Alex Merino

MTH 4311

Dr. Pei Liu

Program 1

**Part 1:**

The given equation for the satellites needed to be reorganized into functions where each are equal to zero (shown below).

$$\sqrt{(X-A_i)^2+(y-B_i)^2+(z-C_i)^2} = c(t_i-d)$$

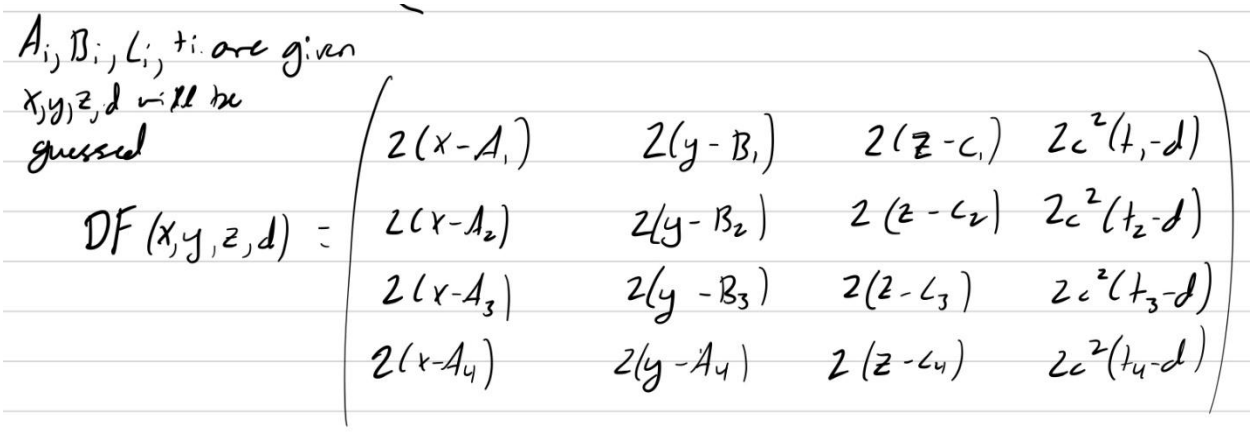
$$(X-A_i)^2+(y-B_i)^2+(z-C_i)^2 = (c(t_i-d))^2$$

$$(X-A_i)^2+(y-B_i)^2+(z-C_i)^2-(c(t_i-d))^2 = \emptyset$$

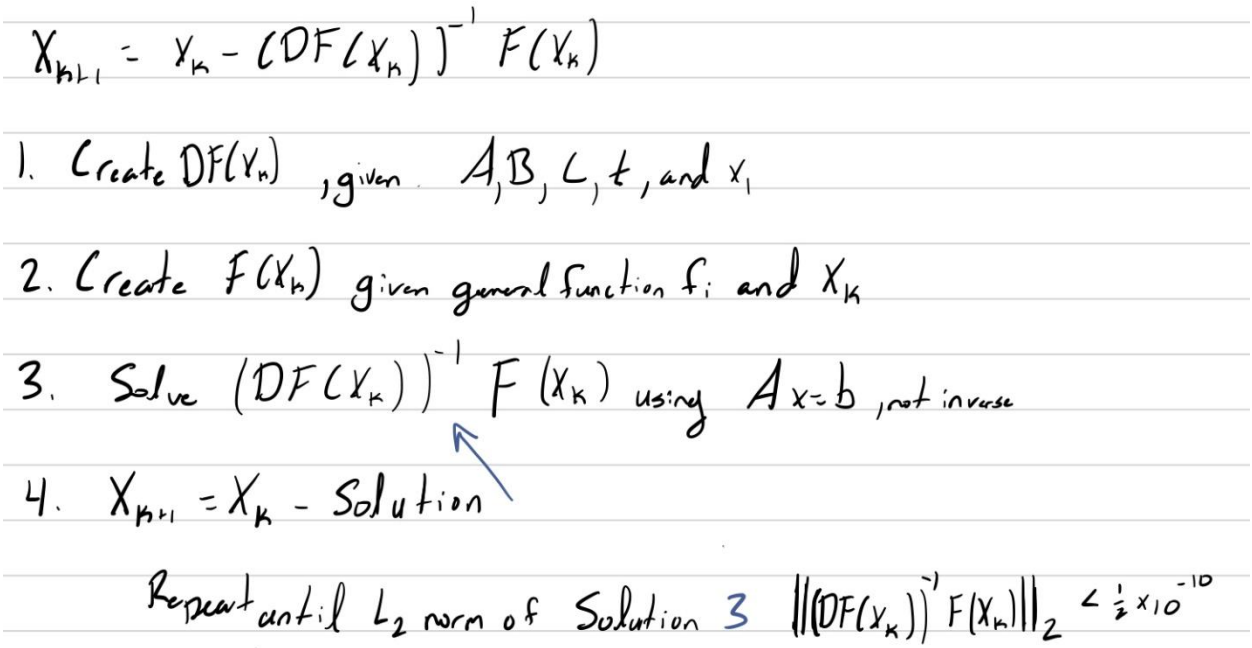
Using this formula for the four equations, we get the system of equations:

$$\begin{cases} (X-A_1)^2+(y-B_1)^2+(z-C_1)^2-(c(t_1-d))^2 = \emptyset \\ (X-A_2)^2+(y-B_2)^2+(z-C_2)^2-(c(t_2-d))^2 = \emptyset \\ (X-A_3)^2+(y-B_3)^2+(z-C_3)^2-(c(t_3-d))^2 = \emptyset \\ (X-A_4)^2+(y-B_4)^2+(z-C_4)^2-(c(t_4-d))^2 = \emptyset \end{cases}$$

Using these as the system of equations for the problem, we next need to create the Jacobian matrix, DF, which is shown below:




Since the vector $F(X_k)$ is trivial, it is not shown worked out. Now that we have all of the components necessary to run the Multivariable Newton's formula, we need to code it. The program flow for this is shown below:




Since there was no stopping condition, it was arbitrarily picked to have an $L_2$ norm of below $0.5 * 10^{-10}$. The code for these problems are in the *multiNewt.py* file and is documented.

The *multiNewtMethod* method takes in two parameters, the position of the satellites, and the initial guess. The method then loops; inside the loop, a DF matrix is created from the *createDF* method, a F vector is created from the *createF* method, the inverse matrix problem is solved using *scipy.inalg.solve* and then subtracted from the initial guess. This loop continues until the $L_2$ norm of the inverse matrix problem is less than $0.5 * 10^{-10}$.

**Part 2:**

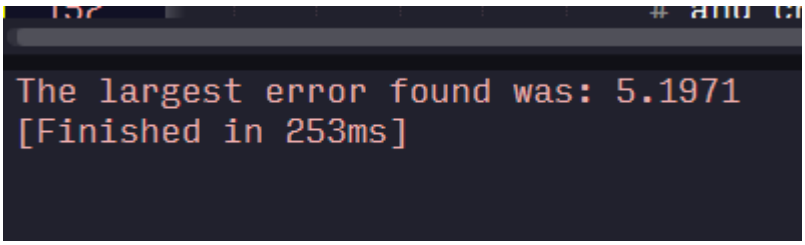
Using the inputs given for the satellites and the initial guess, I ran the inputs through the code from Part 1 and got the formatted output of:

```
The receiver's position based on the four satellites are:
x: -41.7727
y: -16.7892
z: 6370.0596
Time: 0.0675 seconds
[Finished in 244ms]
```

Assuming that the radius of the earth is 6,370 km, this point falls directly on the surface of the earth, which means that this is the correct output of the system of equations.

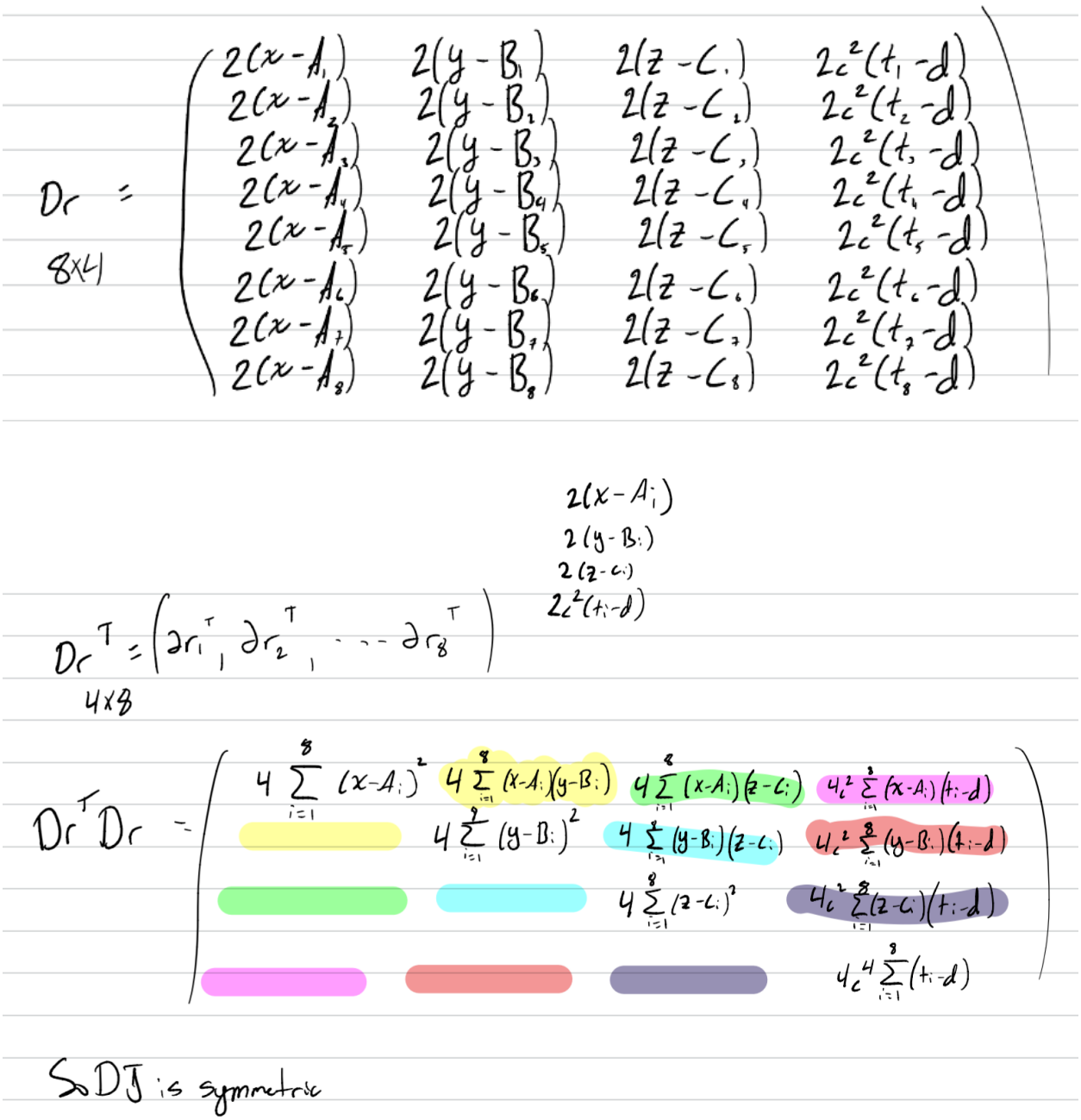
**Part 3:**

Since each satellite could have a difference of +/- $10^{-8}$, there were $2^4$ trial runs needed. The base is 2 because there are only two option and the power is four because there are four satellites. So after 16 trial runs, the largest error found was 5.1971 km, which when considering the accuracy of real GPS technology, is a very large range. This is because even with four satellites, the accuracy of the location is still not perfect.




**Part 4:**

For the Gauss-Newton method, I decided to write it in a different file called *gaussNewt.py*. Multivariable Newton and Gauss-Newton are similar in the way that they are both based off Newton's iterative method so the structure of the code was portable, but the major changes occurred in the *createDr* function, which was the former *createDf* where the Df matrix is being created but it is through the matrix multiplication of $Dr^{T}Dr$. The matrix was calculated by hand, with the work shown below.




When we take the transpose of the partial derivative matrix, Dr, and multiply against its transpose we get the summation of the partials of each variable (x, y, z, d) for each satellite squared on the diagonal. Then each partial multiplied by each other in the upper triangle of the matrix. The lower triangle of the matrix is symmetric to the upper, so the values were only calculated once and copied to the lower half.

**Part 5:**

When running the program to find the true position, which has already been calculated to be (0,0,6670,0), I get the output of:

```
Part 5a:
The receiver's position based on the eight satellites are:
x: 0.0000
y: -0.0000
z: 6670.0000
Time: 0.0000 seconds
```

My inputs for the method are as follows:

```
receiver = [100, -200, 6000, 0]

sats = [[-2019, 2279, 26395, 0.0665747329503],
        [-17706, -15990, 11695, 0.0813262620503],
        [-6503, 13883, 21701, 0.0716159594731],
        [17200, 2763, 20062, 0.0732945778521],
        [7161, 8292, 24205, 0.0689690379518],
        [10690, -21329, 11695, 0.081327616831],
        [-25706, 0, 6722, 0.0857461617486],
        [2559, -1659, 26394, 0.0665739901458]]

# Run Gauss Newton
position = gaussNewt(sats, receiver)
```

If I were to change the receiver position, I would still end up at the correct output. After finding the true position, I tried all $2^8$=256 possibilities of time deltas and got the output below:

```
Part 5b:
The largest error found was: 2.8802
[Finished in 1.3s]
```

This error of 2.88 km is about half of the smallest error that was found with the Multivariable-Newton method, and this can be attributed to having eight satellites and a fixed true position that will always be found.