hw11_code

November 19, 2021

Michael Goforth CAAM 550 HW 11 Due 11/19/2021

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import math
```

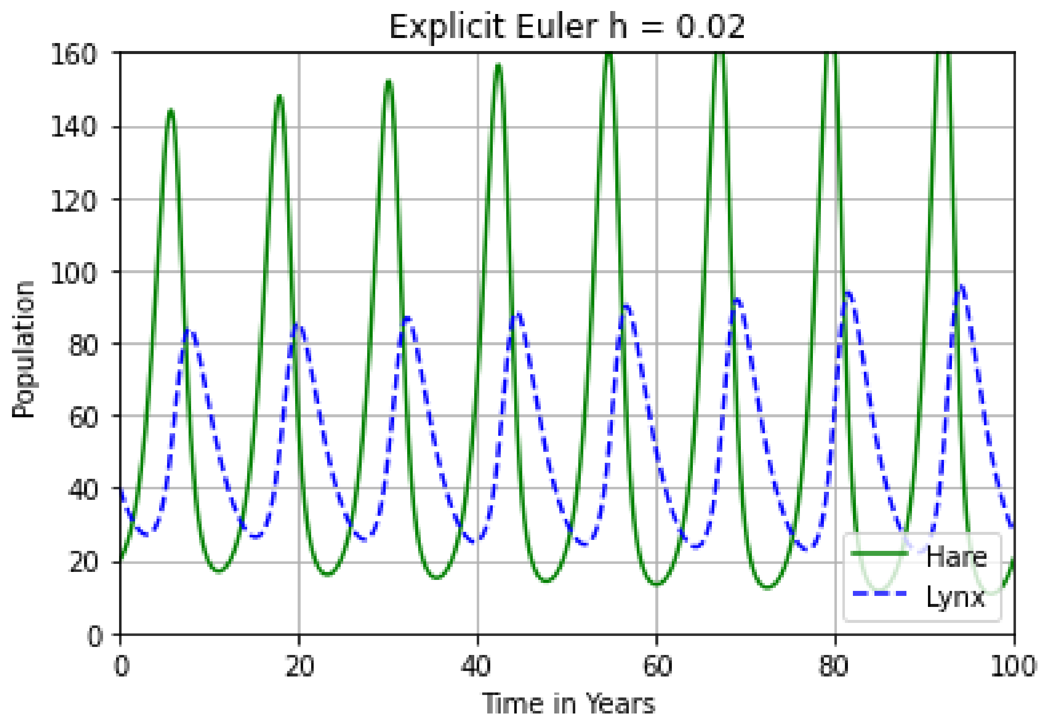
Problem 1. part a.

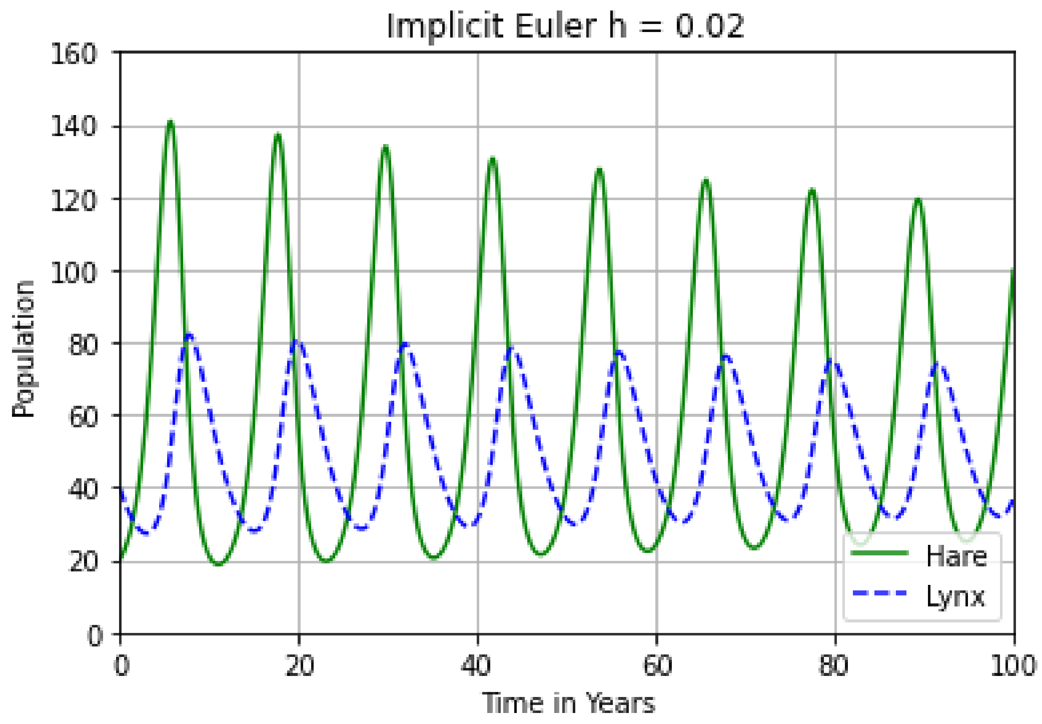
[2]:
```
def Trap(x0, x_end, y0, h, rhs_func, rhs_jac, p):
    '''
    Solve the initial value problem using trapezoid method

    Parameters
    ----------
    x0 : value
         initial x value
    x_end : value
            final x value
    y0 : np.array
         intial value of y
    h : value
        step size
    rhs_func : function
               function evaluating f(x, y) for given x and y
    p : np.array
        vector of parameter values for rhs_func

    Returns
    -------
    x : np.array
        vector of x values x(i) = x0 + (i - 1) * h
    y : np.array
        vector of approximate y values corresponding to x vector

    Michael Goforth
    CAAM 550
    Rice University
    November 19, 2021
    '''

    mx = math.ceil((x_end - x0) / h)
    n = y0.size
    x = np.array([x0 + i * h for i in range(mx+1)])
    y = np.zeros([n, mx+1])

    newt_tol = np.amin([.01 * h, .001])
    y[:, 0] = y0

    for i in range(mx):
        y[:, i+1] = y[:, i]
        newt_it = 0
        newt_fct = y[:, i+1] - h / 2 * (rhs_func(x[i+1], y[:, i+1], p)
                                         + rhs_func(x[i], y[:, i], p)) - y[:, i]
        while(np.linalg.norm(newt_fct) > newt_tol * np.linalg.norm(y[:, i+1])):
            newt_jac = np.eye(n) - h / 2 * (rhs_jac(x[i+1], y[:, i+1], p)
                                             + rhs_jac(x[i], y[:, i], p))
            newt_step = - (np.linalg.solve(newt_jac, newt_fct))
            y[:, i+1] = y[:, i+1] + newt_step
            newt_it = newt_it + 1
            newt_fct = y[:, i+1] - h * rhs_func(x[i+1], y[:, i+1], p) - y[:, i]
    return x, y
```

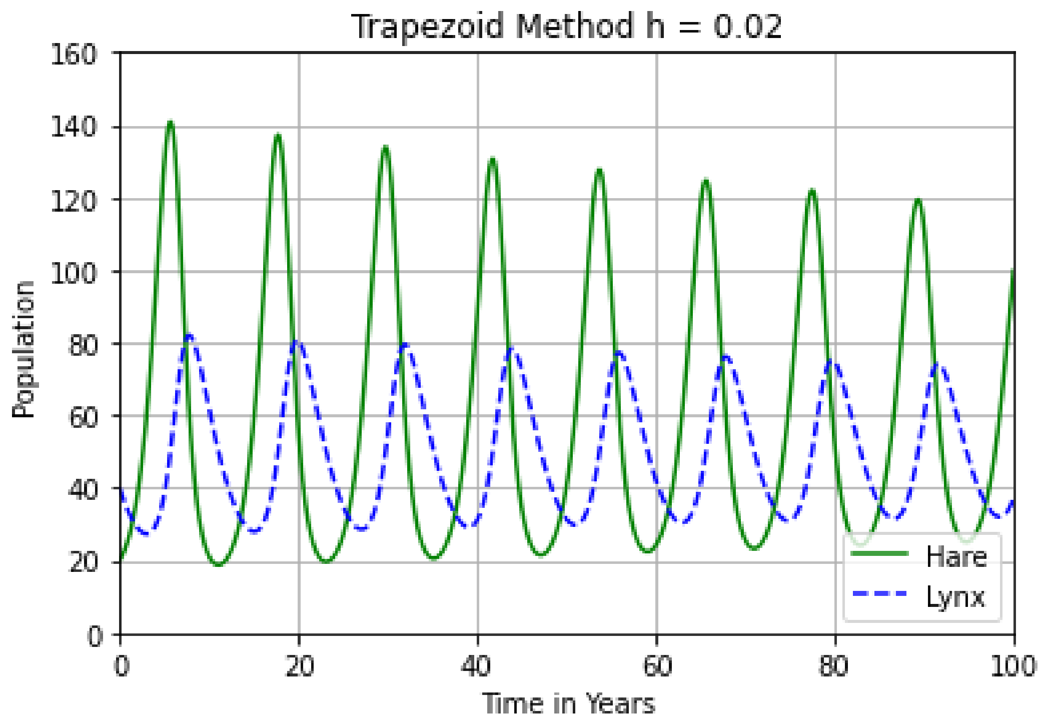
part b.

[3]:
```
# Code converted from demo_ode_euler.m originally written by M. Heinkenschloss.

def ode_predprey(t, y, p):
    y1 = p[0]*y[0] - p[1]*y[0]*y[1]
    y2 = -p[2] * y[1] + p[3] * y[0] * y[1]
    return np.array([y1, y2])



def ode_predpreyjac(t, y, p):
    return np.array([[p[0] - p[1] * y[1], -p[1] * y[0]],
                     [p[3] * y[1], -p[2] + p[3] * y[0]]])



def ExpEuler(x0, x_end, y0, h, rhs_func, p):
    '''
    Solve the initial value problem using explicit Euler method

    Parameters
    ----------
    x0 : value
         initial x value
    x_end : value
            final x value
    y0 : np.array
         intial value of y
    h : value
        step size
    rhs_func : function
               function evaluating f(x, y) for given x and y
    p : np.array
        vector of parameter values for rhs_func

    Returns
    -------
    x : np.array
        vector of x values x(i) = x0 + (i - 1) * h
    y : np.array
        vector of approximate y values corresponding to x vector

    Michael Goforth
    CAAM 550
    Rice University
    November 19, 2021
    '''

    mx = math.ceil((x_end - x0) / h)
    n = y0.size
    x = np.array([x0 + i * h for i in range(mx+1)])
    y = np.zeros([n, mx+1])

    y[:, 0] = y0

    for i in range(mx):
        y[:, i+1] = y[:, i] + h * rhs_func(x[i], y[:, i], p)

    return x, y


def ImpEuler(x0, x_end, y0, h, rhs_func, rhs_jac, p):
    '''
    Solve the initial value problem using implicit Euler method

    Parameters
    ----------
    x0 : value
         initial x value
    x_end : value
            final x value
    y0 : np.array
         intial value of y
    h : value
        step size
    rhs_func : function
               function evaluating f(x, y) for given x and y
    rhs_jac : function
              function evaluating jacobian w.r.t y of f(x, y) for given x and y
    p : np.array
        vector of parameter values for rhs_func

    Returns
    -------
    x : np.array
        vector of x values x(i) = x0 + (i - 1) * h
    y : np.array
        vector of approximate y values corresponding to x vector

    Michael Goforth
    CAAM 550
    Rice University
    November 19, 2021
    '''

    mx = math.ceil((x_end - x0) / h)
    n = y0.size
    x = np.array([x0 + i * h for i in range(mx+1)])
    y = np.zeros([n, mx+1])

    newt_tol = np.amin([.01 * h, .001])
    y[:, 0] = y0

    for i in range(mx):
        y[:, i+1] = y[:, i]
        newt_it = 0
        newt_fct = y[:, i+1] - h * rhs_func(x[i+1], y[:, i+1], p) - y[:, i]
        while(np.linalg.norm(newt_fct) > newt_tol * np.linalg.norm(y[:, i+1])):
            newt_jac = np.eye(n) - h * rhs_jac(x[i+1], y[:, i+1], p)
            newt_step = - (np.linalg.solve(newt_jac, newt_fct))
            y[:, i+1] = y[:, i+1] + newt_step
            newt_it = newt_it + 1
            newt_fct = y[:, i+1] - h * rhs_func(x[i+1], y[:, i+1], p) - y[:, i]
    return x, y


p = np.array([1, .02, .3, .005])
```

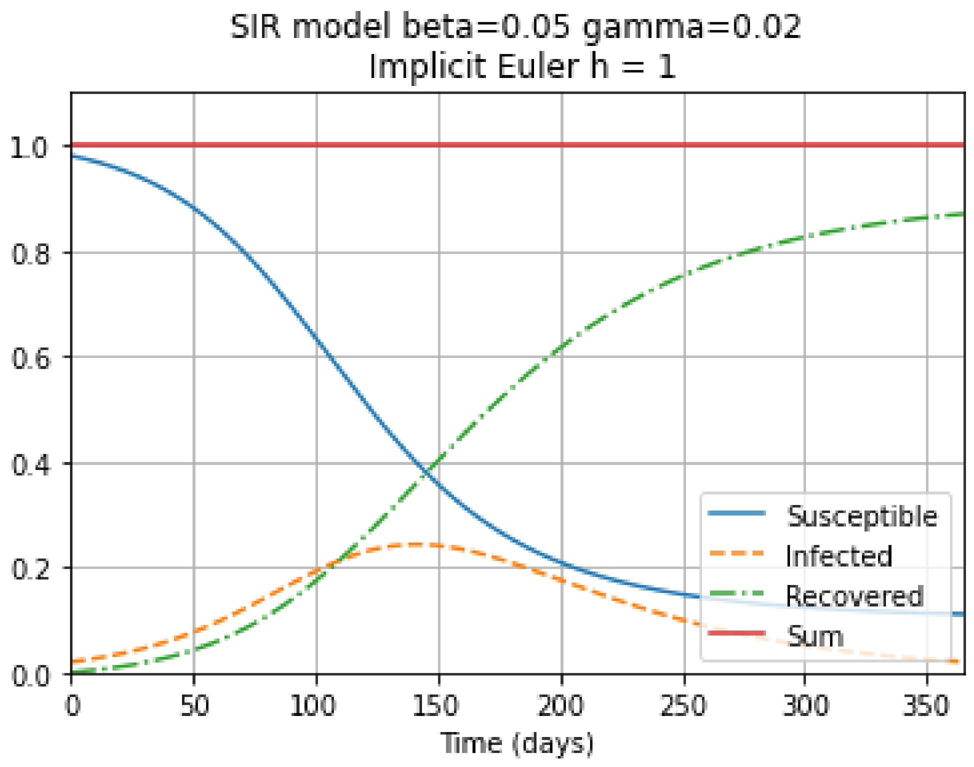
```python
t0 = 0
tfinal = 100
y0 = np.array([20,40])
h = .02

texp, yexp = ExpEuler(t0, tfinal, y0, h, ode_predprey, p)

plt.plot(texp, yexp[0, :], 'g-')
plt.plot(texp, yexp[1, :], 'b--')
plt.xlabel('Time in Years')
plt.ylabel('Population')
plt.title('Explicit Euler h = ' + str(h))
plt.legend(['Hare', 'Lynx'], loc='lower right')
plt.xlim(0, 100)
plt.ylim(0, 160)
plt.grid()
plt.show()

timp, yimp = ImpEuler(t0, tfinal, y0, h, ode_predprey, ode_predpreyjac, p)

plt.plot(timp, yimp[0, :], 'g-')
plt.plot(timp, yimp[1, :], 'b--')
plt.xlabel('Time in Years')
plt.ylabel('Population')
plt.title('Implicit Euler h = ' + str(h))
plt.legend(['Hare', 'Lynx'], loc='lower right')
plt.xlim(0, 100)
plt.ylim(0, 160)
plt.grid()
plt.show()

ttrap, ytrap = Trap(t0, tfinal, y0, h, ode_predprey, ode_predpreyjac, p)

plt.plot(ttrap, ytrap[0, :], 'g-')
plt.plot(ttrap, ytrap[1, :], 'b--')
plt.xlabel('Time in Years')
plt.ylabel('Population')
plt.title('Trapezoid Method h = ' + str(h))
plt.legend(['Hare', 'Lynx'], loc='lower right')
plt.xlim(0, 100)
plt.ylim(0, 160)
plt.grid()
plt.show()
```

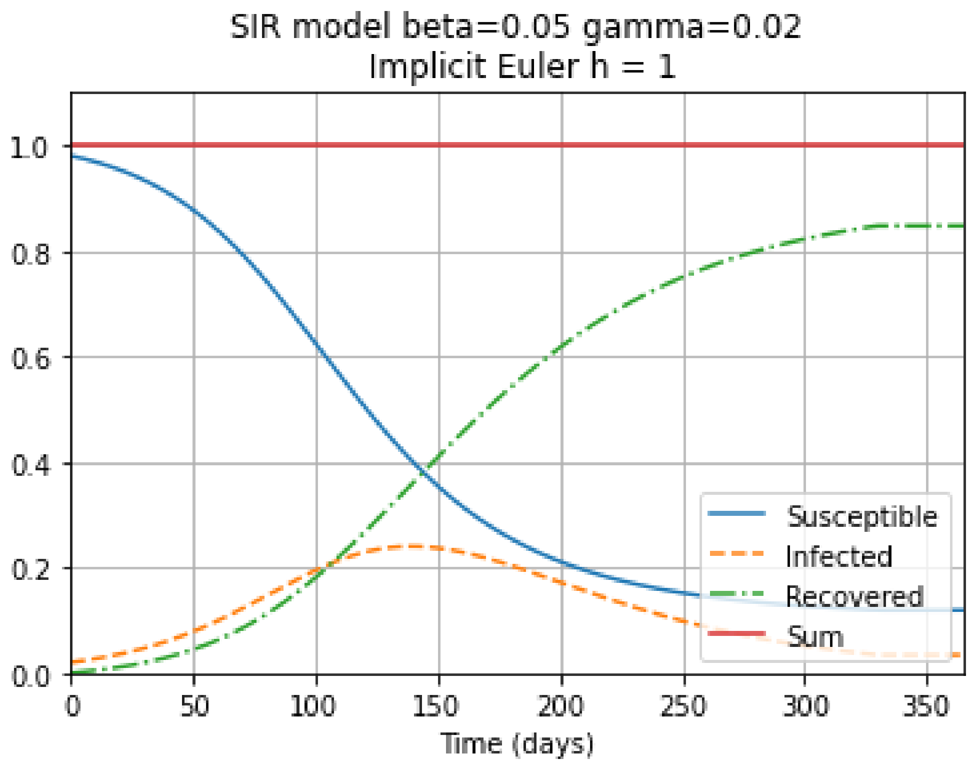

Explicit Euler h = 0.02
Population
Time in Years
Hare
Lynx

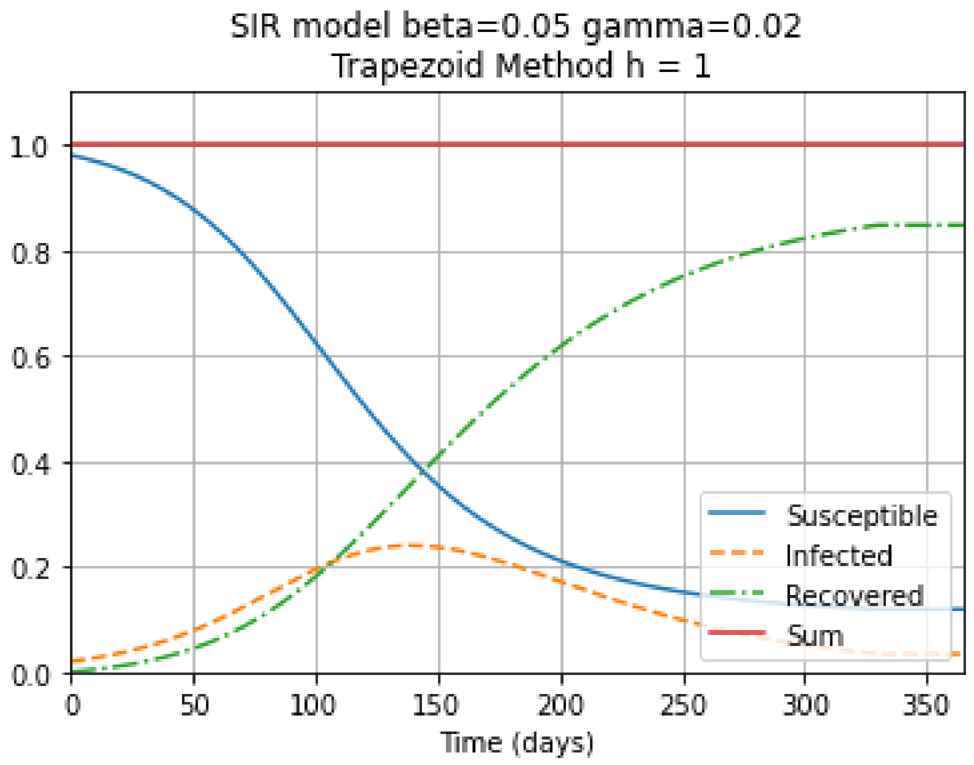

Implicit Euler h = 0.02
Population
Time in Years
Hare
Lynx



[4]:
```
def ode_SIR(t, y, p):
    y1 = -p[0] * y[0] * y[1]
    y2 = p[0] * y[0] * y[1] - p[1] * y[1]
    y3 = p[1] * y[1]
    return np.array([y1, y2, y3])

def ode_SIRjac(t, y, p):
    return np.array([[-p[0] * y[1], -p[0] * y[0], 0],
                    [p[0] * y[1], p[0] * y[0] - p[1], 0],
                    [0, p[1], 0]])

p = np.array([.05, .02])

t0 = 0
tfinal = 365
y0 = np.array([.98, .02, 0])
h = 1

texp, yexp = ExpEuler(t0, tfinal, y0, h, ode_SIR, p)

plt.plot(texp, yexp[0, :], '-')
plt.plot(texp, yexp[1, :], '--')
plt.plot(texp, yexp[2, :], '-.')
plt.plot(texp, sum(yexp, 0))
plt.xlabel('Time (days)')
plt.title('SIR model beta='+str(p[0]) + ' gamma=' + str(p[1]) + '\n Implicit␣
 ↪Euler h = ' + str(h))
plt.legend(['Susceptible', 'Infected', 'Recovered', 'Sum'], loc='lower right')
plt.xlim(0, 365)
plt.ylim(0, 1.1)
plt.grid()
plt.show()

timp, yimp = ImpEuler(t0, tfinal, y0, h, ode_SIR, ode_SIRjac, p)

plt.plot(timp, yimp[0, :], '-')
plt.plot(timp, yimp[1, :], '--')
plt.plot(timp, yimp[2, :], '-.')
plt.plot(timp, sum(yimp, 0))
plt.xlabel('Time (days)')
plt.title('SIR model beta='+str(p[0]) + ' gamma=' + str(p[1]) + '\n Implicit␣
 ↪Euler h = ' + str(h))
plt.legend(['Susceptible', 'Infected', 'Recovered', 'Sum'], loc='lower right')
plt.xlim(0, 365)
plt.ylim(0, 1.1)
plt.grid()
plt.show()

ttrap, ytrap = Trap(t0, tfinal, y0, h, ode_SIR, ode_SIRjac, p)
plt.plot(ttrap, ytrap[0, :], '-')
plt.plot(ttrap, ytrap[1, :], '--')
plt.plot(ttrap, ytrap[2, :], '-.')
plt.plot(ttrap, sum(ytrap, 0))
plt.xlabel('Time (days)')
plt.title('SIR model beta='+str(p[0]) + ' gamma=' + str(p[1]) + '\n Trapezoid␣
 ↪Method h = ' + str(h))
plt.legend(['Susceptible', 'Infected', 'Recovered', 'Sum'], loc='lower right')
plt.xlim(0, 365)
plt.ylim(0, 1.1)
plt.grid()
plt.show()
```


SIR model beta=0.05 gamma=0.02
Implicit Euler h = 1
1.0
0.8
0.6
0.4
0.2
0.0
0
50
100
150
200
250
300
350
Time (days)
Susceptible
Infected
Recovered
Sum

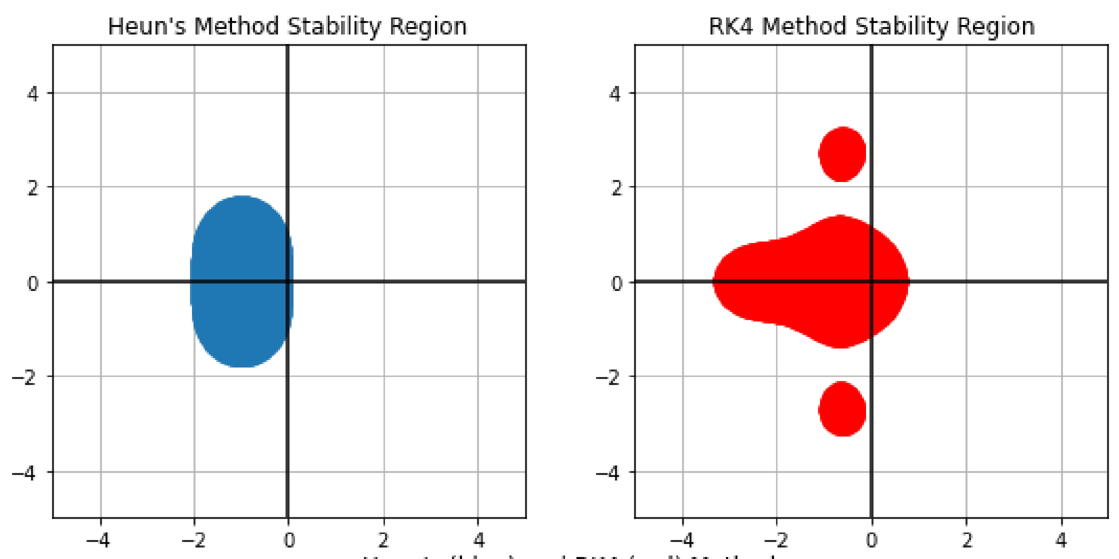

SIR model beta=0.05 gamma=0.02
Implicit Euler h = 1
1.0
0.8
0.6
0.4
0.2
0.0
0
50
100
150
200
250
300
350
Time (days)
Susceptible
Infected
Recovered
Sum

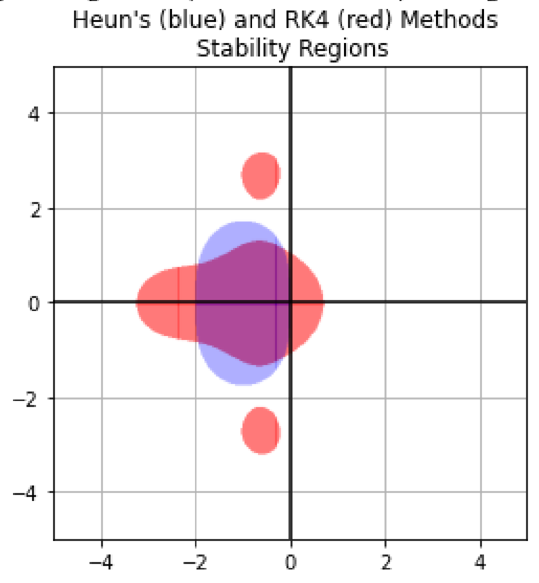


Problem 2. part d.

```
[5]: axlim = 5 * np.array([-1, 1, -1, 1])
     n = 500
     z = (np.outer(np.linspace(axlim[0], axlim[1], n), np.ones([n]))
         + np.outer(1j * np.ones([n]), np.linspace(axlim[2], axlim[3], n)))
     heunstabregion = abs(np.polyval([.5, 1, 1], z))
     heunind = heunstabregion<1
     fig = plt.figure(figsize=(10,10))
     ax1 = fig.add_subplot(221)
     ax1.plot(z[heunind].real, z[heunind].imag, '.')
     ax1.title.set_text("Heun's Method Stability Region")
     ax1.set_xlim([axlim[0], axlim[1]])
     ax1.set_ylim([axlim[2], axlim[3]])
     ax1.axhline(y=0, color='k')
     ax1.axvline(x=0, color='k')
     ax1.grid(True, which='both')
     ax1.set_aspect('equal')
     rkstabregion = abs(np.polyval([1/24, 1/6, .5, 1, 0], z))
     rkind = rkstabregion<1
     ax2 = fig.add_subplot(222)
```

```python
ax2.plot(z[rkind].real, z[rkind].imag, 'r.')
ax2.title.set_text("RK4 Method Stability Region")
ax2.set_xlim([axlim[0], axlim[1]])
ax2.set_ylim([axlim[2], axlim[3]])
ax2.axhline(y=0, color='k')
ax2.axvline(x=0, color='k')
ax2.grid(True, which='both')
ax2.set_aspect('equal')
ax3 = fig.add_subplot(212)
ax3.plot(z[rkind].real, z[rkind].imag, 'r.', ms=.1)
ax3.plot(z[heunind].real, z[heunind].imag, 'b.', ms=.05)
ax3.set_xlim([axlim[0], axlim[1]])
ax3.set_ylim([axlim[2], axlim[3]])
ax3.axhline(y=0, color='k')
ax3.axvline(x=0, color='k')
ax3.grid(True, which='both')
ax3.set_aspect('equal')
ax3.title.set_text("Heun's (blue) and RK4 (red) Methods \n Stability Regions")
```



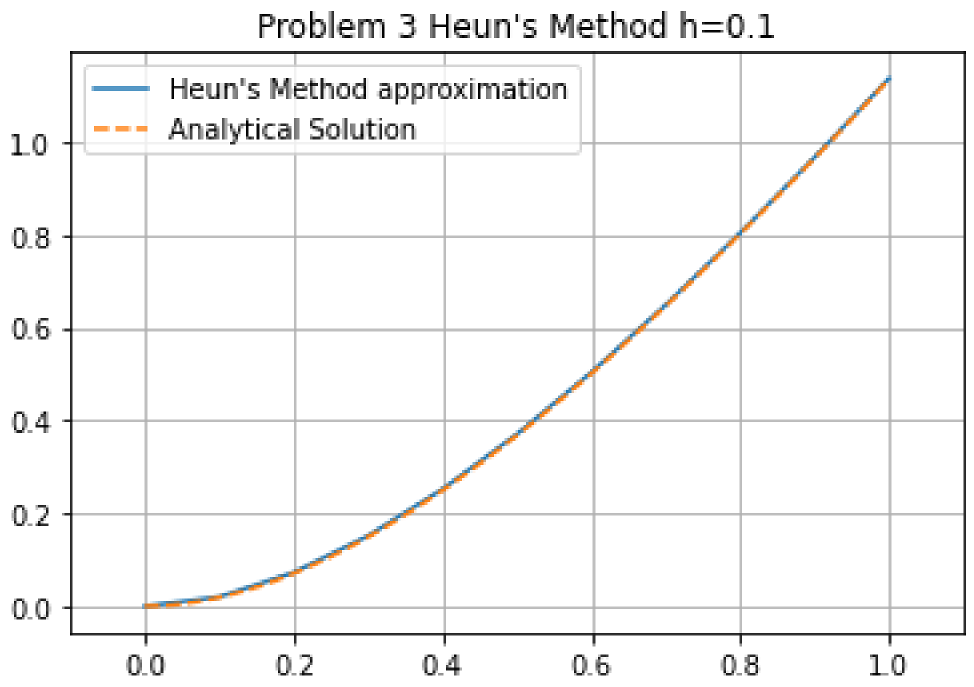


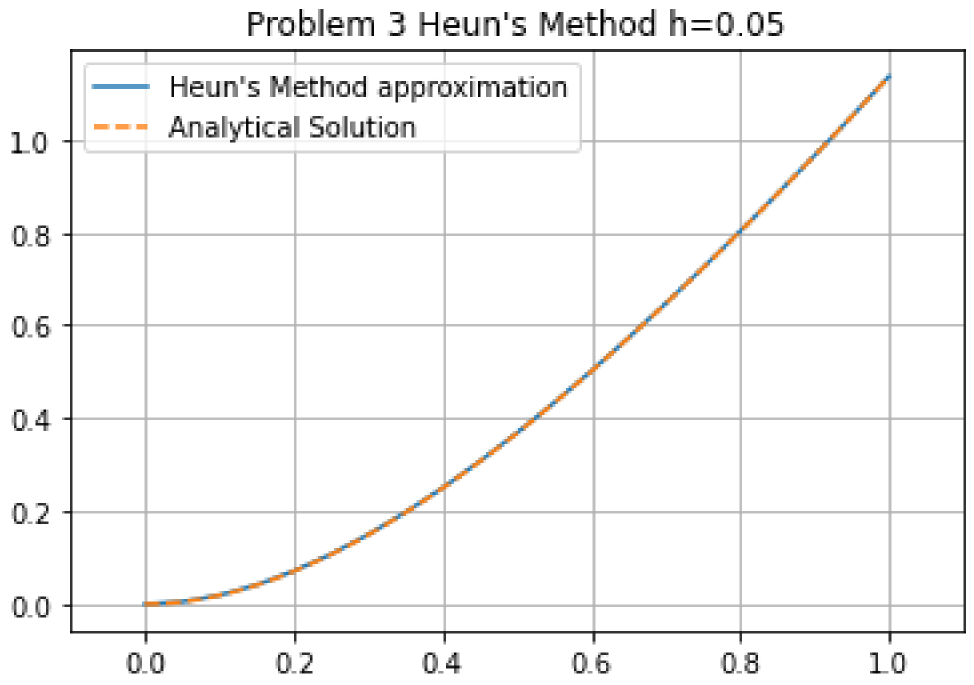
Problem 3 part b.

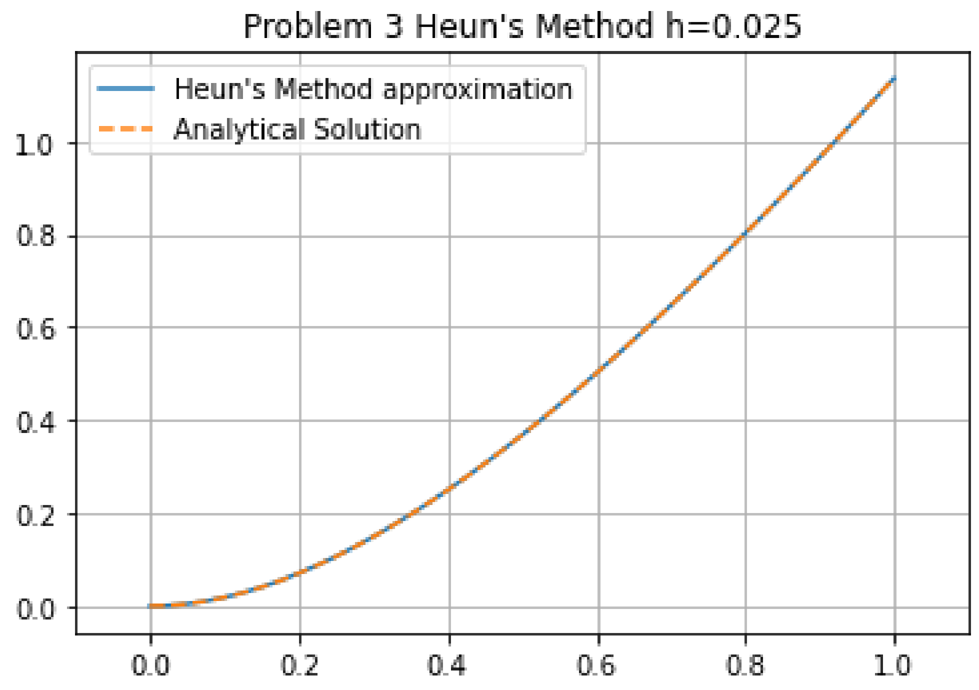
```
[6]: def Heun(x0, x_end, y0, h, rhs_func):
         '''
         Solve the initial value problem using Heun's method

         Parameters
         ----------
         x0 : value
              initial x value
         x_end : value
                 final x value
         y0 : np.array
```

```
        intial value of y
    h : value
        step size
    rhs_func : function
               function evaluating f(x, y) for given x and y

    Returns
    -------
    x : np.array
        vector of x values x(i) = x0 + (i - 1) * h
    y : np.array
        vector of approximate y values corresponding to x vector

    Michael Goforth
    CAAM 550
    Rice University
    November 19, 2021
    '''

    mx = math.ceil((x_end - x0) / h)
    x = np.array([x0 + i * h for i in range(mx+1)])
    y = np.zeros([mx+1])

    y[0] = y0

    for i in range(mx):
        y[i+1] = y[i] + h / 2 * (rhs_func(x[i], y[i])
                                  + rhs_func(x[i+1], y[i] + h *␣
↪rhs_func(x[i], y[i])))
    return x, y

def prob3fun(x, y):
    return 4 * x - 2 * y

def prob3soln(x):
    return 2 * x - 1 + np.exp(-2 * x)

x0 = 0
xfinal = 1
y0 = np.array([0])

xsoln = np.linspace(x0, xfinal, 500)
ysoln = prob3soln(xsoln)

h = .1

xheun1, yheun1 = Heun(x0, xfinal, y0, h, prob3fun)
```

```
plt.plot(xheun1, yheun1, '-')
plt.plot(xsoln, ysoln, '--')
plt.title("Problem 3 Heun's Method h=" + str(h))
plt.legend(["Heun's Method approximation", "Analytical Solution"])
plt.xlim(-.1, 1.1)
plt.grid()
plt.show()

h = .05

xheun2, yheun2 = Heun(x0, xfinal, y0, h, prob3fun)

plt.plot(xheun2, yheun2, '-')
plt.plot(xsoln, ysoln, '--')
plt.title("Problem 3 Heun's Method h=" + str(h))
plt.legend(["Heun's Method approximation", "Analytical Solution"])
plt.xlim(-.1, 1.1)
plt.grid()
plt.show()

h = .025

xheun3, yheun3 = Heun(x0, xfinal, y0, h, prob3fun)

plt.plot(xheun3, yheun3, '-')
plt.plot(xsoln, ysoln, '--')
plt.title("Problem 3 Heun's Method h=" + str(h))
plt.legend(["Heun's Method approximation", "Analytical Solution"])
plt.xlim(-.1, 1.1)
plt.grid()
plt.show()
```

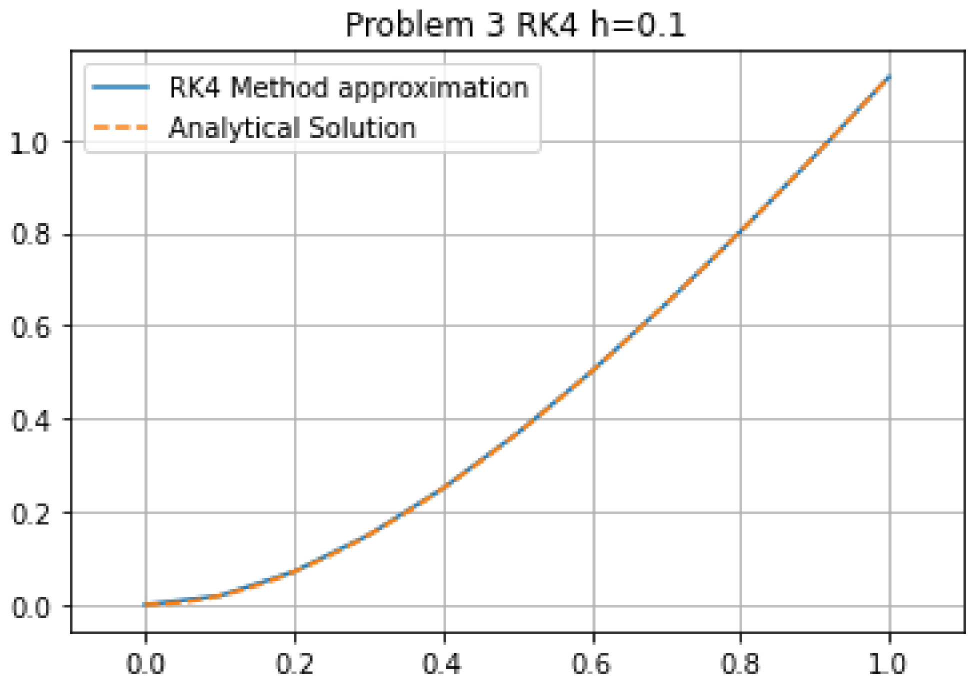

Problem 3 Heun's Method h=0.1
Heun's Method approximation
Analytical Solution
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

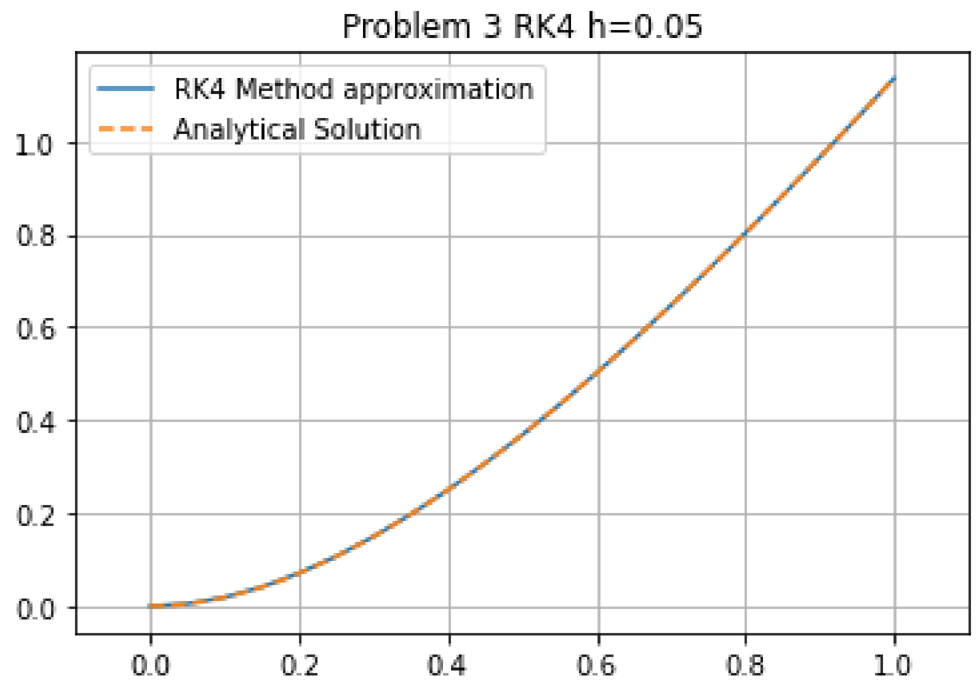

Problem 3 Heun's Method h=0.05
Heun's Method approximation
Analytical Solution
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

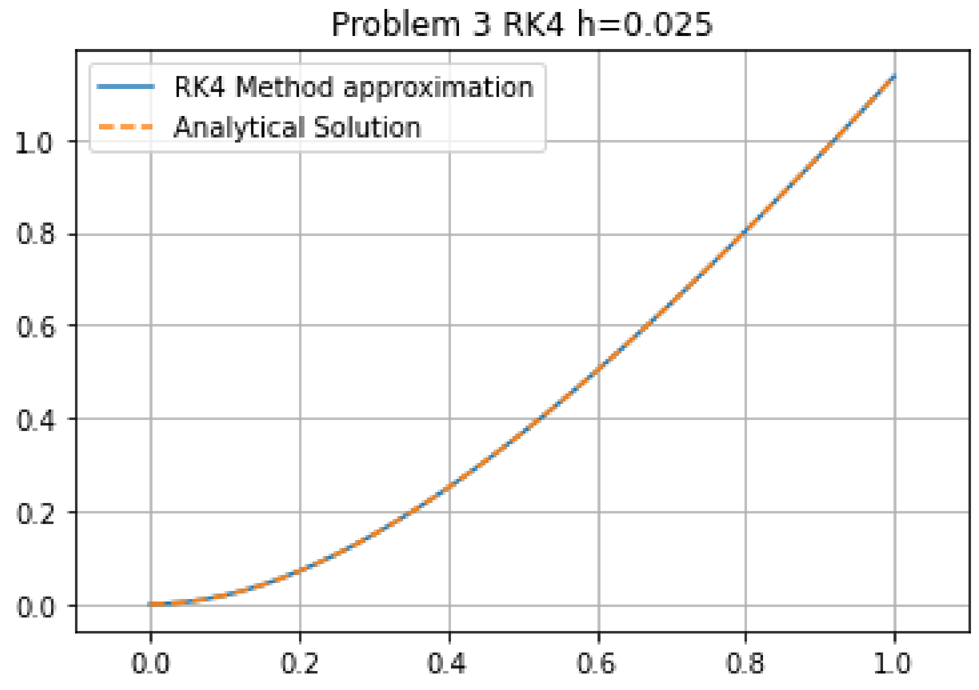


part c.

[7]:
```
def rk4(x0, x_end, y0, h, rhs_func):
    '''
    Solve the initial value problem using Heun's method

    Parameters
    ----------
    x0 : value
         initial x value
    x_end : value
            final x value
    y0 : np.array
         intial value of y
    h : value
        step size
    rhs_func : function
               function evaluating f(x, y) for given x and y

    Returns
    -------
    x : np.array
        vector of x values x(i) = x0 + (i - 1) * h
    y : np.array
        vector of approximate y values corresponding to x vector
    Michael Goforth
    CAAM 550
    Rice University
    November 19, 2021
    '''

    mx = math.ceil((x_end - x0) / h)
    x = np.array([x0 + i * h for i in range(mx+1)])
    y = np.zeros([mx+1])

    y[0] = y0

    for i in range(mx):
        xi = x[i]
        yi = y[i]
        Y1 = yi
        Y2 = yi + h / 2 * rhs_func(xi, Y1)
        Y3 = yi + h / 2 * rhs_func(xi + h / 2, Y2)
        Y4 = yi + h * rhs_func(xi + h / 2, Y3)
        y[i+1] = yi + h/6 * (rhs_func(xi, Y1) + 2 * rhs_func(xi + h / 2, Y2)
                            + 2 * rhs_func(xi + h / 2, Y3) + rhs_func(x[i+1],␣
↪Y4))
    return x, y


h = .1

xrk1, yrk1 = rk4(x0, xfinal, y0, h, prob3fun)

plt.plot(xrk1, yrk1, '-')
plt.plot(xsoln, ysoln, '--')
plt.title("Problem 3 RK4 h=" + str(h))
plt.legend(["RK4 Method approximation", "Analytical Solution"])
plt.xlim(-.1, 1.1)
plt.grid()
plt.show()

h = .05

xrk2, yrk2 = rk4(x0, xfinal, y0, h, prob3fun)

plt.plot(xrk2, yrk2, '-')
plt.plot(xsoln, ysoln, '--')
plt.title("Problem 3 RK4 h=" + str(h))
plt.legend(["RK4 Method approximation", "Analytical Solution"])
plt.xlim(-.1, 1.1)
plt.grid()
plt.show()

h = .025

xrk3, yrk3 = rk4(x0, xfinal, y0, h, prob3fun)

plt.plot(xrk3, yrk3, '-')
plt.plot(xsoln, ysoln, '--')
plt.title("Problem 3 RK4 h=" + str(h))
plt.legend(["RK4 Method approximation", "Analytical Solution"])
plt.xlim(-.1, 1.1)
plt.grid()
plt.show()
```

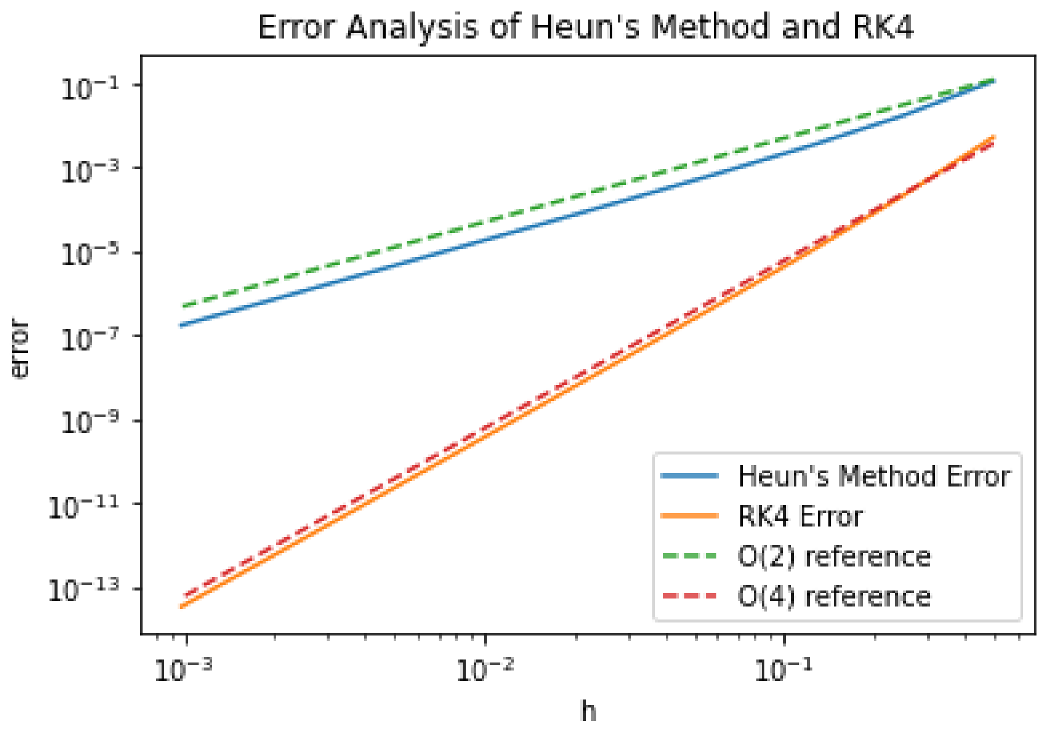



Problem 3 RK4 h=0.05
RK4 Method approximation
Analytical Solution
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


Problem 3 RK4 h=0.025
RK4 Method approximation
Analytical Solution
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

part d.

[9]:
```
hvec = [2**-j for j in range(1, 11)]
heunerr = [0 for h in hvec]
rk4err = [0 for h in hvec]
ytrue = prob3soln(1)
for i in range(10):
    h = hvec[i]
    xheun, yheun = Heun(x0, xfinal, y0, h, prob3fun)
    heunerr[i] = abs(ytrue - yheun[-1])
    xrk4, yrk4 = rk4(x0, xfinal, y0, h, prob3fun)
    rk4err[i] = abs(ytrue - yrk4[-1])

plt.loglog(hvec, heunerr)
plt.loglog(hvec, rk4err)
o2ref = [.5 * h**2 for h in hvec]
o4ref = [.06 * h**4 for h in hvec]
plt.loglog(hvec, o2ref, '--')
plt.loglog(hvec, o4ref, '--')
plt.legend(["Heun's Method Error", 'RK4 Error', 'O(2) reference', 'O(4)␣
 ↪reference'])
plt.title("Error Analysis of Heun's Method and RK4")
plt.xlabel('h')
plt.ylabel('error')
plt.show()
```



[ ]:
```

```