Python Applications

Robert Appleton

May 11 2019

1 Data Analysis of Global Temperature Data

1.1 Introduction

The amount of research being done in climate change and global warming is extremely large and in most cases very complex. Despite the wide range agreement among scientist, the public and more importantly government policy does not seem to reflect the severity of the problems. In this section, data will be retrieved on the mean global temperature from the years 1880 to the present. This data will be imported into a python notebook where the plot from the data source will be reproduced. The next step will be to plot best fit lines to the data in segments of 20 years. The goal will be to take the slope from each best fit line and use it as a very rough description of the rate of change of temperature with time. Conclusions will be drawn on how the different time periods compare to each other.

1.2 Getting the Data

The data used in this paper was retrieved from NASA's Goddard Institute for Space Studies. The data consists of the annual mean of the global land-ocean temperature index for the years 1880 to the present. First off, the global land-ocean temperature index is a global average of the different measured surface air temperatures (SATs) and sea surface temperatures (SSTs). Weather stations reports SATs are limited to mostly land, which covers only 30 percent of the planet. On the oceans SATs are rare, however bouys and ships measure SSTs and now even satellites [1].

The data does not contain the absolute values of the annual means for the global land-ocean index, but instead they are temperature anomalies. A temperature anomaly indicates how warmer or colder it is compared to a normal base temperature. The normal for this data was the mean of the data from the 30 year period 1951-1980 [1]. The data also includes a LOWESS smoothing curve for the data. LOWESS stands for Locally Weighted Scatterplot Smoothing. It is simply a popular tool used in regression analysis that creates a smooth

line through a timeplot or scatter plot to help you to see relationship between variables and foresee trends [3]. The data was downloaded from the public NASA website for Global Climate Change and then imported to a python notebook using numpy command loadtxt.

1.3 Plotting the Data

After importing the data, the data was printed to see how it was stored. The numpy array contained 139 rows and 3 columns. Each column corresponded to a year, a mean global temperature anomaly, and a LOWESS smoothing point. The data set was sliced to save each column as a list for plotting. The plot created in the notebook and the plot from the website are displayed below.

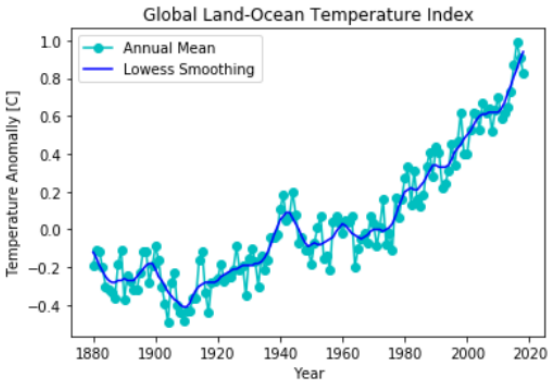

Figure 1: Recreation

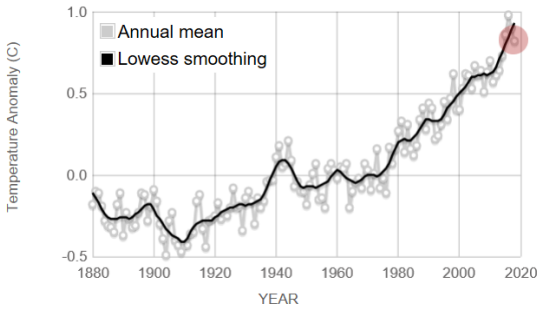

Figure 2: Original

The next step is to add the best fit lines. numpy has a command polyfit that performs a least squares polynomial fit [2]. This command is applied for a first degree fit of the data in intervals of 20. The plot displayed below includes the best fit lines added to the previous plot.

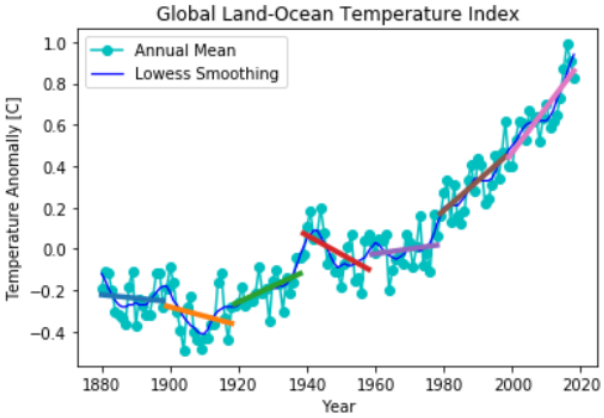

Figure 3: Data Set with Lines of Best Fit

The best fit lines in the plots above appear to rigidly follow the curve of the data. It is interesting to note the time periods when the linear fit had a positive or negative slope and the magnitudes. It appears that the slope of these lines fluctuates from positive to negative for the years 1880 to 1960. Interestingly, it also appears the slopes are growing in magnitude in this period. After 1960, the slopes of these lines is always positive and they have the highest magnitudes of the entire set. The year 1960 stands out because that is the same time period as the Chinese Industrial Revolution. This is not a statement of specifically this being the cause of the rapid increase. This is more of an indication that this was a time that included global spread of industrialization practices, China's revolution is simply a historical reference. As much of the climate change research has revealed, when these industrial practices reached a global scale, the global climate was influenced.

1.4 Conclusion

From the analysis of the data it is clear that since 1960 the global temperature has been on a steep increase as compared to the rest of the data from 1880 to the present. This paper provided some remarks on the possible human historical influences on these results. Of course there is a variety of research out there being done to more clearly tie the change in the global climate to human influence, however, this analysis is very simple but still emphasizes the clear issue at hand, which is the rapid increase in global temperature that is being measured.

2 Understanding Classes Using Physics

2.1 Introduction

As the power of computers increases, more and more scientific research and analysis is being done with computers. Physics is no exception, and physicists all over are taking advantage of these tools, such as python. As a physics student, it can be very helpful to learn python to solve physics problems. In a previous project for this course, a python program was created to model the motion of a rocket. The calculations required to do these by hand is simply impractical for analysis purposes. This is just one example of using python to solve a physics problem.

When a student is learning python, they will eventually come across the discussion of classes. This topic seems to give most people the most confusion when first trying to understand them. For most projects in python, classes are rarely a necessity for the code. However, if classes were to be implemented there are some powerful advantages they can provide. This paper aims to give an introduction to classes and how to construct them, and then provide physical examples to illustrate the advantages of using them.

2.2 Python Classes

In python, whenever one tries to create some code to do something, what can be done and how it is done, completely depends on the type of objects initially created. Some examples of different types of objects are:

list = [a, b, c] - defined an object called 'list' and it is of type list

int = 15 - we defined an object called 'int' and it is of type integer

There are tons of these types that have many different advantages and uses. These types are built in classes that are so common that they became just a part of the basic python language. A list can only have certain functions, also referred to as methods. For example, one can use the list.append to add elements to the list, but if one tries int.append there would be an error because the method .append only applies to lists and not integers. This mechanism of where python recognizes types of objects and uses that information to determine the feasible methods that can be done on that object is a what is known as 'object oriented programming'. This is why it is very important when learning python to always be aware of the types of objects being created.

The object types, which are built in python classes, are very useful and can do almost everything needed for a simple project. However, python provides not only these built in classes, but it allows the user create their own. This has a lot of advantages and hope to make this clear in the examples below.

2.3 Creating a Class

A class is made up of attributes and methods. Methods were already briefly discussed as basically being a function that is specific to the class it is defined under. As for an attribute, in python it corresponds to an input needed to describe the uniqueness of the object of that class. In other words, what are the parameters of the objects that belong to that class. The attributes are defined in the init function. The init function is in every class that has attributes and is simply there to provide a mechanism for defining an object as a member of a class and initialize the inputs as the attributes. The init function will always have the self variable as one of its inputs. This self variable can be the trickiest thing to understand for classes. One way to think about it is this, if there is a class A that takes inputs (a, b, c) for its attributes, then the init function is used to define the specific inputs from an object as the attributes. One could define an object test as test = A(1, 2, 3). Test is now an object of class A with a = 1, b = 2, and c =3. The self variables in the init function are used to define these inputs for a, b, and c as the attributes self.a, self.b, and self.c. The syntax self.{} is used to allow the methods, or function that is defined under a class, to have access to the information stored in the object that is defined with given inputs. This means that if the methods are functions that include the attributes as variables, then one must use the syntax self. in the functions. This also applies if one wants to use the output from one

method as a variable in another method, one must use the self. syntax. This becomes more clear following examples and creating new classes, but in brief it is a 'signal' that makes the variable available to be used from one method to another.

2.4 First Step: Projectile Motion

In this paper, a class was defined fora projectile. When one solves a problem for projectile motion, there is always some information given. A projectile can be completely described by a velocity, an angle, and a starting position. Therefore, these were the attributes for this class. From these attributes one can calculate the components of the velocity in the x and y directions, flight time, range, and max height. These calculations will represent the methods for the class. Below is a summary of the attributes and methods that the class was constructed from.

Attributes - velocity, angle, starting position

Methods - components of velocity, flight time, range and max height

If the angle is taken to be from horizontal, then the components can defined by:

$$v_x = v\cos(t),$$

and,

$$v_y = v\sin(t).$$

Now the y component of the velocity can be used to determine the flight time. Basically it becomes a problem of just a ball going up and coming to rest from the acceleration due to gravity g. The time to fall is equal to this so it must be multiplied by a factor of 2 to get the full flight time.

Thus, the calculation starts from the kinematic equation:

$$v_f = v_i + at,$$

(note: t is time here). It is known that v_f is 0 at apogee and a = g so one can solve for time:

$$t = 2\frac{v_i}{g},$$

(note: v_i is v_y). Using flight time one can find the range. Range is the distance traveled in the x direction. There is no acceleration in this direction so we take the x component in the velocity and use the equation for constant velocity:

$$v = \frac{d}{t}.$$

Solve for distance:

$$d = vt.$$

The only thing left to calculate is the max height. One could start from the kinematic equation:

$$v_f^2 = v_i^2 + 2ad.$$

Solve for distance and $v_f = 0$:

$$d = \frac{v_i^2}{2a}.$$

[4]

Using these equations the methods for the class were created. Now one can be given the velocity, the angle and the starting position of a projectile and by simply calling the methods the calculations are performed immediately. It is fun to define one projectile and to execute the functions, but the real power is the ability to design the class functions to be passed lists of attributes. For this paper, 50 launch velocities from 1 to 50 m/s and 5 different launch angles of $\frac{\pi}{8}$, $\frac{\pi}{6}$, $\frac{\pi}{4}$, $\frac{\pi}{3}$ and , $\frac{2\pi}{5}$ were passed to the projectile class. Other methods were added to the projectile class to create plots for time vs velocity, range vs velocity, and max height vs velocity. The resulting plots are displayed below.

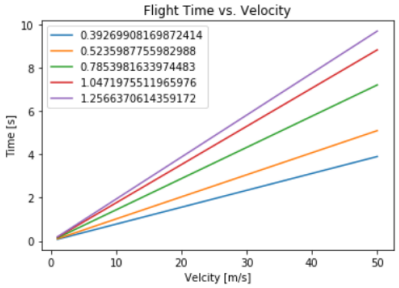

Figure 4: Flight Time vs Velocity

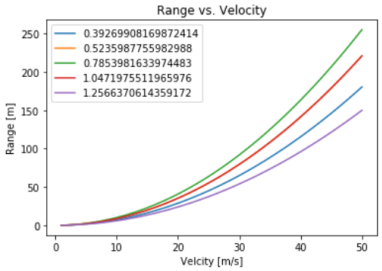

Figure 5: Range vs Velocity

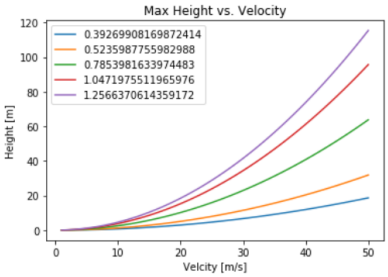

Figure 6: Max Height vs Velocity

These plots demonstrate the relationships for projectile motion. The same plots can be created in python by hard coding each object, but the important thing is that all of the work done for one object is the same for the rest. This is how classes come in to play to reduce the work load. After just solving the problem once for a general object, specific objects can be passes and calculations can be ran in real time.

2.5 Second Step: Rocket Motion

Expanding from the example above and one can make a more complicated class called Rocket. A rocket will have attributes: exhaust velocity, initial mass, burn rate, wind coefficient of the rocket, and burn time. Some constants throughout include acceleration of gravity, projected surface area of the rocket, and the air density. This attributes and constants make up the necessary pieces for making the equations of motion for a rocket.

The equations of motion are derived directly from the forces acting on the rocket. This model will consider gravity, thrust, and drag for our rocket. These rockets will not go high enough to have to consider the change in the force of gravity with distance, thus the simple equation $F_G = mg$ can be used. For the force of thrust, the equation $F_T = u\dot{m}$ is used, where u is the exhaust velocity and $\dot{m}$ is the burn rate for the fuel. For drag, the equation with drag proportional to velocity squared, $F_D = \frac{1}{2}AC_w\rho v^2$, is used, where A is the projected surface area, C_w is the wind coefficient, ρ is the density of air, and v is velocity of the rocket. [4] On the way up, the forces of gravity and drag will point in the same direction while the force of thrust opposes them, and for this model it is assumed that up is positive. If the net force equation is divided by mass of the rocket, the following equation for the acceleration as a function of time is derived:

$$a = -g - \frac{u\dot{m}}{m_0 + \dot{m}} - \frac{\frac{1}{2}AC_w\rho v^2}{m_0 + \dot{m}t}.$$

One can see that the mass is a function of time equal to: $m(t) = m_0 + \dot{m}t$, where m_0 is the initial mass. Mass is decreasing as fuel burns and so it is important to note that $\dot{m}$ is negative. This also implies that the middle term, that corresponds to the thrust, is positive while the other two terms are negative.

If one writes the equation in the form of a differential equation they get:

$$\frac{dv}{dt} = -g - \frac{u\dot{m}}{m_0 + \dot{m}} - \frac{\frac{1}{2}AC_w\rho v^2}{m_0 + \dot{m}t}.$$

In the python function for calculating the kinematics of the rocket there is a for loop that steps through time in small segments (0.01 sec) and calculates the next steps acceleration by evaluating the equation above with the velocity of the previous step. The function is not only evaluated using small times steps, but the time steps are also broken into pieces to get a weighted average over that step. This weighted average is then used to calculate the velocity at that point in time. Then the process repeated over the entire flight time. The specific mathematical approximation used in this model is called the Fourth Order Runge Kutta Method, and it is similar to other methods of solving differential equations such as Simpson's Rule or Euler's Method. In fact, a simple Simpson's Rule is also applied to approximate the position of the rocket over time from the velocity calculated from the RK4

method. This paper will not go any further in depth about the mathematical method of solving our differential equation, but it is encouraged to research this types of mathematical approximations and also to experiment with the effects of step size on the accuracy of the approximation.

From the mathematical approximations of the differential equation of motion, arrays for the rocket kinematics (height, velocity, acceleration, time) are generated that will be used for plotting. Like the projectile motion class a method is defined for plotting, and when the model is passed a rocket with the following given attributes, typical of a common model rocket:

Exhaust Velocity : 565.2 m/s

Starting Mass : 0.0815 kg

Burn Rate : -0.0069 kg/s

Wind Coefficient : 0.04 (unitless)

Burn Time : 1.1 $sec.$

When an object of the rocket class is passed these inputs, the following plot below can be generated after the kinematic arrays are calculated.

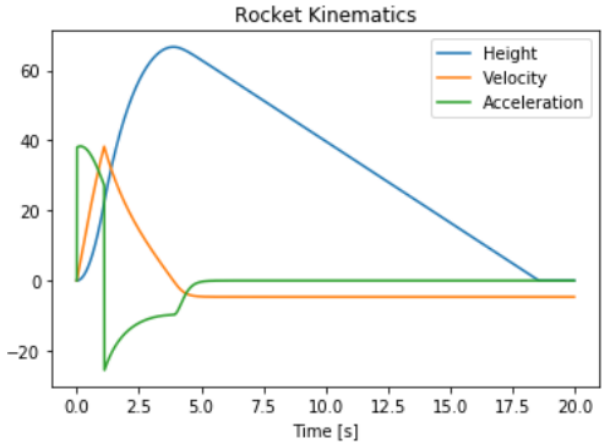

Figure 7: Generated Plot for Specified Rocket

2.6 Plot Breakdown

The green line is acceleration. Right away the acceleration shoots up to almost 40 m/s^2, then it gradually starts to decrease. What happened is the rocket was initially dominated by the thrust force and as velocity increased the drag term was taking effect. After 1.1 sec, the plot shows the acceleration drastically decrease to past -20 m/s^2. This is after burn out, so the thrust force disappears and the combination of drag and gravity (both negative) are

all that is left. The acceleration decreases in magnitude during this time as it slows down the drag term gets weaker. At approximately 4 sec, the rocket reaches apogee where its velocity is zero and it changes direction and begins to fall back to the ground. Here the drag term flips to positive, working against gravity. This causes the acceleration to go from some negative value and approach zero. In this model, a parachute is deployed after apogee and so this happens very quickly. This is what is called terminal velocity and is when the force of gravity and the force of drag reach an equilibrium. The acceleration will remain at zero the rest of the flight.

The yellow/orange line is the velocity. The velocity increases rapidly for the 1.1 sec of burn time. The velocity curve almost appears linear but if looking closely it has a slight bend towards the x axis as it approaches 1.1 sec. After burn out, the velocity drops drastically, as one would expect from the analysis of the acceleration at this point flips to negative. The point where the velocity curve crosses 0 for the y axis corresponds to the point where the rocket has reached its max height, or apogee. After this point, the velocity curve approaches a negative value where it remains constant. As stated before, after the rocket has reached apogee, it begins to fall back to the ground, and thus the drag term now opposes the force of gravity. They will reach an equilibrium and the velocity will become constant and this velocity is referred to as terminal velocity. This happens very quickly in this case because of a parachute to greatly increase the magnitude of the drag coefficient. The velocity remains constant as the rocket safely floats back to the ground.

The blue line is the height. During the 1.1 sec of fuel burn the rocket's height increases very sharply. After fuel burn, the acceleration is negative, however the velocity is still positive. While the velocity is still positive the rocket's height will still continue to increase. However, as seen in the graph, the velocity is positive but decreasing rapidly, this causes the height to tapper off and approach a maximum, the point of apogee. After apogee, the parachute is deployed, the rocket approached a safe terminal velocity, and approached the ground. This is represented in the graph by the height curve being linear after apogee. It is important to note that the time to apogee was approximately 4 sec and the time to the ground after apogee was approximately 14 sec, which is a clear effect of the parachute.

2.7 Conlcusion: Advantages of Classes and Final Remarks

The goal of this project was to provide an introduction to understanding python classes and to see an example of applying it to a physics problem. As a physics student, one may find it easiest to learn concepts in python in the scope of a physics application. The examples provided an illustration of how to use a class and it is clear to see the advantages of using them. When one does physics problems and models in python it can be easy to start programming the code to be very specific to the current problem one is trying to solve. The obvious limitation is that if one wanted to use the model with a change of parameter values, changes would have to be made to the code. That is why it can be powerful to get better with python classes. The clear advantage of using classes for this project is that one

could very quickly create multiple objects of the rocket class with variable inputs. It may be good practice to use classes as often as you can for reasons of versatility.

References

[1] NASA/GISS. Global surface temperature — nasa global climate change. *NASA's Goddard Institute for Space Studies*, 4 2019. https://climate.nasa.gov/vital-signs/global-temperature/.

[2] The SciPy Community. *NumPy basics*, v1.16 edition, 1 2019. https://docs.scipy.org/doc/numpy/user/basics.html.

[3] Statistics How To. Lowess smoothing in statistics: What is it? *Data Science Central*, 1 2019. https://www.statisticshowto.datasciencecentral.com/lowess-smoothing/.

[4] Freedman R. A. Ford A. L. Sears F. W. Young, H. D. *Sears and Zemansky's University Physics: With Modern Physics*. San Francisco: Pearson Addison Wesley, 2004.