Minimisation in N dimensions

$N > 1$

Why this is very hard !!

Direction Set methods

Powell's search

Simplex Search (Nelder-Mead)

Other approaches

In general, minimisation in N>1 dimensions is *much harder.*

1. Bracketing is inherently almost impossible

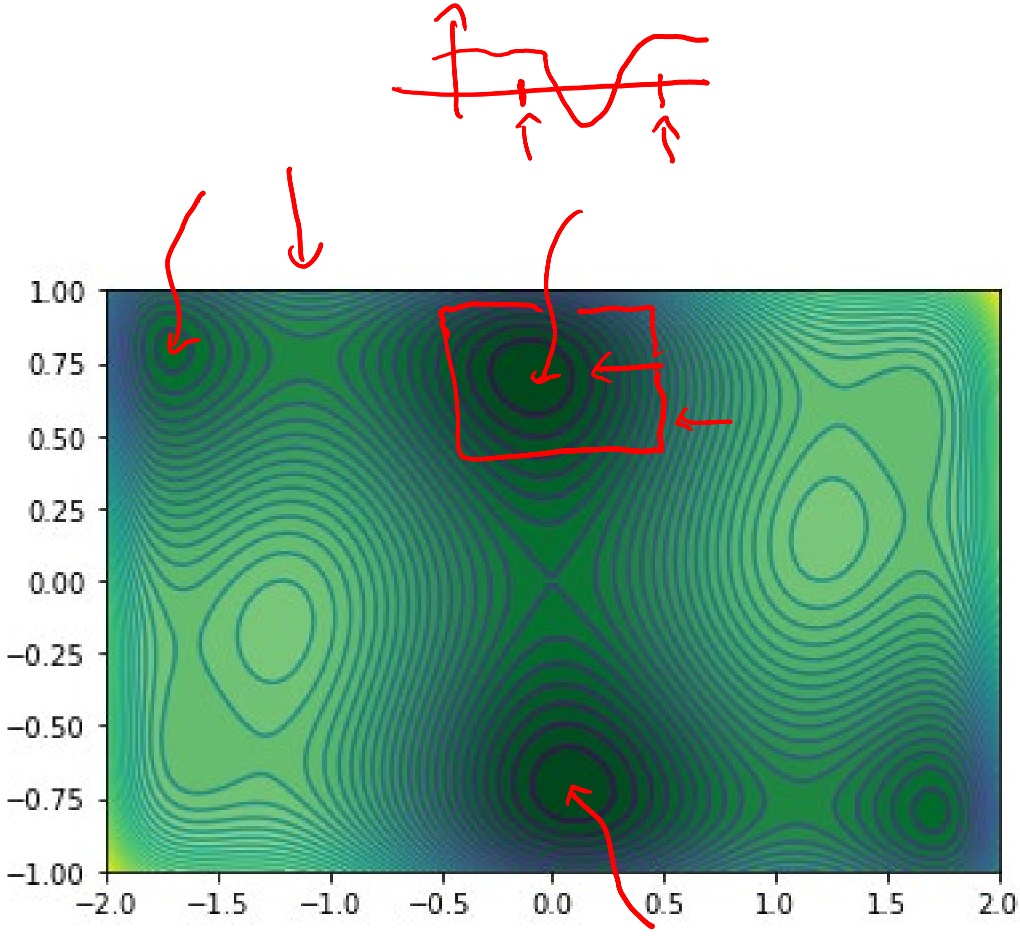
   In 1D you need two points for maintaining a bracket.
   In 2D you need a line, in 3D you need a surface etc.

   + it is almost impossible to stop the root from "leaking out the edges"

2. All the instabilities and problems from 1D are multiplied

3. Bisection, which relies on bracketing, is generally unfeasible

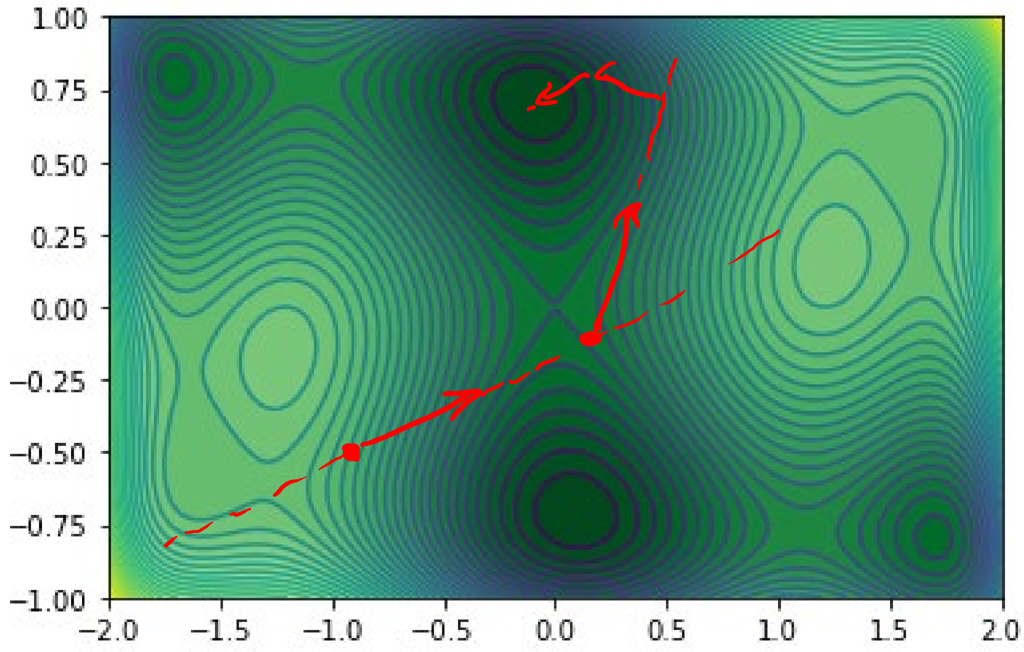
4. You have N times the number of unknown variables, so everything is in general much slower



<u>Direction Set methods: overview</u>

These methods all follow the following strategy:

1. Pick a direction in the parameter space
2. Minimise the value of the function along this direction (using 1D minimisation)
3. Switch to a new direction
4. Repeat



The methods differ in how they *choose and maintain the set of directions* that they're minimising along.

Each direction set method relies on a search of the function

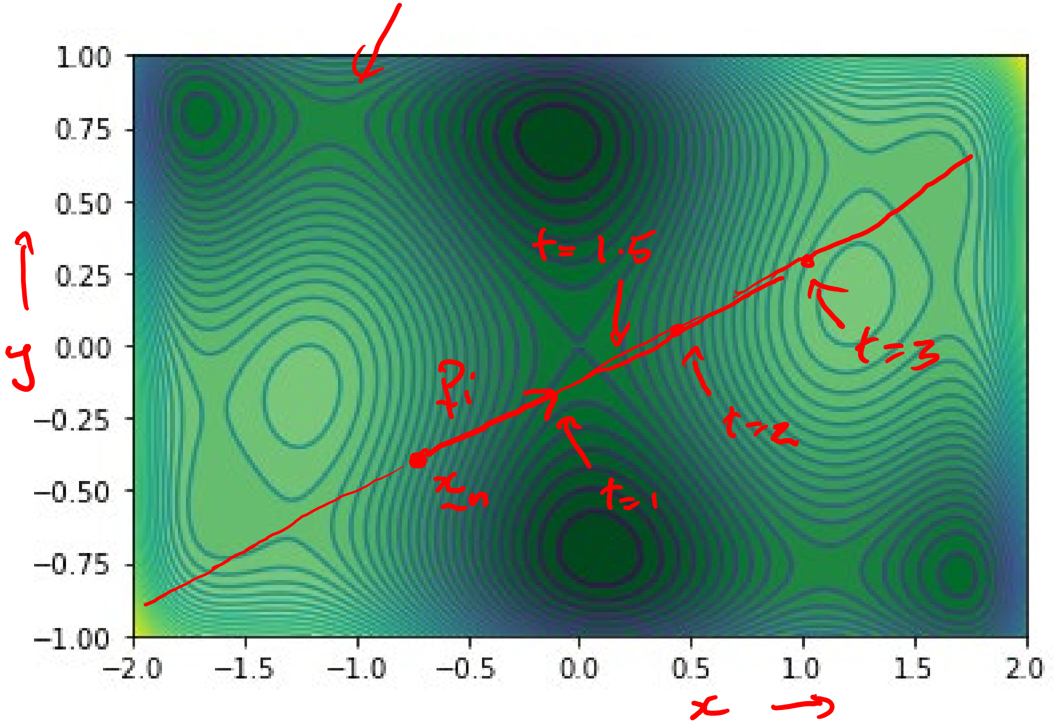
$$f(\mathbf{x}(t))$$

Where $\boldsymbol{x}(t)$ are points along the line given by

direction vector

$$\mathbf{x} = \mathbf{x}_n + \mathbf{p}_i t \quad \text{parameter}$$

$\underset{\sim}{x}(t)$ initial point



Here $\mathbf{x}_n$ is the starting point

$\mathbf{p}_i$ is the i-th vector direction of the line

$\underset{\sim}{x} = \langle x, y \rangle$.

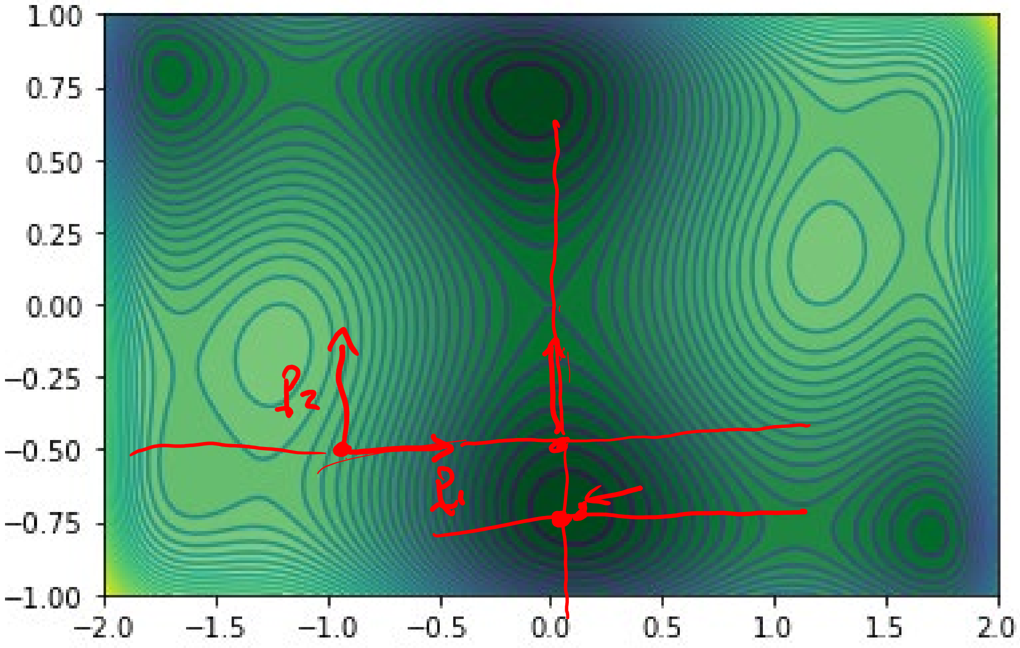
Both $\underset{\sim}{x}$ and p are N-dimensional

<u>Simple coordinate search</u>

Here we pick a fixed set of directions corresponding to the unit coordinate axes:

In 2D: $\mathbf{p}_1 = (1,0)$
$\mathbf{p}_2 = (0,1)$ 2D

In 3D: $\mathbf{p}_1 = (1,0,0)$
$\mathbf{p}_2 = (0,1,0)$
$\mathbf{p}_3 = (0,0,1)$



Simple search algorithm:

Loop over i = 1…N

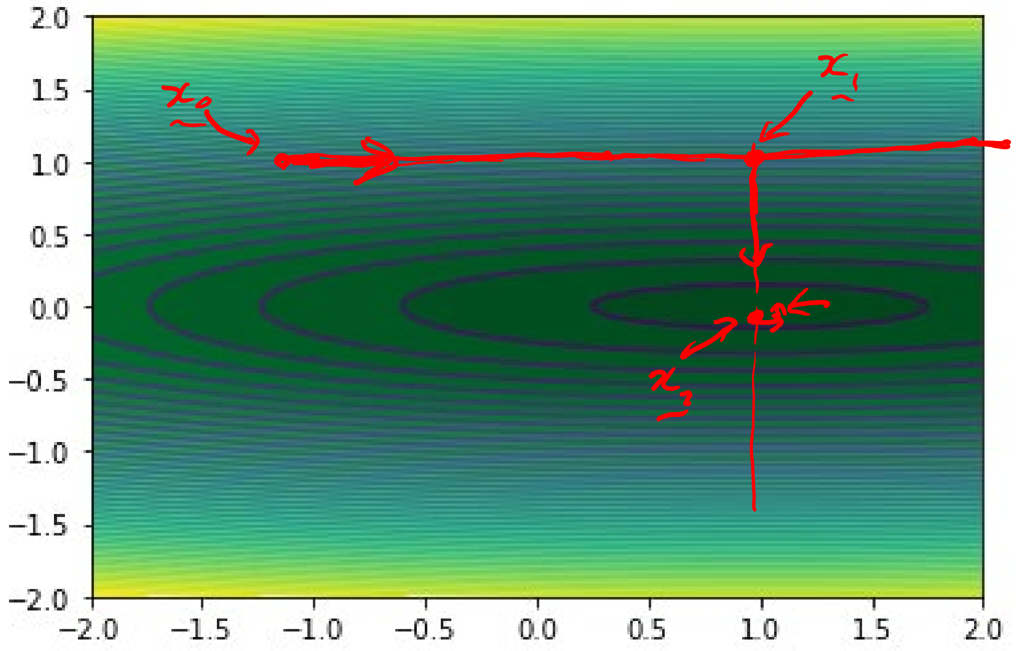
Starting from $\mathbf{x}_n$, minimise $f(\boldsymbol{x})$ along the line

$$\underline{\mathbf{x} = \mathbf{x}_n + \mathbf{p}_i t}$$

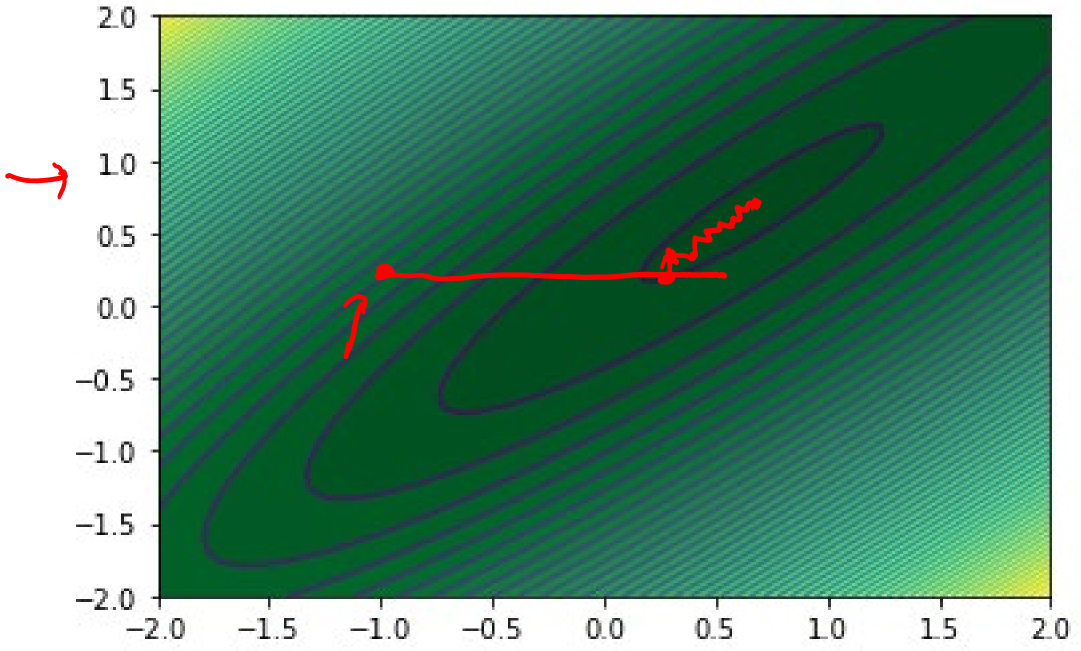
pick new $\mathbf{x}_n$ as the minimum point

Repeat until converged

This will work really well for some situations…



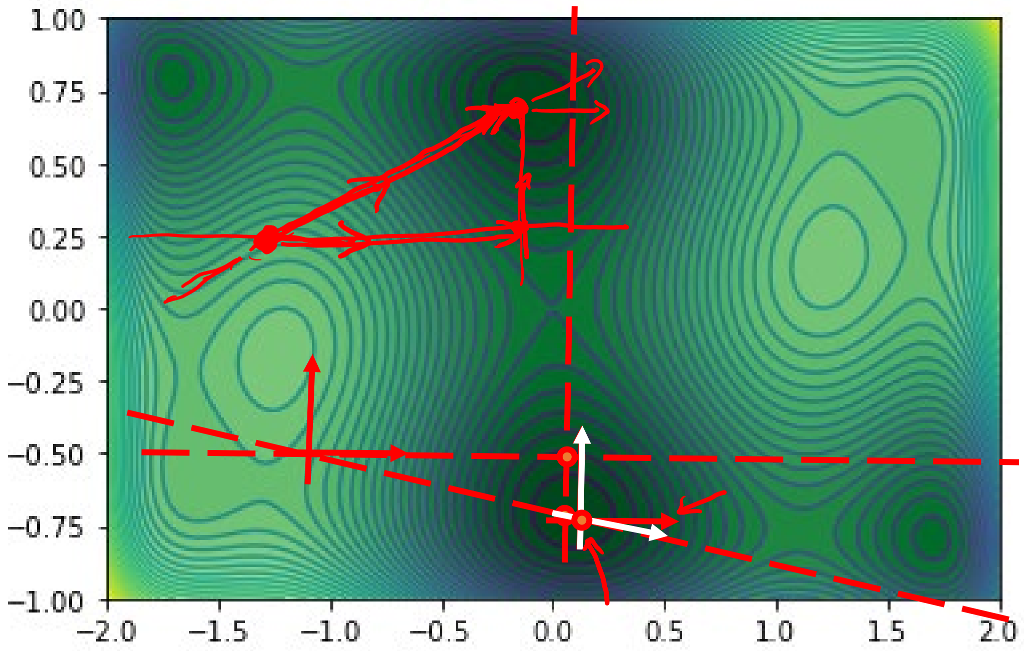
But not so well for others.



Powell's direction-set method

This is one of the "gold-standard" methods for multi-dimensional search.

The central idea: update the direction set each time, replacing the *best direction* each time with the vector connecting the old point to the new point.



Powell's search:

1. Loop over the different directions i = 1 to N

   Starting at $\mathbf{x}_0$, perform a minimum search in direction $\mathbf{p}_i$. Call the minimum $\mathbf{x}_i$, with value f($\mathbf{x}_i$).

   start the next search at $\mathbf{x}_i$

2. Perform a final minimum search starting at the original point $\mathbf{x}_0$ and going in the direction towards the final point $\mathbf{x}_N$

3. Check for convergence with $|\mathbf{x}_0 - \mathbf{x}_i| < tol$

4. If not converged, find the direction i_{max} with the biggest decrease, i.e. where f($\mathbf{x}_0$)-f($\mathbf{x}_i$) is largest.

5. Replace $\mathbf{p}_{imax} =$ $\mathbf{x}_0$-$\mathbf{x}_N$, then repeat the whole process.

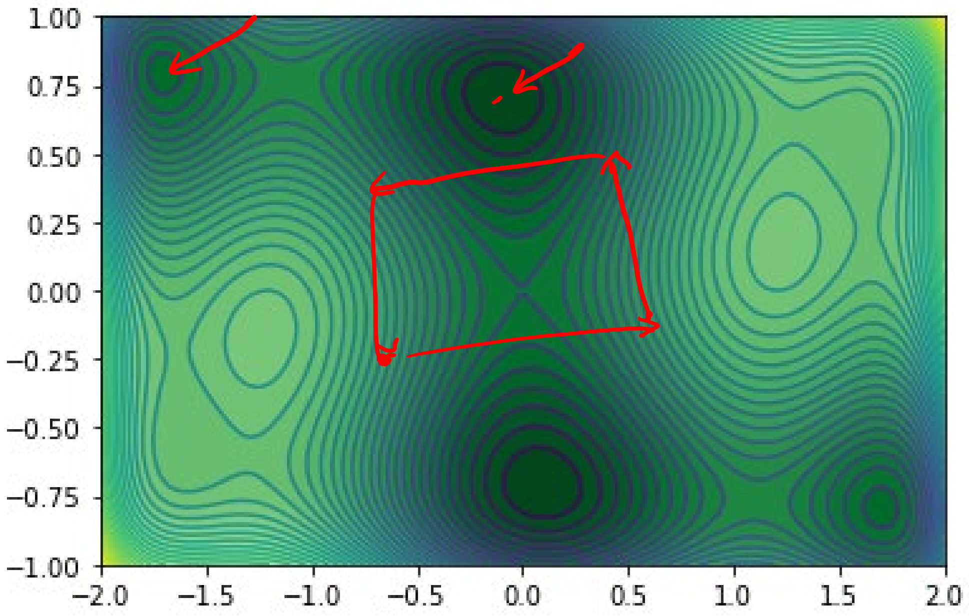
Advantages of Powell's method:

It is pretty robust

No derivatives are needed

It is linear, but is quick, converging on the minimum.

The search time scales linearly with the number of dimensions



Disadvantages:

You might not get the "right" minimum

it can sometimes get "stuck" in an analogous way to Newton's method.

(A warning for newcomers):
It is very dependent on your 1D minimisation working flawlessly

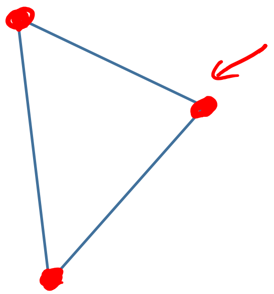
The Downhill Simplex method (Nelder-Mead algorithm)

This is a completely different approach which works really well and is extremely robust.

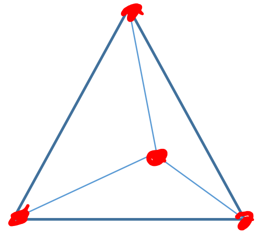
The idea: create an "amoeba" which tries out points in the surrounding space, then either expands to crawl downhill or contracts around the minimum.

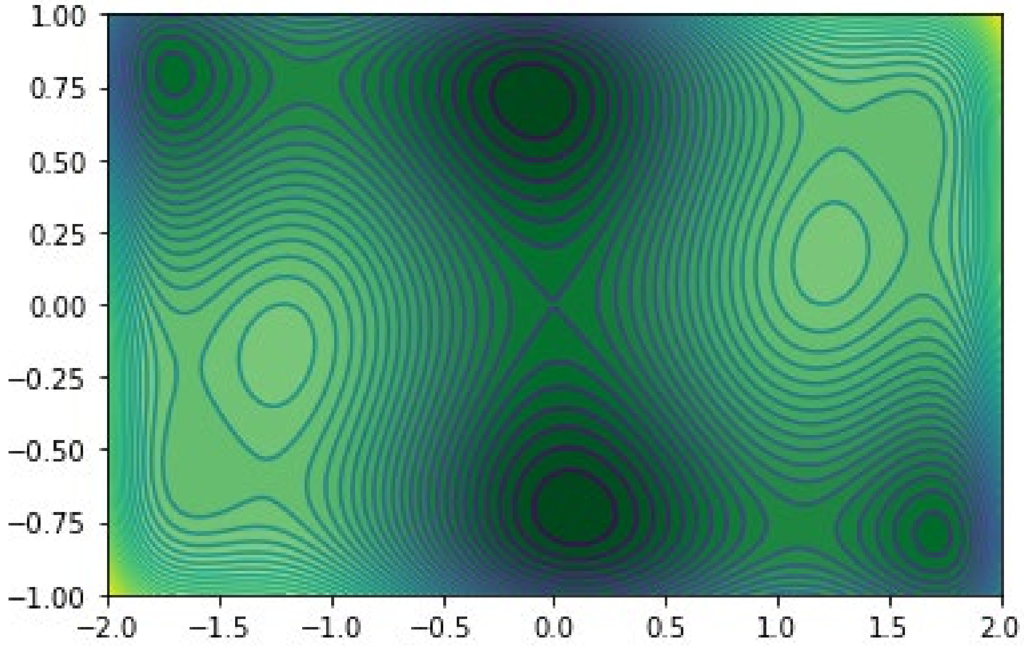
Definition: a *simplex* is a set of N+1 points in N dimensions.

In 2D:



In 3D:





The Nelder-Mead algorithm gives a set of rules to transform the simplex so that it converges on a minimum.

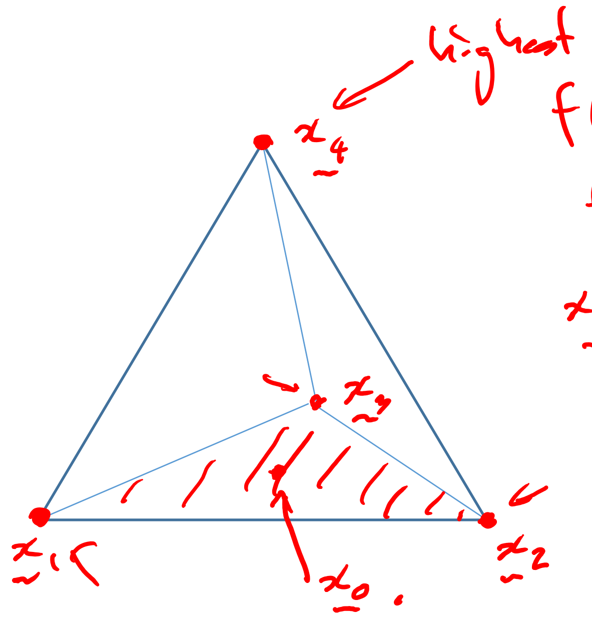
First note that for a given simplex we can order the vertices from lowest to highest

$$f(\mathbf{x_1}) \leq f(\mathbf{x}_2) \leq \dots \leq f(\mathbf{x}_{N+1})$$

$$x_0 = \frac{1}{2}(x_1 + x_2)$$

and compute the *centroid of all the x's but the highest one:*

$$\mathbf{x}_0 = \frac{1}{N}\sum_{j=1}^{N} \mathbf{x}_j$$

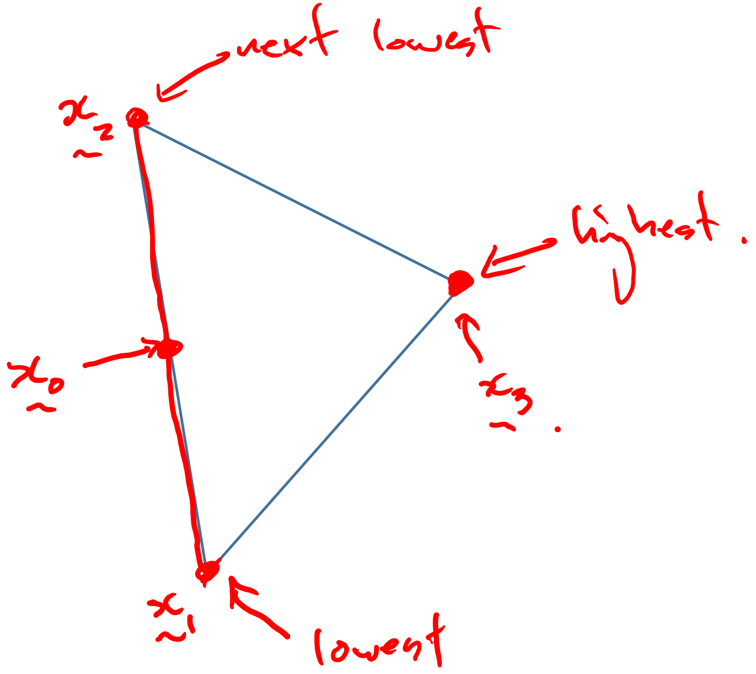


$$f(x_4) \geq f(x_i)$$

for $i = 1, 2, 3.$

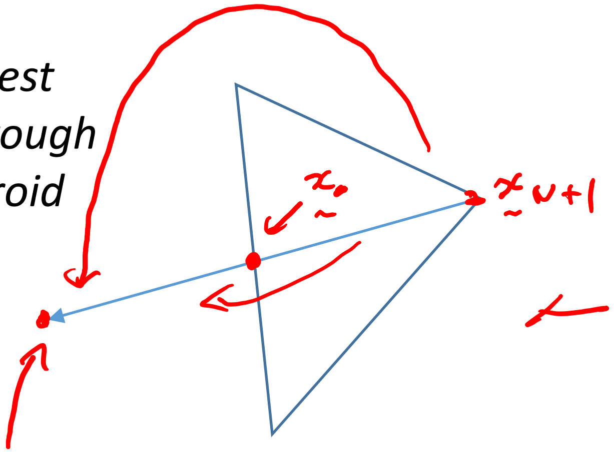
$$x_0 = \frac{1}{3}\sum_{j=1}^{3} x_j$$

$$= \frac{1}{3}(x_1 + x_2 + x_3)$$



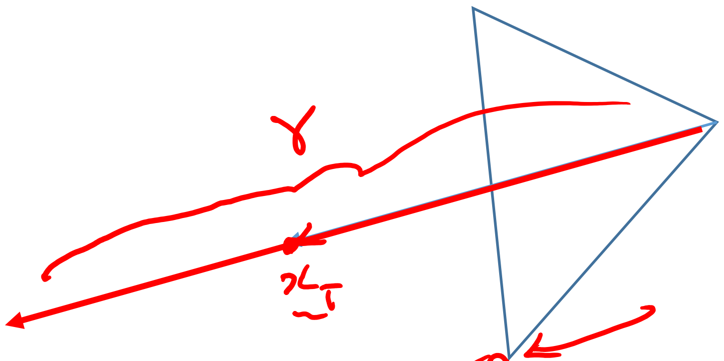
Things the simplex can do:

Reflect
The highest point through the centroid



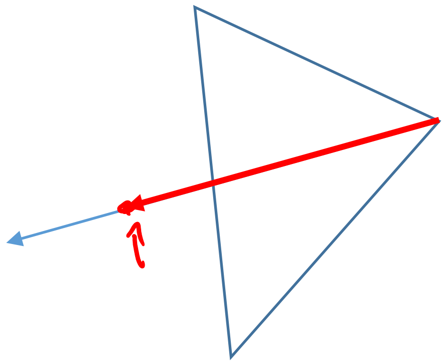
$$\mathbf{x}_T = \mathbf{x}_0 + \alpha(\mathbf{x}_0 - \mathbf{x}_{N+1})$$

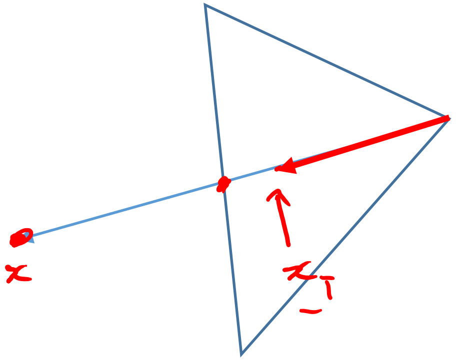
Expand a reflected point



$$\mathbf{x}_e = \mathbf{x}_0 + \gamma(\mathbf{x}_T - \mathbf{x}_0)$$

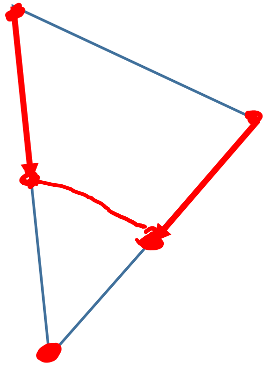
Contract a reflected point on the outside or inside of the simplex





$$\mathbf{x}_T = \mathbf{x}_0 + \rho(\mathbf{x}_0 - \mathbf{x}_{N+1}) \qquad \mathbf{x}_T = \mathbf{x}_0 - \rho(\mathbf{x}_0 - \mathbf{x}_{N+1})$$

Shrink
Towards the best point



$$\mathbf{x}_i = \mathbf{x}_i - \sigma(\mathbf{x}_1 - \mathbf{x}_i)$$

Algorithm for simplex search:

1. Order the vertices from best to worst
2. Test for convergence
3. Transform the simplex (below)
4. Repeat

Transformation sequence

1. Reflect

$$\mathbf{x}_T = \mathbf{x}_0 + \alpha(\mathbf{x}_0 - \mathbf{x}_{N+1})$$

Compute $f(\mathbf{x}_T)$ and compare it to the existing values.

If $f(\mathbf{x}_T)$ is...

great

$f(\mathbf{x}_T) < f(\mathbf{x}_1)$

Expand, retest and Form a new simplex with whichever new point is better, Discard the worst point

Good but not great

$f(\mathbf{x}_T) < f(\mathbf{x}_N)$

$f(\mathbf{x}_T) > f(\mathbf{x}_1)$

current worst point

Keep the reflected point
Discard the worst point

Bad, but not that bad

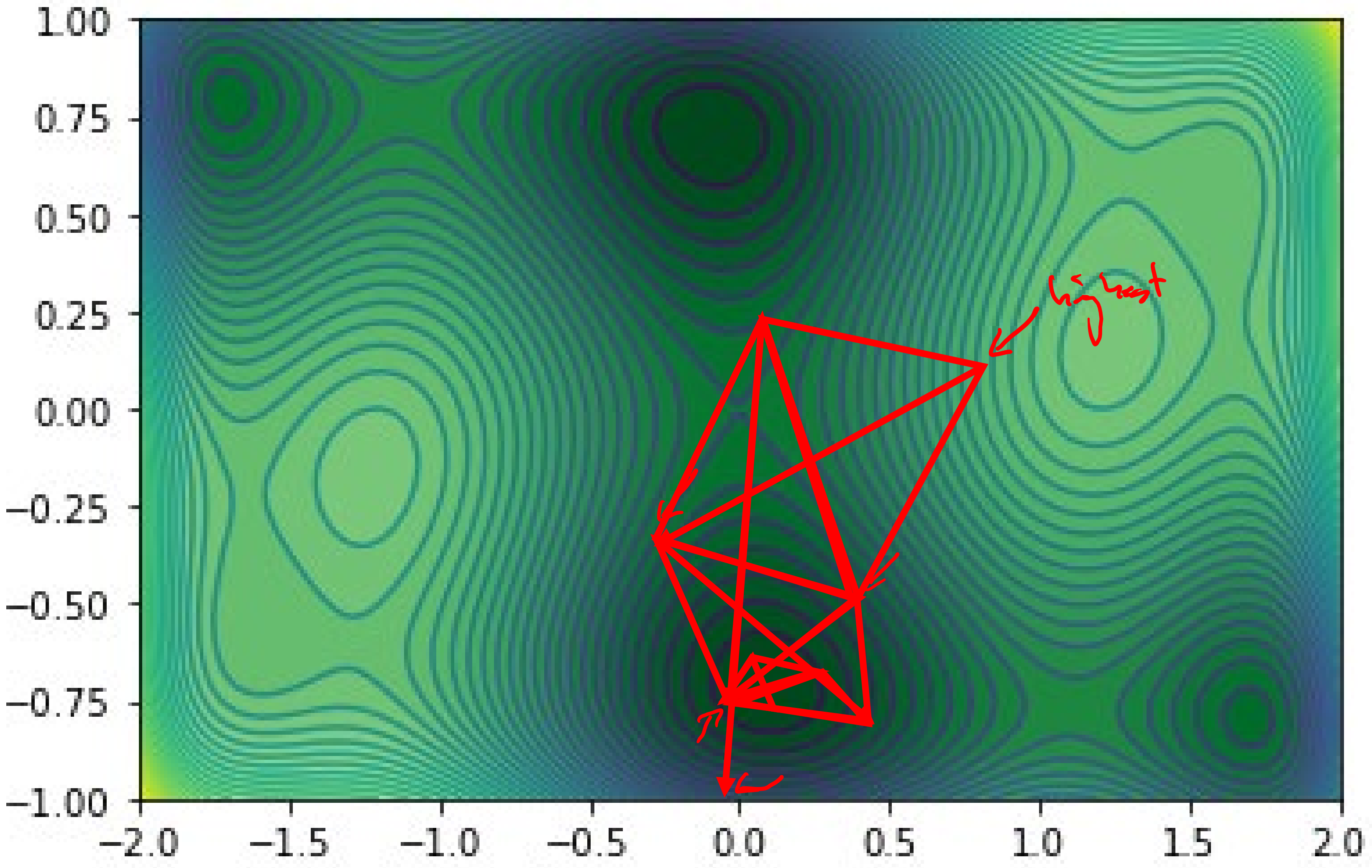
$f(\mathbf{x}_T) < f(\mathbf{x}_{N+1})$

Contract outside the simplex, Keep it if it improves, discarding the worst point, otherwise *Shrink*

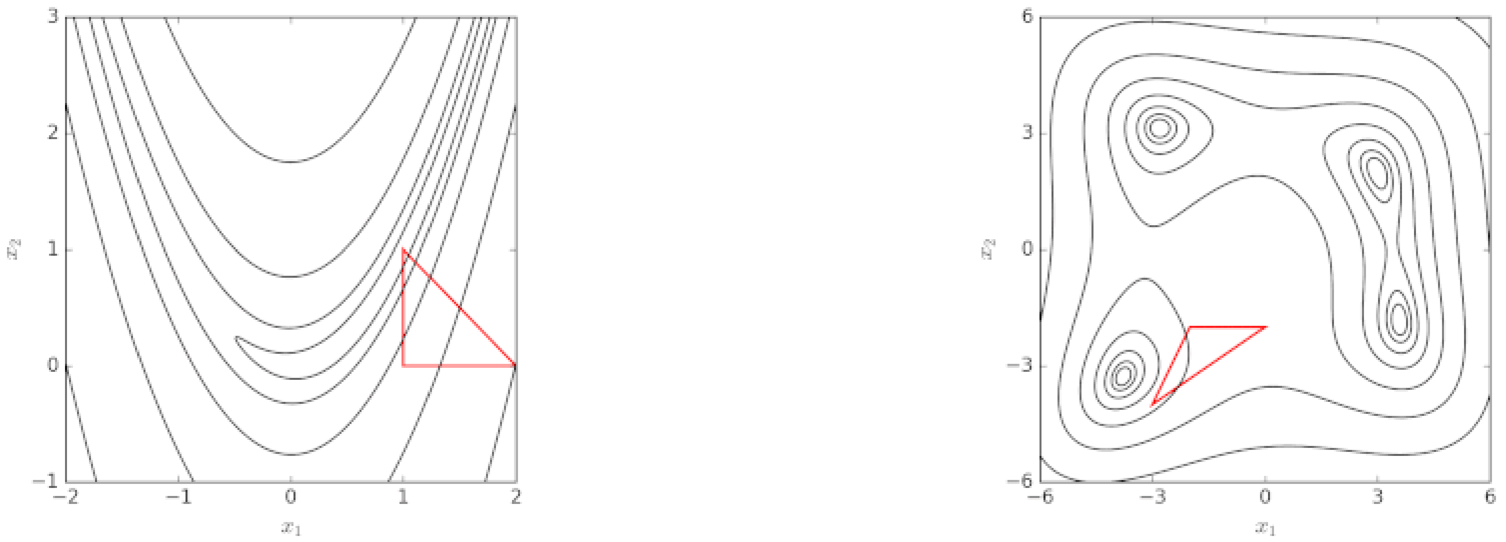
terrible

$f(\mathbf{x}_T) \geq f(\mathbf{x}_{N+1})$

Contract inside the simplex, Keep it unless it's still terrible, Discarding the worst point, otherwise *Shrink*



1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
highest



x_1
x_2
-2
-1
0
1
2
3
x_1
x_2
-6
-3
0
3
6

Other methods:

Steepest descent method

-basically “go downhill, increase the stepsize as you do so, then backtrack when you start going uphill”

Conjugate gradient methods (Fletcher-Reeves algorithm)

- concentrate on moving downhill in an “optimal” way