```python
import matplotlib.pyplot as plt
import numpy as np
def feval(funcName, *args):
   return eval(funcName)(*args)

def forwardEuler(func, yinit, x_range, h):

   m = len(yinit)
   n = int((x_range[-1] - x_range[0])/h)

   x = x_range[0]
   y = yinit

   xsol = np.empty(0)
   xsol = np.append(xsol, x)

   ysol = np.empty(0)
   ysol = np.append(ysol, y)

   for i in range(n):
       yprime = feval(func, x, y)

       for j in range(m):
           y[j] = y[j] + h*yprime[j]

       x += h
       xsol = np.append(xsol, x)

       for r in range(len(y)):
           ysol = np.append(ysol, y[r])

   return [xsol, ysol]

def myFunc(x, y):

   dy = np.zeros((len(y)))
   dy[0] = 3*(1+x) - y[0]
   return dy
```

```python
h = 0.2
x = np.array([1.0, 2.0])
yinit = np.array([4.0])

[ts, ys] = forwardEuler('myFunc', yinit, x, h)

dt = int((x[-1] - x[0]) / h)
t = [x[0]+i*h for i in range(dt+1)]
yexact = []
for i in range(dt+1):
   ye = 3*t[i] + np.exp(1-t[i])
   yexact.append(ye)

plt.plot(ts, ys, 'r')
plt.plot(t, yexact, 'b')
plt.xlim(x[0], x[1])
plt.legend(["Forward Euler method",
            "Exact solution"], loc=2)
plt.xlabel('x', fontsize=17)
plt.ylabel('y', fontsize=17)
plt.tight_layout()
plt.show()
```

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

def feval(funcName, *args):
   return eval(funcName)(*args)


def backwardEuler(func, yinit, x_range, h):
   m = len(yinit)
   n = int((x_range[-1] - x_range[0])/h)

   x = x_range[0]
   y = yinit

   xsol = np.empty(0)
   xsol = np.append(xsol, x)

   ysol = np.empty(0)
   ysol = np.append(ysol, y)

   for i in range(n):
       yprime = feval(func, x+h, y)/(1+h)

       for j in range(m):
           y[j] = y[j] + h*yprime[j]

       x += h
       xsol = np.append(xsol, x)

       for r in range(len(y)):
           ysol = np.append(ysol, y[r])

   return [xsol, ysol]


def myFunc(x, y):

   dy = np.zeros((len(y)))
   dy[0] = 3*(1+x) - y[0]
   return dy
```

```
h = 0.2
x = np.array([1.0, 2.0])
yinit = np.array([4.0])


[ts, ys] = backwardEuler('myFunc', yinit, x, h)


# Calculates the exact solution, for comparison
dt = int((x[-1] - x[0]) / h)
t = [x[0]+i*h for i in range(dt+1)]
yexact = []
for i in range(dt+1):
   ye = 3 * t[i] + np.exp(1 - t[i])
   yexact.append(ye)


plt.plot(ts, ys, 'r')
plt.plot(t, yexact, 'b')
plt.xlim(x[0], x[1])
plt.legend(["Backward Euler method",
           "Exact solution"], loc=2)
plt.xlabel('x', fontsize=17)
plt.ylabel('y', fontsize=17)
plt.tight_layout()
plt.show()
```

```python
import matplotlib.pyplot as plt
import math


def feval(funcName, *args):
   return eval(funcName)(*args)


def mult(vector, scalar):
   newvector = [0]*len(vector)
   for i in range(len(vector)):
       newvector[i] = vector[i]*scalar
   return newvector


def midpoint(func, yinit, x_range, h):
   numOfODEs = len(yinit)
   sub_intervals = int((x_range[-1] - x_range[0])/h)

   x = x_range[0]
   y = yinit

   xsol = [x]
   ysol = [y[0]]

   for i in range(sub_intervals):
       y0prime = feval(func, x, y)

       k1 = mult(y0prime, h/2)

       ypredictor = [u + v for u, v in zip(y, k1)]

       y1prime = feval(func, x+h/2, ypredictor)

       for j in range(numOfODEs):
           y[j] = y[j] + h*y1prime[j]

       x = x + h
       xsol.append(x)
```

```
        for r in range(len(y)):
            ysol.append(y[r])

    return [xsol, ysol]

def myFunc(x, y):
    dy = [0] * len(y)
    dy[0] = 3 * (1 + x) - y[0]
    return dy

h = 0.2
x = [1.0, 2.0]
yinit = [4.0]

[ts, ys] = midpoint('myFunc', yinit, x, h)

# Calculates the exact solution, for comparison
dt = int((x[-1] - x[0]) / h)
t = [x[0]+i*h for i in range(dt+1)]
yexact = []
for i in range(dt+1):
    ye = 3 * t[i] + math.exp(1 - t[i])
    yexact.append(ye)


plt.plot(ts, ys, 'r')
plt.plot(t, yexact, 'b')
plt.xlim(x[0], x[1])
plt.legend(["Midpoint method", "Exact solution"], loc=2)
plt.xlabel('x', fontsize=17)
plt.ylabel('y', fontsize=17)
plt.tight_layout()
plt.show()
```

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


def feval(funcName, *args):
   return eval(funcName)(*args)


def HeunsMethod(func, yinit, x_range, h):
   m = len(yinit)
   n = int((x_range[-1] - x_range[0])/h)

   x = x_range[0]
   y = yinit

   xsol = np.empty(0)
   xsol = np.append(xsol, x)

   ysol = np.empty(0)
   ysol = np.append(ysol, y)

   for i in range(n):
       y0prime = feval(func, x, y)

       k1 = y0prime * h

       ypredictor = y + k1

       y1prime = feval(func, x+h, ypredictor)

       for j in range(m):
           y[j] = y[j] + (h/2)*y0prime[j] + (h/2)*y1prime[j]

       x = x + h
       xsol = np.append(xsol, x)

       for r in range(len(y)):
           ysol = np.append(ysol, y[r])
```

```python
    return [xsol, ysol]


def myFunc(x, y):
    dy = np.zeros((len(y)))
    dy[0] = 3 * (1 + x) - y[0]
    return dy

h = 0.2
x = np.array([1, 2])
yinit = np.array([4.0])


[ts, ys] = HeunsMethod('myFunc', yinit, x, h)


dt = int((x[-1] - x[0]) / h)
t = [x[0]+i*h for i in range(dt+1)]
yexact = []
for i in range(dt+1):
    ye = 3 * t[i] + np.exp(1 - t[i])
    yexact.append(ye)


plt.plot(ts, ys, 'r')
plt.plot(t, yexact, 'b')
plt.xlim(x[0], x[1])
plt.legend(["Heun's method", "Exact solution"], loc=2)
plt.xlabel('x', fontsize=17)
plt.ylabel('y', fontsize=17)
plt.tight_layout()
plt.show()
```

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


def feval(funcName, *args):
   return eval(funcName)(*args)


def RK3rdOrder(func, yinit, x_range, h):
   m = len(yinit)
   n = int((x_range[-1] - x_range[0])/h)

   x = x_range[0]
   y = yinit

   xsol = np.empty(0)
   xsol = np.append(xsol, x)

   ysol = np.empty(0)
   ysol = np.append(ysol, y)

   for i in range(n):
       k1 = feval(func, x, y)

       yp1 = y + k1 * (h/2)

       k2 = feval(func, x+h/2, yp1)

       yp2 = y - (k1 * h) + (k2 * 2*h)

       k3 = feval(func, x+h, yp2)

       for j in range(m):
           y[j] = y[j] + (h/6)*(k1[j] + 4*k2[j] + k3[j])

       x = x + h
       xsol = np.append(xsol, x)

       for r in range(len(y)):
```

```python
            ysol = np.append(ysol, y[r])

    return [xsol, ysol]


def myFunc(x, y):
    dy = np.zeros((len(y)))
    dy[0] = np.exp(-2 * x) - 2 * y[0]
    return dy



h = 0.2
x = np.array([0, 2])
yinit = np.array([1.0/10])


[ts, ys] = RK3rdOrder('myFunc', yinit, x, h)


dt = int((x[-1]-x[0])/h)
t = [x[0]+i*h for i in range(dt+1)]
yexact = []
for i in range(dt+1):
    ye = (1.0/10)*np.exp(-2*t[i]) + t[i]*np.exp(-2*t[i])
    yexact.append(ye)


plt.plot(ts, ys, 'r')
plt.plot(t, yexact, 'b')
plt.xlim(x[0], x[1])
plt.legend(["3rd Order RK", "Exact solution"], loc=1)
plt.xlabel('x', fontsize=17)
plt.ylabel('y', fontsize=17)
plt.tight_layout()
plt.show()
```

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


def feval(funcName, *args):
   return eval(funcName)(*args)


def RK4thOrder(func, yinit, x_range, h):
   m = len(yinit)
   n = int((x_range[-1] - x_range[0])/h)

   x = x_range[0]
   y = yinit

   xsol = np.empty(0)
   xsol = np.append(xsol, x)

   ysol = np.empty(0)
   ysol = np.append(ysol, y)

   for i in range(n):
       k1 = feval(func, x, y)

       yp2 = y + k1*(h/2)

       k2 = feval(func, x+h/2, yp2)

       yp3 = y + k2*(h/2)

       k3 = feval(func, x+h/2, yp3)

       yp4 = y + k3*h

       k4 = feval(func, x+h, yp4)

       for j in range(m):
           y[j] = y[j] + (h/6)*(k1[j] + 2*k2[j] + 2*k3[j] + k4[j])
```

```python
        x = x + h
        xsol = np.append(xsol, x)

        for r in range(len(y)):
            ysol = np.append(ysol, y[r])

    return [xsol, ysol]


def myFunc(x, y):
    dy = np.zeros((len(y)))
    dy[0] = np.exp(-2*x) - 2*y[0]
    return dy

h = 0.2
x = np.array([0.0, 2.0])
yinit = np.array([1.0/10])


[ts, ys] = RK4thOrder('myFunc', yinit, x, h)


dt = int((x[-1]-x[0])/h)
t = [x[0]+i*h for i in range(dt+1)]
yexact = []
for i in range(dt+1):
    ye = (1.0/10)*np.exp(-2*t[i]) + t[i]*np.exp(-2*t[i])
    yexact.append(ye)


diff = ys - yexact
print("Maximum difference =", np.max(abs(diff)))

plt.plot(ts, ys, 'r')
plt.plot(t, yexact, 'b')
plt.xlim(x[0], x[1])
plt.legend(["4th Order RK", "Exact solution"], loc=1)
plt.xlabel('x', fontsize=17)
plt.ylabel('y', fontsize=17)
plt.tight_layout()
```

```
plt.show()
```

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


def feval(funcName, *args):
    return eval(funcName)(*args)


def RK4thOrder(func, yinit, x_range, h):
    m = len(yinit)
    n = int((x_range[-1] - x_range[0])/h)

    x = x_range[0]
    y = yinit

    # Containers for solutions
    xsol = np.empty(0)
    xsol = np.append(xsol, x)

    ysol = np.empty(0)
    ysol = np.append(ysol, y)

    for i in range(n):
        k1 = feval(func, x, y)

        yp2 = y + k1*(h/2)

        k2 = feval(func, x+h/2, yp2)

        yp3 = y + k2*(h/2)

        k3 = feval(func, x+h/2, yp3)

        yp4 = y + k3*h

        k4 = feval(func, x+h, yp4)

        for j in range(m):
            y[j] = y[j] + (h/6)*(k1[j] + 2*k2[j] + 2*k3[j] + k4[j])

        x = x + h
        xsol = np.append(xsol, x)
```

```
        for r in range(len(y)):
            ysol = np.append(ysol, y[r])

    return [xsol, ysol]


def myFunc(x, y):
    # Van der Pol oscillator
    a = 1.0
    dy = np.zeros((len(y)))
    dy[0] = y[1]
    dy[1] = a*(1 - y[0]**2)*y[1] - y[0]
    return dy

h = 0.01
x = np.array([0.0, 30.0])
yinit = np.array([2.0, 0.0])


[ts, ys] = RK4thOrder('myFunc', yinit, x, h)

node = len(yinit)
ys1 = ys[0::node]
ys2 = ys[1::node]



plt.plot(ts, ys1, 'r')
plt.plot(ts, ys2, 'b')
plt.xlim(x[0], x[1])
plt.legend(["y(1)", "y(2)"], loc=2)
plt.xlabel('x', fontsize=17)
plt.ylabel('y', fontsize=17)
plt.tight_layout()
plt.show()
```

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


def feval(funcName, *args):
    return eval(funcName)(*args)


def HeunsMethod(func, yinit, x_range, h):
    m = len(yinit)
    n = int((x_range[-1] - x_range[0])/h)

    x = x_range[0]
    y = yinit

    # Containers for solutions
    xsol = np.empty(0)
    xsol = np.append(xsol, x)

    ysol = np.empty(0)
    ysol = np.append(ysol, y)

    for i in range(n):
        y0prime = feval(func, x, y)

        k1 = y0prime * h

        ypredictor = y + k1

        y1prime = feval(func, x+h, ypredictor)

        for j in range(m):
            y[j] = y[j] + (h/2)*y0prime[j] + (h/2)*y1prime[j]

        x = x + h
        xsol = np.append(xsol, x)

        for r in range(len(y)):
            ysol = np.append(ysol, y[r])

    return [xsol, ysol]
```

```
def myFunc(x, y):
    '''
    Example from Computational Cell Biology, Fall et al
    Exercise #1 of Chapter 9, page 256
    '''
    dy = np.zeros((len(y)))
    a = 1; b = 5; c = 4; r = 1; y0 = 0; epsi = 0.1
    dy[0] = ((a + b*y[0]**2)/(1 + y[0]**2 + r*y[1])) - y[0]
    dy[1] = epsi*(c*y[0] + y0 - y[1])
    return dy

h = 0.1
x = np.array([1, 100])
yinit = np.array([1.0, 1.0])


[ts, ys] = HeunsMethod('myFunc', yinit, x, h)

ys1 = ys[0::2]
ys2 = ys[1::2]


plt.plot(ts, ys1, 'r')
plt.plot(ts, ys2, 'b')
plt.xlim(x[0], x[1])
plt.legend(["x", "y"], loc=1)
plt.xlabel('t', fontsize=17)
plt.ylabel('Solutions', fontsize=17)
plt.title("Activator-Inhibitor System")
plt.tight_layout()
plt.show()
```