Probabilistic Algorithms

Paul Cotofrei

information management institute

PA 2016

Outline

Introduction

- Direct random search for minimizing $L(\mathbf{d})$ subject to parameter vector $\mathbf{d}$ lying in some set $\mathbf{D}$
- Information required:
  - Input - $\mathbf{d}$; Output - $L(\mathbf{d})$ (noise free)
  - no requirements about the gradient of L
  - no requirement about local/global minima
  - *Zero-th order methods*
- Stopping criteria
  - the "budget" criterion : stops after N evaluations of $L(\mathbf{d})$
  - the "stable" criteria (at least one must hold at iteration $n \geq N$, for all $1 \leq j \leq N$):

$$|L(\hat{\mathbf{d}}_n) - L(\hat{\mathbf{d}}_{n-j})| \leq \eta$$

$$\|\hat{\mathbf{d}}_n - \hat{\mathbf{d}}_{n-j}\| \leq \eta$$

  where $\eta > 0$ fixed
- No one guarantees that $\hat{\mathbf{d}}_n$ close to $\mathbf{d}^*$!

Some Attributes of Direct Random Search

- Easy to program
- Works for:
  - non-convex, non-differentiable $L()$ over a continuous, discrete, or mixed continuous-discrete domain
  - non-numeric L (the only constraint: each pair of solutions can be compared)
- Reasonable computational efficiency (if the problem dimension $p = dim(\mathbf{d})$ is not too large)
- Provide a means of finding "good" initial conditions for more sophisticated algorithms
- Generality: algorithms apply to virtually any function
- Theoretical foundation
  - Performance guarantees, sometimes in finite samples
  - Polynomial time if the goal is a solution with a high probability to be optimal
  - Global convergence in some cases

Outline

Classification of Random Search Methods

- Classification:
  - Random Jump: generates huge number of data points assuming uniform distribution of decision variables and selects the best one
  - Random Walk: generates trial solution with sequential improvements using scalar step length and unit random vector.
  - Random Walk with Direction Exploitation: Improved version of random walk search, successful direction of generating trial solution is found out and steps are taken along this direction

Algorithm A: Simple Random ("Blind") Search

Step 0. (Initialization) Choose an initial value of $\mathbf{d}$, $\hat{\mathbf{d}}_0 \in \mathbf{D}$. Calculate $L(\hat{\mathbf{d}}_0)$. Set $k = 0$.

Step 1. (Candidate value) Generate a new independent value $\mathbf{d}_{new} \in \mathbf{D}$, according to the chosen probability distribution. If $L(\mathbf{d}_{new}) < L(\hat{\mathbf{d}}_k)$, set $\hat{\mathbf{d}}_{k+1} = \mathbf{d}_{new}$. Else take $\hat{\mathbf{d}}_{k+1} = \hat{\mathbf{d}}_k$.

Step 2. (Return or Stop) Stop if maximum number of L evaluations has been reached or user is satisfied with the current estimate for $\mathbf{d}$; else, $k = k + 1$ and return to Step 1.

- Batch implementation: generate N candidate values $\hat{\mathbf{d}}_i \in \mathbf{D}$ and takes $Min_i\{L(\hat{\mathbf{d}}_i)|i = 1...N\}$
- **Random sampling**: easy for **D** a hypercube (a p-fold cartesian product of intervals on the real line). If **D** irregular shape, generate from a hypercube superset containing **D** and reject sample points outside **D**.

Random walk algorithms

- For algorithm A, the sampling does not take account of where the previous estimates of **d** have been ("blind" search).
- Ameliorate the search by random sampling depending of the position of the current best estimate for **d**.
  - **Localized algorithms**
  - the search is more localized in the neighborhood of that estimate
  - allows for a better exploitation of information about L

Algorithm B: Localized Random Search

Step 0. (Initialization) Choose an initial value of $\mathbf{d} = \hat{\mathbf{d}}_0 \in \mathbf{D}$ (randomly or with prior information). Set $k = 0$.

Step 1. (Candidate value) Generate a random vector $\mathbf{v}_k$. Check if $\hat{\mathbf{d}}_k + \mathbf{v}_k \in \mathbf{D}$. If not, generate a new $\mathbf{v}_k$ or move $\hat{\mathbf{d}}_k + \mathbf{v}_k$ to nearest valid point in $\mathbf{D}$. Let $\mathbf{d}_{new} \in \mathbf{D}$ be $\hat{\mathbf{d}}_k + \mathbf{v}_k$ or the modified point.

Step 2. (Check for improvement) If $L(\mathbf{d}_{new}) < L(\hat{\mathbf{d}}_k)$ set $\hat{\mathbf{d}}_{k+1} = \mathbf{d}_{new}$. Else take $\hat{\mathbf{d}}_{k+1} = \hat{\mathbf{d}}_k$.

Step 3. (Return or Stop) Stop if maximum number of L evaluations has been reached or user satisfied with current estimate; else, set $k = k + 1$ and return to Step 1.

Generating deviation vector $\mathbf{v}_k$

- Standard distribution: multivariate normal distribution $N(0, \rho^2 I_p)$
- General distribution:
  - should have mean zero
  - each component should have a variation consistent with the magnitude of the corresponding **d** elements
  - example: $\mathbf{d} = [d_1, d_2]$, where $d_1 \in [0, 0.05]$ and $d_2 \in [0, 500]$; the distribution for generating $\mathbf{v}_k$ may be

$$N\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \rho^2 & 0 \\ 0 & 100^2\rho^2 \end{pmatrix}\right)$$

- In practice, the variability in $\mathbf{v}_k$ is reduced as k increase
  - focus the search more tightly on the location of the solution

Algorithm C: Enhanced Localized Random Search

- Similar to algorithm B
- Exploits knowledge of good/bad directions
- If move in one direction produces decrease in loss, add bias to next iteration to continue algorithm moving in "good" direction
- If move in one direction produces increase in loss, add bias to next iteration to move algorithm in opposite way
- Slightly more complex implementation than algorithm B

Algorithm C: Enhanced Localized Random Search

Step 0. (Initialization) Choose an initial value of $\mathbf{d} = \hat{\mathbf{d}}_0 \in \mathbf{D}$. Set $k = 0$. Set bias vector $\mathbf{b}_0 = 0$

Step 1. Generate a random vector $\mathbf{v}_k$. Check if $\hat{\mathbf{d}}_k + \mathbf{b}_k + \mathbf{v}_k \in \mathbf{D}$. If not, generate a new $\mathbf{v}_k$ or move $\hat{\mathbf{d}}_k + \mathbf{b}_k + \mathbf{v}_k$ to nearest valid point in $\mathbf{D}$. Let $\mathbf{d}_{new}$ be $\hat{\mathbf{d}}_k + \mathbf{b}_k + \mathbf{v}_k$ or the modified point.

Step 2. If $L(\mathbf{d}_{new}) < L(\hat{\mathbf{d}}_k)$ then set $\hat{\mathbf{d}}_{k+1} = \mathbf{d}_{new}$, $\mathbf{b}_{k+1} = 0.2\mathbf{b}_k + 0.4\mathbf{v}_k$ and go to Step 5. Else go to Step 3.

Step 3. Let $\mathbf{d}'_{new} = \hat{\mathbf{d}}_k + \mathbf{b}_k - \mathbf{v}_k \in \mathbf{D}$ or the nearest valid point in $\mathbf{D}$. If $L(\mathbf{d}'_{new}) < L(\hat{\mathbf{d}}_k)$ then set $\hat{\mathbf{d}}_{k+1} = \mathbf{d}'_{new}$, $\mathbf{b}_{k+1} = \mathbf{b}_k - 0.4\mathbf{v}_k$ and go to Step 5. Else go to Step 4.

Step 4. Set $\hat{\mathbf{d}}_{k+1} = \hat{\mathbf{d}}_k$ and $\mathbf{b}_{k+1} = 0.5\mathbf{b}_k$. Go to Step 5.

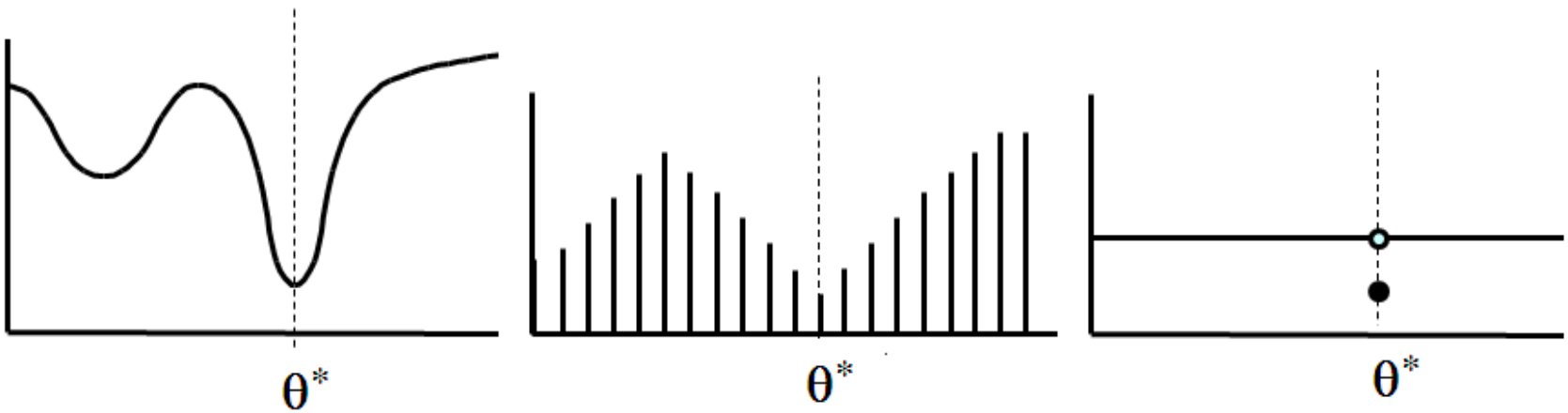
Step 5. Stop if max. number of L evaluations has been reached or user satisfied with current estimate; else set $k = k + 1$ and go to Step 1.

Formal Convergence of Random Search Algorithms

- Well-known results on convergence of random search
  - Applies to convergence of **d** and/or L values
  - Applies when **noise-free** L measurements used in algorithms
- Algorithm A (blind random search) converges under very general conditions
  - Applies to continuous or discrete functions
- Conditions for convergence of algorithms B and C somewhat more restrictive, but still quite general
- Convergence rate theory also exists: how fast to converge?
  - Algorithm A generally slow in high-dimensional problems

Convergence/Nonconvergence of Blind Random Search

- Convergence condition: $P(\mathbf{d}_{new} : L(\mathbf{d}_{new}) < L(\mathbf{d}^*) + \delta) > 0$ (the probability to select a better solution than $\mathbf{d} \neq \mathbf{d}^*$ is positive)
  a) for any $d \neq d^*$, $\exists [d_1, d_2]$ such that $\forall d_{new} \in [d_1, d_2]$, $L(d_{new}) < L(d)$
  b) if $d \neq d^*$, the set $S = \{d_{new} | L(d_{new}) < L(d)\}$ contains at least $d_i = d^*$, and $P(S) > 0$
  c) $S = \{d^*\}$, but $P(S) = 0$



a) Continuous L(θ) b) Discrete L(θ) c) Non-continuous L(θ)

Outline

Compass search

- Consider $L(\mathbf{d}), \mathbf{d} \in R^n$
- Let be the set of directions $E = \{\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_n, -\mathbf{e}_1, ..., -\mathbf{e}_n\}$ (denoted *coordinate directions*), where $\mathbf{e}_i$ is the unit coordinate vector (all coordinates are zero, except on i position, which is 1)
- Let α_k denote the *step-length control parameter* that controls the lengths of the steps taken during k iteration.
  - α_{err} - the tolerance used for convergence test;
  - $\alpha_0 > \alpha_{err}$ - the initial value of the step-length control parameter.

For each iteration $k = 0, 1, ...$

Step 0. If $\alpha_k \leq \alpha_{err}$ then STOP;

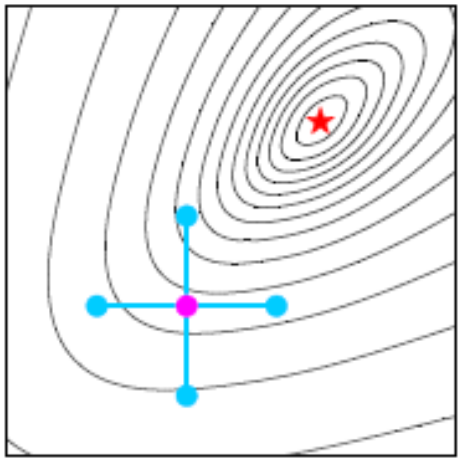
Step 1. If there exists $\mathbf{e}_i \in E$ such that $L(\hat{\mathbf{d}}_k + \alpha_k \mathbf{e}_i) < L(\hat{\mathbf{d}}_k)$, then do:

- $\hat{\mathbf{d}}_{k+1} = \hat{\mathbf{d}}_k + \alpha_k \mathbf{e}_i$ (*change iterate*)
- $\alpha_{k+1} = \alpha_k$ (*keep step-length control param.*)

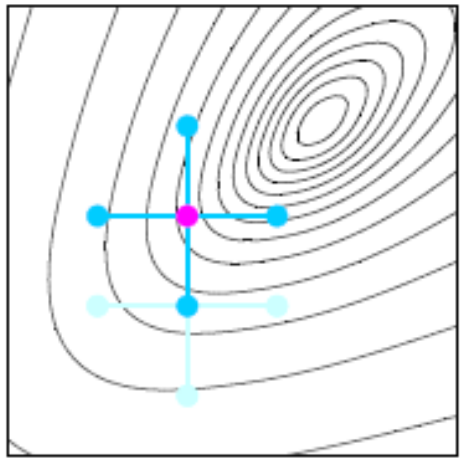
Step 2. Else do:

- $\hat{\mathbf{d}}_{k+1} = \hat{\mathbf{d}}_k$ (*keep iterate*)
- $\alpha_{k+1} = \frac{1}{2}\alpha_k$ (*contract step-length control param.*)

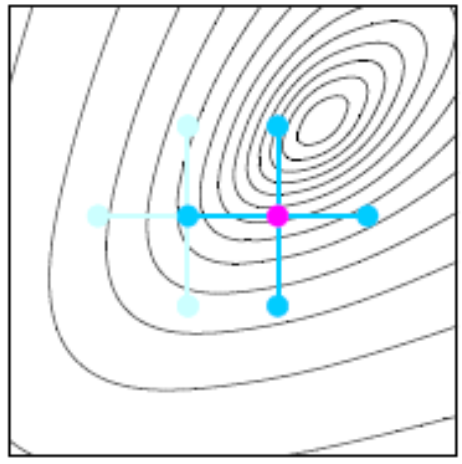
Compass search in 2D



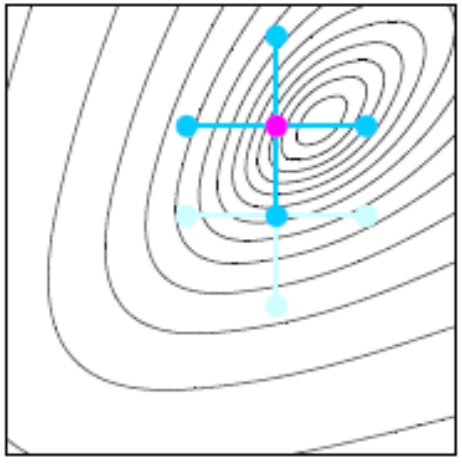
(a) Initial pattern



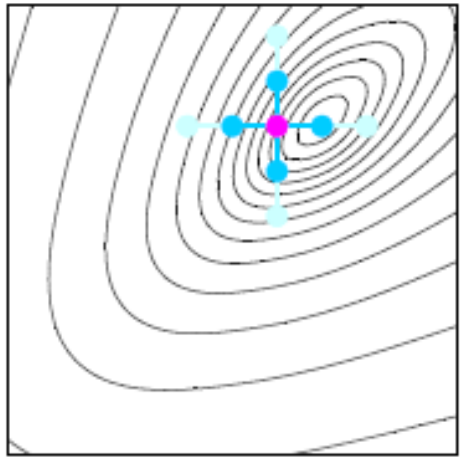
(b) Move North



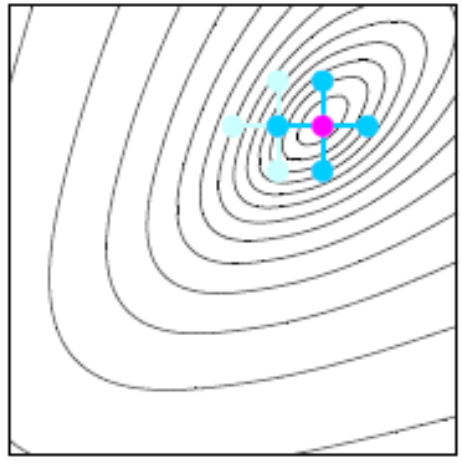
(c) Move West



(d) Move North



(e) Contract



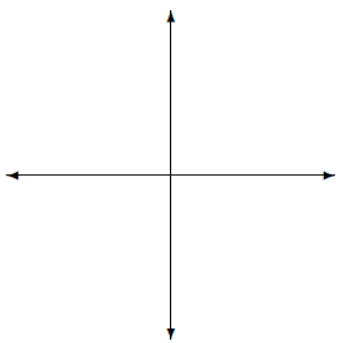
(f) Move West

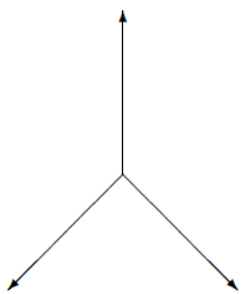
Neighbors evaluation

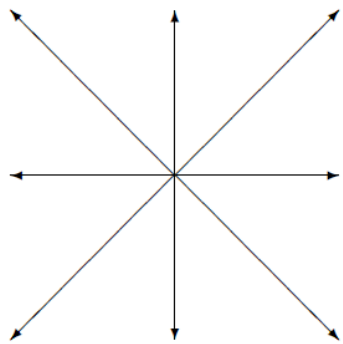
- How one does the evaluations to determine if there exists a $\mathbf{e}_i \in E$ satisfying the simple decrease condition?
  1. Select a deterministic order for the elements of E (as $\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_{2n}$), or
  2. select a random order or dynamic order (step k dependent).
  3. Stop at the first neighbor satisfying the decrease condition, or
  4. Evaluates all $2n$ neighbors and choose those that yields the greatest decrease in L.
- The compass algorithm use only exploratory moves !
  - Theorem. If $\mathbf{d} \in R^n$ and $\nabla L(\mathbf{d})$ (the gradient of L in $\mathbf{d}$) is not null, then at least one of the coordinate directions must be a descent direction.

Positive spanning sets

- Each iteration in "compass search" algorithm uses directions from the set of coordinates directions E
- Consider a general set $\mathcal{E} = \{\mathbf{v}_1, \ldots, \mathbf{v}_p\}$, with $p \geq n + 1$
- $\mathcal{E}$ is a positive spanning set if, *for any vector* $\mathbf{x} \in R^n$, *there exist* $\lambda_i \geq 0, i = 1..p$ *such that* $\mathbf{x} = \sum_{i=1}^{p} \lambda_i \mathbf{v}_i$.
  - i.e., non-negative linear combinations of the elements of $\mathcal{E}$ span R^n
- Examples:







Principle of pattern search

- *Perform a sequence of exploratory moves around a base estimator $\hat{\mathbf{d}}_k$; if successful, perform a pattern move.*
- Exploratory moves - acquire information about the function $L(\mathbf{d})$ in the neighbourhood of the current base estimator $\hat{\mathbf{d}}_k$
  - a fixed number of directions $\mathbf{v}_i$ generates a fixed number of neighbors $\{\hat{\mathbf{d}}_k + \alpha_k \mathbf{v}_i, i = 1..m\}$ (α_k is the step size).
  - if for any $i = 1..m$ we have $L(\hat{\mathbf{d}}_k + \alpha_k \mathbf{v}_i) > L(\hat{\mathbf{d}}_k)$ then $\hat{\mathbf{d}}_{k+1} = \hat{\mathbf{d}}_k$ and $\alpha_{k+1} = \lambda \alpha_k$ ($\lambda < 1$).
  - if exist $j \in \{1..m\}$ such that $L(\hat{\mathbf{d}}_k + \alpha_k \mathbf{v}_j) < L(\hat{\mathbf{d}}_k)$ then move toward $\tilde{\theta}_k = \hat{\mathbf{d}}_k + \alpha_k \mathbf{v}_j$.
- Pattern move - attempts to speed up the search by using the information already acquired about $L(\mathbf{d})$
  - the new base estimator $\tilde{\mathbf{d}}_{k+1} = \hat{\mathbf{d}}_k + \beta_k \mathbf{v}_j$, with $\beta_k > \alpha_k$
  - perform exploratory moves around $\tilde{\mathbf{d}}_{k+1}$ using step-length $\alpha_{k+1} = \beta_k$; if a better solution is found, then this solution becomes $\hat{\mathbf{d}}_{k+1}$; if not, $\hat{\mathbf{d}}_{k+1} = \tilde{\theta}_k$

Basic Pattern Search Algorithm

Init Set $\mathcal{E} = \{\mathbf{v}_1, \ldots, \mathbf{v}_p\}$ a positive spanning set, $\hat{\mathbf{d}}_0$ the initial approximation, α_0 the initial step-length parameter, α_{err} the tolerance parameter, $\gamma > 1$ the increasing constant and $\lambda < 1$ the decreasing constant.

Repeat

Step1 If exists $\mathbf{v}_j \in \mathcal{E}$ such that $L(\hat{\mathbf{d}}_k + \alpha_k \mathbf{v}_j) < L(\hat{\mathbf{d}}_k)$ then do:

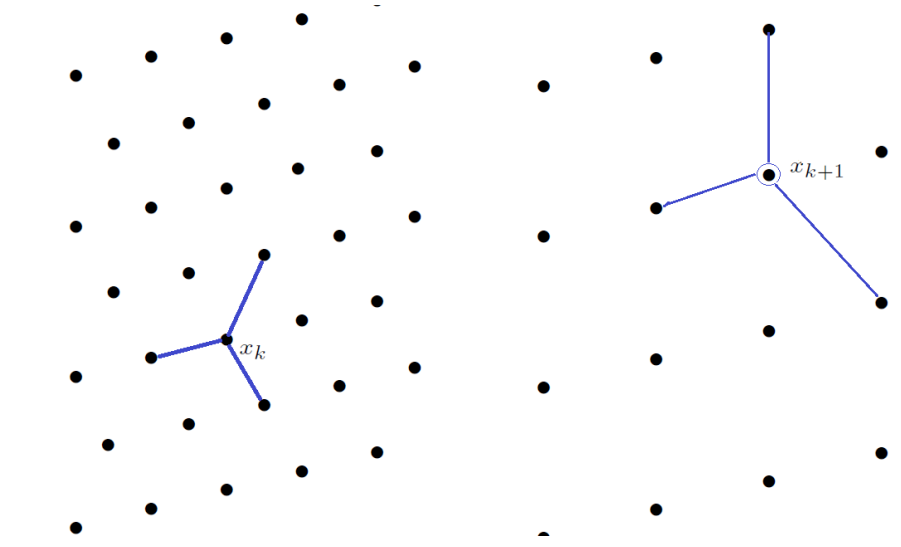
$$\hat{\mathbf{d}}_{k+1} = \hat{\mathbf{d}}_k + \alpha_k \mathbf{v}_j$$
$$\alpha_{k+1} = \gamma \alpha_k$$

Step2 If not,

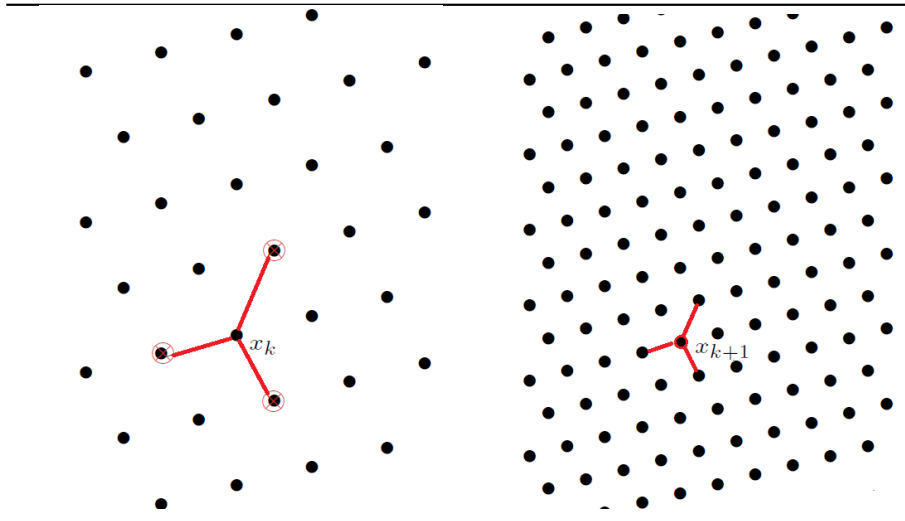
$$\hat{\mathbf{d}}_{k+1} = \hat{\mathbf{d}}_k$$
$$\alpha_{k+1} = \lambda \alpha_k$$

While $\alpha_{k+1} > \alpha_{err}$

Example



Step 1 succeed



Step 1 fails

Nonlinear Simplex (Nelder-Mead) Algorithm

- Nonlinear simplex method is a popular search method
- Simplex is convex hull of p + 1 points in $\mathcal{R}^p$
  - Convex hull is smallest convex set enclosing the $p + 1$ points
  - For $p = 2$ convex hull is a triangle
  - For $p = 3$ convex hull is a pyramid
- Algorithm searches for $\mathbf{d}^*$ by moving convex hull within **D**
- If algorithm works properly, convex hull shrinks/collapses onto $\mathbf{d}^*$
- No injected randomness (contrast with algorithms A, B, and C)
- Frequently effective, but no general convergence theory and many numerical counterexamples to convergence

Basic ideas

- At each iteration a new point is generated in or near the current simplex
- The new point replaces one of the current simplex vertices, yielding a new simplex
- The reflection step: the new point is the reflection (across the hyperplane spanned by the other vertices) of the vertex that currently has the highest value of $L(\mathbf{d})$
- So the simplex is moved in a direction away from the high values of the loss function and toward the lowest values of the loss
- Repeat until the simplex size is sufficiently small

Steps of Nonlinear Simplex Algorithm

Step 0 (Initialization) Generate initial set of $p+1$ points in $\mathbf{D} \in \mathcal{R}^p$, $\mathbf{d}_i, i = 1, \ldots, p+1$, vertices of initial simplex. (Any vertex generated outside $\mathbf{D}$ is moved to the nearest point in $\mathbf{D}$). Calculate $L(\mathbf{d}_i)$. Set $\alpha = 1.0, \beta = 0.5, \gamma = 2.0, \delta = 0.5$ (default setting, not compulsory).

Step 1 (Reflection) Identify where max, second highest, and min loss values occur; denote them by $\mathbf{d}_{max}, \mathbf{d}_{2max}$, and $\mathbf{d}_{min}$, respectively. Let $\mathbf{d}_{cent}$ = centroid (mean) of all $\mathbf{d}_i$ except for $\mathbf{d}_{max}$. Generate candidate vertex $\mathbf{d}_{refl}$ by reflecting $\mathbf{d}_{max}$ through $\mathbf{d}_{cent}$ using $\mathbf{d}_{refl} = (1+\alpha)\mathbf{d}_{cent} - \alpha\mathbf{d}_{max}, (\alpha > 0)$.

Step 1a (Accept reflection) If $L(\mathbf{d}_{min}) \leq L(\mathbf{d}_{refl}) < L(\mathbf{d}_{2max})$, then replaces $\mathbf{d}_{max}$ with $\mathbf{d}_{refl}$ and proceed to Step 5; else go to Step 2.

Steps of Nonlinear Simplex Algorithm

Step 2 (Expansion) If $L(\mathbf{d}_{refl}) < L(\mathbf{d}_{min})$, then expand reflection using $\mathbf{d}_{exp} = \gamma \mathbf{d}_{refl} + (1-\gamma)\mathbf{d}_{cent}, \gamma > 1$ and go to Step 2a; else go to Step 3.

Step 2a (Check expansion) If $L(\mathbf{d}_{exp}) < L(\mathbf{d}_{refl})$, then replaces $\mathbf{d}_{max}$ with $\mathbf{d}_{exp}$; otherwise reject expansion and replace $\mathbf{d}_{max}$ by $\mathbf{d}_{refl}$. Go to Step 5.

Step 3 (Contraction) If $L(\mathbf{d}_{refl}) < L(\mathbf{d}_{max})$ (*outside contraction*), the contraction point is $\mathbf{d}_{cont} = \beta \mathbf{d}_{refl}$ $+(1-\beta)\mathbf{d}_{cent}$, $0 \leq \beta \leq 1$. If $L(\mathbf{d}_{max}) \leq L(\mathbf{d}_{refl})$ (*inside contraction*), then $\mathbf{d}_{cont} = \beta \mathbf{d}_{max}$ $+(1-\beta)\mathbf{d}_{cent}$.

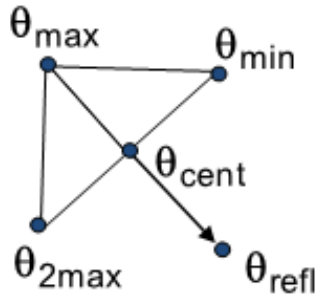
Steps of Nonlinear Simplex Algorithm

Step 3a Check contraction If outside contraction, replace $\mathbf{d}_{max}$ with the point minimizing $\{L(\mathbf{d}_{cont}), L(\mathbf{d}_{refl})\}$. If inside contraction and $L(\mathbf{d}_{cont}) < L(\mathbf{d}_{max})$ then replace $\mathbf{d}_{max}$ by $\mathbf{d}_{cont}$ and go to Step 5; otherwise go to Step 4.

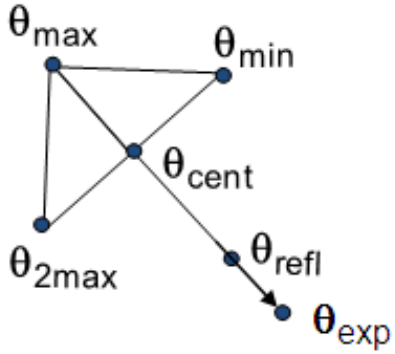
Step 4 (Shrink) Shrink entire simplex using a factor $0 < \delta < 1$, retaining only $\mathbf{d}_{min}$ (replace each vertex $\mathbf{d}_{(i)}$ (except $\mathbf{d}_{min}$) with $\delta\mathbf{d}_{(i)} + (1 - \delta)\mathbf{d}_{min}$. Go to Step 5.

Step 5 (Termination) Stop if convergence criterion or maximum number of function evaluations is met; else return to Step 1.

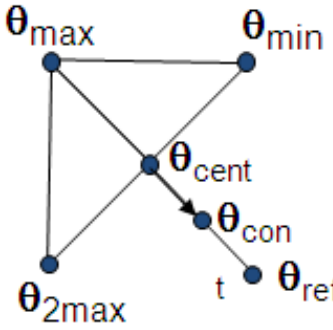
Illustration of Steps of Nonlinear Simplex Algorithm with $p = 2$



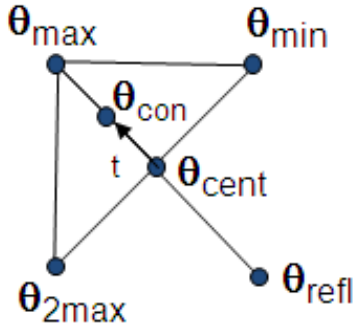
Reflection



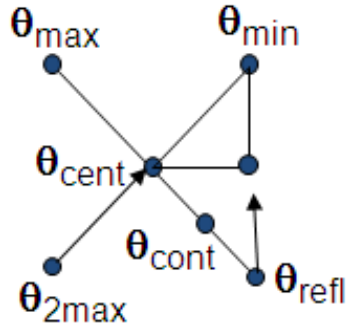
Expansion when $L(\theta_{refl}) < L(\theta_{min})$



Contraction when $L(\theta_{refl}) < L(\theta_{max})$ ("outside")

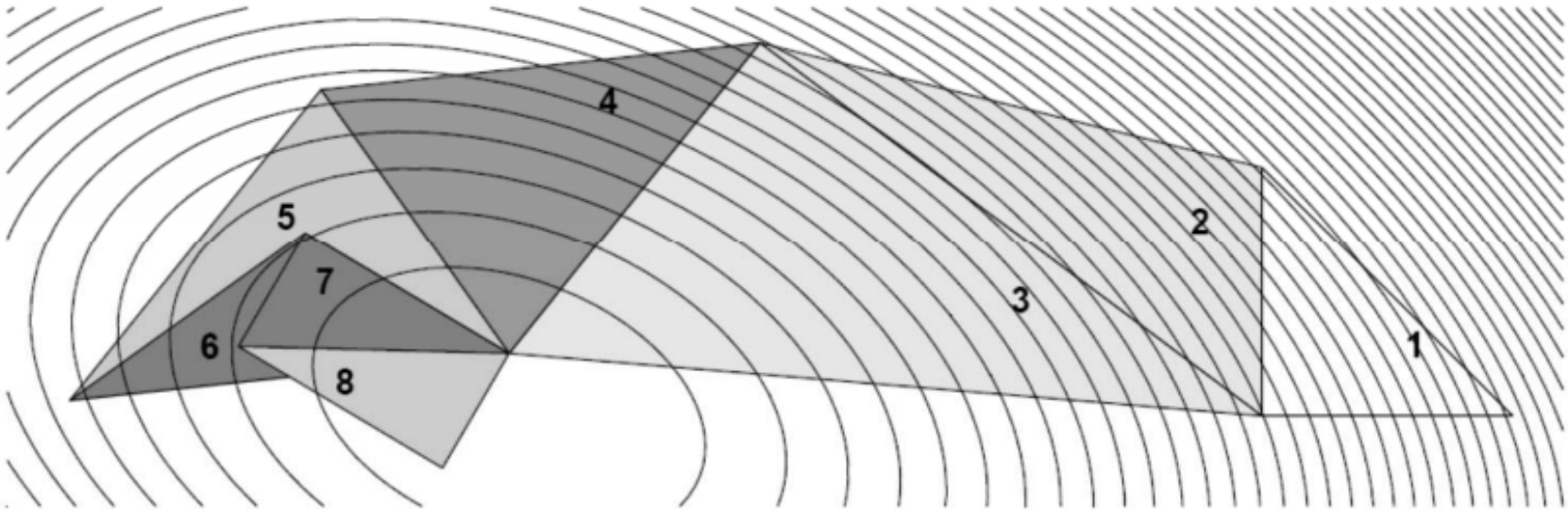


Contraction when $L(\theta_{refl}) \geq L(\theta_{max})$ ("inside")



Shrink after failed contraction when $L(\theta_{refl}) < L(\theta_{max})$

A Run of Nonlinear Simplex Algorithm for $p = 2$



(1) Expansion;
(2) Expansion;
(3) Outer contraction;
(4) Reflection;
(5) Inner contraction;
(6) Inner contraction;
(7) Reflection;