Math 104B Homework 6

Rad Mallari, 8360828

May 22nd, 2022

1.) (a) Implement the tridiagonal solver seen in class

In [1]:
```python
from math import pi, sin, sqrt
from matplotlib import pyplot as plt
def tridiagonal_matrix_solver(a, b, c, d):
    n = len(a)
    # Initialize variables
    m, l, y, x = [], [], [], [0 for i in range(n)]

    m.append(a[0])
    for j in range(n-1):
        l.append(c[j] / m[j])
        m.append(a[j+1] - l[j] * b[j])

    # Forward substitution
    y.append(d[0])
    for j in range(1, n):
        y.append(d[j] - l[j - 1] * y[j - 1])

    # Backward substitution
    x[-1] = y[-1] / m[-1]
    for j in reversed(range(n-1)):
        x[j] = (y[j] - b[j] * x[j + 1]) / m[j]
    return x
```

(b) Test implementation

In [2]:
```python
b_n = [-1,-1,-1,-1]
a_n = [3,3,3,3,3]
c_n = [-1,-1,-1,-1]
d_n = [2,1,1,1,2]
result = tridiagonal_matrix_solver(a_n, b_n, c_n, d_n)
print(result)
```

```
[1.0, 1.0, 1.0, 1.0, 1.0]
```

2.) Consider the boundary value problem:

$$\begin{aligned} -u'' + \pi^2 u &= 2\pi^2 \sin(\pi x) \quad 0 < x < 1, \\ u(0) &= u(1) = 0 \end{aligned} \tag{1}$$

We can find a numerical approximation to the solution of this problem by employing the finite difference method.
Use a uniform grid with $N-1$ interior nodes to obtain, by replacing the second order derivative with a second order finite difference and neglecting the (truncation) error, the linear system

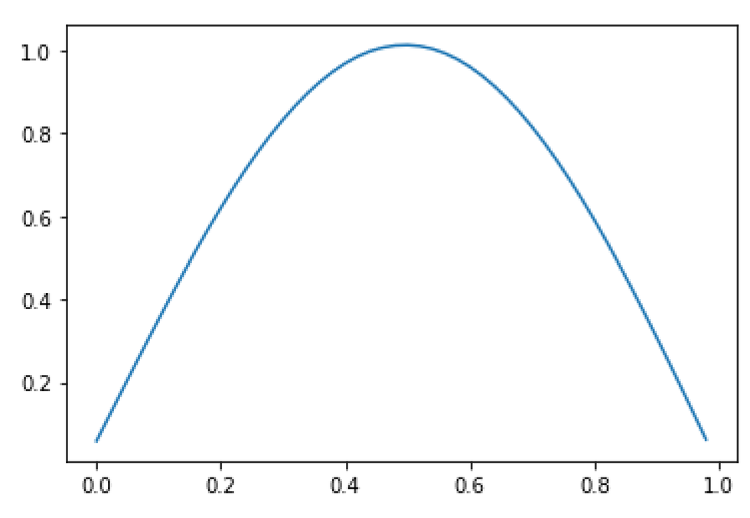
$$\frac{-v_{j-1} + 2v_j - v_{j+1}}{h^2} + \pi^2 v_j = 2\pi^2 \sin(\pi x_j) \quad \text{for } j = 1, 2, \ldots, N-1 \tag{2}$$

where $h = \frac{1}{N}$, v_j is the approximation to $u(x_j)$ for $j = 1, 2, \ldots, N-1$, and $v_0 = v_N = 0$.

(a) Use your tridiagonal solver to solve the equation **(1)** for $N = 50$ and plot your corresponding solution.

In [3]:
```python
N = 50
h = 1/N

x_n = [x*h for x in range(N)]
a_j, b_j, c_j, f_j = [], [], [], []

for i in range(N-1):
    a_j.append(-1)
    c_j.append(-1)
for i in range(N):
    b_j.append(2+h**2*pi**2)

f_j.append(0)
for i in range(1, N-1):
    two_pi_squared = 2*(pi**2)*h**2
    f_j.append(two_pi_squared*sin(pi*x_n[i]))
f_j.append(0)

v_j = tridiagonal_matrix_solver(b_j, a_j, c_j, f_j)

plt.plot(x_n, v_j)
```

Out[3]:
```
[<matplotlib.lines.Line2D at 0x1f12b9d0370>]
```



(b) The exact solution to the boundary value problem **(1)** is $u(x) = \sin(\pi x)$. Check this.

In [4]:
```python
exact_solution = []
for i in range(N):
    exact_solution.append(sin(x_n[i]*pi))
print("Approximated solution\t|\tExact Solution")
print("-"*51)
for i in range(N):
    print(f"{v_j[i]}\t|\t{exact_solution[i]}")
```

```
Approximated solution   |       Exact Solution
---------------------------------------------------
0.058961398156278544    |       0.0
0.11815556658245212     |       0.06279051952931337
0.1773204204182933      |       0.12533323356430426
0.2361957156679686      |       0.1873813145857246
0.29452397066990016     |       0.2486898871648548
0.3520513830587968      |       0.3090169943749474
0.4085287382093966      |       0.3681245526846779
0.4637123052087771      |       0.4257792915650727
0.5173647164755499      |       0.4817536741017153
0.5692558272296767      |       0.5358267949789967
0.6191635511157552      |       0.5877852522924731
0.6668746683951311      |       0.6374239897486896
0.7121856032477072      |       0.6845471059286886
0.7549031668624244      |       0.7289686274214116
0.7948452631456027      |       0.7705132427757893
0.8318415540381138      |       0.8090169943749475
0.865734081605131       |       0.8443279255020151
0.8963778442453433      |       0.8763066800438637
0.9236413245593448      |       0.9048270524660196
0.9474069666187114      |       0.9297764858882513
0.9675716005873014      |       0.9510565162951535
0.9840468128637724      |       0.9685831611286311
0.9967592601383737      |       0.9822872507286886
1.0056509259869097      |       0.9921147013144779
1.0106793188594947      |       0.9980267284282716
1.0118176105604415      |       1.0
1.0090547145574316      |       0.9980267284282716
1.0023953037020754      |       0.9921147013144778
0.9918597671891558      |       0.9822872507286886
0.977484106827301       |       0.9685831611286312
0.9593197729386244      |       0.9510565162951536
0.9374334404480374      |       0.9297764858882515
0.9119067259635584      |       0.9048270524660195
0.8828358468860659      |       0.8763066800438635
0.8503312238196679      |       0.844327925502015
0.8145170277812697      |       0.8090169943749475
0.7755306739291411      |       0.7705132427757893
0.7335222637444507      |       0.7289686274214114
0.6886539778060164      |       0.6845471059286888
0.6410994214961135      |       0.6374239897486899
0.5910429261633173      |       0.5877852522924732
0.5386788084463018      |       0.5358267949789967
0.4842105906295753      |       0.4817536741017156
0.4278501850576573      |       0.4257792915650729
0.36981704577758145     |       0.36812455268467814
0.3103372907102867      |       0.3090169943749475
0.249642797768923       |       0.24868988716485482
0.18797027844588857     |       0.18738131458572457
0.1255603324801298      |       0.12533323356430454
0.06265648729151906     |       0.06279051952931358
```

(c) Compute the error of your approximation in the 2-norm for $N = 50$. Solve **(1)** for $N = 100$, by how much do you expect the error to decrease? Verify your answer by comparing the error for $N = 50$ and $N = 100$.

In [5]:
```python
error_2_norm_50 = sqrt(1/N*sum([x - y for x, y in zip(v_j, exact_solution)]))
print(f"Error of approximation in 2-norm for N = 50: {error_2_norm_50}")
print(f"I expect the same amount of error between N = 50 and N = 100")
N = 100
h = 1/N

x_n = [x*h for x in range(N)]
a_j, b_j, c_j, f_j = [], [], [], []

for i in range(N-1):
    a_j.append(-1)
    c_j.append(-1)
for i in range(N):
    b_j.append(2+h**2*pi**2)

f_j.append(0)
for i in range(1, N-1):
    two_pi_squared = 2*(pi**2)*h**2
    f_j.append(two_pi_squared*sin(pi*x_n[i]))
f_j.append(0)

v_j = tridiagonal_matrix_solver(b_j, a_j, c_j, f_j)

error_2_norm_100 = sqrt(1/N*sum([x - y for x, y in zip(v_j, exact_solution)]))
print(f"Error of approximation in 2-norm for N = 100: {error_2_norm_50}")
```

```
Error of approximation in 2-norm for N = 50: 0.13332401513551698
I expect the same amount of error between N = 50 and N = 100
Error of approximation in 2-norm for N = 100: 0.13332401513551698
```

(d) In real applications we do not know the exact solution. Describe a process to check the convergence and rate of convergence of your approximation if you don't know the exact solution.

Proof:

3.) Consider the linear system

$$x_1 - 2x_2 + x_3 = -1$$
$$2x_1 + x_2 - 3x_3 = 3$$
$$x_1 - x_2 + x_3 = 0$$

(a) Do the first iterations of Jacobi.

Proof:
Solving for x_1 in the first equation, x_2 in the second equation, and x_3 in the third equation, we have:

$$x_1^{k+1} = -1 - 2x_2^k - x_3^k$$
$$x_2^{k+1} = 3 - 2x_1^k + 3x_3^k$$
$$x_3^{k+1} = -x_1^k + x_2^k$$

where k is the number of iterations. Now taking the first guess $x_1 = x_2 = x_3 = 0$, we have the first iteration:

$$x_1^1 = -1 + 2 \cdot 0 - 0 = -1$$
$$x_2^1 = 3 - 2 \cdot 0 + 3 \cdot 0 = 3$$
$$x_3^1 = -0 + 0 = 0$$

which tells us that $x^1 = [-1, 3, 0]$

(b) Do the first two iterations of Gauss-Seidel.

Proof:
We begin simlarly by solving for x_1 in the first equation, x_2 in the second equation, and x_3 in the third equation, yielding:

$$x_1^{k+1} = -1 - 2x_2^k - x_3^k$$
$$x_2^{k+1} = 3 - 2x_1^k + 3x_3^k$$
$$x_3^{k+1} = -x_1^k + x_2^k$$

Taking the first guess $x_1 = x_2 = x_3 = 0$ (this is our first iteration), then using the most recent value for x_1, x_2, x_3, we have the second iteration:

$$x_1^1 = -1 + 2 \cdot 0 - 0 = -1$$
$$x_2^1 = 3 - 2 \cdot -1 + 3 \cdot 0 = 5$$
$$x_3^1 = -(-1) + 5 = 6$$

Giving us $x^1 = [-1, 5, 6]$

(c) Which of the two approximations is closer to the exact solution $(1, 1, 0)$?

Proof:
The Jacobi is a closer approximation since the distance (ℓ^2 norm) is lesser than Gauss-Seidel.

4.) Consider the system

$$2x_1 - x_2 + x_3 = -1$$
$$2x_1 + 2x_2 + 2x_3 = 4$$
$$-x_1 - x_2 + 2x_3 = -5$$

By finding the spectral radius of the Jacobi and Gauss-Seidel iteration matrices prove that the Jacobi method diverges while Gauss-Seidel's method converges for this system.

Proof:
The spectral radius of some matrix A is given by:

$$\rho(A) = \max_{1 \le i \le n} |\lambda_i|$$

Where λ_i are the eigenvalues of the matrix and n is number of rows in the matrix.
The Jacobi method decomposes A to it's diagonal, D, lower triangular, L, and upper triangular U matrices.
Through iteration, we obtain the solution of the form:

$$\begin{aligned} x^{k+1} &= D^{-1}(b - (L+U)x^k) \\ &= D^{-1}b - D^{-1}(L+U)x^k \end{aligned} \tag{3}$$

Meanwhile the condition for convergence is $\rho(D^{-1}(L+U)) < 1$ so for our system:

$$A = \begin{bmatrix} 2 & -1 & 1 \\ 2 & 2 & 2 \\ -1 & -1 & 2 \end{bmatrix}, \quad b = \begin{bmatrix} -1 \\ 4 \\ -5 \end{bmatrix}$$

And our decomposed matrix is given by:

$$A = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 2 & 0 & 0 \\ -1 & -1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & -1 & 1 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \end{bmatrix}$$

So $D^{-1}(L+U)$ is:

$$\begin{aligned} D^{-1}(L+U) &= \begin{bmatrix} \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} \end{bmatrix} \left(\begin{bmatrix} 0 & 0 & 0 \\ 2 & 0 & 0 \\ -1 & -1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & -1 & 1 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \end{bmatrix} \right) \\ &= \begin{bmatrix} 0 & -\frac{1}{2} & \frac{1}{2} \\ 1 & 0 & 1 \\ -\frac{1}{2} & -\frac{1}{2} & 0 \end{bmatrix} \end{aligned} \tag{4}$$

After calculations, we find that the eigenvalues are $\lambda = \{-\frac{\sqrt{5}}{2}i, \frac{\sqrt{5}}{2}i, 0\}$ and certainly the max eigenvalue is greater than 1.
Therefore, the Jacobi method diverges for this system.

Now the Gauss-Seidel method is defined as:

$$L_* x^{k+1} = b - Ux^k$$

where L_* is the lower triangular component of A and U is the strictly upper trianglur component.
This can be rewritten as:

$$x^{k+1} = L_*^{-1}b - L_*^{-1}Ux^k$$

Therefore, we must check that the spectral radius $\rho(L_*^{-1}U) < 1$ for this to converge, which in this case is given by:

$$\begin{bmatrix} \frac{1}{2} & 0 & 0 \\ -\frac{1}{2} & \frac{1}{2} & 0 \\ 0 & \frac{1}{4} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 0 & -1 & 1 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -1/2 & 1/2 \\ 0 & 1/2 & 1/2 \\ 0 & 0 & 1/2 \end{bmatrix}$$

Which have the eigenvalues $\lambda = \{0, \frac{1}{2}\}$, which clearly $\frac{1}{2} < 1$, therefore Gauss-Seidel method converges.