ME 2450 Assignment HW 6

Name: CHRISTOPHER WALL

I declare that the assignment here submitted is original except for source material explicitly acknowledged.

I also acknowledge that I am aware of University policy and regulations on honesty in academic work, and of the disciplinary guidelines and procedures applicable to breaches of such policy and regulations, as contained in the University website.

Christopher Wall	11/5/2024
Name	Date
CHRISTOPHER WALL	u1467634
Signature	Student ID

Score

Total: 35 + 5

NOTE: All problems in this HW is ODE BVPs. No order reduction is needed for BVPs

Q1 (10 pts)

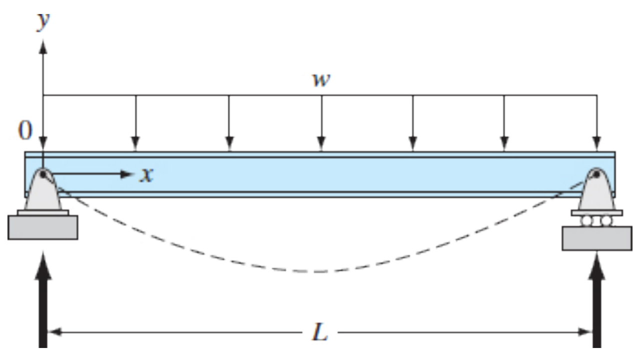
Use the finite-difference method to solve the BVP:

$$7\frac{d^2y}{dx^2} - 2\frac{dy}{dx} - y + x = 0$$

with $y(0) = 5$ and $y(20) = 8$.

1. Use a step size of 5 and solve the problem by hand. Include your work in the submitted pdf. *Hint:The result should be a linear system, which you can use any code/function to solve. In the end, please write down all matrices, vectors, and solution values clearly.*
2. Write a code to solve the problem. Plot $y(x)$ for at least 5 different step sizes (starting with $h = 5$ to check your hand-written results) on the same plot to illustrate convergence. *If you have difficulties assembling the matrix of different sizes by coding, take a look at the example code files in the Canvas folder for this HW.* Submit your plot in the pdf. Include ALL your code.

Q2 (10 pts)



The basic differential equation of the elastic curve for a uniformly loaded beam is given as:

$$EI\frac{d^2y}{dx^2} = \frac{wLx}{2} - \frac{wx^2}{2}$$

where E is the modulus of elasticity and I is the moment of inertia. Solve for the deflection, y, of the beam using the finite-difference approach with $h = 2$ [ft]. The following parameter values apply: $E = 30,000$ [ksi], $I = 800$ [in^4], $w = 1$ [kip/ft], $L = 10$ [ft]. Be careful of units. Compare your numerical results to the following analytical solution by plotting both on the same plot:

$$y = \frac{wLx^3}{12EI} - \frac{wx^4}{24EI} - \frac{wL^3x}{24EI}$$

Submit your plot in the pdf. Include ALL your code.

Q3: (15+5 pts) Heat Transfer in a Thin Rod

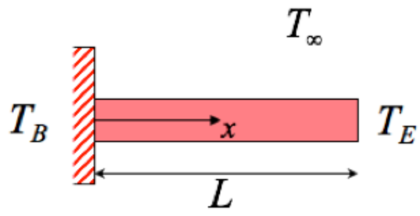


Figure 1: Schematic of heat transfer in a thin rod with round cross-section.

In class (videos), a model of heat transfer in a thin rod (Figure 1) was presented. This might represent the cooling effect of a heatsink in a computer, or the cooling of a rod in a nuclear reactor. In the case shown, the ODE for temperature, T, with respect to location on the rod, x, is:

$$\frac{d^2T}{dx^2} = \frac{hP}{\kappa A}(T - T_\infty) \tag{1}$$

where the parameters are defined as follows:

- h is the heat transfer coefficient (in W/m˙K)
- κ is the thermal conductivity of the rod material (in W/m · K)
- P is the perimeter of the cross-sectional area of the rod (in m)
- A is the cross-sectional area of the rod (in m^2)
- T_∞ is the ambient temperature (in K)

Finite Difference Method

The finite difference method (Section 27.1.2 of the Textbook) involves replacing derivatives in a differential equation with finite difference approximations. For this assignment, you will use this method AND one of your linear system methods (e.g. Gauss elimination, Gauss-Seidel) to solve for temperatures in a cooling rod.

Parameter Definitions

For all exercises, use the following parameters:

$h = 20\text{W/m}^2 \cdot \text{K}$	$\kappa = 200\text{W/m} \cdot \text{K}$	
$L = 2.0\text{m}$	$D = 0.1\text{m}$	
$T_\infty = 300\text{K}$	$T_B = 600\text{K}$	$T_E = 350\text{K}$

- D is the diameter of the round rod (in m)
- L is the length of the round rod (in m)

Exercise 1 (5 pts): Finite Difference Method by Hand

- Discretize the rod into 5 nodes, where one node lies at the left edge and one lies at the right edge and write the system of equations.
- Solve this system of equations either by hand or using one of your codes (e.g. Gauss elimination, LU, or Gauss-Seidel). Include ALL your code.

Exercise 2 (10 pts): Finite Difference Method

- Discretize the rod into 51 nodes, where one node lies at the left edge and one lies at the right edge and write a code to generate the system of equations. Include ALL you code.
  (Hint: **You matrix should be 49-by-49 in size.** If you have difficulties assembling the matrix by coding, take a look at the example code files in the Canvas folder for this HW)
- Solve this system of equations using one of your codes (e.g. Gauss elimination, LU, or Gauss-Seidel). Include ALL you code.
- Plot the temperature vs. position along the rod. How do these values compare to your results from the calculation in Exercise 1?

Extra Credit Exercise (+5 pts): Built-in Solvers

- Use a built-in direct ODE BVP solver (e.g. `bvp4c` in Matlab or `scipy.integrate.solve_bvp` in Python or any other built-in method of your choosing) to solve the problem.
- Compare this result with your results from the finite difference code by plotting them together.
- NOTE: Dr. Pai and your TA will NOT answer any question regarding this extra credit excercise.

  Using a built-in solver is generally much easier than coding up your own method.

  The only challenge here is your self-study ability to find the correct way to use a new tool.

$$7\left(\frac{1}{h^2}(y_{i+1}-2y_i+y_{i-1})\right)-2\left(\frac{1}{2h}(y_{i+1}-y_{i-1}\right)-y_i$$

$$\underbrace{\left(\frac{7}{h^2}+\frac{2}{2h}\right)}_{\alpha}y_{i-1}+\underbrace{\left(\frac{-14}{h^2}-1\right)}_{\beta}y_i+\underbrace{\left(\frac{7}{h^2}-\frac{2}{2h}\right)}_{\gamma}y_{i+1}$$

$\alpha = .48$

$\beta = -1.56$

$\gamma = 0.08$

$$\begin{bmatrix} -1.56 & .08 & 0 \\ .48 & -1.56 & .08 \\ 0 & .48 & -1.56 \end{bmatrix}\begin{bmatrix} y_5 \\ y_{10} \\ y_{15} \end{bmatrix} = \begin{bmatrix} -5-5(.48) \\ -10 \\ -15-8(.08) \end{bmatrix}$$

Using code:

$y_5 = 5.188$

$y_{10} = 8.687$

$y_{15} = 12.689$

Q2

$$EI\left(\frac{1}{h^2}(y_{i+1}-2y_i+y_{i-1})\right) = \frac{wLx}{2} - \frac{wx^2}{2}$$

$\alpha : \dfrac{EI}{h^2}$

$\beta : \dfrac{-2EI}{h^2}$

$\gamma : \dfrac{EI}{h^2}$

Q3

nodes : $[0.0, 0.5, 1.0, 1.5, 2.0]$

$$\frac{\partial^2 T}{\partial x^2} = \frac{hP}{KA}(T - T_\infty)$$

$$\Rightarrow \left(\frac{1}{h^2}(y_{i-1} - 2y_i + y_{i+1})\right) = \frac{hP}{KA}(T - T_\infty)$$

$$\left(\frac{1}{h^2}(y_{i-1} - 2y_i + y_{i+1})\right) - \frac{hP}{KA} y_i = -\frac{hP}{KA} T_\infty$$

$$\alpha = \frac{1}{h^2} = 0.04$$

$$\beta = \frac{-2}{h^2} - \frac{hP}{KA} = -4.08$$

$$\gamma = \frac{1}{h^2} = 0.04$$

$$-\frac{hP}{KA} T_\infty = -1200$$

$$\begin{bmatrix} \beta & \gamma & 0 \\ \alpha & \beta & \gamma \\ 0 & \alpha & \beta \end{bmatrix} \begin{bmatrix} y_{.5} \\ y_{1.0} \\ y_{1.5} \end{bmatrix} = \begin{bmatrix} -1200 - \alpha(600) \\ -1200 \\ -1200 - \gamma(350) \end{bmatrix}$$

$$\Rightarrow [600, 417, 350, 333, 350]$$

```
import Linsolve
import numpy as np
#Problem 1:
A = [[-1.56, .08, 0],
     [.48, -1.56, .08],
     [0, .48, -1.56]]

b = [-5 - 5*(.48), -10, -15 - 8*(.08)]

x = Linsolve.naive_gauss_elimination(A, b)
solution = [5.0]
solution.extend(x)
solution.append(8.0)
print(f'values for x = 0:20 in increments of 5: {solution}')



#Problem 3:

#Constants:
h0 = 20       #W/mK
k = 200      #W/mK
L = 2        #m
D = 0.1      #m
Tinf = 300   #K
Tb = 600     #K
Te = 350     #K

P = D*np.pi
A = np.pi * (D/2)**2
c = h0*P/(k*A)

h = .5
alpha = 1/h**2
beta = -2/h**2 - 4 #hP/kA = 4
gamma = 1/h**2
hpkaT = -1200 #
A = [[beta, gamma, 0],
     [alpha, beta, gamma],
     [ 0, alpha, beta]]
b = [ hpkaT - 600*alpha, hpkaT, hpkaT - 350*gamma]
solution = [600]
solution.extend( Linsolve.naive_gauss_elimination(A, b))
solution.append(350)
print(solution)
```

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

def F(x):
    return -x

plt.figure()
#Plot results for different sizes of h
for i in [5, 2.5, 1.25, .625, .0375]:
    params = [7/i**2 + 1/i ,    #alpha
              -14/i**2 - 1,     #beta
              7/i**2 - 1/i]     #gamma
    results, independent = Linsolve.BVPsolve(0, 20, 5, 8, i, params, F, solveMethod='Gauss')
    plt.plot(independent, results, label=f'h = {i}')
plt.title('BVP solution')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```

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

E = 30000           #ksi
I = 0.03858024688   #in^4
w = 1               #kip/ft
L = 10              #ft

#RHS of ODE
def F(x):
    return w*L*x/2 - w*x**2/2

def analytical(x):
    return w*L*x**3/(12*E*I) - w*x**4/(24*E*I) - w*L**3*x/(24*E*I)

plt.figure()
h = 2
alpha = E*I/h**2
beta = -2*E*I/h**2
gamma = E*I/h**2
params = [alpha, beta, gamma]
results, independent = Linsolve.BVPsolve(0, L, 0, 0, h, params, F,solveMethod='Seidel')

#plotting my results vs analytical
plt.plot(independent, results, label=f'h = {h}')
i = np.arange(0, L, .1)
plt.plot( i, analytical(i), label='Analytical')
plt.title('BVP solution')
plt.xlabel('x [ft]')
plt.ylabel('deflection [ft]')
plt.legend()
plt.show()
```

```
import numpy as np
import matplotlib.pyplot as plt
import Linsolve
import scipy.integrate as sci

#Constants:
h = 20      #W/mK
k = 200     #W/mK
L = 2       #m
D = 0.1     #m
Tinf = 300  #K
Tb = 600    #K
Te = 350    #K

P = D*np.pi
A = np.pi * (D/2)**2
c = h*P/(k*A)           #So I dont have to rewite hP/kA
print(c)

def F(x):
    return -c*Tinf

plt.figure()
h = .04
params = [1/h**2 ,          #alpha
        -2/h**2 - c,        #beta
        1/h**2 ]            #gamma
results, independent = Linsolve.BVPsolve(0, L, Tb, Te, h, params, F)
plt.plot(independent, results, label=f'h = {h}')
plt.title('BVP solution')
plt.xlabel('x [ft]')
plt.ylabel('temperature [K]')
plt.legend()
plt.show()

#The results from my computed results match my results by hand - bless up
```

```python
import numpy as np

def BVPsolve(xi, xf, yi, yf, h, params, F, solveMethod='Gauss'):
    tVals = np.arange(xi, xf + h, h)
    yVals = np.ones_like(tVals)
    num_interior_nodes = len(tVals) - 2
    bVector = np.zeros(len(tVals) - 2)

    for i in range(num_interior_nodes):
        bVector[i] = F(tVals[i + 1])

    # Define finite difference coefficients for the given equation
    alpha, beta, gamma = params

    # Apply boundary conditions to the first and last elements of bVector
    bVector[0] -= yi * alpha
    bVector[-1] -= yf * gamma
    A = np.zeros((num_interior_nodes, num_interior_nodes))
    A = A + np.diag(beta * np.ones(num_interior_nodes))
    A = A + np.diag(gamma * np.ones(num_interior_nodes - 1), 1)
    A = A + np.diag(alpha * np.ones(num_interior_nodes - 1), -1)

    #Solve matrix
    if solveMethod == 'Seidel':
        x = np.random.random(len(bVector)) * 10*( yi + yf)/2
        interior_nodes = seidel_solve(A, bVector, x)
    else:
        interior_nodes = naive_gauss_elimination(A, bVector)

    #output results including boundary conditions
    output =[yi]
    output.extend(interior_nodes)
    output.append(yf)
    return output, tVals

#from another HW
def seidel_solve(A, b, x, tol=1e-6, max_iter=1000):
    n = len(b)
    iter = 0
    while iter < max_iter:
        previous_x = x.copy()  # Make a full copy of the current solution
        for i in range(n):
            sigma = 0
            for j in range(n):
                if i != j:
                    sigma += A[i][j] * x[j]
            x[i] = (b[i] - sigma) / A[i][i]  # Gauss-Seidel update
        if max(np.abs(np.subtract(previous_x, x))) < tol:
            return x
        iter += 1

    print("Warning: Max iterations exceeded without convergence")
    return x


#from another HW
def naive_gauss_elimination(a,b):
    #size checking:
    m,n = np.shape(a)
    if m != n:
        raise TypeError('A is not a square matrix')
    if len(a) != len(b):
        raise TypeError('A is not the same size as B')

    #Forward elimination
    for k in range(0, n-1):
        for i in range(k+1, n):
            s = a[i][k]/a[k][k]
            for j in range(k, n):
                a[i][j] = a[i][j] - s*a[k][j]
            b[i] = b[i] - s*b[k]

    #Backwards solve
    x = np.zeros(n)
    x[-1] = b[-1]/a[-1][-1]
    for i in range(n-2, -1, -1):
        s = 0
        for j in range( i +1, n):
```

```
        s = s + a[i][j]*x[j]
    x[i] = (b[i] - s)/a[i][i]

return x
```