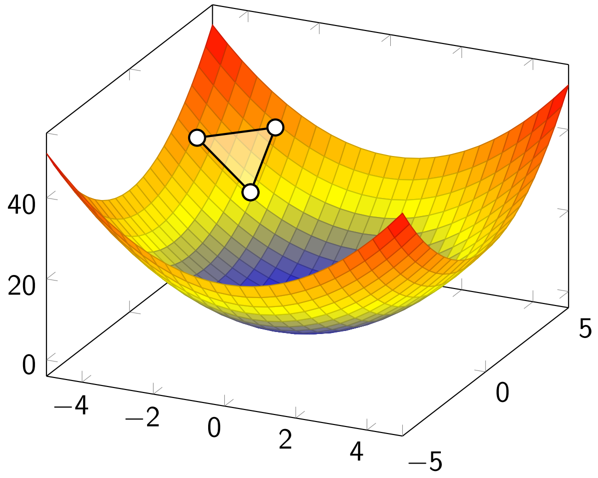
Nelder-Mead Simplex

for Optimization



Background

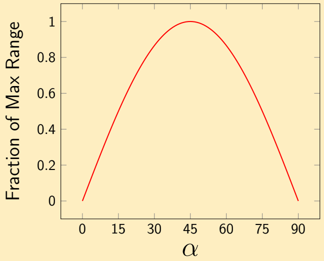
Optimization is the process of finding a combination of parameters that yields the “best” result for a given problem.

Example

At what angle should a ball be thrown to travel the maximum distance away from you (in a vacuum)?

$$Range = \frac{V^2}{g} sin(2\alpha)$$

$Range$ maximized at $\alpha = 45^\circ$



Background (cont)

The **objective function**, f, evaluates the quantity (or quantities) of interest as a function of the **design variables**, x. We usually try to minimize the objective function.

$$\textit{minimize } f(x)$$

$$\textit{subject to } x \in X$$

where X is the **design space** of the problem.

There are many iterative computational algorithms for optimization, including:

- gradient-based methods
- evolutionary methods
- other fancy math methods that make no sense

Motivating Problem

Voices in your head indicate that treasure is buried at the deepest point in a local pond. With only a rowboat and a tape-measure to measure the depth of water below you, how might you find the treasure?

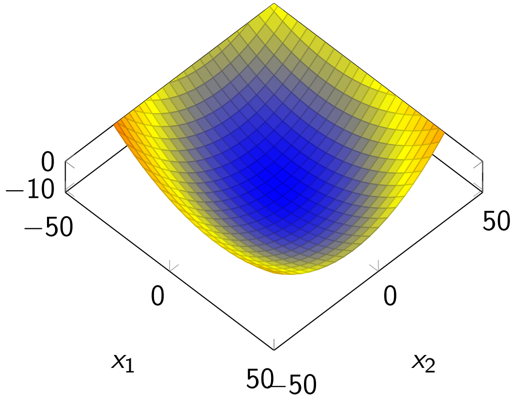
Would you...

- Spend all night measuring the depth everywhere in the pond?
- Make an initial guess, perturb your guess East and North, compute the slopes, move some distance in the direction of maximum descent, and repeat until you can't find a deeper point?
- Make 3 initial guesses, make a 4th guess based on the others, throw away the worst of your 4 guesses, and repeat until you can't find a deeper point?

Motivating Problem (cont)

Instead of taking physical measurements, we will evaluate:

$$Depth = f(x_1, x_2) = .006(x_1 - 1)^2 + .009(x_2 - 2)^2 + .005x_1x_2 - 10$$



Introduction to Nelder-Mead

Nelder-Mead is an iterative numerical method to find the (local) minimum or maximum of an objective function.

For an optimization problem with n variables, a simplex of $n + 1$ points is created which gradually moves and contracts around the local optimal value. This method does not involve computing the derivatives/gradient of the objective function.

A **simplex** is basically a generalization of what a triangle is in 2D space. For example, a simplex in 3D would be a tetrahedron, and a simplex in 1D would be a line segment.

The algorithm is very simple to understand. I have adapted this explanation from the original paper by JA Nelder and R Mead, which is available for free online:

`Nelder, J. A., & Mead, R. (1965). A Simplex Method for Function Minimization. The Computer Journal, 7(4), 308-313.`

The Algorithm

I will explain the algorithm generically for a n-dimensional design space, but to help you visualize, I will illustrate the algorithm on a 2D design space (using a triangular simplex).

We will attempt to find the local minimum of our objective function. If you would like to find a local maximum instead, negate the objective function:

$$\textit{maximize } g(x) = \textit{minimize } f(x)$$
$$\textit{for } f(x) = -g(x)$$

Step 0, Initialize Simplex

The zeroth step is to evaluate the initial vertices of the simplex. Evaluate the objective function at $n + 1$ points:

$$P_0, P_1, P_2, \dots P_n$$

$$P_i = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}_i$$

These points should ideally by centered close to our best guess of the optimum.

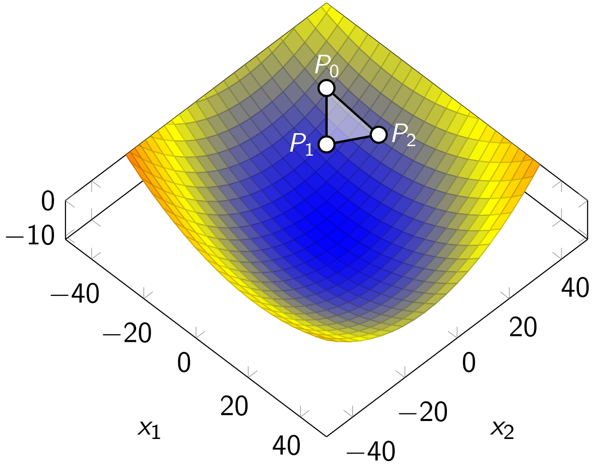
Keep in mind that this algorithm may only converge on a local optimum, and that may not in general be the global optimum.

Let the function value at the vertices be represented by y_i at point P_i:

$$y_i = f(P_i)$$

Step 0, Initialize Simplex (cont)

Here is how an initial simplex might look for our treasure-seeking problem:



$$P_0 = \begin{bmatrix}-30 & 30\end{bmatrix}$$
$$y_0 = f(P_0) = -1.678$$

$$P_1 = \begin{bmatrix}-20 & 20\end{bmatrix}$$
$$y_1 = f(P_1) = -6.438$$

$$P_2 = \begin{bmatrix}-10 & 30\end{bmatrix}$$
$$y_2 = f(P_2) = -3.718$$

Step 0, Initialize Simplex (cont)

Because our task is to eventually find a P_i associated with a local minimum y_i, we can compare the y_i values that we have in order to get an idea of the direction in which the function f is decreasing most.

Obviously the P_i associated with the highest (worst) y_i is likely **NOT** in the direction of the optimum, whereas the P_i associated with the lowest (best) y_i **MAY BE** in the direction of the optimum.

Although it's not explicitly necessary, it may be useful to order the points P_i by increasing/decreasing values of y_i to help keep track of our point ranks.

Step 1, REFLECT

The first simplex transformation is basically to **REFLECT** the worst point of the simplex across the centroid defined by the other points.

For simplicity, we will define P_{best} and P_{worst} as the best and worst points in the current simplex and y_{best} and y_{worst} as the corresponding function values.

$$y_{best} = \min_i(y_i) = \min_i(f(P_i)) = f(P_{best})$$

$$y_{worst} = \max_i(y_i) = \max_i(f(P_i)) = f(P_{worst})$$

We also define $\bar{P}$ as the centroid of all but the worst points in the simplex:

$$\bar{P} = \frac{1}{n} \sum_{P_i \neq P_{worst}} P_i$$

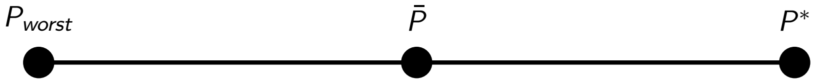
Step 1, REFLECT (cont)

The new reflected simplex vertex P^* can be expressed:

$$P^* = (1 + \alpha)\bar{P} - \alpha P_{worst}$$

where α is a positive term called the **reflection coefficient** and is often taken to be 1.

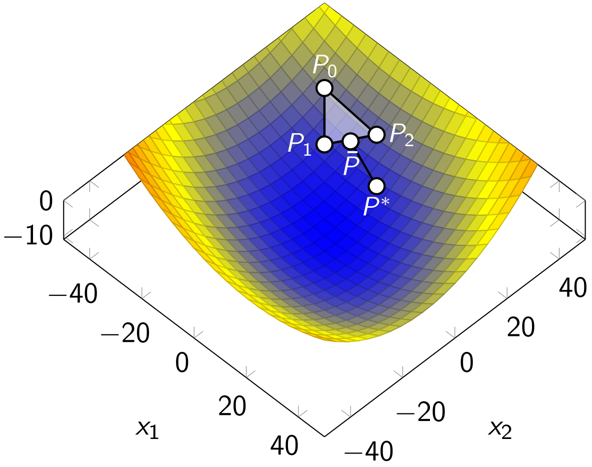
For $\alpha = 1$, $\bar{P}$ is exactly between P_{worst} and P^*.



Evaluate $y^* = f(P^*)$.

Step 1, REFLECT (cont)

Here is how a REFLECT operation might look for our treasure-seeking problem:



$$\bar{P} = \begin{bmatrix} -15 & 25 \end{bmatrix}$$

$$P^* = \begin{bmatrix} 0 & 20 \end{bmatrix}$$
$$y^* = f(P^*) = -7.078$$

Step 1, REFLECT (cont)

There a few options for the value of y^*:

- $y* \leq y_{best}$ (the reflected point is the best so far):
  - We should look further in the direction of P^*, continue to the **EXPAND** step.
- $y* \geq y_{2^{nd}worst}$ (if the reflected point replaced the worst point, it would still be the worst):
  - Reflecting didn't work, and our simplex is likely too big. We should contract our existing simplex away from of P_{worst}, continue to the **CONTRACT** step.
- $y_{best} < y* < y_{2^{nd}worst}$ (the reflected point is at most second-worst):
  - We can improve our simplex by replacing P_{worst} with P^*. Do so, and if we haven't satisfied the **STOPPING CRITERIA**, return to the **REFLECT** step.

Step 2, EXPAND

The next simplex transformation is basically to **EXPAND** the reflected point P^* on our simplex further in that direction. It helps us quickly traverse the design space in the direction of the expected minimum.

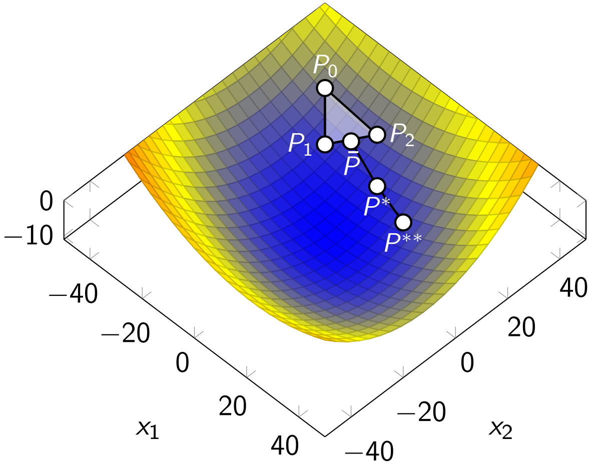
We define an expanded point P^{**}:

$$P^{**} = \gamma P^* + (1 - \gamma)\bar{P}$$

where γ is a term greater than 1 called the **expansion coefficient** and is often taken to be 2.

Step 2, EXPAND (cont)

Here is how an EXPAND operation might look for our treasure-seeking problem:



$$\bar{P} = \begin{bmatrix} -15 & 25 \end{bmatrix}$$

$$P^* = \begin{bmatrix} 0 & 20 \end{bmatrix}$$
$$y^* = f(P^*) = -7.078$$

$$P^{**} = \begin{bmatrix} 15 & 15 \end{bmatrix}$$
$$y^{**} = f(P^{**}) = -6.178$$

Step 2, EXPAND (cont)

There a few options for the value of y^{**}:

- $y^{**} \leq y_{best}$ (the expanded point is still the best so far, not counting y^{*}):
  - We can improve our simplex by replacing P_{worst} with P^{**}. Do so, and if we haven't satisfied the **STOPPING CRITERIA**, return to the **REFLECT** step.
- $y^{**} > y_{best}$ (the expanded point is not as good as y^{*} or y_{best}):
  - We can improve our simplex by replacing P_{worst} with P^{*}. Do so, and if we haven't satisfied the **STOPPING CRITERIA**, return to the **REFLECT** step.

Step 3, CONTRACT

The next simplex transformation is basically to **CONTRACT** the worst point P_{worst} inward on our simplex. It helps us reduce our simplex size to help converge on a minimum.

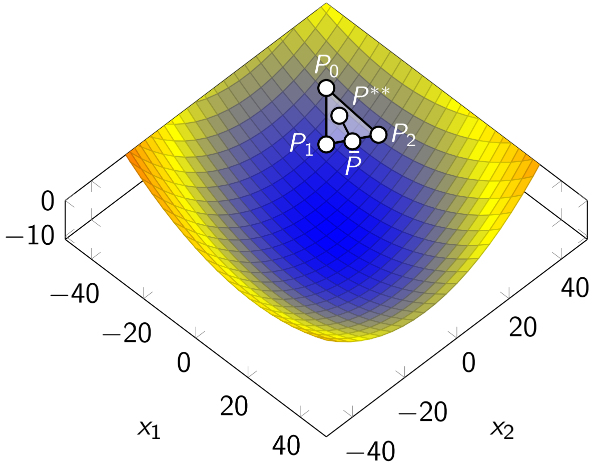
We define a contracted point P^{**}:

$$P^{**} = \beta P_{worst} + (1 - \beta)\bar{P}$$

where β is a term between 0 and 1 called the **contraction coefficient** and is often taken to be 0.5.

Step 3, CONTRACT (cont)

Here is how a CONTRACT operation might look for our treasure-seeking problem:



$$P_{worst} = \begin{bmatrix} -30 & 30 \end{bmatrix}$$
$$y_{worst} = f(P_{worst}) = -1.678$$

$$P^{**} = \begin{bmatrix} -22.5 & 27.5 \end{bmatrix}$$
$$y^{**} = f(P^{**}) = -3.916$$

$$\bar{P} = \begin{bmatrix} -15 & 25 \end{bmatrix}$$

Step 3, CONTRACT (cont)

There a few options for the value of y^{**}:

- $y^{**} > y_{worst}$ (the contracted point is somehow the worst one yet, very rare):
  - We can hopefully improve our simplex by shrinking all but P_{best} towards P_{best}. This is accomplished in the **SHRINK** step.
- $y^{**} \leq y_{worst}$ (the contracted point is better than P_{worst}):
  - We can improve our simplex by replacing P_{worst} with P^{**}. Do so, and if we haven't satisfied the **STOPPING CRITERIA**, return to the **REFLECT** step.

Step 4, SHRINK

The final simplex transformation is basically to **SHRINK** the worst points $P_i \neq P_{best}$ inward on our simplex. It helps us reduce our simplex size to help converge on a minimum.

We define the shrunken points P_i':

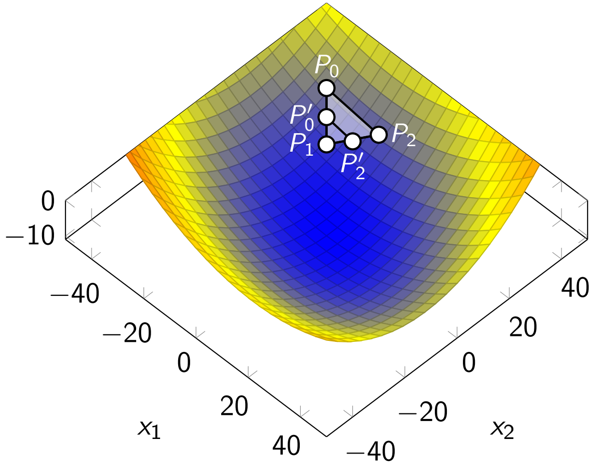
$$P_i' = P_{best} + \sigma(P_i - P_{best})$$

where σ is a term between 0 and 1 called the **shrink coefficient** and is often taken to be 0.5.

After the shrink operation, evaluate the **STOPPING CRITERIA**, and either **STOP** or return to the **REFLECT** step.

Step 4, SHRINK (cont)

Here is how a SHRINK operation might look for our treasure-seeking problem:



$$P_0' = \begin{bmatrix} -25 & 25 \end{bmatrix}$$
$$y_0' = f(P_0') = -4.308$$

$$P_2' = \begin{bmatrix} -15 & 25 \end{bmatrix}$$
$$y_2' = f(P_2') = -5.578$$

STOPPING CRITERIA

After each iteration of the evolutionary algorithm, we can evaluate whether or not there is enough information in the simplex to motivate another transformation. If the simplex is **flat**, that is, the values of y_i are all close to one another, there is no reason to believe any more transformations will further improve the simplex.

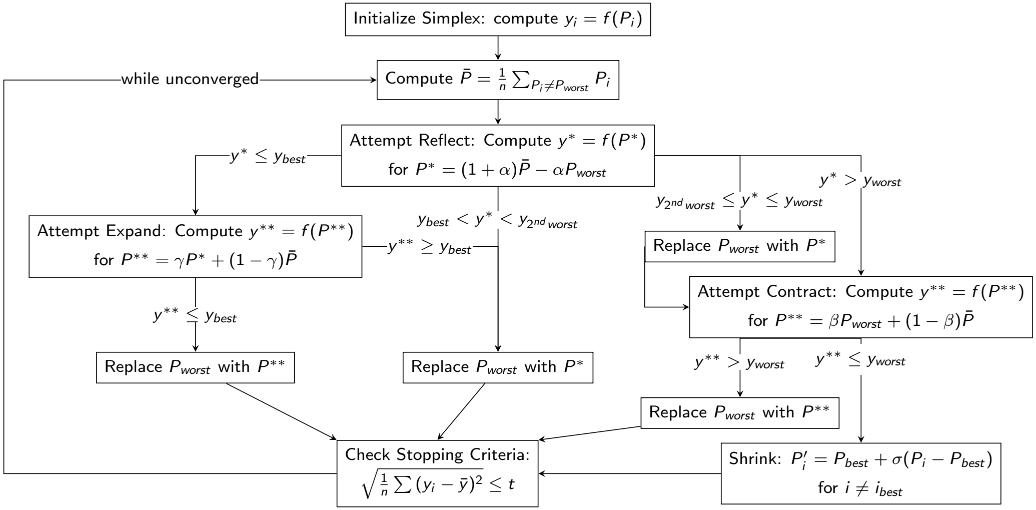
In this case, we can take P_{best} as the optimal combination of design parameters and y_{best} as the optimal value of the objective :function.

One common measure of simplex **flatness** is the standard deviation of y_i:

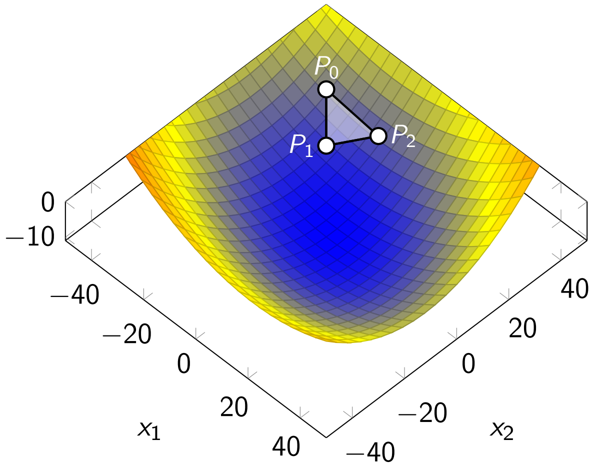
$$\sqrt{\frac{1}{n}\sum (y_i - \bar{y})^2} \leq t$$

where t is a tolerance and $\bar{y}$ is the mean of y_i. The tolerance value may vary for different objective functions.

Algorithm Summary


Initialize Simplex: compute $y_i = f(P_i)$
Compute $\bar{P} = \frac{1}{n}\sum_{P_i \neq P_{worst}} P_i$
while unconverged
Attempt Reflect: Compute $y^* = f(P^*)$ for $P^* = (1+\alpha)\bar{P} - \alpha P_{worst}$
$y^* \leq y_{best}$
$y_{best} < y^* < y_{2^{nd}\,worst}$
$y_{2^{nd}\,worst} \leq y^* \leq y_{worst}$
$y^* > y_{worst}$
Attempt Expand: Compute $y^{**} = f(P^{**})$ for $P^{**} = \gamma P^* + (1-\gamma)\bar{P}$
$y^{**} \geq y_{best}$
$y^{**} \leq y_{best}$
Replace P_{worst} with P^{**}
Replace P_{worst} with P^*
Replace P_{worst} with P^*
Attempt Contract: Compute $y^{**} = f(P^{**})$ for $P^{**} = \beta P_{worst} + (1-\beta)\bar{P}$
$y^{**} > y_{worst}$
$y^{**} \leq y_{worst}$
Replace P_{worst} with P^{**}
Shrink: $P_i' = P_{best} + \sigma(P_i - P_{best})$ for $i \neq i_{best}$
Check Stopping Criteria: $\sqrt{\frac{1}{n}\sum(y_i - \bar{y})^2} \leq t$

Motivating Problem: Initialized Simplex

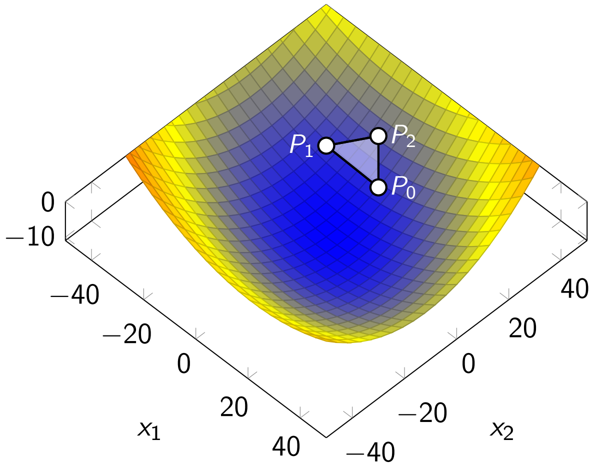


$$P_0 = \begin{bmatrix}-30 & 30\end{bmatrix}$$
$$y_0 = f(P_0) = -1.678$$

$$P_1 = \begin{bmatrix}-20 & 20\end{bmatrix}$$
$$y_1 = f(P_1) = -6.438$$

$$P_2 = \begin{bmatrix}-10 & 30\end{bmatrix}$$
$$y_2 = f(P_2) = -3.718$$

Motivating Problem: Iteration 1

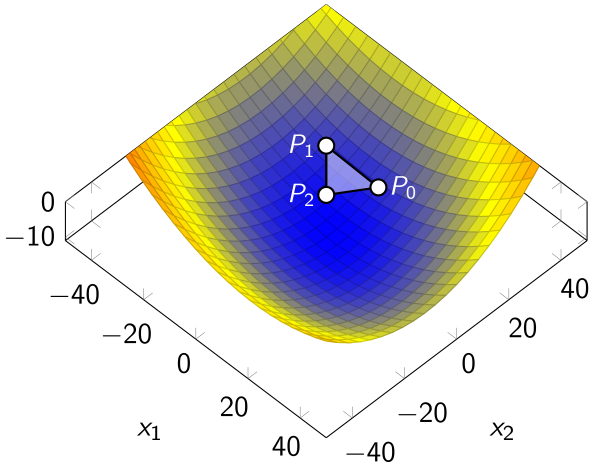


$$P_0 = \begin{bmatrix} 0 & 20 \end{bmatrix}$$
$$y_0 = f(P_0) = -7.078$$

$$P_1 = \begin{bmatrix} -20 & 20 \end{bmatrix}$$
$$y_1 = f(P_1) = -6.438$$

$$P_2 = \begin{bmatrix} -10 & 30 \end{bmatrix}$$
$$y_2 = f(P_2) = -3.718$$

Motivating Problem: Iteration 2

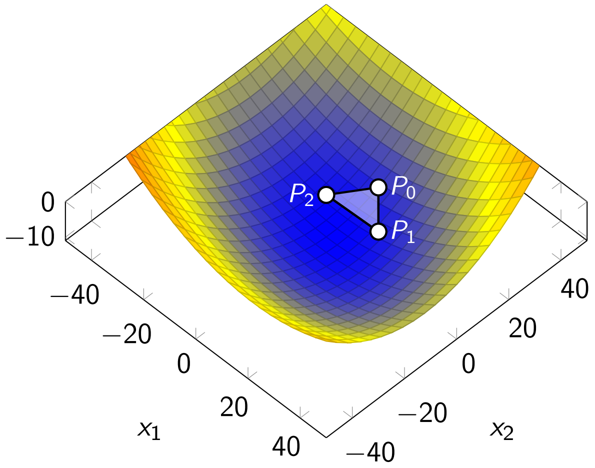


$$P_0 = \begin{bmatrix} 0 & 20 \end{bmatrix}$$
$$y_0 = f(P_0) = -7.078$$

$$P_1 = \begin{bmatrix} -20 & 20 \end{bmatrix}$$
$$y_1 = f(P_1) = -6.438$$

$$P_2 = \begin{bmatrix} -10 & 10 \end{bmatrix}$$
$$y_2 = f(P_2) = -9.198$$

Motivating Problem: Iteration 3

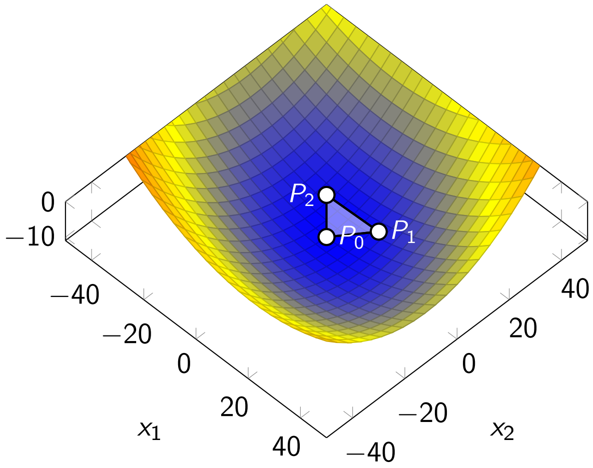


$$P_0 = \begin{bmatrix} 0 & 20 \end{bmatrix}$$
$$y_0 = f(P_0) = -7.078$$

$$P_1 = \begin{bmatrix} 10 & 10 \end{bmatrix}$$
$$y_1 = f(P_1) = -8.438$$

$$P_2 = \begin{bmatrix} -10 & 10 \end{bmatrix}$$
$$y_2 = f(P_2) = -9.198$$

Motivating Problem: Iteration 4

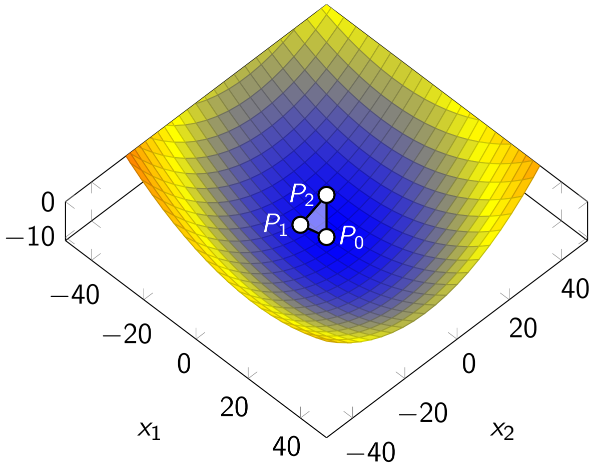


$$P_0 = \begin{bmatrix} 0 & 0 \end{bmatrix}$$
$$y_0 = f(P_0) = -9.958$$

$$P_1 = \begin{bmatrix} 10 & 10 \end{bmatrix}$$
$$y_1 = f(P_1) = -8.438$$

$$P_2 = \begin{bmatrix} -10 & 10 \end{bmatrix}$$
$$y_2 = f(P_2) = -9.198$$

Motivating Problem: Iteration 5

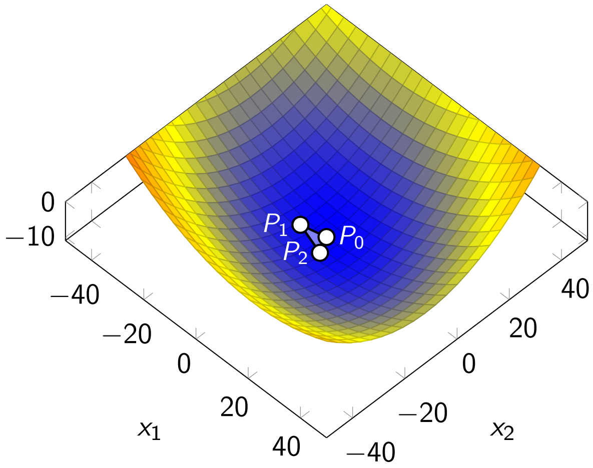


$$P_0 = \begin{bmatrix} 0 & 0 \end{bmatrix}$$
$$y_0 = f(P_0) = -9.958$$

$$P_1 = \begin{bmatrix} -7.5 & -2.5 \end{bmatrix}$$
$$y_1 = f(P_1) = -9.291$$

$$P_2 = \begin{bmatrix} -10 & 10 \end{bmatrix}$$
$$y_2 = f(P_2) = -9.198$$

Motivating Problem: Iteration 6

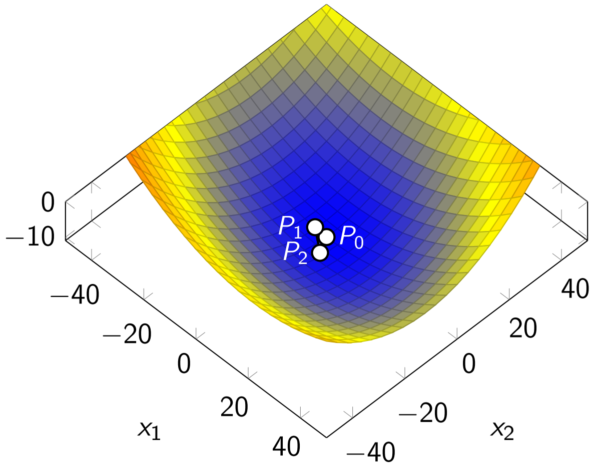


$$P_0 = \begin{bmatrix} 0 & 0 \end{bmatrix}$$
$$y_0 = f(P_0) = -9.958$$

$$P_1 = \begin{bmatrix} -7.5 & -2.5 \end{bmatrix}$$
$$y_1 = f(P_1) = -9.291$$

$$P_2 = \begin{bmatrix} 3.13 & -5.63 \end{bmatrix}$$
$$y_2 = f(P_2) = -9.538$$

Motivating Problem: Iteration 7



$$P_0 = \begin{bmatrix} 0 & 0 \end{bmatrix}$$
$$y_0 = f(P_0) = -9.958$$

$$P_1 = \begin{bmatrix} -4.53 & .156 \end{bmatrix}$$
$$y_1 = f(P_1) = -9.789$$

$$P_2 = \begin{bmatrix} 3.13 & -5.63 \end{bmatrix}$$
$$y_2 = f(P_2) = -9.538$$

Motivating Problem: Convergence History

Iter.	Func Evals	y_{best}	P_{best}	$Error(y_{best})$	$Error(P_{best})$
1	5	-7.078000	$\begin{bmatrix}0 & 20\end{bmatrix}$	2.916188	$\begin{bmatrix}-0.188 & 18.052\end{bmatrix}$
2	7	-9.198000	$\begin{bmatrix}-10 & 10\end{bmatrix}$	0.796188	$\begin{bmatrix}-10.188 & 8.052\end{bmatrix}$
3	8	-9.198000	$\begin{bmatrix}-10 & 10\end{bmatrix}$	0.796188	$\begin{bmatrix}-10.188 & 8.052\end{bmatrix}$
4	10	-9.958000	$\begin{bmatrix}0 & 0\end{bmatrix}$	0.036188	$\begin{bmatrix}-0.188 & -1.948\end{bmatrix}$
5	12	-9.958000	$\begin{bmatrix}0 & 0\end{bmatrix}$	0.036188	$\begin{bmatrix}-0.188 & -1.948\end{bmatrix}$
...	...	...	...	...	...
10	22	-9.994024	$\begin{bmatrix}0.2441 & 2.0605\end{bmatrix}$	0.000164	$\begin{bmatrix}0.056 & 0.113\end{bmatrix}$
...	...	...	...	...	...
15	42	-9.994180	$\begin{bmatrix}0.2288 & 1.9317\end{bmatrix}$	0.000008	$\begin{bmatrix}0.040 & -0.016\end{bmatrix}$
...	...	...	...	...	...
20	59	-9.994188	$\begin{bmatrix}0.1891 & 1.9466\end{bmatrix}$	0.0000005	$\begin{bmatrix}0.001 & -0.001\end{bmatrix}$
...	...	...	...	...	...
25	76	-9.994188	$\begin{bmatrix}0.1885 & 1.9475\end{bmatrix}$	0.0000005	$\begin{bmatrix}0.0001 & -0.0001\end{bmatrix}$

Concluding Remarks

Pros of Nelder-Mead:

- easy to understand and implement programmatically
- no need to compute derivatives (non differentiable functions)
- only need to track $n + 1$ data points
- low amount of function evaluations per iteration (not counting **SHRINK** step)

Cons of Nelder-Mead:

- may sometimes be slow to converge or may fail entirely
- no way to ensure global optimum (not unique to NM)

Concluding Remarks (cont)

This implementation was “unconstrained optimization” without constraints on the design variables or boundaries on the design space. Constraints can be imposed within the objective function with relative ease (see “Penalty Methods”, etc.). That is out of the scope of this document.

It is also possible to impose simple boundary constraints on the design variables within the algorithm itself. Can you think of an easy way to do this?