The future of GPS is really fascinating.
Hofmann-Wellenhof et al (2nd ed.), p. 288

Mathematics 5610—Fall 2021 [−1−]

List of Contents

General Comments

0. Getting full credit on a problem does not guarantee that your formulas, or their implementation, will work in your programs!

1. We use ψ for latitude and λ for longitude. Why would you change to some other notation, particularly without stating that you are doing so?

2. Geographic coordinates are related to spherical coordinates, but they are not the same thing!

3. The fact that $\frac{y}{x} = \tan\lambda$ does not imply that $\lambda = \arctan\frac{y}{x}$. This is an egregious error. Several people fell into that trap.

4. This is a small LaTeX point. Use "\sin" instead of "sin", "\min" instead of "min", etc. This will get you the correct font and spacing for mathematical formulas.

5. Lower and upper case variables are distinct and not interchangeable!

6. Most computer trig functions assume angles are specified in radians. If you enter angles as (decimal) degrees you'll get the wrong answers. .

7. Most submissions in our class incorporated the rotation of the earth by adding the relevant angle to longitude and then readjusting by adding or subtracting suitable multiples of π if necessary. That works, but it's jarring because longitude actually does not change on a rotating planet. Instead incorporate rotation by multiplying vectors of cartesian coordinates with suitable rotation matrices and leave longitude alone. There are more details below. No offense, but rotating the earth by adding to longitude qualifies as a hack!

8. The use of first person singular is inappropriate in a joint paper. Instead of "I solved ..." write "We solved ...". If appropriate write "One of us ..." possibly followed by a name in parenthesis.

9. In an equation like

$$z = \arctan\frac{z}{\sqrt{x^2+y^2}} \tag{1}$$

[−1−] by Peter Alfeld, pa@math.utah.edu, 801-581-6842, JWB 127, TEX processing date: September 14, 2021. **Please let me know of any errors or inconsistencies you find in this document.**

you must explain what happens when the denominator is zero. Certainly there needs to be a provision in your program for that case.

10. Don't use numerical values in your answer or your programs. For example, write

$$\frac{2\pi}{s}$$

instead of $7.29211E-5$. If you must use numerical values keep in mind that for our term project you need about 16 decimal digits! Your software won't work if you tell it that $\pi = 3.14$!

Introduction

The primary numerical task in this semester project is the solution of certain systems of nonlinear equations. However, beyond that the project illustrates a recent technological development which I hope you will find interesting and motivating. To get everything to work in this project we will have to put many pieces together, and they have to work just right, as in most real life problems. The results of your work, a couple of programs, will need to work with the programs of your class mates and my own. When we reach that stage we will be able to say with assurance that we truly understand the problem and its solution.

The original US Global Positioning System[−2−] (GPS) consists of 24 satellites that orbit the earth and constantly transmit their position and the precise time of that transmission, in addition to other data. Commercially available receivers, (*Global Positioning Devices)* process that information and, using the differences in the runtimes of the satellite signals, compute their position and the precise time. The position can be expressed, e.g., as longitude, latitude, and altitude. Its accuracy depends on the type and quality of the device and its software, and other circumstances. It ranges from less than one centimeter to about 100 meters[−3−].

In this project, we will write two programs that simulate a much simplified version of this system.

GPS was developed originally for the US armed forces, but there are of course also civilian and scientific applications, many of which are still being discovered. As far as volume and resources involved, civilian applications now vastly outstrip military applications. The following is an incomplete list of applications I have read or heard about (or, in the case of the first three applications, actually practiced):

- Navigating in a car. This may be the largest civilian application at present. You can buy devices that will tell you where you are and what's near you, and give you oral turn by turn instructions on how to get to your destination. Many high end models now come with built-in GPS based navigation systems.
- Finding your way in the wilderness.
- Marine navigation.
- *Geocaching* is a high-tech treasure hunting game played throughout the world by adventure seekers equipped with GPS devices. This is a highly popular past time where people search for a cache established by others, given it's coordinates, remove a token item, and replace it with another. The basic idea is to locate hidden containers, called geocaches, outdoors and then share your experiences online. The official web site is http://www.geocaching.com/ .
- Electronically marking a spot (a good fishing place, a buried treasure or mine, man or item overboard) in a featureless area such as the ocean, a lake, a desert, or the Salt Flats.
- Aerial navigation, including precision approach and landing.

[−2−] Full Disclosure: GPS is a multibillion dollar industry. I have no commercial connections or interests in GPS. This project is strictly for scientific purposes, and specific GPS devices or publications are mentioned or shown only for illustration and information. I'm not endorsing or recommending any specific GPS devices.

[−3−] A meter is a little more than 3 feet. Most of the literature on GPS uses the metric system, and we follow that convention.

- Measuring continental drift and expansion.
- Automatic grading and paving in road construction.
- Telling blind people (via Braille or Audio) where they are.
- Steering a tractor planting vine plants along a suitable trajectory.
- Surveying with high accuracy.
- Controlling a fleet of vehicles (e.g., Police, Taxi, Truck or Bus companies).
- Transfer of extremely accurate time information.
- Installation of several GPS devices in a vehicle, like a ship or plane, and determination not just of position but also of attitude, roll, pitch, and yaw.
- By coupling a cell phone with a GPS device, in an emergency it can tell your location to the dispatcher even if you don't know it yourself. If the phone is installed in your car, and the car is stolen, you might be able to call your car and ask where it is.

  GPS is available anywhere on or near earth, 24 hours a day, in fog, space, or darkness. By making several measurements over time one can compute and display such quantities as speed, bearing, or estimated time of arrival. It's eerie to have your car give you detailed instructions on how to get where you are going, as you are going. Or, for example, you can transform a simple rental boat into a sophisticated yacht with compass and speedometer simply by placing a GPS device next to the wheel.

Sources of Information

- You can download my software to test yours. For details see

  `http://www.math.utah.edu/~pa/5610/tp`

  If you are interested in GPS and like to learn more, here are some places to get started:
- A standard text on GPS is B. Hofmann-Wellenhof, H. Lichtenegger, and J. Collins: *GPS, Theory and Practice*, Springer-Verlag Wien New York, 5th edition, 2001, ISBN 3211835342. This excellent but very technical book gives lots of details and references.
- Another standard detailed and technical text is Elliott D. Kaplan (ed.), Understanding GPS Principles and Applications, Artech House, Boston, London, 1996, ISBN 0890067937.
- There's a great deal of information on the web. The following Table shows the number of results found by a Google search for *GPS*.

May 2004	17,000,000
May 2005	58,600,000
August 2007	257,000,000
May 2008	407,000,000
August 2009	244,000,000
May 2012	1,330,000,000
May 2013	720,000,000
May 2014	298,000,000
May 2015	620,000,000
July 2017	258,000,000
July 2018	774,000,000
December 2019	1,760,000,000
January 2021	1,230,000,000
August 2021	1,180,000,000

Table: Number of Google results for *GPS*

- There is excellent information in a concise and readable form about the shape of the earth and its gravitational field in the Encyclopedia Britannica, particularly in the article *The Earth* in the macropedia.
- A useful text giving all kinds of mathematical formulas and information is the *VNR Concise Encyclopedia of Mathematics*. It's published by Van Nostrand Reinhold Company. My copy was published in 1975. The ISBN is 0442226462.

Note on notation

Notation can get quite cumbersome in this project. I use boldface letters to denote vectors in cartesian coordinates. The coordinates of these vectors themselves are lower case Roman letters that may be subscripted. For example, the vector $\mathbf{x}$ is usually denoted by

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{2}$$

but sometimes, for example in the discussion of Newton's method, by

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}. \tag{3}$$

Subscripts are often used with an S to denote correspondence to a satellite. for example, $\mathbf{x}_{S_i}$ denotes correspondence to the i-th of several satellites.

The Space Segment

The number and configuration of the orbiting satellites is evolving. We will consider a set of 24 satellites orbiting in six planes at an inclination of 55° to the equator at an altitude of about 20,200 km.

To help you get going on the project, sprinkled throughout this assignment are a total of 17 **exercises**. You will have to solve these exercises or closely related problems during the work on the project, so they collectively form our first home work in this class.

The Model

The model will consist of three modules, i.e., programs, the `vehicle`, the `satellite`, and the `receiver`. The `vehicle` generates positional data, the `satellite` converts those into data of the kind processed by GPS receivers, and the `receiver` converts these back into positional data. The three modules are piped together in the standard Unix fashion:

```
vehicle | satellite | receiver
```
(4)

If all goes well the standard output of `receiver` will equal (almost) the standard output of `vehicle`. Your job is to write `satellite` and `receiver` (and perhaps a rudimentary `vehicle` for testing purposes).

The model depends on a set of data provided in a file called

```
data.dat
```

which is read by `satellite`. The beginning part of that file is also read by `receiver`. Figure 1 summarizes the model. Arrows indicate data flow.

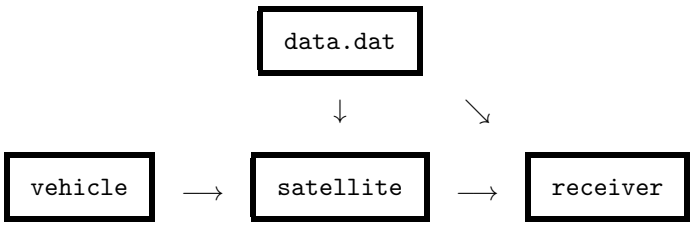

Figure 1: The Model

Following are some of the assumptions we'll make to construct our model. Numerical values are given here for illustration. They will be actually read from `data.dat` and the precise numerical values are subject to change (with appropriate notification). You or I may also change those parameters for the purpose of testing our programs.

1. The Earth is perfectly spherical and has a radius R of

$$R = 6,367,444.50 \text{ m}. \tag{5}$$

2. The Earth turns at a constant rate and completes one revolution in one sidereal day[−4−] of

$$s = 86,164.09 \text{ seconds}. \tag{6}$$

3. The satellites move at a constant speed in perfectly circular orbits at an altitude of

$$h = 20,200,000 \text{ m} \tag{7}$$

and with an orbital period

$$p = \frac{s}{2} \tag{8}$$

of exactly half a sidereal day.

4. The satellites are evenly spaced in groups of four in six planes, each of which is inclined 55° to the equator. According to Hofmann-Wellenhof et al this provides global coverage of four to eight satellites being more than 15° above the horizon at all times and in all places on earth.

Coordinate Systems

We use two coordinate systems: longitude, latitude and altitude to express positions on and near earth, and a cartesian coordinate system to describe the space in which the satellites orbit and the earth rotates. Like commercial global positioning devices, we express angles in degrees, minutes of degree, and seconds of degree. The two coordinate systems are linked by the following conventions:

The North Pole is located at $(0, 0, R)$, the South Pole at $(0, 0, -R)$, (assuming both have altitude 0, which is definitely wrong for the South Pole). At time 0 the point **O** of zero longitude, latitude, and altitude, is at (R,0,0), and the Earth rotates (west to east) once in a sidereal day.

Exercise 1: Find a formula that describes the trajectory of the point **O** in cartesian coordinates as a function of time.

Solution: Let

$$\mathbf{O} = \mathbf{O}(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} \tag{9}$$

Since the earth rotates around the z-axis, **O** moves in a circle of radius R in the (x, y)-plane at a periodicity of s, in a counterclockwise direction as viewed from the North Pole (at a speed we consider constant). That trajectory is described by the parametric representation

$$\boxed{\mathbf{O}(t) = \begin{bmatrix} R\cos\dfrac{2\pi t}{s} \\ R\sin\dfrac{2\pi t}{s} \\ 0 \end{bmatrix}.} \tag{10}$$

[−4−] A *sidereal day* is the time required for a complete rotation of the earth in reference to a fixed star. It differs from a standard *solar day* of 24 hours because in the course of a day the earth not only rotates, but also changes position in its orbit around the Sun. Hence the sun changes its position in the sky and the earth needs to rotate an additional 4 minutes or so to catch up with the sun before it is noon again.

This assumes that angles are measured in radians. You may be thinking about them as expressed in degrees, but most likely your programming language will consider them specified in radians.

The Programs

As stated above, our model will consist of three programs:

1. **The Vehicle.** This program, which you can download from our web page, produces a stream of data sets (to standard output) each of which has the form:

$$t_V \quad \psi_d \quad \psi_m \quad \psi_s \quad \mathbf{NS} \quad \lambda_d \quad \lambda_m \quad \lambda_s \quad \mathbf{EW} \quad h \tag{11}$$

where t_V denotes Universal time[−5−] in seconds, ψ_d, ψ_m, ψ_s is latitude in degrees, minutes, and seconds, and, similarly, λ_d, λ_m, λ_s is longitude in degrees, minutes, and seconds. More specifically,

t_V is a real number given to an accuracy of 10^{-2} seconds ranging from 0 to 10^6. This is the time at which the `vehicle` is at the specified position.

ψ_d is an integer ranging from 0° (i.e., the Equator) to +90° (i.e., the North or South Pole).

ψ_m is an integer ranging from 0 to 59 minutes of degree.

ψ_s is a real number ranging from 0 to 59.9999 seconds of degree. It should be given to an accuracy of 10^{-2} (which corresponds to an accuracy of about a foot).

NS is an integer that is +1 North of the equator and −1 South of the equator.

λ_d is an integer ranging from 0° (i.e., the meridian of Greenwich) to 180 (i.e., 180 degrees east, or west, the *date line*).

λ_m is an integer ranging from 0 to 59 minutes of degree.

λ_s is a real number ranging from 0 to 59.9999 seconds of degree, given to the same accuracy as ψ_s.

EW is an integer that is +1 east of Greenwich and −1 west of Greenwich.

h is a real number giving the altitude in meters, to an accuracy of 1cm.

For example, according to my Magellan Trailblazer the street light labeled B12[−6−] in front of the South Window of my office (at time t) is located at:

$$t \quad 40 \quad 45 \quad 55.0 \quad 1 \quad 111 \quad 50 \quad 58.0 \quad -1 \quad 1372.00 \tag{12}$$

i.e., at latitude 40° 45' 55" North, longitude 111° 50' 58" West, and an altitude of 1372 m.

[−5−] The mean solar time of the Greenwich meridian (0° longitude). Universal time replaced the designation Greenwich mean time in 1928; it is now used to denote the solar time when an accuracy of about 1 second suffices. In 1955 the International Astronomical Union defined several categories of Universal Time of successively increasing accuracy. UT0 represents the initial values of Universal Time obtained by optical observations of star transits at various astronomical observatories. These values differ slightly from each other because of the effects of polar motion. UT1, which gives the precise angular coordinate of the Earth about its spin axis, is obtained by correcting UT0 for the effects of polar motion. Finally, an empirical correction to take account of annual changes in the Earth's speed of rotation is added to UT1 to convert it into UT2. Coordinated Universal Time, the international basis of civil and scientific time, is obtained from an atomic clock that is adjusted so as to remain close to UT1; in this way, the solar time that is indicated by Universal time is kept in close coordination with atomic time. [Encyclopedia Britannica, 1992]. We will assume that all the times described here are identical and call them *Universal Time*.

[−6−] If you look around on campus you'll see that all street lights have been labeled with a letter and a number. On some lights, the labels have disappeared, or, as in the case of B12, obstructed by an attachment.

Exercise 2: Write a program that converts angles from degrees, minutes, and seconds to radians, and vice versa. Make sure your program does what it's supposed to do.

Solution: It's worthwhile to pause and think about the range of the angles that will occur in this project. Longitude and latitude range from 0° to 180° or 90°. But since in our coordinate calculations we may have to describe rotations of several days, or undo rotations leading to negative angles, the degrees in radians can assume essentially any values at all. To make conversion possible it is therefore necessary first to map the radians into the interval $(-\pi, \pi]$ (by adding or subtracting a suitable multiple of 2π), then extracting the sign defining the hemisphere, and finally turning the angle into degrees, minutes, and seconds. The java code `angles` listed below does the job.

```
1  public class angles {
2
3      /*-------------------------------------------------------------------------
4
5      Convert from degrees, minutes, and seconds to radians and vice versa.
6
7      Use as follows:
8
9      1. Convert from
10     int degrees, 0 <= degrees < 180,
11     int minutes, 0 <= minutes < 60,
12     double seconds, 0 <= seconds < 60,
13     boolean plus = true if angle is positive, false if it's negative
14     to double radians:
15
16     angles a = new angles(degrees, minutes, seconds, plus);
17     double radians = a.radians;
18
19     or use
20
21     double radians = angles.rad(degrees, minutes, seconds, plus);
22
23     2. Convert from double radians to degrees, minutes, seconds:
24
25     angles a = new angles(radians);
26     int degrees = a.degrees;
27     int minutes = a.minutes;
28     double seconds = a.seconds;
29     boolean plus = a.plus;
30
31     3. To test the program run it self contained:
32
33     java angles;
34
35     If all is well the message
36     "angles passed self test"
37     will be printed
38
39     -------------------------------------------------------------------------*/
40
41     double radians, seconds;
42     int degrees, minutes;
43     boolean plus;
44
45     final static double pi = 2*java.lang.Math.acos(0.0);
46     final static double twopi = 2.0*pi;
47
```

```
48      public angles (double rad) {
49          radians = rad;
50          if (radians < -pi || radians > pi) {
51              int n = (int) ((radians + pi) / twopi);
52              if (radians < 0.0) {
53                  n--;
54              }
55              radians = radians - n*twopi;
56          }
57          double theta = radians;
58          if (theta >= 0.0) {
59              plus = true;
60          }
61          else {
62              plus = false;
63              theta = -theta;
64          }
65          double degs = theta*360/twopi;
66          degrees = (int) degs;
67          degs = degs - degrees;
68          degs = 60*degs;
69          minutes = (int) degs;
70          degs = degs - minutes;
71          seconds = 60*degs;
72          if (java.lang.Math.abs(60-seconds) < 1.0E-3) {
73              seconds = 0;
74              minutes++;
75          }
76          if (minutes == 60) {
77              minutes = 0;
78              degrees++;
79          }
80          check();
81      }
82
83      public angles(int deg, int min, int sec, boolean pm) {
84          degrees = deg;
85          minutes = min;
86          seconds = sec;
87          plus = pm;
88          radians = rad(deg, min, sec, pm);
89          check();
90      }
91
92      static double rad(int deg, int min, double sec, int NS) {
93          if (NS == 1) {
94              return rad(deg, min, sec, true);
95          }
96          else if (NS == -1) {
97              return rad(deg, min, sec, false);
98          }
99          else {
100             return 0;
101         }
```

```
102     }
103
104     static double rad(int deg, int min, double sec, boolean pm) {
105         double Deg = (double) deg;
106         double Min = (double) min;
107         double result = twopi/360*(Deg + Min/60 + sec/60/60);
108         if (!pm) {
109             result = -result;
110         }
111         return result;
112     }
113
114     void report (String note) {
115         String s = note + radians + " radians = ";
116         if (!plus) {
117             s = s + " - ";
118         }
119         s = s + degrees + " " + minutes + " " + seconds;
120         System.out.println(s);
121     }
122
123     public static void main(String[] args) {
124         boolean correct = true;
125         boolean pm = false;
126         for (int k = 0; k < 2; k++) {
127             if (k == 0) {
128                 pm = true;
129             }
130             else {
131                 pm = false;
132             }
133             for (int deg = 0; deg < 180; deg++) {
134                 for (int min = 0; min < 60; min++) {
135                     for (int sec = 0; sec < 60; sec++) {
136                         angles a = new angles(deg, min, sec, pm);
137                         angles b = new angles(a.radians);
138                         if (b.degrees != deg || b.minutes != min
139                             || java.lang.Math.abs(((double) sec)-b.seconds) > 1.0E-6
140                             || (b.plus != pm && b.radians != 0.0)) {
141                             correct = false;
142                             System.out.println("Angles error: ");
143                             a.report("input:  ");
144                             b.report("output: ");
145                         }
146                     }
147                 }
148             }
149         }
150         if (correct) {
151             System.out.println("angles passed self test");
152         }
153     }
154
```

```
155     void check() {
156         if (degrees < 0 || degrees > 180
157             || minutes < 0 || minutes > 59
158             || seconds < 0.0 || seconds >= 60.0
159             || radians < -pi || radians > pi) {
160             report("angles error ");
161         }
162     }
163 }
```

Regarding the use of π and other constants in a program, you should not hard code them! This is error prone and intrinsically limits the accuracy of your constants. It is better to compute them, for example by the equation $\pi = 2\text{acos}(0)$. That way there is less risk of a typing error, and, more importantly, if some day you run your program on a more accurate computer, say in 512 bit arithmetic, your constant will automatically be available to that precision. The data file for our project is generated by a program that adheres to that principle. So if we do get more accurate floating point arithmetic some day I'll only need to rerun the program. You may think that using 163 lines of code to convert from radians to degrees and vice versa is a bit of an overkill, and perhaps you are right. My other programs are not nearly as verbose. My receiver has 298 lines of code, and my satellite has 219 lines of code. (The vehicle has 119 lines of code.) I have not kept track, but satellite and receiver codes from students in the past have ranged from slightly less than my numbers to vastly more.

For the following four exercises assume that t_V equals true Universal time and denote it by t.

Exercise 3: Find a formula that converts position as given in (11) at time $t = 0$ into cartesian coordinates.

Solution: It follows from elementary geometry (and can be found in any textbook dealing with the geometry of the sphere) that the coordinates of a point

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{13}$$

at latitude ψ, longitude λ, and altitude h on a sphere of radius R are given by

$$\boxed{\begin{aligned} x &= (R+h)\cos\psi\cos\lambda \\ y &= (R+h)\cos\psi\sin\lambda \\ z &= (R+h)\sin\psi. \end{aligned}} \tag{14}$$

The data given in (11) can be converted to λ and ψ by the formulas

$$\boxed{\begin{aligned} \psi &= 2\pi\mathbf{NS}\left(\frac{\psi_d}{360} + \frac{\psi_m}{360\times 60} + \frac{\psi_s}{360\times 60^2}\right) \\ \lambda &= 2\pi\mathbf{EW}\left(\frac{\lambda_d}{360} + \frac{\lambda_m}{360\times 60} + \frac{\lambda_s}{360\times 60^2}\right). \end{aligned}} \tag{15}$$

(This formula is also incorporated into the program `angles` listed above.) Some wrong answers can be detected by checking that

$$x^2 + y^2 + z^2 = (R + h)^2. \tag{16}$$

Indeed, this is true for (14) since

$$\cos^2\psi\cos^2\lambda + \cos^2\psi\sin^2\lambda + \sin^2\psi = 1. \tag{17}$$

Exercise 4: Find a formula that converts position and general time t as given in (11) into cartesian coordinates.

Solution: We have to rotate the point $\mathbf{x}$ around the z-axis by an angle

$$\boxed{\alpha = \frac{2\pi t}{s}} \tag{18}$$

Most people in class observed that the rotation of the earth changes only longitude on a hypothetically fixed earth, and therefore that angle of rotation simply has to be added to longitude. That will work but I prefer the following approach which leaves longitude attached to the rotating earth and simply rotates the vector in Cartesian coordinates. The approach also is useful in other parts of the project. A rotation by the angle α can be accomplished by multiplying with the matrix[−7−]

$$\boxed{R_3(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}.} \tag{19}$$

An explicit description of the trajectory of $\mathbf{x}$ is therefore

$$\boxed{\mathbf{x}(t) = R_3(\alpha)\mathbf{x}} \tag{20}$$

where α is given by (18) and $\mathbf{x}$ is given by (14) and (15).

Exercise 5: Find a formula that converts a position given in cartesian coordinates at time $t = 0$ into a position of the form (11).

Solution: Letting a point $\mathbf{x}$ be given by (13) we have to solve the equations (14) for λ, ψ and h. It's tempting to use the inverse tan function indiscriminately, but note that it returns an angle in the interval $\left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$. On the other hand, longitude is an angle in the interval $(-\pi, \pi)$. Expressing λ and ψ in radians, and assuming longitude is in the interval $[0, 2\pi]$, it's easy to

[−7−] This kind of matrix is known as a *Givens Rotation* in numerical analysis.

check that

$$
\begin{aligned}
&h = \sqrt{x^2+y^2+z^2} - R \\
&\psi = \begin{cases} \arctan \dfrac{z}{\sqrt{x^2+y^2}} & \text{for } x^2+y^2 \neq 0 \\ \frac{\pi}{2} & \text{for } x=y=0, \quad z>0 \\ \frac{-\pi}{2} & \text{for } x=y=0, \quad z<0 \end{cases} \\
&\lambda = \begin{cases} \arctan \frac{y}{x} & \text{for } x>0, \quad y>0 \\ \pi + \arctan \frac{y}{x} & \text{for } x<0, \\ 2\pi + \arctan \frac{y}{x} & \text{for } x>0, \quad y<0. \end{cases}
\end{aligned}
\tag{21}
$$

To transfer longitude to the interval (π, π), and to convert radians into degrees, minutes, and seconds we use the formulas in the program `angles` (or, of course, the program itself). Formulas for x, y and z different from (21) but giving the same values are possible. For example,

$$\psi = \arcsin\left(\frac{z}{R+h}\right). \tag{22}$$

A number of people stated that since $\frac{y}{x} = \tan\lambda$ we get that $\lambda = \arctan\frac{y}{x}$. It's not that simple since the tan function, as well as the other trig functions, is not actually invertible. We can see right away that there is a problem since longitude is between plus and minus pi, while the atan function returns an angle in the interval $(-\frac{\pi}{2}, \frac{\pi}{2})$. When programming the conversion we also need to guard against a division by zero (in the case $x = 0$.)

Exercise 6: Find a formula that converts general time t and a position given in cartesian coordinates into a position of the form (11).

Solution: The solution is just as before except that we have to undo first the rotation of Earth by the angle α given in (18). Thus we multiply $\mathbf{x}$ with $R_3(-\alpha)$ where R_3 is given in (19) and α is given in (18). Alternatively, we could compute h, λ, and ψ as in (21) and then replace λ by $\lambda - \alpha$. However, in that case we would then have to make sure (by adding a suitable multiple of 360° that the longitude measured in degrees is in the correct range).

Exercise 7: Find a formula that describes the trajectory of lamp post B12 in cartesian coordinates as a function of time.

Solution: We first find latitude and longitude of B12 using formulas (15). Thus

$$
\begin{aligned}
\psi &= 2\pi\left(\frac{40}{360} + \frac{45}{360\times 60} + \frac{55}{360\times 60^2}\right) \\
&= 0.7114883177 \\
\lambda &= -2\pi\left(\frac{111}{360} + \frac{50}{360\times 60} + \frac{58}{360\times 60^2}\right) \\
&= -1.9521410721.
\end{aligned}
\tag{23}
$$

Using these values, the altitude $h = 1372m$, and the value of R given in (5) we obtain

$$\mathbf{x} = \begin{bmatrix} -1795225.28 \\ -4477174.36 \\ 4158593.45 \end{bmatrix} \tag{24}$$

Substituting (24), (6), and (18) into (20) gives

$$\mathbf{x}(t) = \begin{bmatrix} \cos\dfrac{2\pi t}{86164.09} & -\sin\dfrac{2\pi t}{86164.09} & 0 \\ \sin\dfrac{2\pi t}{86164.09} & \cos\dfrac{2\pi t}{86164.09} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1795225.28 \\ -4477174.36 \\ 4158593.45 \end{bmatrix}. \tag{25}$$

Thus, just in case you were wondering, we are 4,158,593 m above the plane of the equator. Based on past experience with the Trailblazer this statement would be accurate to within about 100m if the earth were indeed perfectly spherical.

The `vehicle` should not produce impossible positions (like a latitude of 90° 59' 59"). For testing purposes we will use `vehicle`s that produce a stream of data corresponding, for example, to a walk from JWB to the Marriott Library, a hike up to Mount Olympus, or a flight from Salt Lake City to the North Pole. The data sets should be spaced at least 1 second (of time) apart.

In addition `vehicle` writes a copy of the standard input and the standard output into the file

`vehicle.log`.

2. The Satellite. This program reads data from `data.dat` and then reads the data generated by a `vehicle` and processes those data as follows:

A. It computes the position $\mathbf{x}_V$ of the vehicle in cartesian coordinates (measured in meters) at the time t_V.

B. For each satellite S that is above the horizon[−8−] at the vehicle's position it computes the time t_S at which a signal needs to be sent to reach the vehicle at time t_V and position $\mathbf{x}_V$, assuming the signal moves at the speed c of light in vacuum where

$$c = 2.99792458 \times 10^8 \text{m/s}. \tag{26}$$

The satellite needs to be above the horizon at time t_S, The satellite also computes the position $\mathbf{x}_S$ of the satellite at the time t_S It then writes the following data to standard output:

$$i_S \quad t_S \quad \mathbf{x}_S \tag{27}$$

where i_S is the reference number of the satellite (ranging from 0 to 23). The time t_S should be given to an accuracy of 10^{-11} seconds, and the position $\mathbf{x}_S$ should be given as a triple of cartesian coordinates with an accuracy of 1cm.

C. In addition, `satellite` writes a copy of the standard input and the standard output into the file

`satellite.log`.

It should terminate gracefully when the data stream from the satellite program is terminated.

Exercise 8: Given a point $\mathbf{x}$ on earth and a point $\mathbf{s}$ in space, both in cartesian coordinates, find a condition that tells you whether $\mathbf{s}$ as viewed from $\mathbf{x}$ is above the horizon.

[−8−] My own `receiver` will check this condition. I figured that assuming a transparent earth would be too much of a simplification. However, calculations in the actual GPS system are usually based on satellites at least 15° above the horizon.

Solution: Consider the function

$$f(\mathbf{z}) = \mathbf{x}^T\mathbf{z}. \tag{28}$$

This is a linear function which assumes constant values on planes in $\mathbb{R}^3$ that are perpendicular to $\mathbf{x}$. (If you don't believe this consider some real number α. The plane through $\alpha\mathbf{x}$ that is perpendicular to $\mathbf{x}$ is the set of all points $\mathbf{z}$ that can be written as

$$\mathbf{z} = \alpha\mathbf{x} + \mathbf{y} \tag{29}$$

where $\mathbf{y}$ is a vector perpendicular to $\mathbf{x}$, i.e.,

$$\mathbf{x}^T\mathbf{y} = 0. \tag{30}$$

Then clearly

$$f(\mathbf{z}) = \mathbf{x}^T\mathbf{z} = \mathbf{x}^T(\alpha\mathbf{x} + \mathbf{y}) = \alpha\mathbf{x}^T\mathbf{x} + \mathbf{x}^T\mathbf{y} = \alpha\mathbf{x}^T\mathbf{x} \tag{31}$$

is independent of $\mathbf{y}$.) In particular f assumes the value $\mathbf{x}^T\mathbf{x}$ everywhere in the plane tangent to the Earth at $\mathbf{x}$. At the origin, it assumes the value 0. Thus, for points above the tangent plane it assume values greater than $\mathbf{x}^T\mathbf{x}$. A point is above the horizon if it's on the side of the tangent plane away from the origin. Thus we obtain the extremely simple condition

$$\boxed{\mathbf{s} \text{ above horizon} \iff \mathbf{x}^T\mathbf{s} > \mathbf{x}^T\mathbf{x}.} \tag{32}$$

This derivation of (32) describes how I first thought of the problem. Several people in class came up with simpler more geometric explanations. In particular, the following argument is simpler: Draw the triangle of points $\mathbf{x}$, $\mathbf{s}$, and the origin. For $\mathbf{s}$ to be above the horizon, the angle made by $\mathbf{x}$ and $\mathbf{s} - \mathbf{x}$ must be *acute*, i.e., between zero and $\pi/2$. Recall that the inner product of two vectors $\mathbf{x}$ and $\mathbf{y}$ is given by

$$\mathbf{x}^T\mathbf{y} = \|x\|_2\|\mathbf{y}\|_2 \cos\theta \tag{33}$$

where θ is the angle made by $\mathbf{x}$ and $\mathbf{y}$. In our case we have to have

$$\mathbf{x}^T(\mathbf{s} - \mathbf{x}) > 0 \tag{34}$$

which is of course equivalent to (32). The calculation described here assumes that you are on a plane at altitude h. That's reasonable for someone actually on earth. However, if you were in a spaceship higher than the satellites, then according to our calculation all of them would be below the horizon which wouldn't make sense.

Benjamin Eagar in an earlier class came up with the following clever argument. It took me a few minutes to believe it. I paraphrase: Let s be the location of the satellite and x the location of the vehicle. Then $\|s - x\|$ is smallest when s is directly above x and larges when directly below x (on the other side of the earth). Because $\|s\|$ is fixed (the satellite moves in a circular orbit around the origin), Because $\|s\|$ is fixed, s is in the tangent plane to x only when

$$\|s - x\| = \sqrt{\|s\|^2 - \|x\|^2}$$

(by the Pythagorean Theorem, draw a picture). If

$$\|s - x\| < \sqrt{\|s\|^2 - \|x\|^2}$$

then the satellite is above the horizon, and if

$$\|s - x\| > \sqrt{\|s\|^2 - \|x\|^2}$$

then it is above the horizon.

Exercise 9: Discuss how to compute t_S and $\mathbf{x}_S$.

Solution: As we discussed in class, the iteration:

$$\begin{aligned} t_0 &= t_V \\ t_{k+1} &= t_V - \frac{\|\mathbf{x}_S(t_k) - \mathbf{x}_V\|_2}{c}, \quad k = 0, 1, 2, \ldots \end{aligned} \tag{35}$$

works fine. However, my program actually does use Newton's method, even though (35) seems more straightforward. We describe the orbit of the satellite as

$$\mathbf{x}_S(t) = (R+h)\left[\mathbf{u}_S \cos\left(\frac{2\pi t}{p} + \theta_S\right) + \mathbf{v}_S \sin\left(\frac{2\pi t}{p} + \theta_S\right)\right]. \tag{36}$$

Let $\mathbf{x}_V$ be the location of the vehicle (in Cartesian coordinates). We want to find t such that

$$\|\mathbf{x}_S(t) - \mathbf{x}_V\|_2 = c(t_V - t). \tag{37}$$

To avoid the square roots we write this as

$$f(t) = (\mathbf{x}_S(t) - \mathbf{x}_V)^T (\mathbf{x}_S(t) - \mathbf{x}_V) - c^2(t_V - t)^2 = 0. \tag{38}$$

This equation can be solved by Newton's method:

$$\begin{aligned} t_0 &= \text{given} \\ t_{k+1} &= t_k - \frac{f(t_k)}{f'(t_k)}, \qquad k = 0, 1, 2, \ldots, \quad \text{until convergence.} \end{aligned} \tag{39}$$

Clearly,

$$\begin{aligned} f'(t) &= 2(\mathbf{x}_S(t) - \mathbf{x}_V)^T \mathbf{x}_S'(t) + 2c^2(t_V - t) \\ &= \frac{4\pi(R+h)}{p}(\mathbf{x}_S(t) - \mathbf{x}_V)^T \left[-\mathbf{u}_S \sin\left(\frac{2\pi t}{p} + \theta_S\right) + \mathbf{v}_S \cos\left(\frac{2\pi t}{p} + \theta_S\right)\right] \\ &\qquad + 2c^2(t_V - t) \end{aligned} \tag{40}$$

As usual in Newton's method, we have to find a suitable starting value t_0. A good way to do this in this case is to pretend that at the time the satellite sends the signal it is at the position it will be when the vehicle receives the signal. Thus

$$t_0 = t_V - \frac{\|\mathbf{x}_S(t_V) - \mathbf{x}_V\|_2}{c}. \tag{41}$$

To get a spatial accuracy of 1 cm we stop the iteration when

$$|t_{k+1} - t_k| < \frac{0.01}{c} \approx 3.0 \times 10^{-11} \tag{42}$$

since light takes about 3.0×10^{-11} seconds to travel 1cm. With these choices, Newton's iteration usually converges in two steps.

---

3. The Receiver. This program reads the data written by `satellite` and converts them into position and time data in the same form as the data generated by the vehicle. If all goes well those data should closely approximate the data sent by `vehicle`. Note that `satellite` does not transfer t_V. This has to be reconstructed by `receiver`, thus simulating the transfer of

precise time. In addition the program writes a copy of the standard input and the standard output into the file

`receiver.log`.

It should terminate gracefully when the data stream from the satellite program is terminated.

Letting $\mathbf{x}_v$ denote the position at which the vehicle receives the satellite signal, t_V the time that it receives the signal, and, similarly, $\mathbf{x}_S$ and t_S the position and time at which the satellite sends the signal, the basic equation for our system is

$$\|\mathbf{x}_V - \mathbf{x}_S\| = c(t_V - t_S). \tag{43}$$

In this equation we know t_S and $\mathbf{x}_S$, and we don't know t_V and $\mathbf{x}_V$. We have one such equation for each satellite from which we receive a signal.

The time t_V enters these equations linearly, and can be eliminated. It also plays a different role than the coordinates of $\mathbf{x}_V$. It needs to be computed to a much higher accuracy than the location. So it is indeed best to eliminate it. Suppose we have equations of the form (43) for two satellites S_1 and S_2. Taking the difference gives the equation

$$\|\mathbf{x}_V - \mathbf{x}_{S_1}\| - \|\mathbf{x}_V - \mathbf{x}_{S_2}\| = c(t_{S_2} - t_{S_1}). \tag{44}$$

Exercise 10: Suppose you have data of the from (27) from 4 satellites. Write down a set of four equations whose solutions are the position of the vehicle in cartesian coordinates, and t_V.

Solution: Denote the location of the `vehicle` by $\mathbf{x}$ (in cartesian coordinates). The distance between the `vehicle` and the `satellite` is the speed of light multiplied with the run time of the signal. That run time equals $t_V - t_S$, where t_V is the unknown time at which the vehicle receives the signal. Let's denote the four data sets by

$$i_j \quad t_{S_j} \quad \mathbf{x}_{S_j}, \quad j = 1, 2, 3, 4 \tag{45}$$

we thus obtain the four equations

$$\boxed{\|\mathbf{x}_{S_i} - \mathbf{x}\|_2 = c\,(t_V - t_{S_i})\,, \quad i = 1, 2, 3, 4,} \tag{46}$$

where, as usual for $\mathbf{x} = [\,x_1 \quad x_2 \quad x_3\,]^T$,

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{3} x_i^2}. \tag{47}$$

Of course, once we have the location in cartesian coordinates we still have to convert them into longitude, latitude and altitude, taking into account the rotation of the earth. It's worthwhile to note that the unknown time t_V can be eliminated from (46) simply by taking differences. Thus we can obtain for example the following system of three equations in the three coordinates making up the unknown location $\mathbf{x}$:

$$\boxed{\begin{aligned} \|\mathbf{x}_{S_2} - \mathbf{x}\|_2 - \|\mathbf{x}_{S_1} - \mathbf{x}\|_2 &= c\,(t_{S_1} - t_{S_2}) \\ \|\mathbf{x}_{S_3} - \mathbf{x}\|_2 - \|\mathbf{x}_{S_2} - \mathbf{x}\|_2 &= c\,(t_{S_2} - t_{S_3}) \\ \|\mathbf{x}_{S_4} - \mathbf{x}\|_2 - \|\mathbf{x}_{S_3} - \mathbf{x}\|_2 &= c\,(t_{S_3} - t_{S_4}) \end{aligned}} \tag{48}$$

Apporva Pedgaonkar, **Bao Le**, and **Kaden Dvorak** in our class had a very interesting idea. Simply square the two sides of (46). This gives the equations

$$\boxed{\|\mathbf{x}_{S_i} - \mathbf{x}\|_2^2 = c^2 \left(t_V - t_{S_i}\right)^2, \quad i = 1, 2, 3, 4,} \tag{49}$$

That set of equations is **polynomial**, which greatly simplifies the derivatives required for Newton's method. On the other hand, you can no longer easily eliminate t_v, and so you can expect numerical problems with this approach. Apporva, Bao, and Kaden, if you stick with this approach, let us know how this works for you! A similar idea can be applied to the difference equations, after eliminating t_v, although in that case the resulting equations are not polynomial.

Usually you will have data from more than four satellites. To use the above approach one would have to choose four specific ones of those satellites. This is in itself a nontrivial problem. A better way is to use all observations and to think of the (possibly overdetermined) nonlinear system of equations

$$F(\mathbf{x}) = 0 \tag{50}$$

as a nonlinear Least Squares problem

$$f(\mathbf{x}) = F(\mathbf{x})^T F(\mathbf{x}) = \min. \tag{51}$$

The least squares problem can be solved by setting the gradient of f in (51) to zero, which gives rise to another system of four equations in four unknowns.

If m is the number of satellites being received, I recommend that you form $m-1$ equations of the form (44) and apply the Least Squares approach to find $\mathbf{x}_V$. Once you have $\mathbf{x}_V$ you can compute t_V.

Exercise 11: Suppose you have data of the form (45) from more than 4 satellites. Write down a least squares problem whose solution the position of the vehicle in cartesian coordinates, and t_V.

Solution: Suppose we have m data, with $m \geq 4$, and we denote them as in (45) with $i = 1, 2, \ldots, m$. We obtain the Least Squares problem by taking the difference of the left and right hand sides in (46), squaring them, and summing them:

$$\sum_{i=1}^{m} \left[\|\mathbf{x}_{S_i} - \mathbf{x}\|_2 - c\left(t_V - t_{S_i}\right)\right]^2 = \min. \tag{52}$$

Alternatively, we could make the equations corresponding to (48) into a Least Squares problem:

$$\sum_{i=1}^{m-1} \left[\|\mathbf{x}_{S_{i+1}} - \mathbf{x}\|_2 - \|\mathbf{x}_{S_i} - \mathbf{x}\|_2 - c\left(t_{S_i} - t_{S_{i+1}}\right)\right]^2 = \min. \tag{53}$$

Exercise 12: Think about the *ground track* of satellite 1, i.e., the position in geographic coordinates directly underneath the satellite on the surface of the earth, as a function of time. Do you notice anything particular? What is the significance of the orbital period being exactly one half sidereal day?

Solution: An easy trap to fall into here is to insist on expressing the ground track immediately in terms of latitude and longitude. The problem is much simplified if it is expressed in terms of cartesian coordinates that rotate with the earth. From those one can easily compute longitude and latitude as given by (21). Consider first the case of a non-rotating earth. In

that case the ground track ($h = 0$) of a satellite with orbital period $p = \frac{s}{2}$ and phase 0 is obtained from (36):

$$\mathbf{x}(t) = R\left[\mathbf{u}\cos\frac{4\pi t}{s} + \mathbf{v}\sin\frac{4\pi t}{s}\right]. \tag{54}$$

However, earth does rotate, and so the point underneath the satellite is not the $\mathbf{x}(t)$ given in (54) but the one that was in the position of $\mathbf{x}(t)$ t seconds earlier. That point can be obtained by multiplying $\mathbf{x}(t)$ with the rotation matrix $R_3(\alpha)$ given in (19), with

$$\alpha = -\frac{2\pi t}{s}. \tag{55}$$

Thus in terms of a cartesian coordinate system that rotates with earth the ground track $\mathbf{y}(t)$ of the satellite is given by

$$\mathbf{y}(t) = R_3\left(-\frac{2\pi t}{s}\right) R\left[\mathbf{u}\cos\frac{4\pi t}{s} + \mathbf{v}\sin\frac{4\pi t}{s}\right]. \tag{56}$$

This needs to be converted to geographic coordinates using the formulas from question 6. Note that the cartesian coordinates are periodic of periodicity s. So at time s, the satellite is in the same spot as at time 0. At that its projection onto the earth is also the same as it was at time 0. The ground track is therefore the same every single (sidereal) day. The same holds for the other satellites so that the constellations of satellites in the sky repeat every sidereal day.

Mathematical Approach

Mathematically, we have four unknowns: three coordinates of the location, and time. Each satellite signal provides a range $c(t_S - t_V)$ where we don't know t_V. Given four data we should be able to compute position and time. So `receiver` has to solve a system of four nonlinear equations for the four unknowns. In addition the programs will have to do some coordinate conversions.

The natural way (why?) to solve the nonlinear equations arising in this project is by Newton's Method (for systems of nonlinear equations). We will discuss it in detail in class. However, for reference here is a preliminary description: Let $F : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ be a given function, and suppose we want to solve the nonlinear system of equations

$$F(\mathbf{x}) = 0. \tag{57}$$

Let J denote the *Jacobian Matrix* of F. Thus the (i, j) entry of J is the partial derivative of the i-th component of F with respect to the j-th component of $\mathbf{x}$, i.e.,

$$J(\mathbf{x}) = \left[\frac{\partial F_i}{\partial x_j}(\mathbf{x})\right]_{i,j=1,\ldots,n.} \tag{58}$$

Given a starting point $\mathbf{x}^{(0)}$, Newton's method proceeds iteratively by defining

$$\begin{aligned} &\text{for} \quad k = 0, 1, 2, \cdots, \quad \text{until satisfied} \\ &\qquad \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \left[J\left(\mathbf{x}^{(k)}\right)\right]^{(-1)} F\left(\mathbf{x}^{(k)}\right). \end{aligned} \tag{59}$$

Of course, we never invert a matrix. Instead, your program should do something like this

$$\boxed{\begin{aligned} &\text{For } k = 0, 1, 2, \cdots \text{ until satisfied :} \\ &\qquad \text{Solve} \quad J\left(\mathbf{x}^{(k)}\right)\mathbf{s}^{(k)} = -F\left(\mathbf{x}^{(k)}\right) \\ &\qquad\quad \text{Let} \quad \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{s}^{(k)} \end{aligned}} \tag{60}$$

The performance of Newton's method is very sensitive to the choice of the starting point $\mathbf{x}^{(0)}$. As always, the precise choice of that point depends on the problem. For our purposes you may assume that at least initially you are operating your GPS device in the vicinity of Salt Lake City, i.e., you can use (12) as a starting point.

Exercise 13: Find a precise description of Newton's method as it is applied to the nonlinear system obtained by processing data from 4 satellites, as derived in an earlier exercise. Your answer should include an explicit specification of the derivatives involved.

Solution: We could apply Newton's method to the equations (46) or (48). To be definite consider the equations (48). Also, the notation is becoming burdensome since we will have to consider entries of a vector $\mathbf{x}_{S_i}$ that is already doubly subscripted. So let's write

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad \text{and} \quad \mathbf{x}_{S_i} = \begin{bmatrix} x_{S_i} \\ y_{S_i} \\ z_{S_i} \end{bmatrix}. \tag{61}$$

We first have to phrase the equations (48) in homogeneous form:

$$F(\mathbf{x}) = \begin{bmatrix} \|\mathbf{x}_{S_2} - \mathbf{x}\|_2 - \|\mathbf{x}_{S_1} - \mathbf{x}\|_2 - c\,(t_{S_1} - t_{S_2}) \\ \|\mathbf{x}_{S_3} - \mathbf{x}\|_2 - \|\mathbf{x}_{S_2} - \mathbf{x}\|_2 - c\,(t_{S_2} - t_{S_3}) \\ \|\mathbf{x}_{S_4} - \mathbf{x}\|_2 - \|\mathbf{x}_{S_3} - \mathbf{x}\|_2 - c\,(t_{S_3} - t_{S_4}) \end{bmatrix} = 0. \tag{62}$$

The Jacobian $J(\mathbf{x})$ is the matrix of partial derivatives of the components of f with respect to x, y, and z. To compute these first note that for any vector

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \tag{63}$$

$$\begin{aligned} \frac{\partial}{\partial x_i}\|y - x\|_2 &= \frac{\partial}{\partial x_i}\sqrt{\sum_{k=1}^{3}(y_k - x_k)^2} \\ &= -\frac{1}{2}\frac{1}{\|y - x\|_2}(y_i - x_i) \\ &= -\frac{y_i - x_i}{\|y - x\|_2} \end{aligned} \tag{64}$$

where for the moment we write

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{65}$$

Applying this result and the notation (61) to F defined in (62) we see that

$$J(\mathbf{x}) = \begin{bmatrix} \frac{x_{S_1}-x}{\|x_{S_1}-\mathbf{x}\|_2} - \frac{x_{S_2}-x}{\|x_{S_2}-\mathbf{x}\|_2} & \frac{y_{S_1}-y}{\|x_{S_1}-\mathbf{x}\|_2} - \frac{y_{S_2}-y}{\|x_{S_2}-\mathbf{x}\|_2} & \frac{z_{S_1}-z}{\|x_{S_1}-\mathbf{x}\|_2} - \frac{z_{S_2}-z}{\|x_{S_2}-\mathbf{x}\|_2} \\ \frac{x_{S_2}-x}{\|x_{S_2}-\mathbf{x}\|_2} - \frac{x_{S_3}-x}{\|x_{S_3}-\mathbf{x}\|_2} & \frac{y_{S_2}-y}{\|x_{S_2}-\mathbf{x}\|_2} - \frac{y_{S_3}-y}{\|x_{S_3}-\mathbf{x}\|_2} & \frac{z_{S_2}-z}{\|x_{S_2}-\mathbf{x}\|_2} - \frac{z_{S_3}-z}{\|x_{S_3}-\mathbf{x}\|_2} \\ \frac{x_{S_3}-x}{\|x_{S_3}-\mathbf{x}\|_2} - \frac{x_{S_4}-x}{\|x_{S_4}-\mathbf{x}\|_2} & \frac{y_{S_3}-y}{\|x_{S_3}-\mathbf{x}\|_2} - \frac{y_{S_4}-y}{\|x_{S_4}-\mathbf{x}\|_2} & \frac{z_{S_3}-z}{\|x_{S_3}-\mathbf{x}\|_2} - \frac{z_{S_4}-z}{\|x_{S_4}-\mathbf{x}\|_2} \end{bmatrix}. \tag{66}$$

Exercise 14: Similarly, find Newton's method for the nonlinear system obtained from the least squares approach. Again, your answer should include an explicit specification of the derivatives involved.

Solution: Let's first consider how to apply Newton's method to a general minimization problem. Suppose we want to solve

$$f(\mathbf{x}) = \min \tag{67}$$

where f is a scalar valued function of an n-vector $\mathbf{x}$. At the solution the gradient of f vanishes, i.e.,

$$F(\mathbf{x}) = \nabla f(\mathbf{x}) = [\, \tfrac{\partial f}{\partial x_i}(\mathbf{x}) \,]_{i=1,2,\ldots,n} = 0. \tag{68}$$

This is a nonlinear system just like (57), with F being the vector of partial derivatives of f. Therefore, J is the matrix of second partial derivatives of f. Partial derivatives commute (if all derivatives involved are continuous) and $J(\mathbf{x})$ will therefore be symmetric in this particular case. As for the case of data from four satellites we can choose to eliminate t_V prior to the minimization, or leave it as a parameter. For simplicity let us eliminate it, i.e., we minimize the function

$$f(\mathbf{x}) = \sum_{i=1}^{m-1} \left[\|\mathbf{x}_{S_{i+1}} - \mathbf{x}\|_2 - \|\mathbf{x}_{S_i} - \mathbf{x}\|_2 - c\left(t_{S_i} - t_{S_{i+1}}\right) \right]^2 \tag{69}$$

given in (53). To run Newton's method we have to compute the first and second order partial derivatives of f. This is straightforward, if tedious. To obtain compact notation and code it helps to give names to individual ingredients. So we set

$$\begin{aligned}
N_i &= \|\mathbf{x}_{S_i} - \mathbf{x}\|_2, \\
A_i &= N_{i+1} - N_i - c\left(t_{S_i} - t_{S_{i+1}}\right) \\
X_i &= \frac{\partial A_i}{\partial x} = -\frac{x_{S_{i+1}} - x}{N_{i+1}} + \frac{x_{S_i} - x}{N_i} \\
Y_i &= \frac{\partial A_i}{\partial y} = -\frac{y_{S_{i+1}} - y}{N_{i+1}} + \frac{y_{S_i} - y}{N_i} \\
Z_i &= \frac{\partial A_i}{\partial z} = -\frac{z_{S_{i+1}} - z}{N_{i+1}} + \frac{z_{S_i} - z}{N_i}
\end{aligned} \tag{70}$$

Hence

$$\begin{aligned}
f(\mathbf{x}) &= \sum_{i=1}^{m-1} A_i^2 \\
\frac{\partial f}{\partial x}(\mathbf{x}) &= 2 \sum_{i=1}^{m-1} A_i X_i \\
\frac{\partial f}{\partial y}(\mathbf{x}) &= 2 \sum_{i=1}^{m-1} A_i Y_i \\
\frac{\partial f}{\partial z}(\mathbf{x}) &= 2 \sum_{i=1}^{m-1} A_i Z_i
\end{aligned} \tag{71}$$

and

$$\begin{aligned}
\frac{\partial^2 f}{\partial x^2}(\mathbf{x}) &= 2\sum_{i=1}^{m-1}\left(X_i^2 + A_i\frac{\partial X_i}{\partial x}\right)\\
\frac{\partial^2 f}{\partial x\partial y}(\mathbf{x}) = \frac{\partial^2 f}{\partial y\partial x}(\mathbf{x}) &= 2\sum_{i=1}^{m-1}\left(X_iY_i + A_i\frac{\partial X_i}{\partial y}\right)\\
\frac{\partial^2 f}{\partial x\partial z}(\mathbf{x}) = \frac{\partial^2 f}{\partial z\partial x}(\mathbf{x}) &= 2\sum_{i=1}^{m-1}\left(X_iZ_i + A_i\frac{\partial X_i}{\partial z}\right)\\
\frac{\partial^2 f}{\partial y^2}(\mathbf{x}) &= 2\sum_{i=1}^{m-1}\left(Y_i^2 + A_i\frac{\partial Y_i}{\partial y}\right)\\
\frac{\partial^2 f}{\partial y\partial z}(\mathbf{x}) = \frac{\partial^2 f}{\partial z\partial y}(\mathbf{x}) &= 2\sum_{i=1}^{m-1}\left(Y_iZ_i + A_i\frac{\partial Y_i}{\partial z}\right)\\
\frac{\partial^2 f}{\partial z^2}(\mathbf{x}) &= 2\sum_{i=1}^{m-1}\left(Z_i^2 + A_i\frac{\partial Z_i}{\partial z}\right)
\end{aligned} \tag{72}$$

We still have to compute the remaining partial derivatives in (72):

$$\begin{aligned}
\frac{\partial X_i}{\partial x} &= \frac{N_{i+1}^2 - \left(x_{S_{i+1}} - x\right)^2}{N_{i+1}^3} - \frac{N_i^2 - \left(x_{S_i} - x\right)^2}{N_i^3}\\
\frac{\partial Y_i}{\partial x} = \frac{\partial X_i}{\partial y} &= -\frac{\left(y_{S_{i+1}} - y\right)\left(x_{S_{i+1}} - x\right)}{N_{i+1}^3} + \frac{\left(y_{S_i} - y\right)\left(x_{S_i} - x\right)}{N_i^3}\\
\frac{\partial X_i}{\partial z} = \frac{\partial Z_i}{\partial x} &= -\frac{\left(x_{S_{i+1}} - x\right)\left(z_{S_{i+1}} - z\right)}{N_{i+1}^3} + \frac{\left(x_{S_i} - x\right)\left(z_{S_i} - z\right)}{N_i^3}\\
\frac{\partial Y_i}{\partial y} &= \frac{N_{i+1}^2 - \left(y_{S_{i+1}} - y\right)^2}{N_{i+1}^3} - \frac{N_i^2 - \left(y_{S_i} - y\right)^2}{N_i^3}\\
\frac{\partial Y_i}{\partial z} = \frac{\partial Z_i}{\partial y} &= -\frac{\left(y_{S_{i+1}} - y\right)\left(z_{S_{i+1}} - z\right)}{N_{i+1}^3} + \frac{\left(y_{S_i} - y\right)\left(z_{S_i} - z\right)}{N_i^3}\\
\frac{\partial Z_i}{\partial z} &= \frac{N_{i+1}^2 - \left(z_{S_{i+1}} - z\right)^2}{N_{i+1}^3} - \frac{N_i^2 - \left(z_{S_i} - z\right)^2}{N_i^3}
\end{aligned} \tag{73}$$

A common problem with nonlinear equations is non-existence of a solution, or existence of multiple solutions. For our project, for example, if the difference in run times of two satellite signals exceeds the time corresponding to the maximum distance of the satellites then there is no solution. On the other hand, the way we set up the problem there should always be a solution since we start with a well defined position. Non-existence of a solution would therefore indicate a programming error of some sort. It is also possible to have several solutions. For simplicity, consider a situation where you are given the *distances* (i.e., the receiver has an exact clock) from three satellites at known positions. So your location is on the intersection of three spheres (with the satellites as their centers and the known distances as their radii). Two spheres intersect in a circle, and the third sphere intersects the circle in two points. There are thus two solutions of the nonlinear equations. Which should we decide on as our current location? In practice, global positioning devices use the following criteria: closeness to the previously computed position, closeness to the surface of the earth, and slow motion.

Exercise 15: Think about the number of solutions obtained by analyzing four satellite signals with an unknown vehicle time t_V. This is an open ended question that will not be graded!

Solution: I once consulted an expert in Algebraic Geometry on this question, and he could not answer it. Here are a few examples and comments that show what might happen.

1. Suppose all satellites are at the same distance. This would mean that we are in the center of the earth, but it would not be an unreasonable approximation if the radius of the satellite orbits is much larger than the radius of earth. In that case all differences in runtimes are zero, and so each pair tells us a plane that contains our solution. If the satellites are in a generic position then 2 pairs (3 satellites) determine a line, and four determine our location uniquely.
2. However, the satellites may be in a special configuration for which the above argument would not work. For example, let S be the sphere centered at the origin with radius $R + h$. Intersect S with a plane that is perpendicular to the line from the origin through the location of the vehicle. You obtain a circle. Now consider any number of satellites in that circle. They are all at the same distance from the vehicle, and the planes corresponding to pairs of satellites all intersect in the axis of the cone. Thus there are infinitely many solutions. Geographically speaking we can determine longitude and latitude but not altitude.
3. Consider the case that the vehicle knows the actual time. In that case each satellite defines a sphere centered at the satellite such that the vehicle is on the sphere. Two such spheres determine a circle. A third sphere intersects the circle in two points. The vehicle is at one of those two points.
4. It's hard to visualize when the previous idea is applied to differences in run times. A pair of satellites determines a hyperboloid. Two hyperboloids (two pairs, three satellites) determine a curve. A third hyperboloid (three pairs, 4 satellites) can intersect that curve in one or more points. But I find this very hard to visualize, and I don't know the maximum possible number of solutions.
5. Note that the question of the number of solutions is different from the question of which solution do we find? At the beginning of a trip we know that we are close to B12 and we will find our location. Thereafter, at each step, we know that we are close to where we were before, and so again we will find the right solution. Any spurious solution is likely to have attributes that rule it out as a physical solution. For example, it might be in space, or moving at an unrealistically high speed. I once talked with a GPS Engineer who told me that GPS devices actually pay attention to such attributes.

Here are some technical comments on writing the programs:

- Use the highest available accuracy throughout, e.g., use the *double* data type in java. Keep in mind that roughly speaking a meter makes a difference in the 7th or 8th digit of the distances involved. Light takes about 3×10^{-9} seconds to pass that distance, and since we contemplate times ranging up to one million seconds we need to process time information accurately to at least 15 digits.
- Make sure your standard output carries enough digits.
- Since your programs will have to work with those written by others you have to follow carefully the conventions laid out here. `vehicle` and `satellite` can't write anything to standard output other than what's specified (and for better comparability, `receiver` also should not write anything else).
- Your `satellite` should read the data file in the form given in the appendix and should not be deterred by the embedded comments.
- `satellite` will send data for all visible satellites. `receiver` does not know beforehand the number of those satellites. It will therefore have to decide when a group of signals starts and ends, and it will have to take reasonable action if the number of satellite signals is different from 4. If it's less it could write a message saying that there isn't enough information, if there are more it could select a subset of 4 signals, or it could solve a least squares problem.

My own inclination is to set up a least squares problem and solve it whenever the number of satellite signals is 4 or greater.

- You should have your `receiver` check whether or not your `satellite` sends only signals from satellites that are above the horizon. Otherwise you may be simulating a technical breakthrough that renders the earth transparent, but your `satellite` will not work with my less advanced `receiver`.
- Both `receiver` and `satellite` should terminate gracefully when they receive no more input. Thus they should shut down without irrelevant system messages, and they should close their log file before terminating.

Exercise 16: I gave an early draft of this assignment to my friend Meg Ikkal Anna Liszt[−9−]. After muttering about the federal deficit she said that she has been talking to the Air Force (who operate GPS) for years. She does not understand why they are being so hard on themselves. She could save them billions of dollars because to determine position and altitude you only need three satellites, not four! Three satellites would give you three differences in signal run times, those would constitute three equations for the three components of position, once you know position you can compute true run time to the satellite, and from that you can compute the current time. She thinks that the Air Force is not implementing this approach because they don't want to pay her fee of 10% of the savings in launch costs of satellites alone. What do you think of this?

Solution: Meg, one of my favorite characters, is the *Magical Analyst*. She has these ideas that look compelling at first, but then they evaporate in the cold light of day. The particular idea she is proposing here actually occurred to me one evening, and it seemed plausible to me at the time. I could not sleep that night, and had visions of informing the Air Force (who was developing GPS at the time) that they could get away with three satellites. Surely they would pay me some reasonable fee for this insight! In any case, here is what Meg is saying: Suppose you have three signals. Then we can form the differences:

$$\begin{aligned} \|x_{S_1} - \mathbf{x}\|_2 - \|x_{S_2} - \mathbf{x}\|_2 &= c\,(t_{S_1} - t_{S_2}) \\ \|x_{S_2} - \mathbf{x}\|_2 - \|x_{S_3} - \mathbf{x}\|_2 &= c\,(t_{S_2} - t_{S_3}) \\ \|x_{S_3} - \mathbf{x}\|_2 - \|x_{S_1} - \mathbf{x}\|_2 &= c\,(t_{S_3} - t_{S_1})\,. \end{aligned} \tag{74}$$

So Meg is right that three satellites give us three differences in runtimes, and usually three equations determine three unknowns. However, these particular equations are not *independent*. Adding the first and second equations in (74) gives the third (multiplied by -1). That third equation thus does not tell us anything we don't already know. Several people argued on general grounds that you need four satellites since you compute four data altogether, including t_v. That reasoning does not address Meg's point. She does have as many equations as she has unknowns. Besides, it is not actually true in general that you need at least as many equations as you have unknowns. For example, the single equation $x^2 + y^2 + z^2 = 0$ in three variables has the unique solution $x = y = z = 0$.

Exercise 17: After venting her frustration about the federal deficit Meg went to task with *me*. She said that "you academic types" like to be so "cumbersome". She thinks we don't use "common sense" because the very phrase isn't rooted in Latin or Greek. Why, she says, do I have to have integers **NS** and **EW** to indicate which hemisphere I'm on? Why, she says, don't I just make the degrees positive or negative? Indeed, why not?

Solution: The trouble is that by assigning a negative sign to the degrees we could not distinguish for example between 25 minutes south and 25 minutes north since positive and negative zero (degrees) are the same. We could avoid carrying plus or minus 1 with our data by transferring the sign to the minutes if the number of degrees is zero (and to the seconds if the minutes are zero as well.) However, this is not what Meg actually proposed!

[−9−] Can you tell the meaning of her name? Meg will make several appearances during this course.

What To Do

1. Work the exercises in this assignment, making up home work 1, and hand in your answers on

Friday, September 10, 2021, before the lecture.

2. Write the programs `satellite` and `receiver`. You will need a rudimentary `vehicle` for testing purposes. You can download one from

`http://www.math.utah.edu/~pa/5610/tp/`

or you can write your own. Your programs must run on all or some of our Unix systems. They may be written in any language, but of course they must accommodate the Unix standard input and output concepts. E-mail your programs and a pdf file with a written report of your work to me, by

Friday, November 12, before the lecture.

The report should be suitable for distribution among your class mates. Describe your mathematical approach, your software, how to compile and use it, the lessons you learned in this project, and anything else you may find worth mentioning.

Note: It is essential that your software compiles and runs on our departmental Unix systems. The only way you can assure that this is true is by testing your software on our systems yourself.

3. After I test and process your programs we will have a discussion of this project. I will ask some (or all if practical) teams or individuals to present part of their work to the whole class. Details will be announced as we approach the deadline.

To aid you in your testing you can download executable versions of my own `receiver` and `satellite` from

`http://www.math.utah.edu/~pa/5610/tp`

I will test each of your two programs in a pipe with my own programs. Note that in theory any combination of the form (4) should work.

The main benefit of this project will come from going through the process of generating the two programs, and making sure that everything fits and works together. When your program works with everybody else's you can make some claim that you truly understand the basic workings of GPS! You are invited to talk with me about the project at any time and if needed we will schedule time in class to discuss relevant aspects of the project. Observe our motto whose relevance for this particular project is paramount:

Procrastination is Hazardous

Note on Team Work

As mentioned in class, I recommend you work with one or two partners, to study together, and to work on the term project and the home works together. If you do work in a team hand in just one set of answers. Each of you will get the same score on all parts of an assignment.

Simplifying Assumptions

As additional information this sections contains an incomplete list of phenomena that have to be incorporated to make GPS viable in practice. This information is taken mostly from Hofmann-Wellenhof et al which gives detailed mathematical descriptions of most of these effects.

- **Signal Acquisition.** The satellite signals are weak and the signal to noise ratio is low. As a consequence the receivers work with a very narrow bandwidth. Due to the Doppler effect the frequency at which the signal is sent varies across a spectrum significantly wider than the bandwidth. Thus the receiver doesn't know the precise frequency at which the satellite is transmitting unless it knows the satellite's position and velocity. As a consequence, if starting from scratch, signal acquisition may take quite a while. The problem is alleviated by the fact that each satellite sends information on the current location of all other satellites (this information is called *ephemerides*).
- **Positioning with Doppler data or Carrier Phases.** Our programs model positioning with *range data*. More accurate positioning can be accomplished by taking into account Doppler data and carrier phases. The latter determine distance only within an integer multiple of the wavelength (which is 19.0cm or 24.4 cm). However, phases can be measured to better than 1% of wavelength, which potentially gives GPS an accuracy of about 1mm or less.
- **Relative Positioning.** In many applications the important item to be measured is the vector between two points, rather than the location of a point with respect to a global coordinate system. This can usually be done more accurately than the determination of a position.
- **Selected Availability and Anti Spoofing.** In the interest of national security the Department of Defense used to limit availability and accuracy of GPS to civilian users by encrypting the high accuracy version of the satellite signal (selected availability) and by dithering (distorting) the public signal (anti-spoofing). On May 2, 2000, these measures were turned off and the full accuracy of GPS is now available to the public. Interestingly, while selected availability or anti-spoofing were still in effect, they were turned off during crises like the first gulf war or the invasion of Haiti, to make it possible for the armed forces to make full use of commercially available GPS devices.
- **Spherical Earth.** The Earth is of course not spherical. It is more accurately described as an ellipsoid with a semimajor axis of 6,378,137 m and a semiminor axis of 6,356,752 m. When contemplating the determination of altitude within a cm or so that definition is also inadequate. What's really involved here is a precise definition of the term "sea level". The resulting shape of the earth is called the "datum" underlying a map or other navigational tool. For example, the Magellan Trailblazer can use any of 12 built-in datums.
- **Constant axis of rotation.** The North and South Poles move (at the *Chandler period* of about 430 days) within an area that has a diameter of roughly 6 m.
- **Circular Orbits.** The satellites do not move along circular or even elliptical orbits. The orbits are disturbed by a number of factors: non-spherical earth, gravitational anomalies, tidal forces, solar radiation pressure, and even air drag. The problem is overcome by the satellites broadcasting their actual position (rather than one based on an orbit calculation), but this needs to be determined using a sophisticated ground and space based infrastructure.
- **Time Dilation.** Time passes more slowly for an object that is moving at a high speed. This affects the accuracy of the satellite clocks.
- **Time of transmission.** We have to define precisely what we mean by the time of transmission of a signal. Each relevant signal consists of 1023 bits that are separated by about 10^{-6} seconds. Thus at the speed of light the distance between two bits is about 300m. This is called the "chip length" (and the bits are called "chips"). Current timing accuracy is between 0.1% and 1% of the chip length.
- **Ionospheric Refraction.** This is a major cause of inaccuracies. Its precise effects depend for example on current sunspot activity.
- **Tropospheric Refraction.** This depends largely on the amount of water in the various parts of the troposphere, i.e., on the weather.
- **Multipath Effects.** A receiver may receive more than one version of the signal due to reflection (e.g., on walls or cliffs).
- **Position of what?** GPS receivers are larger than the potential accuracy with which they can be used. Thus one has to figure out exactly which point (called the *phase center*, and located usually within the antenna) corresponds to the computed solution. The location of

the phase center of course depends on the geometry of the antenna and receiver, but also on the frequency and the intensity of the signal.

- **Physical Rigor.** Often GPS devices are employed in demanding conditions (e.g., rain, temperature extremes, dirt, vibrations). They need to be physically rugged.
- **Software.** The quality of a GPS device depends very much on its software. The cost of software development constitutes a very significant portion of the total purchase price.
- **Utility.** The utility of a device depends much on what it can do beyond computing position (and perhaps speed and direction). For example a device might contain tables of magnetic deviations from true North (like the Magellan Trail Blazer) or a database of (almost) all streets in the United States (like the Magellan GPS Map 7000) and so be able to give magnetic directions or a map with your current location.

Appendix I: The Data File

The whole project depends on a few parameters, like the speed of light, the radius of earth, and the orbits of the satellites. Rather than hardwiring these data into the programs we'll put them into a file that can then be easily modified. For example, if we wanted to run GPS on Jupiter or the Sun we would only have to modify that file.

We express the (perfectly circular) orbit of a satellite as

$$\mathbf{x}(t) = (R+h)\left[\mathbf{u}\cos\left(\frac{2\pi t}{p}+\theta\right)+\mathbf{v}\sin\left(\frac{2\pi t}{p}+\theta\right)\right] \tag{75}$$

where:

t is time measured in seconds

$\mathbf{x}(t)$ is the location of a satellite at time t expressed in cartesian coordinates with meters being the unit of length

R is the radius of the earth

h is the altitude of the satellite

$\mathbf{u}$ is a unit vector

$\mathbf{v}$ is a unit vector that is orthogonal to u

p is the periodicity of the orbit (assumed to be half a sidereal day)

θ is the phase of the orbit.

Crucial to the project is the data file listed in Table 1. (The leading line numbers are not part of the file.) You can download it from

`http://www.math.utah.edu/~pa/5610/tp`

In the unlikely case that that file needs to be changed I will let you know so you can obtain another file.

```
1   3.14159265358979311E+00          /= pi
2   2.99792458000000000E+08          /= c, speed of light, [m/s]
3   6.36744450000000000E+06          /= R, radius of earth, [m]
4   8.61640899999999651E+04          /= s, length of a sidereal day, [s]
5   1.00000000000000000E+00          /= u1 of Sat.  0
6   0.00000000000000000E+00          /= u2 of Sat.  0
7   0.00000000000000000E+00          /= u3 of Sat.  0
8   0.00000000000000000E+00          /= v1 of Sat.  0
9   5.73576436351046159E-01          /= v2 of Sat.  0
10  8.19152044288991798E-01          /= v3 of Sat.  0
11  4.30820449999999825E+04          /= periodicity of Sat.  0 [s]
12  2.02000000000000000E+07          /= altitude of Sat.  0 [m]
13  0.00000000000000000E+00          /= phase of Sat.  0 [rad]
14  1.00000000000000000E+00          /= u1 of Sat.  1
15  0.00000000000000000E+00          /= u2 of Sat.  1
16  0.00000000000000000E+00          /= u3 of Sat.  1
17  0.00000000000000000E+00          /= v1 of Sat.  1
18  5.73576436351046159E-01          /= v2 of Sat.  1
19  8.19152044288991798E-01          /= v3 of Sat.  1
20  4.30820449999999825E+04          /= periodicity of Sat.  1 [s]
21  2.02000000000000000E+07          /= altitude of Sat.  1 [m]
22  1.57079632679489656E+00          /= phase of Sat.  1 [rad]
23  1.00000000000000000E+00          /= u1 of Sat.  2
24  0.00000000000000000E+00          /= u2 of Sat.  2
25  0.00000000000000000E+00          /= u3 of Sat.  2
26  0.00000000000000000E+00          /= v1 of Sat.  2
27  5.73576436351046159E-01          /= v2 of Sat.  2
28  8.19152044288991798E-01          /= v3 of Sat.  2
29  4.30820449999999825E+04          /= periodicity of Sat.  2 [s]
30  2.02000000000000000E+07          /= altitude of Sat.  2 [m]
31  3.14159265358979311E+00          /= phase of Sat.  2 [rad]
```

```
 1.00000000000000000E+00          /= u1 of Sat.  3
 0.00000000000000000E+00          /= u2 of Sat.  3
 0.00000000000000000E+00          /= u3 of Sat.  3
 0.00000000000000000E+00          /= v1 of Sat.  3
 5.73576436351046159E-01          /= v2 of Sat.  3
 8.19152044288991798E-01          /= v3 of Sat.  3
 4.30820449999999825E+04          /= periodicity of Sat.  3 [s]
 2.02000000000000000E+07          /= altitude of Sat.  3 [m]
 4.71238898038468967E+00          /= phase of Sat.  3 [rad]
 5.00000000000000000E-01          /= u1 of Sat.  4
 8.66025403784438597E-01          /= u2 of Sat.  4
 0.00000000000000000E+00          /= u3 of Sat.  4
-4.96731764892154093E-01          /= v1 of Sat.  4
 2.86788218175523080E-01          /= v2 of Sat.  4
 8.19152044288991798E-01          /= v3 of Sat.  4
 4.30820449999999825E+04          /= periodicity of Sat.  4 [s]
 2.02000000000000000E+07          /= altitude of Sat.  4 [m]
 1.00000000000000000E+00          /= phase of Sat.  4 [rad]
 5.00000000000000000E-01          /= u1 of Sat.  5
 8.66025403784438597E-01          /= u2 of Sat.  5
 0.00000000000000000E+00          /= u3 of Sat.  5
-4.96731764892154093E-01          /= v1 of Sat.  5
 2.86788218175523080E-01          /= v2 of Sat.  5
 8.19152044288991798E-01          /= v3 of Sat.  5
 4.30820449999999825E+04          /= periodicity of Sat.  5 [s]
 2.02000000000000000E+07          /= altitude of Sat.  5 [m]
 2.57079632679489656E+00          /= phase of Sat.  5 [rad]
 5.00000000000000000E-01          /= u1 of Sat.  6
 8.66025403784438597E-01          /= u2 of Sat.  6
 0.00000000000000000E+00          /= u3 of Sat.  6
-4.96731764892154093E-01          /= v1 of Sat.  6
 2.86788218175523080E-01          /= v2 of Sat.  6
 8.19152044288991798E-01          /= v3 of Sat.  6
 4.30820449999999825E+04          /= periodicity of Sat.  6 [s]
 2.02000000000000000E+07          /= altitude of Sat.  6 [m]
 4.14159265358979312E+00          /= phase of Sat.  6 [rad]
 5.00000000000000000E-01          /= u1 of Sat.  7
 8.66025403784438597E-01          /= u2 of Sat.  7
 0.00000000000000000E+00          /= u3 of Sat.  7
-4.96731764892154093E-01          /= v1 of Sat.  7
 2.86788218175523080E-01          /= v2 of Sat.  7
 8.19152044288991798E-01          /= v3 of Sat.  7
 4.30820449999999825E+04          /= periodicity of Sat.  7 [s]
 2.02000000000000000E+07          /= altitude of Sat.  7 [m]
 5.71238898038468967E+00          /= phase of Sat.  7 [rad]
-4.99999999999998890E-01          /= u1 of Sat.  8
 8.66025403784438597E-01          /= u2 of Sat.  8
 0.00000000000000000E+00          /= u3 of Sat.  8
-4.96731764892154093E-01          /= v1 of Sat.  8
-2.86788218175523024E-01          /= v2 of Sat.  8
 8.19152044288991798E-01          /= v3 of Sat.  8
 4.30820449999999825E+04          /= periodicity of Sat.  8 [s]
 2.02000000000000000E+07          /= altitude of Sat.  8 [m]
 2.00000000000000000E+00          /= phase of Sat.  8 [rad]
-4.99999999999998890E-01          /= u1 of Sat.  9
 8.66025403784438597E-01          /= u2 of Sat.  9
 0.00000000000000000E+00          /= u3 of Sat.  9
-4.96731764892154093E-01          /= v1 of Sat.  9
-2.86788218175523024E-01          /= v2 of Sat.  9
 8.19152044288991798E-01          /= v3 of Sat.  9
 4.30820449999999825E+04          /= periodicity of Sat.  9 [s]
 2.02000000000000000E+07          /= altitude of Sat.  9 [m]
 3.57079632679489656E+00          /= phase of Sat.  9 [rad]
-4.99999999999998890E-01          /= u1 of Sat. 10
 8.66025403784438597E-01          /= u2 of Sat. 10
 0.00000000000000000E+00          /= u3 of Sat. 10
-4.96731764892154093E-01          /= v1 of Sat. 10
-2.86788218175523024E-01          /= v2 of Sat. 10
 8.19152044288991798E-01          /= v3 of Sat. 10
 4.30820449999999825E+04          /= periodicity of Sat. 10 [s]
```

```
 2.02000000000000000E+07          /= altitude of Sat. 10 [m]
 5.14159265358793116E+00          /= phase of Sat. 10 [rad]
-4.99999999999998890E-01          /= u1 of Sat. 11
 8.66025403784385966E-01          /= u2 of Sat. 11
 0.00000000000000000E+00          /= u3 of Sat. 11
-4.96731764892154092 9E-01        /= v1 of Sat. 11
-2.86788218175523024 2E-01        /= v2 of Sat. 11
 8.19152044288917986E-01          /= v3 of Sat. 11
 4.30820449999999825E+04          /= periodicity of Sat. 11 [s]
 2.02000000000000000E+07          /= altitude of Sat. 11 [m]
 6.71238898038468967 4E+00        /= phase of Sat. 11 [rad]
-9.99999999999997780E-01          /= u1 of Sat. 12
 1.11022302462515654 0E-16        /= u2 of Sat. 12
 0.00000000000000000E+00          /= u3 of Sat. 12
-5.55111512312578270 2E-17        /= v1 of Sat. 12
-5.73576436351046048 4E-01        /= v2 of Sat. 12
 8.19152044288917986E-01          /= v3 of Sat. 12
 4.30820449999999825E+04          /= periodicity of Sat. 12 [s]
 2.02000000000000000E+07          /= altitude of Sat. 12 [m]
 3.00000000000000000E+00          /= phase of Sat. 12 [rad]
-9.99999999999997780E-01          /= u1 of Sat. 13
 1.11022302462515654 0E-16        /= u2 of Sat. 13
 0.00000000000000000E+00          /= u3 of Sat. 13
-5.55111512312578270 2E-17        /= v1 of Sat. 13
-5.73576436351046048 4E-01        /= v2 of Sat. 13
 8.19152044288917986E-01          /= v3 of Sat. 13
 4.30820449999999825E+04          /= periodicity of Sat. 13 [s]
 2.02000000000000000E+07          /= altitude of Sat. 13 [m]
 4.57079632679489655 8E+00        /= phase of Sat. 13 [rad]
-9.99999999999997780E-01          /= u1 of Sat. 14
 1.11022302462515654 0E-16        /= u2 of Sat. 14
 0.00000000000000000E+00          /= u3 of Sat. 14
-5.55111512312578270 2E-17        /= v1 of Sat. 14
-5.73576436351046048 4E-01        /= v2 of Sat. 14
 8.19152044288917986E-01          /= v3 of Sat. 14
 4.30820449999999825E+04          /= periodicity of Sat. 14 [s]
 2.02000000000000000E+07          /= altitude of Sat. 14 [m]
 6.14159265358793116E+00          /= phase of Sat. 14 [rad]
-9.99999999999997780E-01          /= u1 of Sat. 15
 1.11022302462515654 0E-16        /= u2 of Sat. 15
 0.00000000000000000E+00          /= u3 of Sat. 15
-5.55111512312578270 2E-17        /= v1 of Sat. 15
-5.73576436351046048 4E-01        /= v2 of Sat. 15
 8.19152044288917986E-01          /= v3 of Sat. 15
 4.30820449999999825E+04          /= periodicity of Sat. 15 [s]
 2.02000000000000000E+07          /= altitude of Sat. 15 [m]
 7.71238898038468967 4E+00        /= phase of Sat. 15 [rad]
-5.00000000000000000E-01          /= u1 of Sat. 16
-8.66025403784383745E-01          /= u2 of Sat. 16
 0.00000000000000000E+00          /= u3 of Sat. 16
 4.96731764892153981 9E-01        /= v1 of Sat. 16
-2.86788218175523079 7E-01        /= v2 of Sat. 16
 8.19152044288917986E-01          /= v3 of Sat. 16
 4.30820449999999825E+04          /= periodicity of Sat. 16 [s]
 2.02000000000000000E+07          /= altitude of Sat. 16 [m]
 4.00000000000000000E+00          /= phase of Sat. 16 [rad]
-5.00000000000000000E-01          /= u1 of Sat. 17
-8.66025403784383745E-01          /= u2 of Sat. 17
 0.00000000000000000E+00          /= u3 of Sat. 17
 4.96731764892153981 9E-01        /= v1 of Sat. 17
-2.86788218175523079 7E-01        /= v2 of Sat. 17
 8.19152044288917986E-01          /= v3 of Sat. 17
 4.30820449999999825E+04          /= periodicity of Sat. 17 [s]
 2.02000000000000000E+07          /= altitude of Sat. 17 [m]
 5.57079632679489655 8E+00        /= phase of Sat. 17 [rad]
-5.00000000000000000E-01          /= u1 of Sat. 18
-8.66025403784383745E-01          /= u2 of Sat. 18
 0.00000000000000000E+00          /= u3 of Sat. 18
 4.96731764892153981 9E-01        /= v1 of Sat. 18
-2.86788218175523079 7E-01        /= v2 of Sat. 18
```

```
 8.191520442889917986E-01          /= v3 of Sat. 18
 4.308204499999999825E+04          /= periodicity of Sat. 18 [s]
 2.020000000000000000E+07          /= altitude of Sat. 18 [m]
 7.141592653589793116E+00          /= phase of Sat. 18 [rad]
-5.000000000000000000E-01          /= u1 of Sat. 19
-8.660254037844383745E-01          /= u2 of Sat. 19
 0.000000000000000000E+00          /= u3 of Sat. 19
 4.967317648921539819E-01          /= v1 of Sat. 19
-2.867882181755230797E-01          /= v2 of Sat. 19
 8.191520442889917986E-01          /= v3 of Sat. 19
 4.308204499999999825E+04          /= periodicity of Sat. 19 [s]
 2.020000000000000000E+07          /= altitude of Sat. 19 [m]
 8.712388980384689674E+00          /= phase of Sat. 19 [rad]
 4.999999999999996669E-01          /= u1 of Sat. 20
-8.660254037844384856E-01          /= u2 of Sat. 20
 0.000000000000000000E+00          /= u3 of Sat. 20
 4.967317648921540374E-01          /= v1 of Sat. 20
 2.867882181755229132E-01          /= v2 of Sat. 20
 8.191520442889917986E-01          /= v3 of Sat. 20
 4.308204499999999825E+04          /= periodicity of Sat. 20 [s]
 2.020000000000000000E+07          /= altitude of Sat. 20 [m]
 5.000000000000000000E+00          /= phase of Sat. 20 [rad]
 4.999999999999996669E-01          /= u1 of Sat. 21
-8.660254037844384856E-01          /= u2 of Sat. 21
 0.000000000000000000E+00          /= u3 of Sat. 21
 4.967317648921540374E-01          /= v1 of Sat. 21
 2.867882181755229132E-01          /= v2 of Sat. 21
 8.191520442889917986E-01          /= v3 of Sat. 21
 4.308204499999999825E+04          /= periodicity of Sat. 21 [s]
 2.020000000000000000E+07          /= altitude of Sat. 21 [m]
 6.570796326794896558E+00          /= phase of Sat. 21 [rad]
 4.999999999999996669E-01          /= u1 of Sat. 22
-8.660254037844384856E-01          /= u2 of Sat. 22
 0.000000000000000000E+00          /= u3 of Sat. 22
 4.967317648921540374E-01          /= v1 of Sat. 22
 2.867882181755229132E-01          /= v2 of Sat. 22
 8.191520442889917986E-01          /= v3 of Sat. 22
 4.308204499999999825E+04          /= periodicity of Sat. 22 [s]
 2.020000000000000000E+07          /= altitude of Sat. 22 [m]
 8.141592653589793116E+00          /= phase of Sat. 22 [rad]
 4.999999999999996669E-01          /= u1 of Sat. 23
-8.660254037844384856E-01          /= u2 of Sat. 23
 0.000000000000000000E+00          /= u3 of Sat. 23
 4.967317648921540374E-01          /= v1 of Sat. 23
 2.867882181755229132E-01          /= v2 of Sat. 23
 8.191520442889917986E-01          /= v3 of Sat. 23
 4.308204499999999825E+04          /= periodicity of Sat. 23 [s]
 2.020000000000000000E+07          /= altitude of Sat. 23 [m]
 9.712388980384689674E+00          /= phase of Sat. 23 [rad]
```

Table 1. The file `data.dat`.

You should call it `data.dat` and have your `satellite` read it and use the data. Your `receiver` should use the data contained in the first four lines of that file.

Appendix II: Accessing Math Department Unix Systems

You may develop your software on any machine you have access to. However, it is essential that your software runs on our departmental Unix systems.

A note of caution: Portability is a concept, not a fact! The only way you can be sure your software works is by testing it on our systems. (For example, in the past many people have had difficulties porting java code from the eclipse environment to our systems.) If it runs on some departmental systems but not others let me know which one you would like me to test it on.

As a student in our department you already have a Unix account. You can access our systems directly in our computer lab in the math center, or remotely from a PC, mac, or Ipad. The terminal app on a Mac let's you do this directly, for a PC or Ipad you may need to download some software. For a PC I use putty and psftp, available at `https://www.putty.org/`. On my Ipad I use terminus, available in the app store. There are many other options. For our project, all interfacing is text based, you do not need to be able to create graphics on your machine.

The generic name of our servers is

`xserver.math.utah.edu`

Your login name is of the form

`c-rstuxy`

where all letters are lower case and:

`c-` means "class account".

`r` is the first letter of your last name.

`s` is the last letter of your last name.

`t` is the first letter of your first name.

`u` is the first letter of your middle name. It is missing if you have no middle name.

`xy` is missing for most accounts. However, there may be people with the same initials. For those `xy` is replaced with 1 or 2 decimal digits.

For example, if your name is **Sir Isaac Newton** your login name might be `c-nnsi`, `c-nnsi2`, or perhaps `c-nnsi12`.

Your initial password is `rstuzzzz` where `rstu` is the same letter sequence as in your login name, and `zzzz` are the last 4 digits of your student ID number. For example, if Sir Isaac's student ID number was u1234567 and his login name was any of the possible user names above his initial password would be `nnsi4567` .

After logging in for the first time you probably want to change your password, using the Unix `passwd` command.

If you have trouble logging into our systems let me know and if I cannot help you myself I will refer you to our system staff.