Computational Astrophysics
Assignment # 6

Name: Abraham Mathews
Student Code: SC22M077
Course: MS Astronomy and Astrophysics

Determine the evolution of the position (x) and velocity (dx/dt) for a mass m=*1* on a spring of k=*1*.

The governing equation is,
$\frac{d^2x}{dt^2} = -x$ in the domain $0 \leq t \leq 10$ with initial conditions $x(t = 0) = 0$ and $\frac{dx}{dt}(t = 0) = 1$.

Use (i) simple Euler, (ii) modified Euler, (iii) improved Euler, and (iv) RK4.

1. Plot the position, velocity, and the total energy as a function of time. How much does the computed total energy deviate from its expected value for the four cases?

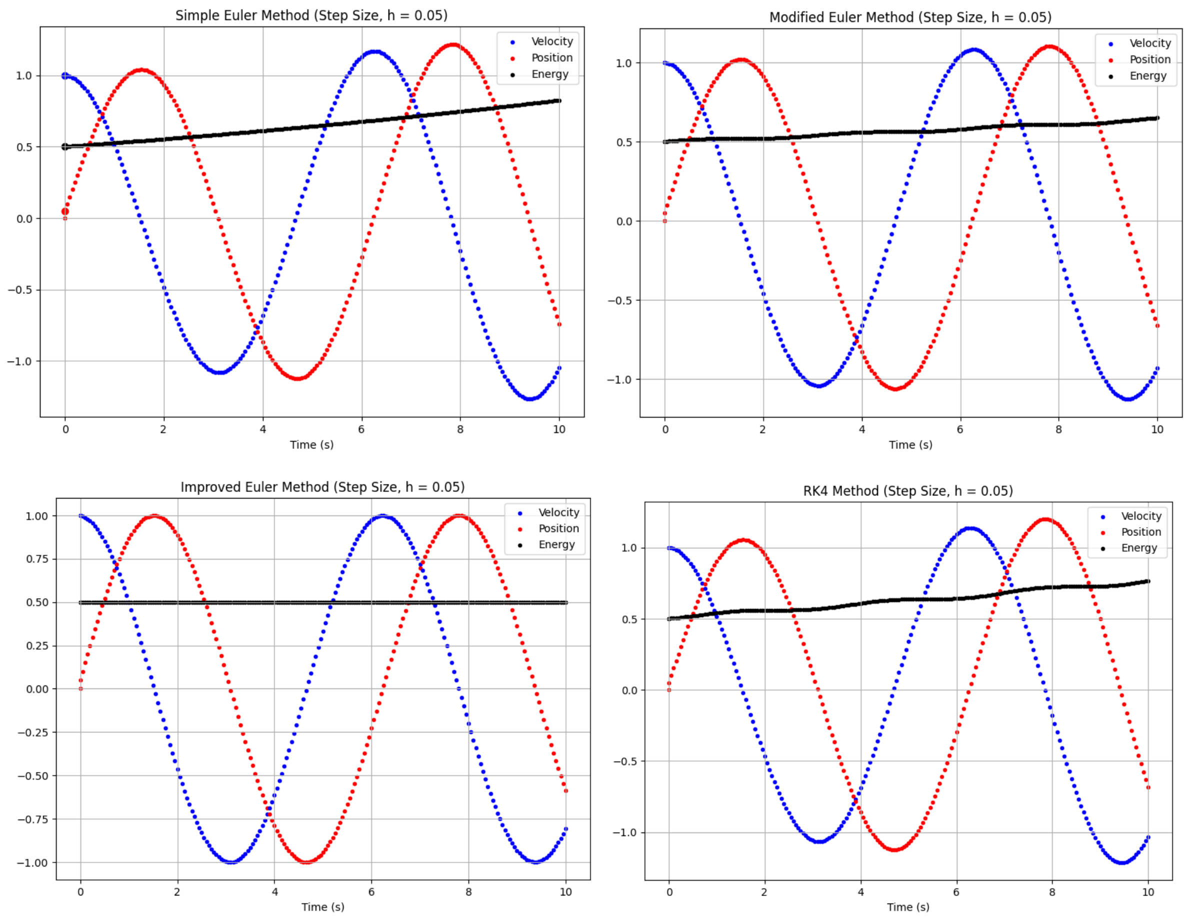


Fig. 1: Plot of position, velocity and energy using different methods

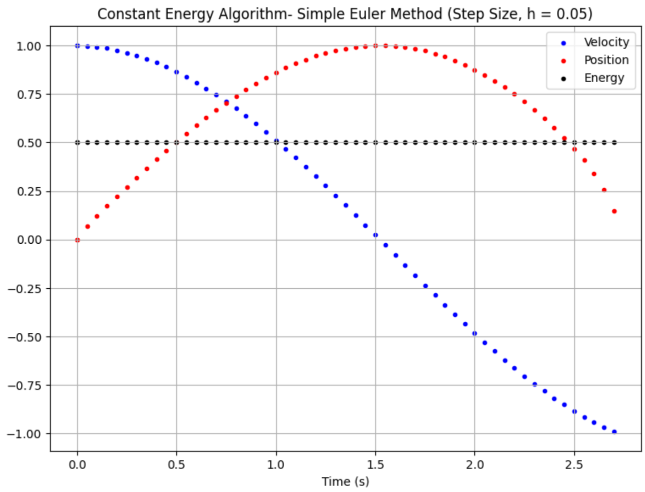
Deviation of Computed Total energy

In this case **Theoretical Total Energy** = (1/2)
Deviation is calculated as (**Numerically Calculated Total Energy - Theoretical Total Energy)**

Sl. No.	Method	Maximum Deviation of Total Energy from actual value (Step size h = 0.05s)
1	Simple Euler	0.3259060435631598
2	Modified Euler	0.15039425813738283
3	Improved Euler	0.00015705578867575287
4	RK4	0.2675458212900216

---

2. Use the fact that the total energy is a constant of motion and redo the calculations for the simple Euler method. Compare the total time taken for the run and the total truncation error at last step (t=10) with that of RK4 in the previous calculation. You can use the total energy as a probe for truncation error.



Here Total Energy E = (1/2)

Time at which computation stopped, t = 2.7499999999999982 s

This is because beyond this point, the magnitude of velocity calculated by the Simple Euler becomes more than 1. Therefore, when we calculate position using,

$$x(t) = \sqrt{(2 * E) - v(t)^2}$$

it becomes imaginary, then it is not possible to continue the computation.

Python Codes

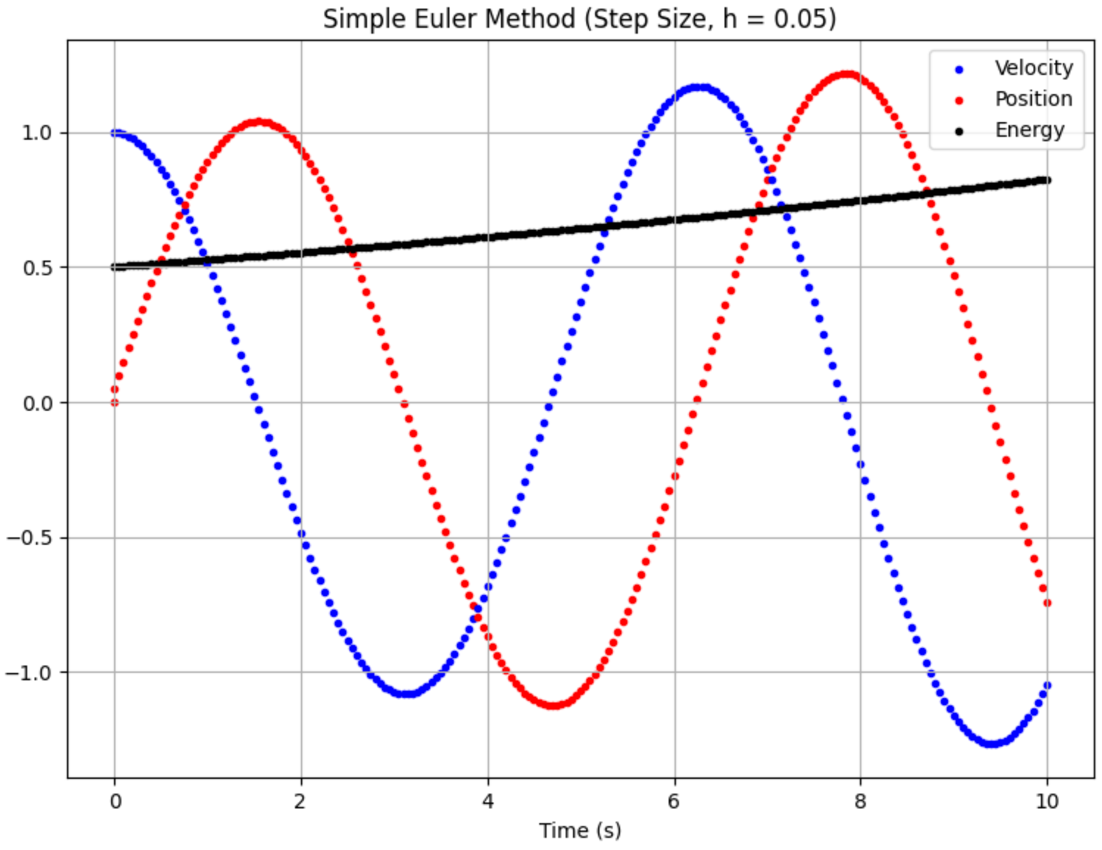
Q1.

- Simple Euler
- Modified Euler
- Improved Euler
- RK4

In [2]:
```
# A) Simpler Euler

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

import numpy as np
import matplotlib.pyplot as plt
plt.style.use('default')
fig = plt.figure(figsize = (9,6.5))

def f(x,t):
    ret = -x
    return(ret)

def g(v,t):
    ret = v
    return(ret)

def euler(t0,y0,f_xy0,h):
    ret = y0 + (h*f_xy0)
    return(ret)

def energy(x,v):
    E = (pow(v,2)+pow(x,2))/2
    return(E)
# for h in [0.02,0.1,0.2]:
h = 0.05
a = 0
b = 10
t = np.arange(a,b,h)
n = len(t)
#print("t = ",t)

# Initial condition (t = 0)
t0 = 0
x0 = 0
v0 = 1

plt.scatter(t0,v0,color = "blue",label="Velocity",marker='.')
plt.scatter(t0,x0,color = "red",label="Position",marker='.')
plt.scatter(t0,energy(x0,v0),color = "black",label="Energy",marker
='.')


f0 = f(x0,t0)
g0 = g(v0,t0)

v1 = euler(t0,v0,f0,h)
x1 = euler(t0,x0,g0,h)

plt.scatter(t0,v1,color = "blue",marker='.')
plt.scatter(t0,x1,color = "red",marker='.')
plt.scatter(t0,energy(x1,v1),color = "black",marker='.')


for i in t:
    t0 = t0 + h
    v0 = v1
```

```
    x0 = x1
    #print("x0,y0 = ",x0,y0)
    f0 = f(x0,t0)
    g0 = g(v0,t0)

    v1 = euler(t0,v0,f0,h)
    x1 = euler(t0,x0,g0,h)

    plt.scatter(t0,v1,color = "blue",marker='.')
    plt.scatter(t0,x1,color = "red",marker='.')
    plt.scatter(t0,energy(x1,v1),color = "black",marker='.')
deviation_energy = energy(x1,v1) - (1/2)
print("Maximum energy deviation = ",deviation_energy)
#x_actual = np.linspace(0,1,5000)
#simple_euler_plt = plt.plot(x_actual,np.tan(x_actual),color="re
d")
plt.grid()
plt.legend()
plt.title("Simple Euler Method (Step Size, h = 0.05)")
plt.xlabel("Time (s)")
plt.show()
```

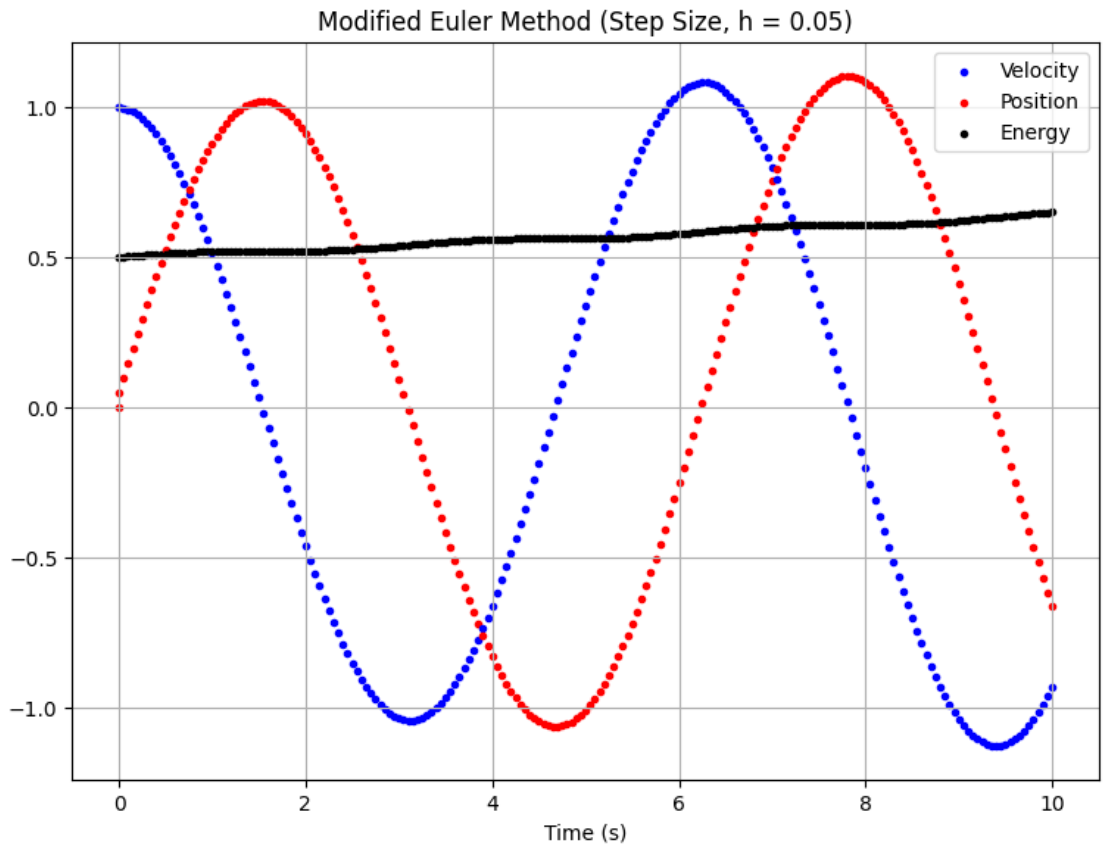
```
Maximum energy deviation =  0.3259060435631598
```



In [3]:
```
# B) Modified Euler

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

import numpy as np
import matplotlib.pyplot as plt
plt.style.use('default')
fig = plt.figure(figsize = (9,6.5))

def f(x,t):
    ret = -x
    return(ret)

def g(v,t):
    ret = v
    return(ret)

def euler_mod_v(t0,x0,v0,f_xv0,g_xv0,h):
    tmid = t0 + (h/2)
    xmid = x0 + (h/2)*(g_xv0)
    f_xy0 = f(xmid,tmid)
    ret = v0 + ((h)*f_xv0)
    return(ret)

def euler_mod_x(t0,x0,v0,f_xv0,g_xv0,h):
    tmid = t0 + (h/2)
    vmid = v0 + (h/2)*(f_xv0)
    g_xv0 = g(vmid,tmid)
    ret = x0 + ((h)*g_xv0)
    return(ret)

def energy(x,v):
    E = (pow(v,2)+pow(x,2))/2
    return(E)
# for h in [0.02,0.1,0.2]:
h = 0.05
a = 0
b = 10
t = np.arange(a,b,h)
n = len(t)
#print("t = ",t)

# Initial condition (t = 0)
t0 = 0
x0 = 0
v0 = 1

plt.scatter(t0,v0,color = "blue",label="Velocity",marker='.')
plt.scatter(t0,x0,color = "red",label="Position",marker='.')
plt.scatter(t0,energy(x0,v0),color = "black",label="Energy",marker
='.')


f0 = f(x0,t0)
g0 = g(v0,t0)

v1 = euler_mod_v(t0,x0,v0,f0,g0,h)
```

```
x1 = euler_mod_x(t0,x0,v0,f0,g0,h)


plt.scatter(t0,v1,color = "blue",marker='.')
plt.scatter(t0,x1,color = "red",marker='.')
plt.scatter(t0,energy(x1,v1),color = "black",marker='.')


for i in t:
    t0 = t0 + h
    v0 = v1
    x0 = x1
    #print("x0,y0 = ",x0,y0)
    f0 = f(x0,t0)
    g0 = g(v0,t0)

    v1 = euler_mod_v(t0,x0,v0,f0,g0,h)
    x1 = euler_mod_x(t0,x0,v0,f0,g0,h)

    plt.scatter(t0,v1,color = "blue",marker='.')
    plt.scatter(t0,x1,color = "red",marker='.')
    plt.scatter(t0,energy(x1,v1),color = "black",marker='.')
deviation_energy = energy(x1,v1) - (1/2)
print("Maximum energy deviation = ",deviation_energy)
#x_actual = np.linspace(0,1,5000)
#simple_euler_plt = plt.plot(x_actual,np.tan(x_actual),color="re
d")
plt.grid()
plt.legend()
plt.title("Modified Euler Method (Step Size, h = 0.05)")
plt.xlabel("Time (s)")
plt.show()
```

```
Maximum energy deviation =  0.15039425813738283
```

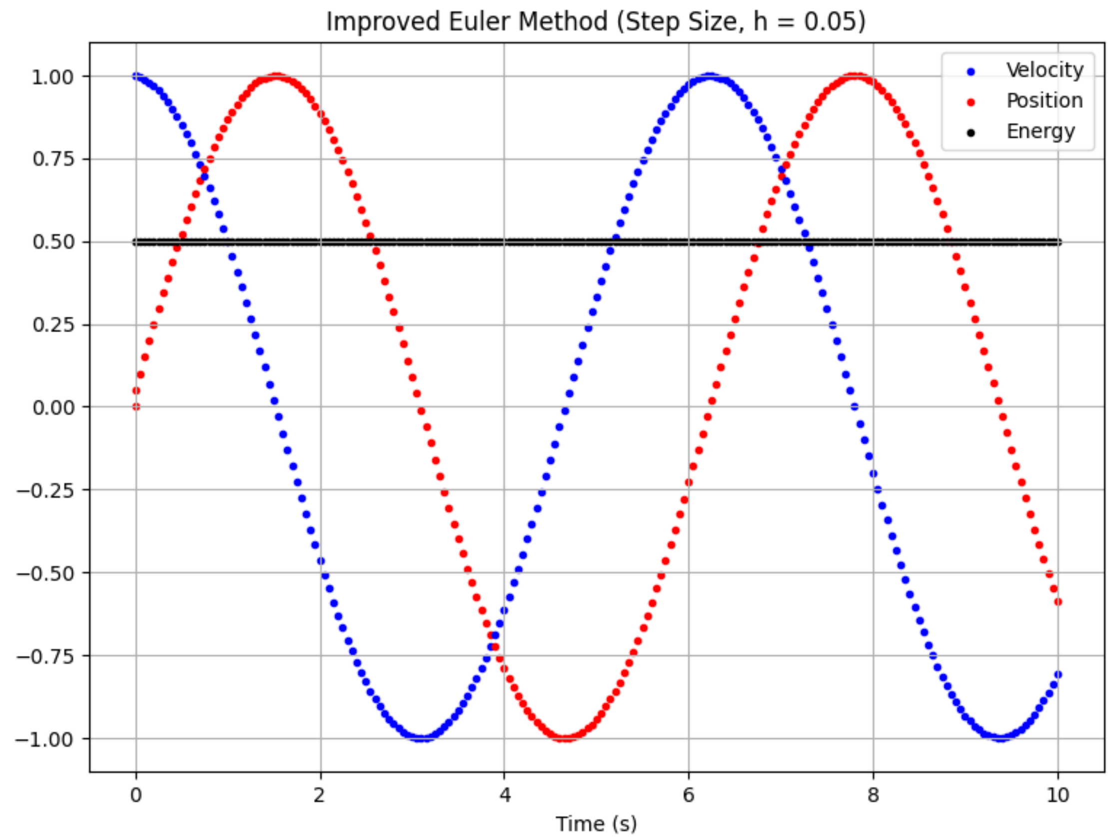


In [4]:

```
# C) Improved Euler Method

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

import numpy as np
import matplotlib.pyplot as plt
plt.style.use('default')
fig = plt.figure(figsize = (9,6.5))

def f(x,t):
    ret = -x
    return(ret)

def g(v,t):
    ret = v
    return(ret)

def euler_imp_v(t0,x0,v0,f_xv0,g_xv0,h):
    tend = t0 + h
    xend = x0 + (h)*(g_xv0)
    f_xv_end = f(xend,tend)
    ret = v0 + ((h)*(f_xv0+f_xv_end)/2)
    return(ret)

def euler_imp_x(t0,x0,v0,f_xv0,g_xv0,h):
    tend = t0 + (h)
    vend = v0 + (h)*(f_xv0)
    g_xv_end = g(vend,tend)
    ret = x0 + ((h)*(g_xv0+g_xv_end)/2)
    return(ret)

def energy(x,v):
    E = (pow(v,2)+pow(x,2))/2
    return(E)
# for h in [0.02,0.1,0.2]:
h = 0.05
a = 0
b = 10
t = np.arange(a,b,h)
n = len(t)
#print("t = ",t)

# Initial condition (t = 0)
t0 = 0
x0 = 0
v0 = 1

plt.scatter(t0,v0,color = "blue",label="Velocity",marker='.')
plt.scatter(t0,x0,color = "red",label="Position",marker='.')
plt.scatter(t0,energy(x0,v0),color = "black",label="Energy",marker
='.')


f0 = f(x0,t0)
g0 = g(v0,t0)

v1 = euler_imp_v(t0,x0,v0,f0,g0,h)
```

```
x1 = euler_imp_x(t0,x0,v0,f0,g0,h)


plt.scatter(t0,v1,color = "blue",marker='.')
plt.scatter(t0,x1,color = "red",marker='.')
plt.scatter(t0,energy(x1,v1),color = "black",marker='.')


for i in t:
    t0 = t0 + h
    v0 = v1
    x0 = x1
    #print("x0,y0 = ",x0,y0)
    f0 = f(x0,t0)
    g0 = g(v0,t0)

    v1 = euler_imp_v(t0,x0,v0,f0,g0,h)
    x1 = euler_imp_x(t0,x0,v0,f0,g0,h)

    plt.scatter(t0,v1,color = "blue",marker='.')
    plt.scatter(t0,x1,color = "red",marker='.')
    plt.scatter(t0,energy(x1,v1),color = "black",marker='.')
deviation_energy = energy(x1,v1) - (1/2)
print("Maximum energy deviation = ",deviation_energy)
#x_actual = np.linspace(0,1,5000)
#simple_euler_plt = plt.plot(x_actual,np.tan(x_actual),color="re
d")
plt.grid()
plt.legend()
plt.title("Improved Euler Method (Step Size, h = 0.05)")
plt.xlabel("Time (s)")
plt.show()
```

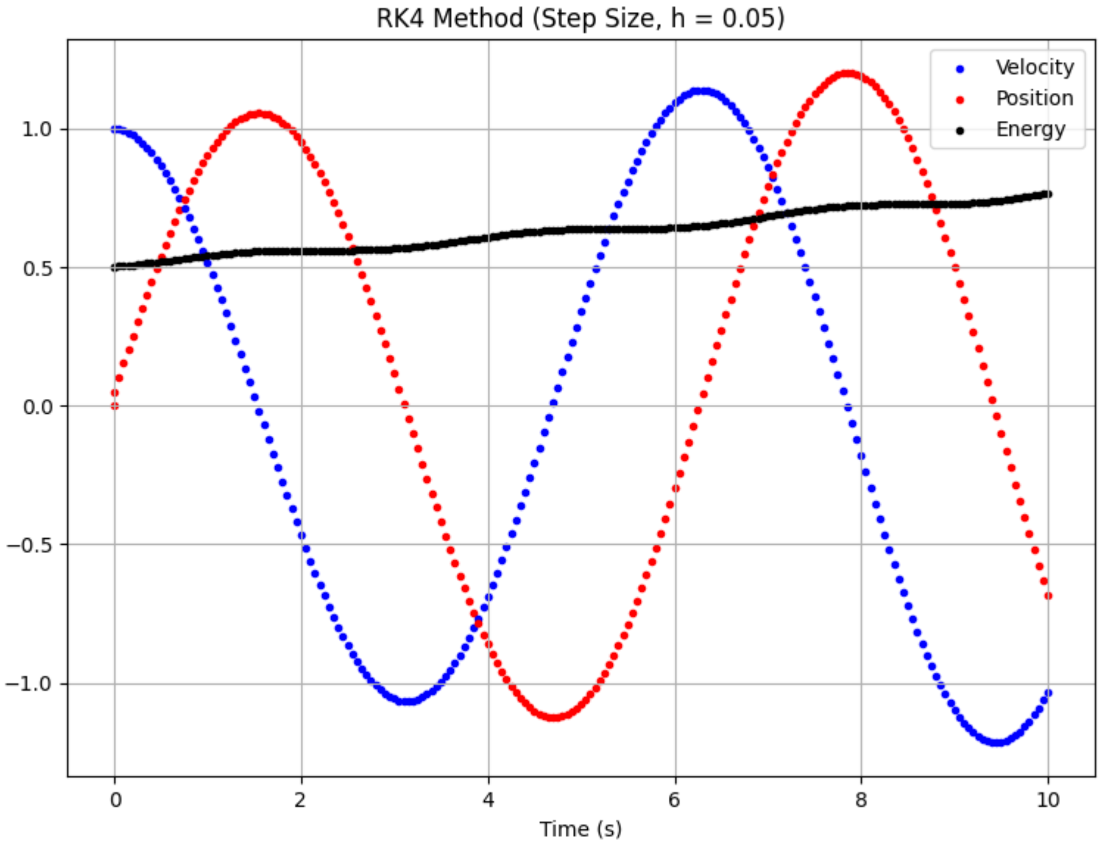
```
Maximum energy deviation =  0.00015705578867575287
```



In [5]:
```python
# D) RK4

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

import numpy as np
import matplotlib.pyplot as plt
plt.style.use('default')
fig = plt.figure(figsize = (9,6.5))

def f(x,t):
    ret = -x
    return(ret)

def g(v,t):
    ret = v
    return(ret)


def RK4_v(t0,x0,v0,f_xv0,g_xv0,h):
    f1 = f((x0+((h/2)*f_xv0)),(t0+(h/2)))
    f2 = f((x0+((h/2)*f1)),(t0+(h/2)))
    f3 = f((x0+((h)*f2)),(t0+(h)))
    ret = v0 + (h/6)*(f_xv0+(2*(f1+f2))+f3)
    return(ret)

def RK4_x(t0,x0,v0,f_xv0,g_xv0,h):
    g1 = g((v0+((h/2)*f_xv0)),(t0+(h/2)))
    g2 = g((v0+((h/2)*g1)),(t0+(h/2)))
    g3 = g((v0+((h)*g2)),(t0+(h)))
    ret = x0 + (h/6)*(g_xv0+(2*(g1+g2))+g3)
    return(ret)

def energy(x,v):
    E = (pow(v,2)+pow(x,2))/2
    return(E)
# for h in [0.02,0.1,0.2]:
h = 0.05
a = 0
b = 10
t = np.arange(a,b,h)
n = len(t)
#print("t = ",t)

# Initial condition (t = 0)
t0 = 0
x0 = 0
v0 = 1

plt.scatter(t0,v0,color = "blue",label="Velocity",marker='.')
plt.scatter(t0,x0,color = "red",label="Position",marker='.')
plt.scatter(t0,energy(x0,v0),color = "black",label="Energy",marker
='.')


f0 = f(x0,t0)
g0 = g(v0,t0)
```

```python
v1 = RK4_v(t0,x0,v0,f0,g0,h)
x1 = RK4_x(t0,x0,v0,f0,g0,h)


plt.scatter(t0,v1,color = "blue",marker='.')
plt.scatter(t0,x1,color = "red",marker='.')
plt.scatter(t0,energy(x1,v1),color = "black",marker='.')


for i in t:
    t0 = t0 + h
    v0 = v1
    x0 = x1
    #print("x0,y0 = ",x0,y0)
    f0 = f(x0,t0)
    g0 = g(v0,t0)

    v1 = RK4_v(t0,x0,v0,f0,g0,h)
    x1 = RK4_x(t0,x0,v0,f0,g0,h)

    plt.scatter(t0,v1,color = "blue",marker='.')
    plt.scatter(t0,x1,color = "red",marker='.')
    plt.scatter(t0,energy(x1,v1),color = "black",marker='.')
deviation_energy = energy(x1,v1) - (1/2)
print("Maximum energy deviation = ",deviation_energy)
#x_actual = np.linspace(0,1,5000)
#simple_euler_plt = plt.plot(x_actual,np.tan(x_actual),color="re
d")
plt.grid()
plt.legend()
plt.title("RK4 Method (Step Size, h = 0.05)")
plt.xlabel("Time (s)")
plt.show()
```

```
Maximum energy deviation =  0.2675458212900216
```



Q2

Constant Energy Algorithm

In [6]:
```
# Q2
# Simpler Euler
# Constant Energy Algorithm

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

import numpy as np
import matplotlib.pyplot as plt
plt.style.use('default')
fig = plt.figure(figsize = (9,6.5))

def f(x,t):
    ret = -x
    return(ret)

def g(v,t):
    ret = v
    return(ret)

def euler(t0,y0,f_xy0,h):
    ret = y0 + (h*f_xy0)
    return(ret)

def energy(x,v):
    E = (pow(v,2)+pow(x,2))/2
    return(E)

def calc_x1(E,v1,x1_sign):
    if((2*E) - pow(v1,2))>0:
        if x1_sign >= 0:
            x1 = math.sqrt((2*E) - pow(v1,2))
        elif x1_sign < 0:
            x1 = -1*math.sqrt((2*E) - pow(v1,2))
        return(x1)
    else:
        return(0)
# for h in [0.02,0.1,0.2]:
h = 0.05
a = 0
b = 10
t = np.arange(a,b,h)
n = len(t)
#print("t = ",t)

# Initial condition (t = 0)
t0 = 0
x0 = 0
v0 = 1

plt.scatter(t0,v0,color = "blue",label="Velocity",marker='.')
plt.scatter(t0,x0,color = "red",label="Position",marker='.')
plt.scatter(t0,energy(x0,v0),color = "black",label="Energy",marker
='.')


f0 = f(x0,t0)
g0 = g(v0,t0)
```

```
v1 = euler(t0,v0,f0,h)
x1_sign = euler(t0,x0,g0,h)
E = (1/2)
x1 = calc_x1(E,v1,x1_sign)

plt.scatter(t0,v1,color = "blue",marker='.')
plt.scatter(t0,x1,color = "red",marker='.')
plt.scatter(t0,energy(x1,v1),color = "black",marker='.')


for i in t:
    t0 = t0 + h
    v0 = v1
    x0 = x1_sign
    #print("x0,y0 = ",x0,y0)
    f0 = f(x0,t0)
    g0 = g(v0,t0)

    v1 = euler(t0,v0,f0,h)
    x1_sign = euler(t0,x0,g0,h)
    x1 = calc_x1(E,v1,x1_sign)
    if(abs(v1)>1):
        print("Time at which computation stopped, t = ", t0)
        break
    plt.scatter(t0,v1,color = "blue",marker='.')
    plt.scatter(t0,x1,color = "red",marker='.')
    plt.scatter(t0,energy(x1,v1),color = "black",marker='.')

#x_actual = np.linspace(0,1,5000)
#simple_euler_plt = plt.plot(x_actual,np.tan(x_actual),color="re
d")
plt.grid()
plt.legend()
plt.title("Constant Energy Algorithm- Simple Euler Method (Step Si
ze, h = 0.05)")
plt.xlabel("Time (s)")
plt.show()
```

```
Time at which computation stopped, t =  2.7499999999999982
```