CHEMICAL ENGINEERING 4G03

Module 10

Nonlinear Programming (III)

Derivative-Free Search Methods

Dr. Jake Nease

Chemical Engineering

McMaster University

Updated April 5, 2023

Outline of Module

Suggested Readings

Note that neither book contains particle swarm optimization. Instead, consider this article.

Rardin (1st edition): Chapter 13.8

Rardin (2nd edition): Chapter 16.8

Introduction and Perspective

In the last section, we looked at several ways of computing search directions Δx and step sizes α for our continuous improving search algorithm using **derivative information**:

Algorithm: Continuous Improving Search

1. **INITIALIZATION**
   Select any feasible starting point $x^{(0)}$ with counter $k = 0$.

2. **MOVE DIRECTION**
   If no improving feasible direction Δx can be found, STOP.
   OTHERWISE, determine an improving feasible direction Δx.

3. **STEP SIZE**
   If there is *no limit* to the improvement when going in the direction Δx while also maintaining feasibility, STOP: Model is *unbounded*.
   OTHERWISE, choose the largest allowable step size α that improves the objective while remaining feasible.

4. **UPDATE**
   $x^{(k+1)} = x^{(k)} + \alpha \Delta x$
   $k = k + 1$
   Return to step (2)

However, there were a few deficiencies of using these methods, such as:

- Derivatives are *hard to compute*, and are usually never computed analytically.
- Since we approximate derivatives, we are at the mercy of *numerical instability* and *propagating precision error* during our iterative search.
- The computational complexity of computing gradients, Hessians, or BFGS deflection matrices gets very high as the size of the system increases.

In this section, instead of derivative-based methods, we are going to try a couple of **derivative-free methods** that instead uses only function evaluations and a few simplifying assumptions. The two methods we will examine in this course are:

1. The Nelder-Mead Derivative-Free Simplex.
2. Particle Swarm Optimization (a meta-heuristic method new to this course).

Each method **SHOULD** give us a search direction Δx and a step size α, although we will find that the particle-swarm technique uses a different approach than the continuous improving search.

Nelder-Mead Derivative-Free Simplex

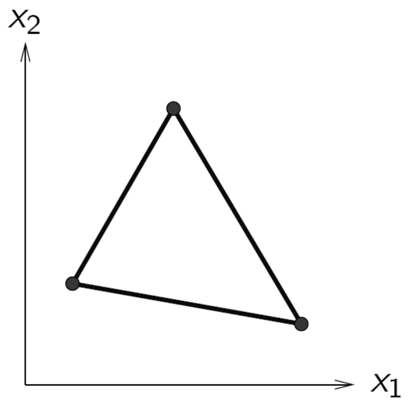
SIMPLEX, you say? Why yes, as a matter of fact! We are going to handle it in a bit of a different form than the LP Simplex Search, however. The basic idea of the Nelder-Mead Simplex (NMS from now on) is:

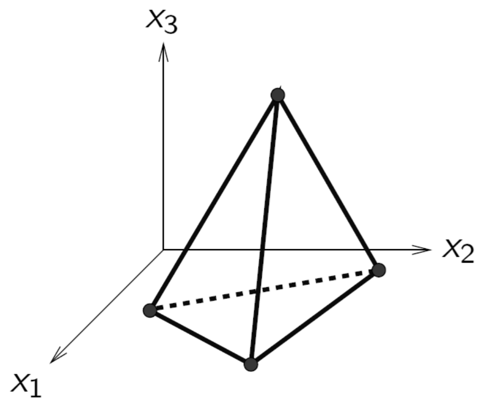
I will consider the objective function value at n+1 carefully chosen points, and select a new point that moves AWAY from the point with the worst objective function value and TOWARDS (and through) the points with highest objective function values.

Definition: Geometric Simplex

A **SIMPLEX** is a geometrical figure formed by $n + 1$ unique points $\{x_1, x_2, \dots, x_{n+1}\}$ in an n-dimensional space. This always works out to be some sort of triangular hyper-space shape, depending on the dimensionality of the system.

Consider for example the figures below. In 2 dimensions, a simplex will always work out to be a triangle. In 3 dimensions, a triangular pyramid. In greater dimensions, the "triangular" nature of the Simplex shape is maintained.





To perform the NMS, the **objective function value** at each point $x^i, i = 1 \dots n + 1$ is evaluated and sorted into a **non-improving sequence** (best to worst). This means:

- For a **maximization**: $\phi(x^1) > \phi(x^2) > \cdots > \phi(x^{n+1})$
- For a **minimization**: $\phi(x^1) < \phi(x^2) < \cdots < \phi(x^{n+1})$

Determining Nelder-Mead Direction

The Nelder-Mead Simplex always assumes an **away-from worst** approach. This means that we always travel *away from the worst point* $(n + 1)$ *in the direction of the* ***centroid*** *of all remaining points*:

Away-From-Worst Direction

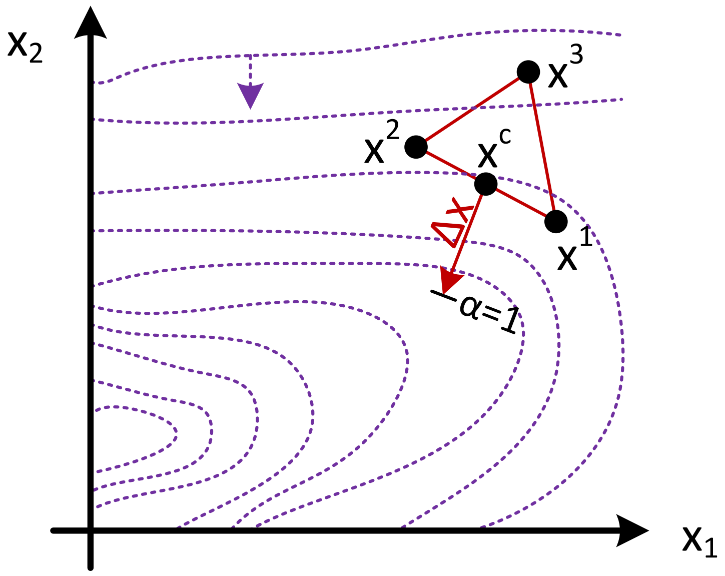
For an **ordered** set of points, the away-from-worst direction Δx is defined to be the vector that passes from the worst point x^{n+1} and through the best-n centroid x^c.

$$\boxed{\boldsymbol{x^c \triangleq \frac{1}{n}\sum_{i=1}^{n} x^i}}$$

$$\boxed{\boldsymbol{\Delta x \triangleq x^c - x^{n+1}}}$$

$$\boxed{\boldsymbol{x^{k+1} = x^c + \alpha \Delta x}}$$

This may be visualized in the two-dimensional space using a triangle and $n + 1 = 3$ points as shown in the figure below.



Some important procedural things to note:

- Since the points and their objective function values are **ordered,** $\boldsymbol{x^{n+1}}$ should ALWAYS be the point with the "worst" objective function value.
- The idea of a best-n centroid carries through to more dimensions than just two. Inspection of the 3D simplex shape on the previous page will show that x^c is the geometric center of any face that is opposite the worst point.

Determining Nelder-Mead Step Sizes

The Nelder-Mead step size computations are actually very straightforward. They are designed around the fact that we wish to obtain a new point that is *at least better than* the previous x^{n+1} (the previous worst point). This way, the *average* value of our simplex is **constantly improving**.

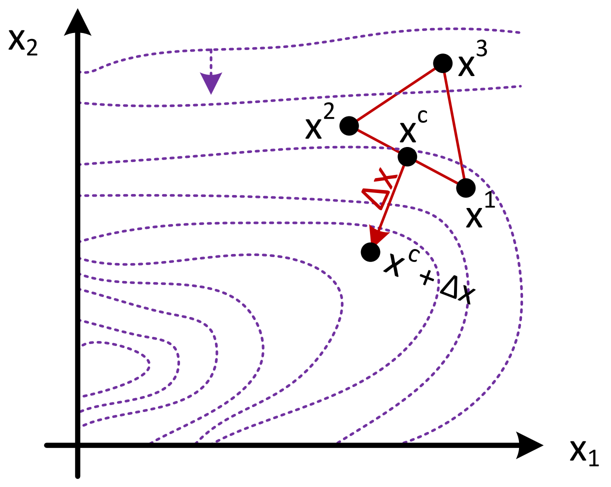
Nelder-Mead Step Size

When determining the step size α for the Nelder-Mead Simplex, use the following rules:

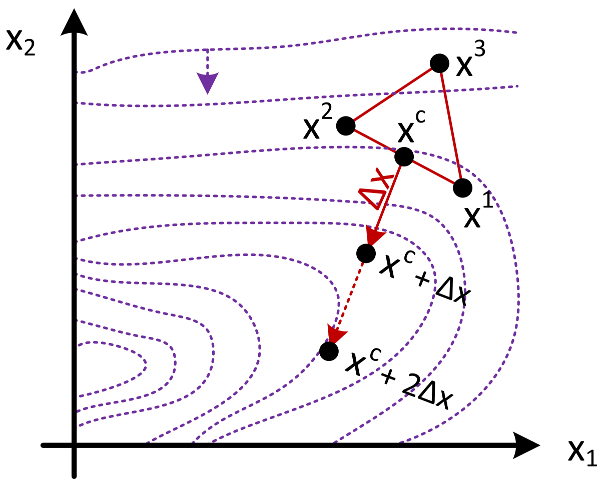
- The Nelder –Mead procedure first **reflects about the centroid** $\boldsymbol{x^c}$ with $\boldsymbol{\alpha = 1}$.
- If $x^c + \Delta x$ is neither *better* than x^1 or *worse* than x^n, **keep** $\boldsymbol{\alpha = 1}$ **with no further trials**.
- If $x^c + \Delta x$ is **a NEW BEST** (as in, will be the new x^1):
    - Attempt an *expansion* with $\alpha = 2$.
    - Accept $\alpha = 2$ if $x^c + 2\Delta x$ is **better than** $x^c + \Delta x$.
- If $x^c + \Delta x$ is **worse than** $\boldsymbol{x^n}$:
    - Attempt a *contraction*: $\alpha = \frac{1}{2}$ IF $\phi(x^c + \Delta x)$ is better than $\phi(x^{n+1})$.
    - Attempt an *inward contraction*: $\alpha = -\frac{1}{2}$ otherwise.
    - Accept a contraction $\alpha = \pm\frac{1}{2}$ **only if you get a value better than** $\boldsymbol{x^n}$.

NB – You may find none of these conditions are satisfied and must *shrink*. This is covered next.

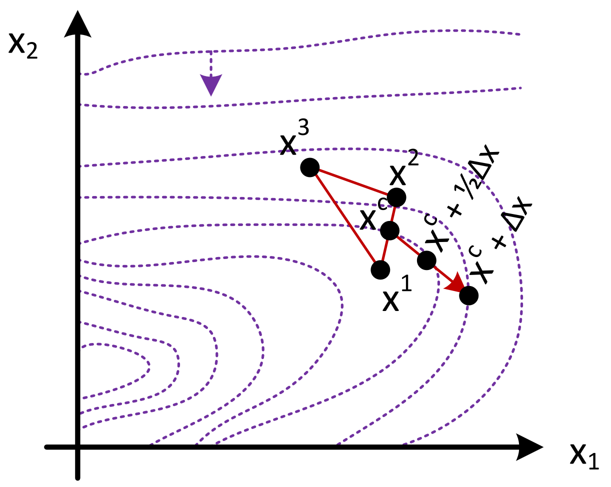
These different step sizes are shown in the following sets of figures.



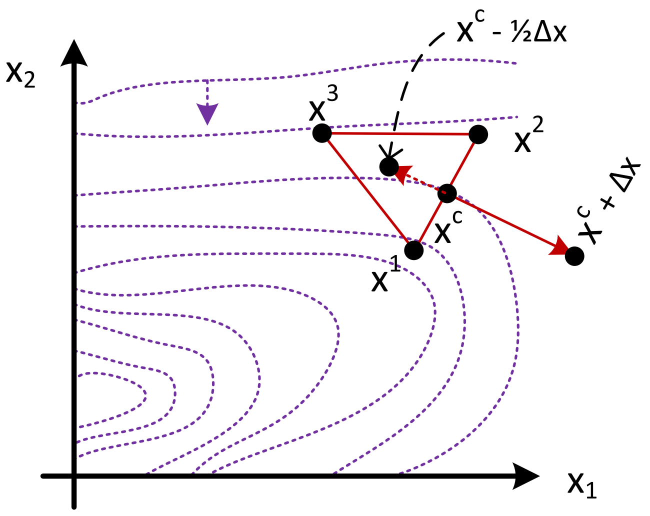
Original



Exapansion



Contraction



Inward Contraction

In the event that **no possible** step size α yields an improved point over x^n, we *shrink* the simplex shape toward the current best point by keeping the best point the same and moving all others toward it.

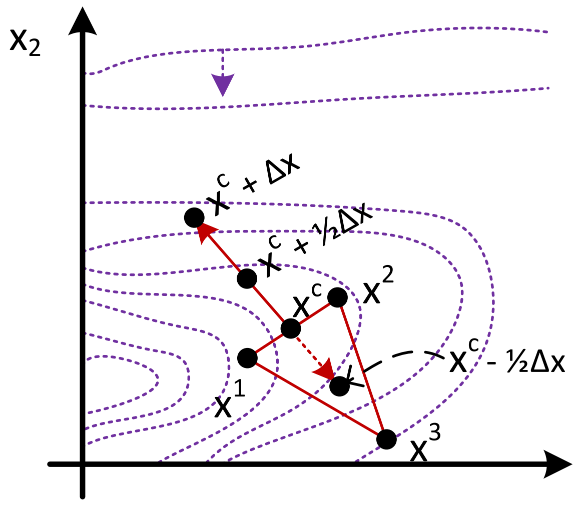
Nelder-Mead Simplex Shrinking

If no step sizes α yield an improvement over x^n, the procedure **shrinks** the entire array of simplex vertices **toward** the current best (x^1):

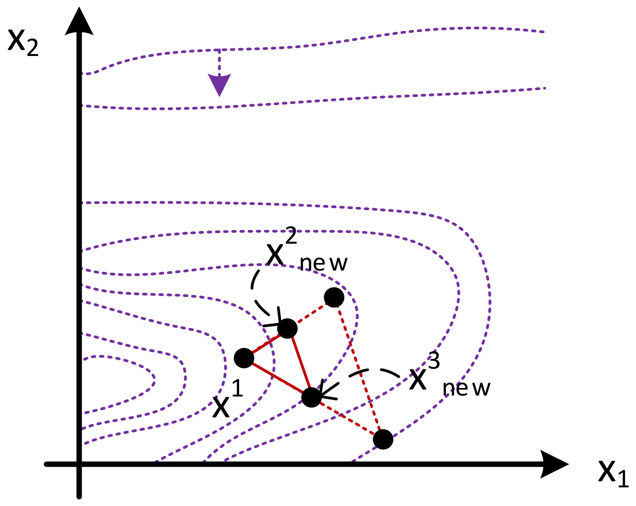
$$\boldsymbol{x^i_{new} \triangleq \frac{1}{2}\left[x^1 + x^i\right] \quad \forall \ i = 1, \dots, n+1}$$

NB – More computationally efficient to ignore $i = 1$, but does not make a difference to the algorithm (why?).

This procedure is illustrated in the figures below.



No Improvement



Shrink Simplex

Algorithm Statement: Nelder-Mead Simplex Search

1. **Initialization**
   Choose $n+1$ distinct points $\{x^1, \dots, x^{n+1}\}$, and evaluate $\phi(x^1), \dots, \phi(x^{n+1})$.
   Set stopping tolerance $\epsilon > 0$.

2. **Move Direction**
   Re-arrange x^i in a **non-improving sequence**.
   Compute best-n centroid as $x^c \triangleq \frac{1}{n} \sum_{i=1}^{n} x^i$
   If $\left|\phi(x^i) - \phi(x^c)\right| < \epsilon$: **STOP** → Report the best of current x^c or x^1 as the approximate solution.
   Else, compute away-from-worst direction as $\Delta x \triangleq x^c - x^{n+1}$.

3. **Step Size**
   IF Δx is shown to improve on current best $\phi(x^1)$:
      Attempt **expansion** with $\alpha = 2$. Use $\alpha = 2$ if further improvement is obtained, and use $\alpha = 1$ otherwise. Proceed to step 4.
   IF Δx does not improve on second-worst x^n:
      Contract by trying $\alpha = \frac{1}{2}$ if $\phi(x^c + \Delta x)$ is better than $\phi(x^{n+1})$, or try $\alpha = -\frac{1}{2}$ otherwise. If improvement is obtained over x^n, set $\alpha = \pm\frac{1}{2}$ and proceed to step 4.

4. **Update**
   IF contraction is non-improving:
      Shrink current simplex by setting $\boldsymbol{x_{new}^i \triangleq \frac{1}{2}\left[x^1 + x^i\right] \quad \forall \; i = 1, \dots, n+1}$
      Compute new function values $\phi(x^i)$ and return to step 2.
   OTHERWISE, replace x^{n+1} in the set of current points by $x^c + \alpha \Delta x$ and return to step 2.

Not too shabby! This is actually a lot of fun, and can be done by hand! Let's get some practice!

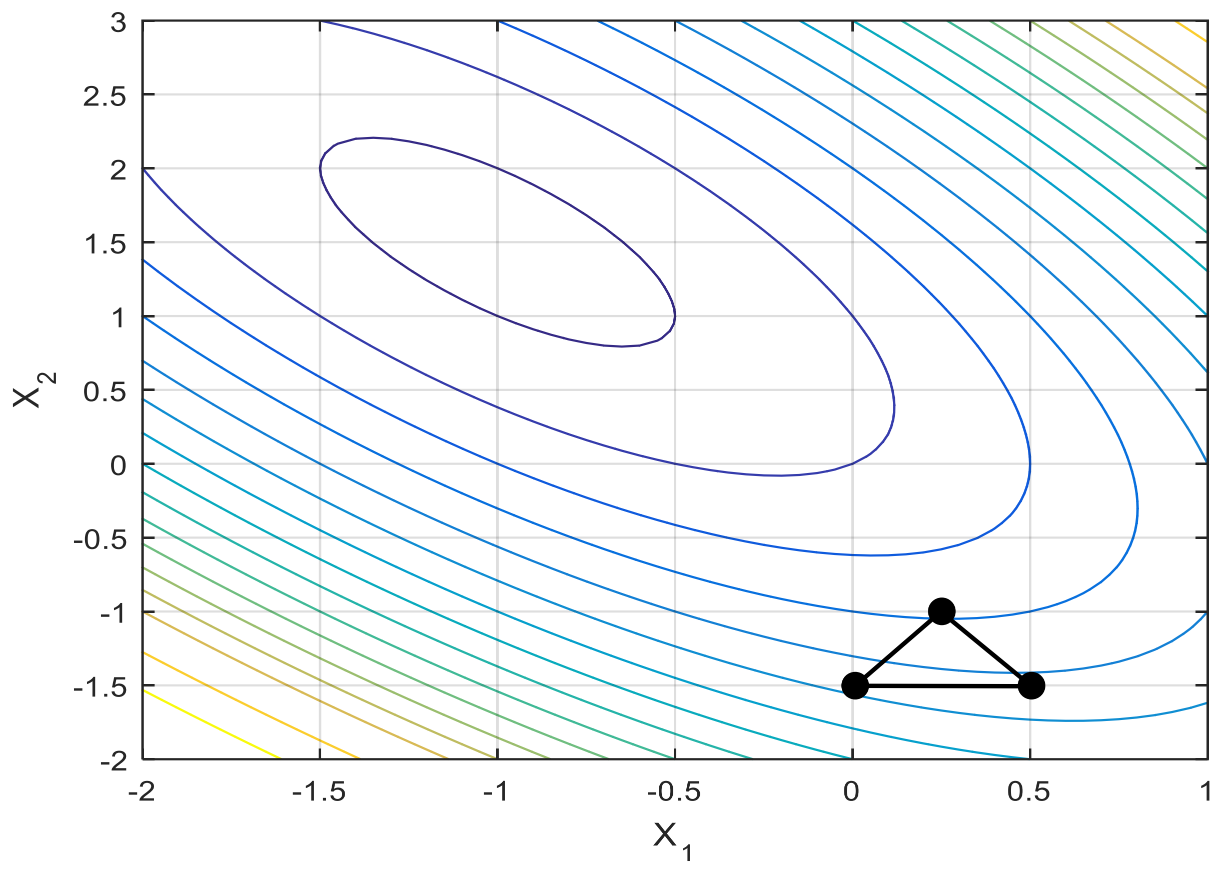
Class Workshop – Nelder-Mead (I)

Consider the unconstrained NLP:

$$\min_{x} \phi(x) = x_1 - x_2 + 2x_1^2 + 2x_1x_2 + x_2^2$$

Starting from the initial points given in the figure below, sketch a few iterations of the Nelder-Mead procedure *graphically*.

Workshop Solution – Nelder-Mead (I)

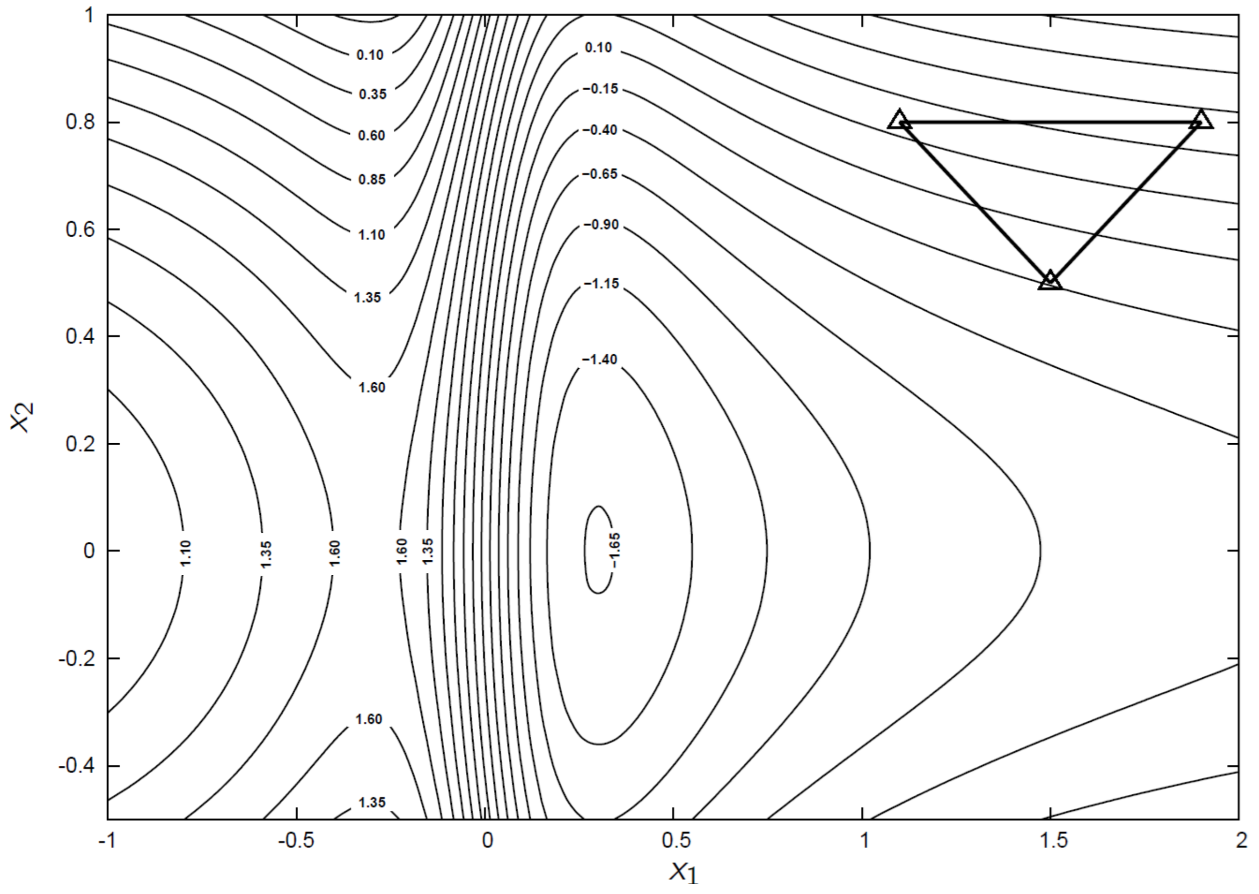


And, of course, there is nothing stopping us from using MUCH more interesting functions than the one in the example above!

Class Workshop – Nelder-Mead (II)

Consider the (much more interesting) unconstrained NLP on the following page. Starting from the initial points given, sketch a few iterations of the Nelder-Mead procedure *graphically*.

Workshop Solution – Nelder-Mead (II)



Meta-Heuristic Method: Particle Swarm Optimization

Meta-Heuristic Methods: A BRIEF Intro

Meta-Heuristic methods, such as particle swarm optimization (PSO), attempt to emulate how many natural occurrences work, such as how birds find the best place to land, how bees locate flowers with the highest pollen content, how generations of people or animals evolve to improve survivability, or even how metals are annealed and strengthened after being exposed to repeated stressing conditions. Such algorithms might include:

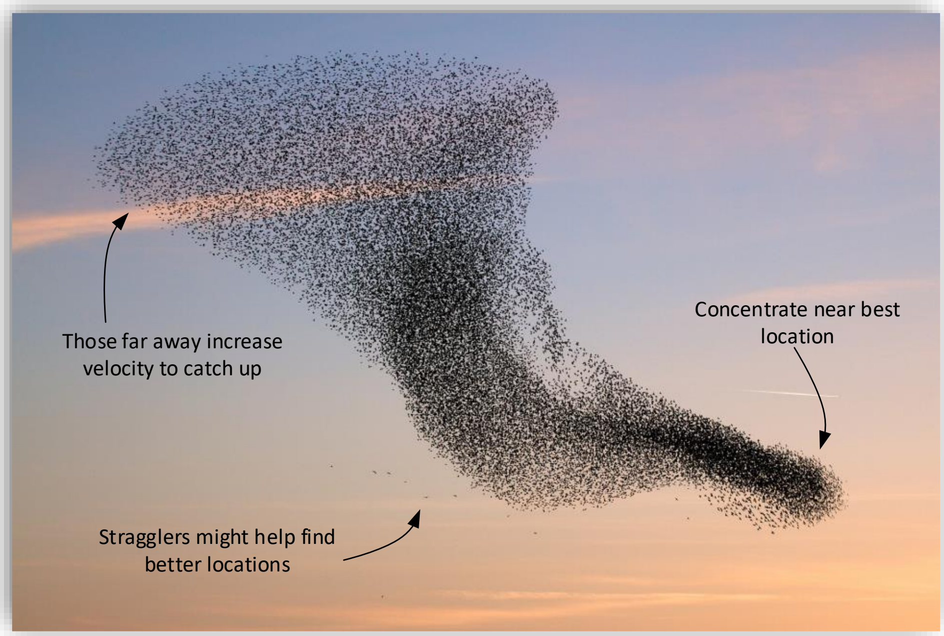
- Particle Swarm Optimization.
- Simulated Annealing.
- Differential Evolution.
- Genetic Algorithms.
- Ant Colony Optimization.

These are **heuristical stochastic programs**. As a result they are sometimes frowned upon by "classical" optimization experts because the *laws governing optimality*, such as the 1st and 2nd order NCO and SCO and the KKT conditions of constrained optimality, *do not apply*. However, we know these do a **really good job** as long as you are not looking to "prove global optimality" (but who can even really do that, anyway?).

PSO: The Idea

The idea behind PSO is actually quite simple. We distribute a series of particles into our search space $x_1, \ldots, x_N$. Each particle has a **memory** of where it has been (with regards to the objective function) and all particles **communicate** with each other so they know which among them has seen the best location so far. Thus, each particle might be a **bird** or **bee** searching for something. They have:

- **MOMENTUM** – each particle is flying in a certain direction with some velocity.
- **MEMORY** – each particle "remembers" where it has been before (just the best place visited).
- **SOCIAL NETWORK** – each particle can communicate with the others.
- **RANDOMNESS** – Each particle has a mind of its own and thus will (partially) take random paths.


Those far away increase velocity to catch up
Concentrate near best location
Stragglers might help find better locations

Useful Scenarios for PSO

PSO works very well in scenarios where derivative information is either unavailable or difficult to calculate. It also works very well for non-convex regions with "holes" and "nooks", because particles will be able to avoid them or find them if they are penalized properly in the objective function. Moreover, PSO, along with most meta-heuristic methods, are **embarrassingly parallelizable**, meaning that we can run each particle's calculations in parallel with each other, thus GREATLY reducing computation time. PSO is thus quite handy for the following scenarios:

- Plant design, where each "function evaluation" is actually running and converging an Aspen Plus simulation.
- Massive (in terms of search space) objective functions with *many* known local minima (in an effort to finding a global optimum).
- Can be used to find a *good initial guess* for a classical deterministic method such as Newton's Method or Quasi-Newton's Method.

We might as well get right into the algorithm since I think it is easiest to explain as we go!

Algorithm Statement: Particle Swarm Optimization

1. **Initialization**
   Choose N distinct points $\{x_1^0, \dots, x_N^0\}$. I choose $N = 20 \rightarrow 30$ for simplicity (up to you though).
   Assign some maximum velocity $||v_i|| \leq v_M$.
   Assign each point an initial velocity in each dimension $\{v_1^0, \dots, v_N^0\}$.
   Assign one point to be the current incumbent solution x_{inc} and $\phi_{inc} = \phi(x_{inc})$.
   Set stopping tolerance for **positions** $\epsilon > 0$ (many options here).
   Set the iterate counter $k = 0$.

2. **Move Particles**
   Update each particle's position by adding the velocity: $x_i^{k+1} = x_i^k + v_i^k$.
   Evaluate each particle's objective function value $\phi(x_i^{k+1})$.
   IF any of $\phi(x_i^{k+1})$ is better than ϕ_{inc}
      Assign $\phi_{inc} = \phi(x_i^{k+1})$.
      Assign $x_{inc} = x_i^{k+1}$ that has the best objective function value.
   IF for any particle $\phi(x_i^{k+1})$ is better than $\phi(x_i^{k<k+1})$, update its personal best:
      Assign $x_i^{PB} = x_i^{k+1}$

3. **Stopping and Velocity Changes**
   UPDATE $v_i^{k+1} = \omega v_i^k + w_1\beta_i(x_i^{PB} - x_i^k) + w_2\beta_2(x_{inc} - x_i^k)$
   Use weight factors $\omega = 0.9$, $w_1 = 2.8$, $w_2 = 1.3$ and $\beta = rand(0,1)$

   IF $\boldsymbol{||x^{k+1} - x^k|| < \epsilon \; \forall \; i}$, **STOP** and report ϕ_{inc} **as the approximate solution**.

4. **Update Iterate**
   Update $k = k + 1$ and return to step 2.
   {Can also perform any other modifications to routine here}

Time for an in-class demo!

Conclusions

Hopefully this has been an interesting introduction to derivative-free optimization methods. It is certainly true that each method has pros and cons (including derivative-based methods). The main point, however, is that there is **no one true way** of finding the global optimum. It will be up to you as the clever Engineer to use the method most appropriate to the job and to *interpret the results* to ensure that what you have done makes sense. Thanks for being such a great class, and congratulations on what i hope was another successful year (with many more to come in your careers)!

~~ END OF MODULE ~~

\|T|/