Introduction to Optimization

Lecture 4: Gradient-based Optimization

September 29, 2017

TC2 - Optimisation

Université Paris-Saclay

Dimo Brockhoff

Inria Saclay – Ile-de-France

Course Overview

1	Mon, 18.9.2017	first lecture
	Tue, 19.9.2017	groups defined via wiki
		everybody went (actively!) through the Getting Started part of github.com/numbbo/coco
2	Wed, 20.9.2017	lecture: "Benchmarking", final adjustments of groups everybody can run and postprocess the example experiment (~1h for final questions/help during the lecture)
3	Fri, 22.9.2017	today's lecture "Introduction to Continuous Optimization"
4	Fri, 29.9.2017	lecture "Gradient-Based Algorithms"
5	Fri, 6.10.2017	lecture "Stochastic Algorithms and DFO"
6	Fri, 13.10.2017	lecture "Discrete Optimization I: graphs, greedy algos, dyn. progr." deadline for submitting data sets
	Wed, 18.10.2017	deadline for paper submission
7	Fri, 20.10.2017	final lecture "Discrete Optimization II: dyn. progr., B&B, heuristics"
	Thu, 26.10.2017 / Fri, 27.10.2017	oral presentations (individual time slots)
	after 30.10.2017	vacation aka learning for the exams
	Fri, 10.11.2017	written exam

All deadlines:
23:59pm Paris time

Details on Continuous Optimization Lectures

Introduction to Continuous Optimization

- examples (from ML / black-box problems)
- typical difficulties in optimization

Mathematical Tools to Characterize Optima

- reminders about differentiability, gradient, Hessian matrix
- unconstraint optimization
  - first and second order conditions
  - convexity
- constraint optimization

Gradient-based Algorithms

- quasi-Newton method (BFGS)
- [DFO trust-region method]

Learning in Optimization / Stochastic Optimization

- CMA-ES (adaptive algorithms / Information Geometry)
- PhD thesis possible on this topic

method strongly related to ML / new promising research area

interesting open questions

Constrained Optimization

Equality Constraint

Objective:

Generalize the necessary condition of $\nabla f(x) = 0$ at the optima of f *when f is in $\mathcal{C}^1$, i.e. is differentiable and its differential is continuous*

Theorem:

Be U an open set of $(E, \|\ \|)$, and $f: U \to \mathbb{R}$, $g: U \to \mathbb{R}$ in $\mathcal{C}^1$.

Let $a \in E$ satisfy

$$\begin{cases} f(a) = \inf \{f(x) \mid x \in \mathbb{R}^n, g(x) = 0\} \\ g(a) = 0 \end{cases}$$

i.e. a is optimum of the problem

If $\nabla g(a) \neq 0$, then there exists a constant $\lambda \in \mathbb{R}$ called *Lagrange multiplier*, such that

$$\underbrace{\nabla f(a) + \lambda \nabla g(a) = 0}_{} \quad \text{Euler} - \text{Lagrange equation}$$

i.e. gradients of f and g in a are colinear

Geometrical Interpretation Using an Example

Exercise:

Consider the problem

$$\inf \{ f(x,y) \mid (x,y) \in \mathbb{R}^2, g(x,y) = 0\}$$

$f(x,y) = y - x^2 \qquad g(x,y) = x^2 + y^2 - 1 = 0$

1) Plot the level sets of f, plot $g = 0$
2) Compute ∇f and ∇g
3) Find the solutions with $\nabla f + \lambda \nabla g = 0$

   equation solving with 3 unknowns (x, y, λ)
4) Plot the solutions of 3) on top of the level set graph of 1)

Interpretation of Euler-Lagrange Equation

Intuitive way to retrieve the Euler-Lagrange equation:

- In a local minimum a of a constrained problem, the hypersurfaces (or level sets) $f = f(a)$ and $g = 0$ are necessarily tangent (otherwise we could decrease f by moving along $g = 0$).
- Since the gradients $\nabla f(a)$ and $\nabla g(a)$ are orthogonal to the level sets $f = f(a)$ and $g = 0$, it follows that $\nabla f(a)$ and $\nabla g(a)$ are colinear.

Generalization to More than One Constraint

Theorem

- Assume $f: U \to \mathbb{R}$ and $g_k: U \to \mathbb{R}$ $(1 \leq k \leq p)$ are $\mathcal{C}^1$.
- Let a be such that

$$\begin{cases} f(a) = \inf \{ f(x) \mid x \in \mathbb{R}^n, \quad g_k(x) = 0, \quad 1 \leq k \leq p \} \\ g_k(a) = 0 \ \text{ for all } 1 \leq k \leq p \end{cases}$$

- If $\left(\nabla g_k(a)\right)_{1 \leq k \leq p}$ are linearly independent, then there exist p real constants $(\lambda_k)_{1 \leq k \leq p}$ such that

$$\nabla f(a) + \sum_{k=1}^{p} \lambda_k \nabla g_k(a) = 0$$

↑ Lagrange multiplier

again: a does not need to be global but local minimum

The Lagrangian

- Define the Lagrangian on $\mathbb{R}^n \times \mathbb{R}^p$ as

$$\mathcal{L}(x, \{\lambda_k\}) = f(x) + \sum_{k=1}^{p} \lambda_k g_k(x)$$

- To find optimal solutions, we can solve the optimality system

$$\begin{cases} \text{Find } (x, \{\lambda_k\}) \in \mathbb{R}^n \times \mathbb{R}^p \text{ such that } \nabla f(x) + \sum_{k=1}^{p} \lambda_k \nabla g_k(x) = 0 \\ g_k(x) = 0 \ \text{ for all } 1 \leq k \leq p \end{cases}$$

$$\Leftrightarrow \begin{cases} \text{Find } (x, \{\lambda_k\}) \in \mathbb{R}^n \times \mathbb{R}^p \text{ such that } \nabla_x \mathcal{L}(x, \{\lambda_k\}) = 0 \\ \nabla_{\lambda_k} \mathcal{L}(x, \{\lambda_k\})(x) = 0 \ \text{ for all } 1 \leq k \leq p \end{cases}$$

Inequality Constraint: Definitions

Let $\mathcal{U} = \{x \in \mathbb{R}^n \mid g_k(x) = 0 \text{ (for } k \in E), \; g_k(x) \leq 0 \text{ (for } k \in I)\}$.

Definition:

The points in $\mathbb{R}^n$ that satisfy the constraints are also called *feasible* points.

Definition:

Let $a \in \mathcal{U}$, we say that the constraint $g_k(x) \leq 0$ (for $k \in I$) is *active* in a if $g_k(a) = 0$.

Theorem (Karush-Kuhn-Tucker, KKT):

Let U be an open set of $(E, || \, ||)$ and $f: U \to \mathbb{R}$, $g_k: U \to \mathbb{R}$, all $\mathcal{C}^1$

Furthermore, let $a \in U$ satisfy

$$\begin{cases} f(a) = \inf(f(x) \mid x \in \mathbb{R}^n, g_k(x) = 0 \ (\text{for } k \in E), g_k(x) \leq 0 \ (\text{for } k \in \text{I}) \\ g_k(a) = 0 \ (\text{for } k \in E) \\ g_k(a) \leq 0 \ (\text{for } k \in I) \end{cases}$$

also works again for a being a local minimum

Let I_a^0 be the set of constraints that are active in a. Assume that $\left(\nabla g_k(a)\right)_{k \, \in \, E \, \cup \, I_a^0}$ are linearly independent.

Then there exist $(\lambda_k)_{1 \leq k \leq p}$ that satisfy

$$\begin{cases} \nabla f(a) + \sum_{k=1}^{p} \lambda_k \nabla g_k(a) = 0 \\ g_k(a) = 0 \ (\text{for } k \in E) \\ g_k(a) \leq 0 \ (\text{for } k \in I) \\ \lambda_k \geq 0 \ (\text{for } k \in I_a^0) \\ \lambda_k g_k(a) = 0 \ (\text{for } k \in E \cup I) \end{cases}$$

Inequality Constraint: Karush-Kuhn-Tucker Theorem

Theorem (Karush-Kuhn-Tucker, KKT):

Let U be an open set of $(E, || \, ||)$ and $f: U \to \mathbb{R}$, $g_k: U \to \mathbb{R}$, all $\mathcal{C}^1$

Furthermore, let $a \in U$ satisfy

$$\begin{cases} f(a) = \inf(f(x) \mid x \in \mathbb{R}^n, g_k(x) = 0 \text{ (for } k \in E), g_k(x) \leq 0 \text{ (for } k \in \mathrm{I}) \\ g_k(a) = 0 \text{ (for } k \in E) \\ g_k(a) \leq 0 \text{ (for } k \in I) \end{cases}$$

Let I_a^0 be the set of constraints that are active in a. Assume that $\left(\nabla g_k(a)\right)_{k \in E \cup I_a^0}$ are linearly independent.

Then there exist $(\lambda_k)_{1 \leq k \leq p}$ that satisfy

$$\begin{cases} \nabla f(a) + \sum_{k=1}^{p} \lambda_k \nabla g_k(a) = 0 \\ g_k(a) = 0 \text{ (for } k \in E) \\ g_k(a) \leq 0 \text{ (for } k \in I) \\ \lambda_k \geq 0 \text{ (for } k \in I_a^0) \\ \lambda_k g_k(a) = 0 \text{ (for } k \in E \cup I) \end{cases}$$

either active constraint or $\lambda_k = 0$

Descent Methods

Descent Methods

General principle

1. choose an initial point $\boldsymbol{x}_0$, set $t = 1$
2. while not happy
   - choose a descent direction $\boldsymbol{d}_t \neq 0$
   - line search:
     - choose a step size $\sigma_t > 0$
     - set $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \sigma_t \boldsymbol{d}_t$
   - set $t = t + 1$

Remaining questions

- how to choose $\boldsymbol{d}_t$?
- how to choose σ_t?

Gradient Descent

Rationale: $\boldsymbol{d}_t = -\nabla f(\boldsymbol{x}_t)$ is a descent direction

indeed for f differentiable

$$f\big(x - \sigma\nabla f(x)\big) = f(x) - \sigma||\nabla f(x)||^2 + o(\sigma||\nabla f(x)||)$$
$$< f(x) \text{ for } \sigma \text{ small enough}$$

Step-size

- optimal step-size: $\sigma_t = \underset{\sigma}{\operatorname{argmin}} f(\boldsymbol{x}_t - \sigma\nabla f(\boldsymbol{x}_t))$
- **Line Search:** total or partial optimization w.r.t. σ
  Total is however often too "expensive" (needs to be performed at each iteration step)
  Partial optimization: execute a limited number of trial steps until a loose approximation of the optimum is found. Typical rule for partial optimization: Armijo rule (see next slides)

Typical stopping criterium:

norm of gradient smaller than ϵ

The Armijo-Goldstein Rule

Choosing the step size:

- Only to decrease f-value not enough to converge (quickly)
- Want to have a reasonably large decrease in f

Armijo-Goldstein rule:

- also known as backtracking line search
- starts with a (too) large estimate of σ and reduces it until f is reduced enough
- what is enough?
  - assuming a linear f e.g. $m_k(x) = f(x_k) + \nabla f(x_k)^T(x - x_k)$
  - expected decrease if step of σ_k is done in direction $\boldsymbol{d}$: $\sigma_k \nabla f(x_k)^T \boldsymbol{d}$
  - actual decrease: $f(x_k) - f(x_k + \sigma_k \boldsymbol{d})$
  - stop if actual decrease is at least constant times expected decrease (constant typically chosen in [0, 1])

The Armijo-Goldstein Rule

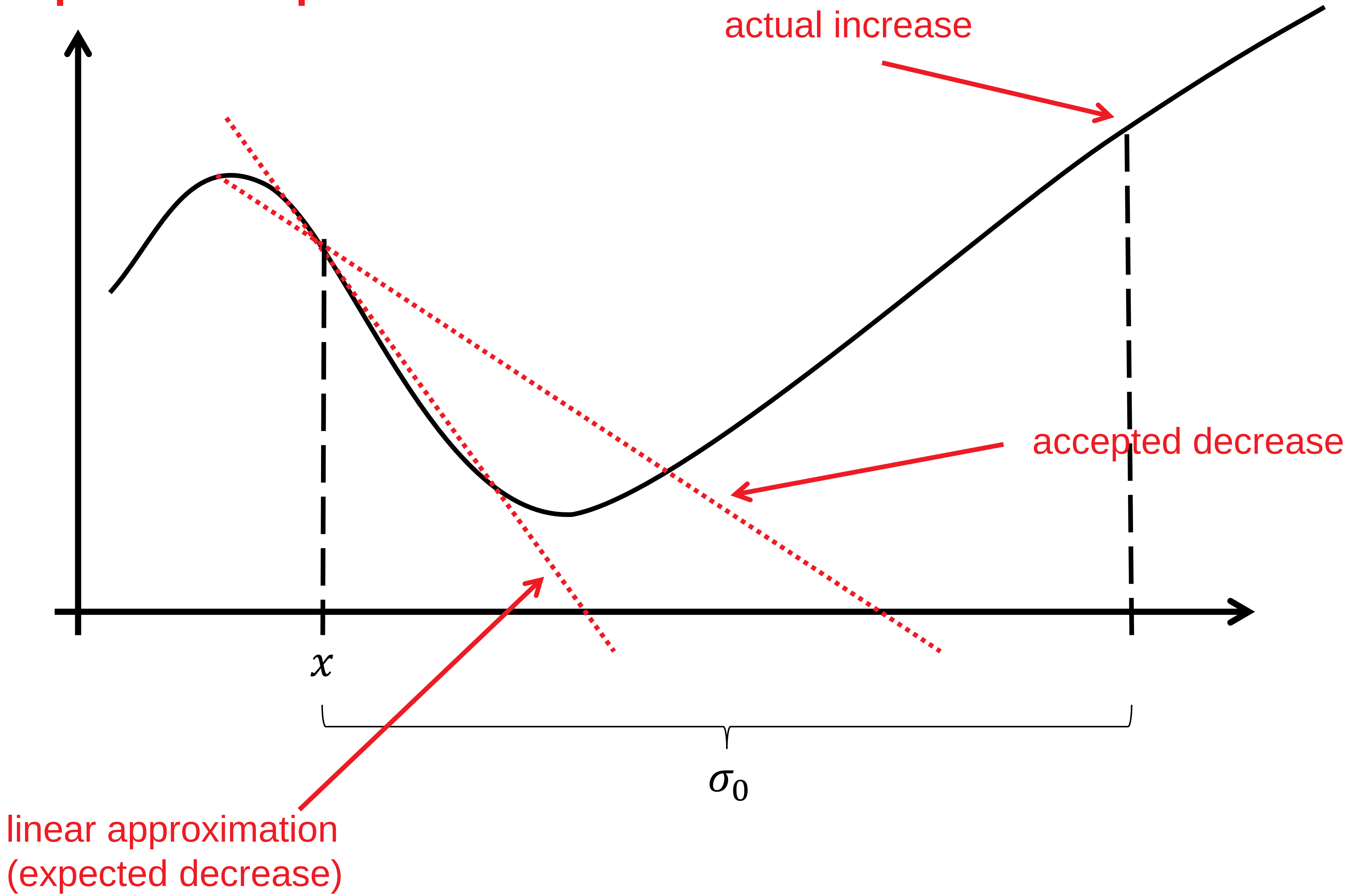
The Actual Algorithm:

Input: descent direction $\mathbf{d}$, point $\mathbf{x}$, objective function $f(\mathbf{x})$ and its gradient $\nabla f(\mathbf{x})$, parameters $\sigma_0 = 10$, $\theta \in [0,1]$ and $\beta \in (0,1)$
Output: step-size σ

Initialize σ: $\sigma \leftarrow \sigma_0$
while $f(\mathbf{x} + \sigma\mathbf{d}) > f(\mathbf{x}) + \theta\sigma\nabla f(\mathbf{x})^T\mathbf{d}$ **do**
    $\sigma \leftarrow \beta\sigma$
end while

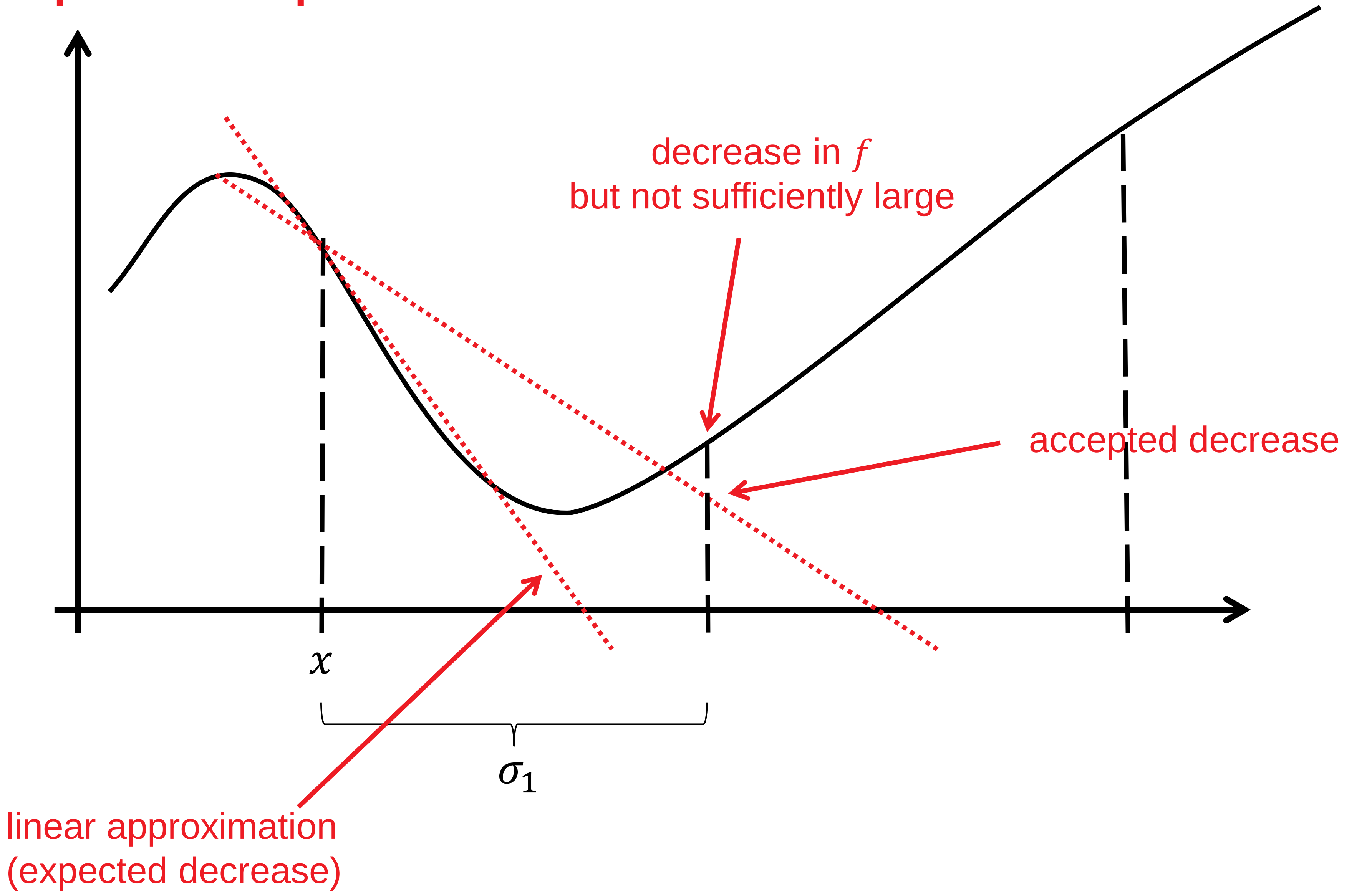
Armijo, in his original publication chose $\beta = \theta = 0.5$.
Choosing $\theta = 0$ means the algorithm accepts any decrease.

Graphical Interpretation

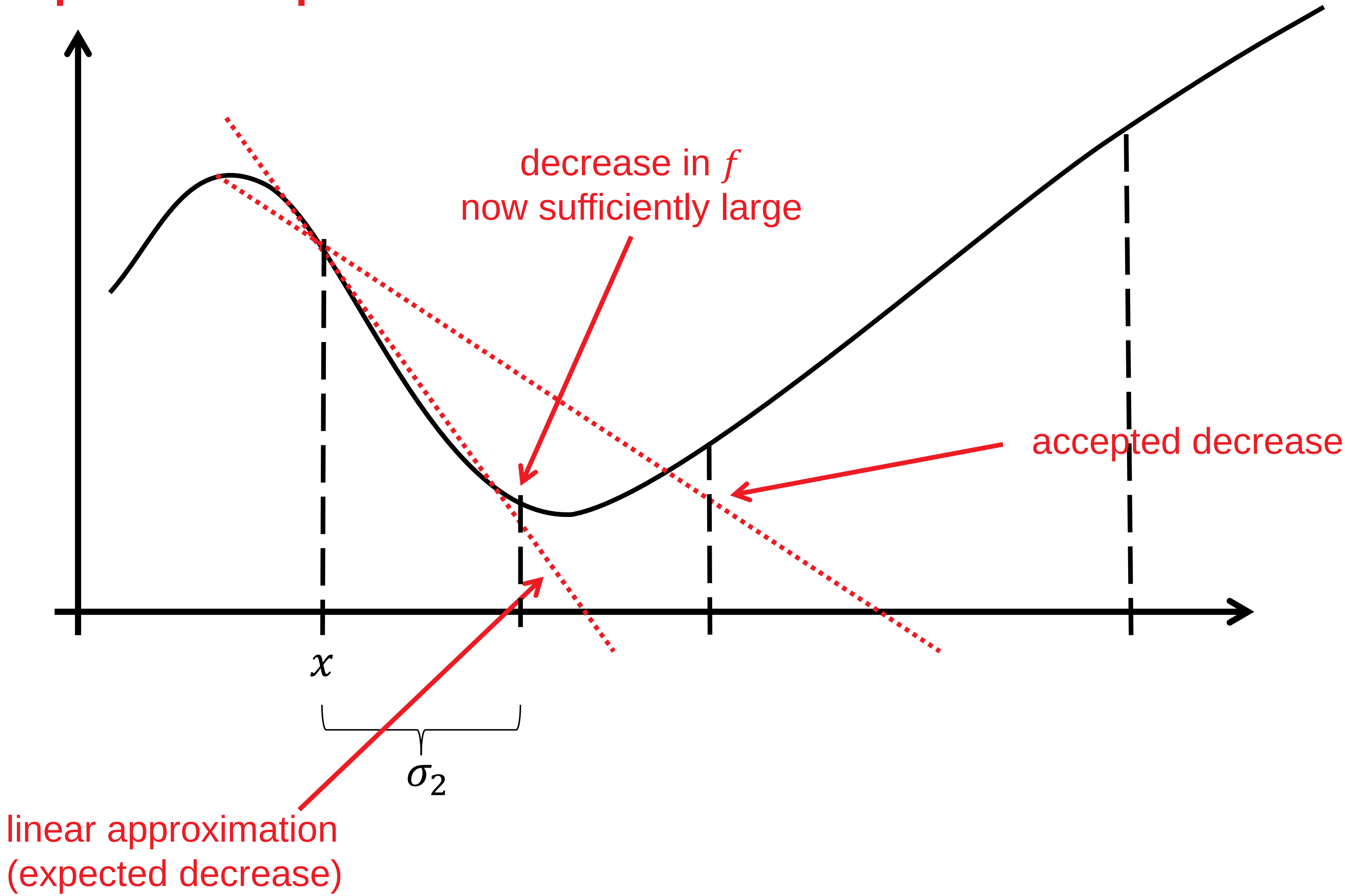


The Armijo-Goldstein Rule

Graphical Interpretation



Graphical Interpretation



Newton Algorithm

Newton Method

- descent direction: $-[\nabla^2 f(x_k)]^{-1} \nabla f(x_k)$ [so-called Newton direction]
- The Newton direction:
  - minimizes the best (locally) quadratic approximation of f:
    $\tilde{f}(x + \Delta x) = f(x) + \nabla f(x)^T \Delta x + \frac{1}{2} (\Delta x)^T \nabla^2 f(x) \Delta \mathrm{x}$
  - points towards the optimum on $f(x) = (x - x^*)^T A (x - x^*)$
- however, Hessian matrix is expensive to compute in general and its inversion is also not easy

quadratic convergence

(i.e. $\lim_{k \to \infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|^2} = \mu > 0$)

Remark: Affine Invariance

Affine Invariance: same behavior on $f(x)$ and $f(Ax + b)$ for $A \in \mathrm{GLn}(\mathbb{R})$ = set of all invertible $n \times n$ matrices over $\mathbb{R}$

- Newton method is affine invariant

  see **http://users.ece.utexas.edu/~cmcaram/EE381V_2012F/Lecture_6_Scribe_Notes.final.pdf**

- same convergence rate on all convex-quadratic functions
- Gradient method not affine invariant

Quasi-Newton Method: BFGS

$x_{t+1} = x_t - \sigma_t H_t \nabla f(x_t)$ where H_t is an approximation of the inverse Hessian

Key idea of Quasi Newton:

successive iterates x_t, x_{t+1} and gradients $\nabla f(x_t)$, $\nabla f(x_{t+1})$ yield second order information

$$q_t \approx \nabla^2 f(x_{t+1}) p_t$$

where $p_t = x_{t+1} - x_t$ and $q_t = \nabla f(x_{t+1}) - \nabla f(x_t)$

Most popular implementation of this idea: Broyden-Fletcher-Goldfarb-Shanno (BFGS)

- default in MATLAB's **`fminunc`** and python's **`scipy.optimize.minimize`**

Conclusions

I hope it became clear...

...what are the difficulties to cope with when solving numerical optimization problems

in particular dimensionality, non-separability and ill-conditioning

...what are gradient and Hessian

...what is the difference between gradient and Newton direction

...and that adapting the step size in descent algorithms is crucial.

Derivative-Free Optimization

Derivative-Free Optimization (DFO)

DFO = blackbox optimization

Why blackbox scenario?

- gradients are not always available (binary code, no analytical model, ...)
- or not useful (noise, non-smooth, ...)

- problem domain specific knowledge is used only within the black box, e.g. within an appropriate encoding
- some algorithms are furthermore function-value-free, i.e. *invariant* wrt. monotonous transformations of f.

Derivative-Free Optimization Algorithms

- (gradient-based algorithms which approximate the gradient by finite differences)

- coordinate descent
- pattern search methods, e.g. Nelder-Mead
- surrogate-assisted algorithms, e.g. NEWUOA or other trust-region methods
- other function-value-free algorithms
  - typically stochastic
  - evolution strategies (ESs) and Covariance Matrix Adaptation Evolution Strategy (CMA-ES)
  - differential evolution
  - particle swarm optimization
  - simulated annealing
  - ...

Downhill Simplex Method by Nelder and Mead

While not happy do:

[assuming minimization of f and that $x_1, \dots, x_{n+1} \in \mathbb{R}^n$ form a simplex]

1) Order according to the values at the vertices: $f(x_1) \leq f(x_2) \leq \cdots \leq f(x_{n+1})$

2) Calculate x_o, the centroid of all points except x_{n+1}.

3) Reflection

Compute reflected point $x_r = x_o + \alpha\,(x_o - x_{n+1})$ $(\alpha > 0)$

If x_r better than second worst, but not better than best: $x_{n+1} := x_r$, and go to 1)

4) Expansion

If x_r is the best point so far: compute the expanded point

$$x_e = x_o + \gamma\,(x_r - x_o)(\gamma > 0)$$

If x_e better than x_r then $x_{n+1} \coloneqq x_e$ and go to 1)

Else $x_{n+1} \coloneqq x_r$ and go to 1)

Else (i.e. reflected point is not better than second worst) continue with 5)

5) Contraction (here: $f(x_r) \geq f(x_n)$)

Compute contracted point $x_c = x_o + \rho(x_{n+1} - x_o)$ $(0 < \rho \leq 0.5)$

If $f(x_c) < f(x_{n+1})$: $x_{n+1} \coloneqq x_c$ and go to 1)

Else go to 6)

6) Shrink

$x_i = x_1 + \sigma(x_i - x_1)$ for all $i \in \{2, \dots, n+1\}$ $(\sigma < 1)$ and go to 1)

J. A Nelder and R. Mead (1965). "A simplex method for function minimization". Computer Journal. ***7****: 308–313. doi:10.1093/comjnl/7.4.308*

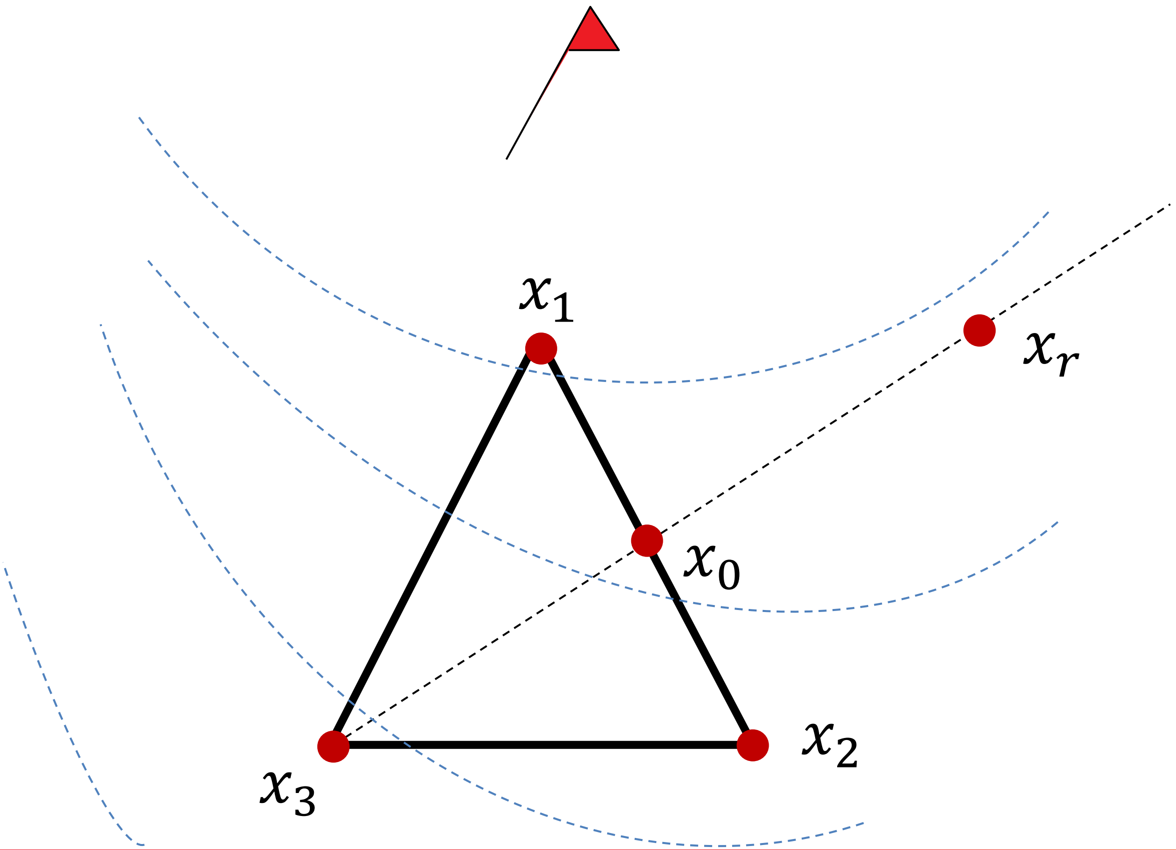
Nelder-Mead: Reflection

2) Calculate x_o, the centroid of all points except x_{n+1}.

3) Reflection

Compute reflected point $x_r = x_o + \alpha\,(x_o - x_{n+1})\ (\alpha > 0)$

If x_r better than second worst, but not better than best: $x_{n+1} := x_r$, and go to 1)



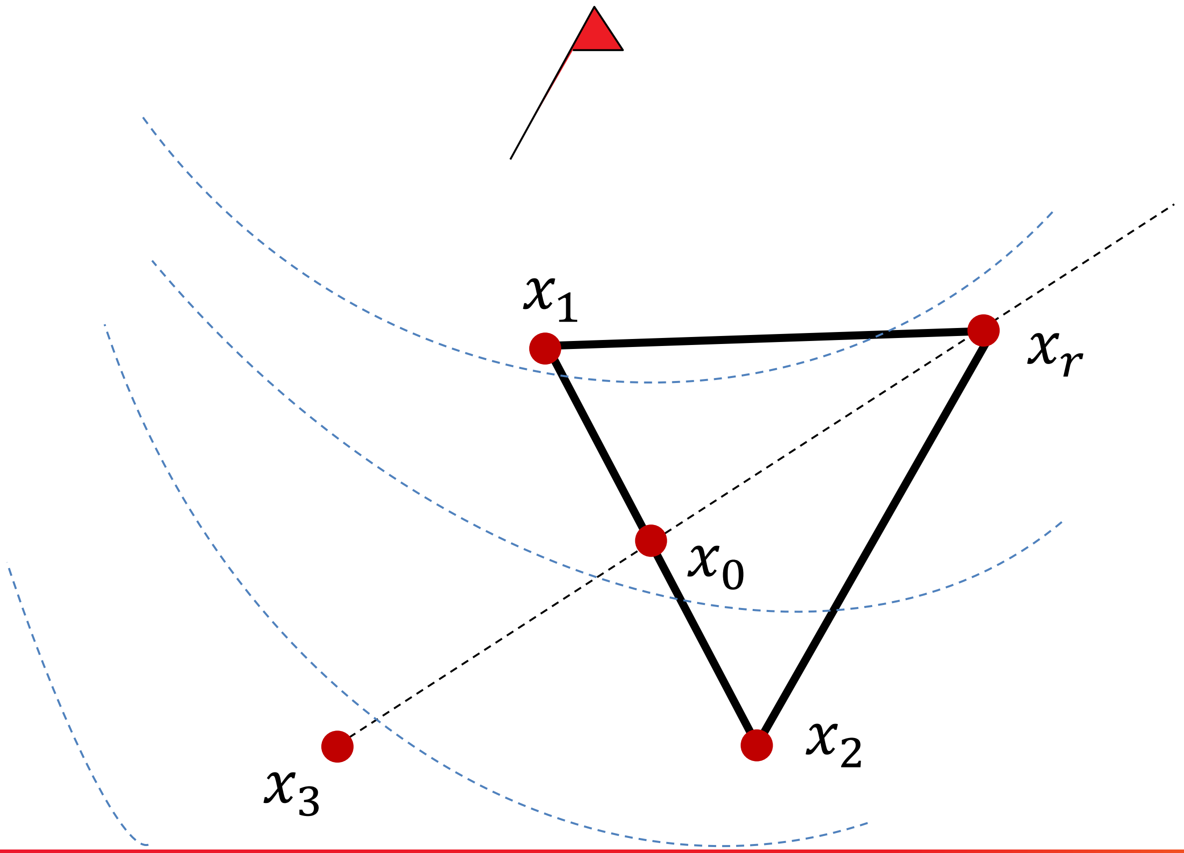
Nelder-Mead: Reflection

2) Calculate x_o, the centroid of all points except x_{n+1}.

3) Reflection

Compute reflected point $x_r = x_o + \alpha\,(x_o - x_{n+1})$ $(\alpha > 0)$

If x_r better than second worst, but not better than best: $x_{n+1} := x_r$, and go to 1)



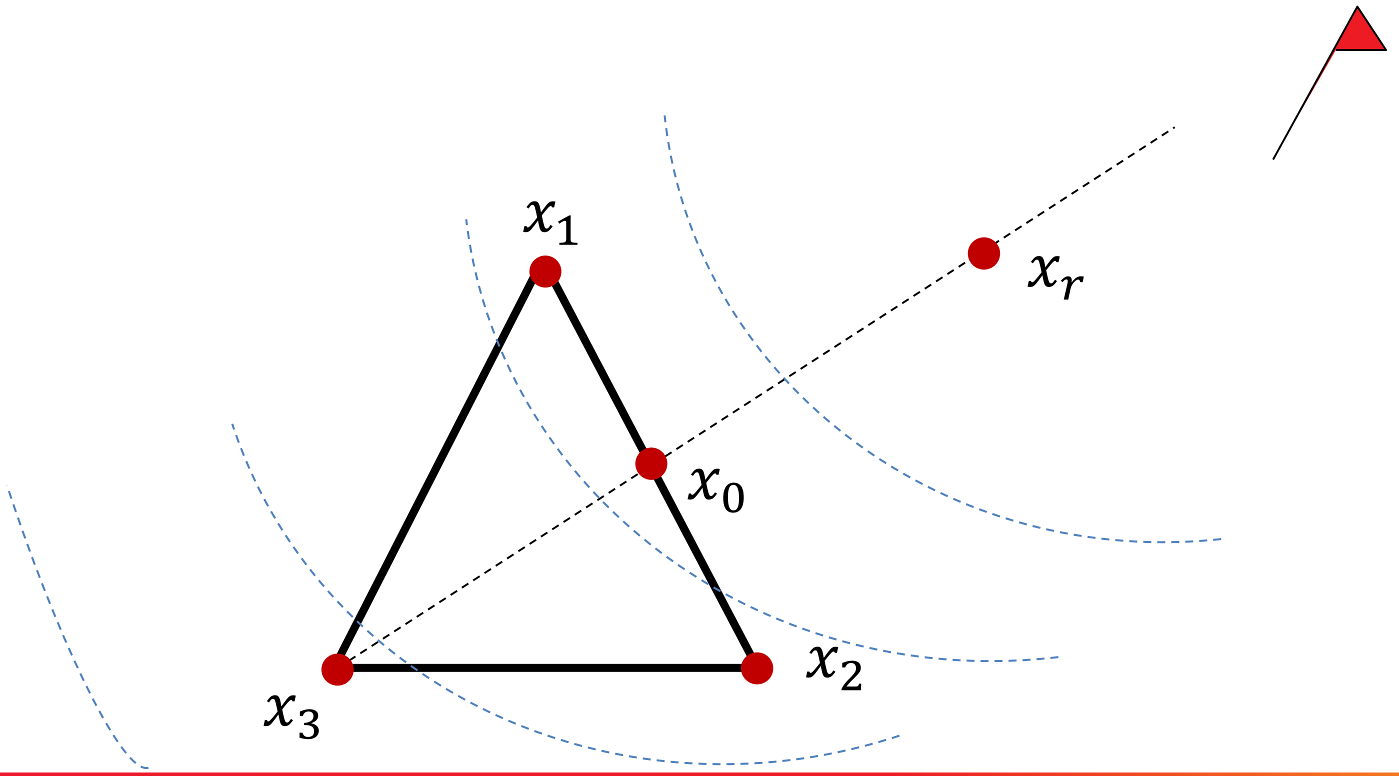
Nelder-Mead: Reflection

2) Calculate x_o, the centroid of all points except x_{n+1}.

3) Reflection

Compute reflected point $x_r = x_o + \alpha\,(x_o - x_{n+1})\ (\alpha > 0)$

If x_r better than second worst, but not better than best: $x_{n+1} := x_r$, and go to 1)



Nelder-Mead: Expansion

2) Calculate x_o, the centroid of all points except x_{n+1}.

4) Expansion

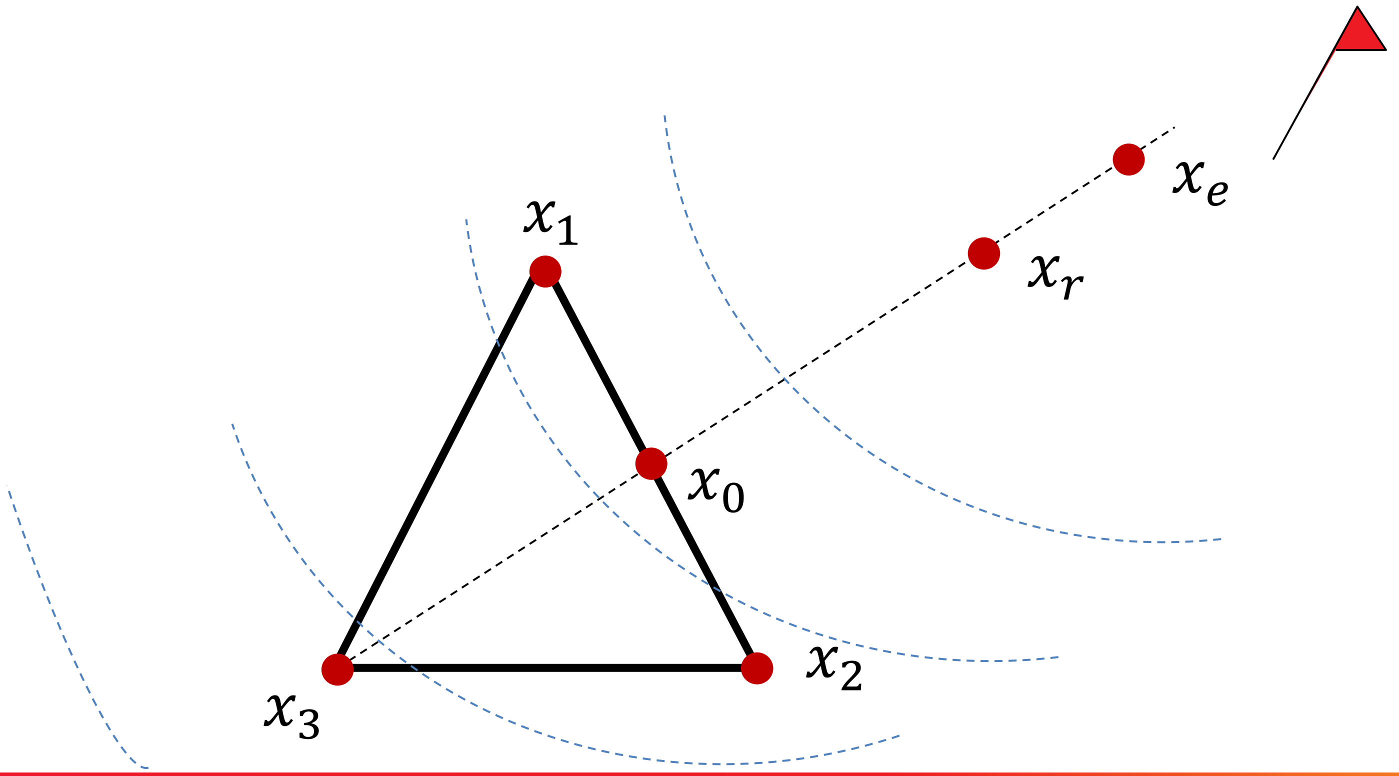
If x_r is the best point so far: compute the expanded point

$$x_e = x_o + \gamma (x_r - x_o) (\gamma > 0)$$

If x_e better than x_r then $x_{n+1} \coloneqq x_e$ and go to 1)

Else $x_{n+1} \coloneqq x_r$ and go to 1)

Else (i.e. reflected point is not better than second worst) continue with 5)



Nelder-Mead: Expansion

2) Calculate x_o, the centroid of all points except x_{n+1}.

4) Expansion

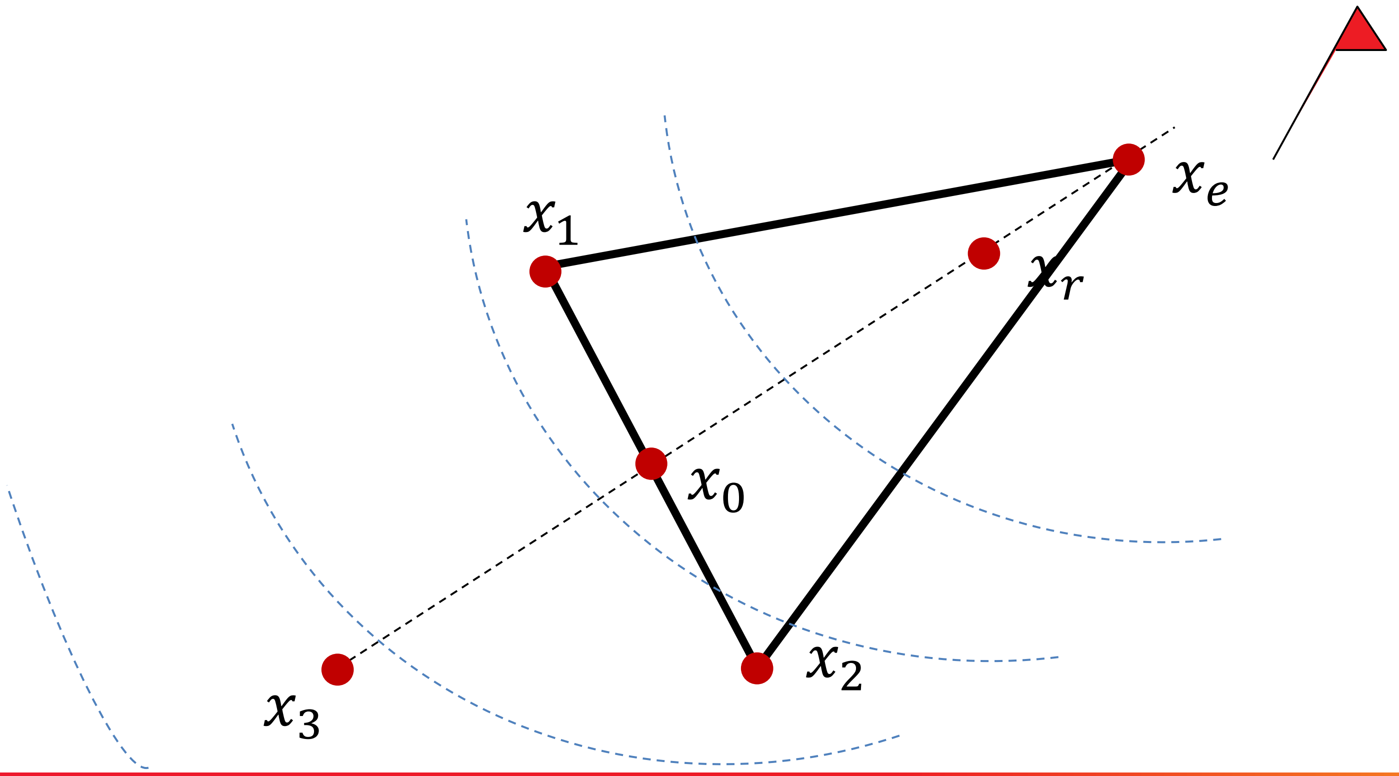
If x_r is the best point so far: compute the expanded point

$$x_e = x_o + \gamma (x_r - x_o)(\gamma > 0)$$

If x_e better than x_r then $x_{n+1} := x_e$ and go to 1)

Else $x_{n+1} := x_r$ and go to 1)

Else (i.e. reflected point is not better than second worst) continue with 5)



Nelder-Mead: Expansion

2) Calculate x_o, the centroid of all points except x_{n+1}.

4) Expansion

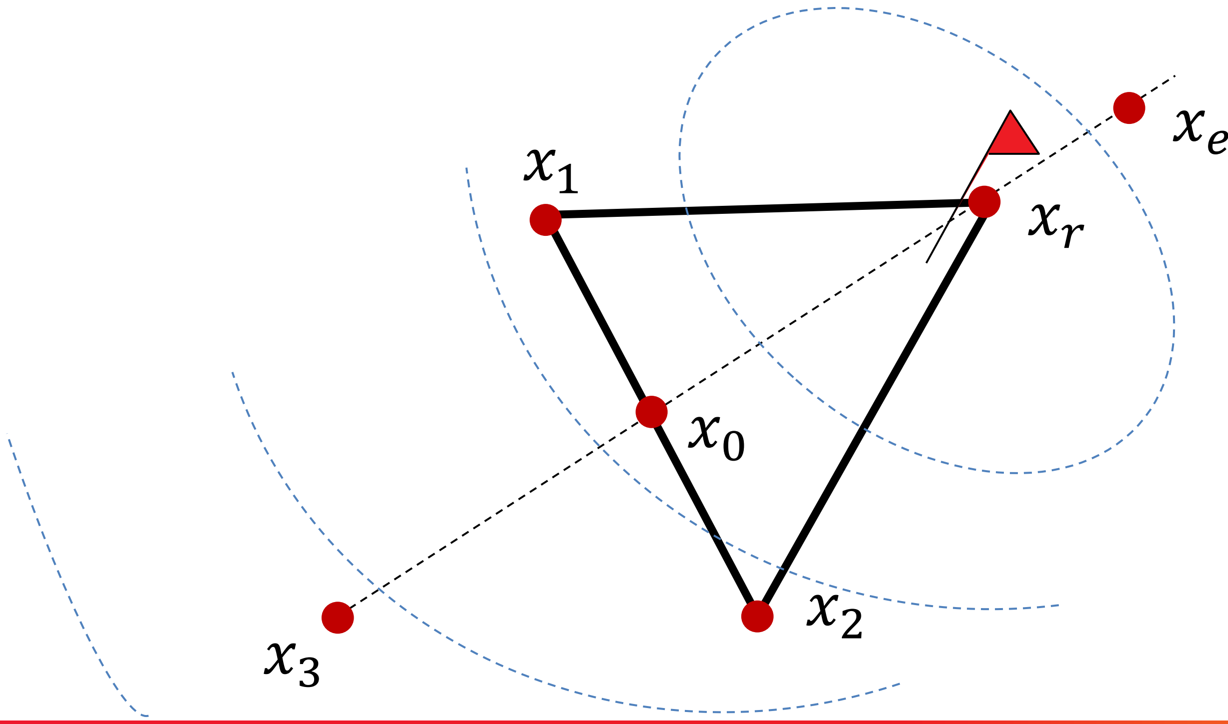
If x_r is the best point so far: compute the expanded point

$$x_e = x_o + \gamma\,(x_r - x_o)(\gamma > 0)$$

If x_e better than x_r then $x_{n+1} \coloneqq x_e$ and go to 1)

Else $x_{n+1} \coloneqq x_r$ and go to 1)

Else (i.e. reflected point is not better than second worst) continue with 5)



Nelder-Mead: Expansion

2) Calculate x_o, the centroid of all points except x_{n+1}.

4) Expansion

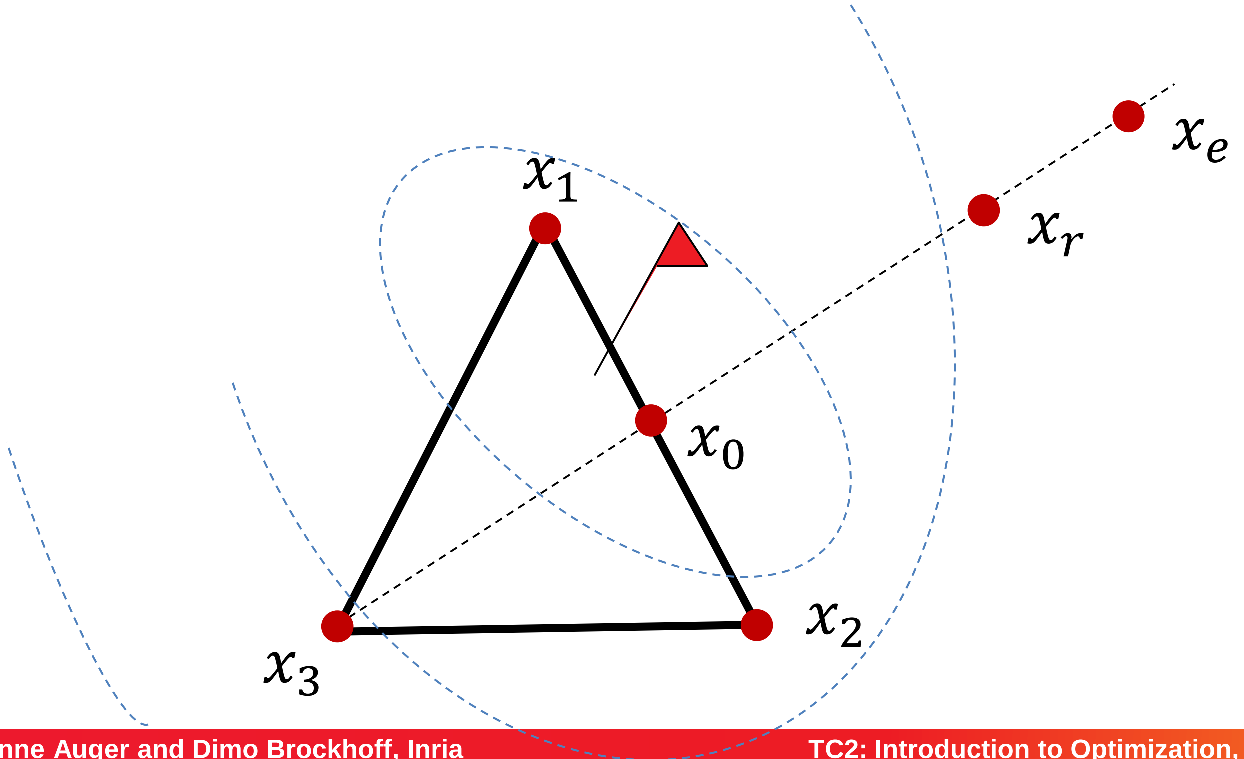
If x_r is the best point so far: compute the expanded point

$$x_e = x_o + \gamma (x_r - x_o) (\gamma > 0)$$

If x_e better than x_r then $x_{n+1} \coloneqq x_e$ and go to 1)

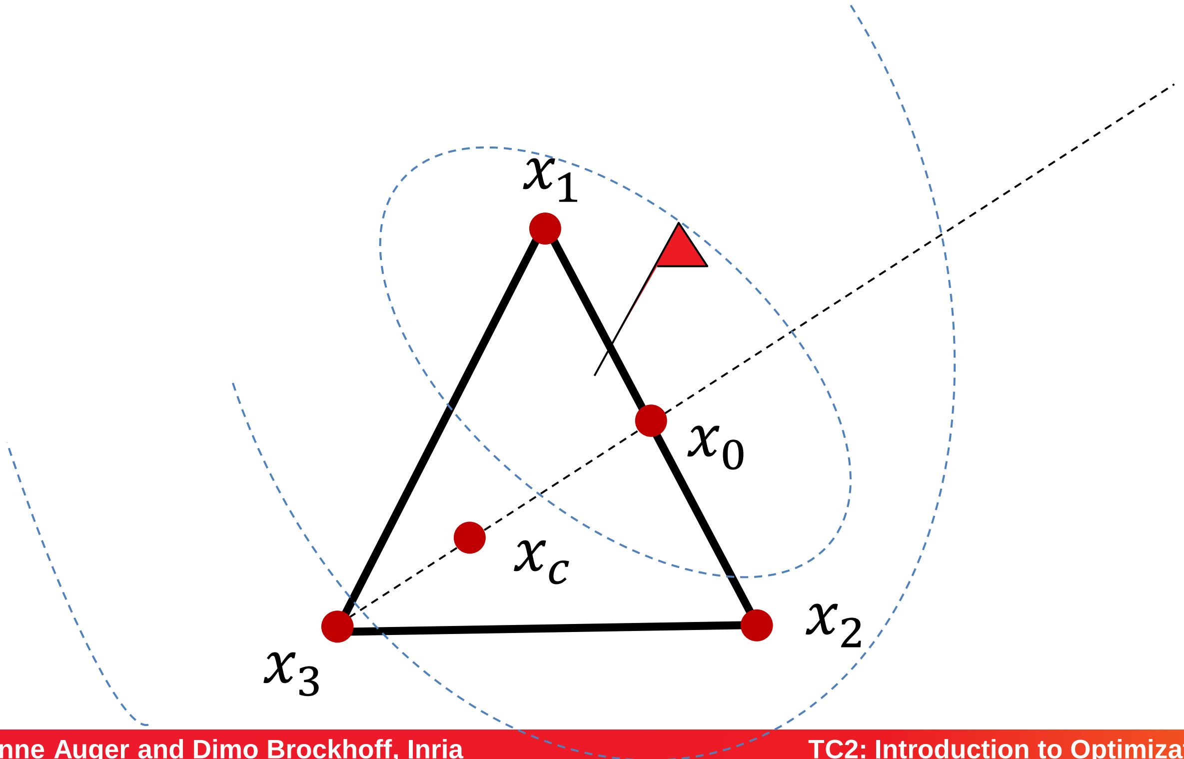
Else $x_{n+1} \coloneqq x_r$ and go to 1)

Else (i.e. reflected point is not better than second worst) continue with 5)



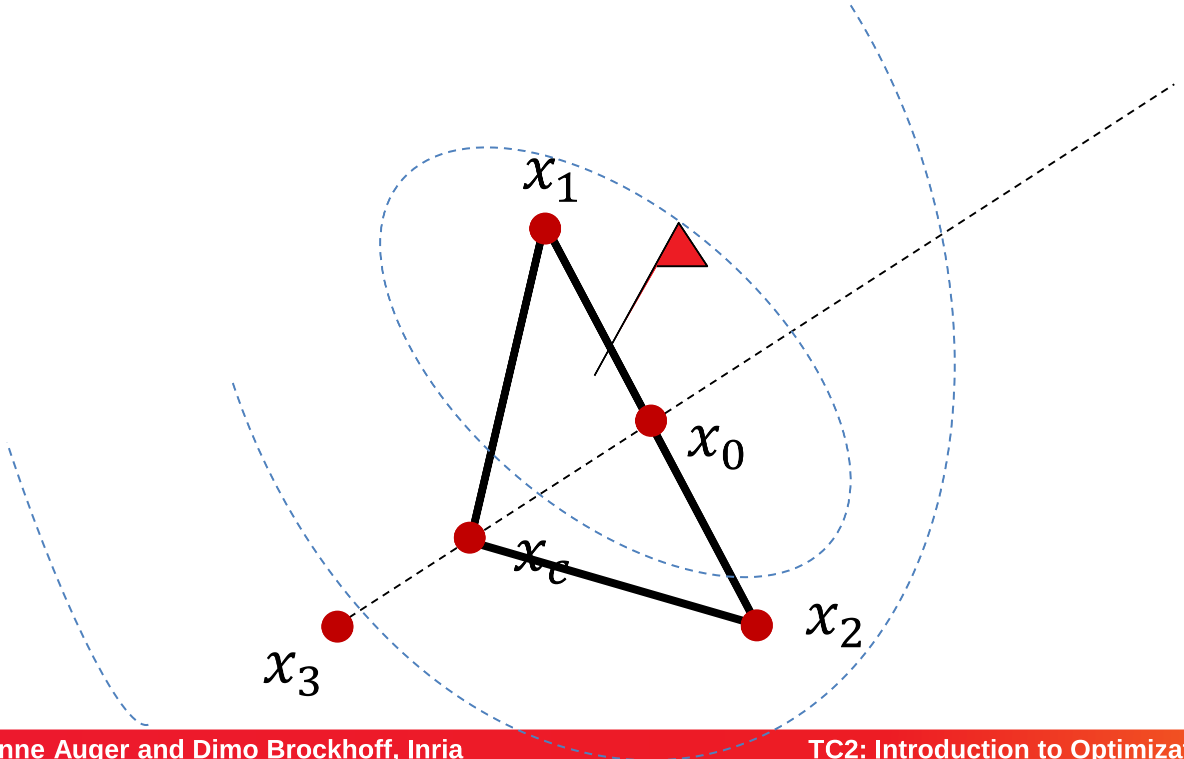
Nelder-Mead: Expansion

2) Calculate x_o, the centroid of all points except x_{n+1}.
5) Contraction (here: $f(x_r) \geq f(x_n)$)
Compute contracted point $x_c = x_o + \rho(x_{n+1} - x_o)$ $(0 < \rho \leq 0.5)$
If $f(x_c) < f(x_{n+1})$: $x_{n+1} := x_c$ and go to 1)
Else go to 6)



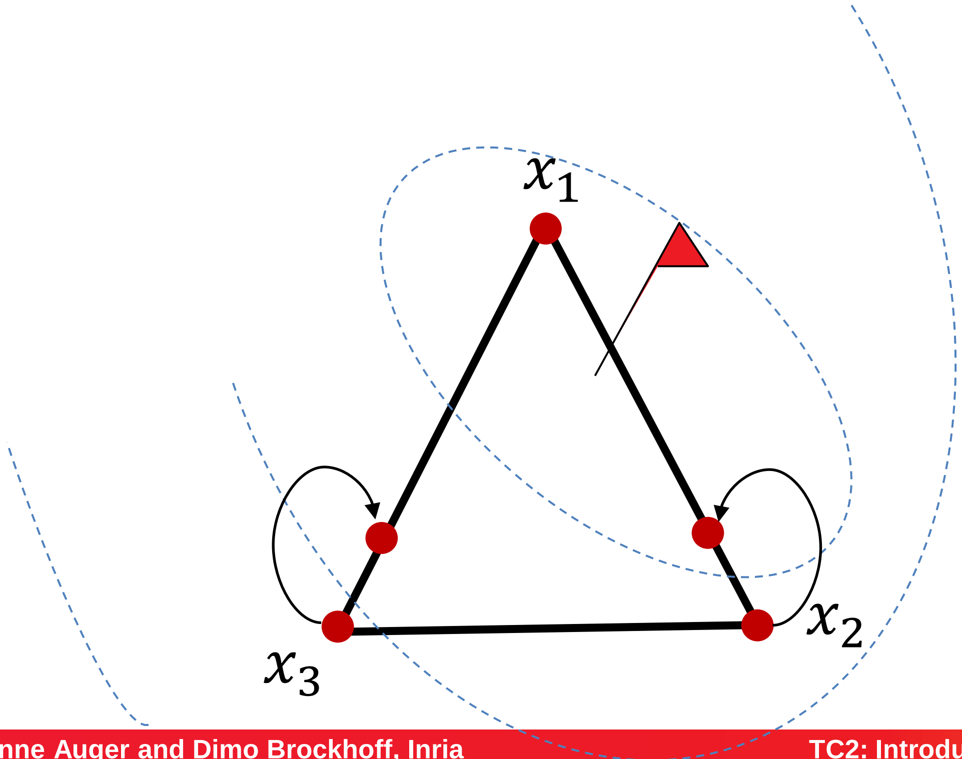
Nelder-Mead: Expansion

2) Calculate x_o, the centroid of all points except x_{n+1}.
5) Contraction (here: $f(x_r) \geq f(x_n)$)
Compute contracted point $x_c = x_o + \rho(x_{n+1} - x_o)$ $(0 < \rho \leq 0.5)$
If $f(x_c) < f(x_{n+1})$: $x_{n+1} := x_c$ and go to 1)
Else go to 6)



Nelder-Mead: Expansion

2) Calculate x_o, the centroid of all points except x_{n+1}.
6) Shrink
$x_i = x_1 + \sigma(x_i - x_1)$ for all $i \in \{2, \ldots, n+1\}$ and go to 1)

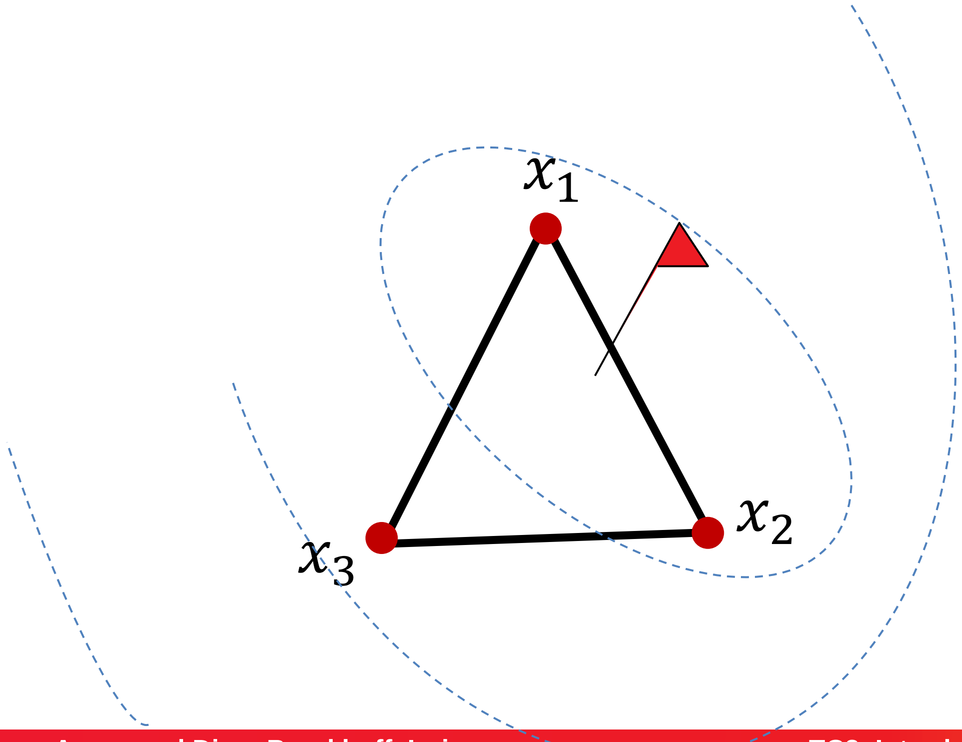


Nelder-Mead: Expansion

2) Calculate x_o, the centroid of all points except x_{n+1}.

6) Shrink

$x_i = x_1 + \sigma(x_i - x_1)$ for all $i \in \{2, \ldots, n+1\}$ and go to 1)



Nelder-Mead: Standard Parameters

- reflection parameter : $\alpha = 1$
- expansion parameter: $\gamma = 2$
- contraction parameter: $\rho = \frac{1}{2}$
- shrink paremeter: $\sigma = \frac{1}{2}$

some visualizations of example runs can be found here:
https://en.wikipedia.org/wiki/Nelder%E2%80%93Mead_method

stochastic algorithms

Stochastic Search Template

A stochastic blackbox search template to minimize $\boldsymbol{f}: \mathbb{R}^{\boldsymbol{n}} \to \mathbb{R}$

Initialize distribution parameters θ, set population size $\lambda \in \mathbb{N}$

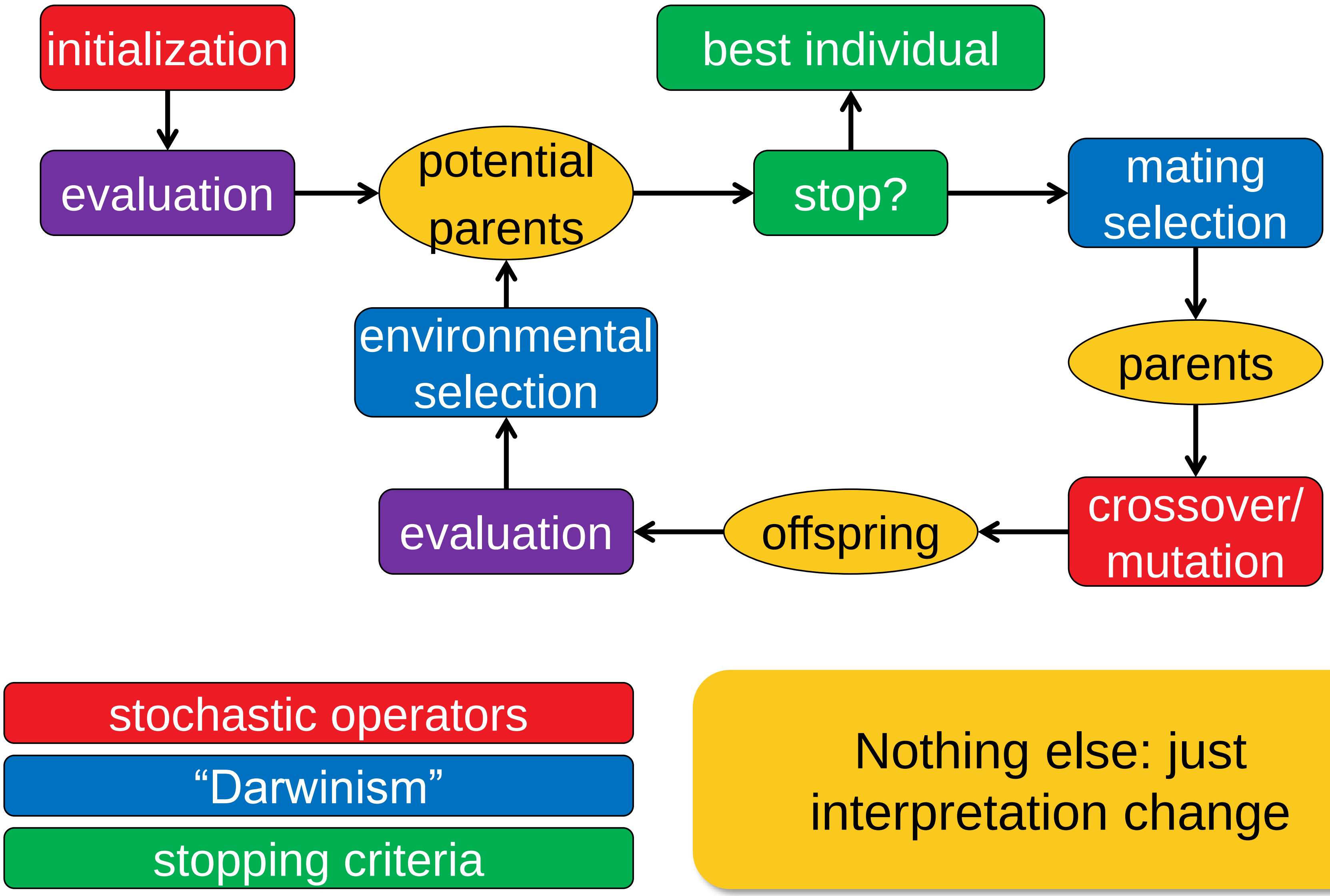
While happy do:

- Sample distribution $P(\boldsymbol{x}|\theta) \to \boldsymbol{x}_1, \dots, \boldsymbol{x}_\lambda \in \mathbb{R}^n$
- Evaluate $\boldsymbol{x}_1, \dots, \boldsymbol{x}_\lambda$ on f
- Update parameters $\theta \leftarrow F_\theta(\theta, \boldsymbol{x}_1, \dots, \boldsymbol{x}_\lambda, f(\boldsymbol{x}_1), \dots, f(\boldsymbol{x}_\lambda))$

- All depends on the choice of P and F_θ

deterministic algorithms are covered as well

- In Evolutionary Algorithms, P and F_θ are often defined implicitly via their operators.



Nothing else: just interpretation change

The CMA-ES

Input: $\boldsymbol{m} \in \mathbb{R}^n$, $\sigma \in \mathbb{R}_+$, λ
Initialize: $\mathbf{C} = \mathbf{I}$, and $\boldsymbol{p}_c = \mathbf{0}$, $\boldsymbol{p}_\sigma = \mathbf{0}$,
Set: $c_c \approx 4/n$, $c_\sigma \approx 4/n$, $c_1 \approx 2/n^2$, $c_\mu \approx \mu_w/n^2$, $c_1 + c_\mu \leq 1$, $d_\sigma \approx 1 + \sqrt{\frac{\mu_w}{n}}$,
and $w_{i=1\ldots\lambda}$ such that $\mu_w = \frac{1}{\sum_{i=1}^{\mu} {w_i}^2} \approx 0.3\,\lambda$

While not terminate

$$\boldsymbol{x}_i = \boldsymbol{m} + \sigma\,\boldsymbol{y}_i, \quad \boldsymbol{y}_i \sim \mathcal{N}_i(\mathbf{0}, \mathbf{C}), \quad \text{for } i = 1, \ldots, \lambda \qquad \text{sampling}$$

$$\boldsymbol{m} \leftarrow \sum_{i=1}^{\mu} w_i\,\boldsymbol{x}_{i:\lambda} = \boldsymbol{m} + \sigma\,\boldsymbol{y}_w \quad \text{where } \boldsymbol{y}_w = \sum_{i=1}^{\mu} w_i\,\boldsymbol{y}_{i:\lambda} \qquad \text{update mean}$$

$$\boldsymbol{p}_c \leftarrow (1 - c_c)\,\boldsymbol{p}_c + \mathbb{1}_{\{\|\boldsymbol{p}_\sigma\| < 1.5\sqrt{n}\}}\sqrt{1 - (1 - c_c)^2}\sqrt{\mu_w}\,\boldsymbol{y}_w \qquad \text{cumulation for } \mathbf{C}$$

$$\boldsymbol{p}_\sigma \leftarrow (1 - c_\sigma)\,\boldsymbol{p}_\sigma + \sqrt{1 - (1 - c_\sigma)^2}\sqrt{\mu_w}\,\mathbf{C}^{-\frac{1}{2}}\boldsymbol{y}_w \qquad \text{cumulation for } \sigma$$

$$\mathbf{C} \leftarrow (1 - c_1 - c_\mu)\,\mathbf{C} + c_1\,\boldsymbol{p}_c\boldsymbol{p}_c^{\mathrm{T}} + c_\mu \sum_{i=1}^{\mu} w_i\,\boldsymbol{y}_{i:\lambda}\boldsymbol{y}_{i:\lambda}^{\mathrm{T}} \qquad \text{update } \mathbf{C}$$

$$\sigma \leftarrow \sigma \times \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|\boldsymbol{p}_\sigma\|}{\mathsf{E}\|\mathcal{N}(\mathbf{0},\mathbf{I})\|} - 1\right)\right) \qquad \text{update of } \sigma$$

Not covered on this slide: termination, restarts, useful output, boundaries and encoding

The CMA-ES

Input: $m \in \mathbb{R}^n$, $\sigma \in \mathbb{R}_+$, λ
Initialize: $\mathbf{C} = \mathbf{I}$, and $p_c = \mathbf{0}$, $p_\sigma = \mathbf{0}$,
Set: $c_c \approx 4/n$, $c_\sigma \approx 4/n$, $c_1 \approx 2/n^2$, $c_\mu \approx \mu_w/n^2$, $c_1 + c_\mu \leq 1$, $d_\sigma \approx 1 + \sqrt{\frac{\mu_w}{n}}$,
and $w_{i=1\ldots\lambda}$ such that $\mu_w = \frac{1}{\sum_{i=1}^{\mu} {w_i}^2} \approx 0.3\,\lambda$

While not terminate

$$x_i = m + \sigma y_i, \quad y_i \sim \mathcal{N}_i(\mathbf{0}, \mathbf{C}), \quad \text{for } i = 1, \ldots, \lambda$$ sampling

$$m \leftarrow \sum_{i=1}^{\mu} w_i x_{i:\lambda} = m + \sigma y_w \quad \text{where } y_w = \sum_{i=1}^{\mu} w_i y_{i:\lambda}$$ update mean

$$p_c \leftarrow (1 - c_c)\, p_c + \mathbb{1}_{\{\|p_\sigma\| < 1.5\sqrt{n}\}} \sqrt{1 - (1 - c_c)^2} \sqrt{\mu_w}\, y_w$$ cumulation for $\mathbf{C}$

$$p_\sigma \leftarrow (1 - c_\sigma)\, p_\sigma + \sqrt{1 - (1 - c_\sigma)^2} \sqrt{\mu_w}\, \mathbf{C}^{-\frac{1}{2}} y_w$$ cumulation for σ

$$\mathbf{C} \leftarrow (1 - c_1 - c_\mu)\, \mathbf{C} + c_1\, p_c {p_c}^{\mathrm{T}} + \ldots$$ update C

$$\sigma \leftarrow \sigma \times \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|p_\sigma\|}{\mathsf{E}\|\mathcal{N}(\mathbf{0},\mathbf{I})\|} - 1\right)\right)$$

Not covered on this slide: terminati[...]
encoding

Goal of next lecture:
Understand the main principles of this state-of-the-art algorithm.