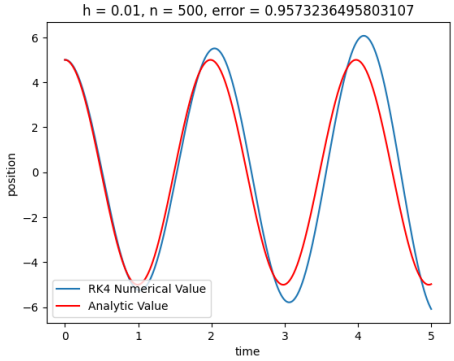
and I used this method to solve the system with the given initial conditions,

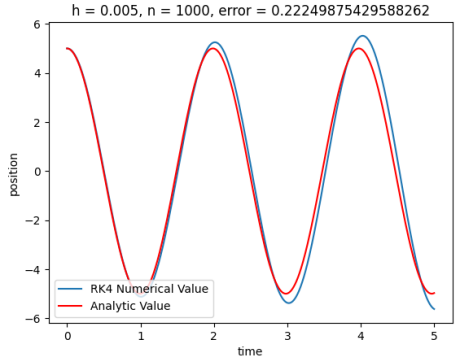
```
hs = []
errors = []
for j in range(0, 101, 10):
    if j == 0:
        h = 0.1
        n = 50
    else:
        h = 0.1/j
        n = 50 * j
    x_0, v_0, w2 = 5, 0, 10
    t, x = RK4(h, n, x_0, v_0, w2)
    plt.plot(t, x, label = "RK4 Numerical Value")
    x2 = 5 * np.cos(np.sqrt(w2)*t)
    error= np.sum((x2-x) ** 2)/n
    errors.append(error)
    hs.append(h)
    plt.plot(t, x2, color = 'r', label = "Analytic Value")
    print("h = ", h, " n =  ", n, " error = ", error)
    plt.legend(loc = 'lower left')
    plt.xlabel("time")
    plt.ylabel("position")
    plt.title(f'h = {h}, n = {n}, error = {error}')
    plt.show()
print("hs = ", hs, "\nerrors = ", errors)
plt.plot(hs, errors)
plt.scatter(hs, errors)
plt.xlabel("distance between the points")
plt.ylabel("error")
plt.show()
```

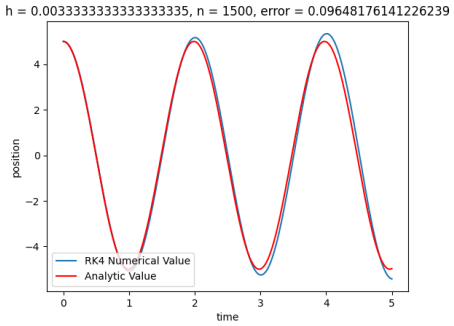
and these are the results I got,

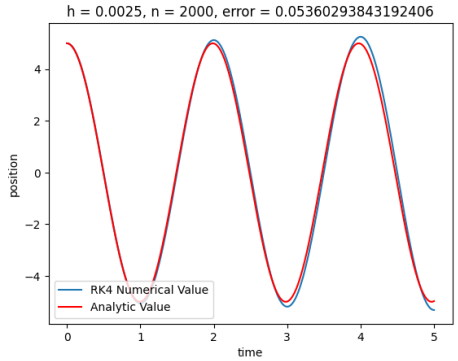
h	number of points	error
0.1	50	119.46
0.01	500	0.957
0.005	1000	0.222
0.003	1500	0.096
0.0025	2000	0.053
0.002	2500	0.034
0.001	3000	0.023
0.001	3500	0.017
0.00125	4000	0.013
0.00111	4500	0.010
0.001	5000	0.008

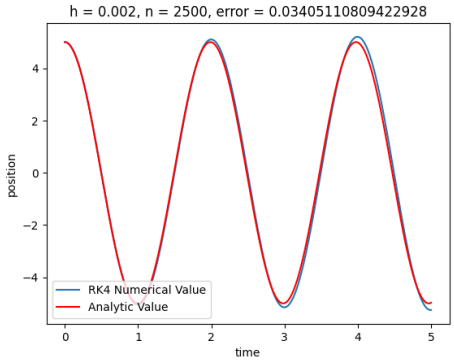

h = 0.1, n = 50, error = 119.46996134333988
position
time
RK4 Numerical Value
Analytic Value

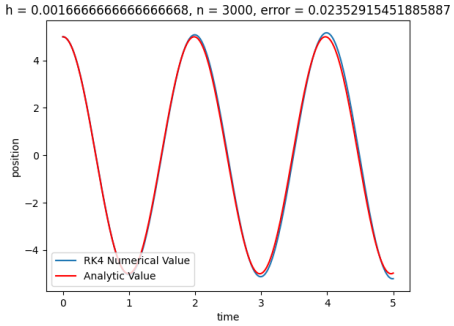

h = 0.01, n = 500, error = 0.9573236495803107
position
time
RK4 Numerical Value
Analytic Value

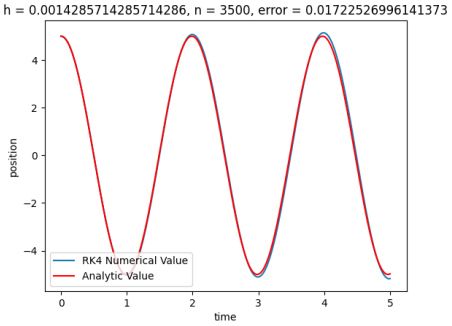

h = 0.005, n = 1000, error = 0.22249875429588262
position
time
RK4 Numerical Value
Analytic Value

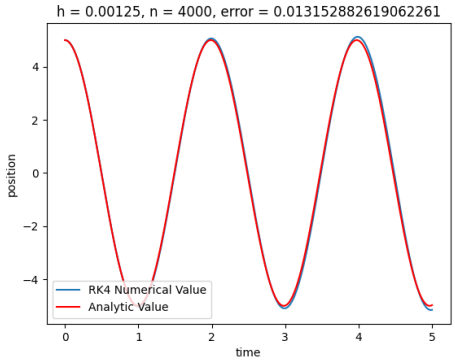

h = 0.0033333333333333335, n = 1500, error = 0.09648176141226239
position
time
RK4 Numerical Value
Analytic Value

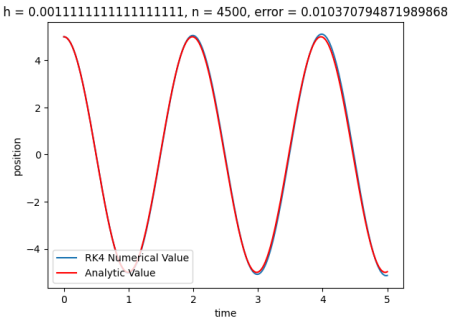

h = 0.0025, n = 2000, error = 0.05360293843192406
position
time
RK4 Numerical Value
Analytic Value

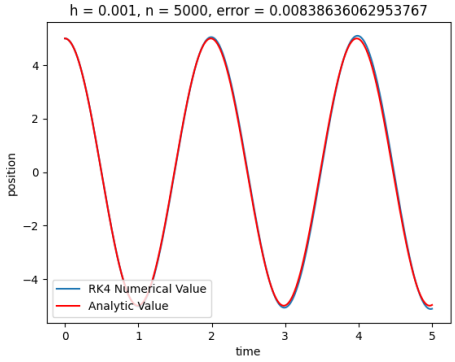

h = 0.002, n = 2500, error = 0.03405110809422928
position
time
RK4 Numerical Value
Analytic Value


h = 0.0016666666666666668, n = 3000, error = 0.02352915451885887
position
time
RK4 Numerical Value
Analytic Value


h = 0.0014285714285714286, n = 3500, error = 0.01722526996141373
position
time
RK4 Numerical Value
Analytic Value


h = 0.00125, n = 4000, error = 0.013152882619062261
position
time
RK4 Numerical Value
Analytic Value


h = 0.0011111111111111111, n = 4500, error = 0.010370794871989868
position
time
RK4 Numerical Value
Analytic Value


h = 0.001, n = 5000, error = 0.00838636062953767
position
time
RK4 Numerical Value
Analytic Value

The error as a function of the distance between the points is,

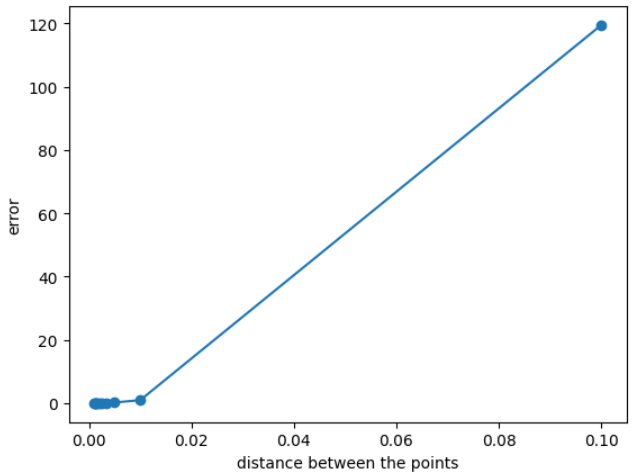


Figure 3: The Error of the Method as a Function of h, the distance between each point

As the other method, as the distances between the points decrease, the more accurate the solution is. However, this yields a more accurate result.

2.3 Comparison Between RK2 and RK4

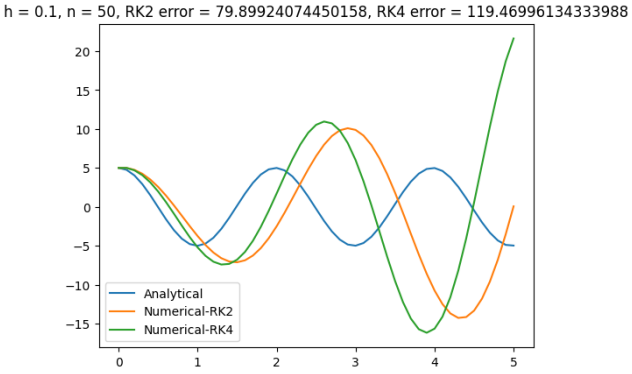
I compared the two orders of Runge-Kutta method in solving the same harmonic oscillator system,

```
hs = []
errors1 = []
errors2 = []
for j in range(0, 101, 10):
    if j == 0:
        h = 0.1
        n = 50
    else:
        h = 0.1/j
        n = 50 * j
    x_0, v_0, w2 = 5, 0, 10
    t1, x1 = RK2(h, n, x_0, v_0, w2)
    t2, x2 = RK4(h, n, x_0, v_0, w2)
    x3 = x_0 * np.cos(np.sqrt(w2) * t1)
    plt.plot(t2, x3, label = 'Analytical')
    plt.plot(t1, x1, label = 'Numerical-RK2')
    plt.plot(t2, x2, label = 'Numerical-RK4')
```

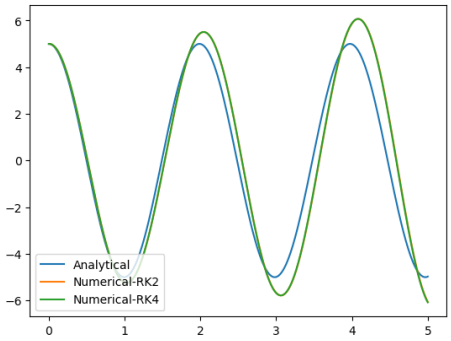
```
    plt.legend(loc = "lower left")
    error1= np.sum((x3-x1) ** 2)/n
    errors1.append(error1)
    error2 = np.sum((x3-x2) ** 2)/n
    errors2.append(error2)
    hs.append(h)
    print("h = ", h, " n = ", n, " RK2 Error = ", error1, " RK4 Error = ", error2)
    plt.title(f'h = {h}, n = {n}, RK2 error = {error1}, RK4 error = {error2}')
    plt.show()
plt.plot(hs, errors1, label = "RK2 Errors")
plt.scatter(hs, errors1)
plt.scatter(hs, errors2)
plt.plot(hs, errors2, label = "RK4 Errors")
plt.xlabel("distance between the points")
plt.ylabel("error")
plt.show()
```

and these are the results I obtained,

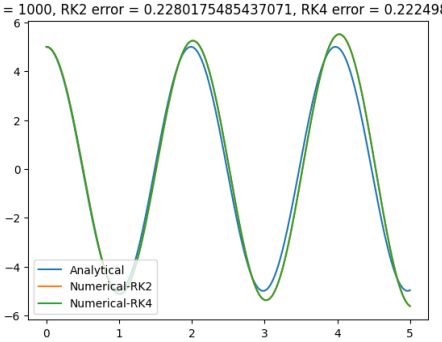
h	number of points	RK2 error	RK4 error
0.1	50	79.89	119.46
0.01	500	1.006	0.957
0.005	1000	0.228	0.222
0.003	1500	0.098	0.096
0.0025	2000	0.054	0.053
0.002	2500	0.034	0.034
0.001	3000	0.023	0.023
0.001	3500	0.017	0.017
0.00125	4000	0.013	0.013
0.00111	4500	0.010	0.010
0.001	5000	0.008	0.008



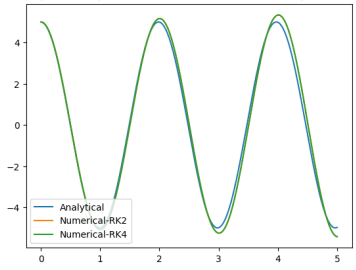
h = 0.01, n = 500, RK2 error = 1.0061552206859878, RK4 error = 0.9573236495803107



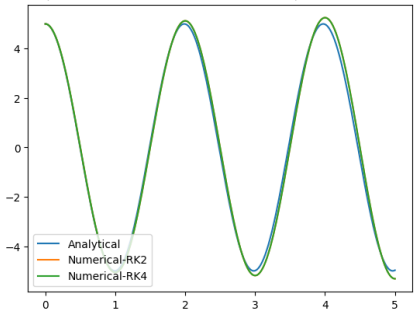
h = 0.005, n = 1000, RK2 error = 0.2280175485437071, RK4 error = 0.22249875429588262



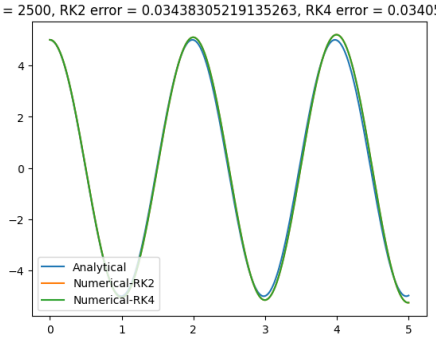
h = 0.0033333333333333335, n = 1500, RK2 error = 0.09806172636026785, RK4 error = 0.09648176141226239



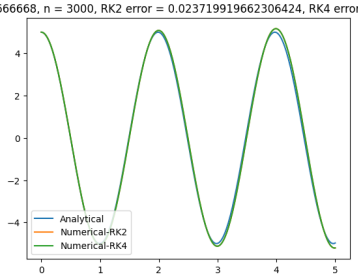
h = 0.0025, n = 2000, RK2 error = 0.054258054832312726, RK4 error = 0.05360293843192406



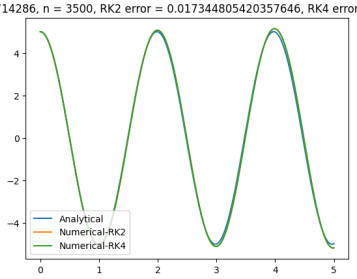
h = 0.002, n = 2500, RK2 error = 0.03438305219135263, RK4 error = 0.03405110809422928



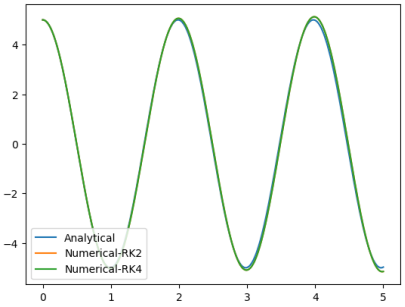
h = 0.0016666666666666668, n = 3000, RK2 error = 0.023719919662306424, RK4 error = 0.02352915451885887



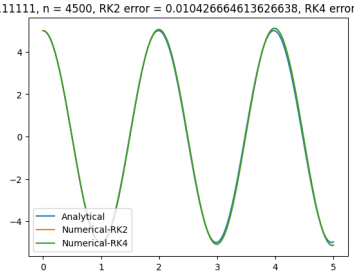
h = 0.0014285714285714286, n = 3500, RK2 error = 0.017344805420357646, RK4 error = 0.01722526996141373



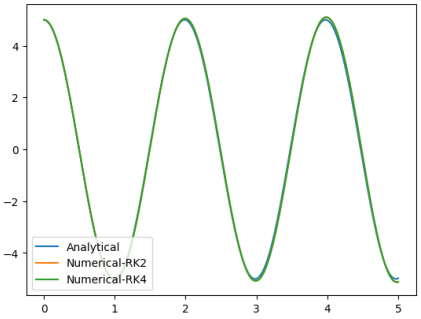
h = 0.00125, n = 4000, RK2 error = 0.013232663299795422, RK4 error = 0.013152882619062261



h = 0.0011111111111111111, n = 4500, RK2 error = 0.010426664613626638, RK4 error = 0.010370794871989868



h = 0.001, n = 5000, RK2 error = 0.008426994924650225, RK4 error = 0.00838636062953767



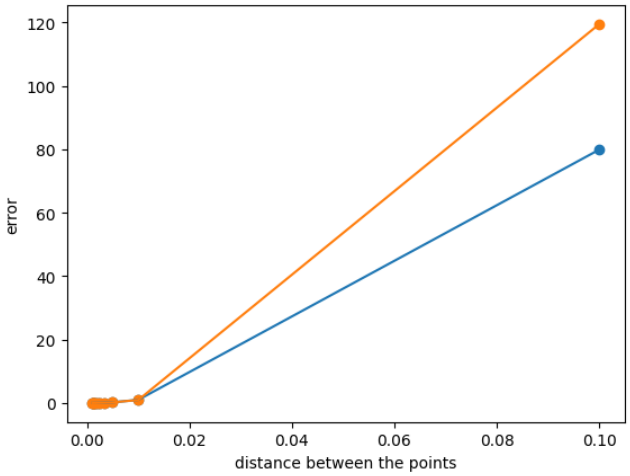


Figure 4: The Error of the Method as a Function of h, the distance between each point

Comparing the two method, we can see that with bigger h, RK4 had a lower error, hence it is more accurate. But, as the distance decreased, the method started getting identical and indistinguishable; I am not sure if this is how it is supposed to be, or it there is an error in computing one of the methods.