The Software and Compiling

The software is written in Python 2.7 outside of primitive python packages the only neccesary package is Numpy.

It is compatible and written to read the output from any Vehicles or Satellites regardless of code.

To run the code on the math machines just unzip the JBDKtp.zip file in the same directory as the files from the all.zip file from the term project.

The Satellite

The satelite program takes in a stream of data in the form given by the vehicle file $(t_V, \psi_d, \psi_m, \psi_m, NS, \lambda_d, \lambda_m, \lambda_s, EW, h)$ that represent a trip starting from a point "b12" at the Unviersity of Utah and ending at any given point. For each step put through the program the satelite outputs a group of satelite data given by (i_S, t_S, x_S) for every satelite above the horizon. Thus the program takes 3 steps: finding which satelites are above the horizon, calculating the time, and finding where the satelite was at that time.

The satelite uses the simple condition $x^T s > x^T x$ to check if a satelite is above the horizon. The position of the point x is given by the vehicle and is simply converted from the vehicle format to cartesian coordinates. Then for each satellite the point s is just the position of the satellite at the given time t_V.

Our satelite program chose to use the simplest iterative method to calculate the time the signal was sent by the satellite. We take the given time sent as $t_V = t_0$ and use the iteration:

$$t_{k+1} = t_V - \frac{|x_s(t_k) - x_V||_2}{c}$$

This iteration ends when the difference between sucessive iterations is strictly less than the time it takes for light to travel 1 cm in a vacuum. As an intermediary step to this procedure, the position of each satellite is calculated several times. We compute the position of the satellite using the function:

$$x_S(t) = (R + h)[u_S \cos(\frac{2\pi t}{p} + \theta_S) + v_S \sin(\frac{2\pi t}{p} + \theta_S)]$$

Where the final position is stored as a close approximation to the actual position of the satellite.

The major design decisions for the satellite were guided by the hope to make the program as simple as possible at the cost of having a sligtly longer run time. We use the simple iterative method instead of newtons method because while it takes a larger number of steps to get the specified accuracy, it does not require taking derrivatives of a complicated function. Using a simpler function, in our opinon, reduces the chance for a human error while coding the satellite, while retaining the same level of accuracy.

While finding which satellites were above the horizon we chose to use the position of the satellite at tv instead of the position at the time the satellite sent the signal. We decided the possibility for a sattelite to move past the horizon was not worth accounting for. In general it is unlikey for a satellite to be above the horizon but then not above the horizon a tenth of a second later. While it is still possible we did not think it was a significant enough problem to adress. This decsision was made to take advantadge of only calculating a subsection of the satellites. Since we know which satelites we are going to use before we start calculating the time and position our code only considers those satellites instead of all satellites. If we wanted to remove this potential problem we would need to calculate the time and position for every satellite then find which satellites are actually above the horizon. Doing this would make our code slower and intoduces a need to store the satellite data (which we avoid by immediately sending the data to standard output) so, we decided not to address this minor problem.

The Receiver

The program receives the number (i_s), the time at (t_s), and the position of (x_s) of each satellite above the horizon in relationship to the vehicle.

It outputs the positional information in the form of the vehicle $(t_V\ \psi_d\ \psi_m\ \psi_m\ NS\ \lambda_d\ \lambda_m\ \lambda_s\ EW\ h)$.

Using the information from the satellites we must solve the equation:

[F(x) =

$$\begin{cases} ||xs_2 - x||_2 - ||xs_1 - x||_2 - c(ts_1 - ts_2) \\ ||xs_3 - x||_2 - ||xs_2 - x||_2 - c(ts_2 - ts_3) \\ ||xs_4 - x||_2 - ||xs_3 - x||_2 - c(ts_3 - ts_4) \end{cases}$$

]

Where $F(x) = 0$.

This becomes a root finding problem where we can implement Newtons Method in the form:

$x_{k+1} = x_k - J(x)^{-1}F(x)$ where $J(x)$ is the Jacobian of $F(x)$.

In order to prevent the problem of taking the inverse of a possibly singular matrix $J(x)$

the problem can be converted to $J(x)s = -F(x)$ where $s = x_{k+1} - x_k$ is our step and instead compute a vector s which minimizes Euclidean 2 norm $||F(x) + J(x)s||_2^2$.

We set our initial x_k equal to the position of SLC at $t = t_s$, where t_s is the time at one of our satellites, providing a decent estimate.

We then iterated through Newtons method in the following process:

Compute $-F(x_k)$

Compute $J(x_k)$

Solve $J(x_k)s = -F(x_k)$

Compute our new x_{k+1}

Check if step s < 0.1. If it we have our x_v otherwise repeat process.

The output of this iterative process would be an approximate position for the vehicle x_v. To compute $t_v

insert into equation $||x_s - x||_2 = c(t_v - t_s)$. Where t_s is the time at our first satellite given. Solve for t_v:

$t_v = \frac{|x_s - x||_2}{c} + t_s$

Given that we would recieve input from several iterations of satellite information, we created a loop that would solve this problem

for each given iteration of data (i_s, t_s, x_s).

Then finaly we convert our position x_v from cartesian coordinates to the vehicle format $(t_V\ \psi_d\ \psi_m\ \psi_m\ NS\ \lambda_d\ \lambda_m\ \lambda_s\ EW\ h)$ and print the vehicle

Lessons Learned

Newtons Method is extremely useful and easy to apply. For well defined problems, you can organize equations(s) in form $F(x) = 0$.

Write code to solve by iterating through Newton's Method until reaching a desired level of precision and you have solved your unknowns.

For problems where your system is over defined instead of solving $F(x) = 0$, solve for $F(x)^2$ = min to find a more precise solution.

Given $F(x)$ = min, we must solve $J(x) = 0$ (or $F'(x) = 0$) to find where the gradient equals zero and thus find our minimum.

Then you were able to use Newtons Method. It was interesting to see how practical and efficient this method is

in use. It's obvious that this approach can be applied widely for future use.

Challenges

We encountered several problems throughout the Term Project.

We struggled finding an efficient way to solve our system of equations for $J(x)s = -F(x)$.

At first we attempted to take the inverse but on some iterations we would encounter singular matrices and our code would fail.

We ended up using a Least Squares Method to find the solution to these equations, quite similar to the formulas found in HW 1.

While working on this project we found that it was crucial to stay organized.

At first we underestimated how important it was which caused a significant amount of trouble as we tried to track down bugs later in the project.

We found that writing Psudeo-Code was helpful to stay on track. Working with the math and coding for this project we found that there are a lot of moving parts,

and, at least for now, the concepts are unintuitive so we needed to be very careful while working on the project.

While working on the receiver, we redid our code for our jacobian calculation several times before realzing that the error was not with

the jacobian but with our conversion from cartesian to vehicle format. It was quite the relief when we found the error and suddenly

our code was giving accurate vehicle information.