WIKI

Motivation

In a society enriched by technology, it's inevitable that `GPS` will become increasingly commonplace. Not far does one have to go to find an application, like Facebook or, oddly enough, Angry Birds, that takes advantage of location information.

The unavoidable reality is that this technology is ubiquitous, that privacy as a consumer is dwindling. With the technology being everywhere, being ubiquitous, this also means that tools in order to analyze this data are more available.

Hence, tools like `phyphox` provide a means of recording your own coordinate data, exporting it in a rather limited (but understandable, given data format conventions) variety of file formats.

Coordinate data is inherently fascinating; in its raw form, you know exactly where that person is based on the geographical coordinate system. However, as with any data, one set of points is rather boring; it tells you minimum information. As such, it's more interesting to observe a change in this data. For something like `GPS`, this is motion.

Seeing how position changes as a function of a generic coordinate system (and predicting it, via fitting algorithms) is the most obvious application of such data. As such, this is the motivation of this project: to observe how my location changes across different trips, all localized within a few miles of my residence. We can also observe quantities such as velocity, acceleration, jerk, snap, crackle, and pop.

Background Information

Before we begin analyzing any data, it's important to establish some groundwork.

First, all data comes from phyphox, an application that turns your smartphone into a "mobile lab." There are a variety of experiments listed on their website, but the one we're most concerned with, of course, is their GPS experiment. Really, all they do here is record your position as discrete variables of altitude (m), latitude (°), and longitude (°). Alongside this, the standard deviation of horizontal and vertical position (m) are provided. They also provide information on velocity and acceleration. More information on this can be found on their website.

Second, the data for these results are provided in `CSV` and `Excel` formats. I opt to use `CSV` because that data form is overall more accessible and less bulky.

Third, most of the background data processing is done in my own custom made `.py` files. For two reasons, really: that the code is out of sight so more of the space can be used for discussion and that unit testing can be performed on some of the statistically invariant functions. A deeper dive into this code (and the code in this notebook) can be found in the `readme` and `code` in the github repository.

Fourth, all data can be found in the `data` folder, the figures in the `figures` folder, and the code in the `utitlity` folder. This is an act of cleanliness.

Fifth, the motion I chose to imitate is rather simple; walking in a straight line does not generalize to higher orders of motion or equations of other forces. So, my experiment can only be replicated as it by walking in a straight line.

Experiment

The procedure for the experiment is simple, but must be elaborated such that it can be repeated. First, consider that I am using a Samsung Galaxy S9 in an area with poor cell reception, so there may be inherent variance due to operating system differences, hardware, or the availability of cellular data among other things.

Then, consider that the starting point of my experiment begins at the end of my driveway, adjacent to my garage. The stopping point is, therefore, at the end of my driveway. This is but a distance of a few meters, taking about 30 seconds from point A to point B. A picture of this can be seen below:

Figure 1: my driveway. Despite popular belief, it is not inclined at 45 degrees. The illustration was added via GIMP.

Effectively, I start the `phyphox` experiment at the start of my driveway, walk in a straight line to the end of my driveway, and then stop the recording. Afterwards, I email myself the data while walking back to point A. I repeat this 9 more times. I could keep going, but for such a small journey this should be enough data. In trying to imitate the exact same initial conditions every time, the data should be easily combined such that any gaps in the trajectory are nearly filled.

Regardless, we can see what exactly this data looks like next.

Data

Once the data has been recorded over `phyphox`, it can be exported by a variety of means. As mentioned, I opted to export as a `CSV`, since it's less bulky and far more accessible.

Regardless, the data is formatted such that each column has a header and represents a different quantity. The format of the `CSV` can be seen below:

Time (s)	Latitude (°)	Longitude (°)	Altitude (m)	Altitude WGS84 (m)	Speed (m/s)	Direction (°)	Distance (km)	Horizontal Accuracy (m)	Vertical Accuracy (m)	Satellites
0	42.6525294	-78.8715136	304.1564857	270.3000183	0	0	0.05660056634	16.34799957	2.299987793	0
1.21936937	42.6530241	-78.87135081	255.467743	221.6101074	23.14999962	0.100000002	0.07120831871	17.15200043	48	0
2.20448437	42.6531551	-78.87133687	250.8141348	216.9561768	23.37999916	359.8999939	0.09937281602	13.93600082	32	11
3.207192599	42.6534082	-78.87134983	253.891192	220.0325928	24.13999939	0.300000012	0.1241426057	13.93600082	32	12
4.207082494	42.653631	-78.87135271	254.8272383	220.9680786	23.98999977	359.7999878	0.1491155714	10.72000027	24	12
5.203026244	42.6538555	-78.87136072	256.1218728	222.262146	24.03000069	0	0.1740937001	10.72000027	24	12
6.20853442	42.65408	-78.87137	258.4441569	224.5838623	24.18000031	359.7000122	0.2001435617	9.648000717	24	12
7.204141816	42.6543142	-78.87137733	258.1363985	224.2755127	24.18000031	359.5	0.2240603922	8.576000214	24	12
8.206242649	42.6545293	-78.87138124	259.3334733	225.4720459	23.75	359.7000122	0.2506695809	8.576000214	24	12
9.204293378	42.6547686	-78.87138263	258.7449633	224.8829346	24.29999924	359.7999878	0.2747226291	8.576000214	24	12
10.20294452	42.6549848	-78.87139337	260.0267607	226.1641846	23.37999916	359.2999878	0.2989544865	7.504000187	16	12
11.2065615	42.6552027	-78.871392	261.1322876	227.269165	23.77000046	359.6000061	0.3242494101	7.504000187	16	10
12.20793603	42.6554302	-78.87139527	261.2573717	227.3936768	24.07999992	359.7999878	0.3487701711	6.43200016	16	10

Figure 2: a table displaying some data native to the GPS export. Each column is a different quantity representing data relevant to what GPS tracking would produce.

I'll take this opportunity to describe what each quantity represents and why (or why not) it's useful in this experiment:

- Time: this is consistent across most coordinate systems, so in that it's self explanatory. It's of course useful, especially since each time step isn't uniform. So, in any physical system time is a useful metric.
- Latitude: this represents the angle of the poles, i.e. how many degrees from the North Pole. In a spherical coordinate system, this is but one of three vital metrics needed in order to understand position. As such, it is useful.
- Longitude: this represents the angle of the plane, which is perpendicular to the poles. This is the second metric needed in order to describe position. It is useful for the same reason as latitude.
- Altitude: this describes distance above sea level, perpendicular to the plane that longitude is measured from. An altitude of zero would suggest that the person is at or in the sea. This is not to be confused with absolute altitude, which measures distance between the object and the ground. Since this is the third metric for position, this is also useful.
- Altitude WGS84: this is just altitude adjusted for the eccentricity of the Earth's shape using the WGS84 system [1]. It's not useful, since in my work I assume that the Earth is not an oblate spheroid, but a perfect sphere.
- Speed: this is the directionless velocity $|v|^2$ produced by these measurements. Since it's a magnitude of the measurements and not a vector decomposition, it is not particularly useful unless the user would like to predict $|v|^2$ and compare to these measurements (at least for this experiment).
- Direction: this is the angle that the velocity is moving. This can be used to decompose speed into the vector form. Since I'm only predicting position, velocity as a measure is not needed unless the user wishes to predict velocity or generate a time-specific equation of motion. However, translation between coordinate systems must be considered here.
- Distance: the directionless distance $|s|^2$ produced by these measurements. Like velocity, this doesn't tell us much unless decomposed using direction. I opt not to use this measurement, as I define my own coordinate system which could contradict the results by not accounting for the translation. The same applies to speed.

- Horizontal Accuracy: this is the standard deviation of measurements in the x and y coordinates. This is useful in adjusting χ^2 to account for the reliability of the measurement. Primarily, these accuracies occur due to errors on the system level that the user can't avoid, such as the GPS satellites own accuracy.
- Vertical Accuracy: similar to horizontal accuracy, but for the z coordinates. It is useful for the same reason.
- Satellites: this tells you how many GPS satellites were involved in producing that measurement. More satellites result in more measurements, which increases the density of those measurements. So, it gives you a more focused answer at the cost of measurements being spread a bit more. It is desirable to void the results where there are no satellites as the truth to the measurement cannot be spoken to.

As such, the measurements I wish to include are time, latitude, longitude, altitude, horizontal accuracy, and vertical accuracy. From this, I can develop my own coordinate system for this experiment. Although, the number of satellites is used in cleansing the data such that measurements of 0 are disregarded.

Theory

Geographical to Cartesian Coordinate Systems

Consider that position data is recorded by three variables: altitude, latitude, and longitude. The units provided by `phyphox` are meters, degrees, and degrees. Really, there's only a slight difference between those conventions and spherical coordinates. As such, I'll denote these z, θ, and ϕ respectively. A simple illustration of the geographical coordinate system can be seen below:

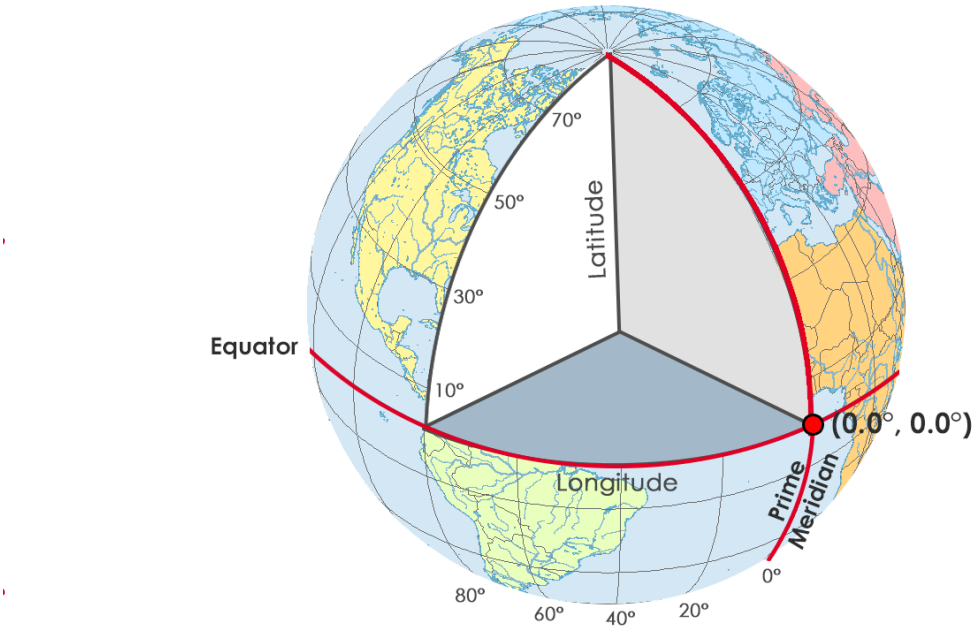

Figure 3: an illustration of the geographical coordinate system as projected onto Earth. [2]

However, the most important consideration here is that the Earth is not a perfect sphere. Rather, it is an oblate spheroid or, simply put, a sphere with eccentricity greater than zero. So, the closed form conversion to a Cartesian coordinate system isn't quite a reflection of the spherical conversion.

But, for the purposes of this project, I opt to work under the assumption that the Earth *is* a perfect sphere. This is because the distances covered in the data are not large enough such that the eccentricity of the Earth are observed. After all, travel outside of the country is quite heavily restricted right now.

With that, we can observe the projection of the spherical coordinate system onto the Cartesian coordinate system:

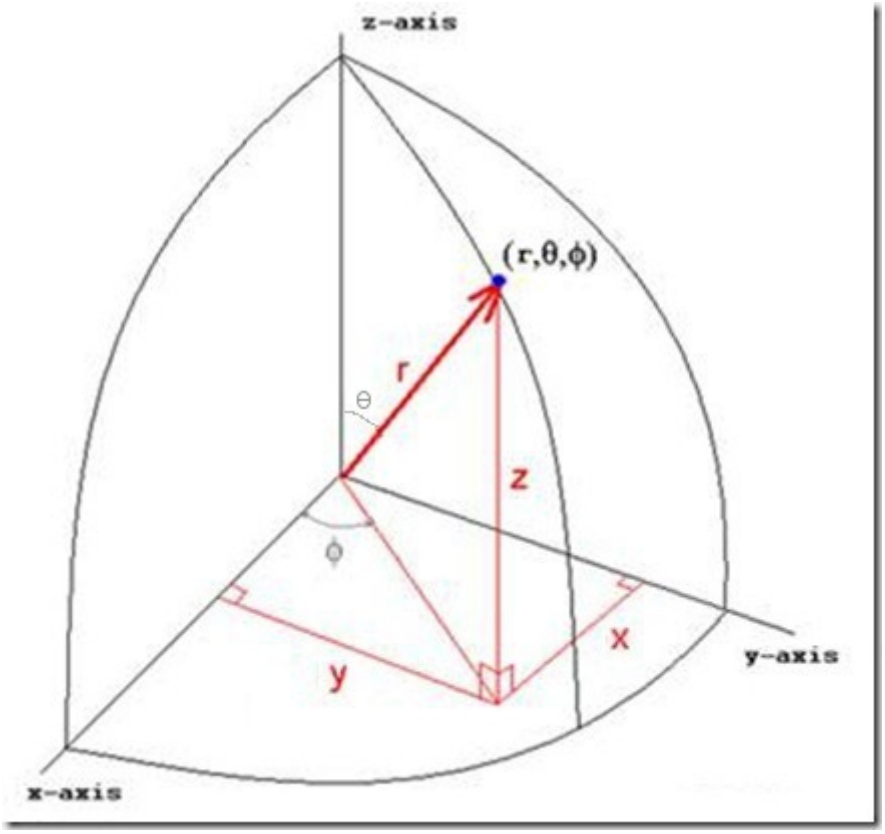

Figure 4: the spherical coordinate system projected onto the Cartesian coordinate system. [3]

The conversion between spherical and Cartesian are then given by the generic equations

- $x = r\cos\phi\sin\theta$
- $y = r\sin\phi\sin\theta$
- $z = r\cos\theta$

where $r = \sqrt{x^2 + y^2 + z^2}$. The conversion from geographical to Cartesian is then simplified. Of those equations, most remain the same. However, consider the transformation $\theta \to \theta - \frac{\pi}{2}$, since θ in geographical is defined by another vertex. But, since they're right triangles, we know that it'll just be that subtraction to get the other angle.

Another important consideration is that $\sin\left(\theta - \frac{\pi}{2}\right) = \cos\theta$ since they're similar functions, shifted by a factor of $\frac{\pi}{2}$. The geographical coordinate system conversion is then:

- $x = r\cos\phi\cos\theta$
- $y = r\sin\phi\cos\theta$
- $z = r\sin\theta$

This allows us to convert the `phyphox` data to a form people more easily understand. Of course, a quick Google search will produce the answers they desire, but the intent is for this to be readable from start to finish with introductory science knowledge.

Differential Position

Once the data has been converted to a typical Cartesian system, the issue then arises that, for an ordinary person, these measurements still don't have a definitive meaning. What I mean by that is someone cannot easily interpret what 300 meters in x means. Of course, something like altitude, z, is a bit easier to interpret, since we have conventions for altitude as is.

As such, I intend to introduce the quantity s, composed of s_i which represents any element of the position vectors of x, y, z. In doing so, I redefine x, y, z from a continuous set of positions to a discrete set of positions, such that they can be indexed by the variable i. Since the data is already vectorized, this is a sensible change.

Hence, $(x, y, z) \to (x_i, y_i, z_i), i \in \{0, 1, 2, \dots, N-1\}$ where N is the size of the data. With this, we can introduce a means to interpret the data as the distance from the starting point. Normally, this would be defined as $ds(t) = s(t) - s(0)$, where $ds(t)$ represents the distance from the starting point at time t. However, since the data has been discretized we can redefine the quantity to be $ds_i = s_i - s_0$ or by its components:

- $dx_i = x_i - x_0$
- $dy_i = y_i - y_0$
- $dz_i = z_i - z_0$

This effectively shifts the data such that the starting point begins at the Cartesian origin, $(0, 0, 0)$. It is important to note that it doesn't fundamentally transform the data, as in the shape and trend of the data is the same (but shifted). However, in physical interpretation, it is easier to understand that I'm 10 meters from where I started than I'm at 300 meters. Of course, we also have the discretized time vector $dt_i = t_i - t_0$. However, the data begins at $t_0 = 0$ seconds in almost all cases. See Appendix A for more on this.

We could simplify this further by taking the absolute value of the quantity ds_i, but preserving direction is important when it comes to examining stuff like velocity and acceleration. In simplifying information, it's important to still be able to retrieve important qualities and taking the absolute value deprives us of perhaps one of the most consequential physics properties: direction.

Time Equations of Motion

Since the data is in Cartesian format, the equations of motion can be easily derived. First, we acknowledge that $F = m\ddot{s} = ma$, where F is an arbitrary force, m is the mass of the system, $\ddot{s}$ is the acceleration of the system, and a is an acceleration constant. This is an extension of Newton's Law of Motion. Since we're not in free-fall (I would hope so) $a \neq g$ such that g is the acceleration due to gravity. However, we also assume that $a = 0$, since I moved in a straight line with constant velocity. Here, we also assume that the force is not subject to Newton's Law of Universal Gravitation. This is because that function is dependent on r in and, for this experiment, $dr \approx 0$ (due to distances inconsequentially changing) such that no velocity would be observed, which is contrary to real-life motion.

Then, the initial velocity is something that must be experimentally determined. That'll be addressed later. We can then derive the velocity of the system by integrating both sides such that

- $m\ddot{s} = m\dfrac{dv}{dt} = 0$
- $dv = 0dt$
- $\int_{v_0}^{v} dv = \int_{t_0}^{t} 0dt$
- $v - v_0 = 0$
- $v = v_0$

However, by convention we define t_0 = 0 seconds such that $v = v_0$ where v_0 is the initial velocity and v is the velocity of the system. We can further take advantage of the fact that $v = \frac{ds}{dt}$ such that

- $\frac{ds}{dt} = v_0$
- $ds = v_0 dt$
- $\int_{s_0}^{s} ds = \int_{t_0}^{t} v_0 dt$
- $s - s_0 = v_0(t - t_0)$
- $s = v_0(t - t_0) + s_0$

Once again, we can take advantage of $t_0 = 0$ seconds, but also that $s_0 = 0$ meters from us shifting the coordinate system. As such, $s(t) = v_0 t$. We can then generalize this for our coordinate system such that:

- $x(t) = v_x t$
- $y(t) = v_y t$
- $z(t) = v_z t$

If we wish to include a higher-order term such as jerk or acceleration (assuming you're aware of the systems at play), methodology can be found here. With that, we can make decisions on the appropriate fit such that it follows the form of this equation and, thus, the physics.

Fitting the Time Equations

Consider that in the previous section we deduced that $s(t) = v_0 t$. This motion takes on the form of a linear equation. If we generalize it further, we can state that $s(t) \approx c_1 t + c_0 = f(t)$, where c_n represents a constant associated with that motion. c_0, for example, is related to the initial position, c_1 the initial velocity, etc. Although we assume that $s_0 = 0$, c_0 might not be zero due to error in fitting the curve.

Then, for a set of points P such that size(P) $> N + 1$, where N is the highest order, we can utilize the fact that we can generate a matrix $T\vec{c}$ of size $N\times$ size(P), which linearizes the equations. Thus,

$$T\vec{c} = \begin{bmatrix} c_0 + c_1 t_1 \\ c_0 + c_1 t_2 \\ \cdots\cdots\cdots \\ c_0 + c_1 t_P \end{bmatrix}$$

which can be further broken down into T and $\vec{c}$ by factoring out the constants such that

$$T = \begin{bmatrix} 1 & t_1 \\ 1 & t_2 \\ \cdots & \cdots \\ 1 & t_P \end{bmatrix}$$

$$\vec{c} = \begin{bmatrix} c_0 \\ c_1 \end{bmatrix}$$

Of course, this can be generalized to higher dimensions, but for our focus this is the highest it should go. But this begs the question, what about the solutions, $\vec{s}$? These can also be generalized in the same way such that

$$\vec{s} = \begin{bmatrix} s_1 \\ s_2 \\ \cdots \\ s_P \end{bmatrix}$$

We then have the vectorized equation $T\vec{c} = \vec{s}$. This is a Vandermonde Matrix. However, since this matrix cannot be square (size $N \times N$ or $size(P) \times size(P)$) in order to have sufficient data , then we cannot apply Gaussian elimination without producing linear dependence. But we can take advantage of the fact that a $(n \times m) \times (m \times n)$ matrix multiplication produces a $n \times n$ size matrix, whereas a $(n \times m) \times (n \times q)$ multiplication produces a $n \times q$ size matrix.

We can utilize this knowledge by multiplying both sides of the equation by the transpose of T, T^T, such that $A = T^T T$ and $\vec{b} = T^T \vec{s}$. Thus, after applying this, we have $A\vec{c} = \vec{b}$, which can be row-reduced since the matrix is now non-singular (nonzero determinant) and square.

Solving this equation then produces the constants from the original equation $s(t) \approx c_0 + c_1 t = f(t)$ such that we now have an approximation to the position. Typically, the error to this fit is not described by something akin to χ^2 such that $\chi^2 = \sum_i^{size(P)} [\frac{s(t_i) - f(t_i)}{\sigma_i}]^2$, where σ_i is the standard deviation associated with the measurement of $s(t_i)$. Further, root-mean squared error is given by $RMSE = \sqrt{\frac{\chi^2}{size(P)}}$. Ultimately, these tells us the goodness of fit, but don't present themselves as a vectorization of that quantity. Alternatively, we can just observe $|s(t_i) - f(t_i)|$ vs. t_i to see how the error trends at each vector element. Since this is predicting real data (not an exact function), this measure won't really tells us much if we're looking to observe some trend in error.

Another common means of fitting raw data is producing a Lagrange Interpolating Polynomial. This particular method is nice in that it has to pass through all the reference points by the equation

$$P_n(t) = \sum_j^n s(t_j) \Pi_{i \neq j}^{n-1} \frac{t - t_i}{t_j - t_i}$$

where $P_n(t)$ represents the interpolating polynomial with n reference points. So, for enough points, it should exactly approximate the raw data. However, this fails to represent the physical model, since a highly-accurate model would require many near n points. This would produce a near t^n polynomial, which becomes a problem because the highest that we can observe on this scale is likely jerk, $\mathcal{O}(t^3)$. So, although it may provide an accurate polynomial, it does not represent the physics of the system and doesn't tell us anything about physical constants such as v_0, a, etc. Least-squares can generalize higher order motion via the Vandermonde matrices anyway, as long as we know the system driving it.

Further, I will not smooth the data by a Fourier transform because there is no periodicity to it explainable by physical systems. Because of that, cleansing would not be algorithmic in most cases; rather, it would be case-by-case looking for inconsistencies in that particular data. A better solution would be to follow the same path with the exact same initial conditions over and over again, then average the data. However, even that isn't feasible since the data produced by `phyphox` doesn't have consistent time-steps such that you could match-up entries easily. So, the average of K trials is also ignored due to the structure of the data. Regardless, we can supplement missing data by repeating the experiment as described and just combining each set, assuming the initial conditions are nearly identical.

As such, I opt to follow the least-squares path instead since the matrix is formed by the actual predicted equations of motion. Further, if we were to opt to use 3 points, the choice in them is arbitrary. In such a large set of data, the user has to choose which points to interpolate. This can't really be optimized without a rigorous algorithm that the motion doesn't necessary fit in all cases. Instead, least-squares makes that decision for you by producing the correct order for a physics polynomial while implementing all points. Now, we can move on to eliminating time dependence in our equations of motion.

Eliminating Time Dependence

In order to eliminate time dependence, i.e., treat $x(t)$ as $x(y, z)$, we can take advantage of the fact that the time domain, t, is constant between all three vector components $< x, y, z >$. As such, we can rearrange the equations such that

- $t(x) = \frac{x}{v_x}$
- $t(y) = \frac{y}{v_y}$
- $t(z) = \frac{z}{v_z}$

We can then substitute these time equations into any of the other position equations such that

- $x(y) = v_x t(y) = \frac{v_x}{v_y} y = \frac{v_x}{v_z} z$
- $y(x) = v_y t(x) = \frac{v_y}{v_x} x = \frac{v_y}{v_z} z$
- $z(x) = z(y) = v_z t(x) = v_z t(y) = \frac{v_z}{v_x} x = \frac{v_z}{v_y} y$

Consider that we opt to totally eliminate z from the equations because of the real possibility that elevation doesn't change. If that were the case, then quantities such as $x(z)$ would blow up to infinity when $v_z \to 0$, which is not reflective of the motion. Rather, it is safer to assume that x and y will always change due to the nature of the geographical coordinate system. See Appendix B for more.

With this, we produce a position vector $r' =< \frac{v_x}{v_y} y, \frac{v_y}{v_x} x, \frac{v_z}{v_y} y >$, which we can compare to the actual position vector as defined by $r =< x, y, z >$. We can do this in two ways: first by observing a 3D plot such that there is no time dependence. The associated error would just be $|r - r'|$, which can also be represented in a 3D plot.

These proportions can be found by calculating least squares, treating it as you would in the previous section. This time, with x as an example, x would be the output and y the input. You should not be able to use the coefficients calculated in the previous section unless there is perfect accuracy.

However, we can also take the magnitude of the vector such that $r = \sqrt{x^2 + y^2 + z^2}$ and compare that to the magnitude of the prediction, or $r = \sqrt{\frac{v_x^2}{v_y^2} y^2 + \frac{v_y^2}{v_x^2} x^2 + \frac{v_z^2}{v_y^2} y^2}$. This is only a brief mention, as this equation would be time dependent.

With that, there is no more theory to establish. Any comparisons, such as comparing our methods to `np.polyfit()`, will be seen in the actual notebook. Thank you.

Conclusion

In the end, it seems that the big take-away is that motion overall more strongly correlates with its other vector components than it does time. Truly, this does make sense - time is a non-factor, i.e., time itself will not change with position, but linearly. However, position will correlate more strongly against itself because we're examining each individual component. So, hypothetically, our equations of motion are all dependent on each other. And we show that, by showing that the equations can be rewritten in terms of each other. Even further, something like v_x can be considered in that $v = \sqrt{v_x^2 + v_y^2 + v_z^2}$, so it could be reduced more.

However, much of the barrier in fitting the time data came with noise. This could perhaps be reduced via Fourier analysis, but there exists an assumption such that all data of the same procedure will have the same noise. This is a falsehood, so applying that generically is rather difficult. Instead, it's something that has to be tweaked with to fit the data specifically.

Regardless, the strict linearity of the position data is useful. It's useful because it tells us which velocities are dominant, by virtue of the least-squares constants being a proportion of the quantities. In this, we can tell which direction of motion is more influential. This linearity is also useful because, by eliminating time dependence, we can describe motion fairly accurately via other position alone.

Surely, there are more things to be said about the data - however, I am unable to conclude those exact things, since the experiments themselves weren't exhaustive. Perhaps a further exploration of data could show more complicated physics, especially in acknowledging the time-dependent noise.

Discussion

Overall, the feeling leaving this project is that I wish I had more time. I wish I had more time to thoroughly explore other types of motion, or applying this data in other physics such as energy or momentum. But it is what it is. Regardless, I think that this collectively satisfies all of the objectives, so in that I achieved the minimum.

However, if I could improve one thing in the immediate it would be the data. It has a lot of statistical noise that even walking several trials could not fix. I think perhaps with $N > 100$ trials, or a more smooth walking path, the data would become more clean. If I had more time, I would outline a very specific path or simply just find a larger stretch of line. Or maybe my phone just isn't the best design for this experiment. There are too many factors to be certain.

It would also be neat to implement the WGS84 coordinate system such that the coordinate conversions more accurately map to the shape of the Earth. There is literature out there, but much of it is bickering such that I wasn't really sure what they were trying to convey. Who knew people could be that pretentious over something so niche?

Ultimately, there's plenty to expand on. Surely, fitting the motion isn't the end of a project of this scope. Perhaps I'll return to this eventually, but for now I think the assignment is done and dusted.

Appendix

[A] Consider that the `phyphox` data is for the most part reliable. However, depending on the phone and its operating system, there are some exceptions such that the data faces issues. Namely, if the GPS experiment is started and it cannot ping the cellphone, then the first position will be the last known location. I personally haven't observed this elsewhere in the time domain, so the fix for this is easy enough. It does have standard deviation in the data such that you can reasonably adjust for mistakes, but it can't always be incorporated. Overall, I've seen only but a few issues with time and position being misreported, but nothing that should affect the experiment since the data is cherrypicked as is.

[B] However true this is, my actual data reflects a change in the z, so I could have totally run with the functions being dependent on altitude. Despite this, I wanted to generalize as much as possible, since the standard deviation of the z is rather large after all, such that the actual values could all be 0 when examining dz. And also, the cases of others who may read this might not reflect mine.

References

[1] https://confluence.qps.nl/qinsy/latest/en/world-geodetic-system-1984-wgs84-182618391.html

[2] https://gisgeography.com/decimal-degrees-dd-minutes-seconds-dms/

[3] https://rbrundritt.wordpress.com/2008/10/14/conversion-between-spherical-and-cartesian-coordinates-systems/

Jeremy Kazimer

jdkazime@buffalo.edu