MATH 417 502
Homework 3

Keegan Smith

September 8, 2024

Problem 1

a.) Our system of equations can be re-written as:

$$\begin{aligned} x_1 + 2x_2 + 3x_3 - \lambda x_1 &= 0 \\ 4x_1 + 5x_2 + 6x_3 - \lambda x_2 &= 0 \\ 7x_1 + 8x_2 + 10x_3 - \lambda x_3 &= 0 \\ x_1^2 + x_2^2 + x_3^2 - 1 &= 0 \end{aligned}$$

The jacobian of this system is:

$$\begin{bmatrix} 1-\lambda & 2 & 3 & -\lambda x_1 \\ 4 & 5-\lambda & 6 & -x_2 \\ 7 & 8 & 10-\lambda & -x_3 \\ 2x_1 & 2x_2 & 2x_3 & 0 \end{bmatrix}$$

Thus the Newton iteration looks like:

$$x^{n+1} = x^n - \begin{bmatrix} 1-\lambda & 2 & 3 & -\lambda x_1^n \\ 4 & 5-\lambda & 6 & -x_2^n \\ 7 & 8 & 10-\lambda & -x_3^n \\ 2x_1^n & 2x_2^n & 2x_3^n & 0 \end{bmatrix}^{-1} \begin{bmatrix} x_1^n + 2x_2^n + 3x_3^n - \lambda x_1^n \\ 4x_1^n + 5x_2^n + 6x_3^n - \lambda x_2^n \\ 7x_1^n + 8x_2^n + 10x_3^n - \lambda x_3^n \\ (x_1^n)^2 + (x_2^n)^2 + (x_3^n)^2 - 1 \end{bmatrix}$$

Essentially we will pick an initial vector x_0 and plug this value into our equation above to get the next vector x^{n+1}. We continuously do this until we are pretty close to 0. We repeat the process for multiple x_0 until we find all of our solutions.

b.) My program found the following solutions to the system of equations:
Solution 0: [-0.22464024 -0.59734221 -1.06500369 16.65147157]
Solution 1: [1.04516341 -0.32819325 -0.41221205 -1.18226152]

Solution 2: [0.39884723 -1.03663168 0.58667158 0.12967112]
where the first 3 values are eigenvectors x and the last value is the corresponding eigenvalue λ. Thus we have the eigenvectors:

$$\begin{bmatrix} -0.22464024 \\ -0.59734221 \\ -1.06500369 \end{bmatrix}, \begin{bmatrix} 1.04516341 \\ -0.32819325 \\ -0.41221205 \end{bmatrix} \begin{bmatrix} 0.39884723 \\ -1.03663168 \\ 0.58667158 \end{bmatrix}$$

And their respective eigenvalues:
16.6515, -1.1823, 0.1297
my code to accomplish this is below:

```
import numpy as np
import random
from concurrent.futures import ThreadPoolExecutor,
    as_completed
import threading
import copy
NUM_ITER = 500
global_solutions = []
lock = threading.Lock()
def compute_jacobian_matrix(x_vector):
    my_jacobian = np.array([[0, 2, 3, 0], [4, 0, 6, 0],
        [7, 8, 0, 0], [0, 0 , 0, 0]])

    my_jacobian[0][0] = 1 - x_vector[3][0]
    my_jacobian[0][3] = -x_vector[0][0]
    my_jacobian[1][1] = 5 - x_vector[3][0]
    my_jacobian[1][3] = -x_vector[1][0]
    my_jacobian[2][2] = 10 - x_vector[3][0]
    my_jacobian[2][3] = -x_vector[2][0]
    my_jacobian[3][0] = 2 * x_vector[0][0]
    my_jacobian[3][1] = 2 * x_vector[1][0]
    my_jacobian[3][2] = 2 * x_vector[2][0]
    return my_jacobian
def compute_f(x_vector):
    my_f = np.array([[0], [0], [0], [0]])

    my_f[0][0] = x_vector[0][0] + 2 * x_vector[1][0] + 3
         * x_vector[2][0] - x_vector[3][0] * x_vector
        [0][0]
    my_f[1][0] = 4 * x_vector[0][0] + 5 * x_vector[1][0]
         + 6 * x_vector[2][0] - x_vector[3][0] *
        x_vector[1][0]
    my_f[2][0] = 7 * x_vector[0][0] + 8 * x_vector[1][0]
         + 10 * x_vector[2][0] - x_vector[3][0] *
        x_vector[2][0]
    my_f[3][0] = x_vector[0][0]**2 + x_vector[1][0]**2 +
         x_vector[2][0]**2 - 1
```

```
    return my_f
def iterate(initial_x):
    for i in range(0, NUM_ITER):
        jacobian = compute_jacobian_matrix(initial_x)
        my_f = compute_f(initial_x)
        initial_x = initial_x + np.linalg.solve(jacobian
            , -my_f)
    return initial_x
def perform_iteration(i, start, end):
    #print("got here")
    j = random.uniform(start, end)
    k = random.uniform(start, end)
    l = random.uniform(start, end)
    m = random.uniform(start, end)
    try:
        result = tuple(iterate(np.array([[j], [k], [l],
            [m]])).flatten())

        return result
    except Exception as e:
        return None
if __name__ == "__main__":
    start = -10
    end = 10
    num_attempts = 10000
    with ThreadPoolExecutor() as executor:
        futures = [executor.submit(perform_iteration, i,
             start, end) for i in range(num_attempts)]
        for future in as_completed(futures):
            result = future.result()
            if(result):
                with lock:
                    global_solutions.append(result)
    while(True):
        result = []
        for i in range(0, 3):
            result.append(np.array(global_solutions[
                random.randint(0, len(global_solutions))
                ]))
        result = np.array(result)
        eigenvectors = []
        for i in range(0, len(result)):
            eigenvectors.append(result[i][:3])
        determinant = np.linalg.det(eigenvectors)
        if(not (determinant <= 10**(-2) and determinant
            >= -10**(-2))):
            for i in range(0, len(result)):
                print(f"Solution {i}: ", result[i])
            print("determinant was: ", determinant)
            break;
```

```
73          print("determinant was approx. 0, trying again")
```