Adaptive grid generation

Motivation:

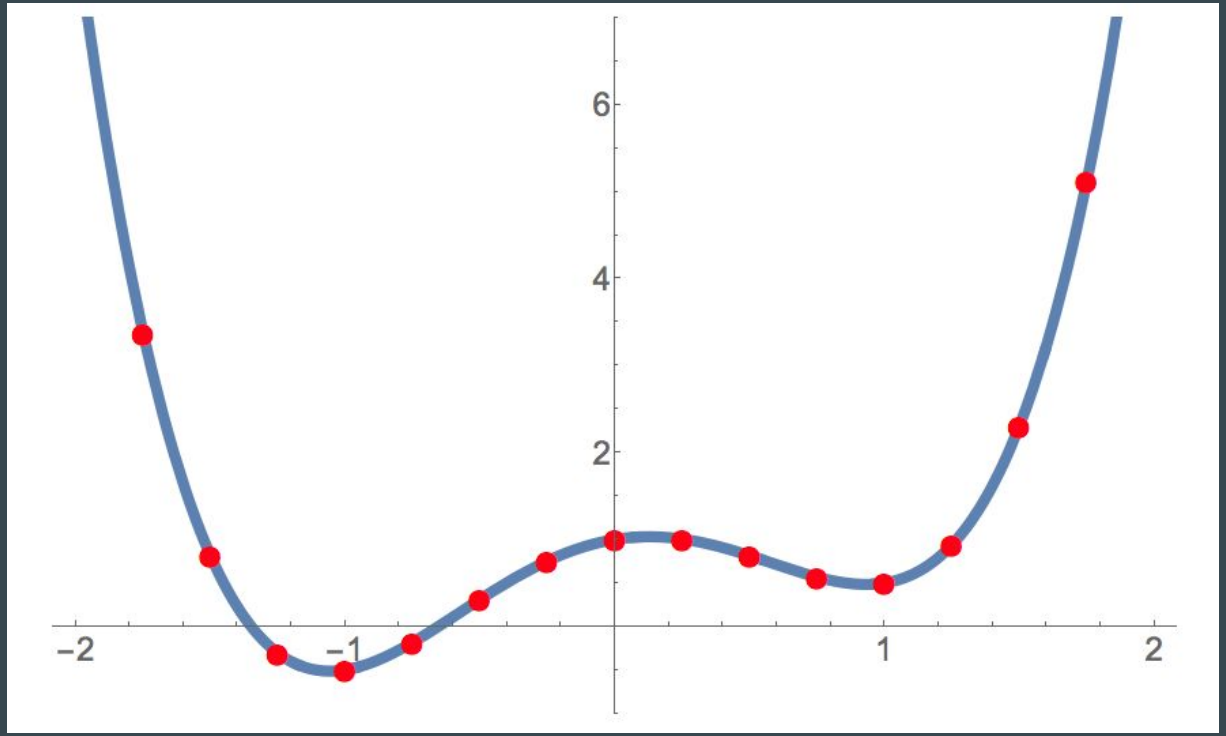
Given a cheap potential to calculate approximate energies for configurations in a given configuration space, devise an automated method for determining the fewest number of configuration space points with accurate energies needed for fitting a polynomial within a given region of interest.

Adaptive grid generation

Linear functions need the least number of points to describe them: 2. Conversely, the more curvy a function is, the more points we need to get good interpolation. Motivated by that trend, this method adds a point in between pairs of points that provide a poor linear approximation of the underlying cheap potential function.

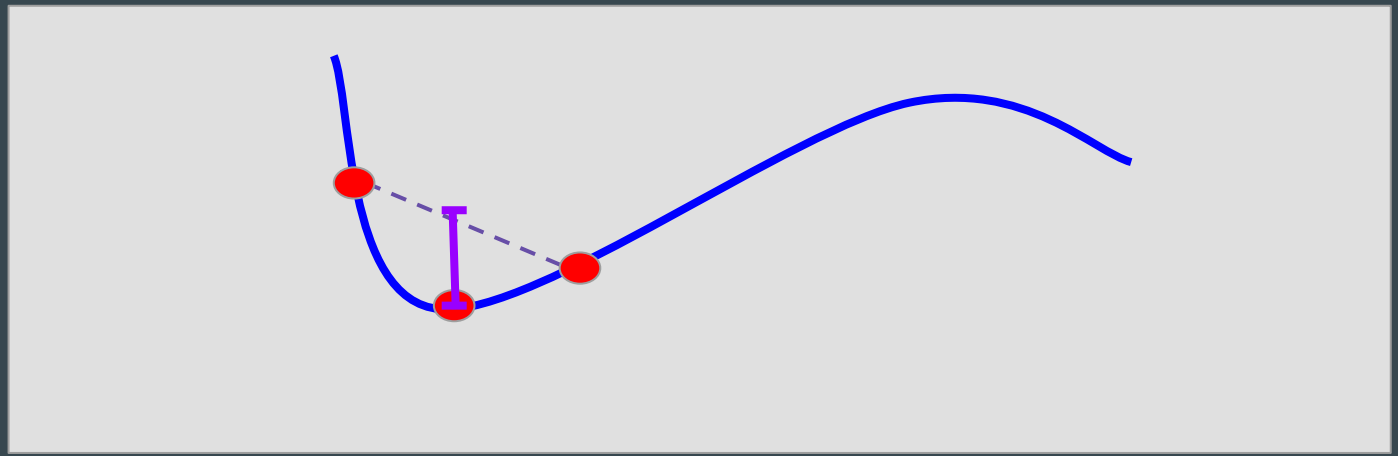
Adaptive grid generation

Start with a coarse, evenly spaced grid:



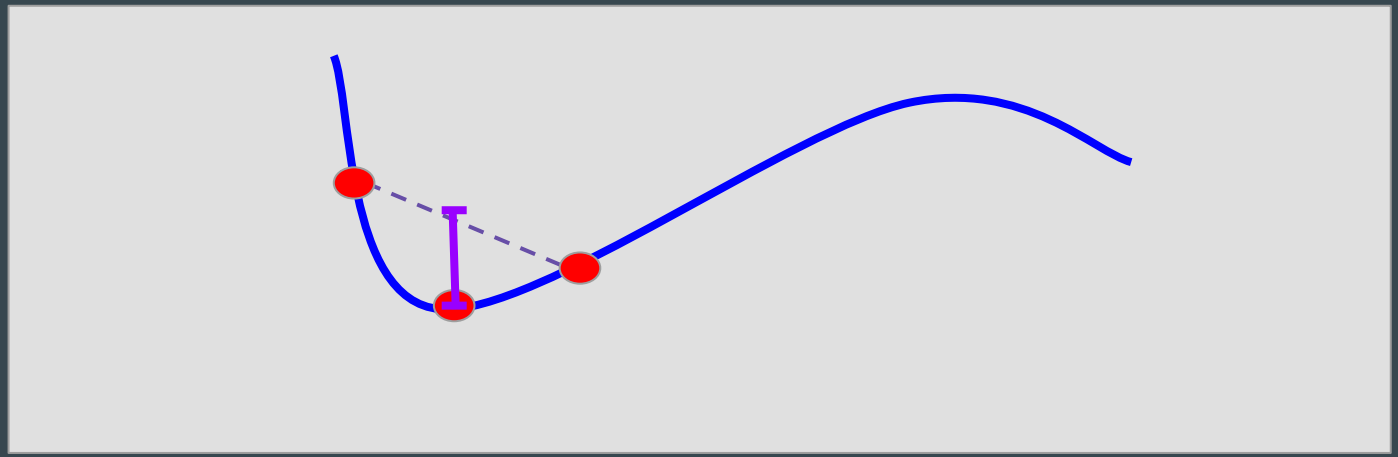
Adaptive grid generation

A test point is placed in between each pair of adjacent points. The energy of these points is calculated using the cheap potential and added to the set of known points. A difference is calculated between the energy from the cheap potential and a linear approximation between each pair of adjacent points (purple).



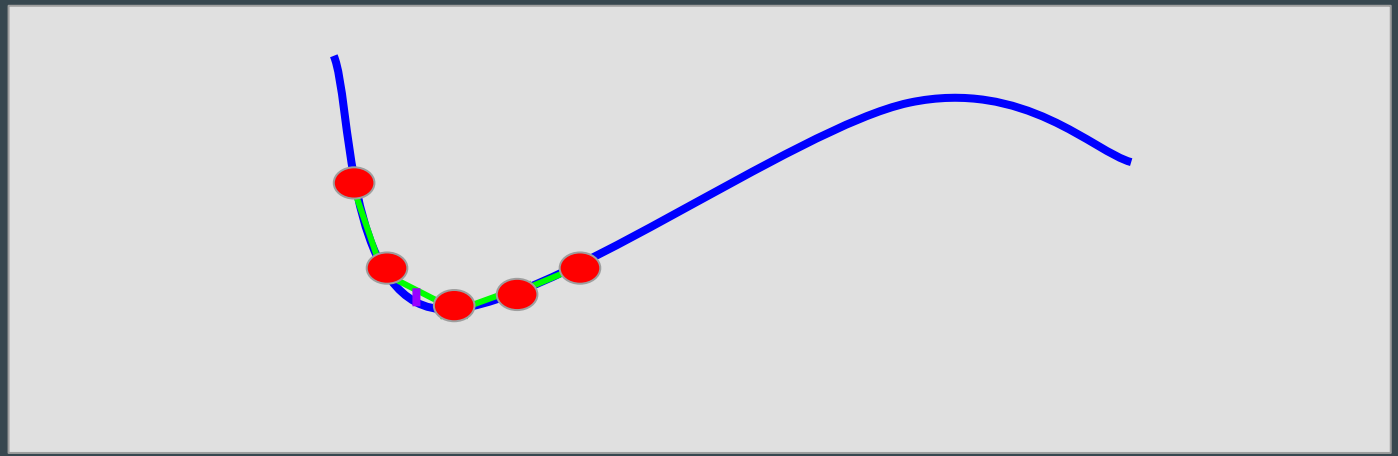
Adaptive grid generation

If the energy difference between the linear approximation and the cheap potential is below our chosen threshold, the three points are removed from the test list. If a failure occurs, new test points are placed between all failing pairs, and these new points are included in the test list along with the points on either side of them.



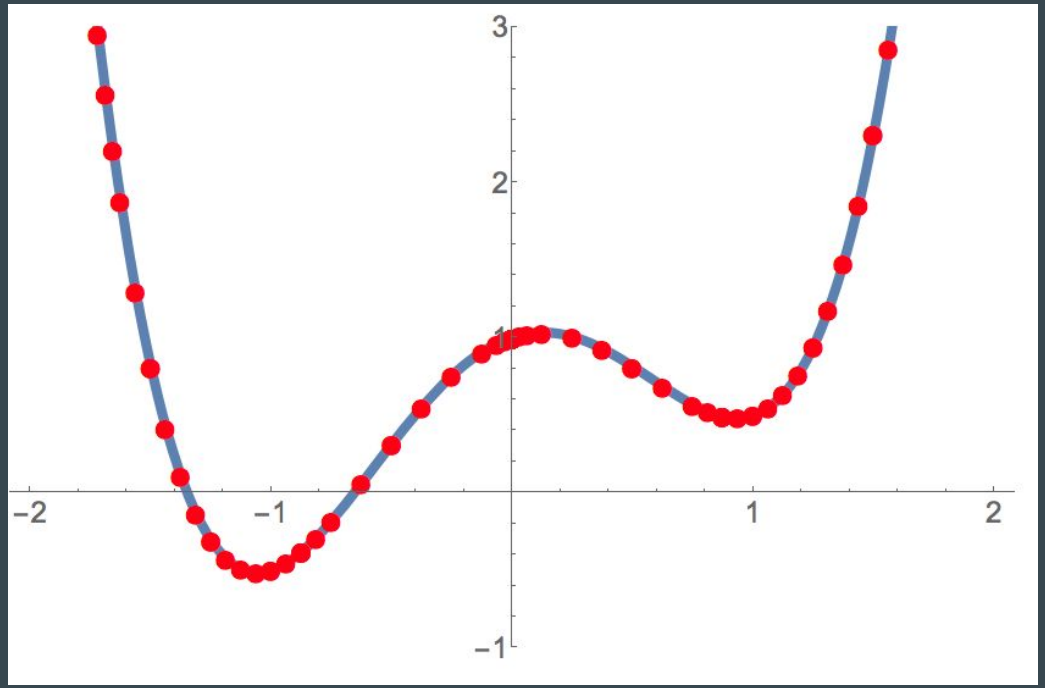
Adaptive grid generation

Let's say that this difference was above our threshold. We add two points halfway between the others, and measure the energy differences between the cheap energy potential and the linear interpolation between the new points (green lines). Here we see a much smaller difference than the previous iteration.



Adaptive grid generation

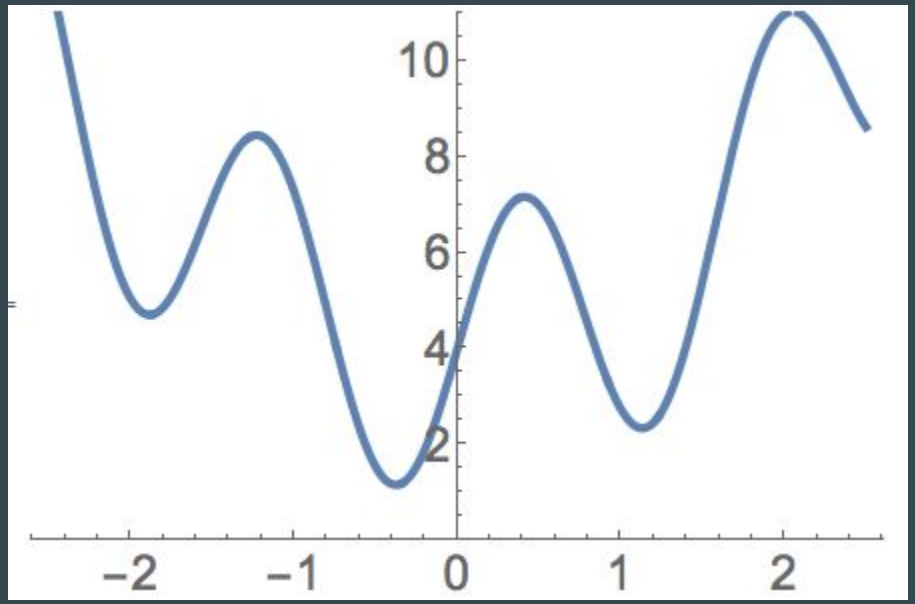
After iterating over this procedure until the test list is empty, we obtain the set of points that produce the best fit by linear interpolation:



Analysis of fit quality:

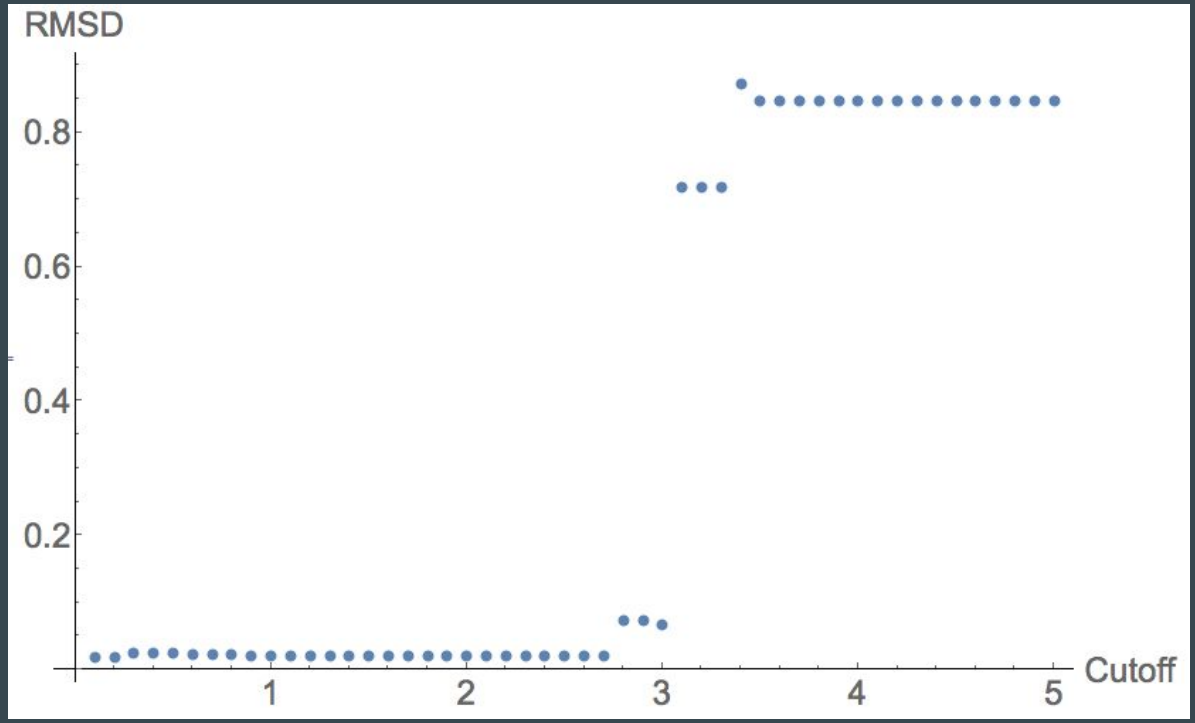
In this example we attempt to fit an 11th order polynomial to the function:

$3\,\mathrm{Sin}(4\,x) + x^2 + 4$

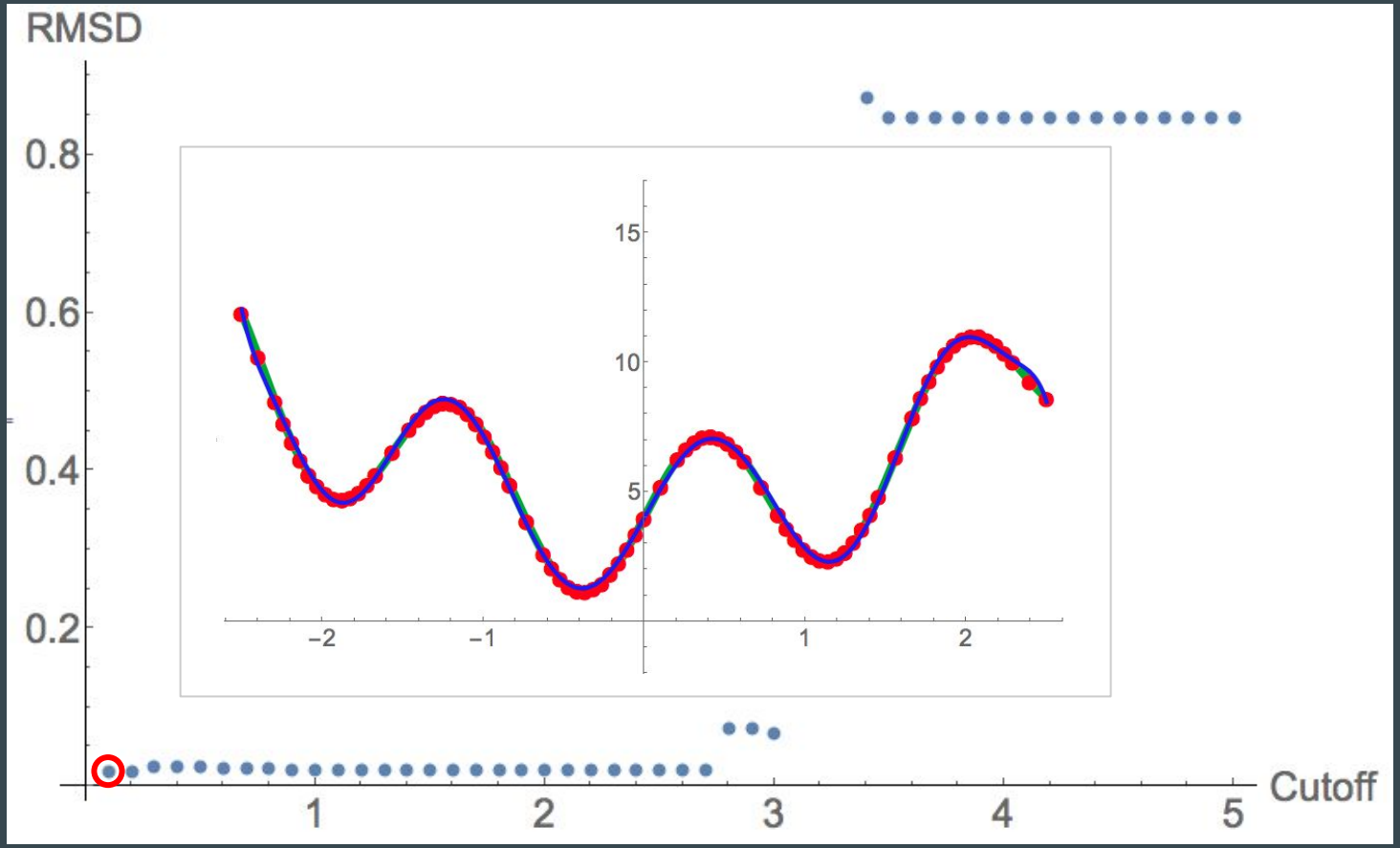


Analysis of fit quality:

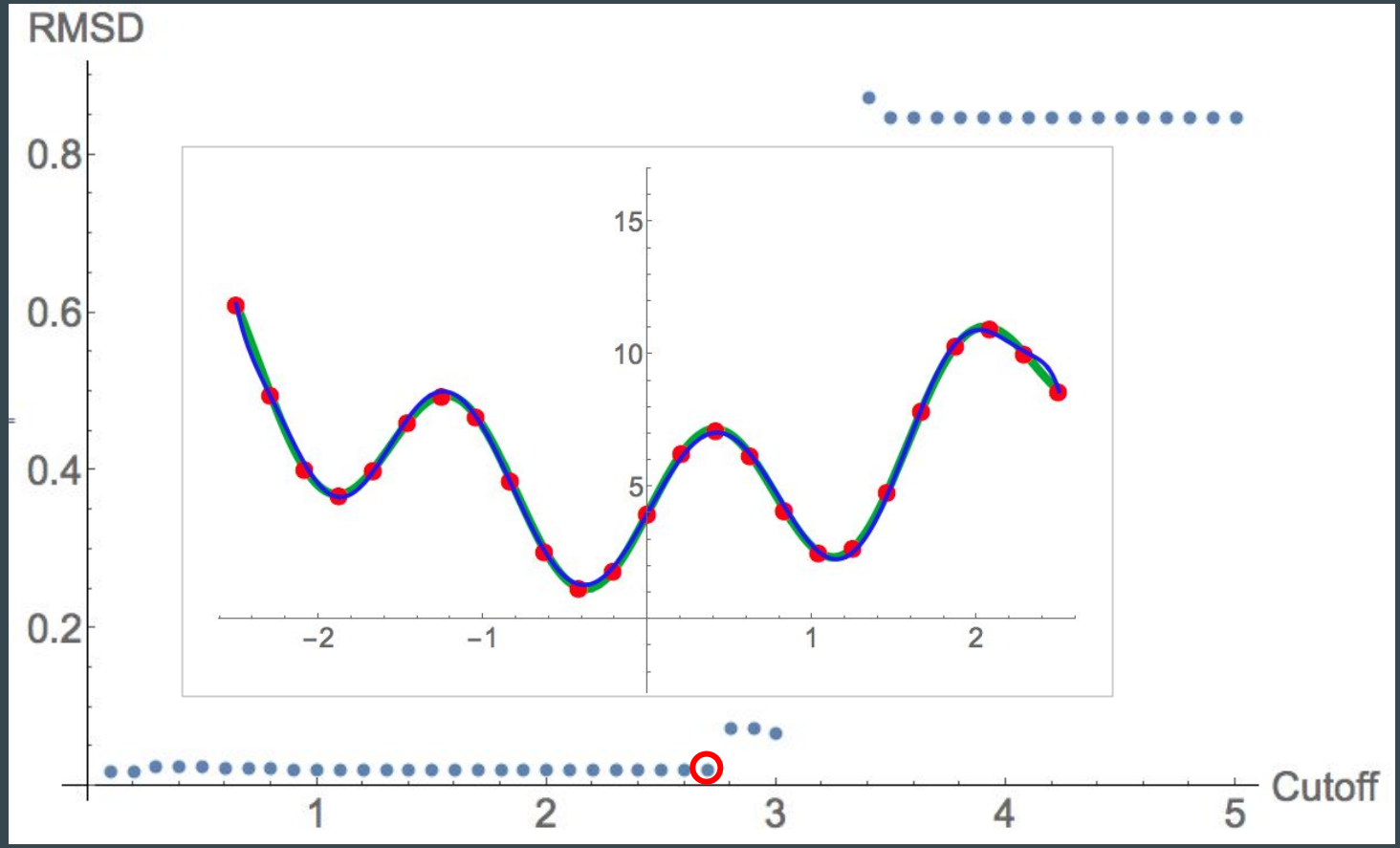
Here we have RMSD of the polynomial fit as a function of the cutoff value used in the adaptive grid. Note that the RMSD is approximately constant for quite some time before making a few jumps.



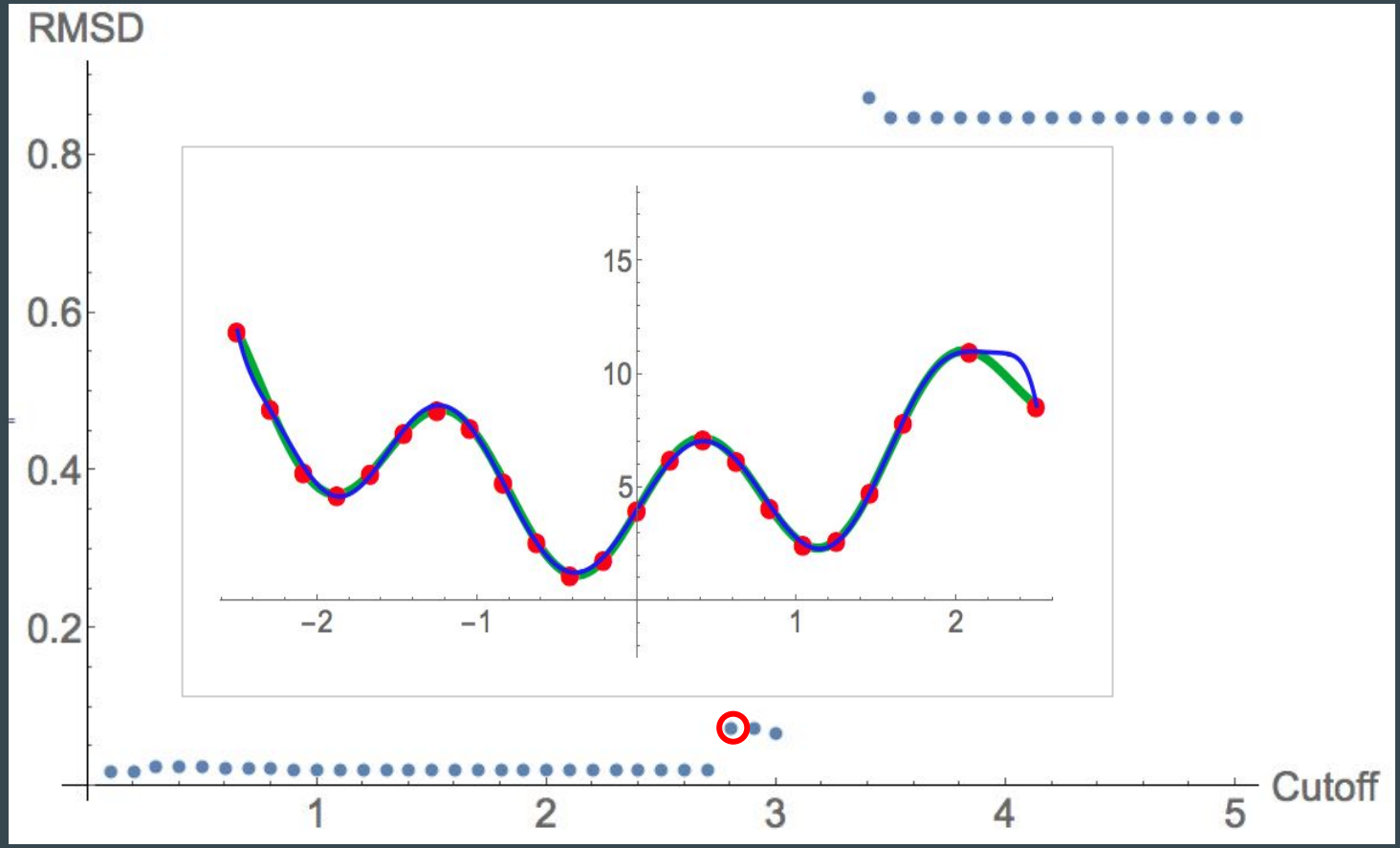
Analysis of fit quality: individual points



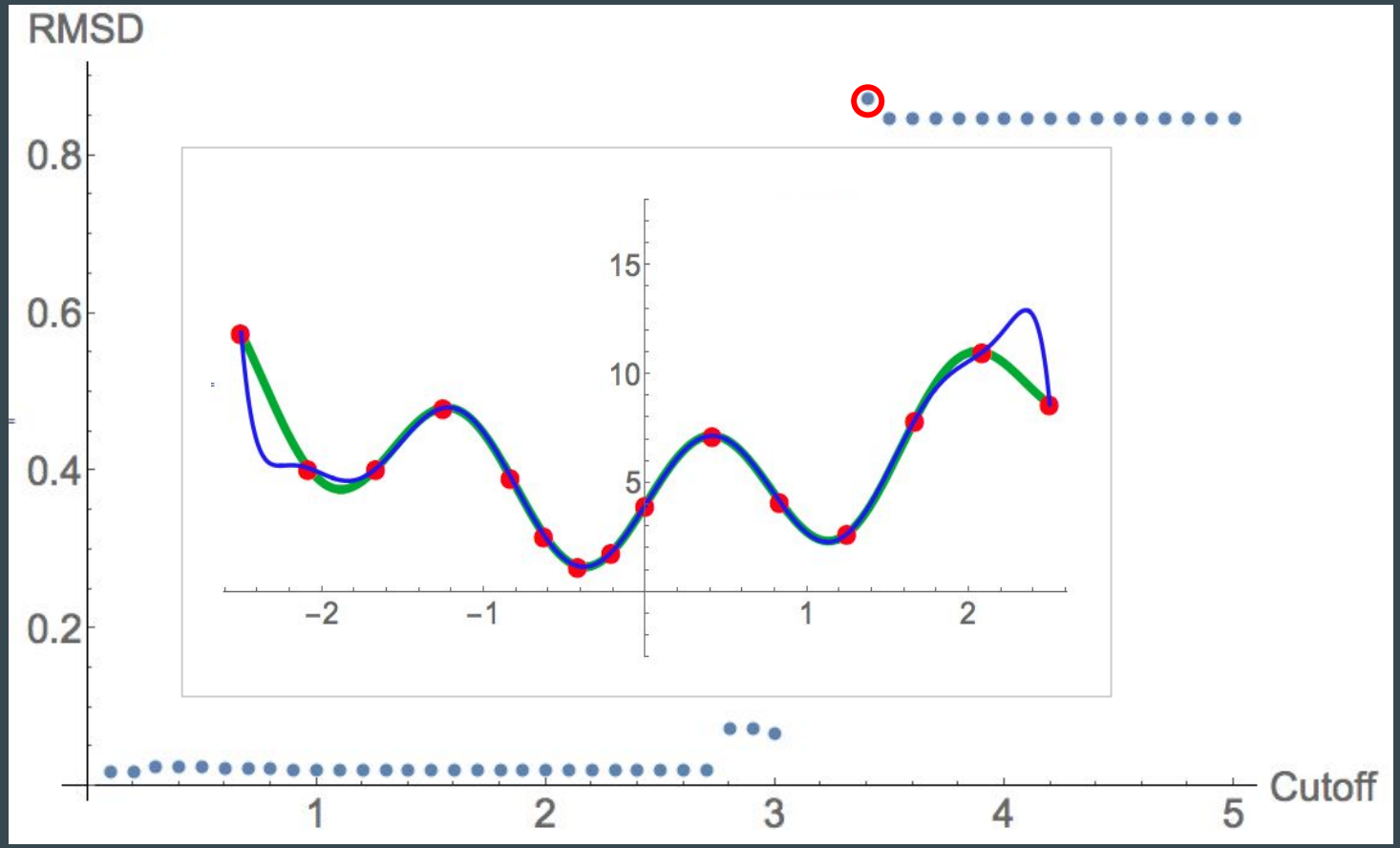
Analysis of fit quality: individual points



Analysis of fit quality: individual points

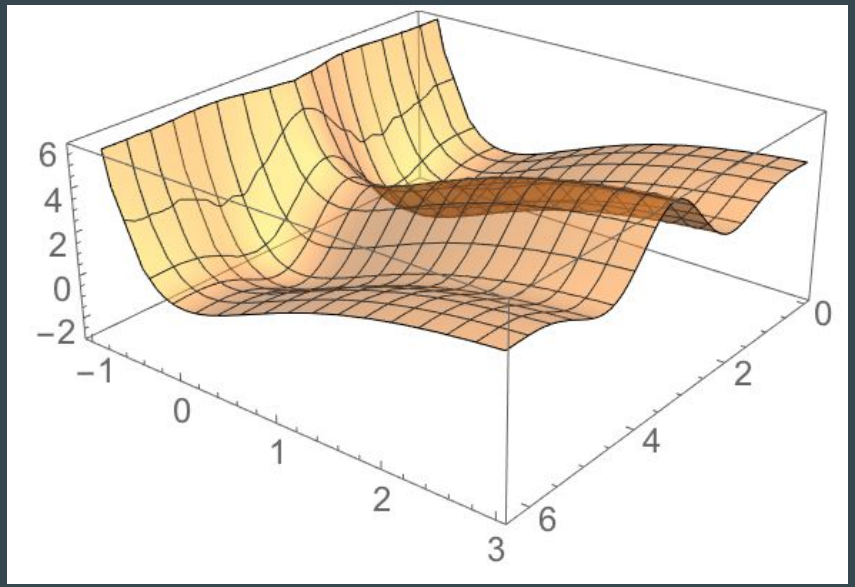


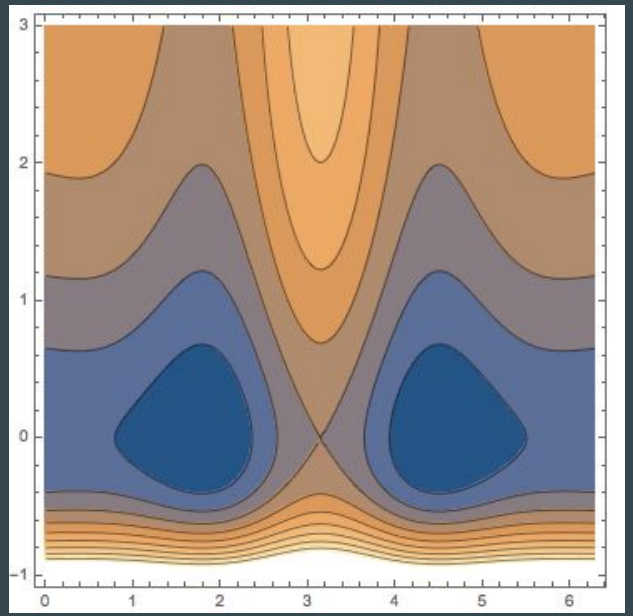
Analysis of fit quality: individual points



Generalizing to higher dimensions

Let's look at a 2D space similar in form to a space where we can move an in-plane negative ion in a circle around a water molecule in one dimension and change the size of the radius of that circle in the other dimension:

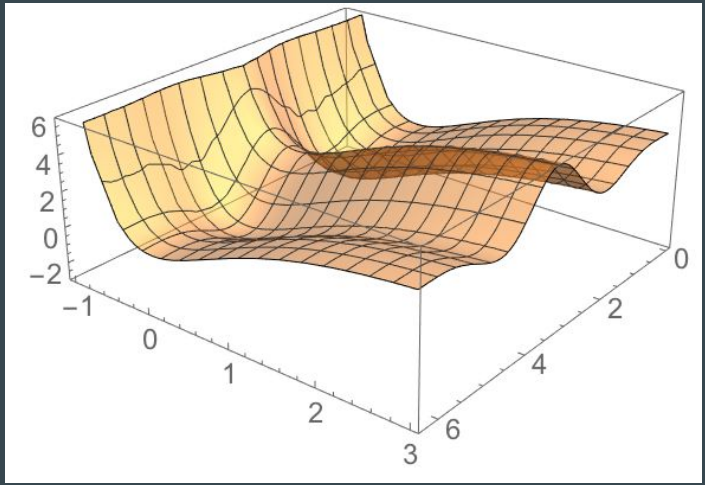


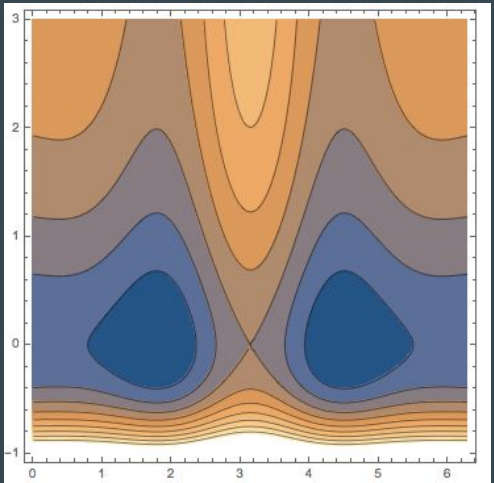


Generalizing to higher dimensions

After writing a function that takes two points in configuration space and constructs an N dimensional adaptive grid along that line or hyperline, exploring the space is just a matter of scanning through different pathways in configuration space.

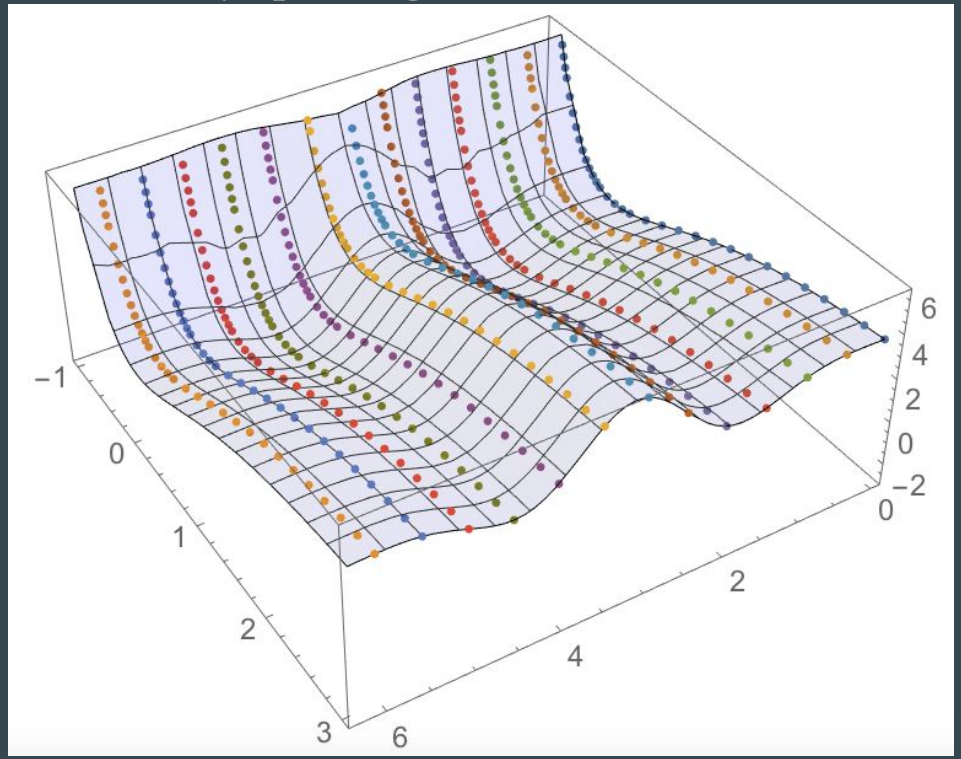
For example, let's say that I want to run the algorithm over the radius function from -.6 to 3 Angstroms for some evenly spaced grid in the rotational dimension...





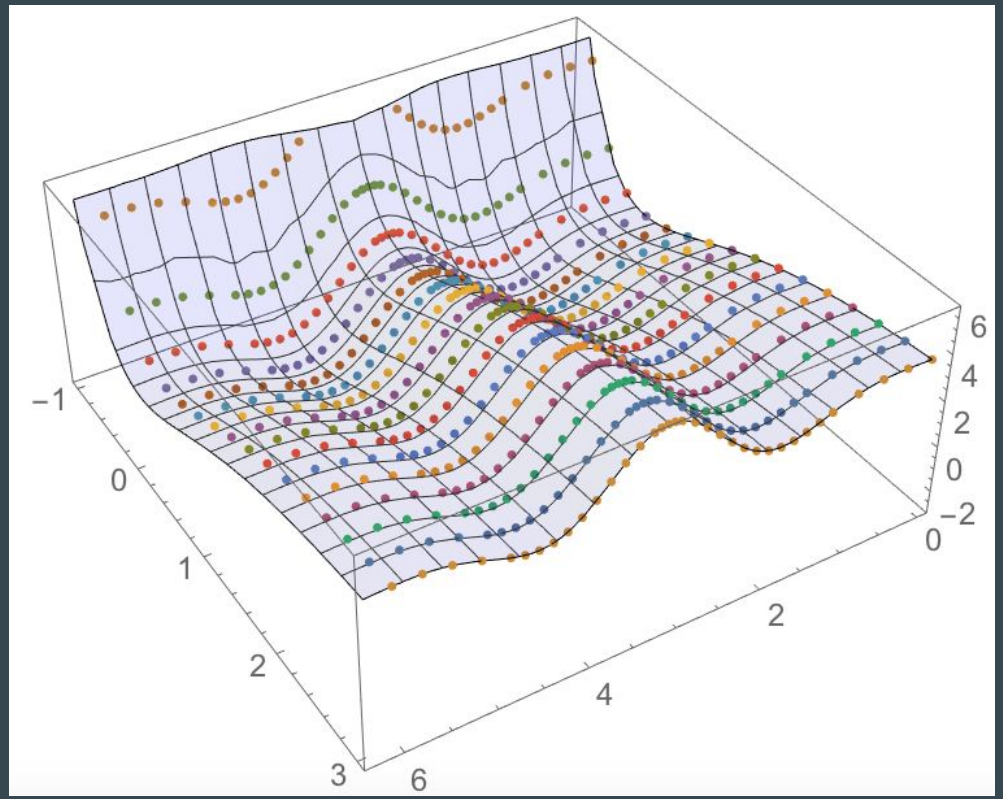
Generalizing to higher dimensions

For example, let's say that I want to run the algorithm over the radius function from -.6 to 3 Angstroms for some evenly spaced grid in the rotational dimension...



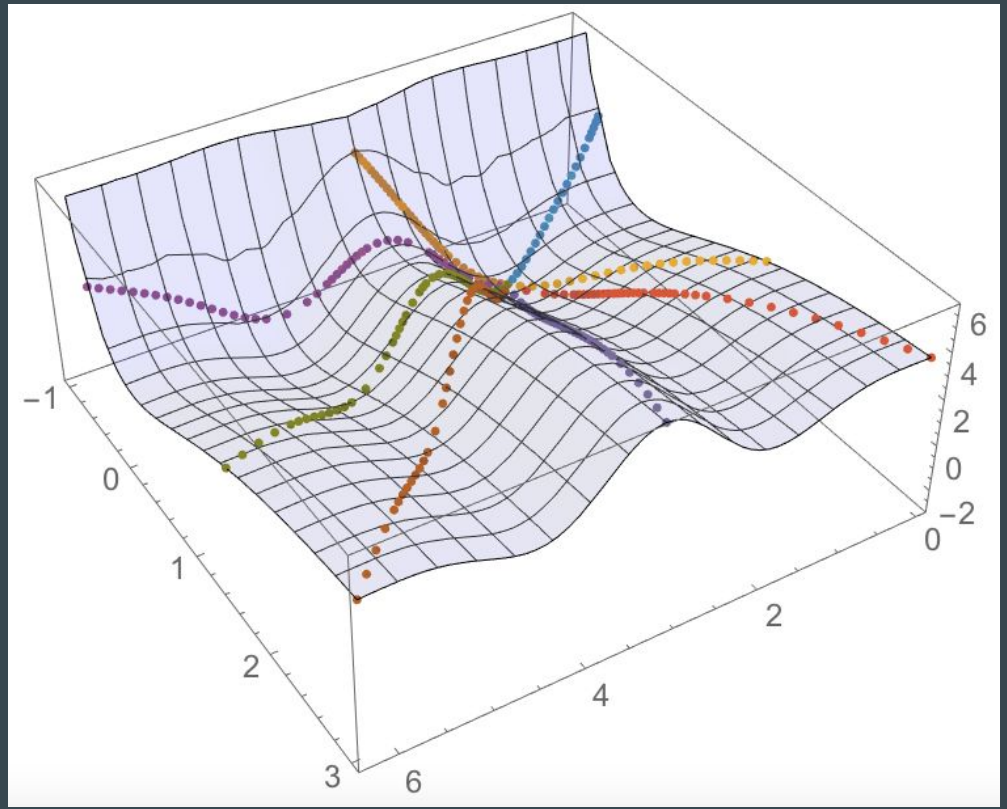
Generalizing to higher dimensions

Next, I could do the opposite approach:



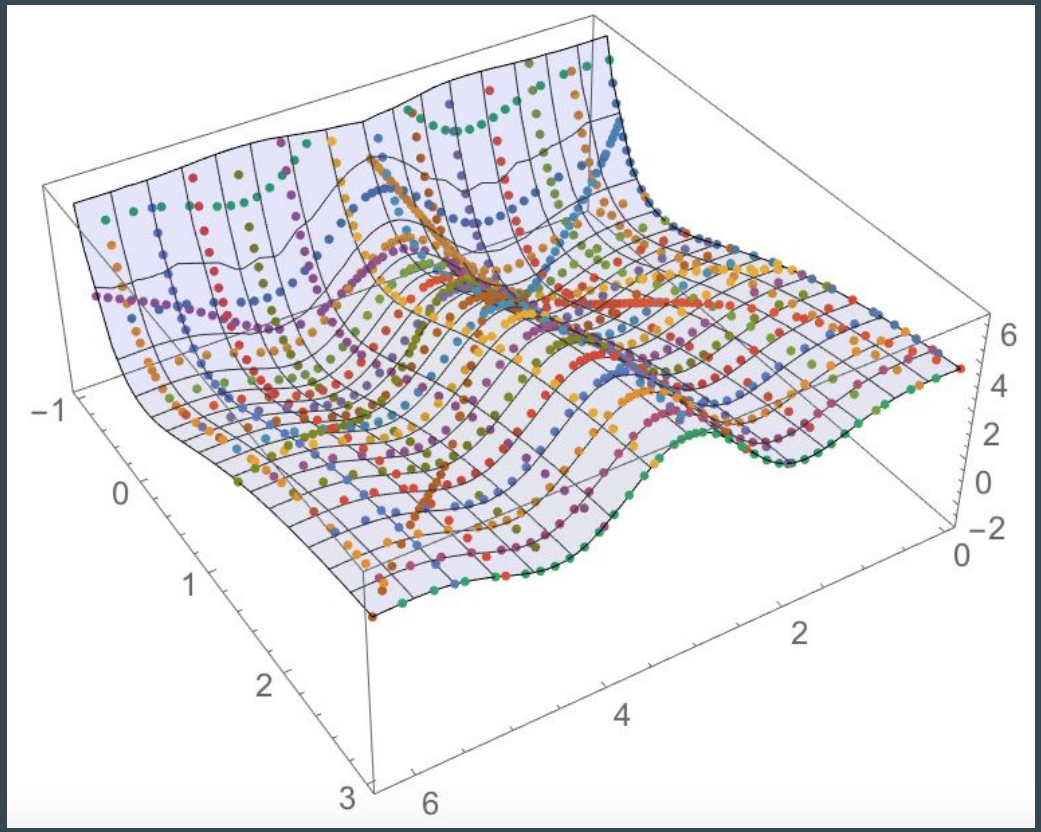
Generalizing to higher dimensions

I could even run lines emanating from some known important local minimum:



Generalizing to higher dimensions

In the end, combining the points from all approaches used:



Adaptive grid generation

I have the whole algorithm prototyped in Mathematica, hopefully we can apply it toward automating the generation of training sets in the very near future!