Project 1

Amanda Lind Davíðsdóttir <amanda19@ru.is>

Bjarmi Valentino B Del Negro <bjarmib19@ru.is>

Eric Ruge <eric22@ru.is>

Fanney Einarsdóttir <fanneye22@ru.is>

T-406-TOLU

RU Science and Engineering

November 27, 2022

Introduction

The GPS system consists of 24 satellites that carry atomic clocks. The satellites simultaneously send a signal to a receiver that locates itself with the given information, at the time t_i. The signal is sent at the speed of light

$$c \ = \ 299792.458 \ km/s \tag{1}$$

The distance between the satellite and the receiver is c multiplied by t_i. The receiver is located on a sphere that is centered at satellite number i and with the radius ct_i.

Using three satellites has been shown to have a serious problem with its analysis since the regular phone does not have very precise timekeeping. Because of that, a small-time measurement error of Δt $\simeq 10^{-6}$s corresponds to a distance error of Δr = cΔt $\simeq$ 3 km, which is unacceptable. Therefore, four satellites will be used in this assignment and four variables with the positions (x, y, z) and d, the difference between the satellite clock and the receiver clock. The i-th satellite is set at location (A_i, B_i, C_i) for i = 1..4 and therefore the following equations can be used for x, y, z, d :

$$f_1(x, y, z, d) = (x - A_1)^2 + (y - B_1)^2 + (z - C_1)^2 - c^2(t_1 - d)^2 = 0 \tag{2}$$

$$f_2(x, y, z, d) = (x - A_2)^2 + (y - B_2)^2 + (z - C_2)^2 - c^2(t_2 - d)^2 = 0 \tag{3}$$

$$f_3(x, y, z, d) = (x - A_3)^2 + (y - B_3)^2 + (z - C_3)^2 - c^2(t_3 - d)^2 = 0 \tag{4}$$

$$f_4(x, y, z, d) = (x - A_4)^2 + (y - B_4)^2 + (z - C_4)^2 - c^2(t_4 - d)^2 = 0 \tag{5}$$

The system has two solutions, with one that is realistic, but it can also be simplified further $F(x, y, z, d)$ = 0 where F : $\mathbb{R}^4 \rightarrow \mathbb{R}^4$ is defined by

$$F = \begin{pmatrix} f_1(x, y, z, d) \\ f_2(x, y, z, d) \\ f_3(x, y, z, d) \\ f_4(x, y, z, d) \end{pmatrix} \tag{6}$$

The Jacobi matrix DF can be computed by taking the partial derivative of the vector function F to transform the coordinates.

$$DF = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}, \frac{\partial f_1}{\partial y_1}, \frac{\partial f_1}{\partial z_1}, \frac{\partial f_1}{\partial d_1} \\ \frac{\partial f_2}{\partial x_1}, \frac{\partial f_2}{\partial y_1}, \frac{\partial f_2}{\partial z_1}, \frac{\partial f_2}{\partial d_1} \\ \frac{\partial f_3}{\partial x_1}, \frac{\partial f_3}{\partial y_1}, \frac{\partial f_3}{\partial z_1}, \frac{\partial f_3}{\partial d_1} \\ \frac{\partial f_4}{\partial x_1}, \frac{\partial f_4}{\partial y_1}, \frac{\partial f_4}{\partial z_1}, \frac{\partial f_4}{\partial d_1} \end{pmatrix} \tag{7}$$

Newton's method is a root-finding algorithm. It can be explained by choosing an initial value of x_0 for the root r and using a linear approach on the function f near the x_0. The equation to find x is in the following equation

$$x_i + 1 = x_i - \frac{f(x_i)}{f'(x_i)} \tag{8}$$

Matlab was chosen to solve this project since it is a strong tool for working with matrices and vectors.

Problem 1

To be able to use the multidimensional Newton method, a vector F was set up from the given equations 2, 3, 4 and 5

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

and the Jacobi matrix (which is called DF) was calculated from the vector F with equation 7

$$DF = \begin{bmatrix} 2x-2A_1 & 2y-2B_1 & 2z-2C_1 & 2t_1c^2d \\ 2x-2A_2 & 2y-2B_2 & 2z-2C_2 & 2t_2c^2d \\ 2x-2A_3 & 2y-2B_3 & 2z-2C_3 & 2t_3c^2d \\ 2x-2A_4 & 2y-2B_4 & 2z-2C_4 & 2t_4c^2d \end{bmatrix}$$

The following figure shows the code written to define the function F which contains the system of equations. One line in vector F is created in each iteration of the for-loop.

```
function F = Ffunc(A,B,C,c,t,pos)
    for i = 1:4
       F(i,1) = (pos(1)-A(i))^2 + (pos(2)-B(i))^2 + (pos(3)-C(i))^2 - c^2*(t(i)-pos(4))^2;
    end
end
```

Figure 1: Function to set up the F vector in Matlab

The next step was to create a function that defines the Jacobi matrix. The for loop iterates through each line, four in total. Inside the for-loop the calculation is divided up by columns since they are calculated differently.

```
function DF = jacobifunc(A, B, C, c, t,pos,n)
    for i = 1:n
        DF(i,1) = 2*(pos(1) - A(i));
        DF(i,2) = 2*(pos(2) - B(i));
        DF(i,3) = 2*(pos(3) - C(i));
        DF(i,4) = 2*t(i)*c^2 - 2*c^2*pos(4);
    end
end
```

Figure 2: Shows the function for the Jacobi matrix

In figure 3 the main Matlab code can be seen

```
%% Main code Problem 1
clear all; close all;clc;

x0 = [0;0;6370;0]; % starting point is north pole
% A,B,C,are the satellite coordinates and t %is the time of sending
A = [15600;18760;17610;19170];
B = [7540; 2750; 14630; 610];
C = [20140; 18610; 13480; 18390];
t = [0.07074; 0.07220; 0.07690; 0.07242];
c = 299792.458; %speed of light
tol = 10^(-3); %so we have the error in meters instead of kilometers
n=4; % nr. of satellites

x = newtonmult(x0,tol,A,B,C,t,c,n); % x is a vector that contains x,y,z and d
```

Figure 3: Shows the main Matlab code, problem 1

where all variables $(x_0, A, B, C, t, c, tol)$ are assigned. Table 1 is used to define A, B, C and t in figure 3.

Table 1

i	A_i	B_i	C_i	t_i
1	156000	7540	20140	0.07074
2	18760	2750	18610	0.07220
3	17610	14630	13480	0.07690
4	19170	610	18390	0.07242

Where A_i, B_i, C_i are the coordinates of satellite nr. i and t_i is the time of sending.

Then x_0, is the initial position vector

$$x_0 = (x_0, y_0, z_0, d_0) = (0, 0, 6370, 0)$$

which is the initial location of the receiver which is in this case at the North pole. The value c, which is the speed of light is shown in equation 1 and lastly the tol value which is the tolerance of the system. It is important to consider what type of system is being solved. In this case, this is a GPS system, where the locations are presented in km. Therefore a tolerance of

$$tol = 1 * 10^{-3}$$

was chosen, which means that the location of the receiver has an accuracy of up to one meter.

Since the functions and all variables are set up, the multidimensional Newton method can be used to solve the coordinates (x, y, z) of the receiver and d, which is the difference between the satellite and receiver clock.

To do that the newtonmult function was used

```
function x = newtonmult(x0,tol,A,B,C,t,c)
x=x0;
oldx=x0+2*tol;
    while norm(x-oldx,inf)>tol
       oldx=x;
       s=-jacobifunc(A,B,C,c,t,x)\Ffunc(A,B,C,c,t,x);
       x=x+s;
    end
end
```

Figure 4: Shows the Newtons method function

In this function, the variable x is given the initial value x_0 (which is the initial position vector). Then another value, $oldx$, is found by using the tolerance. A while loop is then used to iterate until the difference between the current value of x and the previous value of x is less than the tolerance specified. In each iteration in the while loop, the current value becomes closer and closer to the true value, therefore the difference between the old value and the current value decreases in each iteration. When that is obtained the iteration stops since an acceptable value for the location of the receiver has been obtained. The function returns the coordinates of the receiver and the difference between the satellite and the receiver clock. The results are shown in the following table

Table 2

x	**y**	**z**	**d**
-41.772710	-16.789194	6370.059559	$-3.201566 * 10^{-3}$

Assumptions:
For the rest of the assignment (even in the free-choice question), it is assumed that the receiver is located at the north pole that is:

$$x = y = 0 \qquad z = 6370$$

Problem 2

In this problem, the locations of the satellites were described by using spherical coordinates (centered at the center of the Earth). These coordinates are very practical, they specify three numbers: radial distance, polar angles, and azimuth angle, as shown in figure 5.

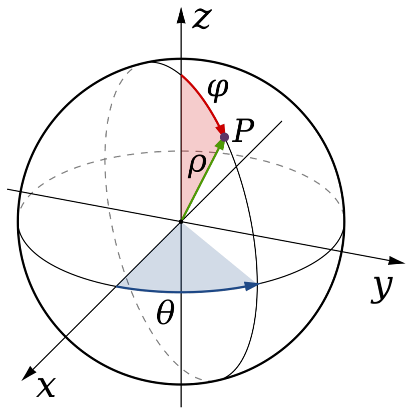


Figure 5: Spherical coordinates [1]

The location for i-th satellite is calculated according to the following equations

$$A_i = \rho sin(\phi_i)cos(\theta_i) \tag{9}$$

$$B_i = \rho sin(\phi_i)sin(\theta_i) \tag{10}$$

$$C_i = \rho cos(\phi_i) \tag{11}$$

where ρ = 26570 km is the constant altitude of the satellites, 0 $\leq \phi_i \leq \pi/2$ is the angle from the positive z-axis and 0 $\leq \theta_i \leq$ 2π is the polar angle. Four different values were chosen for both phi and theta angles, ranging between the allowed values, and can bee seen in table 3.

Table 3

Satellites	ϕ $angle$	θ $angle$
Nr. 1	$\pi/8$	$\pi/2$
Nr. 2	$\pi/6$	π
Nr. 3	$\pi/4$	$3\pi/2$
Nr. 4	$\pi/2$	2π

In figure 6 the main Matlab code can be seen

The ABC-coordinates for the four satellites, the distance from the North Pole, and the time t were then calculated by using the function distance-and-time.m which takes in ρ, phi angles, theta angles, and speed of light, as shown in figure 7.

The ABC-coordinates were calculated by using equations 9, 10, 11 and the results can be seen in table 4

```
%% Main code Problem 2

P=26570; %km
phi=[pi/8, pi/6, pi/4, pi/2];
theta = [pi/2, pi, 3*pi/2, 2*pi];
n=4;

[A B C R t] = distance_and_time(P,phi,theta,c,n); % TO GET COORDINATES OF THE SATELLITE (A, B, C) OR
    ↪  THE DISTANCE (R) OR THE TIME (t)
```

Figure 6: Shows the main Matlab code, problem 2

```
function [A B C R t] = distance_and_time(P, phi, theta, c,n)
    for i = 1:n
        A(i) = P*sin(phi(i))*cos(theta(i));
        B(i) = P*sin(phi(i))*sin(theta(i));
        C(i) = P*cos(phi(i));
    end

    for i = 1:n
        R(i) = sqrt((A(i))^2 + (B(i))^2 + (C(i)-6370)^2);
        t(i) = R(i)/c;
    end
end
```

Figure 7: Shows the distance and time code

Table 4

Satellites	A	B	C
Nr. 1	6.2260e-13	1.0168e+04	2.4547e+04
Nr. 2	-1.3285e+04	1.6269e-12	2.3010e+04
Nr. 3	-3.4513e-12	1.8788e+04	1.8788e+04
Nr. 4	2.6570e+04	-6.5078e-12	1.6269e-12

The distance from the north pole was calculated by the ISO convention of the coordinates. Next, the time, t_i was determined by dividing the distance of the satellite by the speed of light, the results can be seen in table 5.

Table 5

Satellites	Distance from north pole [km]	Time [sec]
Nr. 1	20828.031958	0.069475
Nr. 2	21292.971657	0.071026
Nr. 3	22520.765568	0.075121
Nr. 4	27322.917121	0.091139

Problem 3

A small error of 10^{-8} occurs when the angles ϕ and θ are measured. The effect on the computed receiver's position needs to be computed. The methods used in problem 1 and 2 are combined in order to calculate that. The receiver's position is known to be exactly at (0, 0, 6370) but the actual position of the satellites is assumed at:

Table 6: Actual position of the satellite

i	ϕ	θ
1	$\pi/8$	$-\pi/4$
2	$\pi/6$	$\pi/2$
3	$3\pi/8$	$2\pi/3$
4	$\pi/4$	$\pi/6$

Whereas the satellite is sending false data of:

Table 7: The false data of the satellites

i	ϕ	θ
1	$\pi/8 + 10^{-8}$	$-\pi/4$
2	$\pi/6 + 10^{-8}$	$\pi/2$
3	$3\pi/8 - 10^{-8}$	$2\pi/3$
4	$\pi/4 - 10^{-8}$	$\pi/6$

Utilizing the same approach as problem 2 and using the same distance-and-time.m code seen in figure 7, but this time using the ϕ and θ positions given in Table 6, resolves in t, the time of sending the signal at the correct location. The corresponding code is shown in figure 8.

```
%% Main code Problem 3
clear all; close all; clc;
c = 299792.458; %speed of light
P=26570; %km
x0 = [0;0;6370;0]; % starting point is north pole
tol2 = 10^(-9);
phi=[pi/8, pi/6, 3*pi/8, pi/4];
theta=[-pi/4, pi/2, 2*pi/3, pi/6];
n=4;

[A B C R t] = distance_and_time(P,phi,theta,c,n); %used to calculate t

phi2=[pi/8+10^(-8), pi/6+10^(-8), 3*pi/8-10^(-8), pi/4-10^(-8)];
[A2 B2 C2 R2 t2] = distance_and_time(P,phi2,theta,c,n); %used to calculate A, B, and C at the
    ↪ slightly wrong data

pos_new = newtonmult(x0,tol2,A2,B2,C2,t,c,n); %location with some error

total_err = norm(pos_new-x0);
```

Figure 8: Shows the main Matlab code, problem 3

The previous step is repeated but this time with slightly wrong data from Table 7 to receive A_2, B_2, and C_2, the positions of the satellites calculated from the false data. In other words, the correct locations of the satellites were used to obtain the time, while the wrong data was used to obtain the ABC-coordinates.

The time calculated from the correct locations is then used with the wrong positions of the satellites to calculate the computed receiver's position with the Newton method, the same approach as in problem 1 and the newtonmult function, which was shown in figure 4.

The total error is found by comparing the new position with the actual position of the north pole and calculating the Euclidean distance between the two points with the norm function of Matlab. Geometrically, the distance between the points is equivalent to the magnitude of the vector that extends from one point to the other.

When the small error of 10^{-8} occurs at the angles ϕ and θ when they are measured, the total error on the computed receiver's position is as follows:

Total error [km]
2.2050e-04

Problem 4

In this problem, the first half of problem 3 was repeated where the time was calculated from the correct locations of the satellites using the distance-and-time.m function shown in figure 7. The modification in this problem compared to the previous one is that this time the errors of the four ϕ angles in the false data could either be $+10^{-8}$ or -10^{-8} with a total of 16 possible outcomes. Four for-loops were used, one for each nr. of satellite, to generate these 16 different combinations of the plus and minus signs of the errors and therefore obtaining 16 different errors for the given angles. Figure 9 shows the layout of the for-loops.

```
[A B C R t] = distance_and_time(P,phi,theta,c,n);
max_err = 0; sign_1=0; sign_2=0; sign_3=0; sign_4=0;
counter = 1;
for i = [-1 1]
    for j = [-1 1]
        for k = [-1 1]
            for h = [-1 1]
                phi2(counter,:)=[pi/8+(i)*10^(-8), pi/6+(j)*10^(-8), 3*pi/8+(k)*10^(-8), pi/4+(h)*10^(-8)];
                [A2 B2 C2 R2 t2] = distance_and_time(P,phi2(counter,:),theta,c,n);

                pos_new = newtonmult(x0,tol2,A2,B2,C2,t,c,n); %location with some error

                total_err(counter) = norm(pos_new-x0);
                if total_err(counter) > max_err
                    max_err = total_err(counter);
                    sign_1=i;
                    sign_2=j;
                    sign_3=k;
                    sign_4=h;
                end
                counter = counter + 1;
            end
        end
    end
end
```

Figure 9: Shows the layout of the for-loops, where -1 and 1 represent - and +

The maximum error of the computed receiver's position occurred when the errors of the ϕ angles had the sign combination of (- + - -) as shown in table 8.

Table 8: The sign combination that gives the maximum error

i	ϕ	θ
1	$\pi/8 - 10^{-8}$	$-\pi/4$
2	$\pi/6 + 10^{-8}$	$\pi/2$
3	$3\pi/8 - 10^{-8}$	$2\pi/3$
4	$\pi/4 - 10^{-8}$	$\pi/6$

The maximum error of the computed receiver's position was therefore

Maximum value for the error [km]
2.3798e-04

Problem 5

Up to this point, the assignment has been using data from four different satellites resulting in a precise location of the receiver. The motivation of this problem is to investigate the effect on the error of the computed receiver's position when the satellites are tightly grouped in space. It was decided to investigate what happens when two satellites are located as close together as possible. This is a realistic scenario since this happens when satellites overlap in real life. Therefore it is very interesting to analyze how this affects the error of the computed receiver's position.

The same positions of the four satellites as in problem 3 were used but in this case, satellite nr. 4 was aligned with satellite nr. 1 to see the change in results. Tables 9 and 10 show the input variables.

Table 9: Actual position of the satellite when two satellites align

i	ϕ	θ
1	$\pi/8.00000000000011$	$-\pi/4.00000000000011$
2	$\pi/6$	$\pi/2$
3	$3\pi/8$	$2\pi/3$
4	$\pi/8.00000000000012$	$-\pi/4.00000000000012$

Table 10: The false data of the satellites when two satellites align

i	ϕ	θ
1	$\pi/8.00000000000011 + 10^{-8}$	$-\pi/4.00000000000011$
2	$\pi/6 + 10^{-8}$	$\pi/2$
3	$3\pi/8 - 10^{-8}$	$2\pi/3$
4	$\pi/8.00000000000012 + 10^{-8}$	$-\pi/4.00000000000012$

When deciding the input values in this problem, a little experiment had to be done in order to know what would give a good explanation. When the angles for satellite nr. 4 were set to the exact same location as satellite nr. 1 the solution gave a NaN answer which makes sense since Matlab was not able to calculate that matrix. Instead, the experiment changed to how close the two satellites could align. The final outcome came down to the number with a difference of 0.00000000000001 as can be seen in tables 9 and 10.

The results are shown in table 11 next to the result from problem 3 for comparison. By having two out of the four satellites grouped together the total error increased more than tenfold. Now the GPS system only has three interpretable data sets to work with when computing the location of the receiver since satellites nr. 1 and nr. 4 are giving the same information compared to having four different datasets in problem 3, resulting in less accurate results.

Table 11: Total error of computed receiver's position

	Total error [km]
Problem 3	2.2050e-04
Problem 5	7.026e+03

Problem 6

In problem 6, question 4 was repeated, with a modification that the position of the satellites is randomly set in 100 ways using a for-loop (line 138 - 163) in section H. For each set of position of the satellites there were 16 different total errors due to the 16 possible combinations of the plus and minus signs of the errors.

For each set of position of the satellites the 16 different errors were calculated. The maximum error of those 16 was interpreted as the most realistic value for this set of position and stored in a vector that eventually had 100 elements, one error for each set of the satellite position.

This vector then contained 100 different errors for 100 different set of positions of these satellites. The distribution of those errors is shown in the following histogram.

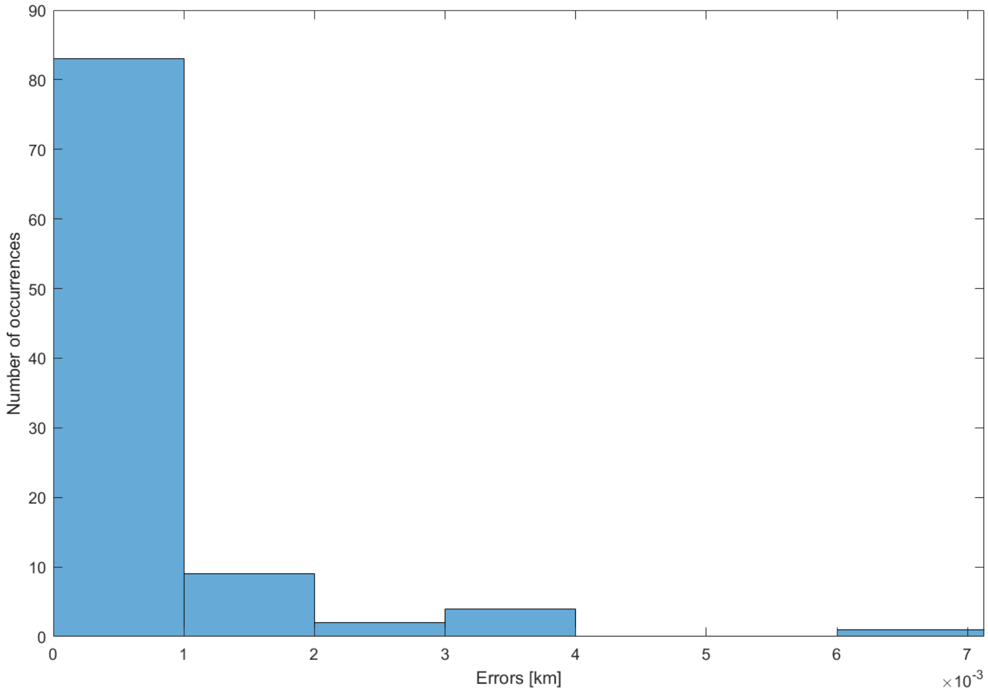


Figure 10: Distribution of the measured errors with respect to the different positions of the satellites

As shown in the histogram, one set of position of the satellites resulted in the maximum error, the column on the far right in the histogram. Other positions resulted in some average value of the errors and the minimum. Those error values are shown in table 12.

Table 12: Measuring errors when the satellites are randomly positioned

	Measuring errors [km]
Minimum error	1.0557e-04
Average error	6.8913e-04
Maximum error	7.1270e-03

The position of the satellites that resulted in the maximum error was the following

Table 13: Location of the four satellites when the **maximum error** occurred

Satellites	A [km]	B [km]	C [km]
Nr. 1	8984.69	23833.33	7563.90
Nr. 2	-12081.92	5821.24	22936.98
Nr. 3	3024.53	-20161.92	17038.60
Nr. 4	10701.17	23259.24	7103.36

The position of the satellites that resulted in the minimum error was the following

Table 14: Location of the four satellites when the **minimum error** occurred

Satellites	A [km]	B [km]	C [km]
Nr. 1	9283.85	24626.88	3645.79
Nr. 2	-20962.04	10099.82	12827.77
Nr. 3	3806.14	-25372.25	6908.48
Nr. 4	853.63	1855.37	26491.39

Figure 11 shows in thee dimensions the positions of the satellites, values from table 13 and 14, that gave the maximum and minimum error.

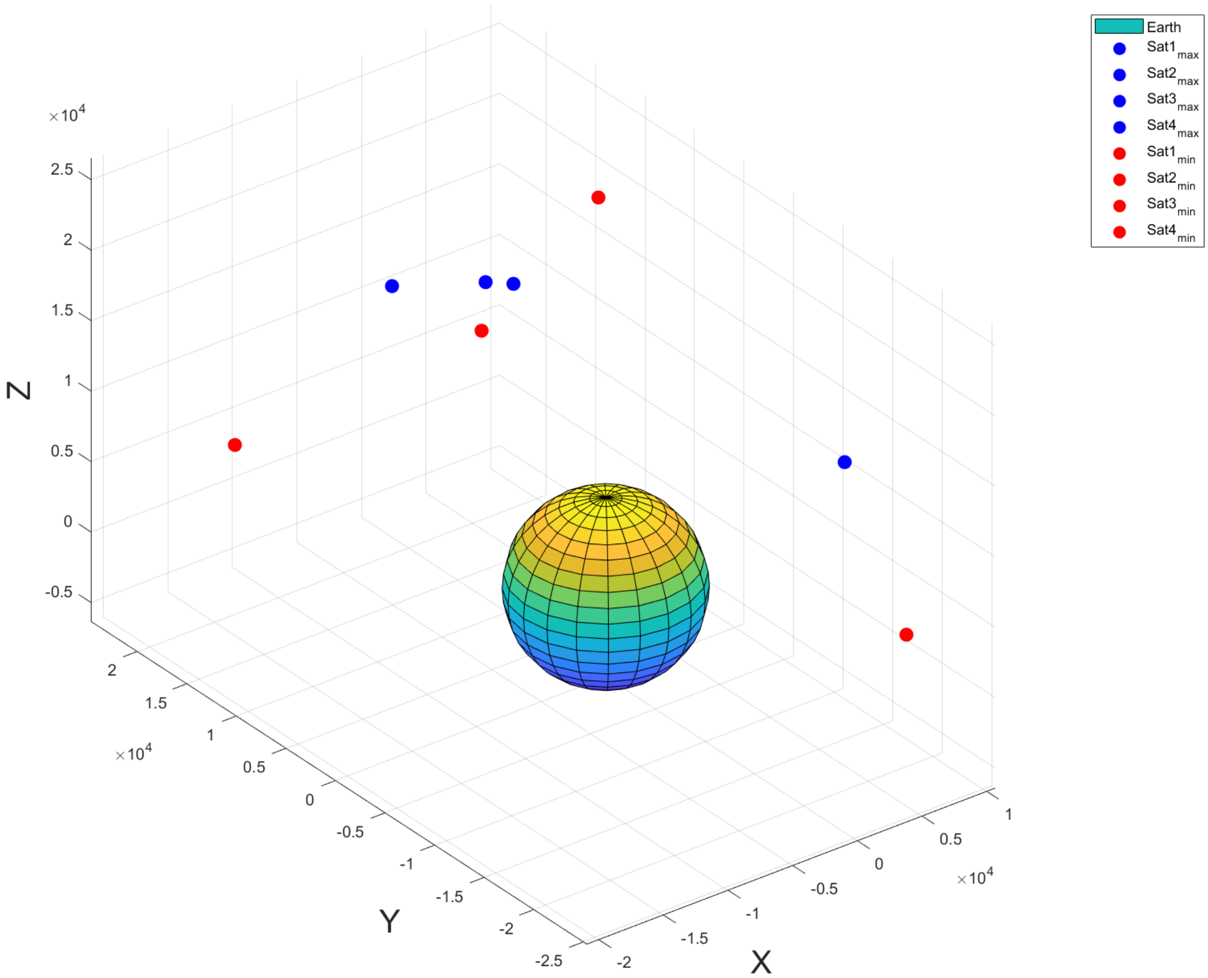


Figure 11: Locations of the satellites shown in 3-D that gave the maximum and minimum error

The blue satellites are the satellites that gave the maximum error, and the red satellites are the satellites that gave the minimum error. The figure shows how three out of four satellites are closely grouped above the earth, resulting in them giving similar data. Therefore this location composition of the satellites resulted in the maximum error. For comparison, the red satellites, which gave the minimum error, are however all spread out around the earth.

As shown in table 12 the maximum error of 7 m is a random result which is depending on the selected random numbers. This quantitative result is improved in the following experiment.

Problem 7

In this section the bisection method was used to determine the value of the measuring error of the ϕ angles to obtain the receiver's position within 10 cm, or in other words, determine the value of the measuring error of the ϕ angles to obtain the lower error in the receiver's position.

A function called bisection-error.m was created that contains a slightly modified code from the code used in problem 6. The only modification was that the code for the minimum error was removed since that value is not desired in this problem and then some slight changes were made so that collected random values of phi and theta from problem 6 could be used. Consequently, as the same random values for those angles are used in these two problems, the results from problem 7 are comparable to the results in problem 6. The bisection function, therefore, takes in values for a, b, the tolerance, and values for phi and theta.

In order to use the bisection method the values of input variables a, b, and the tolerance needed to be defined. The values of a and b are chosen so that the root is in between those values. It is known from problem 6 that by defining the measuring error of the ϕ angles as 10^{-8} the maximum error of the computed receiver's position is $7.1270 * 10^{-3}$ km. Therefore it can be expected that if the measuring error is less than 10^{-8} then the error of the receiver's position will be less, therefore the value of b is set to be this upper boundary

$$b = 10^{-8}$$

If the measuring error for ϕ is zero then it is known that the error of the position of the receiver is 0 cm, therefore a logical lower boundary of the interval, a, is

$$a = 0$$

By choosing these values for a and b it is certain that some value there between will result in an error of 10 cm for the position of the receiver.

The value for the tolerance was found by trial and error. At first, the tolerance was set to 10^{-8} but that led to a result of the error of the position of the receiver being larger than 10 cm. By lowering the value of the tolerance then the result was in accordance with the desired results, or below 10 cm. Therefore the value of the tolerance in the main code was set to

$$tol = 10^{-14}$$

The bisect.m function is called in the main file which returns the error of ϕ that corresponds to an error in the receiver's position being less than 10 cm. This is confirmed by inserting this value into the function bisection-error.m. These two values are compared to the results from problem 6 in the following table

Table 15

	Measuring error of ϕ angle	**Error of receiver's position [km]**
Problem 6	1e-08	7.13e-03
Problem 7	3.23e-12	9.98e-05

Table 15 shows that when the measuring error of ϕ angle was $1 * 10^{-8}$, as in problem 6, the error of the receiver's position was $7.13 * 10^{-3}$ km. In order for the receiver's positions to be calculated within 10 cm the measuring error of the ϕ angles needed to be $3.23 * 10^{-12}$. These results are in line with what was expected, it is logical that using data that has a lower error leads to better results than using data with a higher error.

In this problem it was important to realize what criteria the bisection method is using in the while loop. The bisect.m function is shown in figure 12. In order for the while loop to stop fc needs to be equal to zero. In problem 7 it is important that this occurs when the error of the receiver's position is 10 cm or less. Therefore it is necessary to subtract 10 cm from the outcome of the bisection-error.m function when calling it inside of the bisect.m function, see line 5 in the following code.

```
1 function xc = bisect(a,b,tol,phi,theta)
2 %xc is the angle error that gives the measuring error less than 10 cm.
3
4
5 f=@(measuring_error)bisection_error(measuring_error,phi,theta)-0.0001; % we need to substract 10 cm
    ↪ from f because the bisection method is looking for the "zero station"
6
7
8 if sign(f(a))*sign(f(b)) >= 0
9   error('f(a)f(b)<0 not satisfied!') %ceases execution
10 end
11 fa=f(a);
12 while (b-a)/2>tol
13   c=(a+b)/2;
14   fc=f(c);
15   if fc == 0               %c is a solution, done
16     break
17   end
18   if sign(fc)*sign(fa)<0   %a and c make the new interval
19     b=c;
20   else                     %c and b make the new interval
21     a=c;fa=fc;
22   end
23 end
24 xc=(a+b)/2;               %new midpoint is best estimate
```

Figure 12: Bisect.m

Problem 8

The resulting measuring error of using GPS with four satellites is according to problem 6 unacceptable for locating a responder on earth. Five satellites are used to examine the influence of the number of satellites on the measuring error, which adds an extra equation.

$$f_5(x, y, z, d) = (x - A_5)^2 + (y - B_5)^2 + (z - C_5)^2 - c^2(t_5 - d)^2 = 0 \tag{12}$$

Since it is not possible to use Newton's method when having more equations than variables, the Gauss-Newton (non-linear least squares) is used instead.

Carl Friedrich Gauss modified the Newton method by multiplying the transpose Jacobi matrix to both sides. This resolves in a square matrix, which enables the algorithm to solve the systematlab, this is done by creating a new function that is similar to figure 4 but adding the transpose Jacobi matrix in line 10, as shown in figure 13.

```
1  function  x = Gaussnewton(x0,tol,A,B,C,t,c,n)
2
3  x=x0;
4  oldx = x0+2*tol;
5  jac = jacobifunc(A, B, C, c, t,x0,n); %jacobi
6  jacT = transpose(jac);
7
8      while norm(x-oldx,inf)>tol
9         oldx=x;
10        s=jacT*jac\jacT*Ffunc(A,B,C,c,t,x,n);
11        x = x-s; %new x calculated
12
13     end
14 end
```

Figure 13: Gauss-Newton method

The Gauss-Newton method is then implemented in the code of Problem 6. To adjust the code for one additional satellite, two random numbers, one phi and one theta, are added by creating a for-loop (line 267-299), as shown in section H. Generating these values with a for-loop makes the code scalable for further experiments, as shown in problem 9. Another change to the code of problem 6 is calculating the new position value with the Gauss-Newton method and adding additional plots for the fifth satellite.

Adding another satellite seems to result in smaller measuring errors, which are shown in table 16. The minimum and the maximum errors appear when the satellites are positioned according to table 17 and 18. These randomly generated positions are different each time the code is run.

Table 16: Measuring errors when the satellites are randomly positioned

	Measuring errors
Minimum error	1.0679e-4
Average error	3.1022e-4
Maximum error	9.2631e-4

Table 17: Location of the five satellites when the **maximum error** occurred

Satellites	A	B	C
Nr. 1	20738.58	1095.42	16573.36
Nr. 2	25138.34	1244.55	8513.52
Nr. 3	-14.96	26452.72	2493.67
Nr. 4	23216.84	5142.05	11853.38
Nr. 5	-6822.01	15404.13	20546.00

Table 18: Location of the five satellites when the **minimum error** occurred

Satellites	A	B	C
Nr. 1	8124.65	23748.74	8715.06
Nr. 2	3218.03	-3637.70	26122.33
Nr. 3	21321.02	-13452.46	8391.08
Nr. 4	-15062.24	-21845.83	1361.50
Nr. 5	-19709.34	-1281.97	17772.54

The positions of the satellites are displayed in the following figure 14. The figure shows that the maximum error occurs when the satellites are grouped up and close to each other. When the satellites are evenly distributed, the error is smaller (red).

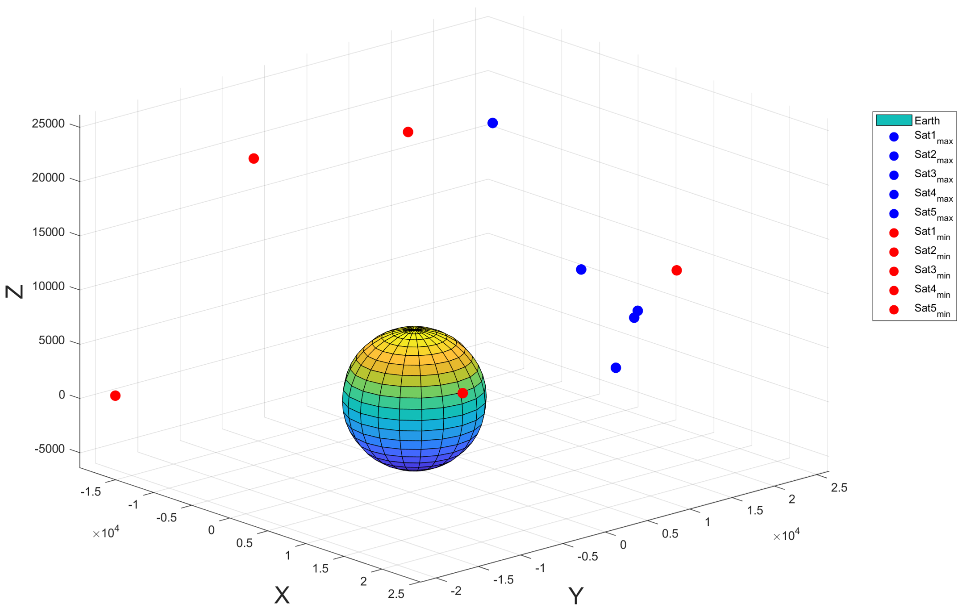


Figure 14: Locations of the satellites shown in 3-D that gave the maximum and minimum error

For comparison with Problem number 6, the histogram of the measured error is shown. Compared to problem number 6 the maximum of distribution is ten times smaller. This is only a quantitative result because the random values have changed.

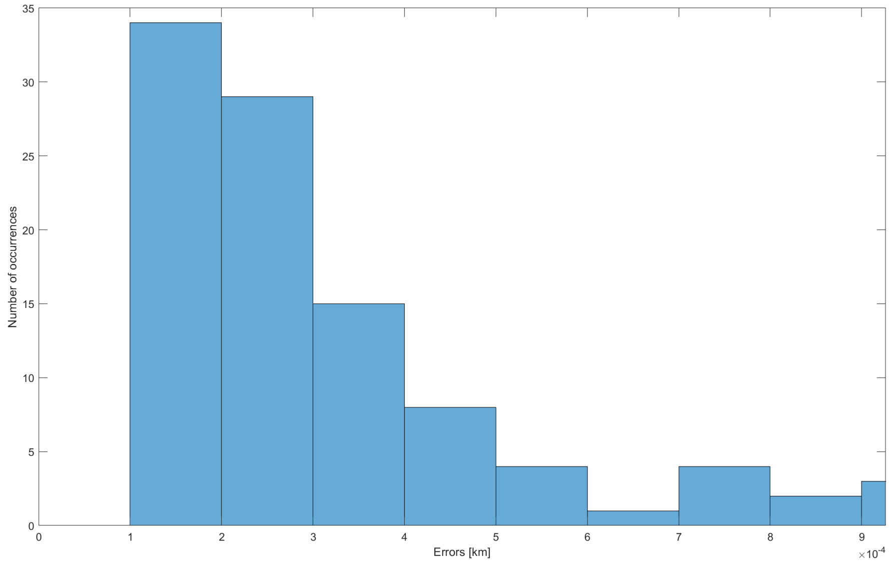


Figure 15: Distribution of the measured errors with respect to the different positions of the satellites

Problem 9

The process from problem 8 is repeated with n (6,7,8,9) satellites and identical random variation of the angles, to further examine the influence of the number of satellites on the measuring error.

The code from problem 8 is adjusted by adding a for-loop for the number of satellites and changing existing parts into scalable for-loops, as shown in the following code. Inside the for-loop, the plot of the n satellites in space and a binary matrix with (-1,1) is generated. The binary matrix adjusts in size and enables the algorithm to continue like in problem 8 with 100 iterations, while ϕ and θ are also generated in a for-loop.

```
for n = 6:1:9 %number of satelites counting

    figure
    [x,y,z] = sphere;  % Make unit sphere
    radius = 6370; % Scale to desire radius.
    x = x * radius;
    y = y * radius;
    z = z * radius;
    offset = 0;%6370;     % Translate sphere to new location.
    surf(x+offset,y+offset,z+offset) % Plot as surface.
    xlabel('X', 'FontSize', 20);
    ylabel('Y', 'FontSize', 20);
    zlabel('Z', 'FontSize', 20);
    axis equal;
    hold on


    out=ff2n(n); % creating -1,1 Matrix, 16x4 for 4 -> 512x9 for 9
    out(out==0) = -1; %replacing '0' with -1

    for m = 1:100
        for i=1:n %Calculate the angles with out the error:
            phi(m,i) = rand*pi/2 ;
            theta(m,i) = rand*pi*2 ;

        end

        [A B C R t] = distance_and_time(P,phi(m,:),theta(m,:),c,n);
```

After the time is calculated, another for-loop is added to calculate the n amount of $\phi 2$, which is then used to calculate the new position and the error according to the North Pole in the same way as in problem 8. The resulting minimum and maximum errors are captured during the 100 iterations. The min and max errors are saved for every n number of satellites at the end of the 100 iterations by creating an if, else if, and else condition. Inside this condition, the values are labeled according to the number of satellites, and the plot into one graph is initiated.

```
for k = 1: 2^n %Calculate the angles with error:
        for l = 1:n
           phi2(m,l) = phi(m,l)+10^(-8)*out(k,l);
        end
        [A2 B2 C2 R2 t2] = distance_and_time(P,phi2(m,:),theta(m,:),c,n);
        pos_new = Gaussnewton(x0,tol2, A2,B2,C2,t,c,n);
        total_err(k,:) = norm(pos_new-x0);


        if total_err(k,:) < min_err %Keep track of the minimum error to be able to plot up the
    ↪ locations of the satellites when that happens
            min_err = total_err(k,:);
            A_min = A2;
            B_min = B2;
            C_min = C2;
        end

        if total_err(k,:) > max_err %Keep track of the maximum error
            max_err = total_err(k,:);
            A_max = A2;
            B_max = B2;
            C_max = C2;
        end

    end


    maximum_vec(m,:) = max(total_err); %We want to achieve the "truest" value of all of those
    ↪ cominations, therefore we take the worst value


    if m==100
        if n==6

            maximum_vec_6 = max(maximum_vec); %We want to obtain the 3 values (max, min, ave) of
    ↪  the maximum vec which contains 100 values from the 100 iterations
            minimum_vec_6 = min(maximum_vec);
            average_vec_6 = mean(maximum_vec);
            for i=1:n
                hold on
                plot3(A_max(i),B_max(i),C_max(i), 'b.', 'MarkerSize',30)
            end
            for i=1:n
                hold on
                plot3(A_min(i),B_min(i),C_min(i), 'r.', 'MarkerSize',30)
            end
            leg = legend('Earth', 'Sat1_{max}', 'Sat2_{max}', 'Sat3_{max}', 'Sat4_{max}','Sat5_{
    ↪ max}','Sat6_{max}', 'Sat1_{min}', 'Sat2_{min}', 'Sat3_{min}', 'Sat4_{min}', 'Sat5_{min}','
    ↪ Sat6_{min}', 'Location', 'NorthEast');
            title('6 satellites');
```

The results of this experiment show, that the min, max, and mean errors are decreasing when more satellites are used. While the min and mean error imply a linear reduction in asymptotic behavior, the maximum error is decreasing stronger, as shown in figure 16. The amplitude of the error is influenced by the random value variations, while the all-over trend is a representative result.

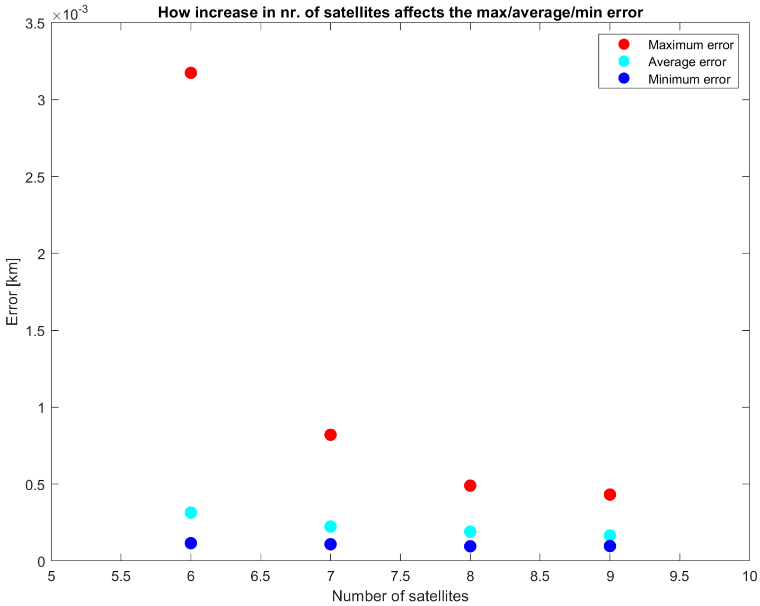
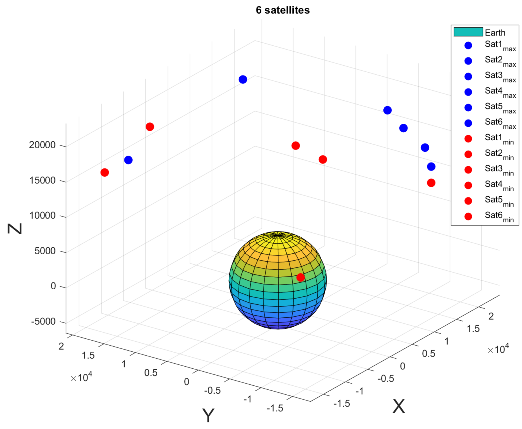
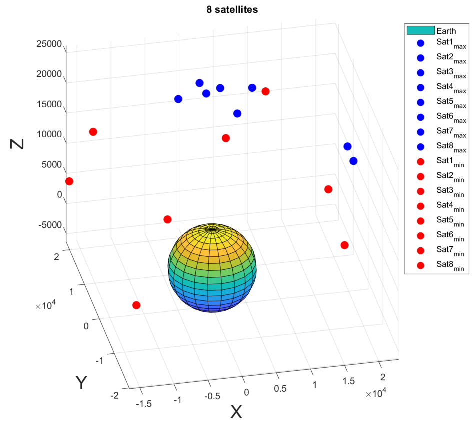


Figure 16: Min, Mean and, max error with n (6,7,8,9) satellites used

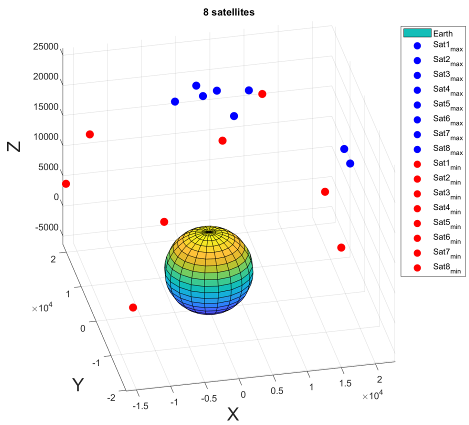
The positions for these results are shown in the following figures 17. The blue satellites represent the maximum error and the red satellites the minimum error. When the satellites are close to each other (17, b) or in similar planes (17, a), then the error seems to be larger. On the other hand, when the satellites are equally distributed (17, c), the error seems smaller. Nine satellites are more difficult to analyze, but as seen in figure 16 the maximum error decreases further.



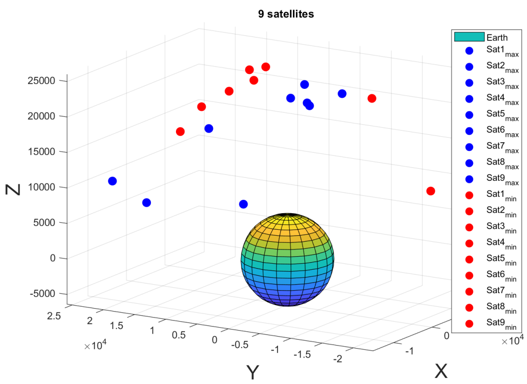
(a) n = 6



(b) n = 7



(c) n = 8



(d) n = 9

Figure 17: Different numbers of satellites result in the max and minimum error.

Free-choice question

In this question, the foundation of problem 3 was reused but was modified so the four satellites have four different orbits which will stay constant for each satellite (angle θ), while the satellites travel in their orbit (angle ϕ, like is shown in figure 5) from angle 0 to $\pi/2$ as can be seen on the left in figure 19. The motivation of this problem is to investigate what angle of the satellites gives the best and worst effect (error) on the computed receiver's position if the satellites are all moving in an orbit that resembles an umbrella, shown in figure 19.

After setting up the program to calculate the errors of the receiver's position, the data wasn't reliable since the values of the errors were around 10^4 km and even some NaN values. It turns out that the orbits can intersect at some point but a satellite can't travel in the same path as another satellite even if it is in the opposite direction as shown in the left of figure 18

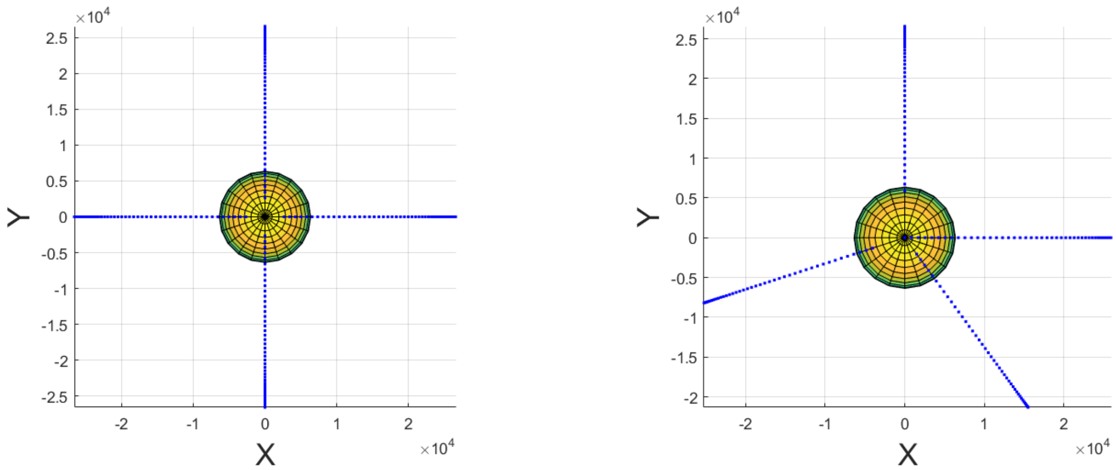


Figure 18: The left figure shows four satellites (the blue dots) moving on only two orbits but in opposite directions. In the right figure, the orbit of the four satellites has been adjusted so each satellite has its own orbit to travel in.

This problem was resolved by adjusting the θ vector (line 582), in section H which controls the angle of the orbits preventing the orbits from aligning, see left and right figure in figure 18. Therefore all the satellites had a unique orbit to travel in as can be seen in the right of figure 18.

After these adjustments, the NaN values disappeared from the error data, but the data was still unreliable and the errors were equally big as before. It turns out that the angle of the satellites when they are sending data matters and can't be the same in all the satellites at once. The solution to this problem is to have different values on the height of the satellites from the earth (which is the C coordinate of the satellites). Table 19 and 20 show before and after the change of the C coordinates of the satellites.

Table 19: Shows the coordinates of the satellites initial position where **C** is the same for all the satellites

Satellites	A [km]	B [km]	**C [km]**
Nr. 1	$2.6526 * 10^3$	0	$2.6437 * 10^4$
Nr. 2	$1.5591 * 10^3$	$-2.1460 * 10^3$	$2.6437 * 10^4$
Nr. 3	$-2.5227 * 10^3$	-819.6903	$2.6437 * 10^4$
Nr. 4	$1.6242 * 10^{-13}$	$2.6526 * 10^3$	$2.6437 * 10^4$

Table 20: Shows the coordinates of the satellites initial position where **C** is different for all the satellites

Satellites	A	B	C
Nr. 1	$2.6570 * 10^4$	0	$2.6570 * 10^4$
Nr. 2	$1.4996 * 10^3$	$-2.0641 * 10^3$	$2.6447 * 10^4$
Nr. 3	$-4.0330 * 10^3$	$-1.3104 * 10^3$	$2.6229 * 10^4$
Nr. 4	$3.6203 * 10^{-13}$	$5.9124 * 10^3$	$2.5904 * 10^4$

Since the orbits and the height of the satellites were all different, the errors in the computed receiver's position were reliable and realistic. The errors were plotted in a graph as a function of the angle ϕ which can be seen on the right, in figure 19.

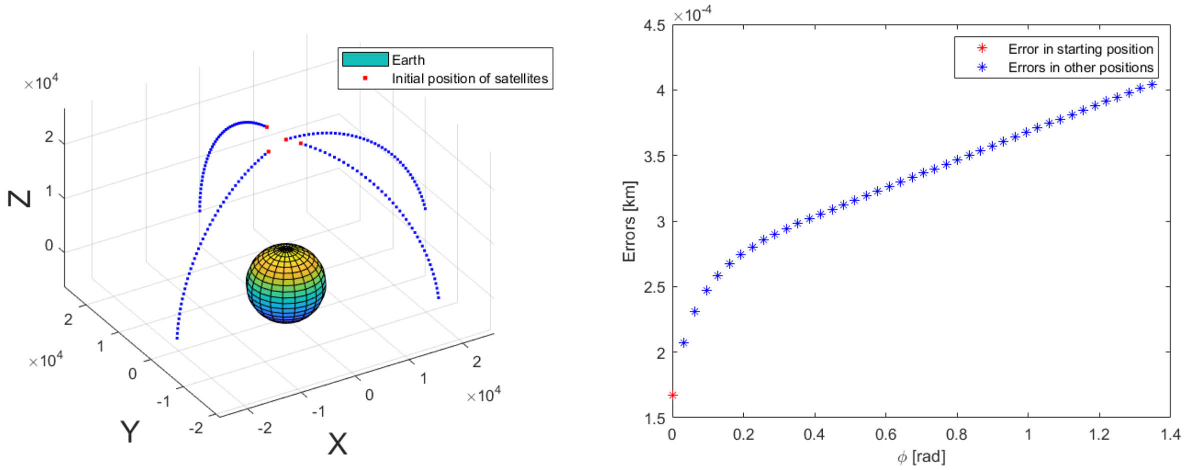


Figure 19: The left figure shows all positions of the four satellites (the red and blue dots) when moving in their orbit. In the right figure, the distribution of the errors is shown with respect to the angle ϕ which controls the positions of the satellites.

As can be seen from the graph, the minimum error happens when the ϕ angle is 0, then as the ϕ angle increases the error increases. What can be interpreted from the left of figure 19 is that when the satellites are over the receiver position (which is at the North pole) the accuracy of the data from the satellites is better than when the angle ϕ increases and the satellites move down in their orbit, due to the satellites being located at the same plane as the equator and trying to locate the receiver which is at the north pole on top of the earth.

References

[1] Wikipedia, "Spherical coordinates," 23.11.2022. [Online]. Available: https://de.wikipedia.org/wiki/Datei:Spherical_Coordinates_%28Colatitude,_Longitude%29.svg

Appendix

A F-Function

```
function F = Ffunc(A,B,C,c,t,pos)
    for i = 1:4
       F(i,1) = (pos(1)-A(i))^2 + (pos(2)-B(i))^2 + (pos(3)-C(i))^2 - c^2*(t(i)-pos(4))^2;
    end
end
```

B Jacobi Function

```
function DF = jacobifunc(A, B, C, c, t,pos,n)
    for i = 1:n
        DF(i,1) = 2*(pos(1) - A(i));
        DF(i,2) = 2*(pos(2) - B(i));
        DF(i,3) = 2*(pos(3) - C(i));
        DF(i,4) = 2*t(i)*c^2 - 2*c^2*pos(4);
    end
end
```

C Distance and Time

```
function [A B C R t] = distance_and_time(P, phi, theta, c,n)
    for i = 1:n
        A(i) = P*sin(phi(i))*cos(theta(i));
        B(i) = P*sin(phi(i))*sin(theta(i));
        C(i) = P*cos(phi(i));
    end

    for i = 1:n
        R(i) = sqrt((A(i))^2 + (B(i))^2 + (C(i)-6370)^2);
        t(i) = R(i)/c;
    end
end
```

D Multidimensional Newton Method

```
function x = newtonmult(x0,tol,A,B,C,t,c)
x=x0;
oldx=x0+2*tol;
    while norm(x-oldx,inf)>tol
       oldx=x;
       s=-jacobifunc(A,B,C,c,t,x)\Ffunc(A,B,C,c,t,x);
       x=x+s;
    end
end
```

E Bisect

```
function xc = bisect(a,b,tol,phi,theta)
%xc is the angle error that gives the measuring error less than 10 cm.


f=@(measuring_error)bisection_error(measuring_error,phi,theta)-0.0001; % we need to substract 10 cm
    ↪ from f because the bisection method is looking for the "zero station"


if sign(f(a))*sign(f(b)) >= 0
  error('f(a)f(b)<0 not satisfied!') %ceases execution
end
fa=f(a);
while (b-a)/2>tol
  c=(a+b)/2;
  fc=f(c);
  if fc == 0              %c is a solution, done
    break
  end
  if sign(fc)*sign(fa)<0  %a and c make the new interval
    b=c;
  else                    %c and b make the new interval
    a=c;fa=fc;
  end
end
xc=(a+b)/2;               %new midpoint is best estimate
```

F Bisection Error

```
function [max_err] = bisection_error(measuring_error,phi,theta)
%This function takes in the measuring error of phi and calculates for
%different random values of the angles and for the specified
%measuring_error and returns the maximum error.


c = 299792.458;
P=26570; %km
x0 = [0;0;6370;0]; % starting point is north pole
tol2 = 10^(-8);
n=4;

max_err = 0;
counter = 1;
total_err=[];
for m = 1:100

    [A B C R t] = distance_and_time(P,phi(m,:),theta(m,:),c,n);

    for i = [-1 1]
        for j = [-1 1]
            for k = [-1 1]
                for h = [-1 1]
                    %Calculate the values with the error:
                    phi2=[phi(m,1)+(i)*measuring_error, phi(m,2)+(j)*measuring_error, phi(m,3)+(k)*
    ↪ measuring_error, phi(m,4)+(h)*measuring_error];
                    [A2 B2 C2 R2 t2] = distance_and_time(P,phi2,theta(m,:),c,n);
```

```

                    pos_new = newtonmult(x0,tol2,A2,B2,C2,t,c,n); %location with some error

                    total_err(counter) = norm(pos_new-x0);

                    counter = counter + 1;
                end
            end
        end
    end
    maximum_errors(m) = max(total_err);
    total_err = 0;

end
max_err = max(maximum_errors);
end
```

G Gauss Newton

```
function  x = Gaussnewton(x0,tol,A,B,C,t,c,n)

x=x0;
oldx = x0+2*tol;
jac = jacobifunc(A, B, C, c, t,x0,n); %jacobi
jacT = transpose(jac);

    while norm(x-oldx,inf)>tol
       oldx=x;
       s=jacT*jac\jacT*Ffunc(A,B,C,c,t,x,n);
       x = x-s; %new x calculated

    end
end
```

H Main Code

```
%% 1
clear all; close all;clc;

x0 = [0;0;6370;0]; % starting point is north pole
A = [15600;18760;17610;19170]; % A,B, C, t are the satellite coordinates
B = [7540; 2750; 14630; 610];
C = [20140; 18610; 13480; 18390];
t = [0.07074; 0.07220; 0.07690; 0.07242];
c = 299792.458; %speed of light
tol = 10^(-3); %so we have the error in meters in stead of kilometers
n=4;


x = newtonmult(x0,tol,A,B,C,t,c,n); % x is a vector that contains x,y,z and d

fprintf('Problem 1\n\nThe location of the reciever is the following\n\nx = %.6f\ny = %.6f\nz = %.6f\
    ↪ n\nAnd the difference between the satellite clock and the receiver clock\nd = %.6s',x(1),x(2)
    ↪ ,x(3),x(4))

%% 2

P=26570; %km
phi=[pi/8, pi/6, pi/4, pi/2];
theta = [pi/2, pi, 3*pi/2, 2*pi];
n=4;

[A B C R t] = distance_and_time(P,phi,theta,c,n); % TO GET COORDINATES OF THE SATELLITE (A, B, C) OR
    ↪  THE DISTANCE (R) OR THE TIME (t)

fprintf('\n\nProblem 2\n\n');
fprintf('The location of the four satellites:\n')
fprintf('Satellite 1: %.2f km, %.2f km, %.2f km\n', A(1), B(1), C(1))
fprintf('Satellite 2: %.2f km, %.2f km, %.2f km\n', A(2), B(2), C(2))
fprintf('Satellite 3: %.2f km, %.2f km, %.2f km\n', A(3), B(3), C(3))
fprintf('Satellite 4: %.2f km, %.2f km, %.2f km \n\n', A(4), B(4), C(4))
fprintf('\nDistance\n');
fprintf('%.6f km\n',R);
fprintf('\nTime\n');
fprintf('%.6f sec\n',t);


%% 3
clear all; close all; clc;
c = 299792.458; %speed of light
P=26570; %km
x0 = [0;0;6370;0]; % starting point is north pole
tol2 = 10^(-9);
phi=[pi/8, pi/6, 3*pi/8, pi/4];
theta=[-pi/4, pi/2, 2*pi/3, pi/6];
n=4;

[A B C R t] = distance_and_time(P,phi,theta,c,n);


phi2=[pi/8+10^(-8), pi/6+10^(-8), 3*pi/8-10^(-8), pi/4-10^(-8)];
[A2 B2 C2 R2 t2] = distance_and_time(P,phi2,theta,c,n);

```

```
pos_new = newtonmult(x0,tol2,A2,B2,C2,t,c,n); %location with some error

total_err = norm(pos_new-x0);
fprintf('\nThe error is: %.4s km',total_err)

%% 4
close all; clear all; clc;
c = 299792.458; %speed of light
P=26570; %km
x0 = [0;0;6370;0]; %starting point is north pole
tol2 = 10^(-9);
phi=[pi/8, pi/6, 3*pi/8, pi/4];
theta=[-pi/4, pi/2, 2*pi/3, pi/6];
n=4;


[A B C R t] = distance_and_time(P,phi,theta,c,n);
max_err = 0; sign_1=0; sign_2=0; sign_3=0; sign_4=0;
counter = 1;
for i = [-1 1]
    for j = [-1 1]
        for k = [-1 1]
            for h = [-1 1]
                phi2(counter,:)=[pi/8+(i)*10^(-8), pi/6+(j)*10^(-8), 3*pi/8+(k)*10^(-8), pi/4+(h)
    ↪ *10^(-8)];
                [A2 B2 C2 R2 t2] = distance_and_time(P,phi2(counter,:),theta,c,n);

                pos_new = newtonmult(x0,tol2,A2,B2,C2,t,c,n); %location with some error

                total_err(counter) = norm(pos_new-x0);
                if total_err(counter) > max_err
                    max_err = total_err(counter);
                    sign_1=i;
                    sign_2=j;
                    sign_3=k;
                    sign_4=h;
                end
                counter = counter + 1;
            end
        end
    end
end

fprintf('\nProblem 4\n\nThe maximum value for the error is: %.4s',max_err)
fprintf('\nThe sign combination that gives the maximum error is the following, where -1 and 1
    ↪ represent - and +:\n%.f\n %.f\n%.f\n%.f\n',sign_1, sign_2,sign_3,sign_4)


%% 5

close all; clear all; clc;
c = 299792.458; %speed of light
P=26570; %km
x0 = [0;0;6370;0]; % starting point is north pole
tol2 = 10^(-9);
n=4;
phi=[pi/8.00000000000011, pi/6, 3*pi/8, pi/8.00000000000012];

theta=[-pi/4.00000000000011, pi/2, 2*pi/3, -pi/4.00000000000012];
```

```

[A B C R t] = distance_and_time(P,phi,theta,c,n);

phi2=[pi/8.00000000000011+10^(-8),pi/6+10^(-8), 3*pi/8-10^(-8), pi/8.00000000000012+10^(-8)];

[A2 B2 C2 R2 t2] = distance_and_time(P,phi2,theta,c,n);

pos_new = newtonmult(x0,tol2,A2,B2,C2,t,c,n); %location with some error
total_err = norm(pos_new-x0);
fprintf('\nPart5\n\nTotal error: %.3s', total_err)



%% 6
clear all; close all; clc;

c = 299792.458; %speed of light
P=26570; %km
x0 = [0;0;6370;0]; % starting point is north pole
tol2 = 10^(-8);
n=4;

max_err = 0;
min_err = 1;
counter = 1;

for m = 1:100
    %Calculate the values with out the error:
    nr_1 = rand; nr_2 = rand; nr_3 = rand; nr_4 = rand; nr_5 = rand; nr_6 = rand; nr_7 = rand; nr_8
    ↪ = rand; %To create different random  numbers, for phi the random numbers for the correct data
    ↪  and wrong data still need to be the same for each satellite in each run
    phi(m,:)= [nr_1*(pi/2), nr_2*(pi/2), nr_3*(pi/2), nr_4*(pi/2)];
    theta(m,:) = [nr_5*(pi*2), nr_6*(pi*2), nr_7*(pi*2), nr_8*(pi*2)];
    [A B C R t] = distance_and_time(P,phi(m,:),theta(m,:),c,n);

    for i = [-1 1]
        for j = [-1 1]
            for k = [-1 1]
                for h = [-1 1]
                    %Calculate the values with the error:
                    phi2=[nr_1*pi/2+(i)*10^(-8), nr_2*pi/2+(j)*10^(-8), nr_3*pi/2+(k)*10^(-8), nr_4*
    ↪ pi/2+(h)*10^(-8)];
                    [A2 B2 C2 R2 t2] = distance_and_time(P,phi2,theta,c,n);
                    loc_A(counter,:) = A2;
                    loc_B(counter,:) = B2;
                    loc_C(counter,:) = C2;
                    pos_new = newtonmult(x0,tol2,A2,B2,C2,t,c,n); %location with some error

                    total_err(counter) = norm(pos_new-x0);

                    counter = counter + 1;
                end
            end
        end
    end

    [val, ind] =  max(total_err); %Out of those 16 errors we want to take the maximum as the "
    ↪ realistic" value from those iterations
    real_total_err(m)= val;
```

```
    A_max(m,:) = loc_A(ind,:);
    B_max(m,:) = loc_B(ind,:);
    C_max(m,:) = loc_C(ind,:);

    total_err = []; loc_A=[]; loc_B=[]; loc_C=[];
end


[min_err, ind_min] = min(real_total_err);
A_min_plot = A_max(ind_min,:);
B_min_plot = B_max(ind_min,:);
C_min_plot = C_max(ind_min,:);

[max_err, ind_max] = max(real_total_err);
A_max_plot = A_max(ind_max,:);
B_max_plot = B_max(ind_max,:);
C_max_plot = C_max(ind_max,:);

avg_err= mean(real_total_err);


figure
histogram(real_total_err)
xlim([0, max(real_total_err)]);
xlabel('Errors [km]')
ylabel('Number of occurrences')

figure
[x,y,z] = sphere; % Make unit sphere
radius = 6370; % Scale to desire radius.
x = x * radius;
y = y * radius;
z = z * radius;
offset = 0;%6370;% Translate sphere to new location.
surf(x+offset,y+offset,z+offset) % Plot as surface.
xlabel('X', 'FontSize', 20);% Label axes.
ylabel('Y', 'FontSize', 20);
zlabel('Z', 'FontSize', 20);
axis equal;


for i=1:n
    hold on
    plot3(A_max_plot(i),B_max_plot(i),C_max_plot(i), 'b.', 'MarkerSize',30)
end

for i=1:n
    hold on
    plot3(A_min_plot(i),B_min_plot(i),C_min_plot(i), 'r.', 'MarkerSize',30)
end


leg = legend('Earth', 'Sat1_{max}', 'Sat2_{max}', 'Sat3_{max}', 'Sat4_{max}', 'Sat1_{min}', 'Sat2_{
    ↪ min}', 'Sat3_{min}', 'Sat4_{min}', 'Location', 'NorthEast');


fprintf('Problem 6:\n')
```

```
fprintf('\nThe minimum error is: %.4s km\nThe average error is: %.4s km\nThe maximum error is: %.4s km\n\n',min_err,avg_err,max_err)
fprintf('The location (A, B, C) of the four satellites when the maximum error occurred is:\n')
for i=1:n
    fprintf('Satellite %d: %.2f km, %.2f km, %.2f km\n',i, A_max_plot(i), B_max_plot(i), C_max_plot(i))

end

fprintf('\nThe location (A, B, C) of the four satellites when the minimum error occurred is:\n')
for i=1:n
    fprintf('Satellite %d: %.2f km, %.2f km, %.2f km\n', i, A_min_plot(i), B_min_plot(i), C_min_plot(i))

end




%% 7

a=0;
b=10^(-8); %We set b as the value we have been using for the measuring error of phi angle
tol=10^(-14);

xc = bisect(a,b,tol,phi,theta); %It is logical if the satellites are sending us data with an error less than 10^(-8) error so that we can get the recievers location within 10 cm.


[max_err] = bisection_error(xc,phi,theta); %Calculates what the error in the recievers location is in km

fprintf('If the measuring error of phi angle is %.2s then we get the receiver position within %.2s\n',xc, max_err) %Note that the measuring error of phi that gives this accuracy of the location of the reciever is less than 10^(-8) which makes sense.

%% 8

clear all; close all; clc;

c = 299792.458; %speed of light
P=26570; %km
x0 = [0;0;6370;0]; % starting point is north pole
tol2 = 10^(-8);

max_err = 0;
min_err = 10000000;
counter = 1;

n = 5 ; %Number of satillites
for m = 1:100
    %Calculate the values with out the error:
    for i=1:n
        phi(m,i) = rand*pi/2 ;
        theta(m,i) = rand*pi*2 ;
    end


    [A B C R t] = distance_and_time(P,phi(m,:),theta(m,:),c,n);
```

```

    for a = [-1 1]
        for j = [-1 1]
            for k = [-1 1]
                for h = [-1 1]
                    for z = [-1 1]
                        %Calculate the values with the error:
                        phi2(m,:)=[phi(m,1)+(a)*10^(-8), phi(m,2)+(j)*10^(-8), phi(m,3)+(k)*10^(-8), phi(m,4)+(h)*10^(-8), phi(m,5)+(z)*10^(-8)];
                        [A2 B2 C2 R2 t2] = distance_and_time(P,phi2(m,:),theta(m,:),c,n);
                        loc_A(counter,:) = A2;
                        loc_B(counter,:) = B2;
                        loc_C(counter,:) = C2;

                        pos_new = Gaussnewton(x0,tol2, A2,B2,C2,t,c,n);

                        total_err(counter) = norm(pos_new-x0);

                        counter = counter + 1;
                    end

                end
            end
        end
    end
    [val, ind] =  max(total_err); %Out of those 16 errors we want to take the maximum as the "realistic" value from those iterations
    real_total_err(m)= val;

    A_max(m,:) = loc_A(ind,:);
    B_max(m,:) = loc_B(ind,:);
    C_max(m,:) = loc_C(ind,:);

    total_err = []; loc_A=[]; loc_B=[]; loc_C=[];
end


[min_err, ind_min] = min(real_total_err);
A_min_plot = A_max(ind_min,:);
B_min_plot = B_max(ind_min,:);
C_min_plot = C_max(ind_min,:);

[max_err, ind_max] = max(real_total_err);
A_max_plot = A_max(ind_max,:);
B_max_plot = B_max(ind_max,:);
C_max_plot = C_max(ind_max,:);

avg_err= mean(real_total_err);


figure
histogram(real_total_err)
xlim([0, max(real_total_err)]);
xlabel('Errors [km]')
ylabel('Number of occurrences')

figure
% Make unit sphere
[x,y,z] = sphere;
```

```
% Scale to desire radius.
radius = 6370;
x = x * radius;
y = y * radius;
z = z * radius;
% Translate sphere to new location.
offset = 0;%6370;
% Plot as surface.
surf(x+offset,y+offset,z+offset)
% Label axes.
xlabel('X', 'FontSize', 20);
ylabel('Y', 'FontSize', 20);
zlabel('Z', 'FontSize', 20);
axis equal;


for i=1:n
    hold on
    plot3(A_max_plot(i),B_max_plot(i),C_max_plot(i), 'b.', 'MarkerSize',30)
end

for i=1:n
    hold on
    plot3(A_min_plot(i),B_min_plot(i),C_min_plot(i), 'r.', 'MarkerSize',30)
end

leg = legend('Earth', 'Sat1_{max}', 'Sat2_{max}', 'Sat3_{max}', 'Sat4_{max}','Sat5_{max}', 'Sat1_{
    ↪ min}', 'Sat2_{min}', 'Sat3_{min}', 'Sat4_{min}', 'Sat5_{min}', 'Location', 'NorthEast')


fprintf('Problem 8:')
fprintf('\nThe minimum error is: %.4s km\nThe average error is: %.4s km\nThe maximum error is: %.4s
    ↪ km\n\n',min_err,avg_err,max_err)
fprintf('The location of the four satellites when the maximum error occurred is:\n')

for i=1:n
    fprintf('Satellite %d: %.2f km, %.2f km, %.2f km\n',i, A_max_plot(i), B_max_plot(i), C_max_plot(
    ↪ i))
end

fprintf('The location of the four satellites when the minimum error occurred is:\n')
for i=1:n
    fprintf('Satellite %d: %.2f km, %.2f km, %.2f km\n', i,A_min_plot(i), B_min_plot(i), C_min_plot(
    ↪ i))
end



%% 9

clear all; close all; clc;

c = 299792.458; %speed of light
P=26570; %km
x0 = [0;0;6370;0]; % starting point is north pole
tol2 = 10^(-8);

max_err = 0;
min_err = 10000000000000;
```

```
A_min = 0; B_min = 0; C_min = 0; A_max = 0; B_max = 0; C_max = 0;

for n = 6:1:9 %number of satelites counting

    figure
    [x,y,z] = sphere;  % Make unit sphere
    radius = 6370; % Scale to desire radius.
    x = x * radius;
    y = y * radius;
    z = z * radius;
    offset = 0;%6370;     % Translate sphere to new location.
    surf(x+offset,y+offset,z+offset) % Plot as surface.
    xlabel('X', 'FontSize', 20);
    ylabel('Y', 'FontSize', 20);
    zlabel('Z', 'FontSize', 20);
    axis equal;
    hold on


    out=ff2n(n); % creating -1,1 Matrix, 16x4 for 4 -> 512x9 for 9
    out(out==0) = -1; %replacing '0' with -1

    for m = 1:100
        for i=1:n %Calculate the angles with out the error:
            phi(m,i) = rand*pi/2 ;
            theta(m,i) = rand*pi*2 ;

        end

        [A B C R t] = distance_and_time(P,phi(m,:),theta(m,:),c,n);

        for k = 1: 2^n %Calculate the angles with error:
            for l = 1:n
               phi2(m,l) = phi(m,l)+10^(-8)*out(k,l);
            end
            [A2 B2 C2 R2 t2] = distance_and_time(P,phi2(m,:),theta(m,:),c,n);
            pos_new = Gaussnewton(x0,tol2, A2,B2,C2,t,c,n);
            total_err(k,:) = norm(pos_new-x0);


            if total_err(k,:) < min_err %Keep track of the minimum error to be able to plot up the locations of the satellites when that happens
                min_err = total_err(k,:);
                A_min = A2;
                B_min = B2;
                C_min = C2;
            end

            if total_err(k,:) > max_err %Keep track of the maximum error
                max_err = total_err(k,:);
                A_max = A2;
                B_max = B2;
                C_max = C2;
            end

        end


```

```
        maximum_vec(m,:) = max(total_err); %We want to achieve the "truest" value of all of those cominations, therefore we take the worst value


        if m==100
            if n==6

                maximum_vec_6 = max(maximum_vec); %We want to obtain the 3 values (max, min, ave) of the maximum vec which contains 100 values from the 100 iterations
                minimum_vec_6 = min(maximum_vec);
                average_vec_6 = mean(maximum_vec);
                for i=1:n
                    hold on
                    plot3(A_max(i),B_max(i),C_max(i), 'b.', 'MarkerSize',30)
                end
                for i=1:n
                    hold on
                    plot3(A_min(i),B_min(i),C_min(i), 'r.', 'MarkerSize',30)
                end
                leg = legend('Earth', 'Sat1_{max}', 'Sat2_{max}', 'Sat3_{max}', 'Sat4_{max}','Sat5_{max}','Sat6_{max}', 'Sat1_{min}', 'Sat2_{min}', 'Sat3_{min}', 'Sat4_{min}', 'Sat5_{min}','Sat6_{min}', 'Location', 'NorthEast');
                title('6 satellites');


            elseif n==7

                maximum_vec_7 = max(maximum_vec);
                minimum_vec_7 = min(maximum_vec);
                average_vec_7 = mean(maximum_vec);
                for i=1:n
                    hold on
                    plot3(A_max(i),B_max(i),C_max(i), 'b.', 'MarkerSize',30)
                end
                for i=1:n
                    hold on
                    plot3(A_min(i),B_min(i),C_min(i), 'r.', 'MarkerSize',30)
                end
                leg = legend('Earth', 'Sat1_{max}', 'Sat2_{max}', 'Sat3_{max}', 'Sat4_{max}','Sat5_{max}','Sat6_{max}','Sat7_{max}', 'Sat1_{min}', 'Sat2_{min}', 'Sat3_{min}', 'Sat4_{min}', 'Sat5_{min}','Sat6_{min}','Sat7_{min}', 'Location', 'NorthEast');
                title('7 satellites');

            elseif n==8

                maximum_vec_8 = max(maximum_vec);
                minimum_vec_8 = min(maximum_vec);
                average_vec_8 = mean(maximum_vec);

                for i=1:n
                    hold on
                    plot3(A_max(i),B_max(i),C_max(i), 'b.', 'MarkerSize',30)
                end
                for i=1:n
                    hold on
                    plot3(A_min(i),B_min(i),C_min(i), 'r.', 'MarkerSize',30)
                end
                leg = legend('Earth', 'Sat1_{max}', 'Sat2_{max}', 'Sat3_{max}', 'Sat4_{max}','Sat5_{max}','Sat6_{max}','Sat7_{max}', 'Sat8_{max}','Sat1_{min}', 'Sat2_{min}', 'Sat3_{min}', '
```

```
↪ Sat4_{min}', 'Sat5_{min}','Sat6_{min}','Sat7_{min}','Sat8_{min}', 'Location', 'NorthEast');
            title('8 satellites');

        else

            maximum_vec_9 = max(maximum_vec);
            minimum_vec_9 = min(maximum_vec);
            average_vec_9 = mean(maximum_vec);

            for i=1:n
                hold on
                plot3(A_max(i),B_max(i),C_max(i), 'b.', 'MarkerSize',30)
            end
            for i=1:n
                hold on
                plot3(A_min(i),B_min(i),C_min(i), 'r.', 'MarkerSize',30)
            end
            leg = legend('Earth', 'Sat1_{max}', 'Sat2_{max}', 'Sat3_{max}', 'Sat4_{max}','Sat5_{
↪ max}','Sat6_{max}','Sat7_{max}', 'Sat8_{max}','Sat9_{max}','Sat1_{min}', 'Sat2_{min}', 'Sat3_
↪ {min}', 'Sat4_{min}', 'Sat5_{min}','Sat6_{min}','Sat7_{min}','Sat8_{min}','Sat9_{min}', '
↪ Location', 'NorthEast');
            title('9 satellites');
        end

    end
end

total_err=[];
max_err = 0;
min_err = 10000000000000;
counter = 1;
maximum_vec = [];

end




fprintf('Problem 9: See the plots')




figure %Plot the max, min, av error
plot(9,maximum_vec_9,'r.','MarkerSize',30)
hold on
plot(9,average_vec_9,'c.','MarkerSize',30)
hold on
plot(9,minimum_vec_9,'b.','MarkerSize',30)

hold on
plot(8,maximum_vec_8,'r.','MarkerSize',30)
hold on
plot(8,average_vec_8,'c.','MarkerSize',30)
hold on
plot(8,minimum_vec_8,'b.','MarkerSize',30)

hold on
plot(7,maximum_vec_7,'r.','MarkerSize',30)
```

```
hold on
plot(7,average_vec_7,'c.','MarkerSize',30)
hold on
plot(7,minimum_vec_7,'b.','MarkerSize',30)

hold on
plot(6,maximum_vec_6,'r.','MarkerSize',30)
hold on
plot(6,average_vec_6,'c.','MarkerSize',30)
hold on
plot(6,minimum_vec_6,'b.','MarkerSize',30)

legend('Maximum error','Average error', 'Minimum error')
xlabel('Number of satellites')
ylabel('Error [km]')

xlim([5 10])
title('How increase in nr. of satellites affects the max/average/min error')


%% 10
close all; clear all; clc;

c = 299792.458; %speed of light
x0 = [0;0;6370;0]; % starting point is the north pole
tol = 10^(-3);
n = 4; %Four satellites
P = 26570; %Constant altitude of the satellites

angle = linspace(0,pi/2,50)';
theta = [0 1.7*pi 1.1*pi pi/2];


[x,y,z] = sphere; % Make unit sphere
radius = 6370; % Scale to desire radius.
x = x * radius;
y = y * radius;
z = z * radius;
offset = 0;%6370; % Translate sphere to new location.
surf(x+offset,y+offset,z+offset)  % Plot as surface.
xlabel('X', 'FontSize', 20); % Label axes.
ylabel('Y', 'FontSize', 20);
zlabel('Z', 'FontSize', 20);
axis equal;
hold on

total_err = [];
pos_new=[];


for i = 1:43

    phi(i,:) = [angle(i,:) angle(mod(i+2,50)+1,:) angle(mod(i+4,50)+1,:) angle(mod(i+6,50)+1,:)];

    [A B C R t] = distance_and_time(P,phi(i,:),theta,c,n);

    phi2(i,:) = [angle(i,:)+10^(-8) angle(mod(i+2,50)+1,:)-10^(-8) angle(mod(i+4,50)+1,:)-10^(-8)
    ↪ angle(mod(i+6,50)+1,:)+10^(-8)]; %IF I DECREASE THE SKEKKJA I GET NOT AS LINEAR DATA
    [A2 B2 C2 R2 t2] = distance_and_time(P,phi2(i,:),theta,c,n);
```

```
    A_loc(i,:)=A2;
    B_loc(i,:)=B2;
    C_loc(i,:)=C2;

    [numRows,numCols] = size(A_loc);
    if numRows == 1
        hold on
        plot3(A_loc(i,1),B_loc(i,1),C_loc(i,1), 'r.', 'MarkerSize',8)
        hold on
        plot3(A_loc(i,2),B_loc(i,2),C_loc(i,2), 'r.', 'MarkerSize',8)
        hold on
        plot3(A_loc(i,3),B_loc(i,3),C_loc(i,3), 'r.', 'MarkerSize',8)
        hold on
        plot3(A_loc(i,4),B_loc(i,4),C_loc(i,4), 'r.', 'MarkerSize',8)
        hold on
    else
        hold on
        plot3(A_loc(i,1),B_loc(i,1),C_loc(i,1), 'b.', 'MarkerSize',5)
        hold on
        plot3(A_loc(i,2),B_loc(i,2),C_loc(i,2), 'b.', 'MarkerSize',5)
        hold on
        plot3(A_loc(i,3),B_loc(i,3),C_loc(i,3), 'b.', 'MarkerSize',5)
        hold on
        plot3(A_loc(i,4),B_loc(i,4),C_loc(i,4), 'b.', 'MarkerSize',5)
        hold on
    end


    format long
    pos_new(i,:) = newtonmult(x0,tol,A2,B2,C2,t,c,n);
    total_err(i) = norm(pos_new(i,:)-x0');

end

 legend('Earth','Initial position of satellites')

figure
%plot(1:43, total_err,'*')
plot(phi(1,1),total_err(1),'r*')
hold on
plot(phi(2:end,1),total_err(2:end),'b*')
xlabel('\phi [rad]')
ylabel('Errors [km]')
%title('Distribution of errors in respecto to the linear increase of \phi angle')


legend('Error in starting position', 'Errors in other positions')
```