PROBLEM SET 1: GEO2300:
DUE: 16 SEPT. 2020

GEO2300: FYSISKE PROSESSER I GEOFAG

SIGURD SANDVOLL SUNDBERG

Contents

1. Problem 1: Matricies 1
2. Problem 2: Poisueille Flow 3
3. Problem 3: More finite differences 7
4. Code addition 8

Date: September 9, 2020.

1. Problem 1: Matricies

In this section we will study the properties of matricies. How they are used to solve a system of linear equations, inverting matricies and finding eiegenvalues and eigenvectors, and lastly expressing matricies as a sum of eigenvectors and exponentials with corresponding eigenvalues.

First we will look at solving the following system of linear equations as a matrix equation

$$\begin{aligned} x+2y+z&=-1 \\ 2x-y+3z&=-5 \\ -x+3y-z&=6 \end{aligned} \tag{1}$$

We can rewrite Eq: 1 on the form $\mathbf{A}\vec{x}=\vec{b}$

$$\begin{bmatrix} 1 & 2 & 1 & -1 \\ 2 & -1 & 3 & -5 \\ -1 & 3 & -1 & 6 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \\ -5 \\ 6 \end{bmatrix} \tag{2}$$

We can rewrite this system on the form $[\mathbf{A}:\vec{b}]$ and solve the augmented matrix by finding the row reduced echelon form. We then have

$$\left[\begin{array}{ccc|c} 1 & 2 & 1 & 1 \\ 2 & -1 & 3 & -5 \\ -1 & 3 & -1 & 6 \end{array}\right] \sim \left[\begin{array}{ccc|c} 1 & 2 & 1 & 1 \\ 0 & -5 & 1 & -7 \\ 0 & 5 & 0 & 7 \end{array}\right] \tag{3}$$

$$\left[\begin{array}{ccc|c} 1 & 2 & 1 & 1 \\ 0 & 1 & 0 & 7/5 \\ 0 & 0 & 1 & 0 \end{array}\right] \sim \left[\begin{array}{ccc|c} 1 & 0 & 0 & -9/5 \\ 0 & 1 & 0 & 7/5 \\ 0 & 0 & 1 & 0 \end{array}\right] \tag{4}$$

giving us that the solution to the set of linear equations given in Eq: 1 is given by

$$\begin{aligned} x &= -9/5 \\ y &= 7/5 \\ z &= 0 \end{aligned}$$

This can easily be confirmed numerically by doing $rref([A:b])$.

Second, we will look at inversing matrixes, both analytically for the 2x2-matrix and numerically for the others. Looking at the 2x2-matrix we have that in the general case

$$A=\begin{bmatrix} a & b \\ c & d \end{bmatrix}, \quad A^{-1}=\frac{1}{ad-bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

We then have for our 2x2-matrix as follows

$$A=\begin{bmatrix} 1 & 2 \\ -1 & 3 \end{bmatrix}, \quad A^{-1}=\frac{1}{3\cdot 1-(-1\cdot 2)}\begin{bmatrix} 3 & -2 \\ 1 & 1 \end{bmatrix}=\frac{1}{5}\begin{bmatrix} 3 & -2 \\ 1 & 1 \end{bmatrix} \tag{5}$$

For the two next matricies being a 3x3 and 4x4 matrix, which are fairly tedious to solve by hand, we will use functions in Python 3 to do the inverting for us, spesifically the Numpy library.

```python
import numpy as np

A = np.matrix([[1, 2, 5], [-1, 3, -1], [2, 1, -2]])  # 3x3 matrix
B = np.matrix([[1, 2, 5, 2], [-1, 3, -1, -1],
               [2, 1, -2, 1], [1, -1, 1, -1]])  # 4x4 matrix
A = np.linalg.inv(A)
B = np.linalg.inv(B)
```

Which gives the following output

```
A =
[[ 0.10416667  -0.1875       0.35416667]
 [ 0.08333333   0.25         0.08333333]
 [ 0.14583333  -0.0625      -0.10416667]]
B =
[[ 4.16666667e-02  -1.38777878e-17   2.91666667e-01   3.75000000e-01]
 [ 8.33333333e-02   2.50000000e-01   8.33333333e-02   0.00000000e+00]
 [ 1.25000000e-01   1.15648232e-18  -1.25000000e-01   1.25000000e-01]
 [ 8.33333333e-02  -2.50000000e-01   8.33333333e-02  -5.00000000e-01]]
```

Where A is the 3x3-matrix, and B is the 4x4-matrix.

Lastly we will take a look at eigenvalues and eigenvectors. Assume that

$$\frac{d}{dt}\vec{M} = \mathbf{A}\vec{M} \tag{6}$$

where

$$A = \begin{bmatrix} 1 & 0 & 5 \\ -1 & 1 & 0 \\ 2 & 1 & -2 \end{bmatrix} \tag{7}$$

and $\vec{M}$ is a 3x1 vector.

So we find the eigenvalues and eigenvectors by hand however this is extremely tedious[1] so we are finding the eigenvalues and vectors numerically. The following Python code finds the eigenvalues and eigenvectors for us.

```python
import numpy as np
from numpy import linalg as LA

A = np.matrix([[1, 0, 5], [-1, 1, 0], [2, 1, -2]])
eigenvalues,  eigenvectors = LA.eig(A)  # v[:,i] = eigenvalue to w[i]
    A
print(f"Eigenvalues for Matrix A = \n {A}")
print(f"Is as follows: {eigenvalues}")
print(f"and their corresponding eigenvectors are \n {eigenvectors} \n"
    )

"""
Eigenvalues for Matrix A =
 [[ 1  0  5]
 [-1  1  0]
 [ 2  1 -2]]
Is as follows: [-4.13640529  2.47760887  1.65879642]
and their corresponding eigenvectors are
 [[-0.69118387 -0.804427   -0.54870228]
 [-0.13456568  0.54441132  0.83288595]
 [ 0.7100401  -0.2377257  -0.07229662]]
"""
```

As writing our these values by hand in each step from now on, we will refer to the eigenvalues as $\lambda_1, \lambda_2, \lambda_3$ from left to right. Their accompanying eigenvectors v_1, v_2, v_3.

Our expression for $\frac{d}{dt}\vec{M}$ is now given by

$$\frac{d}{dt}\vec{M} = \mathbf{A}\vec{M} = (\lambda_1 v_1 + \lambda_2 v_2 + \lambda_3 v_3)\,\vec{M}(t) \tag{8}$$

[1]And would result in a waste of the rain forest

We can write $M'(t) = \frac{d}{dt}M(t)$ as a linear combination of vectors on the form $M'(t) = \sum_{i=1}^{n} \vec{v}_i e^{\lambda_i t}$. This gives us

$$M(t) = v_1 e^{\lambda_1 t} + v_2 e^{\lambda_2 t} + v_3 e^{\lambda_3 t} \tag{9}$$

Looking at what the terms converge or diverge to, we have that $e^{\lambda_1 t} \to 0$ as $t \to \infty$. Both of the two other values $e^{\lambda_2 t}$ and $e^{\lambda_3 t}$ both goes towards infinity as $t \to \infty$. However if we look at the leading coefficients for t, we see that for $\lambda_2 \approx 2.5$ and $\lambda_3 \approx 1.6$ that λ_2 approaches infinity faster than λ_3 and thus is sets the dominating term. So that the dominating term as $t \to \infty$ is given by $v_2 e^{\lambda_2 t}$.

2. Problem 2: Poisueille Flow

Given the equation

$$\mu \frac{\partial^2 v}{\partial y^2} = \frac{\partial p}{\partial x} \tag{10}$$

we are interested in finding the exact solution dependent on v if $\partial p / \partial x = constant$, given the boundary conditions, $v(0) = v(h) = 0$ and the range of y is 0 to h. We then have

$$\mu \int \frac{\partial^2 v}{\partial y^2}\, dy = \frac{\partial p}{\partial x} \int dy \tag{11}$$

$$\mu \frac{\partial v}{\partial y} + C_1 = \frac{\partial p}{\partial x} y + C_2 \tag{12}$$

We define a new constant $D = C_1 + C_2$ and we get

$$\mu \int \frac{\partial v}{\partial y} + D\, dy = \frac{\partial p}{\partial x} \int y\, dy \tag{13}$$

$$\mu v(y) + Dy + C_3 = \frac{\partial p}{2\partial x} y^2 + C_4 \tag{14}$$

We define $E = C_3 - C_4$. We can now rewrite our equation as such[2]

$$\mu v(y) = \frac{1}{2}\frac{\partial p}{\partial x} y^2 + Dy + E \tag{15}$$

Inserting the boundary conditions $v(0) = v(h) = 0$, we get

$$\mu v(0) = 0 = 0 + 0 + E \to E = 0 \tag{16}$$

$$\mu v(h) = 0 = \frac{1}{2}\frac{\partial p}{\partial x} h^2 + Dh + 0 \tag{17}$$

$$D = -\frac{1}{2}\frac{\partial p}{\partial x} h \tag{18}$$

Our exact solution now becomes

$$v(y) = \frac{1}{2\mu}\frac{\partial p}{\partial x}\left(y^2 - hy\right) \tag{19}$$

when we factor our common terms.

We are now interested in finding the finite difference equation to equation 10. We find the taylor expansion of atleast degree 2 of $v(y \pm \Delta y)$ around y. Expanding the polynomial we get

$$v(y \pm \Delta y) = v(y) \pm \Delta y v'(y) + \frac{\Delta y^2}{2!} v''(y) \pm \frac{\Delta y^3}{3!} v^{(3)}(y) + \mathcal{O}(h^4) \tag{20}$$

[2]To note, as D and E are constants, the leading sign is irrelevant till the constants have been found.

If we add the two equations[3] we get from 20 we can find the finitite difference version.

(21)
$$v(y+\Delta y)+v(y-\Delta y)=v(y)+v(y)+\Delta y^2 v''(y) \qquad \text{solving for } v''(y)$$

$$v''(y)=\frac{v(y+\Delta y)-2v(y)+v(y-\Delta y)}{\Delta y^2} \tag{22}$$

If we now insert this expression into our original equation we get

$$\mu\frac{v(y+\Delta y)-2v(y)+v(y-\Delta y)}{\Delta y^2}=\frac{\partial p}{\partial x} \tag{23}$$

If we insert values for $y=0, \frac{1}{4}h, \frac{1}{2}h, \frac{3}{4}h, h$, with $\Delta y = 1/4h$, we get the following

$$\begin{aligned} v(0)&=0\\ \mu\frac{v(\frac{2}{4}h)-2v(\frac{1}{4}h)+v(0)}{\Delta y^2}&=\frac{\partial p}{\partial x}\\ \mu\frac{v(\frac{3}{4}h)-2v(\frac{2}{4}h)+v(\frac{1}{4}h)}{\Delta y^2}&=\frac{\partial p}{\partial x}\\ \mu\frac{v(\frac{4}{4}h)-2v(\frac{3}{4}h)+v(\frac{2}{4}h)}{\Delta y^2}&=\frac{\partial p}{\partial x}\\ v(h)&=0 \end{aligned}$$

Take our difference equation 22 and discretize the equation, such that $v_i \simeq v(y_i)$ and $v_{i\pm1} \simeq v(y_i \pm \Delta y)$ and $v''(y_i) \simeq f_i$, we get that

$$v''(y) \simeq f_i = \frac{v_{i+1}-2v_i+v_{i-1}}{\Delta y^2} \tag{24}$$

Where $v(0) \simeq v_0$. If we insert values for $i = 1, 2, 3, \ldots$ and ignore Δy^2 term for now we have,

$$\begin{aligned} v_2+v_0-2v_1&=f_1 \quad i=1\\ v_3+v_1-2v_2&=f_2 \quad i=2\\ v_4+v_2-2v_3&=f_3 \quad i=3\\ v_5+v_3-2v_4&=f_4 \quad i=4\\ &\vdots\\ v_n+v_{n-2}-2v_{n-1}&=f_{n-1} \quad i=n-1\\ v_{n+1}+v_{n-1}-2v_n&=f_n \quad i=n \end{aligned}$$

We recognize this as a pattern for a tridiagonal matrix. With -2 along the leading diagonal. As we have Dirichlet boundary conditions, we have fixed elements in the matrix. The resulting matrix, we will call A, will be on the following form.

$$A=\begin{bmatrix} 1 & 0 & 0 & \ldots & 0 & 0\\ 1 & -2 & 1 & 0 & \ldots & 0\\ 0 & 1 & -2 & 1 & \ldots & 0\\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots\\ 0 & \ldots & 0 & 1 & -2 & 1\\ 0 & \ldots & \ldots & \ldots & 0 & 1 \end{bmatrix} \tag{25}$$

for an nxn matrix.

[3]Choosing + or - as our sign.

If we look at equation 23 and collect the common terms on the right hand side(RHS). We can define $v''(y) = \mathbf{A}\vec{v}$ where $\mathbf{A}$ is the matrix found in 25 and $\vec{v}$ is the vector $v(y)$ for all values of $y \in [0, h]$. And set our RHS $\vec{b} = \frac{\Delta y^2 \partial p}{\mu \partial x}$

Our equation now read $\mathbf{A}\vec{v} = \vec{b}$. Writing this out for $y = 0, \frac{1}{4}h, \frac{1}{2}h, \frac{3}{4}h, h$ we get the following expression

$$\mathbf{A}\vec{v} = \vec{b}$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v(0) \\ v(\frac{1}{4}h) \\ v(\frac{2}{4}h) \\ v(\frac{3}{4}h) \\ v(h) \end{pmatrix} = \begin{pmatrix} 0 \\ \frac{\Delta y^2}{\mu} \frac{\partial p}{\partial x} \\ \frac{\Delta y^2}{\mu} \frac{\partial p}{\partial x} \\ \frac{\Delta y^2}{\mu} \frac{\partial p}{\partial x} \\ 0 \end{pmatrix} \tag{26}$$

A numerical implementation of the problem could be done in two ways. Firstly it is possible to recognize our matrix A, as a tridiagonal matrix, where we can ignore the endpoints. Doing so means that we can apply the Thomas algorithm to the problem, as solve it as a problem involving vectors and Gaussian elimination. Primarily performing a forwards and backwards substitution. As we have a pattern along our diagonals, it can be further simplified. Our implementation of this is the following

```python
def tridiag_solver_Gauss(n):
    """solve the problem using gaussian elim

    Args:
        n (int): number of points

    Returns:
        1d-array: solution to the problem
    """
    d = np.zeros(n)  # diagonal. Could be done with np.full also.
    d.fill(-2)  # fill the array
    solution = np.zeros(n)  # vector v
    d[0] = d[n-1] = 1  # Set first and last element of diag

    b = vec_b(n)  # Setup vector b

    # Forwards sub
    for i in range(2, n-1):
        d[i] = -2 - 1/d[i-1]
        b[i] = b[i] - b[i-1]/d[i-1]
    # backwards sub
    solution[n-2] = b[n-2]/d[n-2]  # End points
    for i in range(n-3, 0, -1):
        solution[i] = (b[i] - solution[i+1])/d[i]
    return solution
```

For the entire program using in *Problem 2* see page 8 and following pages. The other way is through solving the matrix equation and solving for $\vec{v}$. Doing so we need to solve $\vec{v} = \mathbf{A}^{-1}\vec{b}$, which involves a matrix inversion and matrix multiplication. This is done in the following function

```python
def matrix_solution(n):
    """solve the problem with matrix multiplication and inversion

    Args:
        n (int): number of points

    Returns:
```

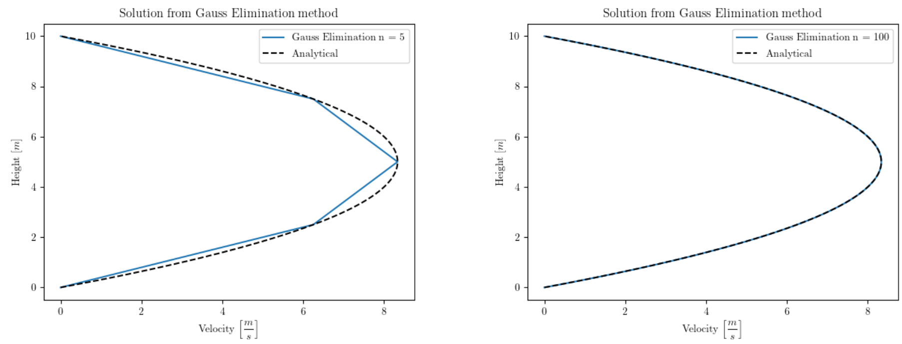

FIGURE 1. Plots using the Gaussian elimination method for $N = 5$ and $N = 100$ respectively. The analytical solution is plotted for $N = 1000$ each time.

```
        1d-array: solution to the answer
    """
    A = np.zeros([n, n])  # Matrix setup
    A[0, 0] = A[n-1, n-1] = 1
    for i in range(1, n-1):
        A[i, i+1] = 1
        A[i, i] = -2
        A[i, i-1] = 1
    A_inv = LA.inv(A)  # Inversion
    return A_inv@vec_b(n)  # Matrix multiplication
```

In both programs the vector $\vec{b}$ is setup in the following way

```
def vec_b(n):
    """Generate vector b. b[0] = b[n-1] = 0

    Args:
        n (int): number of points

    Returns:
        1d-array: returns the array with values in b
    """
    delta_y = h/(n-1)  # steplength
    b = np.full(n, dpdx / my * delta_y * delta_y)  # Setup
    b[0] = b[n-1] = 0  # endpoints
    return b
```

Lastly we want to run our program we created and compared it to the analytical solution. Doing so we can clearly see that both the Gauss Elimination and Matrix equation results in the same plots as seen in figures 1 and 2. For $N = 100$ we can see that our numerical solution seems to fit perfectly with the analytical solution. The maximum speed is as followed, found both numerically and analytically

```
The maximum speed using numerical solution for n = 5 is 8.333[m/s]
The maximum speed using the analytical solution for n = 1000 is 8.333[
    m/s]
The maximum speed using numerical solution for n = 100 is 8.332[m/s]
The maximum speed using the analytical solution for n = 1000 is 8.333[
    m/s]
```

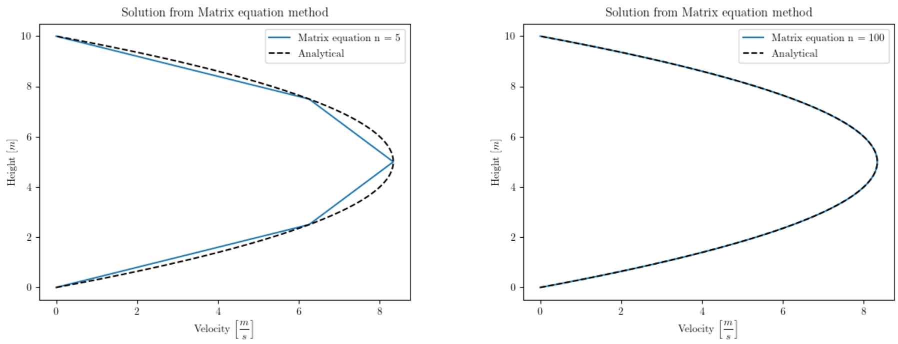

FIGURE 2. Plots using the Matrix equation method for $N = 5$ and $N = 100$ respectively. The analytical solution is plotted for $N = 1000$ each time.

3. PROBLEM 3: MORE FINITE DIFFERENCES

We want to write the FTCS(forward Euler, centered in space) of the following equation

$$\frac{\partial}{\partial t}\rho = k\frac{\partial^2}{\partial x^2}\rho \tag{27}$$

using $s = kdt/dx^2$. Using the same principle for finding the taylor expansion as we did in equation 22 to find the expansion of the second derivative, the RHS of the equation. If we take a look at the left hand side(LHS) we have

$$\frac{\partial \rho}{\partial t} = \frac{\rho(t+dt) - \rho(t)}{dt} + \mathcal{O}(dt) \tag{28}$$

$$= \frac{\rho^{n+1} - \rho^2}{dt} \tag{29}$$

Using this we can insert into our original expression and we get

$$\frac{\rho_i^{n+1} - \rho_i^n}{dt} = k\frac{\rho_{i+1}^n - 2\rho_i^n + \rho_{i-1}^n}{dx^2} \tag{30}$$

Where i references the index of the array and n is the current timestep. Using our definition of s we have

$$\rho_i^{n+1} = s(\rho_{i+1}^n - 2\rho_i^n + \rho_{i-1}^n) + \rho_i^n \tag{31}$$

$$\rho_i^{n+1} = s\rho_{i+1}^n + (1-2s)\rho_i^n + s\rho_{i-1}^n \tag{32}$$

which would be our FTCS.

We also want to find the iFTCS(implicit Euler, centered in space) version of the same equation. This is straight forward to find. The final expression is our equation looking into the future, and knowing the current value. So the method is the same as for the FTCS, so our equation then becomes

$$\rho_i^{n+1} + \rho_i^n = s\rho_{i+1}^{n+1} - 2s\rho_i^{n+1} + s\rho_{i-1}^{n+1} \tag{33}$$

Here we have no good way of solving the RHS as we don't know the value of it. We can rearrange the equation to the following

$$-s\rho_{i+1}^{n+1} + (1+2s)\rho_i^{n+1} - s\rho_{i-1}^{n+1} = \rho_i^n \tag{34}$$

We now how 3 unkowns on the LHS. Let us look back at the simpler scheme we had in problem 2. We see that our matrix equation would take the form of

$\mathbf{A}\rho^{n+1} = \rho^n + \vec{b}$. We are given that $\rho(1) = 1$ and $\rho(5) = 0$, the matrix equation, ignoring the end points, that follows is

$$\mathbf{A}\rho^{n+1} = \rho^n + \vec{b}$$

$$\begin{bmatrix} 1+2s & -s & 0 \\ -s & 1+2s & -s \\ 0 & -s & 1+2s \end{bmatrix} \vec{\rho^{n+1}} = \rho^n + \begin{bmatrix} s \\ 0 \\ 0 \end{bmatrix} \tag{35}$$

Where $\vec{b}$ is found by setting up the set of equations and moving the known terms $\rho(1) = 1$ and $\rho(5) = 0$ inserted for the the values in our implicit scheme 34.

We want to solve this equation for ρ^{n+1}, giving us $\rho^{n+1} = \mathbf{A}^{-1}(\rho^n + \vec{b})$. We would need to invert the matrix A, doing so numerically gives us

```
import sympy as sp

s = sp.Symbol('s')
A = sp.Matrix([[1+2*s, -s, 0], [-s, 1+2*s, -s], [0, -s, 1+2*s]])
A_inv = A.inv()
print("A inverse inserted for s = 0.5 gives")
sp.pprint(A_inv.evalf(subs={s: 0.5}))
"""
A inverse inserted for s = 0.5 gives
|0.535714285714286   0.142857142857143  0.0357142857142857|
|                                                          |
|0.142857142857143   0.571428571428571  0.142857142857143 |
|                                                          |
|0.0357142857142857  0.142857142857143  0.535714285714286 |
"""
```

Our equation for ρ^{n+1} now reads

$$\rho^{n+1} = \mathbf{A}^{-1} \begin{pmatrix} \rho_2^n + s \\ \rho_3^n \\ \rho_4^n \end{pmatrix} \tag{36}$$

Where $\mathbf{A}^{-1}$ is the matrix found numerically.

4. Code addition

```
import numpy as np
import matplotlib.pyplot as plt
from numpy import linalg as LA


def tridiag_solver_Gauss(n):
    """solve the problem using gaussian elim

    Args:
        n (int): number of points

    Returns:
        1d-array: solution to the problem
    """
    d = np.zeros(n)  # diagonal. Could be done with np.full also.
    d.fill(-2)  # fill the array
    solution = np.zeros(n)  # vector v
    d[0] = d[n-1] = 1  # Set first and last element of diag

    b = vec_b(n)  # Setup vector b

    # Forwards sub
    for i in range(2, n-1):
```

```
        d[i] = -2 - 1/d[i-1]
        b[i] = b[i] - b[i-1]/d[i-1]
    # backwards sub
    solution[n-2] = b[n-2]/d[n-2]  # End points
    for i in range(n-3, 0, -1):
        solution[i] = (b[i] - solution[i+1])/d[i]
    return solution


def matrix_solution(n):
    """solve the problem with matrix multiplication and inversion

    Args:
        n (int): number of points

    Returns:
        1d-array: solution to the answer
    """
    A = np.zeros([n, n])  # Matrix setup
    A[0, 0] = A[n-1, n-1] = 1
    for i in range(1, n-1):
        A[i, i+1] = 1
        A[i, i] = -2
        A[i, i-1] = 1
    A_inv = LA.inv(A)  # Inversion
    return A_inv@vec_b(n)  # Matrix multiplication


def vec_b(n):
    """Generate vector b. b[0] = b[n-1] = 0

    Args:
        n (int): number of points

    Returns:
        1d-array: returns the array with values in b
    """
    delta_y = h/(n-1)  # steplength
    b = np.full(n, dpdx / my * delta_y * delta_y)  # Setup
    b[0] = b[n-1] = 0  # endpoints
    return b


def vec_x(n):
    """set up x vector

    Args:
        n (int): number of points

    Returns:
        1d-array: vector of x-values
    """
    x = np.linspace(0, h, n)
    return x


def plotter(x, data, solver):
    """Plotting the data

    Args:
        x (1d-array): x-values
        data (1d-array): data values
        solver (functions): which solver for the data set is used
```

```
    """
    plt.figure()
    plt.plot(data, x, label=(solver + " n = " + str(ele)))
    plt.title("Solution from " + solver + " method")
    plot_analytical_solution()
    plt.xlabel(r"Velocity $\left[\displaystyle\frac{m}{s}\right]$")
    plt.ylabel(r"Height $[\displaystyle m]$")
    plt.legend()
    plt.savefig("./plot/plot_" + solver + str(ele))
    plt.title(solver)


def plot_analytical_solution():
    """Plots the analytical solution and returns the values

    Returns:
        1d-array: return values of the solution on vector form.
    """
    const = dpdx / (2*my)
    n = 1000
    x = np.linspace(0, h, n)
    solution = const * (x*x-h*x)
    plt.plot(solution, x, "k--", label="Analytical")
    return solution


if __name__ == "__main__":
    plt.rcParams.update({
        "text.usetex": True,
        "font.family": "DejaVu Sans",
        "font.sans-serif": ["Helvetica"]})

    # To solve for both N, and possibly more values if needed.
    list_of_n = (5, 100)
    # to get two plots, using either method for solving
    solver = ("Gauss Elimination", "Matrix equation")
    dpdx = -200  # Pa/m
    h = 10  # m
    my = 300  # Pa*s
    for ele in list_of_n:
        x = vec_x(ele)
        data1 = tridiag_solver_Gauss(ele)
        data2 = matrix_solution(ele)
        plotter(x, data1, solver[0])
        plotter(x, data2, solver[1])
        print(
            f"The maximum speed using numerical solution for n = {ele} is {max(data1):5.3f}[m/s]")
        print(
            f"The maximum speed using the analytical solution for n = 1000 is {max(plot_analytical_solution()):5.3f}[m/s]")
    plt.show()

"""
The maximum speed using numerical solution for n = 5 is 8.333[m/s]
The maximum speed using the analytical solution for n = 1000 is 8.333[m/s]
The maximum speed using numerical solution for n = 100 is 8.332[m/s]
The maximum speed using the analytical solution for n = 1000 is 8.333[m/s]

"""
```