# Overview

When designing simulations, controllers, and sensor data filters, the use of linearized state space equations is huge. State space equations are one of the most widely used ways to represent physical systems in a condensed one mathematical equation. The major drawback to a state space function is that it makes use of linear algebra; thus it cannot represent nonlinear systems. The often used way of negating this issue is to linearize the physical system at the point of interest to approximate the nonlinear system as a linear system. This function approximation can be repeated as needed if the point of interest strays far away from the point being approximated.

This begs a question though: When is an equation so nonlinear that I need to resize my re-calculation bounds or re-evaluate my state space model as a whole? If you ask your professor or employer this question, they will likely answer, “It is up to the engineer to justify if their approximation is accurate enough to represent a system.”. While this is very true, it’s also not very helpful because it plants the go / no-go decision on subjective debate alone. It means that one engineer could think differently from another engineer and even crazier is they could both be right. Its also impossible to teach a computer algorithm to make a subjective decision, because computers operate objectively; not subjectively or probabilistically. Scientists and engineers usually overcome this issue by learning how to quantify a debate and then standardizing the quantification for acceptability. For example, scientists and engineers have agreed to standardize 30 or greater as the number of sample points in a “large” sample size versus 29 or less representing a “small” sample size.

In this paper I will explain my derivation of an analytical and numerical m-code algorithm for quantifying nonlinearity of an N-dimensional function with clearly defined boundaries. I will then use crowd-sourced data to propose a standardized set of rules for when a bounded set of variable domains is “too nonlinear” for a standard 1st order Taylor series to approximate. This paper will not attempt to standardize the acceptability of biased 1st order Taylor series.

# What is nonlinearity?

A function is linear when its’ entire domain for all dimensions (from -infinity to +infinity) has a constant slope. By extension, a nonlinear function has an inconsistent slope. In the world of pure mathematics that is all there is to it. However in the world of practical mathematics there is room for caveats. Usually we do not classify a function’s linearity from -infinity to +infinity, because that is impractical and unnecessary. Instead we classify linearity based on a set of bounds that constrain the area of interest to a more practical set of values. Additionally there are many equations which are technically nonlinear, but are so close to being linear under the described bounds, that it really doesn’t matter. To illustrate this, see Figure 1, based off of Equation 1 as an example.

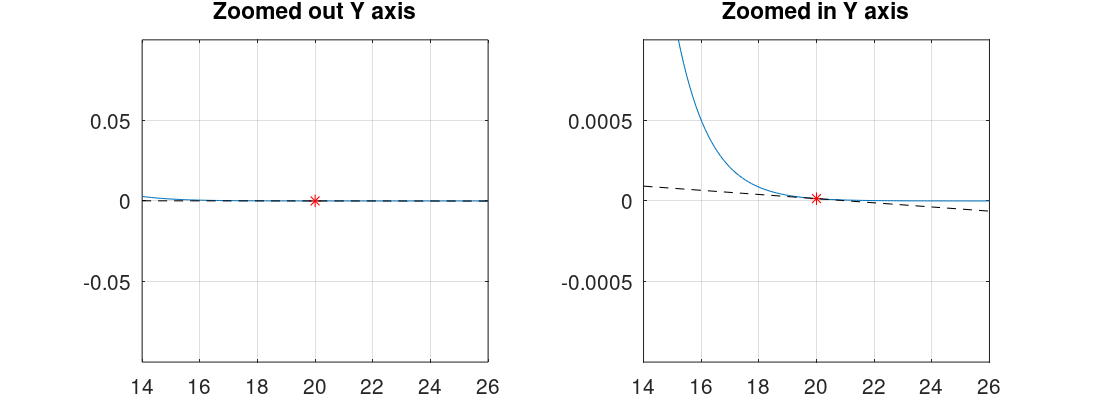
Chart, scatter chart

Description automatically generated

**Figure 1A, 1B, & 1C: Three different snapshots of the same equation but at different bounds**

As you can see, this equation is highly nonlinear for the bounds described in 1A, but for the bounds in 1B and 1C, it is very close to a linear system. All 3 plots are of the same equation, but for different bounds of interest. Figure 1B has the same center point as 1A, but is zoomed in to a smaller set of bounds. Figure 1C has the same scale bounds as 1A, but at a different center point. Therefore it is not acceptable to rationalize the linearity and accuracy of an approximation if only given the center point or only the radius of interest. It is best to define bounds of the format [X1, X2]. This idea extrapolates to N dimensional equations of course.

There is another factor to consider, however. Consider Equation 1 under the bounds of [14, 26]; such was Figure 1C. Depending on how we choose to scale the y axis, we can make the graph visually appear more or less linear. This is even though the center point, radius of interest, parent equation, and linearized equation remain unchanged.



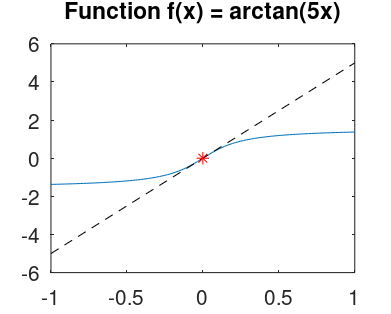
**Figure 2A & 2B: Two different snapshots of the same criteria, only differing in Y axis scale**

What does this mean, and if we can technically do this to any equation why do we accept any approximations? This is where an awareness of your system comes into play. If I am measuring the dimensions of my bedroom and my approximate measurements are off by 1 centimeter, that is not significant for arranging furniture in my room. However, if I am designing a mirror for interferometry or a telescope, a 1 centimeter measurement error is completely unacceptable. There are a couple of ways we can select a proper scale for the y axis. One way is we can mitigate this subjectiveness by converting our measurements from absolute to relative.

An absolute value or measurement is one which is taken relative to some global value, such as the X axis for example. A relative value or measurement is one which is taken relative to itself. Therefor a relative measurement eliminates the relevance of a global Y axis scale because all of the points of interest are taken relative to some other factor, like the center point of interest. In measurement theory, sensor uncertainty can be quantified as absolute or relative. An absolute measurement uncertainty would be something similar to +/- 1 cm error. And that error would be the same for any distance measurement you make. A relative measurement uncertainty would be something similar to +/- 0.5%. This is the possible percent error relative to the “true” measurement. 0.5% = (abs(measured-true)/true). In a similar way we can turn our measurement point from absolute to relative based on the center point of interest.

The method is not without flaw however. Take the lift coefficient as a function of angle of attack of an aircraft for example. It has been shown through potential flow analysis that a good approximation for all streamlined bodies is CL = sin(α) for values near α=0. So 0 degrees alpha is our point of interest. But if we try to use this point of interest we end up with the equation, error/0. A division by zero error is undefined in mathematics and infinity in MATLAB. Unfortunately using a relative approach for values near zero will almost always lead to bad outputs due to the division by zero. It is more practical in this case to use another factor as our relative point or to simply analyze the absolute error. We will revisit this phenomenon in our later derivation. From this however we can also ascertain that the scale of our function outputs with respect to our system accuracy requirements also plays a role in the acceptability of our approximations.

Approximation error IS NOT the same thing as nonlinearity. This is a very important distinguishment. The two can be observed to be directly correlated however, there are some rare instances where a taylor series approximation is awful for a relatively linear function, due to poor luck with which point of interest you linearize around.



**Figure 4: An example of a mostly linear function, which happens to have a poor Taylor series approximation with a lot of error.**

This is not to say that such functions are common, but to distinguish between approximation error and nonlinearity. The two concepts are discussed further later in the paper.

# Quantifying nonlinearity

It is time to throw some equations at the problem. For the reader’s convenience I will show plots of the form f(x) and f(x,y) since both can be visually perceived well, but we will use derivations with respect to an N-dimensional function, such that the concepts shown for f(x) and f(x,y) can confidently be applied to functions with 3 or more inputs as well.

As mentioned previously, a perfectly linear snapshot of a function is one which maintains a constant slope in all directions. This is a standard line or plane equation, and for N dimensions can take the form of Equation 2 below:

Where:

* Xn – the nth input variable to the function f.
* Cn – A constant coefficient for the nth input variable. Can be any constant value from -infinity to +infinity including 0.
* B – Bias constant. Can be any constant value from -infinity to +infinity including 0.

Any equation which cannot be described in this manner is nonlinear in some way. Something unique to equation 2 is that if you calculate the gradient vector field of this equation, it is a uniform vector field.

As a quick refresher, a scalar field is a plot of scalar values. It is the kind of field, or plot, that we reference most of the time. Scalar fields come from a function which accepts some kind of input (scalar, vector, tensor, phasor, etc) and outputs a scalar value. A vector field on the other hand gives a vector as an output. Vector fields are useful for displaying a gradient of a function. A function gradient takes a scalar field and calculates a vector field, comprised of vectors pointing towards the “steepest” direction of change. In other words it produces a vector that can tell you the derivatives in all axes. See the example below in Figure 5.

Chart

Description automatically generated

**Figure 5: Nonlinear gradient example versus linear gradient example**

Like I mentioned before, something unique about linear functions is that they have a uniform vector field. This is because no matter where on the plot you are, the slope would be constant, and therefore the gradient is also consistent at all points. Furthermore, if the vector field is uniform and nonchanging then the calculation of the vector derivative should equal 0. In other words, if the derivative describes a change in a vector as you analyze nearby local points, and if there is no change in the vector field because it is uniform, then naturally, there is 0 change and the vector derivative is 0. Lets describe these two conjectures mathematically.

As we will see, the vector derivative of all gradient terms will be a very recurring and very important array. Therefore I have chosen provide a layer of abstraction, and label it as vector field U. U can be represented as a matrix, in the fashion shown above, or as a 1-dimensional array. The way you represent the U vector field will become relevant when we convert this data from a vector field, back into a scalar field.

Notice that when taking the vector derivative of a gradient, you can only do so with respect to one variable at a time. Therefore for a function with n input variables, you must take n vector derivative to capture the entire description of the vector field. If each individual vector derivative results in n number of equations to describe the resulting vector field, then after calculating all n vector derivatives with respect to all n input variables, you will have n2 results. This operation insinuates at least an O(n2) algorithm, which is typically considered highly undesirable. Luckily, due to Clairot’s theorem, we can simplify the above matrix as symmetric. Thus the operation will require considerably less then O(n2) operations.

You can analytically derive u for an equation, and the result will be a matrix of equations. If U is a matrix of zeros then the equation is linear. If the equation is nonlinear, then the U matrix will contain a collective set of equations to calculate U for any point in the function domain. The U value for a nonlinear function varies depending on where in the function domain you calculate U. The matrix of value you calculate for a nonlinear function is a direct representation of how nonlinear the function is at that particular point.

# Standardizing acceptance

Blah blah blah