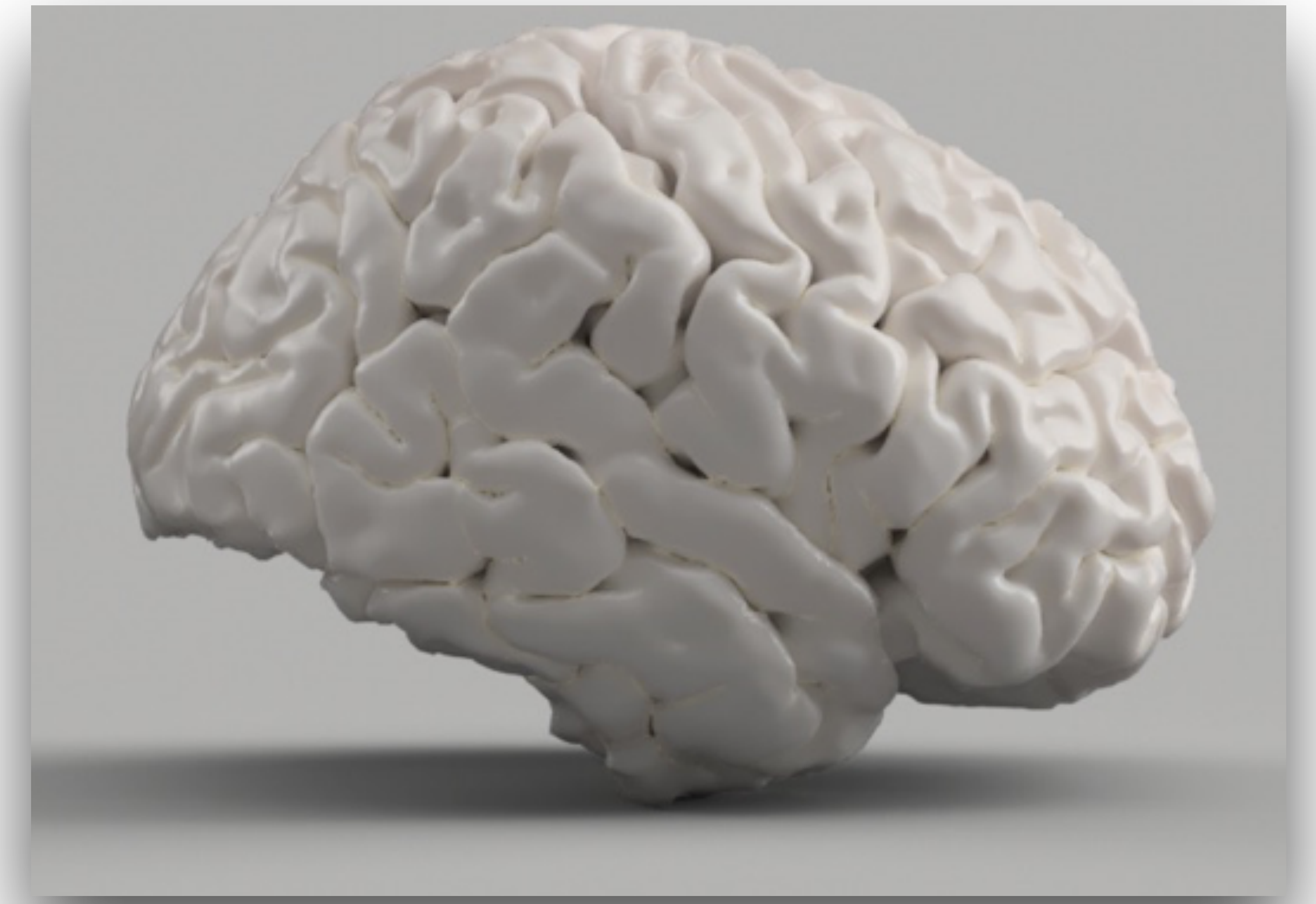
COGS109: Lecture 13



Optimization, Nelder-Mead Simplex, gradient descent, conjugate gradient

July 27, 2023

Modeling and Data Analysis

Summer Session 1, 2023

C. Alex Simpkins Jr., Ph.D.

RDPRobotics LLC | Dept. of CogSci, UCSD

- New cape replacements logistics

Plan for today

- Announcements
- Review of last time
- Project checkpoint 2 EDA
- Optimization
- Gradient descent
- Conjugate gradient
- Machine learning

Announcements

- New cape replacements logistics
- Assignments remaining A2, A3, D5, D6, D7, Q3, Q4, project

Checkpoint 2: EDA

What can we do with this idea of error?

- We now can quantify differences between model and reality
- Gives us a criterion for choosing and creating models
- What do I mean by this?
  - **Let me pose the question - How can we fit a model which is nonlinear in the parameters?**
    - Least squares won't work!
    - Could linearize for the parameters…but what about cases where that is too difficult?

Optimization for regression problems which are nonlinear in the parameters

- **Optimization -** the study of problems where the goal is to minimize or maximize a function by strategically choosing values for a set of variables
  - **This is typically an iterative process, though in many cases one can solve for the optimal point of the function**
- Convex Optimization [Boyd]
  - https://web.stanford.edu/~boyd/cvxbook/
- Numerical Optimization [Nocedal and Wright]
  - http://users.iems.northwestern.edu/~nocedal/book/num-opt.html

Optimization is a popular way to study the human brain, behavior and computation

- There is a tremendous amount of interest in optimization and optimality in general in fields studying human cognition and behavior, such as Cognitive Science
  - **For model fitting in general**
  - **But also because it is intuitive to understand many aspects of human behavior in terms of optimization**

How does this relate to behavior and cognition?

- One popular model group used by cognitive science relates decision processes to minimization of cost and maximization of rewards (behaviorism)
  - **“I’m hungry, I need to eat” ->this hunger instinct and the dislike of discomfort leads us to make choices to minimize hunger, unless another cost/reward outweighs that choice**
  - **You drive on the correct side of the road because you don’t want to have a head on collision with another car, or get a ticket because either of those would be a cost**
- Motor control (control of movement)
  - **Many aspects of human sensorimotor system are optimal in some sense (specifics vary, but examples are energy expenditure/ recovery, time to goal, obstacle avoidance)**

Additional practical applications

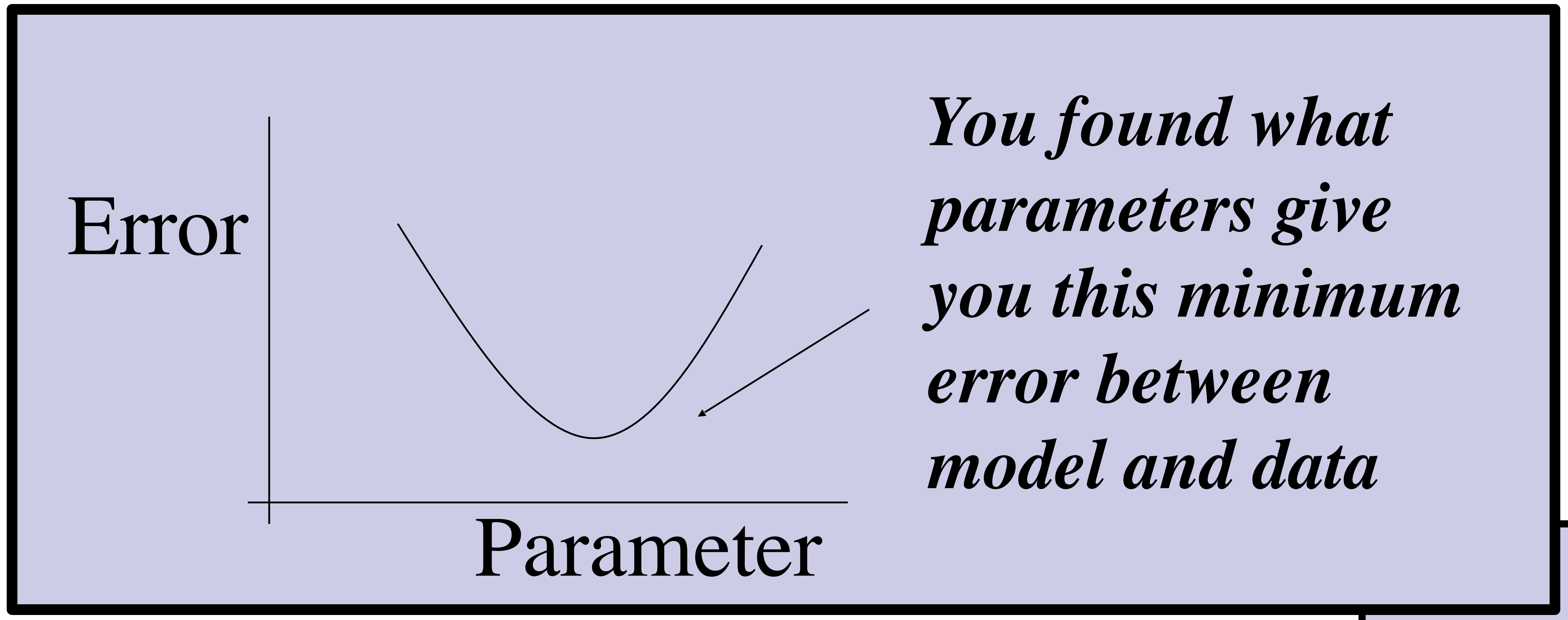
- Optimal control in human movement,
- Optimization of energy usage in society,
- Optimizing storage in hard drives to make information faster to access,
- Optimization in education to improve human learning,
- Optimization in sports to improve performance (power lifting, running, swimming, jumping, throwing, etc)
- Optimization in design to create objects that have reduced wind resistance, are stronger, lighter, less expensive, use cheaper or less impactful materials
- Optimization in network traffic to make cell phones work
- Optimization for traffic flow in vehicles

You have already performed some optimization in this class

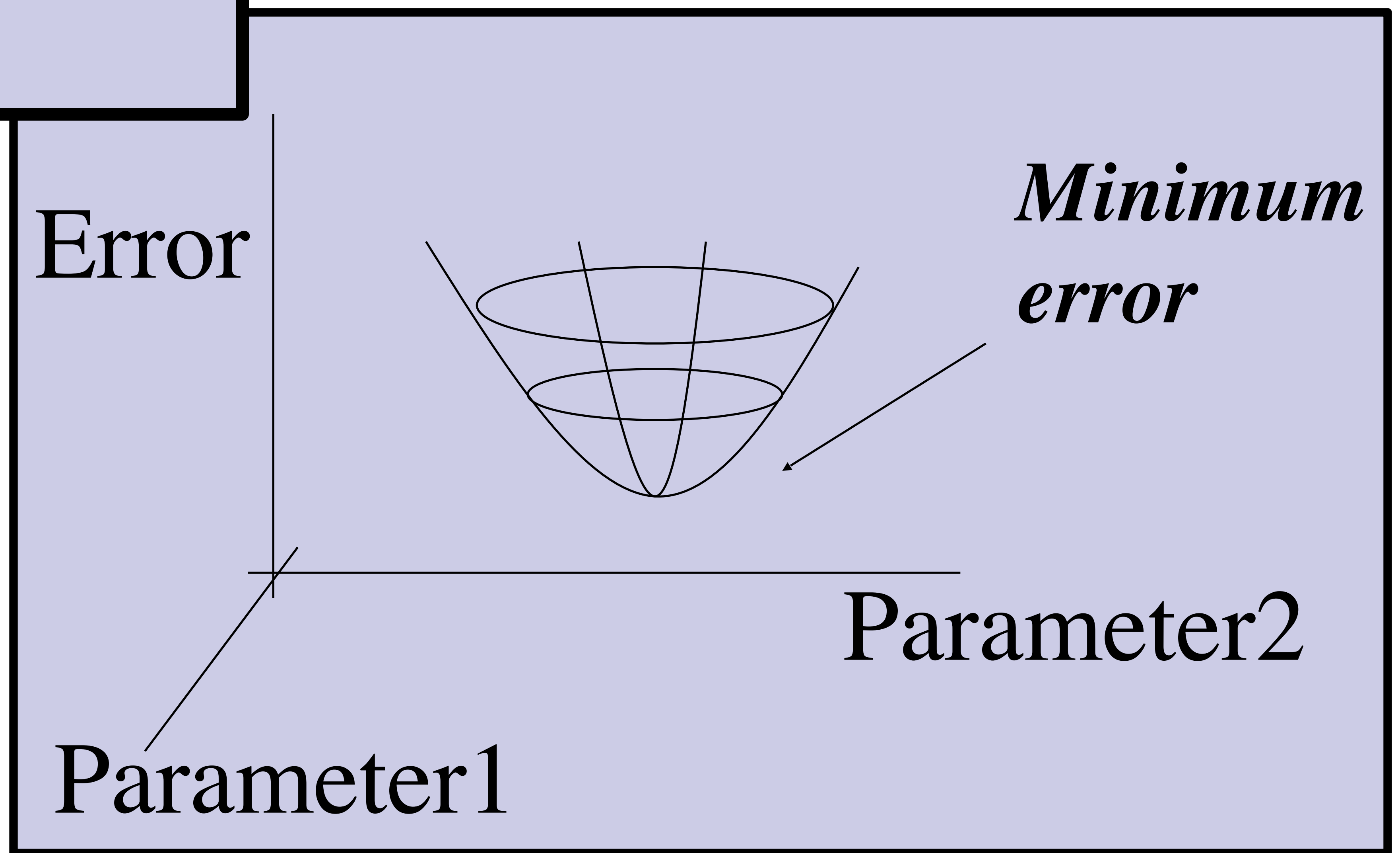
- Least squares
  - **However in that case you could compute the optimal point (which is the minimum of some error function)**
  - **In that case the cost function was a quadratic function (shaped like x^2), but it isn't always**
    - Sometimes there are many minima (we call those local minima)
    - It may be difficult to compute all the minima, or any for that matter

Graphical view of function minimum

One parameter



Two parameters



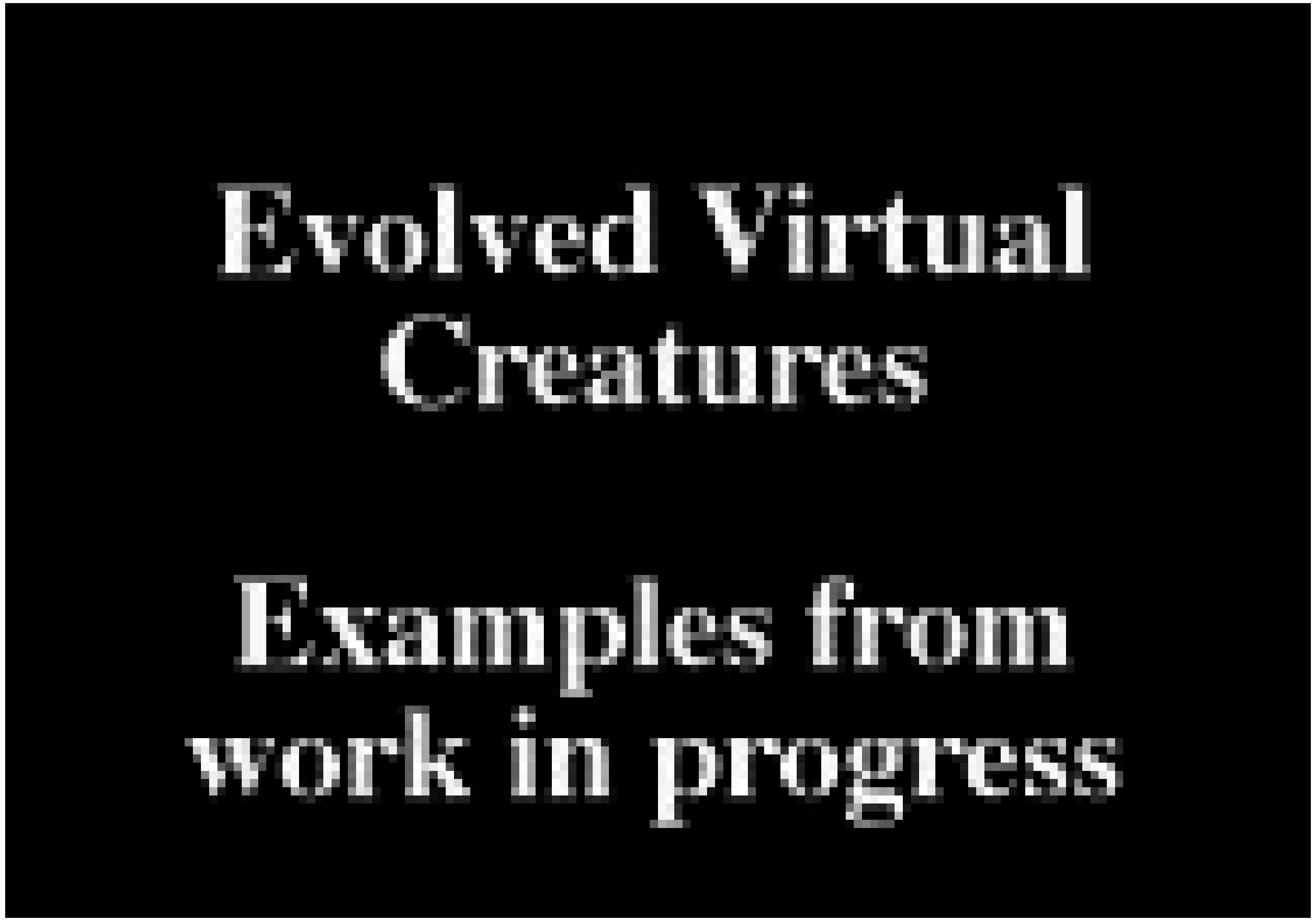
Today we'll discuss approximate solutions

- Works when you CAN'T easily solve the equations exactly (which is VERY frequent in nonlinear systems such as the brain, behavior, motor control, speech processing/synthesis/comprehension, perception, and more cognitively relevant topics)

Remind me again, what exactly are we ‘minimizing’ or ‘maximizing?’

- Minimize cost
- Maximize reward
- We decide what that function is (‘cost function’ or ‘reward function’)
  - **Then have some unknown constants**
  - **Then we use these methods to find the constants**
  - **Those constants give us the smallest cost or largest reward function**
    - Can be then interpreted as the ‘best fit’ given a definition of what ‘goodness’ is

Graphical example - evolving organisms optimize cost, maximize rewards



http://www.karlsims.com/evolved-virtual-creatures.html

What’s one way to do this?

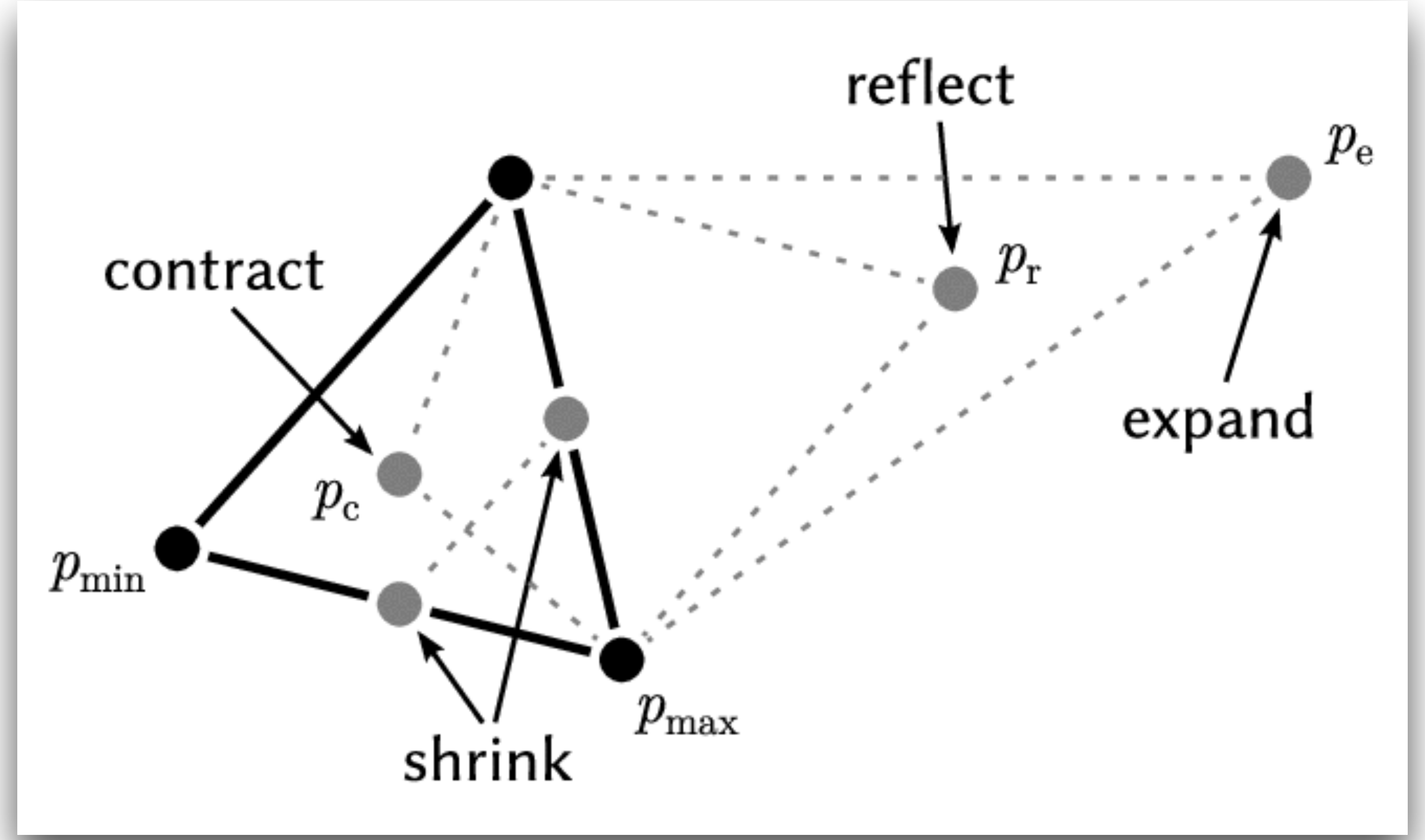
- Start with our simple question - how do we fit a model which is nonlinear in the parameters?

$$y = ax + e^{bx}$$

- We can use optimization methods to intelligently minimize the error between model and data

Nelder-mead simplex method

- Built into python's scipy, and matlab's optimization toolbox
- Simple to implement
- How does it work?
    - **https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html#scipy.optimize.minimize**
    - **Lagarias, J.C., J. A. Reeds, M. H. Wright, and P. E. Wright, "Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions," SIAM Journal of Optimization, Vol. 9 Number 1, pp. 112-147, 1998.**



https://upload.wikimedia.org/wikipedia/commons/7/72/An-iteration-of-the-Nelder-Mead-method-over-two-dimensional-space-showing-point-p-min.png

So the Nelder-Mead Simplex method

- It's the built-in nonlinear function minimization routine in Matlab, built-into **scipy** in ***scipy.optimize.minimize()***
- **fminsearch**() in matlab
- One of the most widely used methods of unconstrained nonlinear optimization
- Published in 1965
    - J. A. Nelder and R. Mead, A simplex method for function minimization, Computer Journal 7 (1965), 308–313.
    - See linked page on website for (short) collection of NM papers

What does NM do?

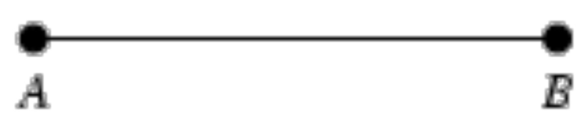
- Uses a simplex (a polytope in N+1 vertices in N dimensions)
  - **A line segment on a line**
  - **A triangle on a plane**
  - **A tetrahedron in 3d space, etc**

- Finds an approximate locally optimal solution to a problem with N variables (if the objective function varies smoothly)

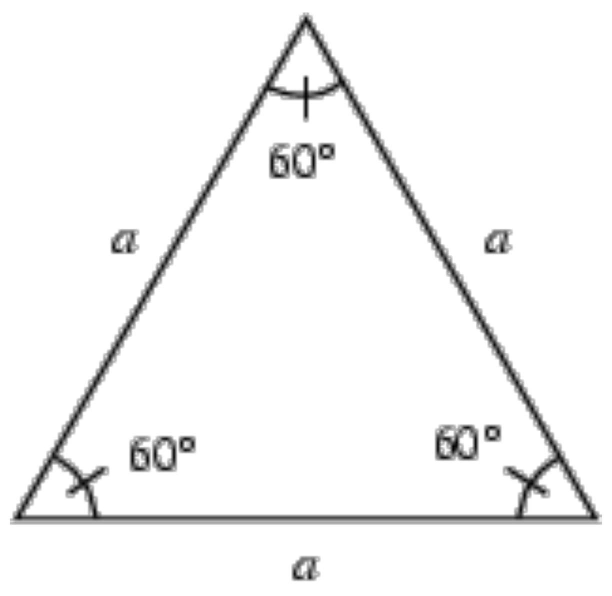
- https://www.youtube.com/watch?v=j2gcuRVbwR0

What does a simplex look like?

- Think of it as an N-Dimensional triangle
  - "simplest possible polytope (a **polytope** is a geometric object with flat sides) in any given dimension"[wikipedia]
  - For specifics, start by reading *mathworld* and *wikipedia* definitions of ***simplex*** and related important details like ***convexity*** and ***convex hulls***:
    - ***http://en.wikipedia.org/wiki/Simplex***
    - ***http://mathworld.wolfram.com/Simplex.html***



1D ->line



2D ->triangle

3D ->tetrahedron

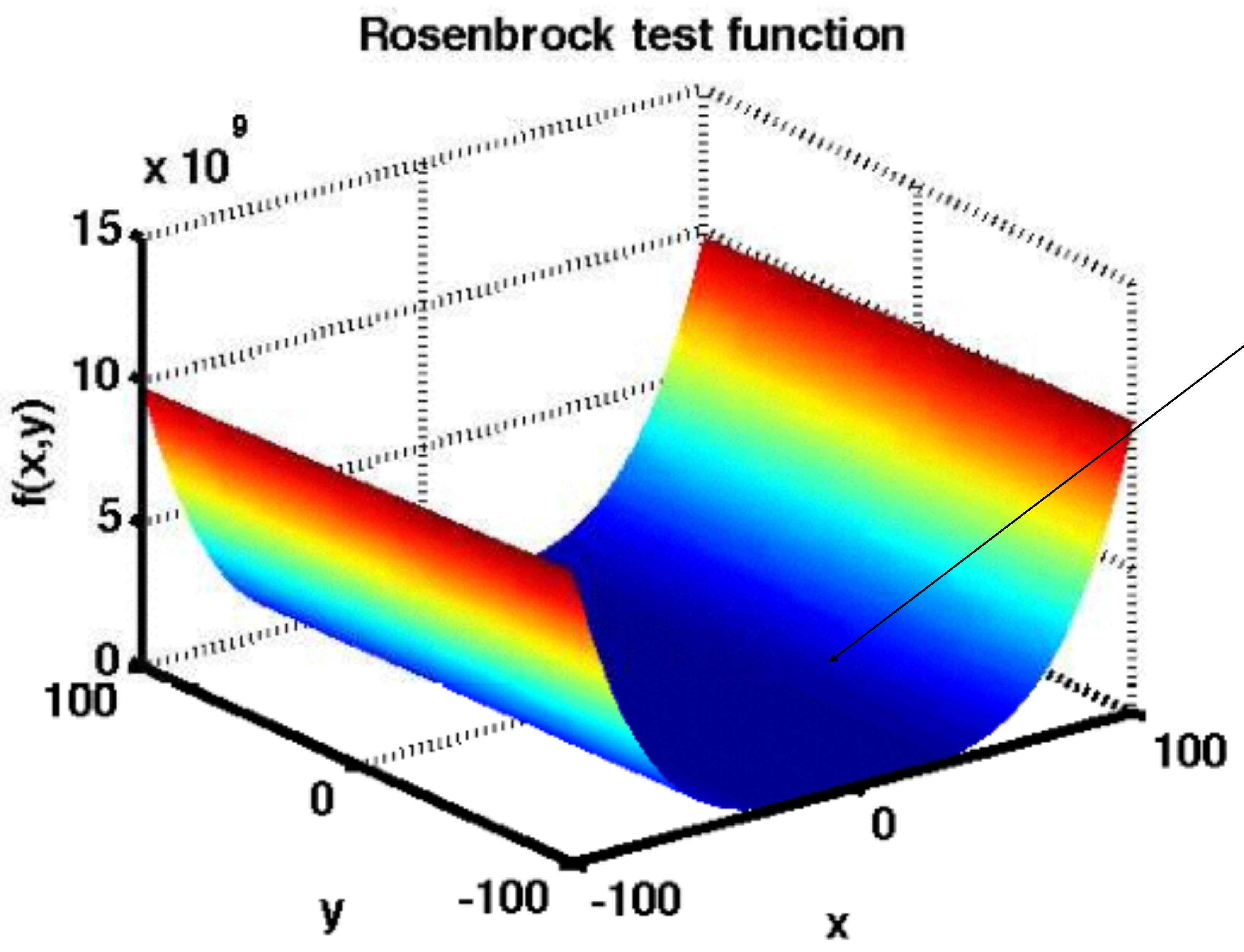
How does NM use the simplex?

- Let's see - first consider the following challenging objective function we want to minimize over the variables x and y ***(this is a typical test problem for optimization algorithms)***

$$f(x,y) = (1-x)^2 + 100(y-x^2)^2$$

Why is this challenging?

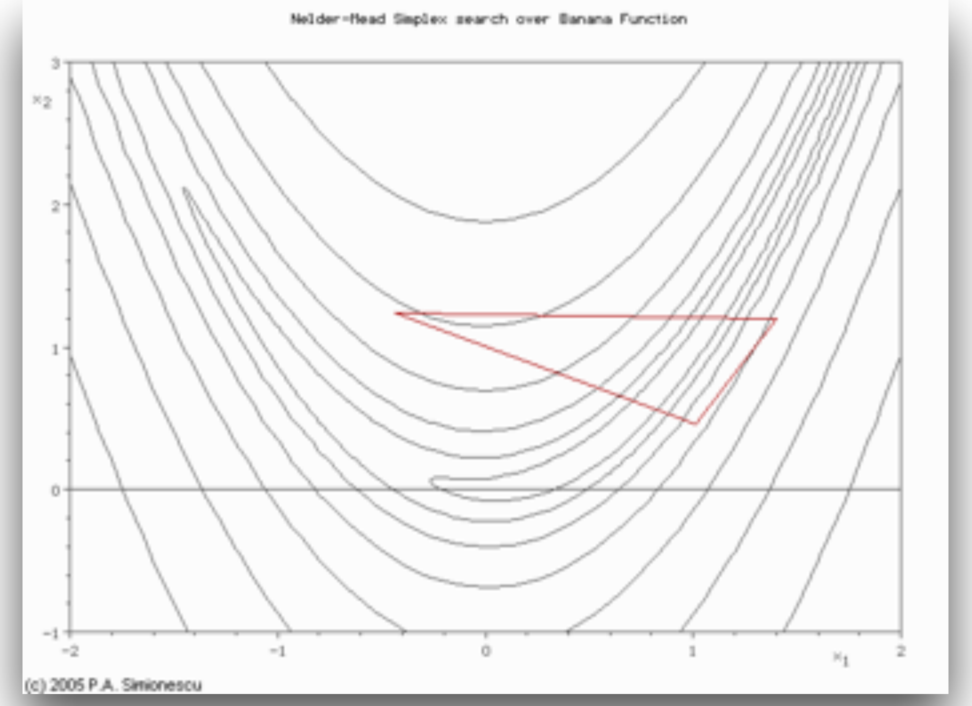
A.k.a. - Rosenbrock's valley *or* ***Rosenbrock's banana function.***

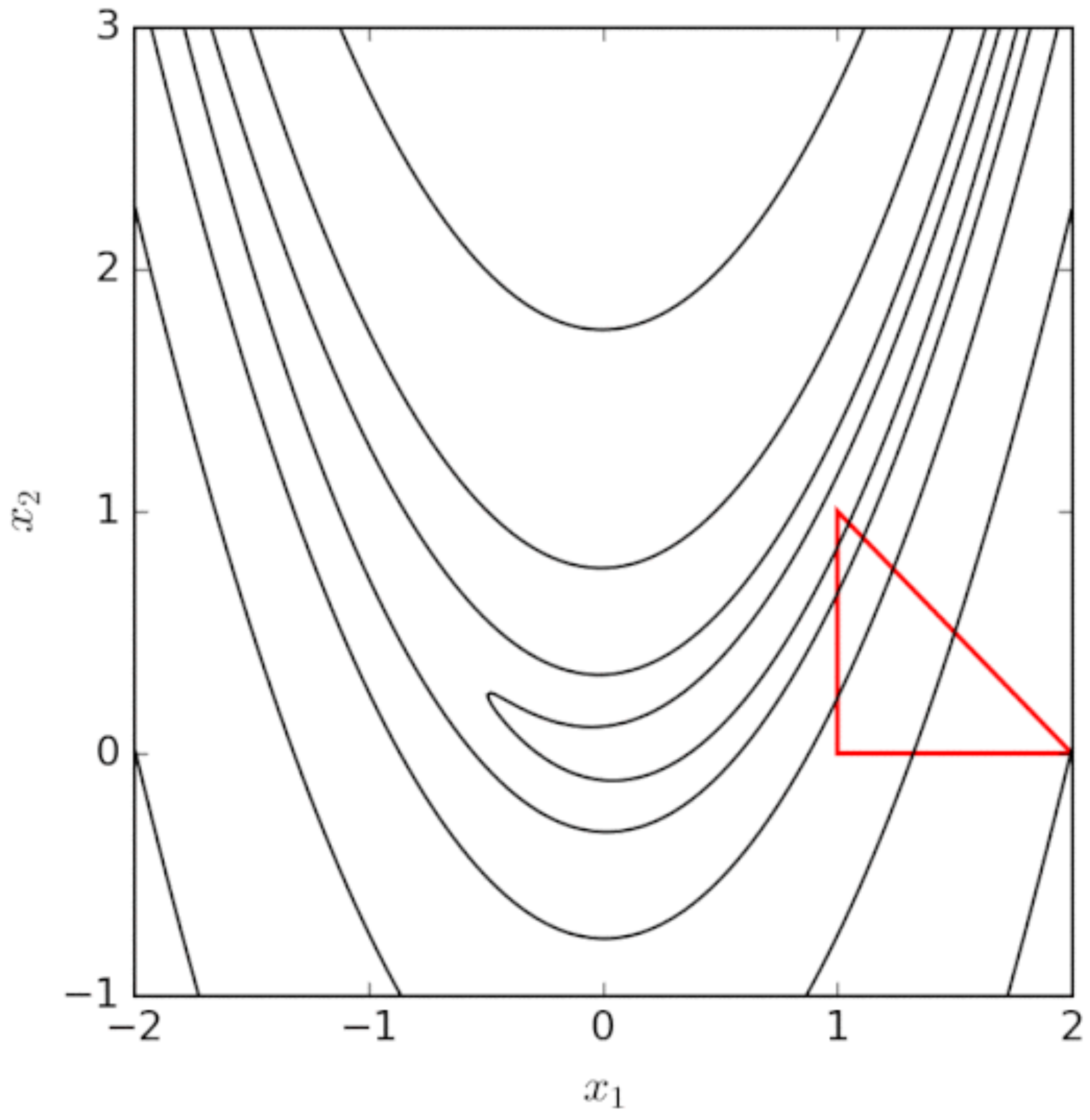


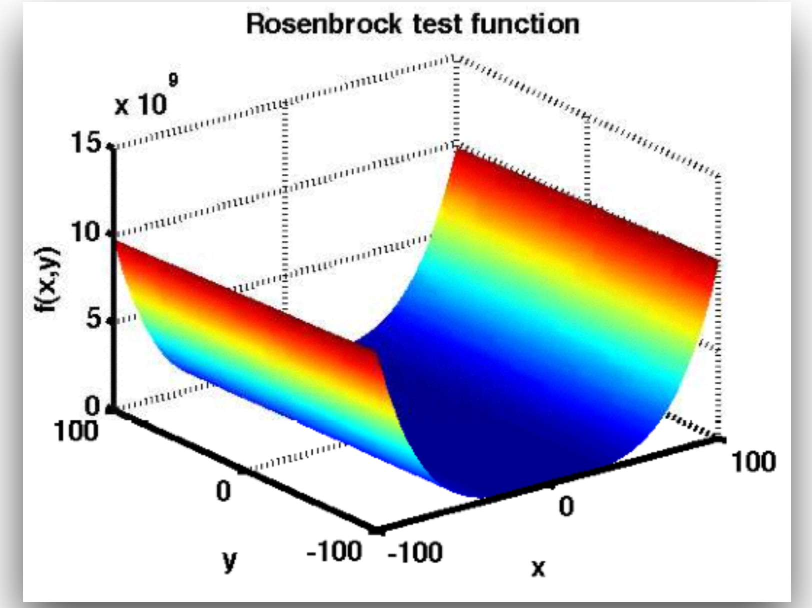
Note the long narrow valley. That makes it tough to find the global minimum with an optimization algorithm

Let’s take a look at the NM simplex algorithm in action

- The NM algorithm trying to minimize the Rosenbrock function:







https://en.wikipedia.org/wiki/File:Nelder-Mead_Rosenbrock.gif

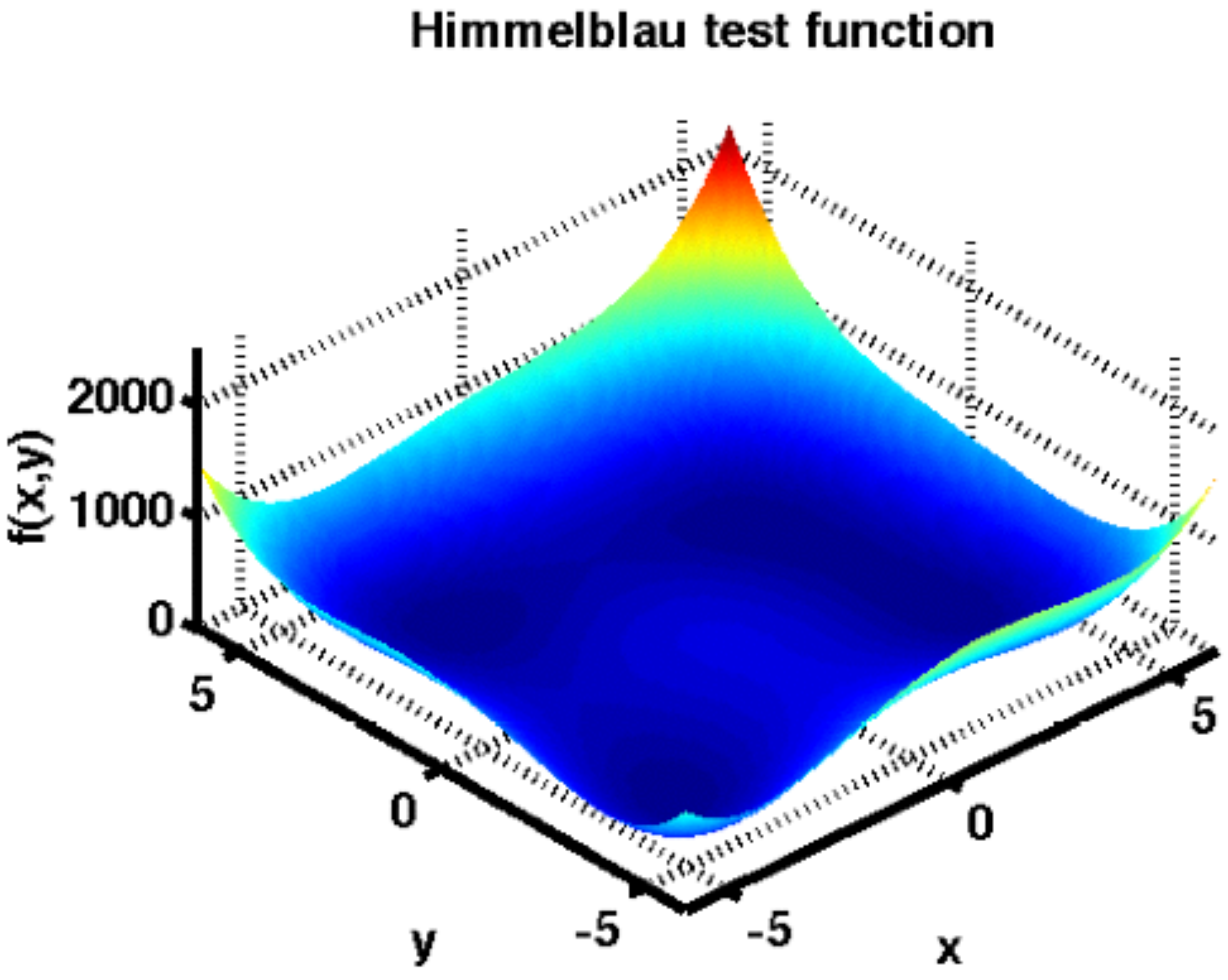
NM computes the simplex, and compares points

- If one is worse (higher) on the cost (objective) function, the simplex reflects that point about the centroid (generalized center) of the simplex and thus makes a new simplex which is hopefully better
- If the points are close in their value the simplex shrinks
- If the points are far away in their value (steep slope) the simplex expands
- See the readings for details
  - **Intro - wikipedia link**
  - **Original 1965 paper**
  - **Convergence properties paper**

Let's look at another function

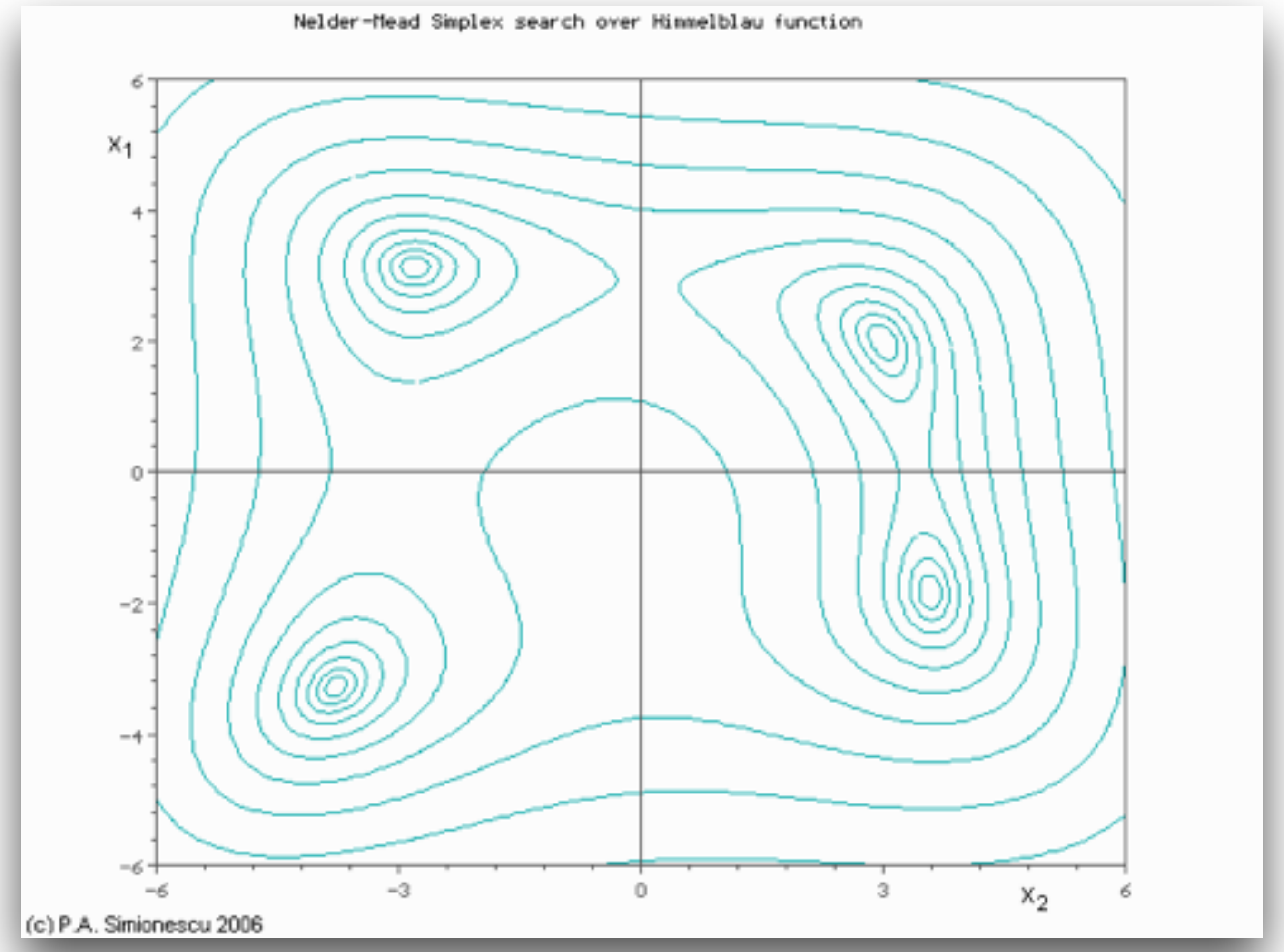
$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$

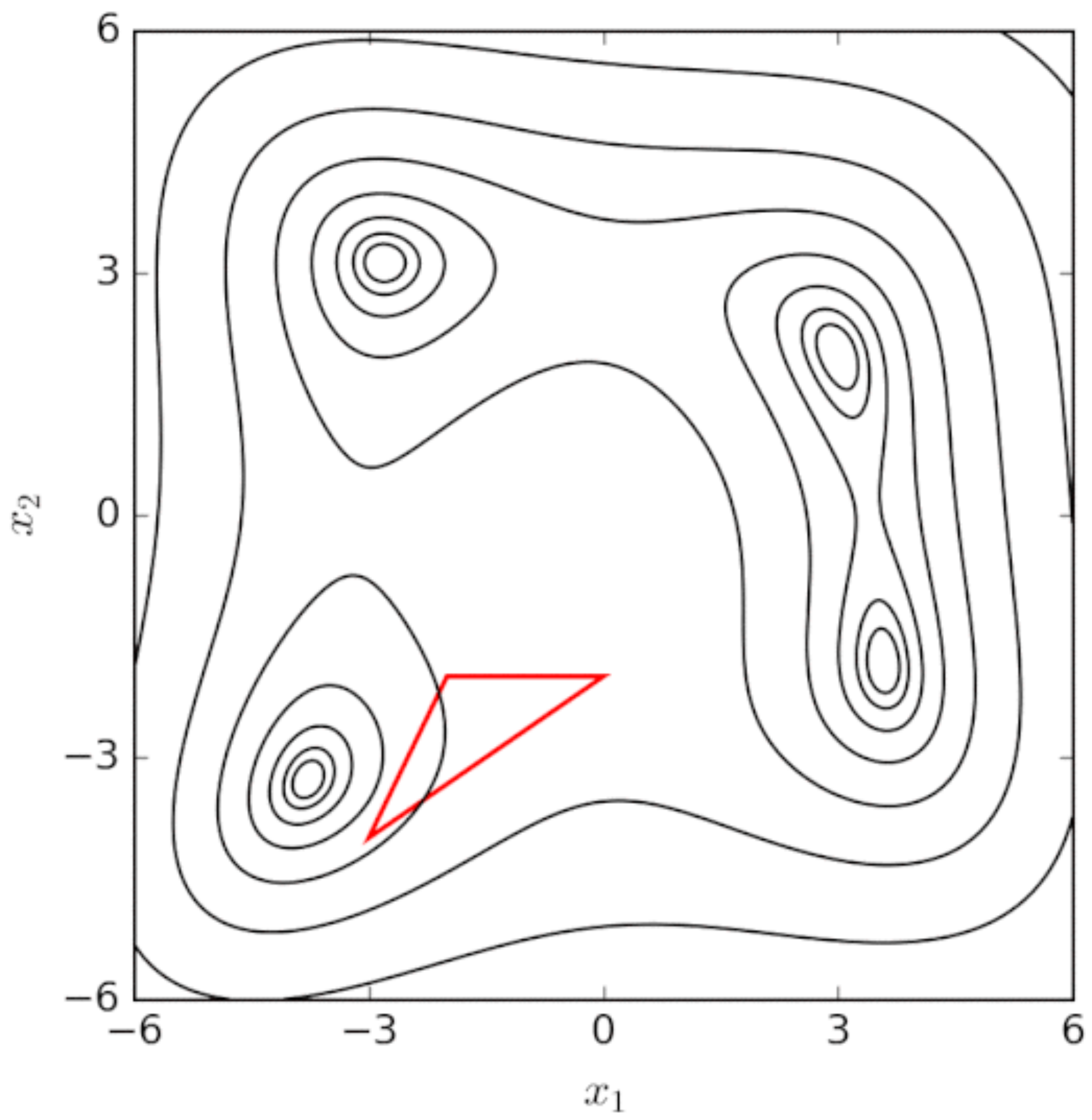
- Himmelblau's function
- Global Minimum
  - f (3,2) = 0
- Local Minima:
  - f (-3.78, -3.28) = 0.0054
  - f (-2.81, 3.13) = 0.0085
  - f (3.85,-1.85) = 0.0011
- In this case, multiple minima exist

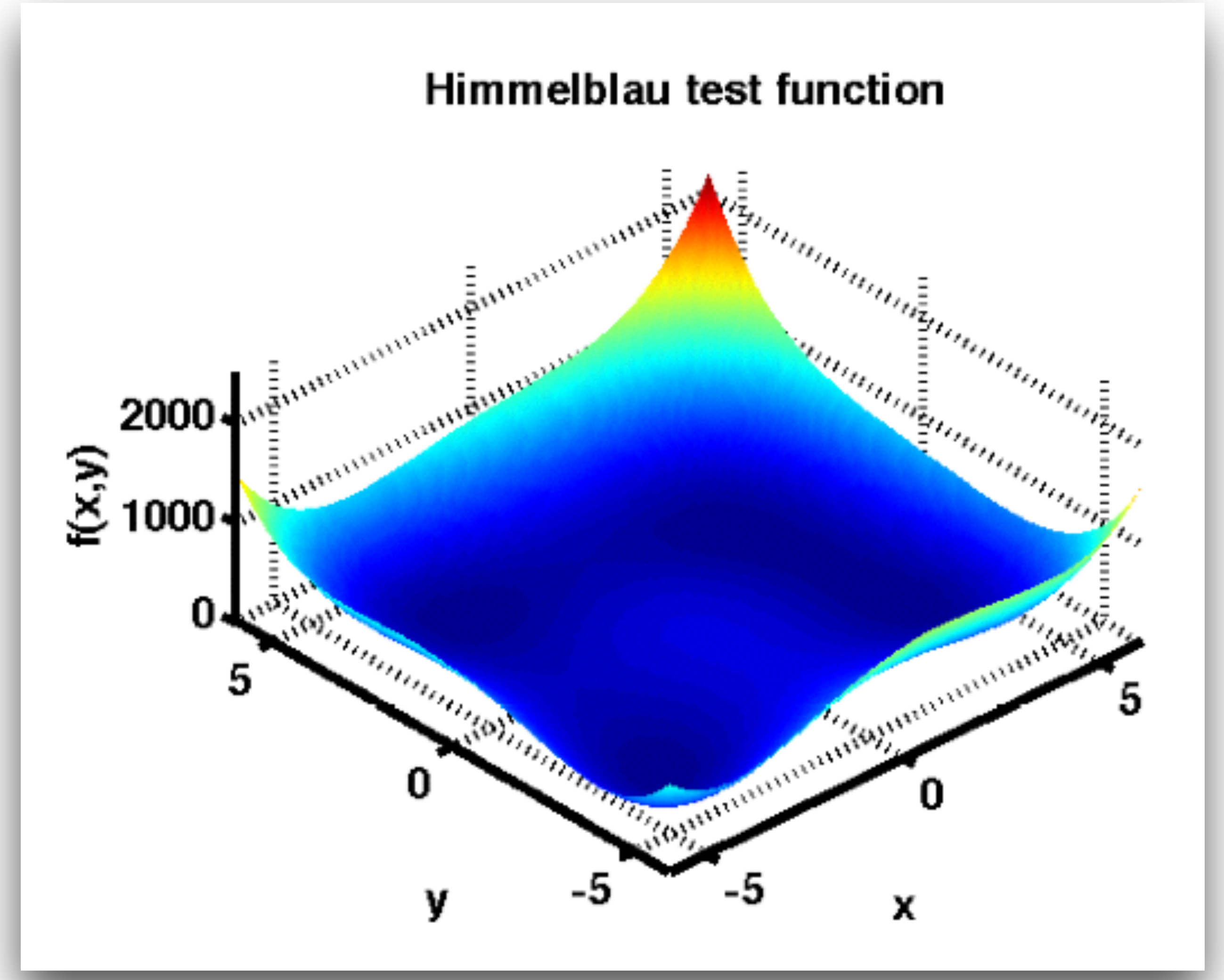


How does NM approach this?

- NM finding local minimum of Himmelblau function





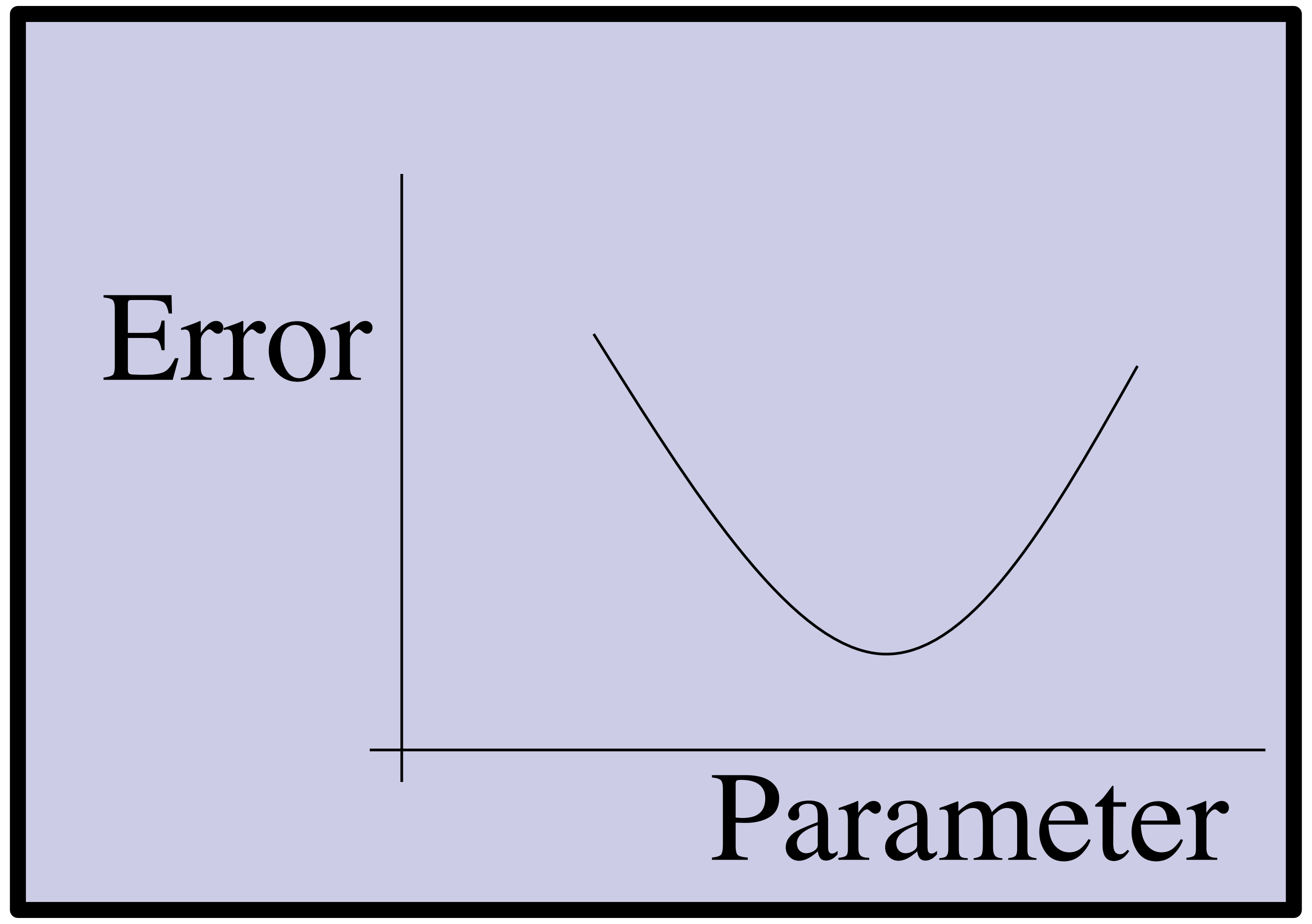


A few quick notes that are important

Convexity and convex problems

- Convex functions have one global minimum and no additional local minima
  - They can still be hard to minimize though - like for the Rosenbrock function
  - There exist many techniques which rapidly converge to the solution of convex functions

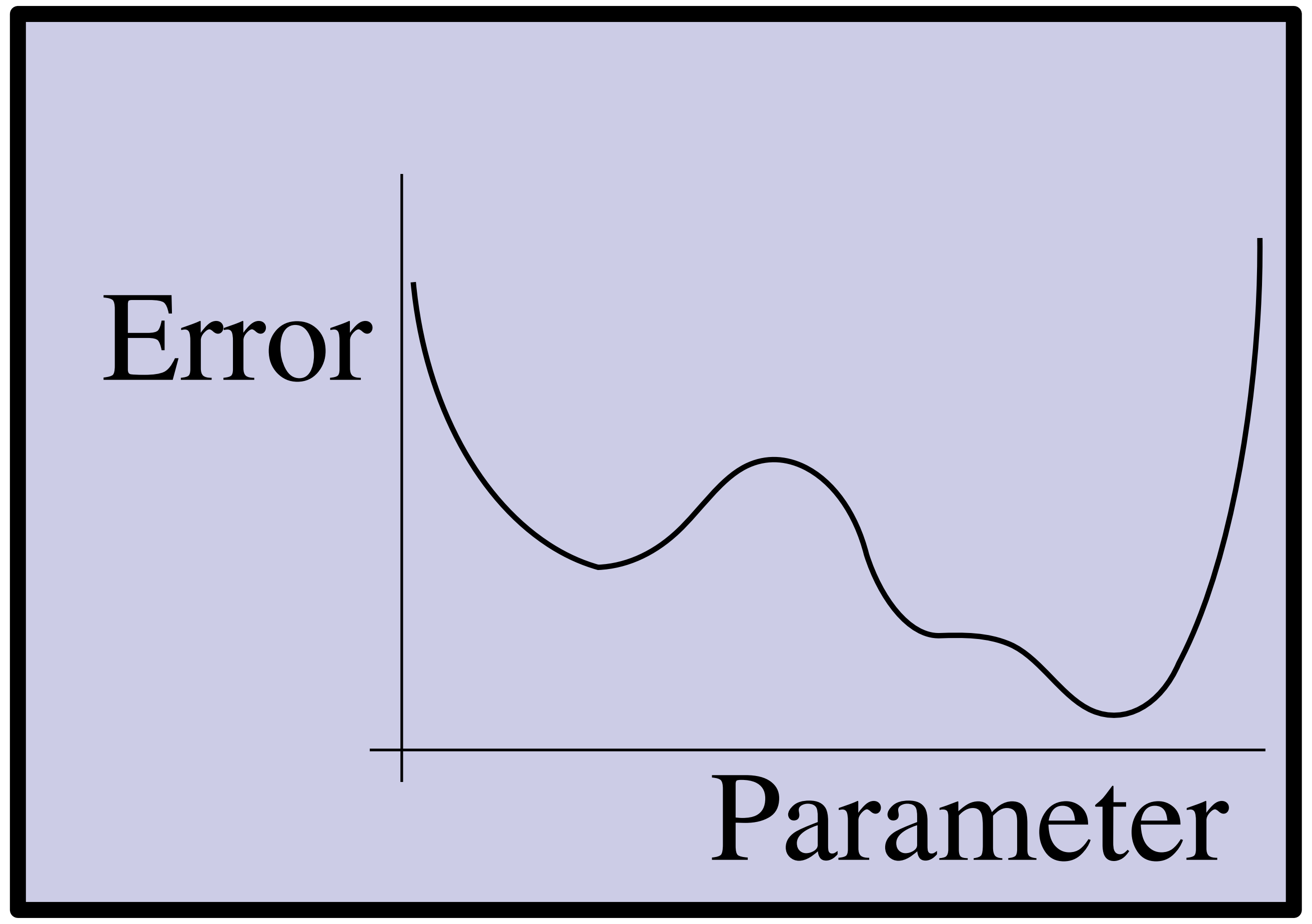
Convex



Non-convexity and local minima

- **Non-convex** functions may have multiple ***local minima*** which are not anywhere near the ***global minimum***
  - **For example, the Himmelblau function**
  - **What can we do?**
    - Many strategies - it's hard to know what is the absolute global minimum when you can't explicitly compute it
      - **Can restart with multiple different initial conditions and see if you get the same minima**
      - **Global optimization is a whole branch of mathematics where one attempts to find deterministic algorithms guaranteed to converge to globally optimal solutions in finite time**
- Take home message - use any algorithm with caution and awareness

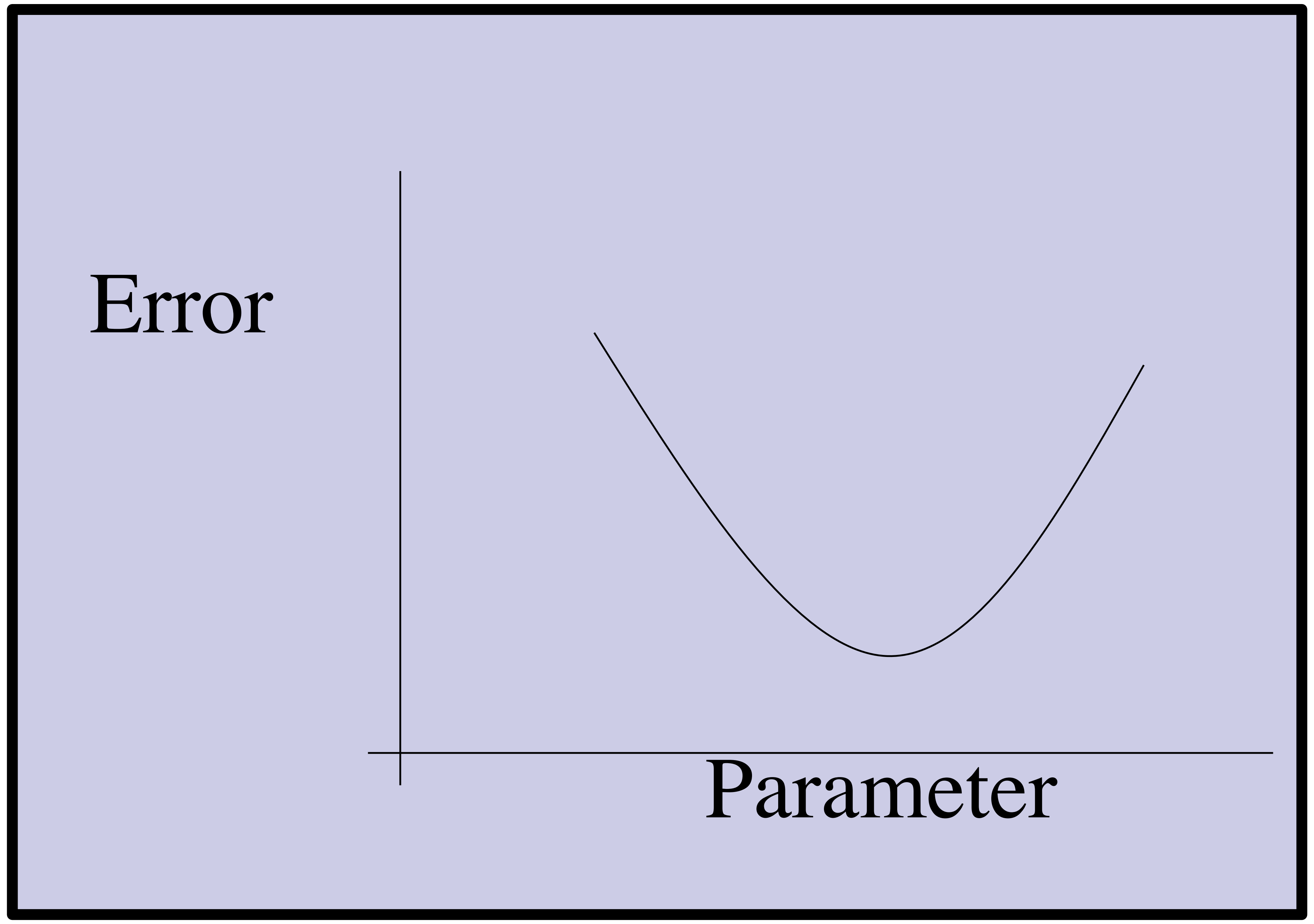
Non convex



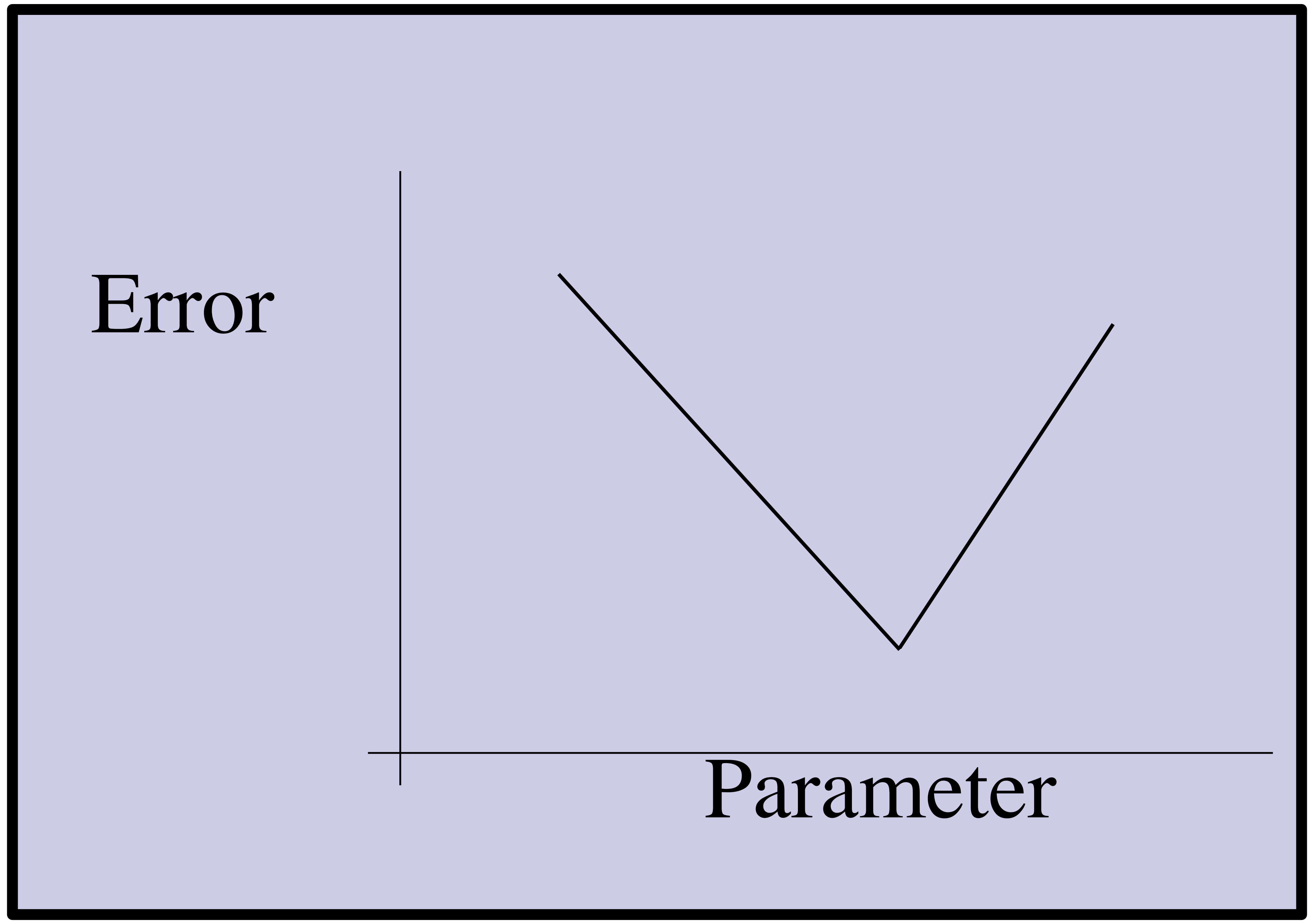
Smooth vs. Non-smooth problems

- Smooth is much easier - derivative is continuous everywhere
- Also

Smooth



Non-smooth



Constrained vs. Unconstrained

- **Constrained optimization** - Process of optimizing some function with respect to some variables subject to constraints on those variables
    - Constraints may be given that we need to satisfy may berange of values, boundary, rules such as shape of differentials, etc
    - Usually boundary - Equality or inequality constraints, such as:
        - (source: https://en.wikipedia.org/wiki/Constrained_optimization#:~:text=In%20mathematical%20optimization%2C%20constrained%20optimization,of%20constraints%20on%20those%20variables.)

A general constrained minimization problem may be written as follows:[2]

$$\begin{array}{lll} \min & f(\mathbf{x}) & \\ \text{subject to} & g_i(\mathbf{x}) = c_i & \text{for } i = 1, \ldots, n \quad \text{Equality constraints} \\ & h_j(\mathbf{x}) \geq d_j & \text{for } j = 1, \ldots, m \quad \text{Inequality constraints} \end{array}$$

where $g_i(\mathbf{x}) = c_i$ for $i = 1, \ldots, n$ and $h_j(\mathbf{x}) \geq d_j$ for $j = 1, \ldots, m$ are constraints that are required to be satisfied (these are called hard constraints), and $f(\mathbf{x})$ is the objective function that needs to be optimized subject to the constraints.

- **Unconstrained optimization** - solve the optimization function, no constraints or range imposed

Why not just compute all the minima of a function over all the space of interest?

- You might not know the function!
  - **Think if I told you to find the lowest part of campus blindfolded and with your ears and sense of smell somehow 'disabled'**
  - **You'd have to feel your way there, you couldn't predict the final lowest point, if you had no prior knowledge**

What if you know the function?

- It might be that you know the function but it's unreasonable to calculate all the minima
  - **Too computationally expensive!**
    - you'd have to compute the function at n points, and if it's an m-dimensional function (ie we have m parameters to find), m being big and n being big, you would have to compute n^m points
    - e.g. - 10D, 100pts would be 100^10=1e20 computations of the function
    - Comparison - our computers presently are on the order of 10^9 computations per second (GHz), so assuming in one cycle we can compute the function, which isn't true, but for the sake of argument, consider that even this would take 10^11 seconds
      - **This is 3.1710e+03 years!!! Oops:)**
- **There has to be a better way!!! And we can use search to do it in a few computations**

Example of application in python

- https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html#scipy.optimize.minimize

- https://machinelearningmastery.com/how-to-use-nelder-mead-optimization-in-python/#:~:text=The%20Nelder%2DMead%20optimization%20algorithm%20can%20be%20used%20in%20Python,initial%20point%20for%20the%20search.

One common theme in optimization is trying to find a minimum

- Sometimes we don't need to deal with nonlinearity, and as such can use search methods which are specifically designed/optimized for such problems
- Skiing - you want to get to the bottom of the hill as fast as possible to get the hot chocolate
  - **Obvious approach is to choose the direction of steepest descent down the mountain**
- Leads us to
  - **Gradient descent (a.k.a. the method of steepest descent)**
  - **Do exactly what we just said**

How does gradient descent work (an introduction)?

- Start with the cost function
  - **Make it (hopefully) quadratic so it has the nice bowl shape, and a definite global minimum (though complicated functions may have local minima)**
  - **We want to find a way to make**
    - Mp-k = *something* as *small as possible*
    - *So we'll start at some guess for p, then change p at each step to be going 'down the hill' of the cost function*

The algorithm

- Algorithm:

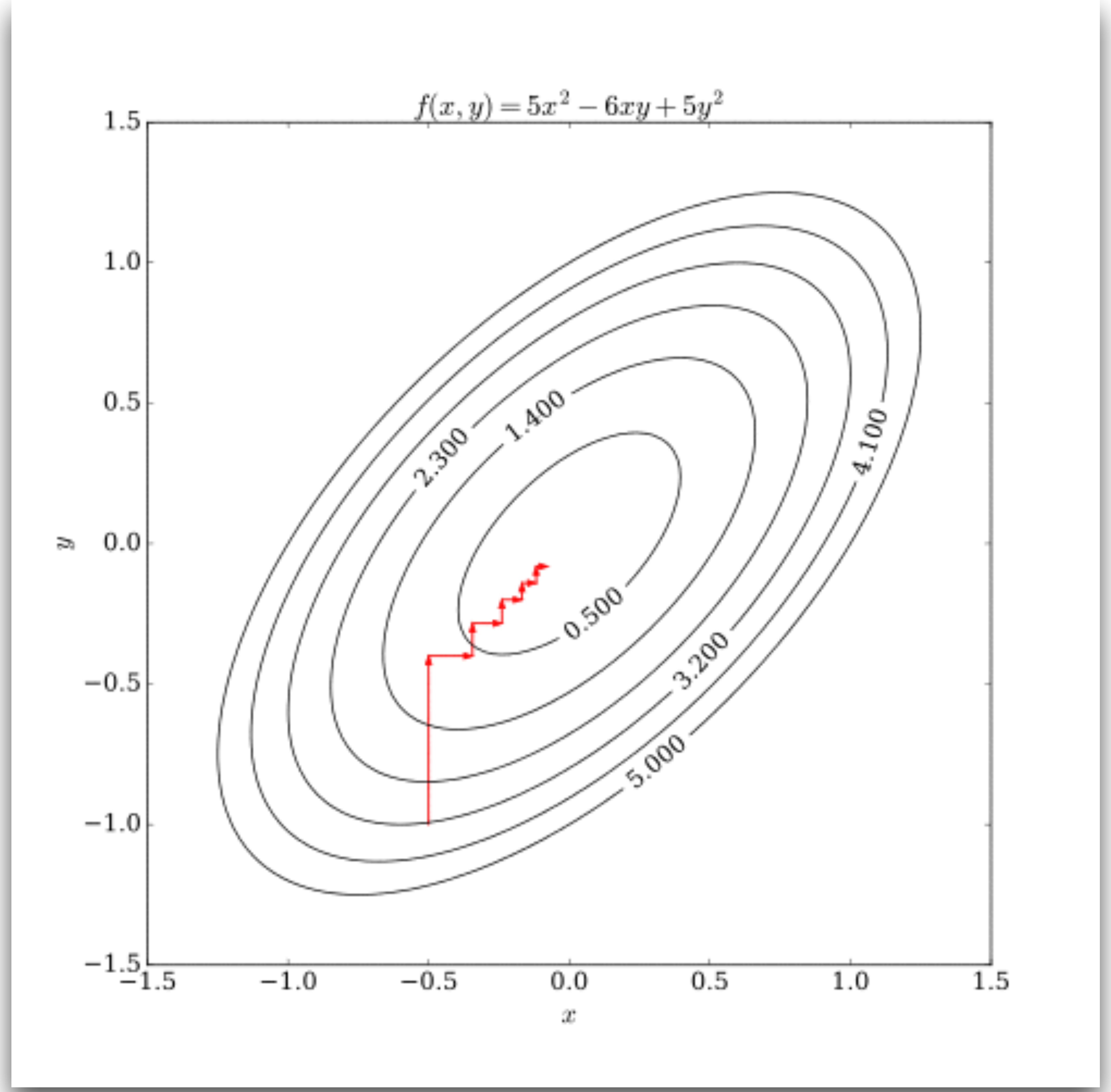
  Choose a starting point p(0)

  - **Repeat this until we're satisfied that we're close**
    - Compute the distance to change the vector p
    - Compute the direction to change the vector p
    - Update p
  - **Goto repeat**

- It turns out that the steepest direction and step distance is found by looking at the 'gradient' of the cost function

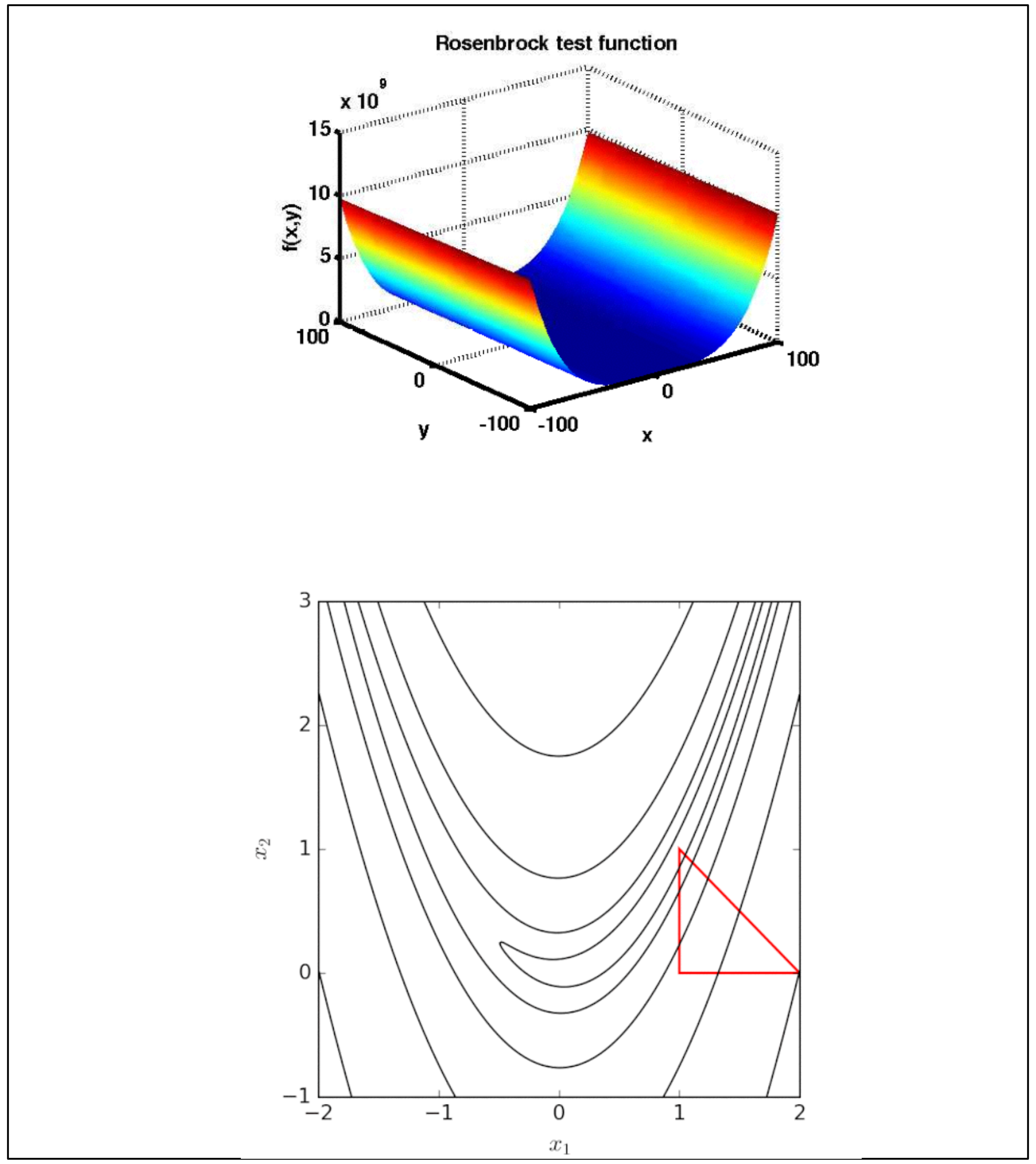
What does the resulting behavior look like?

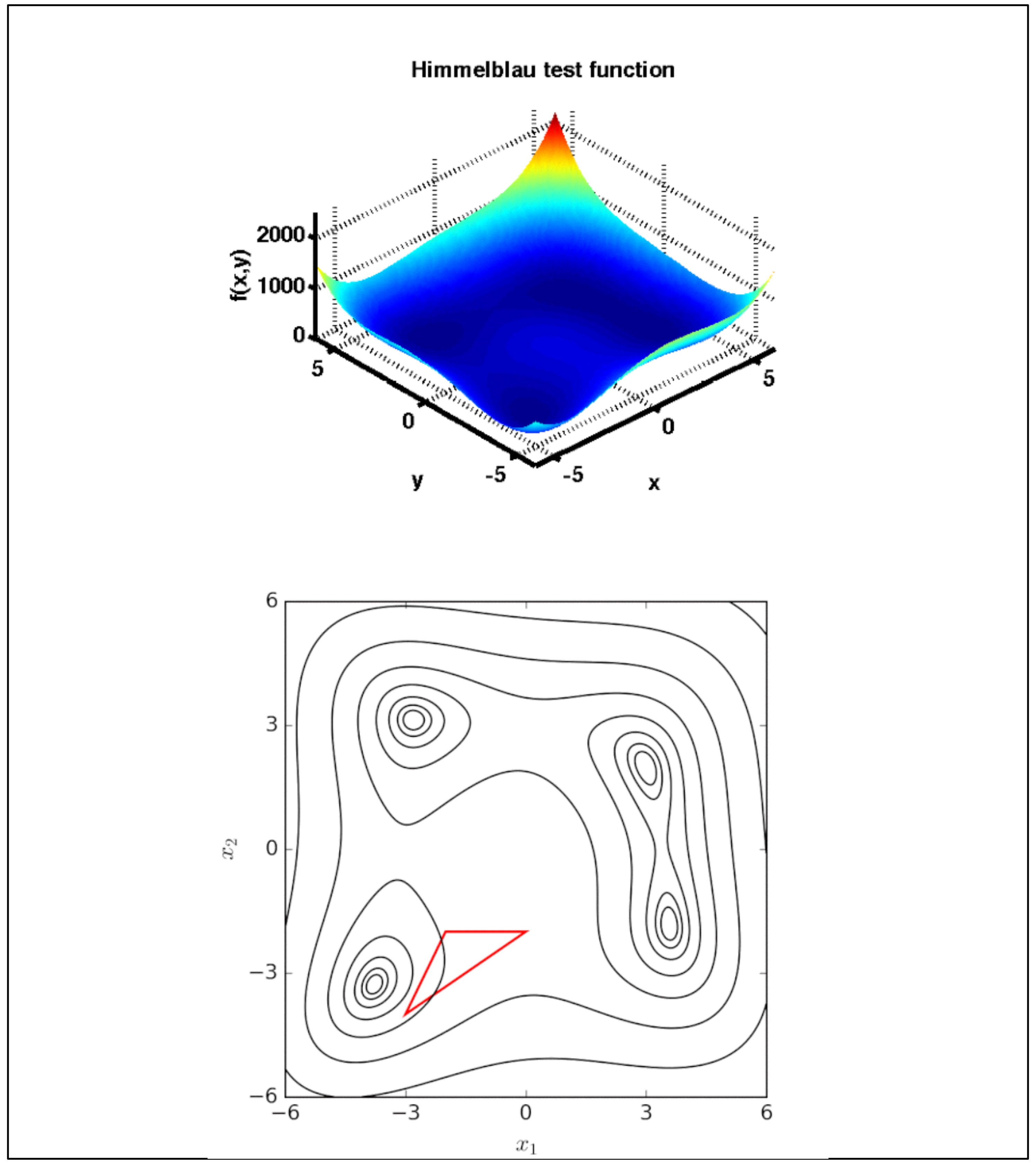
- In 2d for convex function
  - Most basic form
- In 2d for nonlinear function with multiple start points, another form
  - https://en.wikipedia.org/wiki/File:Gradient_Descent_in_2D.webm



About the Nelder-Mead Simplex algorithm

- So we showed examples of the NM algorithm and implementation details in python or matlab





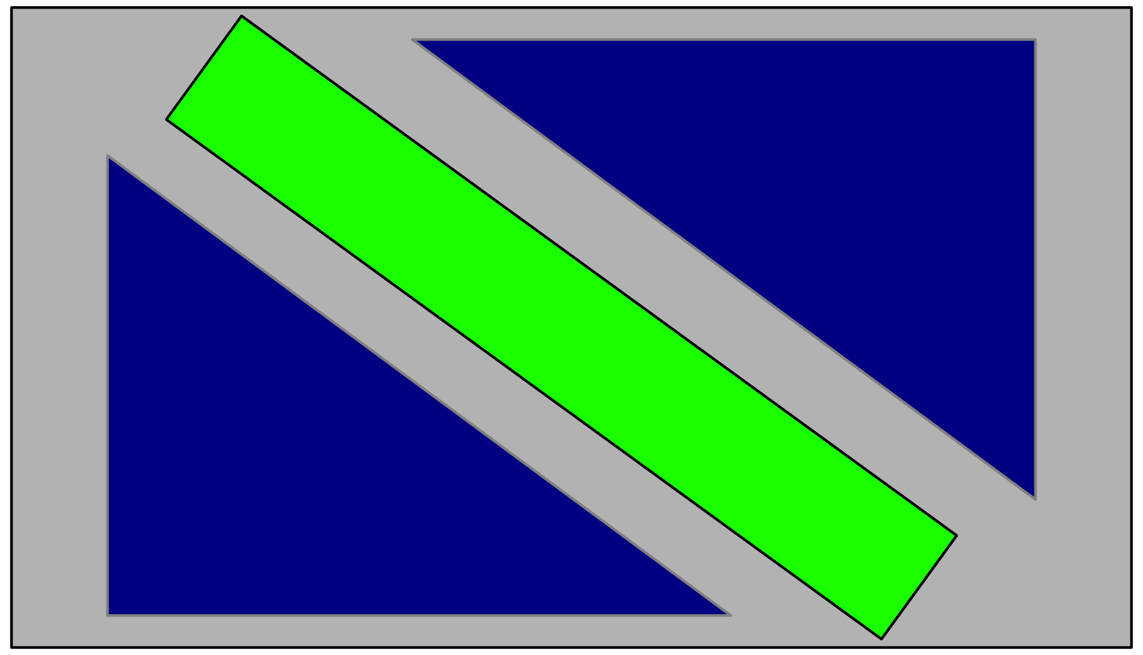
Before we go on, a few definitions

- **Positive definite matrix**
  - **All eigenvalues are positive**

$$(M_{ij} = M_{ji})$$

- **Symmetric matrix** (review) - symmetric about the diagonal

$$\begin{bmatrix} 10 & 2 & 3 & 4 \\ 2 & 4 & 9 & 7 \\ 3 & 9 & 2 & 8 \\ 4 & 7 & 8 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 10 & 2 & 3 & 4 \\ 2 & 4 & 9 & 7 \\ 3 & 9 & 2 & 8 \\ 4 & 7 & 8 & 1 \end{bmatrix}$$



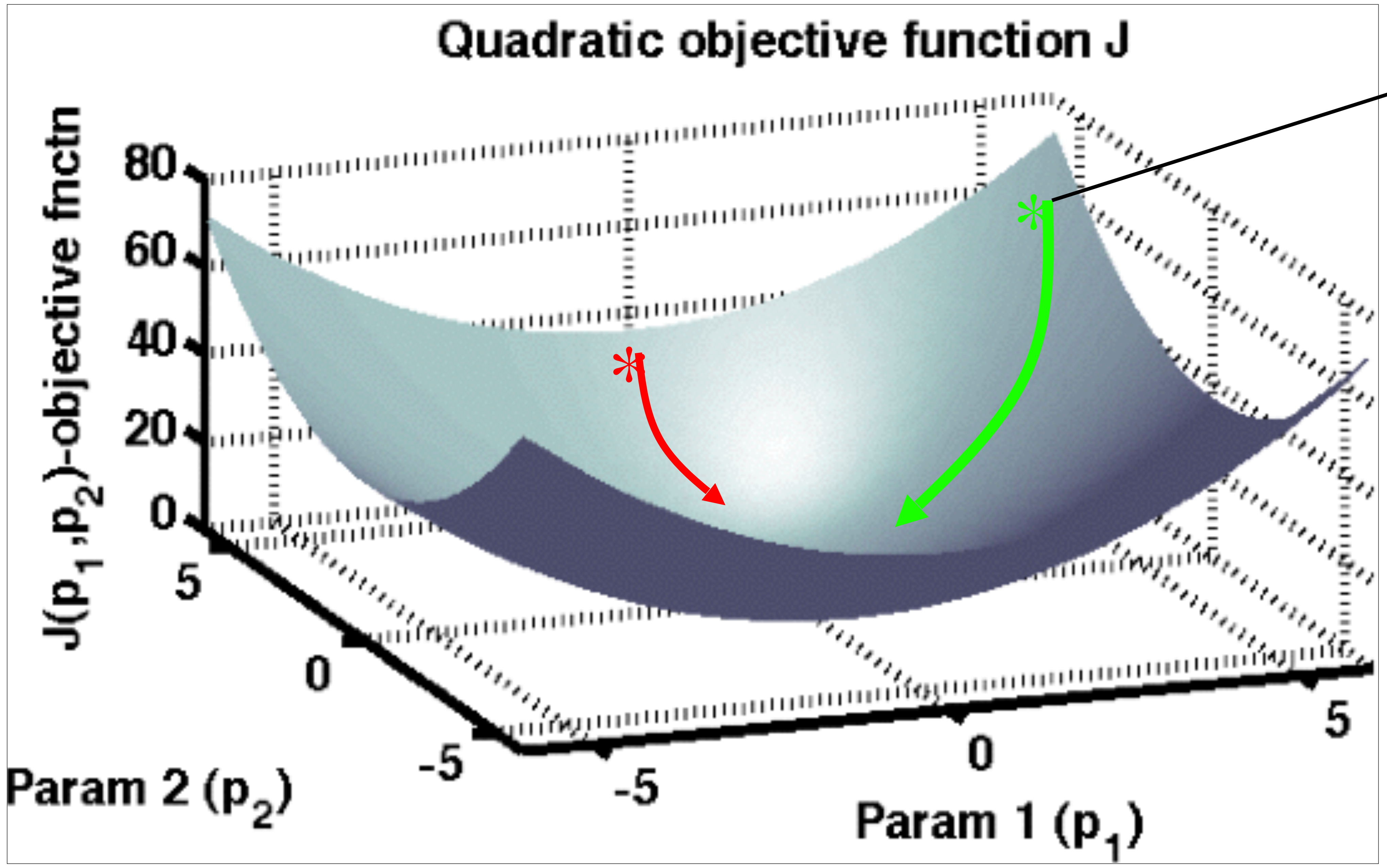
We also introduced the gradient descent method

- Intuitive algorithm - go ‘downhill’ for the parameters in the objective function you want to minimize
- Useful for
  - **Solution of a large linear system of equations**
  - **Solution of a nonlinear systems of equations**
    - Special note - some Artificial Neural Network Algorithms use gradient descent on the weights (more on this later)
  - **Optimization and control of dynamic systems**

How does the gradient descent algorithm work?

- Consider first the objective of gradient descent
  - **You want to get to the bottom of the hill**
  - **Start somewhere, then you ski down the hill**

$$J(\mathbf{x}) = \frac{1}{2}x^T A x - b^T x$$



Start here,

Then move towards the minimum

How do we do this mathematically?

- We want to minimize (A is assumed symmetric positive definite)

$$J(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - b^T\mathbf{x}$$

- We do this by starting with some initial guess for our parameters, and then ‘skiing’ downhill along the direction r with some ‘speed’ alpha at each iteration k

$$x_{k+1} = x_k + \alpha_k r_k$$

- So we’ll proceed iteratively toward the minimum of *J(x)*
  - **We want to move down the opposite of the gradient of J**

Computing the gradient of J(x)

- r at iteration k is given by taking the gradient of J(x) with respect to x

$$r_k = -\nabla J(\mathbf{x}_k) = -(A\mathbf{x}_k - \mathbf{b})$$

- With the gradient of J computed by

Note that
Since A is symmetric positive semi-definite

$$A\mathbf{x} = \mathbf{x}^T A$$

$$J(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - b^T\mathbf{x} \Rightarrow \nabla J(\mathbf{x}) = \frac{1}{2}A\mathbf{x} + \frac{1}{2}\mathbf{x}^T A - b$$
$$= A\mathbf{x} - b$$

So now we have the direction to move at iteration k…

Computing alpha

- We have to determine the step size (distance to go) at the iteration k
- We will compute the alpha at iteration k that minimizes

$$J(\mathbf{x}_k + \alpha_k r_k)$$

After a little work, we find alpha…

$$J(\mathbf{x}+\alpha r) = \frac{1}{2}(\mathbf{x}+\alpha\mathbf{r})^T A(\mathbf{x}+\alpha\mathbf{r}) - b^T(\mathbf{x}+\alpha\mathbf{r})$$

$$\frac{\partial J(\mathbf{x}+\alpha r)}{\partial \alpha} = \frac{1}{2}r^T A(\mathbf{x}+\alpha\mathbf{r}) + \frac{1}{2}(\mathbf{x}+\alpha\mathbf{r})^T A\mathbf{r} - b^T\mathbf{r}$$

$$= \alpha r^T A r + r^T A x - r^T b$$

$$= \alpha r^T A r + r^T (A x - b)$$

$$= \alpha r^T A r - r^T r$$

The minimum occurs at 0

$$\frac{\partial J(\mathbf{x}+\alpha r)}{\partial \alpha} = 0 \qquad 0 = \alpha r^T A r - r^T r$$

$$\alpha r^T A r = r^T r$$

We can divide here because these are all scalars (one number)

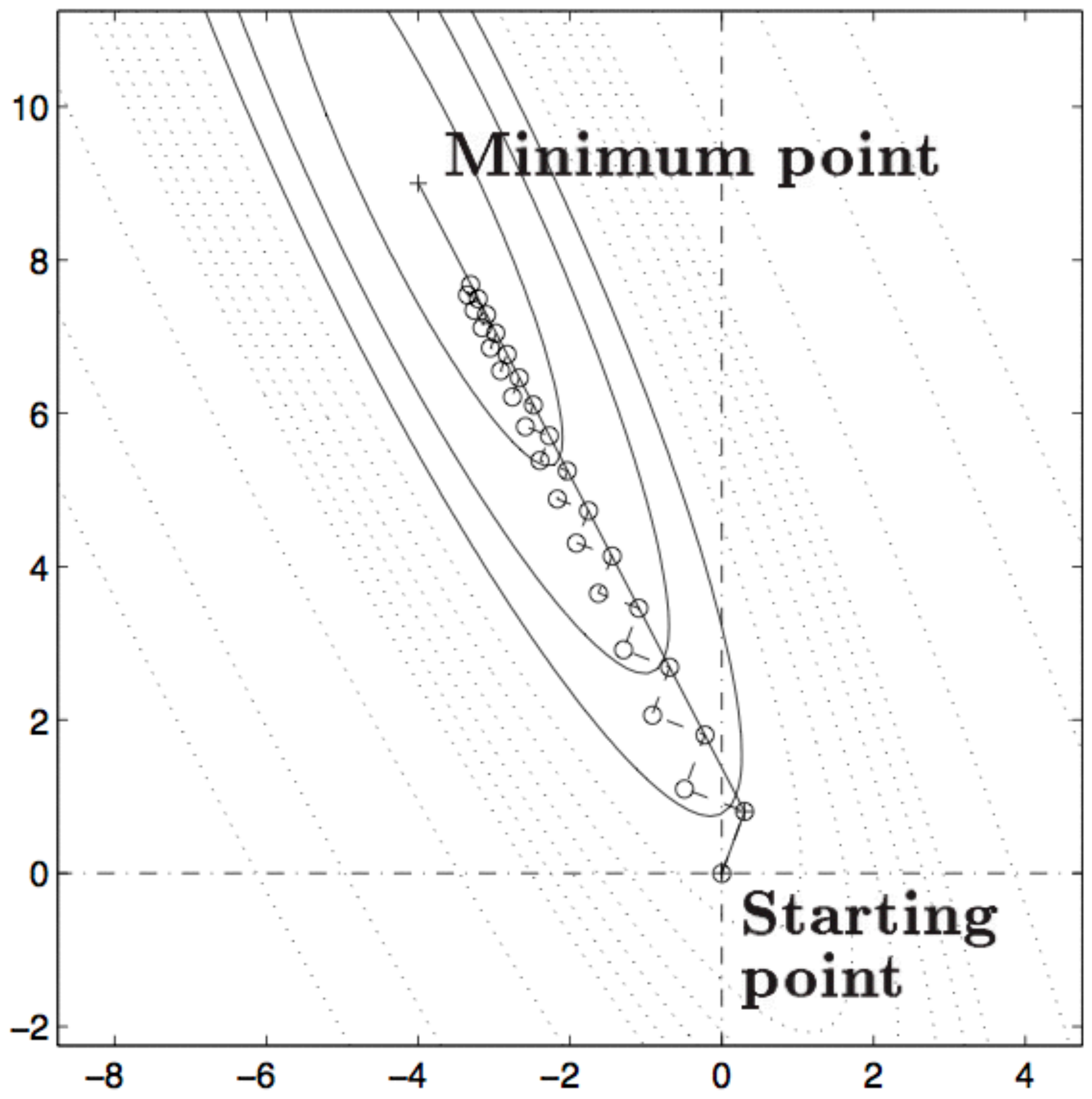
$$\boxed{\alpha = \frac{r^T r}{r^T A r}}$$

So finally we have each part…

- Given an initial condition, we can iteratively head towards the minimum of a function J
  - We compute the direction r and step size alpha at each k
  - If we have a small enough error between Ax-b, we stop
  - Or we stop if we've iterated too many times, as a convergence check

But…

- There are issues with this method when the objective function is more challenging - with very steep sides and long flat valleys (*poorly conditioned*)
- This method also is a bit inefficient since it must ‘tack’ back and forth at 90 degree increments
  - **Due to successive line minimization and lack of momentum from one iteration to the next**
  - **THERE HAS TO BE A BETTER WAY!!!**



THERE IS - Conjugate Gradient Descent

- When you ski, you don’t instantaneously tack back and forth, you have some momentum from the previous moment leading you to the next
- With a slight modification to the previous method we can arrive at a method that doesn’t get hindered by long narrow valleys

How CG improves over steepest descent

- Instead of minimizing over a single alpha, which does one direction at a time for that iteration, we minimize our function in every direction simultaneously while only searching in one direction at a time
  - **In other words, to converge in exactly m iterations to the answer, we should minimize over all the steps we'll take at once**
  - **(ie we can think of this as minimizing in m directions simultaneously)**
- We can do this in any number of search directions
  - **Prevents that 'tacking' phenomena exhibited by the gradient descent method**

How it's done…

- We can reduce this problem to minimizing each direction individually provided the different directions are independent of each other, or ***conjugate*** in the following sense

$$\boxed{p^{(i)^T} A p^{(j)} = 0, i \neq j}$$

- We can choose our p's so they are conjugate in the following way

Solving in m iterations

- Consider that we start with our initial guess x_0, then move to our solution, x_m

$$x_m = x_0 + \sum_{j=0}^{m-1} \alpha_j p_j$$

- Substitute that into J, then compute the partial derivative with respect to each alpha, set that equal to zero

$$J(x_m) \Rightarrow \frac{\partial J(x_m)}{\partial \alpha_k} = 0$$

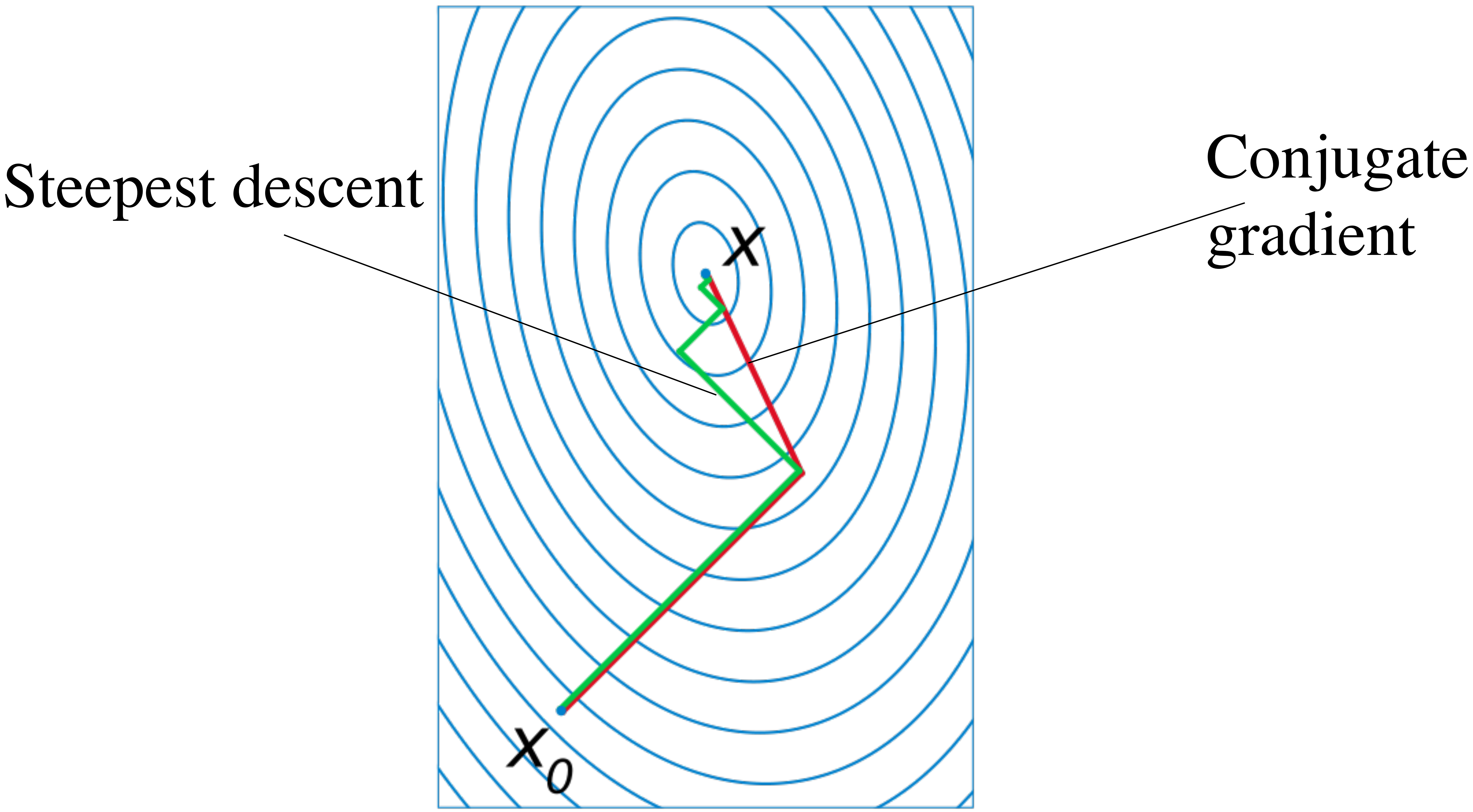
What does it boil down to?

- We compute a sequence of p's which are conjugate
  - **We redefine the descent direction at each iteration after the first to be a linear combination of the direction of steepest descent r and the previous descent direction**

$$\mathbf{p}^{(k)} = \mathbf{r}^{(k)} + \beta\,\mathbf{p}^{(k-1)} \quad \text{and} \quad \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha\,\mathbf{p}^{(k)}$$

$$\beta = \frac{{\mathbf{r}^{(k)}}^T \mathbf{r}^{(k)}}{{\mathbf{r}^{(k-1)}}^T \mathbf{r}^{(k-1)}}, \qquad \alpha = \frac{{\mathbf{r}^{(k)}}^T \mathbf{r}^{(k)}}{{\mathbf{p}^{(k)}}^T A\,\mathbf{p}^{(k)}}.$$

The result - an improvement



Many types of optimization - a whole field

- Golden section
- Gradient-based methods
- NM simplex
- Newton's method
- Bisection
- Many others

In summary

- **Optimization** is an interesting way to understand and model the world as well as solutions to difficult problems

- **Numerical optimization** provides tools for finding parameters for functions we could otherwise not solve for and that fail in simple regression type cases

- At times it provides a tool that is more efficient than solving the problem if solvable

- There are many approaches from simple to complex

- Simple solutions like NM and CGD are very practical

- **Reading: https://scipy-lectures.org/advanced/mathematical_optimization/index.html#smooth-and-non-smooth-problems**