Problem Set 1 - Villostas

Chynelle Ziarah Villostas

September 2025

1 Problem 4.17

We are given:

$$A = \begin{bmatrix} 4 & 4 & 8 & 4 \\ 4 & 5 & 3 & 7 \\ 8 & 3 & 9 & 9 \\ 4 & 7 & 9 & 5 \end{bmatrix} \tag{1}$$

$$b = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix} \tag{2}$$

Performing Gaussian Elimination on this, we find the the augmented matrix:

$$\left[\begin{array}{cccc|c} 4 & 4 & 8 & 4 & 1 \\ 0 & 1 & -5 & 3 & 1 \\ 0 & 0 & -32 & 16 & 6 \\ 0 & 0 & 0 & 0 & 3 \end{array}\right] \tag{3}$$

So we know that the matrix is singular, meaning it has no solution.

If we made a code for this in python without pivoting, it would have an error because obviously it will divide by zero. If you didn't do the math beforehand, and based only on the code, you won't be sure whether the matrix is actually singular or if the rows can be swapped so that there isn't a zero in the diagonal. On the other hand, if we made a code with pivoting, then we will be able to see that the matrix is truly singular because no swapping of any row could make a non-zero element in the diagonal.

Also, replacing the zero with a small quantity like 10^{-20} won't really work. Although the code technically would run, it would give wrong results. This is because you would be dividing by a really small quantity too, and thus could get rounding errors, etc.

2 Problem 4.18

Note: The .ipynb file is also uploaded in this repository

```
import numpy as np

def jacobi(n, tolerance, iterations):
    x = np.zeros(n)
    x_new = np.zeros(n)

    #this part is basically doing 4.242
    for j in range(iterations):
        x_new[0] = (1 + x[1] + x[-1])*(1/4.0)
        for i in range(1, n-1):
            x_new[i] = (1 + x[i-1] + x[(i+1) % n])*(1/4.0)
        x_new[-1] = (1 + x[0] + x[-2])*(1/4.0)

        # check convergence
        if np.linalg.norm(x_new - x, ord=np.inf) < tolerance:
            return x_new
        x[:] = x_new

    return x_new

#making A and b
def matrix(n):
    A = np.zeros((n, n))
    b = np.ones(n)

    #diagonal elements:
    for i in range(n):
        A[i, i] = 4
        if i > 0:
            A[i, i-1] = -1
        if i < n-1:
            A[i, i+1] = -1

    #adding the elements that aren't in the tridiagonal
    A[0, -1] = -1
    A[-1, 0] = -1

    return A, b

#for n = 10
A, b = matrix(10)
print("Using numpy.linalg.solve with n= 10:", np.linalg.solve(A, b))
```

```
print("Using Jacobi Iteration with n= 10:", jacobi(10, 10e-5,50))

#for n = 20
C, d = matrix(20)
print("Using numpy.linalg.solve with n= 20:", np.linalg.solve(C, d))
print("Using Jacobi Iteration with n= 20:", jacobi(20, 10e-5,50))
```