Numerical Analysis Project
MATH 5600
Term Project Documentation

Authors:
Dane Gollero · Ike Griss Salas · Magon Bowling

Due: April 9, 2021

Contents

1 How to Use Code

We give a brief exposition of our programs and instructions on how to operate said programs.

1.1 Programs and Description

For our project we have created two programs, *satellite.py* and *receiver.py*. Both programs require files provided from class; namely, *vehicle.class* necessary to pipe data into the satellite program on the command line, *data.dat* containing all relevant values, and dat files that corresponds to various trip locations. It is important to note that in order for these programs to function correctly, a few conditions are necessary.

Necessary Conditions:

(i) The file *data.dat* is in the same directory as *BOTH satellite.py* and *receiver.py*.

(ii) Use **python3** command in math servers—an updated version of python is necessary.

(iii) *satellite.py* proceeds *receiver.py* in the command line execution code.

Both programs produce log files. The log files, appropriately named *satellite.log* and *receiver.log*, will be created in the same directory once the first instance of either *satellite.py* or *receiver.py* has been run. Upon running either program, the log files will only hold the data of the last execution.

1.2 How to use

Again, in order to use our program we emphasize the need to **use the python3 command in math servers!**. The *python* command will not suffice for convergence. In order to run a successful trip simulation, say *b12.dat* for instance, one would pipe the trip into vehicle.class on the math servers as normal—then pipe the vehicle data into *satellite.py* (and *receiver.py*). Below are the following commands described to see the output of either program—where *b12.dat* can be exchanged for any trip.[1]

≫ *b12.dat* | java vehicle | python3 satellite.py

≫ *b12.dat* | java vehicle | python3 satellite.py | python3 receiver.py

2 Creating the Team & Initial Planning

2.1 Connections

Connections are invaluable, and we count ourselves fortunate to have came together for this term project. Dane and Ike met in Foundations of Analysis a year ago and have shared additional University courses. More recently, they enjoyed the topic of Topology with a delightful professor! While Dane was enjoying the camaraderie in Topology with Ike, Dane was simultaneously introduced to Magon in Introduction to Probability. Little did she know that working with Dane would change her life by guiding her to friendship, unity, and perseverance with like-minded individuals in Numerical Analysis.

[1] It is worth noting that sp.dat will return an error as the trip does not seem to converge with the receiver program.

2.2 Establishing Routine

Group selection was made within the first week of class, and we quickly dove into homework 1. Ike led the charge right out of the gate! We discussed the need for weekly meetings for accountability and support. We first started with meeting once a week, but as April 9th drew nearer the need for daily meetings became the norm. Our goal was to timely execute the expanse of the term project with collective roles and responsibilities. Thankfully Ike encouraged us to complete stages of the term project consistently throughout the semester and not wait until the last minute.

2.3 Supportive Learning

Each member brought strengths and value to the group and the term project. We shared all our work on GitHub for collaboration and access. Ike took the initiative and began programming code for the term project shortly after homework 1 was complete. Magon gladly took on the role of LaTeX writer, while Dane was the rock for checking work and reinforcing the proper coding. The last several weeks have seen the group members diligently meeting nightly to ensure complete execution of this term project.

3 Challenges & Obstacles

3.1 More to Life than Math

An obstacle was in fact our own perception of the difficulty of this project. We underestimated the daunting task by naively thinking we could simply translate our mathematical algorithms into tangible code. This was a discredit to the inner complexities of the project. Knowing the mathematical principles was not enough to create a successful program. Further tasks were required, such as:

- Understanding how to interact with the command line interface and reading data that had been piped in from proceeding programs.
- Reading and using data from *data.dat* file.
- Storing the data.
- Implementing functions to cross-check outputs to known outputs.

Ultimately, once the interaction with the command line was discovered, it appeared quite simple and easy compared to the following tasks.

Reading in *data.dat* for our satellite and receiver code proved to be rather cumbersome. The format of the file posed a problem, such as the *values* of desired variables (i.e., speed of light, radius of earth, satellite information) began flush left, and the associated *variable name* was flush right (equality was also denoted with "/ ="). We needed to decide in which manner to store the data, which Dane tackled by creating a function that would open the *data.dat* file (assumed to be the same directory) and created a program tailored to the format of the *data.dat* file, allowing us to scrape and clean up the data.

For storing the data, specifically satellite data, we opted to store in a dictionary (as seen below). The desire was to easily access this data later in our programs. For the remaining data we simply assigned the values to appropriate variable names.

$$\text{satellite\_dictionary} = \{i_s : [t_s,\ x,\ y,\ z\,], \cdots\}$$

Lastly, and perhaps the most difficult and time consuming challenge was thoroughly testing our code for accuracy. This section of the project proved to be defeating in times with its complexity. More often than not, our code would utterly fail. Only with time and persistence were we able to conquer. Needless to say, many late nights were needed and highlighted to us that understanding the math did not necessarily mean the implementation would be flawless.

3.2 Math understanding vs Implementation

This project taught us all that mathematical understanding does *NOT* imply perfect implantation of code. Where we had a firm grasp of the mathematical theory needed to solve these problems, there was a rather large number of issues we encountered when translating these ideas into python code. Thankfully, we had a guide in Peter's work to ensure our data outputs matched his. Dane approached this project with computer programming background. This became invaluable in catching the reasons for the errors as they appeared.

3.3 A computer is only as smart as the one using it

As much as we joked about the code reading our thoughts, things were not so simple. Infuriating mistakes, sometimes so simple and other times quite complicated, proved to push us to the brink of tears at times. Dane was designated the group coder and scribe in the final receiver and log stages of the project. Many times, he wished for the code to write itself.

3.4 It's Bug Season

An extremely difficult component of this project was understanding why we produced the amount of errors we encountered—whether mathematical or syntactical in nature. Small issues such as typos were expected, however, larger errors such as losing data with a *float* function were quite subtle and much harder to catch. Iteration functions would diverge due to incorrect *initial guess*—in our case we simply chose a the wrong time.

In hind sight, there were a couple of things that we could have done as a team to ease difficult and tedious tasks. For example, using the existing logs and creating our own logs for satellite and receiver. Being able to compare the initial guesses and the number of iteration steps for the receiver code would have been a tremendous time saver—had we realized sooner. Additionally creating separate programs to test our code systematically would have reduced some of the tedious cross-checking. Hindsight always seems to be 20/20.

3.5 Comment Comment Comment

One valuable resource in the coding process was the need for comments to describe the ideas behind the code. Too many times we would return to our program with little recollection as to what specific variable names and functions meant. Once we added comments to every definition, function, and variable, life became more efficient. However, as Ike graciously pointed out, several areas of our code reflected tired college students. Too many comments can be confusing and cumbersome.

3.6 Precision and Future Improvements

One thing about the precision of our code needs to be said. Depending on the size of the time steps between epochs for a given trip, our code may or may not converge. It was noted that np.dat is a file that does not converge when the number of steps of the trip less than 18. Beyond that our code works for that trip.

Future improvements to the code that are of interest to pursue are:

1. Corrections from Time Dilations in calculating the time Satellites need to be above the horizon.
2. Corrections from Photon Redshift/Blueshift on signal acceptance conditions.
3. Corrections to the horizon condition from the bending of light in the Earth's gravitational field.
4. Use higher Precision data types—e.g. Python Floats vs. Python Decimal.

Further questions to clarify about the performance of the code are:

1. Can we have a smaller number of steps in a given trip before divergence in Newton's Method?
2. Can the speed of the code improve?
3. Are alternative algorithms to Newton's Method more efficient?

4 Teamwork & Unity

4.1 Feedback & Criticism

Working in a group is a reward and a challenge. Trusting team members takes commitment and vulnerability. We all had to be willing to listen and reflect. Sharing thoughts and opinions became affluent with deeper trust and support. We all grew through positive feedback and criticism.

4.2 Patience & Encouragement

As a team, we gladly shared what we could afford to the group dynamic. Ike encouraged a development of pirate names in support of our unique skills and abilities. "Dirty D" came because of Dane's incredible ability to dig deep into all the dirty work of the code. He was the mastermind behind the debugging and reading of the data. "Griss the Grit" reflects Ike's tenacity and perseverance in understanding and teaching the math behind the code. "Mighty Tiny" is Magon's small force and extremely keen eye for catching incongruities.

5 Reflection

This project was by no means a trivial task, often making us question our abilities to succeed. With this being said, perhaps one of the most rewarding moments in this process was seeing our first program successfully compile with accuracy within 1cm. The utter joy this instilled in us was well worth the pain and tears that lead up to this moment. Prayers were answered, and faith restored in our abilities and capabilities to accomplish this project. Beyond the mathematical triumphs, an equally rewarding aspect of this project was in working with an amazing team!