# Numerical Techniques Laboratory

## Assignment 7.2 | Tanishq Jasoria | 16MA20047

In [1]:

```
import math
from collections import namedtuple
from fractions import Fraction
from copy import copy
import numpy as np
from numpy.linalg import inv
from sympy import symbols
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
#some boilerplate code
x = symbols('x')
#Equation = namedtuple('Equation',('a','b'))
Conditions = namedtuple('Conditions',('x0','xn','y0','yn','h','e'))
```

In [3]:

```

```

$$\begin{aligned} y'' - (y')^2 - y^2 + y + 1 = 0 \\ x \in (0, \pi) \\ y(0) = 0.5 \\ y(\pi) = -0.5 \end{aligned}$$

In [4]:

```
def initializer(x):
    return (0.5 - x/np.pi)
```

In [5]:

```
def Finite_Diff_Solver(cond, y_k, verbose=True):
    n = int(np.ceil((cond.xn - cond.x0)/cond.h))
    a, b, c, d = np.zeros((n-2)), np.zeros((n-1)), np.zeros((n-2)), np.zeros((n-1))
    #A
    b[0] = -2.0/cond.h**2 + (-2.0 * y_k[1] + 1.0)
    c[0] = 1.0/cond.h**2 - 1.0/(cond.h*2.0) * (y_k[2] - y_k[0])/(cond.h)

    a[n-3] = 1.0/cond.h**2 + 1.0/(cond.h*2.0) * (y_k[-1] - y_k[-3])/(cond.h)
    b[n-2] = -2.0/cond.h**2 + (-2.0 * y_k[-2] + 1.0)
    #d
    d[0] = -1.0 - y_k[1]**2 - ((y_k[2] - y_k[0])/(2.0*cond.h))**2 -\
    (1.0/cond.h**2 + 1.0/(cond.h*2.0) * (y_k[2] - y_k[0])/(cond.h))*0.5
    d[n-2] = -1.0 - y_k[-2]**2 - ((y_k[-1] - y_k[-3])/(2.0*cond.h))**2 -\
    (1.0/cond.h**2 - 1.0/(cond.h*2.0) * (y_k[-1] - y_k[-3])/(cond.h))*(-0.5)
    for i, xi in enumerate(np.arange(cond.x0 + 2.0*cond.h, cond.xn - cond.h, cond.h
                          start=1):
        if (abs((cond.x0 + 2.0*cond.h) - (cond.xn - cond.h)) < 1e-7):
            continue
        a[i-1] = 1.0/cond.h**2 + 1.0/(cond.h*2.0) * (y_k[i+2] - y_k[i])/(cond.h)
        b[i] =  -2.0/cond.h**2 + (-2 * y_k[i+1] + 1.0)
        c[i] = 1.0/cond.h**2 - 1.0/(cond.h*2.0) * (y_k[i+2] - y_k[i])/(cond.h)
        d[i] = -1.0 - y_k[i+1]**2 - ((y_k[i+2] - y_k[i])/(2.0*cond.h))**2

    def tridiag(a, b, c, k1=-1, k2=0, k3=1):
        return np.diag(a, k1) + np.diag(b, k2) + np.diag(c, k3)
    if verbose:
        print("A={}".format(tridiag(a,b,c)))
        print("d={}".format(d.T))

    return(a,b,c,d)
```

In [6]:

```
def Thomas_Algorithm(coeff, cond):
    a, b, c, d = coeff
    n = int(np.ceil((cond.xn - cond.x0)/cond.h))
    c1 = np.zeros((n-2))
    d1 = np.zeros((n-1))
    c1[0] = c[0]/b[0]
    d1[0] = d[0]/b[0]
    for i in range(1,n-2):
        c1[i] = c[i]/(b[i]-a[i-1]*c1[i-1])
        d1[i] = (d[i] - a[i-1] * d1[i-1])/(b[i]-a[i-1]*c1[i-1])
    d1[n-2] = (d[n-2] - a[n-3] * d1[n-3])/(b[n-2] - a[n-3]*c1[n-3])
    def backsubstitution(c,d):
        y = np.zeros_like(d)
        y[-1] = d[-1]
        cache = y[-1]
        for i in reversed(range(d.shape[0]-1)):
            y[i] = d[i] - cache * c[i]
            cache = y[i]
        return y
    fin_y = backsubstitution(c1,d1)
    #print(fin_y)
    y = np.append(fin_y, cond.yn)
    y = np.append(cond.y0, y)
    return y
```

In [7]:

```
def Newton_Solver(cond):
    n = int(np.ceil((cond.xn - cond.x0)/cond.h))
    print(n)
    y_k = np.arange(cond.x0, cond.xn + cond.h, cond.h)
    #print(y_k)
    #print(initializer(y_k))
    solutions = []
    y_k = initializer(y_k)
    tuples_coeff = Finite_Diff_Solver(cond, y_k, verbose=False)
    y_k1 = Thomas_Algorithm(tuples_coeff, cond)
    s = np.max(abs(y_k1 - y_k))
    y_k = y_k1
    while(s > cond.e):
        y_k = copy(y_k1)
        solutions.append(y_k)
        tuples_coeff = Finite_Diff_Solver(cond, y_k, verbose=False)
        y_k1 = Thomas_Algorithm(tuples_coeff, cond)
        s = np.max(abs(y_k1 - y_k))
        #print(s)

    y_k = y_k1
    solutions.append(y_k)
    return solutions
```

In [8]:

```
cond = Conditions(0.0,np.pi,0.5,-0.5,Fraction(1,4),0.01)
```

In [9]:

```
y = Newton_Solver(cond)
print(len(y))
x_range = np.arange(cond.x0, cond.xn+cond.h, cond.h)
```

```
13
14
```

In [10]:

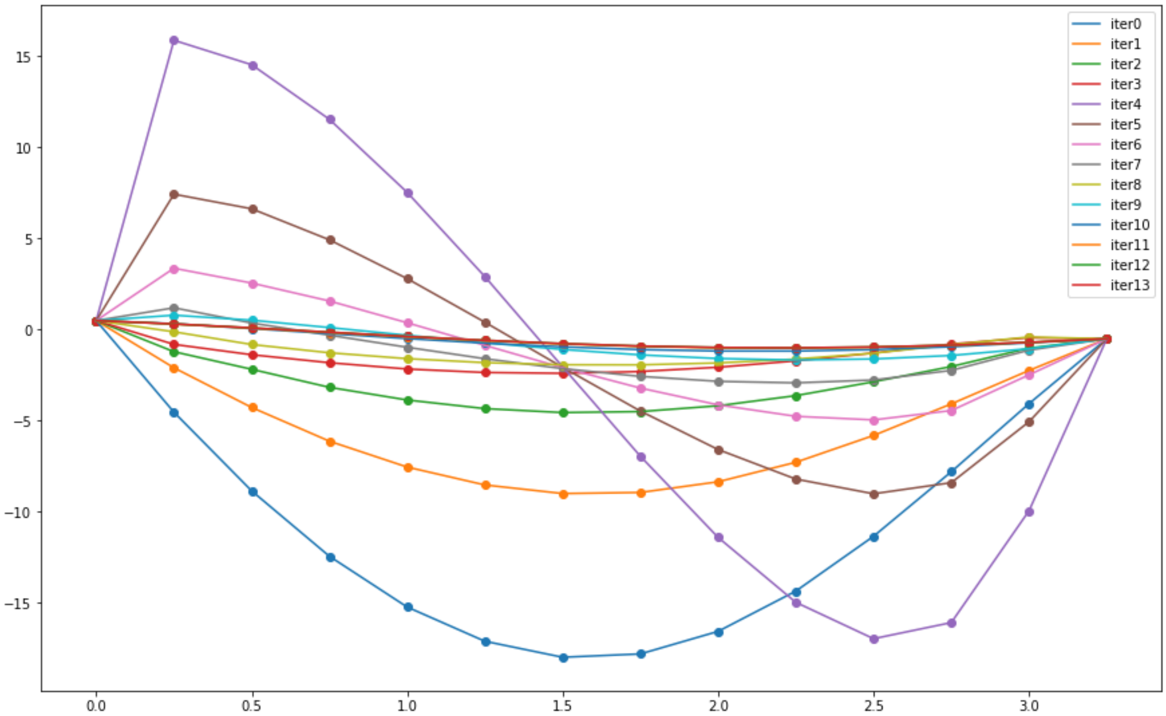
```
y[-1]
```

Out[10]:

```
array([ 0.5       ,  0.31761184,  0.09931491, -0.13569987, -0.3687996
8,
       -0.58286053, -0.76292585, -0.89679162, -0.97554184, -0.9939792
2,
       -0.95089132, -0.8491141 , -0.6953841 , -0.5       ])
```

In [11]:

```
t1 = np.arange(cond.x0, cond.xn, 0.001)
fig = plt.figure(figsize=(15, 32))
ax = fig.add_subplot(311)
g = []
for i in range(len(y)):
    ax.plot(x_range, y[i])
    ax.scatter(x_range, y[i])
    g.append("iter"+str(i))
ax.legend(tuple(g))
fig.show()
```

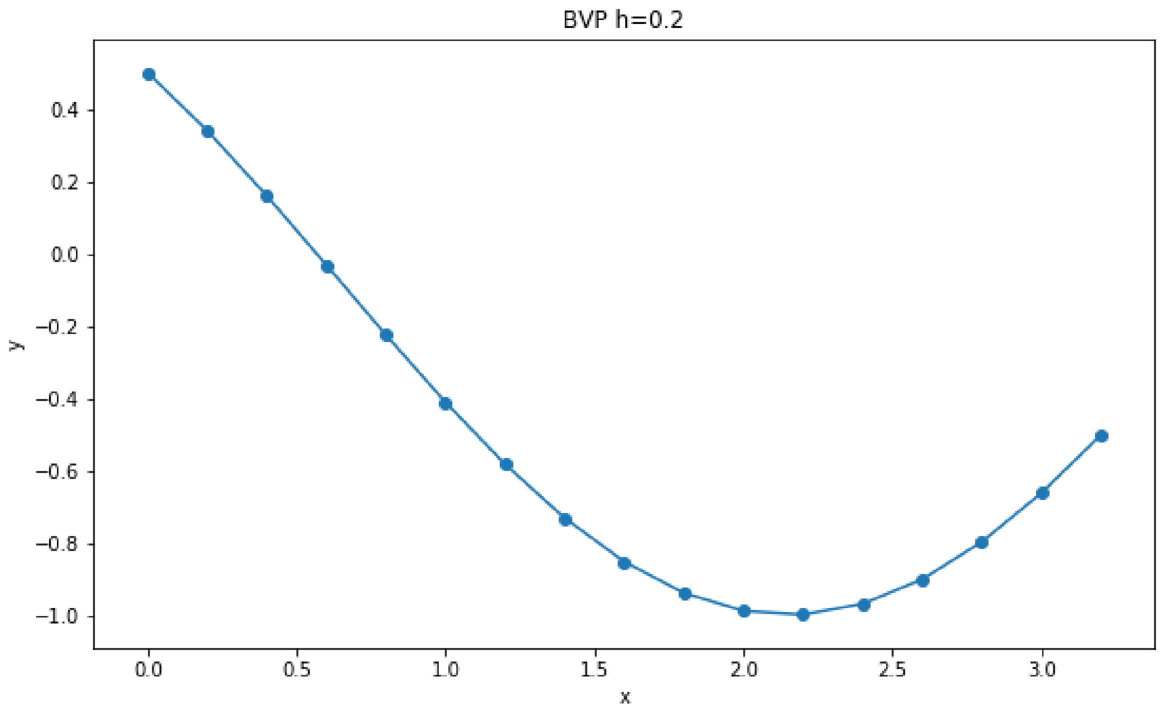


In [12]:

```
cond = Conditions(0.0,np.pi,0.5,-0.5,Fraction(1,5),0.01)
y = Newton_Solver(cond)
#print(len(y))
x_range = np.arange(cond.x0, cond.xn+cond.h, cond.h)
fig = plt.figure(figsize=(10,20))
ax = fig.add_subplot(311)
ax.set(title="BVP h=0.2", xlabel="x", ylabel="y")
ax.plot(x_range, y[-1])
ax.scatter(x_range, y[-1])

fig.show()
```

16



In [13]:

```
cond = Conditions(0.0,np.pi,0.5,-0.5,Fraction(1,10),0.01)
y = Newton_Solver(cond)
x_range = np.arange(cond.x0, cond.xn+cond.h, cond.h)
fig = plt.figure(figsize=(10,20))
ax = fig.add_subplot(311)
ax.plot(x_range, y[-1])
ax.set(title="BVP h=0.1", xlabel="x", ylabel="y")
ax.scatter(x_range, y[-1])
fig.show()
```

32