PRO 5970 Métodos de Otimização Não Linear

Celma de Oliveira Ribeiro

2023

Departmento de Engenharia de Produção
Universidade de São Paulo

PRO 5970 Métodos de Otimização Não Linear

Celma de Oliveira Ribeiro

Aula 6 - Derivative free - 2023

PPGEP - EPUSP

Global Optimization

Practical applications of optimization deal with one or more of the following challenges:

- non-diferentiable functions and/or constraints
- non-convex feasible space
- discrete feasible space
- mixed variables
- dimension
- multiple local minima
- multiple objectives

Global Optimization

An interesting example:

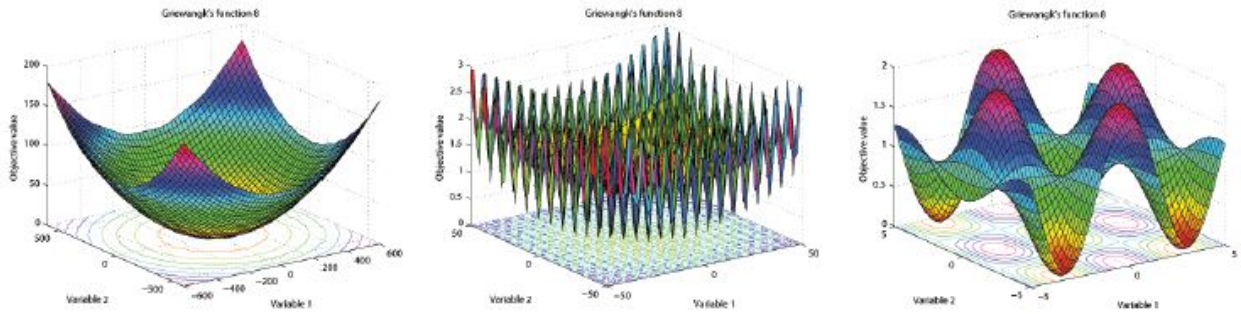


Figure 6.2: The Griewank function looks deceptively smooth when plotted in a large domain (left), but when you zoom in, you can see that the design space has multiple local minima (center) although the function is still smooth (right)

How to solve this problem?

Global Optimization

An interesting example:

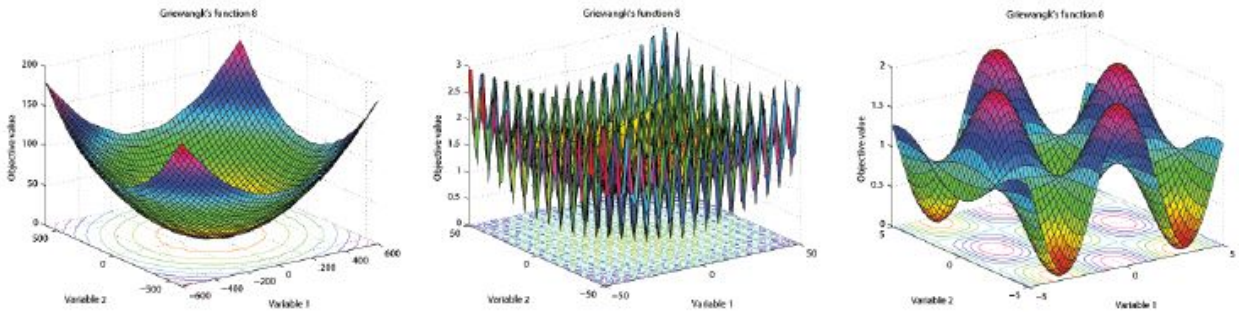


Figure 6.2: The Griewank function looks deceptively smooth when plotted in a large domain (left), but when you zoom in, you can see that the design space has multiple local minima (center) although the function is still smooth (right)

How to solve this problem?

- Multiple point restarts of gradient (local) based optimizer

Global Optimization

An interesting example:

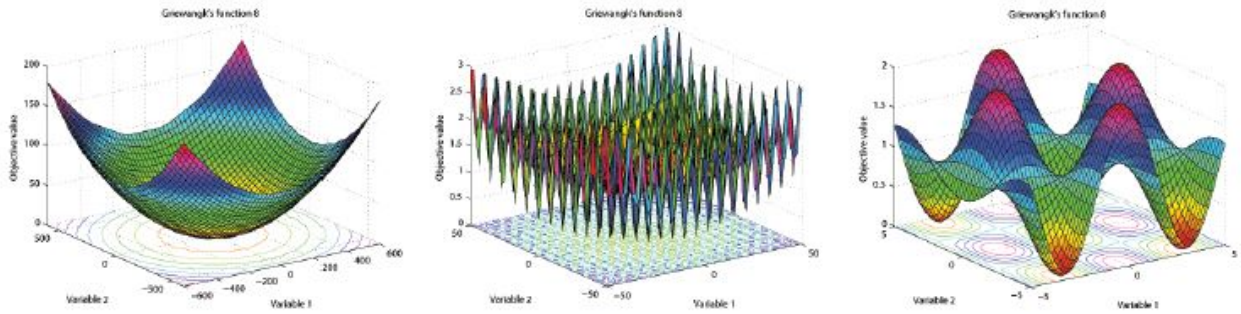


Figure 6.2: The Griewank function looks deceptively smooth when plotted in a large domain (left), but when you zoom in, you can see that the design space has multiple local minima (center) although the function is still smooth (right)

How to solve this problem?

- Multiple point restarts of gradient (local) based optimizer
- Systematically search the design space

Global Optimization

An interesting example:

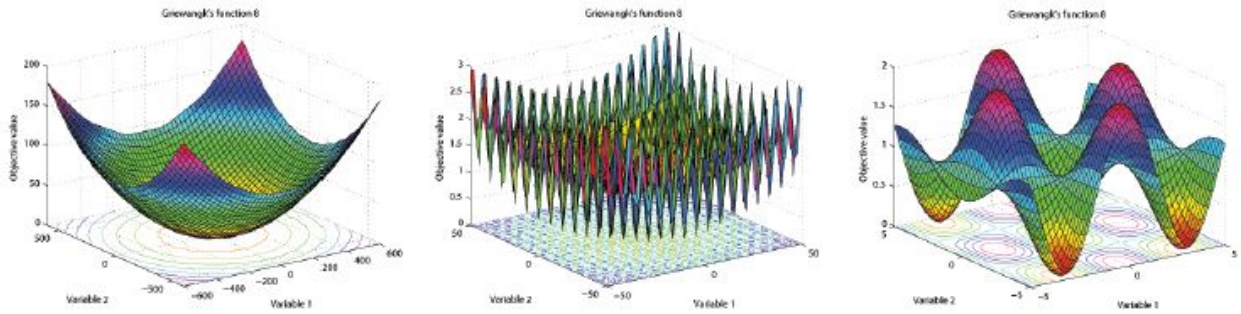


Figure 6.2: The Griewank function looks deceptively smooth when plotted in a large domain (left), but when you zoom in, you can see that the design space has multiple local minima (center) although the function is still smooth (right)

How to solve this problem?

- Multiple point restarts of gradient (local) based optimizer
- Systematically search the design space
- Use gradient-free optimizers

Derivative-based algorithms

Advantages

- global convergence to stationary points of the problem (P) under mild assumptions on the problem class;
- fast local convergence for Newton-like variants.
- can solve large-scale problems for n large (order 10^3) efficiently, even when (P) has nonlinear constraints.

Derivative-based algorithms

Advantages

- global convergence to stationary points of the problem (P) under mild assumptions on the problem class;
- fast local convergence for Newton-like variants.
- can solve large-scale problems for n large (order 10^3) efficiently, even when (P) has nonlinear constraints.

Limitations

- only guaranteed to provide local solutions of the problem (P) when (P) is nonconvex.
- requires accurate or exact first, and sometimes even second, derivatives of the objective f and constraints to be available.

Global Optimization

Gradient-based optimizers:

- are efficient at finding local minima for high-dimensional, nonlinearly constrained, convex problems;

Global Optimization

Gradient-based optimizers:

- are efficient at finding local minima for high-dimensional, nonlinearly constrained, convex problems;
- have problems dealing with noisy and discontinuous functions

Global Optimization

Gradient-based optimizers:

- are efficient at finding local minima for high-dimensional, nonlinearly constrained, convex problems;
- have problems dealing with noisy and discontinuous functions
- not designed to handle multi-modal problems or discrete and mixed discrete-continuous design variables.

Global Optimization

Gradient-based optimizers:

- are efficient at finding local minima for high-dimensional, nonlinearly constrained, convex problems;
- have problems dealing with noisy and discontinuous functions
- not designed to handle multi-modal problems or discrete and mixed discrete-continuous design variables.
- A local optimization algorithm 'gets trapped' at local minimizers and cannot further advance towards the global solution.

Global Optimization

Gradient-based optimizers:

- are efficient at finding local minima for high-dimensional, nonlinearly constrained, convex problems;
- have problems dealing with noisy and discontinuous functions
- not designed to handle multi-modal problems or discrete and mixed discrete-continuous design variables.
- A local optimization algorithm 'gets trapped' at local minimizers and cannot further advance towards the global solution.
- Suppose f non convex and bounded below.
  - How to compute a global minimizer in the presence of local minimizers?
  - How to compute a global minimizer in the presence of high oscillations and sometimes noise?

Direct search methods

- Direct search methods are best known as unconstrained optimization techniques that do not explicitly use derivatives

Direct search methods

- Direct search methods are best known as unconstrained optimization techniques that do not explicitly use derivatives
- Direct search methods attempt to find the optimal solution of optimization problems without explicitly using derivatives

Direct search methods

- Direct search methods are best known as unconstrained optimization techniques that do not explicitly use derivatives
- Direct search methods attempt to find the optimal solution of optimization problems without explicitly using derivatives
- The term *derivative-free optimization* has been used with some frequency in the nonlinear programming literature to mean a number of methods that rely on derivative-free local models of the objective and constraints. These models are derivative-free in the sense that they are not Taylor's series models constructed using analytical or accurate finite-difference estimates of derivatives. Instead, the models are constructed via least-squares fits or interpolation techniques.

Direct search methods

Derivative-Free Optimization (DFO)

- derivatives are unavailable, even if the problem is smooth;
- use only function values to construct iterates that approach a (local) min.

Direct search methods

Derivative-Free Optimization (DFO)

- derivatives are unavailable, even if the problem is smooth;
- use only function values to construct iterates that approach a (local) min.

Why and when to use Derivative-Free Optimization?

Direct search methods

Derivative-Free Optimization (DFO)

- derivatives are unavailable, even if the problem is smooth;
- use only function values to construct iterates that approach a (local) min.

Why and when to use Derivative-Free Optimization?

- Exact first derivatives of f are unavailable: $f(x)$ given by a black-box code, proprietary code or a simulation package.

Direct search methods

Derivative-Free Optimization (DFO)

- derivatives are unavailable, even if the problem is smooth;
- use only function values to construct iterates that approach a (local) min.

Why and when to use Derivative-Free Optimization?

- Exact first derivatives of f are unavailable: $f(x)$ given by a black-box code, proprietary code or a simulation package.
- Computing f(x) for any given x is expensive: $f(x)$ given by a time-consuming numerical simulation or lab experiments.

Direct search methods

Derivative-Free Optimization (DFO)

- derivatives are unavailable, even if the problem is smooth;
- use only function values to construct iterates that approach a (local) min.

Why and when to use Derivative-Free Optimization?

- Exact first derivatives of f are unavailable: $f(x)$ given by a black-box code, proprietary code or a simulation package.
- Computing f(x) for any given x is expensive: $f(x)$ given by a time-consuming numerical simulation or lab experiments.
- The values of f(x) are noisy: the evaluation of f(x) is inaccurate, depends on discretization, sampling, uncertain data; gradient information is meaningless.

Global Optimization

Derivative-Free Optimization (DFO)

- use only objective function values to construct iterates.

Global Optimization

Derivative-Free Optimization (DFO)

- use only objective function values to construct iterates.
- do not essentially compute an approximate gradient. instead, form sample of points ;

Global Optimization

Derivative-Free Optimization (DFO)

- use only objective function values to construct iterates.
- do not essentially compute an approximate gradient. instead, form sample of points ;
- use associated function values to generate x_{k+1} so as to ensure descent;

Global Optimization

Derivative-Free Optimization (DFO)

- use only objective function values to construct iterates.
- do not essentially compute an approximate gradient. instead, form sample of points ;
- use associated function values to generate x_{k+1} so as to ensure descent;
- must also control geometry of sample sets.

Global Optimization

Derivative-Free Optimization (DFO)

- use only objective function values to construct iterates.
- do not essentially compute an approximate gradient. instead, form sample of points ;
- use associated function values to generate x_{k+1} so as to ensure descent;
- must also control geometry of sample sets.

However...

- a derivative-free method typically limits the performance in terms of accuracy, expense or problem size relative to what one might expect from gradient-based optimization methods

Global Optimization

Derivative-Free Optimization (DFO)

- use only objective function values to construct iterates.
- do not essentially compute an approximate gradient. instead, form sample of points ;
- use associated function values to generate x_{k+1} so as to ensure descent;
- must also control geometry of sample sets.

However...

- a derivative-free method typically limits the performance in terms of accuracy, expense or problem size relative to what one might expect from gradient-based optimization methods
- there is no agreed-upon dfinition of what constitutes a direct-search method.

Direct Search Algorithms - Main idea

A broad family of algorithms built on a simple idea: given a point $\bar{x}$ and a finite set of directions $D(\bar{x})$ such that

$$\exists\ d \in D(\bar{x}), d^{'} \nabla f(\bar{x}) \leq 0$$

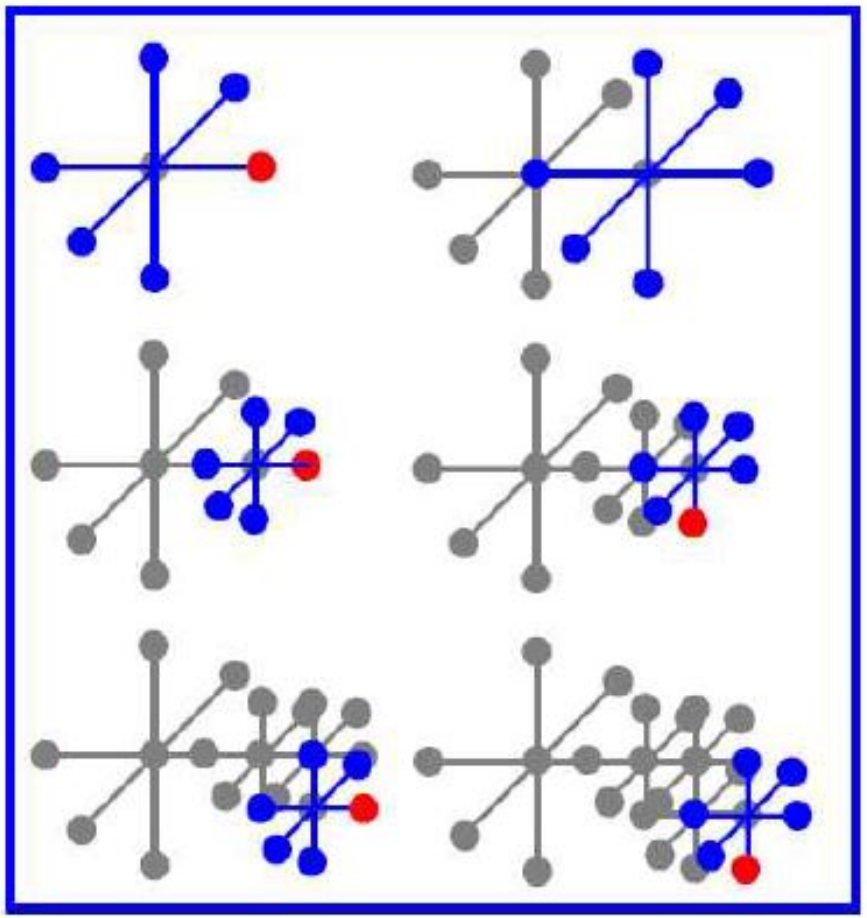
then there exists an $\alpha > 0$ small enough such that

$$f(\bar{x} - \alpha d) < f(\bar{x})$$

Direct Search Algorithms - Main idea

Input: x_0, α
$f^\star \leftarrow f(x_0)$;
$k \leftarrow 0$;
while *stopping criterion* **do**
  $f_k^\star \leftarrow \min_{d \in D(x_k)} f(x_k + \alpha d)$;
  $x_k^\star \leftarrow \arg\min_{d \in D(x_k)} f(x_k + \alpha d)$;
  if $f_k^\star < f^\star$ **then**
    $x_{k+1} \leftarrow x_k^\star$
  else
    update (shrink) α
  end
  $k \leftarrow k + 1$;
end

Direct Search Algorithms - Main idea



Derivative-Free Optimization (DFO)

Direct Search Algorithms - Main idea

- How to define D
- how to select α
- how to deal with constraints

Simplex method

The Simplex method (Nelder Mead) considers a collection of n + 1 the vertices of a simplex in $\mathbb{R}^n$. A simplex is a structure in n-dimensional space formed by n+1 points that are not in the same plane.

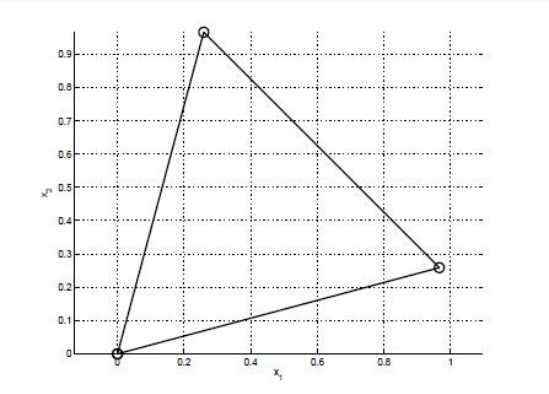


Figure 1: A simplex for n = 2

The Nelder-Mead algorithm performs four main operations to the simplex: *reflection*, *expansion*, *outside contraction*, *inside contraction* and *shrinking*.

The simplex method

- The operations enable the Nelder-Mead simplex method to distort the simplex in order to account for possible curvature present in the objective function.

The simplex method

- The operations enable the Nelder-Mead simplex method to distort the simplex in order to account for possible curvature present in the objective function.
- Each operation generates a new point and the sequence of operations performed in one iteration depends on the value of the objective at the new point relative to the other key points.

The simplex method

- The operations enable the Nelder-Mead simplex method to distort the simplex in order to account for possible curvature present in the objective function.
- Each operation generates a new point and the sequence of operations performed in one iteration depends on the value of the objective at the new point relative to the other key points.
- These operations enable the Nelder-Mead simplex method to distort the simplex in order to account for possible curvature present in the objective function.

The simplex method

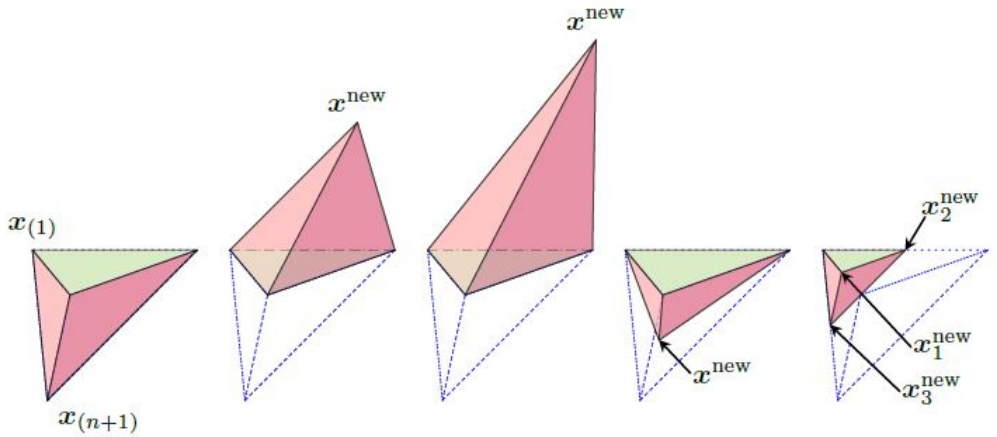


Figure 2.1: Primary Nelder–Mead simplex operations: original simplex, reflection, expansion, inner contraction, and shrink.

Let x_w, x_l and x_b denote the worst, the second worst (or lousy) and best points, respectively, among all n + 1 points of the simplex.

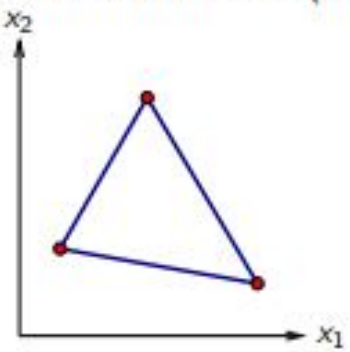
Nelder-Mead

Nelder-Mead Derivative-Free Search

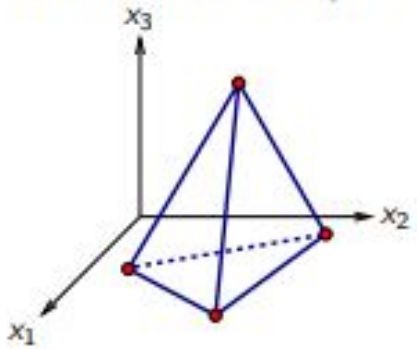
Iteratively consider the cost value at $n+1$ carefully chosen points:

- A simplex is a geometrical figure formed by $n+1$ points, $\{\mathbf{x}^1, \ldots, \mathbf{x}^{n+1}\}$, in an n-dimensional system

2-Dimensional Case ($n = 2$)



3-Dimensional Case ($n = 3$)



- The objective function is evaluated at each point $\mathbf{x}^i$, $i = 1, \ldots, n$, then the points are rearranged in nonimproving sequence:
  - **Maximize Problem:** $f(\mathbf{x}^1) > f(\mathbf{x}^2) > \cdots > f(\mathbf{x}^{n+1})$
  - **Minimize Problem:** $f(\mathbf{x}^1) < f(\mathbf{x}^2) < \cdots < f(\mathbf{x}^{n+1})$

Nelder-Mead

Nelder-Mead Derivative-Free Search (cont'd)

How do we construct a search directions Δx without the aid of (partial) derivatives?

- Nelder-Mead procedure adopts an away-from-worst approach

Away-from-Worst Search Direction

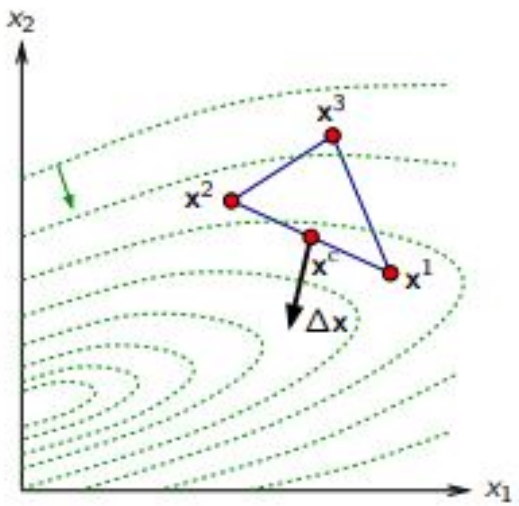
For an ordered set of points $\{\mathbf{x}^1, \ldots, \mathbf{x}^{n+1}\}$:

- Best-n centroid defined as

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

- Search direction $\Delta\mathbf{x}$ defined so as to *move away from worst point*,

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



Nelder-Mead

Nelder-Mead Derivative-Free Search (cont'd)

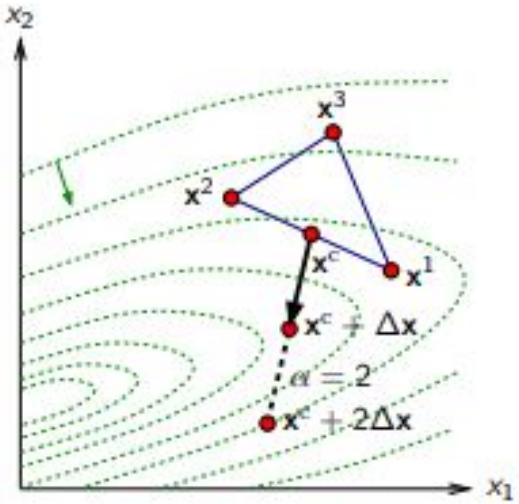
How do we decide the step size α in the direction $\Delta\mathbf{x}$?

- Nelder-Mead procedure first reflects about centroid $\mathbf{x}^c$ (i.e., with step size $\alpha = 1$)
- If the new point $\mathbf{x}^c + \Delta\mathbf{x}$ is neither better than $\mathbf{x}^1$ nor worse than $\mathbf{x}^n$, it is adopted without further trials

Nelder-Mead Expansion

In case $\mathbf{x}^c + \Delta\mathbf{x}$ is a new best:

- Attempt expansion with step $\alpha = 2$
- Accept expansion $\mathbf{x}^c + 2\Delta\mathbf{x}$ only in case of an improvement!



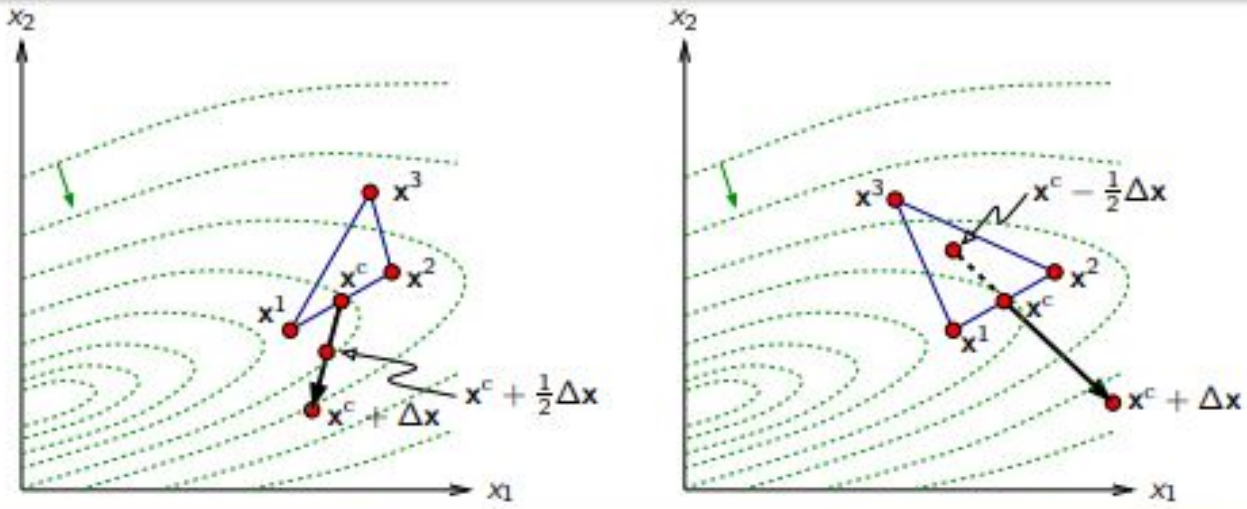
Nelder-Mead

Nelder-Mead Derivative-Free Search (cont'd)

Nelder-Mead Contraction

In case $\mathbf{x}^c + \Delta\mathbf{x}$ would still remain worse than $\mathbf{x}^n$:

1. Attempt contraction with $\alpha = \frac{1}{2}$, if $f(\mathbf{x}^c + \Delta\mathbf{x})$ better than $f(\mathbf{x}^{n+1})$
2. Attempt contraction with $\alpha = -\frac{1}{2}$, otherwise
3. Accept contraction $\mathbf{x}^c \pm \frac{1}{2}\Delta\mathbf{x}$ only in case of an improvement!



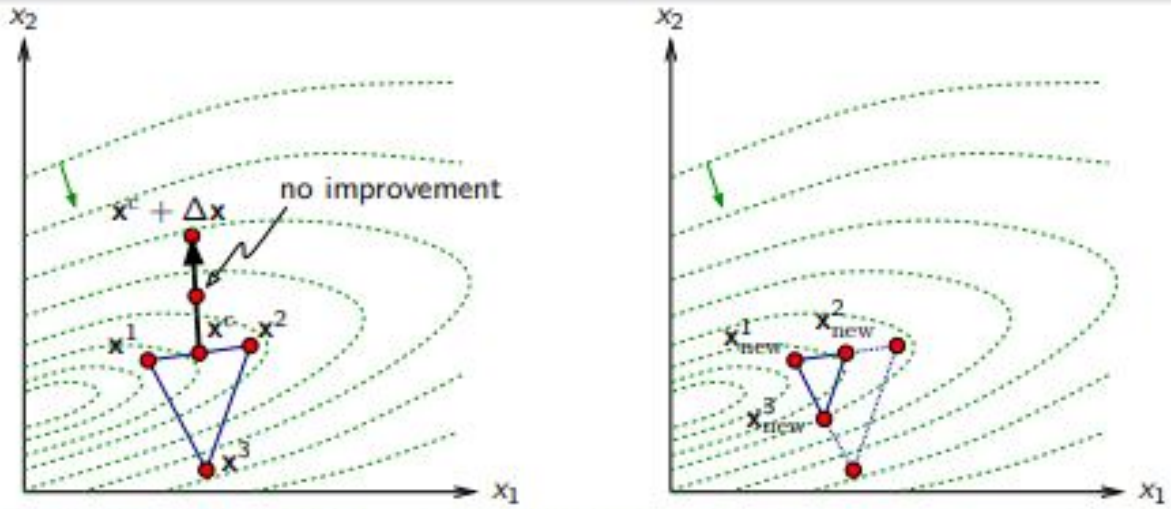
Nelder-Mead

Nelder-Mead Derivative-Free Search (cont'd)

Nelder-Mead Shrinking

In case neither the reflection point nor a contraction alternative yields an improving point w.r.t. $\mathbf{x}^n$, the procedure shrinks the whole array toward current best $\mathbf{x}^1$,

$$\mathbf{x}^i_{\text{new}} \triangleq \frac{1}{2}\left[\mathbf{x}^1 + \mathbf{x}^i\right], \quad \text{for all } i = 2, \ldots, n+1$$



Nelder-Mead

Nelder-Mead Derivative-Free Search Algorithm

- **Step 0: Initialization**
  - Choose $(n+1)$ distinct points $\{\mathbf{x}^1, \ldots, \mathbf{x}^{n+1}\}$, and evaluate $f(\mathbf{x}^1), \ldots, f(\mathbf{x}^{n+1})$; set stopping tolerance $\epsilon > 0$
- **Step 1: Move Direction**
  - Rearrange the $\mathbf{x}^i$ in nonimproving sequence, and compute best-n centroid:

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

  - If $|f(\mathbf{x}^i) - f(\mathbf{x}^c)| < \epsilon$, stop — Report the best of current $\mathbf{x}^c$, $\mathbf{x}^1$
  - Otherwise, compute away-from-worst move direction as

$$\Delta\mathbf{x} \triangleq \mathbf{x}^c - \mathbf{x}^{n+1};$$

proceed to step 2

Nelder-Mead

Nelder-Mead Derivative-Free Search Algorithm (cont'd)

- **Step 2: Step Size**
  - If $f(\mathbf{x}^c + \Delta\mathbf{x})$ improves on current best $f(\mathbf{x}^1)$: expand by trying $f(\mathbf{x}^c + 2\Delta\mathbf{x})$; set $\alpha = 2$ if further improvement is obtained, $\alpha = 1$ otherwise; proceed to step 3
  - If $f(\mathbf{x}^c + \Delta\mathbf{x})$ does not improve on second worst $f(\mathbf{x}^n)$: contract by trying $f(\mathbf{x}^c + \frac{1}{2}\Delta\mathbf{x})$ if $f(\mathbf{x}^c + \Delta\mathbf{x})$ is better than $f(\mathbf{x}^{n+1})$, or trying $f(\mathbf{x}^c - \frac{1}{2}\Delta\mathbf{x})$ otherwise; set $\alpha = \pm\frac{1}{2}$ if improvement is obtained w.r.t. $f(\mathbf{x}^{n+1})$; proceed to step 3
- **Step 3: Update**
  - If contraction is nonimproving, shrink the current simplex toward best $\mathbf{x}^1$ by setting:

$$\mathbf{x}^i_{\text{new}} \stackrel{\Delta}{=} \frac{1}{2}\left(\mathbf{x}^1 - \mathbf{x}^i\right), \quad \text{for all } i = 2, \ldots, n+1;$$

    compute new function values $f(\mathbf{x}^2), \ldots, f(\mathbf{x}^{n+1})$; return to step 1
  - Otherwise, replace $\mathbf{x}^{n+1}$ in the set of points by $\mathbf{x}^c + \alpha\Delta\mathbf{x}$; return to step 1

The simplex method

Consider x_w, x_l and x_b the worst, the second worst and best points.

Calculate

$$x_a = \frac{1}{n} \sum_{i=1, i \neq w}^{n+1} x_i$$

The line from x_w to x_a is a descent direction

Reflection

A new point is found on this line by reflection, given by

$$x_r = x_a + \alpha(x_a - x_w)$$

Expansion

If the value of the function at this reflected point is better than the best point, then the reflection has been especially successful and we step further in the same direction by performing an expansion

$$x_e = x_r + \gamma(x_r - x_a)$$

Usually $\gamma = 1$.

Contraction

If the reflected point is worse than the worst point, we assume that a better point exists between x_w and x_a and perform an inside contraction

$$x_c = x_a + \beta(x_a - x_w)$$

Usually the contraction factor $\beta = 0.5$.

If the reflected point is not worse than the worst but is still worse than the lousy point, then an **outside contraction** is performed

$$x_o = x_a + \beta(x_a - x_w)$$

After the expansion, we accept the new point if the value of the objective function is better than the best point. Otherwise, we just accept the previous reflected point.

Shrinking

If reflection, expansion, and contraction fail, we resort to a shrinking operation. This operation retains the best point and shrinks the simplex, that is, for all point of the simplex except the best one, we compute a new position

$$x_i = x_b + \rho(x_i - x_b)$$

Usually the scaling parameter $\rho = 0.5$

The simplex method

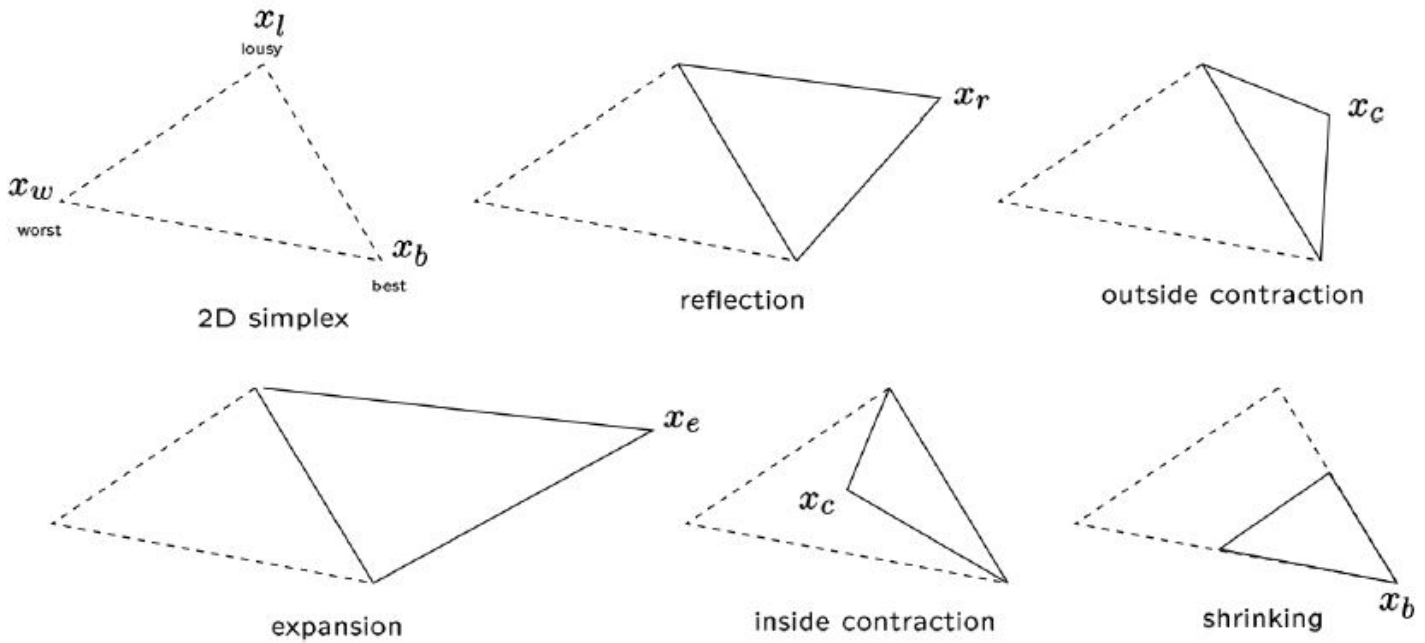


Figure 6.5: Operations performed on the simplex in Nelder–Mead's algorithm for $n = 2$.

The simplex method

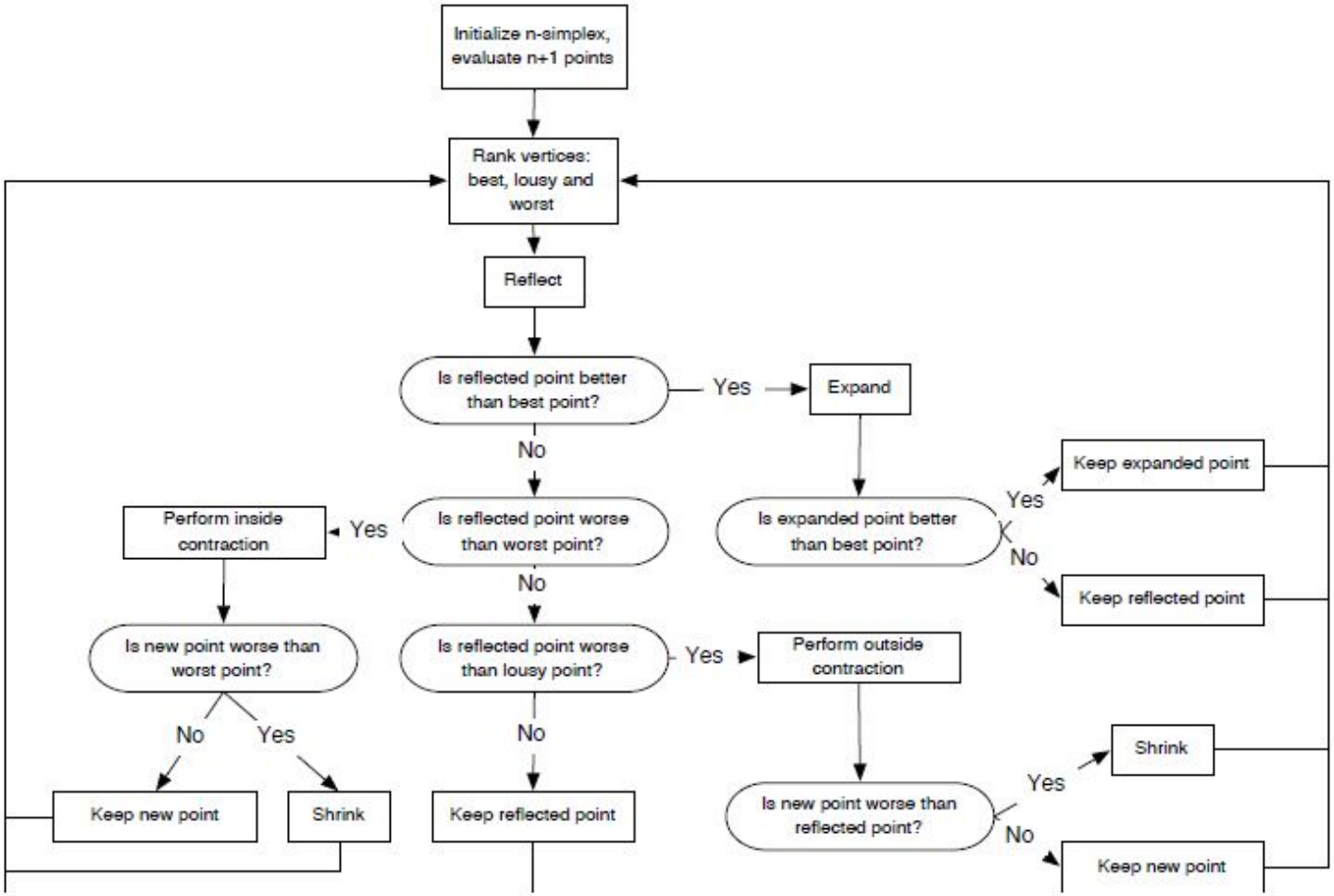


Figure 2: Flow chart

The simplex method

Convergence criterion:

- The size of simplex, i.e.,

$$\sum_{i=1}^{n} \|x_i - x_{n+1}\| \leq \epsilon$$

- Standard deviation

$$\sqrt{\frac{\sum_{i=1}^{n+1}(f_i - \bar{f})}{n+1}} \leq \epsilon$$

Minimization of a Function Using Nelder-Mead

A 2-d NM picture (note the ease of understanding what's happening!):

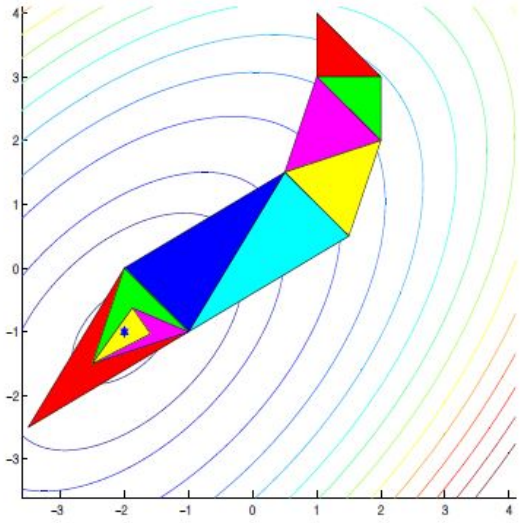


Figure 3: sequence of simplices that results when minimizing a function

Example Função sombrero Entregar - aula

Para a função

$$f(x,y) = 10\frac{sin\sqrt{x^2+y^2}}{\sqrt{x^2+y^2}}$$

- Esboce o gráfico da função
- faça 4 iterações do método de Nelder Mead e apresente os pontos e os simplexos obtidos. Começe com pontos iniciais distintos (explique a escolha dos pontos e sua expectativa em relação ao comportamento do algoritmo)

Minimization of a Function Using Nelder-Mead

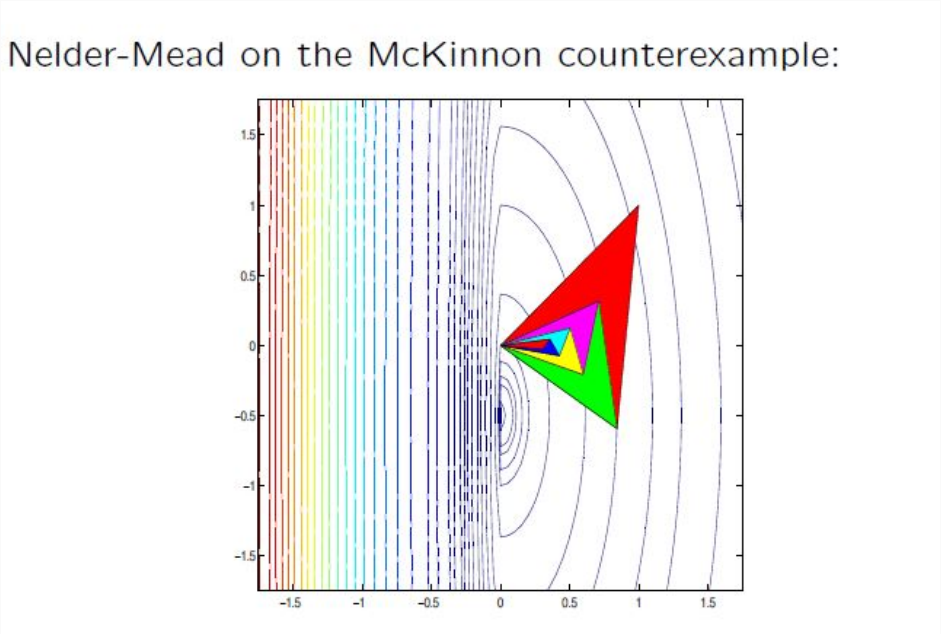


Figure 4: sequence of simplices that results when minimizing a function

Minimization of the Rosenbrock Function Using Nelder-Mead

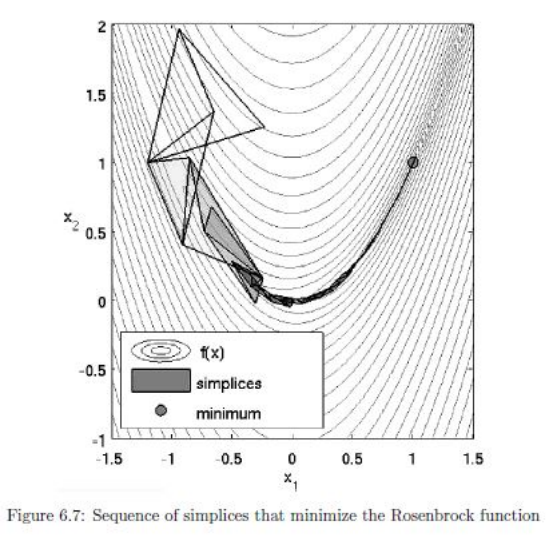


Figure 6.7: Sequence of simplices that minimize the Rosenbrock function

Figure 5: sequence of simplices that results when minimizing the Rosenbrock function

The initial simplex: on the upper left (equilateral). First iteration: inside contraction. Second: reflection, another inside contraction and then an expansion. The simplices then reduce dramatically in size and follow the Rosenbrock valley, slowly converging to the minimum.

Random search

Enumeration or exaustive search

- Most natural approach in global optimization: evaluate all points in the domain

Random search

Enumeration or exaustive search

- Most natural approach in global optimization: evaluate all points in the domain
- When global optimization involves continuous variables, complete enumeration is impossible.

Random search

Enumeration or exaustive search

- Most natural approach in global optimization: evaluate all points in the domain
- When global optimization involves continuous variables, complete enumeration is impossible.
- Commom approach: discretizing the domain **Grid search**

Random search

Enumeration or exaustive search

- Most natural approach in global optimization: evaluate all points in the domain
- When global optimization involves continuous variables, complete enumeration is impossible.
- Commom approach: discretizing the domain **Grid search**
  - Create a equally spaced grid of points over the feasible region

Random search

Enumeration or exaustive search

- Most natural approach in global optimization: evaluate all points in the domain
- When global optimization involves continuous variables, complete enumeration is impossible.
- Commom approach: discretizing the domain **Grid search**
  - Create a equally spaced grid of points over the feasible region
  - Evaluate the objective function at each point

Random search

Enumeration or exaustive search

- Most natural approach in global optimization: evaluate all points in the domain
- When global optimization involves continuous variables, complete enumeration is impossible.
- Commom approach: discretizing the domain **Grid search**
  - Create a equally spaced grid of points over the feasible region
  - Evaluate the objective function at each point
  - The number of function evaluations to achieve an accuracy ϵ is exponencial in the dimension n

Random search

Enumeration or exaustive search

Pure random search

- Stochastic version of grid search

Random search

Enumeration or exaustive search

Pure random search

- Stochastic version of grid search
- Samples repeatedly from the feasible region S, tipically according to a uniform sampling distribution. Although the points are not evenly spaced, they are uniformly scattered over the feasible solution

Random search

Enumeration or exaustive search

Pure random search

- Stochastic version of grid search
- Samples repeatedly from the feasible region S, tipically according to a uniform sampling distribution. Although the points are not evenly spaced, they are uniformly scattered over the feasible solution
- It sacrifices the guarantee of determining the optimal solution within ϵ. However, it can be shown that pure random search converges to the global optimum with probability one.

Random search algorithms

Random search algorithms (Also called Monte Carlo methods or stochastic algorithms) refer to algorithms that use some kind of randomness or probability (typically in the form of a pseudo-random number generator) in the definition of the method.

Generic random search algorithm is:

- A sequence of iterates $\{X_k\}$ on iteration $k = 0, 1, ...$ which may depend on previous points and algorithmic parameters.
- The current iterate X_k may represent a single point, or a collection of points, to include population based algorithms

Random search can use derivatives!

Random search

Generic Random Search Algorithm

Step 0 Initialize algorithm parameters Θ_0, initial points $X_0 \in S$, $k \leftarrow 0$

Step 1 Generate a collection of candidate points $V_{k+1} \in S$ according to a specific generator and associated sampling distribution.

Step 2 Update X_{k+1} based on the candidate points V_{k+1}, previous iterates and algorithmic parameters.Θ_{k+1}

Step 3 If a stopping criterion is met, stop. Otherwise increment k and return to Step 1.

This generic random search algorithm depends on two basic procedures, the generator in Step 1 that produces candidate points, and the update procedure in Step 2.

Single-point Generators

- Maintain and generate a single point at each iteration.
- For these single-point generators, the candidate point V_{k+1} is generated based on a combination of the current point and previous points.
- Usual approach: Step 1 expressed by $V_{k+1} = X_{k+1} + \delta_k D_k$ (step size algorithms)

In continuous problems, the direction of movement D_k may be motivated by gradient information, and the step length may be the result of a line search δ_k. Quasi-Newton methods take advantage of an approximation of the Hessian to provide a search direction.

Random search

Pure random search

Step 0 Initialize $X_0 \in S$, according to a probability measure δ on S [1]
$k \leftarrow 0$. $Y_{best} = Y_0 = f(X_0)$

Step 1 Generate $X_{k+1} \in S$ according to probability measure δ. Set $Y_{k+1} = f(X_{k+1})$.
Update the best point so far, $Y_{best} = \min\{Y_{best}, Y_{k+1}\}$

Step 2 If a stopping criterion is met, stop. Otherwise increment k and return to Step 1.

Random search

Pure adaptative random search

Step 0 Initialize $X_0 \in S$, according to a uniform distribution on S
$k \leftarrow 0$. $W_0 = f(X_0)$

Step 1 Generate $X_{k+1} \in S$ according to a uniform distribution on the improving set
$S_k = \{x : x \in S \text{ and } f(x) < W_k\}$ Set $W_{k+1} = f(X_{k+1})$.

Step 2 If a stopping criterion is met, stop. Otherwise increment k and return to Step 1.

The method adapts the current level set by restricting its search domain to improving points

Multiple points Generators

- Population-based random search algorithms use a collection of current points to generate another collection of candidate points.
- Many of these algorithms are motivated by biological processes, and include genetic algorithms, evolutionary programming, particle swarm optimization and ant colony optimization.

Update Procedure

After a candidate point is generated, Step 2 of the generic random search algorithm specifies a procedure to update the current point and algorithm parameters.

Strictly improving algorithms: update the current point only if the candidate point is improving,

$$X_{k+1} = \begin{cases} V_{k+1} & \text{if } \ f(V_{k+1}) < f(X_k) \\ X_k & \text{otherwise} \end{cases}$$

This type of improving algorithm may get trapped in a local optimum if the neighborhood, or procedure for generating candidate points is too restricted.

A possibility is to accept non-improving points, as in *simulated annealing*.

Update Procedure Simulated annealing

In simulated annealing, the update procedure is often called the Metropolis criterion, and the candidate point is accepted with a probability,

$$X_{k+1} = \begin{cases} V_{k+1} & \text{with probability } \min\left\{1, \frac{f(X_k)-f(V_{k+1})}{T_k}\right\} \\ X_k & \text{otherwise} \end{cases}$$

- T_k is a cooling parameter.
- Improving points are accepted with probability one, and the probability of accepting a worse point decreases as the temperature parameter cools, i.e. decreases to zero.