Hand in 5, part 1 of 3 - BVP:s and Direct Methods

1. Consider the BVP

$$y''(x) + y(x) = 0 \tag{1}$$
$$y(0) = 1 \tag{2}$$
$$y(\pi/2) = 2 \tag{3}$$

1 Pen and paper tasks

1. What is the analytical solution? We studied a very similar problem during the lecture.
2. Write down, with pen an paper, a discretization of the problem. Write down how the matrix would look for a stepsize $h = (\pi/2)/4$
3. Derive the local truncation error. Include your derivation in the hand in.
4. Is your method consistent?

2 Coding tasks

1. Implement your dicretization and plot it next to the analytical solution for the stepsize $h = (\pi/2)/4$ ($N = 3$). You can do this by filling in the missing pieces of the script below (you are of course also allowed to write your own script from scratch). Include the code and the plot in the hand in.
2. Print the matrix for $N = 3$ and compare it to the matrix that you derived by pen and paper. They should be the same. Include the printout in the hand in.
3. Where is the error the biggest? In the beginning of the interval, the end, or in the middle? Compare to how it is for IVP:s.
4. Produce a plot or a table which shows how the error decreases with h. You can look at your error-study from Lab 1 part 2 for inspiration. For measuring the error, choose any of the norms from the lecture. You can use `numpy.linalg.norm` to code the norm, but compensate for the fact that the 1-norm and 2-norm in numpy does not include the step size h, as in the lecture. Include the table/plot in your hand in. The result should be in accordance to the theoretical truncation error.

5. What command is used to solve the matrix-system in the code? In the next lectures we will look closer at what is actually happening in this step.

This should be included in this part of the hand in: The answers to the paper-and pen questions, including your derivation of the truncation error and how your matrix looks like. All answers to the coding questions, as well as the code, the plot for $N = 3$, the print out of the matrix, and the plot/table of how the error decreases with h. This is only part 1 out of 3, so no hand in today.

```
#!/usr/bin/python3

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

def ComputeAnalyticalSolution(N,leftbc,rightbc):
  h=(pi/2)/(N+1)
  x=np.linspace(0,(pi/2),N+2)

  y=

  return x,y


def ComputeNumericalSolution(N,leftbc,rightbc):

  h=(pi/2)/(N+1)
  x=np.linspace(0,(pi/2),N+2)

  A=np.zeros((N,N))
  F=np.zeros(N)

  # Assembly
  A[0,0]=
  A[0,1]=

  F[0]=

  for i in range(1,N-1):
    A[i,i-1]=
    A[i,i]=
    A[i,i+1]=
    F[i]=

  A[N-1,N-2]=
  A[N-1,N-1]=
  F[N-1]=

  y_h_int=np.linalg.solve(A,F)
```

```
  y_h=np.zeros(N+2)

  y_h[0]=
  y_h[1:N+1]=y_h_int
  y_h[-1]=

  return x,y_h,h

leftbc=1
rightbc=2

N=

x,y=ComputeAnalyticalSolution(N,leftbc,rightbc)
x_h,y_h,h=ComputeNumericalSolution(N,leftbc,rightbc)

plt.plot(x,y,Label='analytical')
plt.plot(x_h,y_h,Label='numerical')
plt.show()
```