

# DERIVATIVE-FREE OPTIMIZATION

<http://www.lri.fr/~teytaud/dfo.pdf>

(or Quentin's web page ?)

**Olivier Teytaud**

Inria Tao, en visite dans la belle ville de Liège

using also Slides from A. Auger



***The next slide is the most important  
of all.***

Olivier Teytaud

Inria Tao, en visite dans la belle ville de Liège



**In case of trouble,  
Interrupt me.**

Olivier Teytaud

Inria Tao, en visite dans la belle ville de Liège



**In case of trouble,  
Interrupt me.**

Further discussion needed:

- R82A, Montefiore institute
- [olivier.teytaud@inria.fr](mailto:olivier.teytaud@inria.fr)
- or after the lessons (the 25<sup>th</sup>, not the 18th)

Olivier Teytaud

Inria Tao, en visite dans la belle ville de Liège

## Evolutionary Computation

## Optimization

---

individual, offspring, parent	$\longleftrightarrow$	candidate solution decision variables design variables object variables
population	$\longleftrightarrow$	set of candidate solutions
fitness function	$\longleftrightarrow$	objective function loss function cost function
generation	$\longleftrightarrow$	iteration

- I. Optimization and DFO**
- II. Evolutionary algorithms
- III. From math. programming
- IV. Using machine learning
- V. Conclusions

Olivier Teytaud

Inria Tao, en visite dans la belle ville de Liège

# Derivative-free optimization of $f$

Olivier Teytaud

Inria Tao, en visite dans la belle ville de Liège

# Derivative-free optimization of $f$

$$x_1 = \text{Opt}() \text{ and } \forall n \in \{1, \dots, N - 1\},$$

# Derivative-free optimization of $f$

$$x_1 = \text{Opt}() \text{ and } \forall n \in \{1, \dots, N - 1\},$$

$$x_{n+1} = \text{Opt}(x_1, f(x_1), \dots, x_n, f(x_n)).$$

No gradient !

Only depends on the  $x$ 's and  $f(x)$ 's

# Derivative-free optimization of $f$

$$x_1 = \text{Opt}() \text{ and } \forall n \in \{1, \dots, N - 1\},$$

$$x_{n+1} = \text{Opt}(x_1, f(x_1), \dots, x_n, f(x_n)).$$

Why derivative free optimization ?

# Derivative-free optimization of $f$

$$x_1 = \text{Opt}() \text{ and } \forall n \in \{1, \dots, N - 1\},$$

$$x_{n+1} = \text{Opt}(x_1, f(x_1), \dots, x_n, f(x_n)).$$

Why derivative free optimization ?

- Ok, it's slower

# Derivative-free optimization of $f$

$$x_1 = \text{Opt}() \text{ and } \forall n \in \{1, \dots, N - 1\},$$

$$x_{n+1} = \text{Opt}(x_1, f(x_1), \dots, x_n, f(x_n)).$$

Why derivative free optimization ?

- Ok, it's slower
- But sometimes you have no derivative

# Derivative-free optimization of $f$

$$x_1 = \text{Opt}() \text{ and } \forall n \in \{1, \dots, N - 1\},$$

$$x_{n+1} = \text{Opt}(x_1, f(x_1), \dots, x_n, f(x_n)).$$

Why derivative free optimization ?

- Ok, it's slower
- But sometimes you have no derivative
- It's simpler (by far)  $\implies$  less bugs

# Derivative-free optimization of $f$

$$x_1 = \text{Opt}() \text{ and } \forall n \in \{1, \dots, N - 1\},$$

$$x_{n+1} = \text{Opt}(x_1, f(x_1), \dots, x_n, f(x_n)).$$

Why derivative free optimization ?

- Ok, it's slower
- But sometimes you have no derivative
- It's simpler (by far)
- It's more robust (to noise, to strange functions...)

# Derivative-free optimization of $f$

$x_1 = \text{Opt}()$  and  $\forall n \in \{1, \dots, N-1\}$ ,

$x_n$

## Optimization algorithms

==> Newton

==> Quasi-Newton (BFGS)

==> Gradient descent

==> ...

(S...)

# Derivative-free optimization of $f$

$x_1 = Opt()$  and  $\forall n \in \{1, \dots, N - 1\}$ ,

$x_n$

Optimization algorithms

Derivative-free optimization  
(don't need gradients)

- It's (not) suitable for (black-box functions...)

# Derivative-free optimization of $f$

$x_1 = Opt()$  and  $\forall n \in \{1, \dots, N - 1\}$ ,

$x_n$

Optimization algorithms

Derivative-free optimization

Comparison-based optimization

(coming soon),

just needing comparisons,

including evolutionary algorithms

- It's (including evolutionary algorithms and other functions...)

- I. Optimization and DFO
- II. Evolutionary algorithms**
- III. From math. programming
- IV. Using machine learning
- V. Conclusions

Olivier Teytaud

Inria Tao, en visite dans la belle ville de Liège

## II. Evolutionary algorithms

*a. Fundamental elements*

b. Algorithms

c. Math. analysis

## Preliminaries:

- *Gaussian distribution*

- Multivariate Gaussian distribution

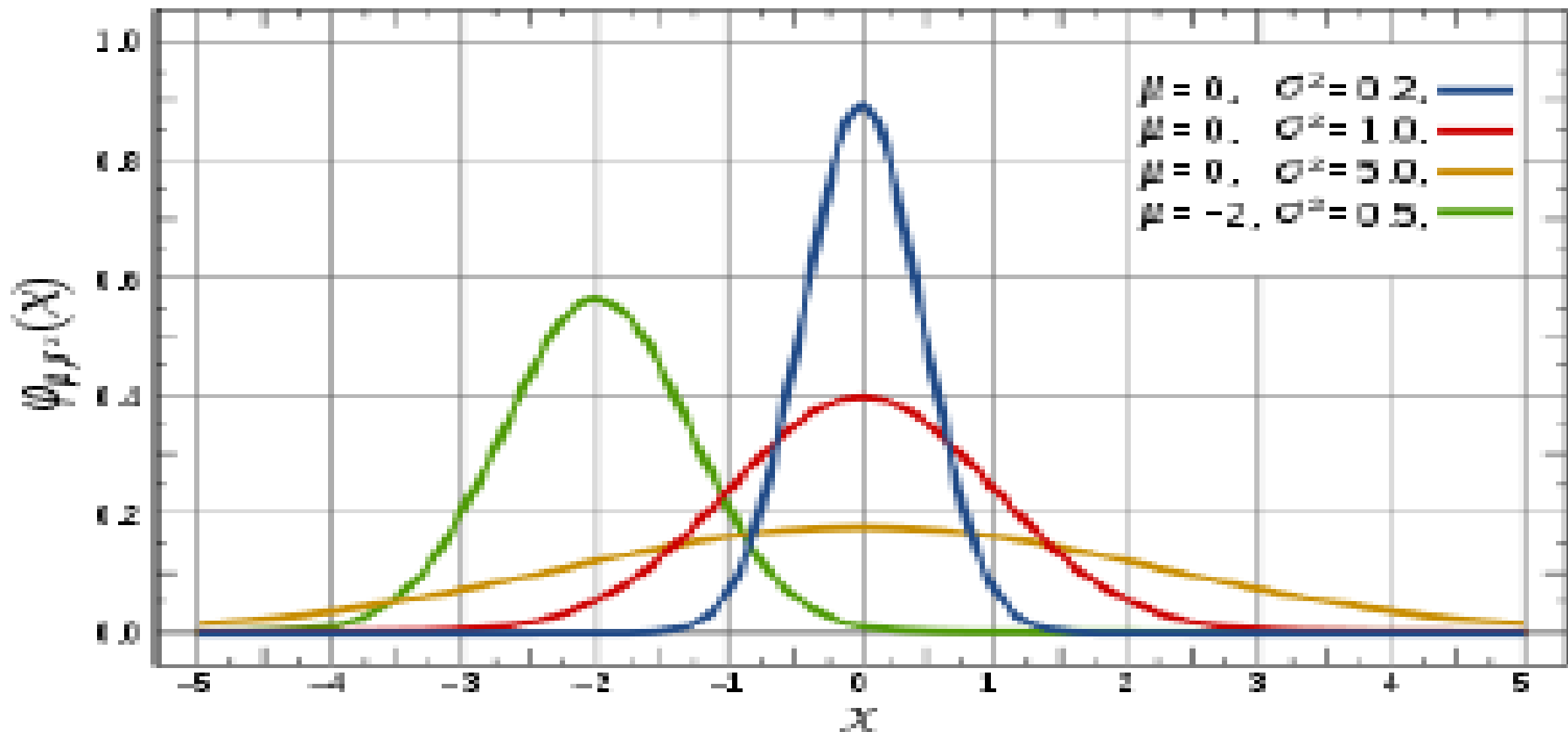
- Non-isotropic Gaussian distribution

- Markov chains

==> for theoretical analysis

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

## Probability density function



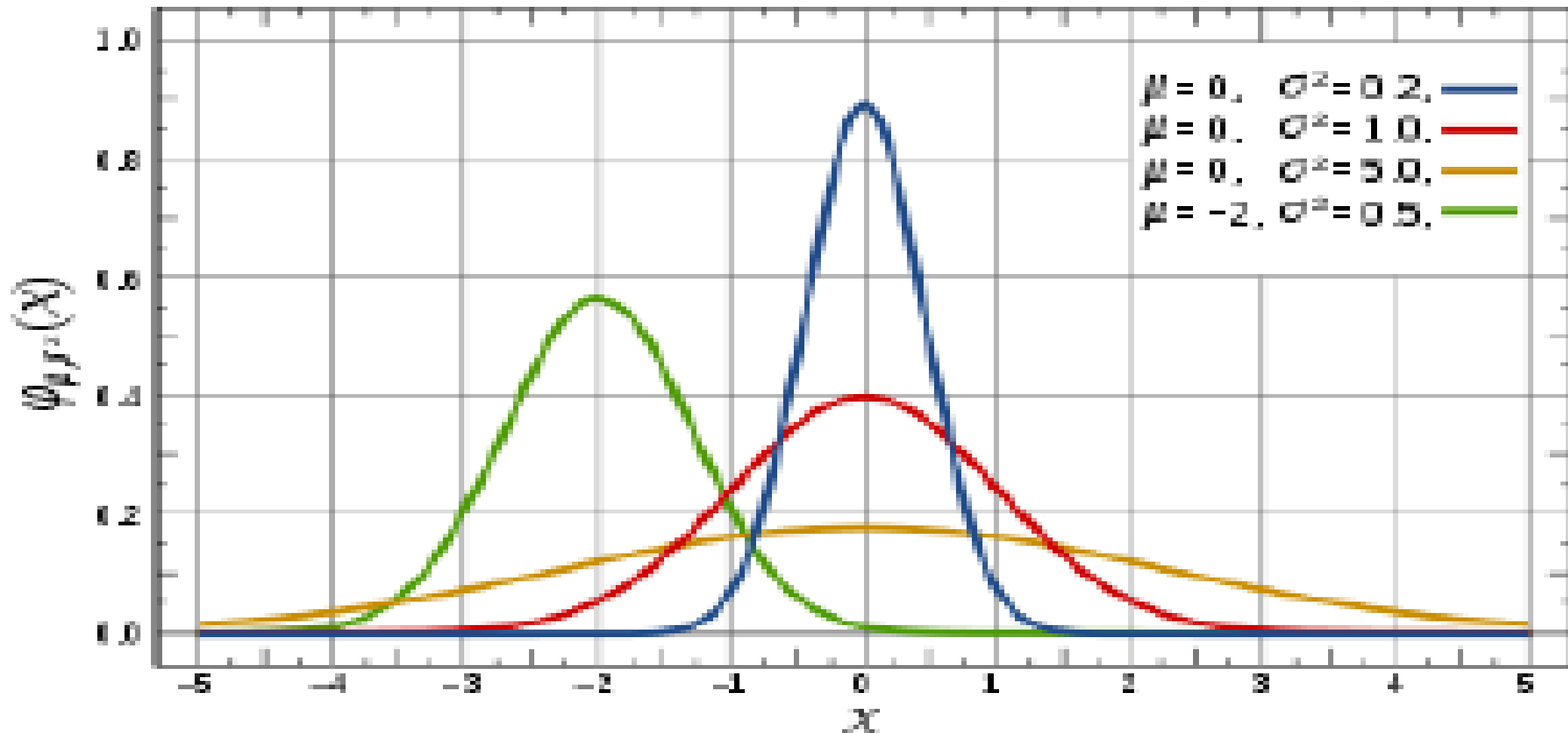
The red line is the standard normal distribution

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

K exp( - p(x) ) with

- p(x) a degree 2 polynomial (neg. dom coef)
- K a normalization constant

## Probability density function



The red line is the standard normal distribution

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

K exp( - p(x) ) with

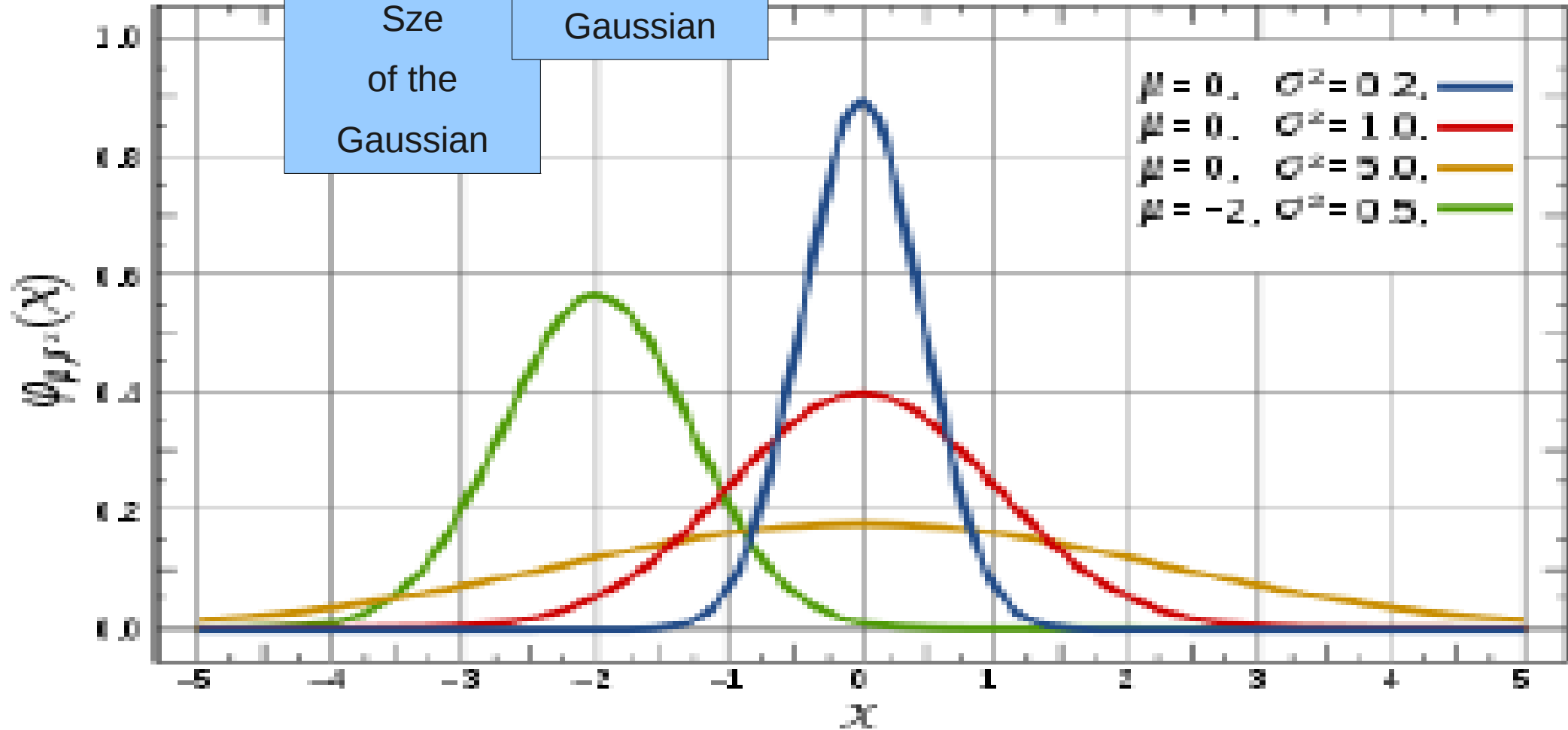
- p(x) a degree 2 polynomial (neg. dom coef)
- K a normalization constant

$\sigma^2$

Size  
of the  
Gaussian

Translation  
of the  
Gaussian

density function

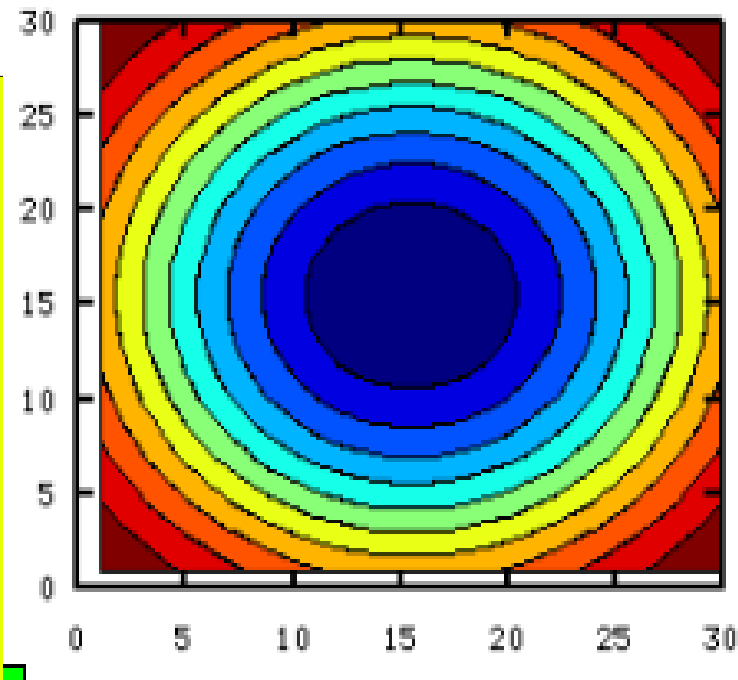


The red line is the standard normal distribution

## Preliminaries:

- Gaussian distribution
- *Multivariate Gaussian distribution*
- Non-isotropic Gaussian distribution

## Isotropic case:



==> general case: density =  $K \exp(- \|x - \mu\|^2 / 2\sigma^2)$

==> level sets are rotationally invariant

==> completely defined by  $\mu$  and  $\sigma$

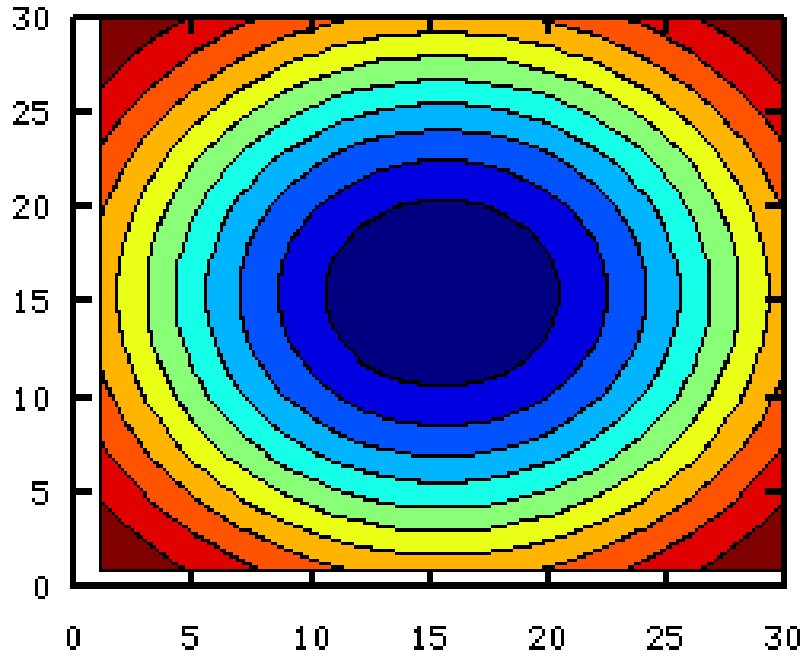
(do you understand why  $K$  is fixed by  $\sigma$ ?)

==> “isotropic” Gaussian

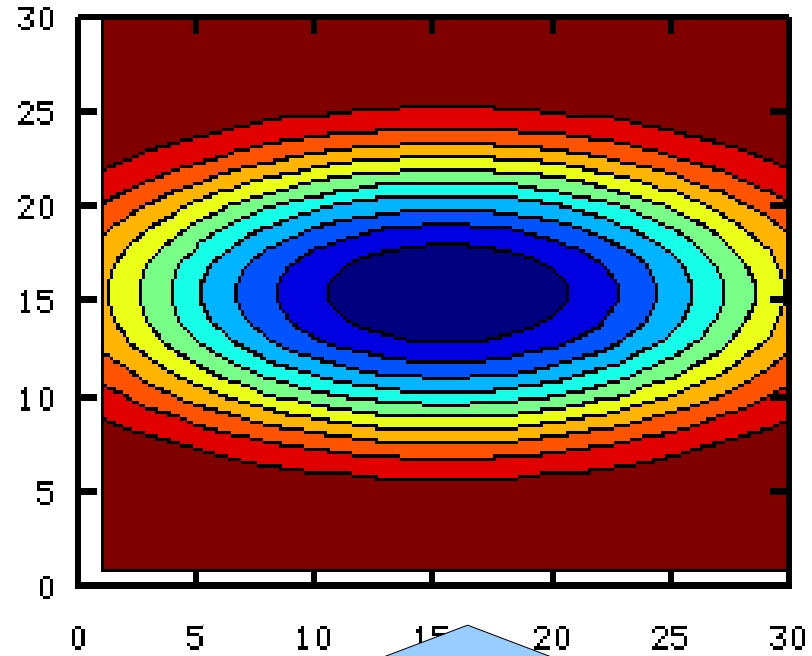
## Preliminaries:

- Gaussian distribution
- Multivariate Gaussian distribution
- *Non-isotropic Gaussian distribution*

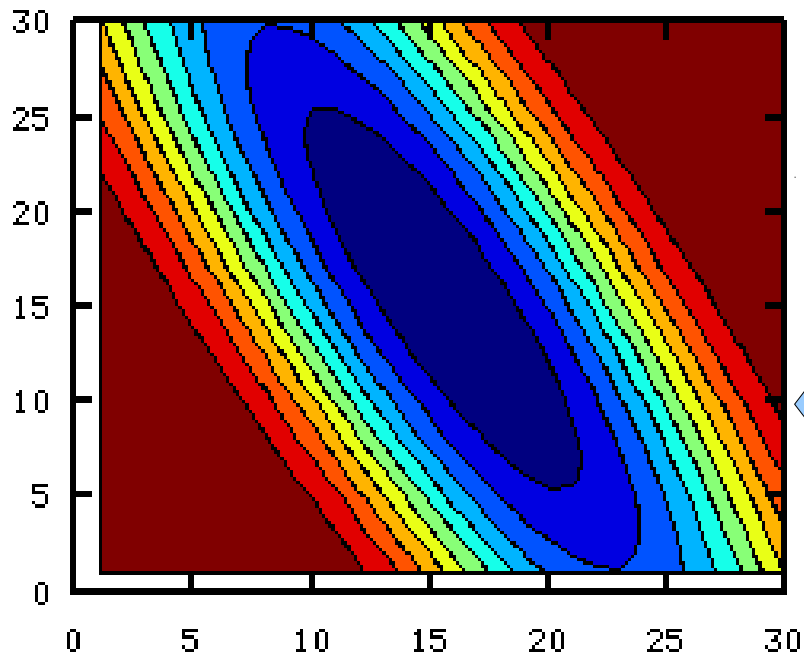
Isotropic multivariate Gaussian



Axis-parallel multivariate Gaussian



Multivariate Gaussian



Step-size different  
on each axis

$K \exp(-p(x))$  with

- $p(x)$  a quadratic form ( $\rightarrow +$  infinity)
- $K$  a normalization constant

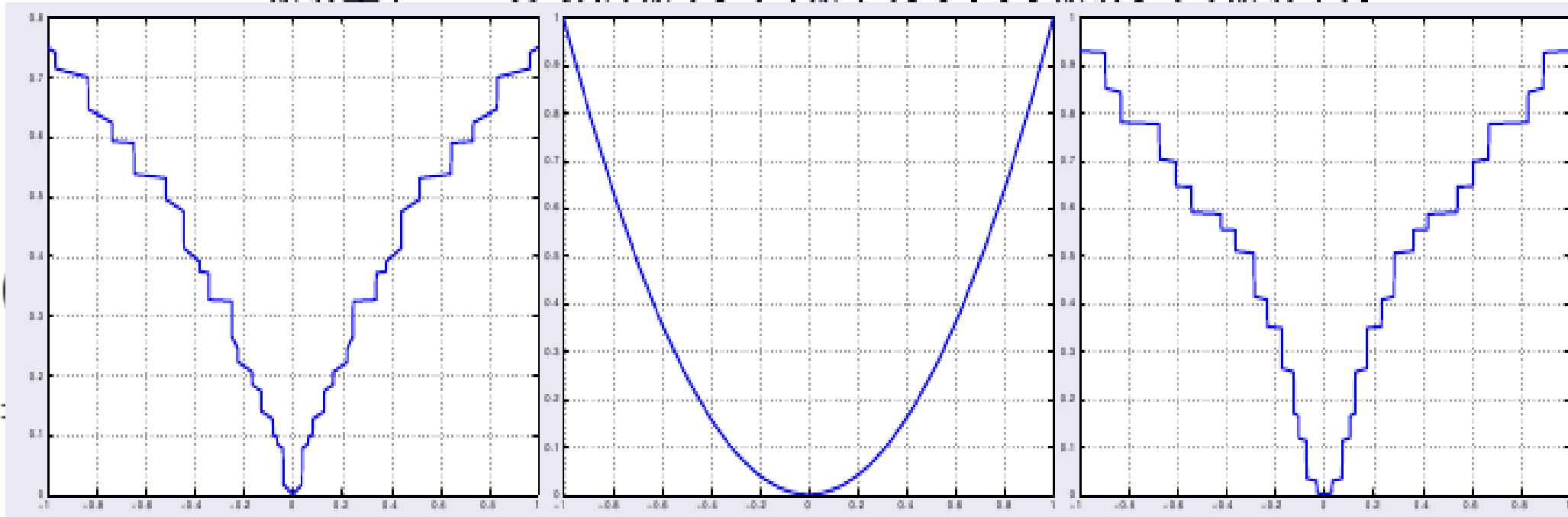
## Notions that we will see:

- Evolutionary algorithm
- Cross-over
- Truncation selection / roulette wheel
- Linear / log-linear convergence
- Estimation of Distribution Algorithm
- EMNA
- Self-adaptation
- (1+1)-ES with  $1/5^{\text{th}}$  rule
- Voronoi representation
- Non-isotropy

# Comparison-based optimization

Observation: we want robustness w.r.t that:

$$\mathcal{X}_{m+1} \equiv \text{OPT}(\mathcal{X}_1, T(\mathcal{X}_1), \dots, \mathcal{X}_m, T(\mathcal{X}_m)).$$



# Comparison-based optimization

$$x_1 = \text{Opt}() \qquad y_i = f(x_i)$$

$$\forall n \in \{1, \dots, N - 1\},$$

$$x_{n+1} = \text{Opt}(x_1, f(x_1), \dots, x_n, f(x_n)).$$

is comparison-based if

$$(\forall i, j, \text{sign}(y_i - y_j) = \text{sign}(y'_i - y'_j))$$

$$\implies \text{Opt}(x_1, y_1, \dots, x_n, y_n) = \text{Opt}(x_1, y'_1, \dots, x_n, y'_n)$$

# Population-based comparison-based algorithms ?

$$X(1) = (x(1,1), x(1,2), \dots, x(1,\lambda)) = \text{Opt}()$$

$$X(2) = (x(2,1), x(2,2), \dots, x(2,\lambda)) = \text{Opt}(x(1), \text{signs of diff})$$

...

...

...

$$x(n) = (x(n,1), x(n,2), \dots, x(n,\lambda)) = \text{Opt}(x(n-1), \text{signs of diff})$$

$\Rightarrow$  let's write it for  $\lambda=2$ .

# Population-based comparison-based algorithms ?

$$x(1) = (x(1,1), x(1,2)) = \text{Opt}()$$

$$x(2) = (x(2,1), x(2,2)) = \text{Opt}(x(1), \text{sign}(y(1,1) - y(1,2)))$$

...

...

...

$$x(n) = (x(n,1), x(n,2)) = \text{Opt}(x(n-1), \text{sign}(y(n-1,1) - y(n-1,2)))$$

$$\text{with } y(i,j) = f(x(i,j))$$

# Population-based comparison-based algorithms ?

Abstract notations:  $x(i)$  is a population

$$x(1) = \text{Opt}()$$

$$x(2) = \text{Opt}(x(1), \text{sign}(y(1,1)-y(1,2)))$$

...

...

...

$$x(n) = \text{Opt}(x(n-1), \text{sign}(y(n-1,1)-y(n-1,2)))$$

# Population-based comparison-based algorithms ?

Abstract notations:  $x(i)$  is a population,  $l(i)$  is an internal state of the algorithm.

$$x(1), l(1) = \text{Opt}()$$

$$x(2), l(2) = \text{Opt}(x(1), \text{sign}(y(1,1)-y(1,2)), l(1) )$$

...

...

...

$$x(n), l(n) = \text{Opt}(x(n-1), \text{sign}(y(n-1,1)-y(n-1,2)) , l(n-1))$$

# Population-based comparison-based algorithms ?

Abstract notations:  $x(i)$  is a population,  $l(i)$  is an internal state of the algorithm.

$$x(1), l(1) = \text{Opt}()$$

$$x(2), l(2) = \text{Opt}(x(1), \Delta(1), l(1))$$

...

...

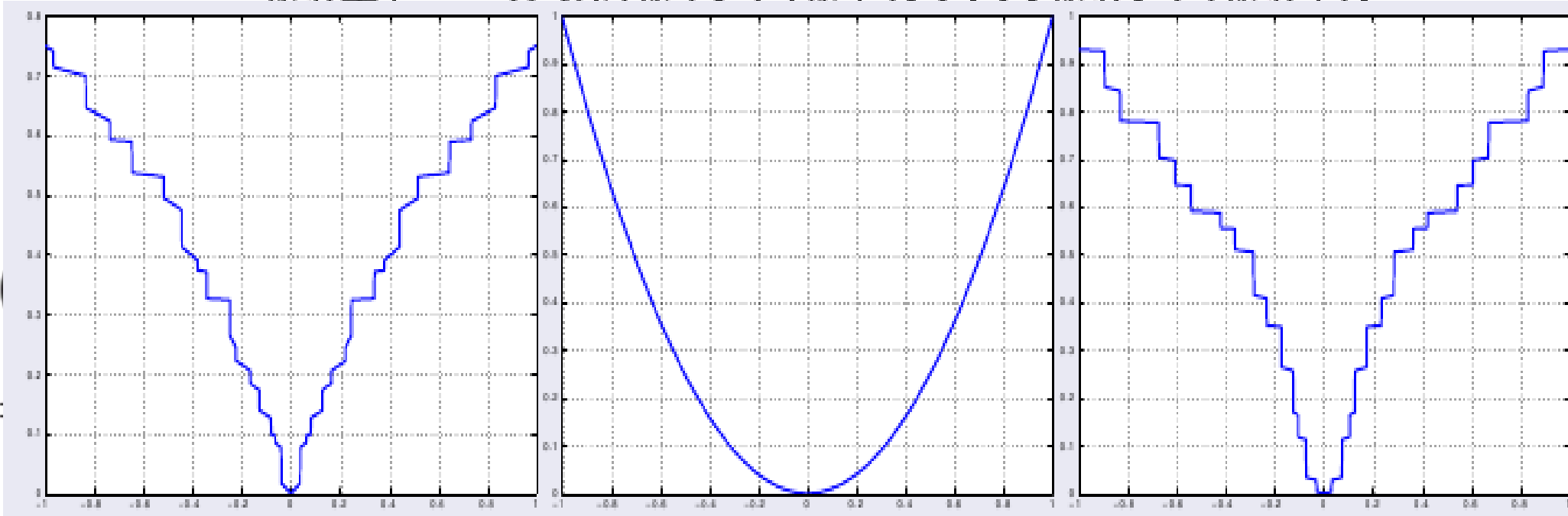
...

$$x(n), l(n) = \text{Opt}(x(n-1), \Delta(n-1), l(n-1))$$

# Comparison-based optimization

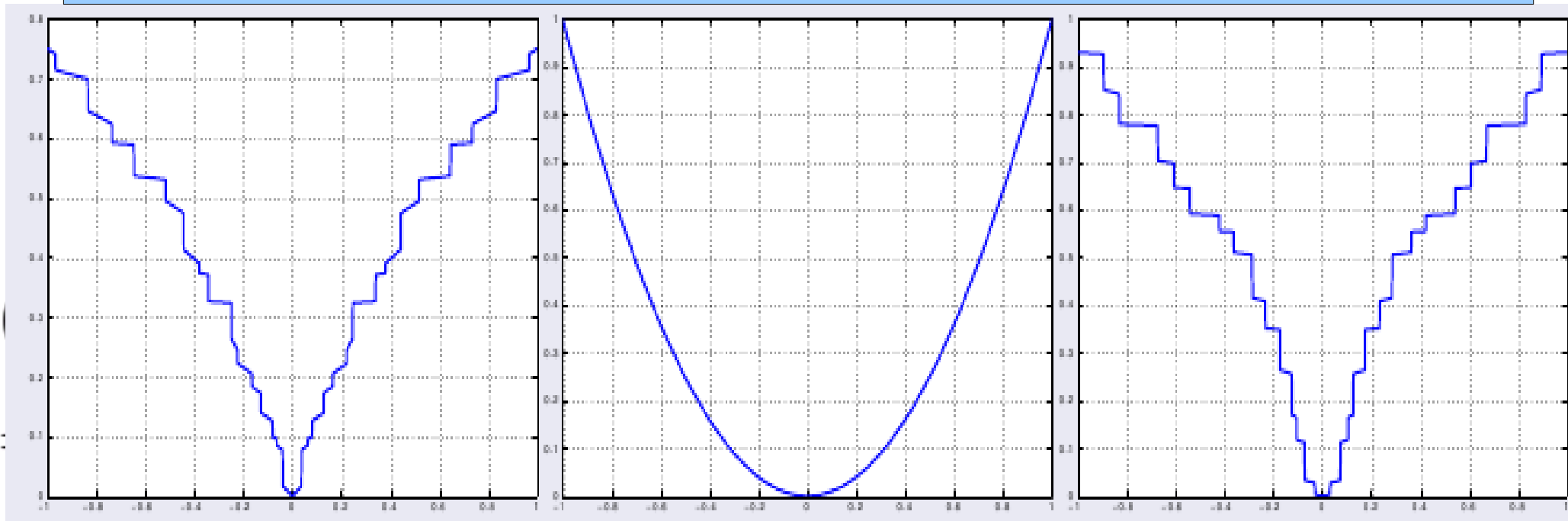
==> Same behavior on many functions

$$\mathcal{X}_{m+1} = \text{OPT}(\mathcal{X}_1, f(\mathcal{X}_1), \dots, \mathcal{X}_m, f(\mathcal{X}_m)).$$



# Comparison-based optimization

==> Same behavior on many functions



Quasi-Newton methods very poor on this.

# Why comparison-based algorithms ?

==> more robust

==> this can be mathematically

formalized: comparison-based opt.

are **slow** ( $d \log \|x_n - x^*\|/n \sim \text{constant}$ )

but **robust** (optimal for some worst case analysis)

## **II. Evolutionary algorithms**

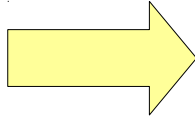
a. Fundamental elements

***b. Algorithms***

c. Math. analysis

Parameters:

$x, \sigma$

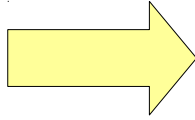


Generate  $\lambda$  points around  $x$   
( $x + \sigma N$  where  $N$  is a standard  
Gaussian)

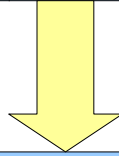
*Basic schema of an  
Evolution Strategy*

Parameters:

$x, \sigma$



Generate  $\lambda$  points around  $x$   
( $x + \sigma N$  where  $N$  is a standard  
Gaussian)



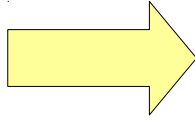
Compute their  $\lambda$  fitness values

*Basic schema of an  
Evolution Strategy*

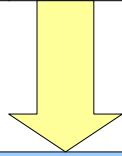
*Basic schema of an  
Evolution Strategy*

Parameters:

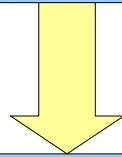
$x, \sigma$



Generate  $\lambda$  points around  $x$   
( $x + \sigma N$  where  $N$  is a standard  
Gaussian)



Compute their  $\lambda$  fitness values

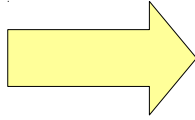


Select the  $\mu$  best

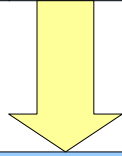
*Basic schema of an  
Evolution Strategy*

Parameters:

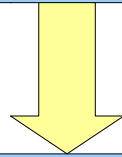
$x, \sigma$



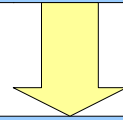
Generate  $\lambda$  points around  $x$   
( $x + \sigma N$  where  $N$  is a standard  
Gaussian)



Compute their  $\lambda$  fitness values



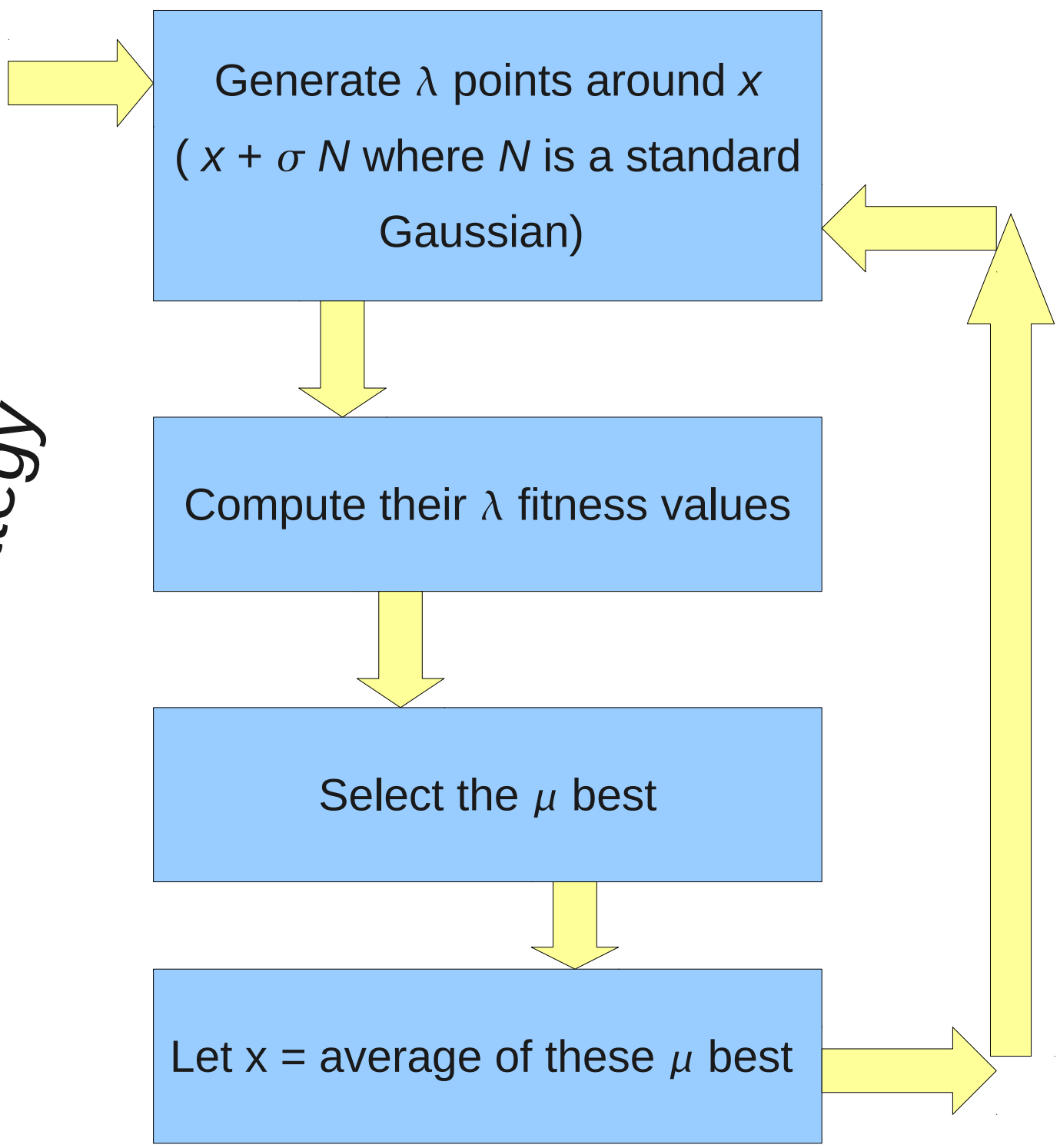
Select the  $\mu$  best



Let  $x =$  average of these  $\mu$  best

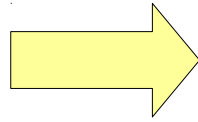
*Basic schema of an Evolution Strategy*

Parameters:  
 $x, \sigma$

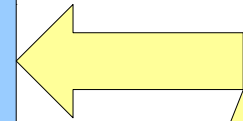


Parameters:

$x, \sigma$



Generate  $\lambda$  points around  $x$   
( $x + \sigma N$  where  $N$  is a standard  
Gaussian)



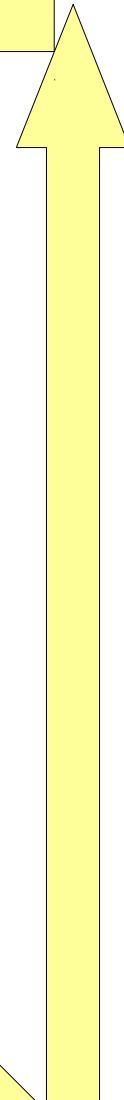
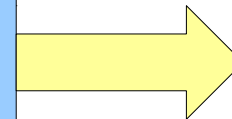
*Obviously  $\lambda$  parallel*

Multi-cores,  
Clusters, Grids...

values

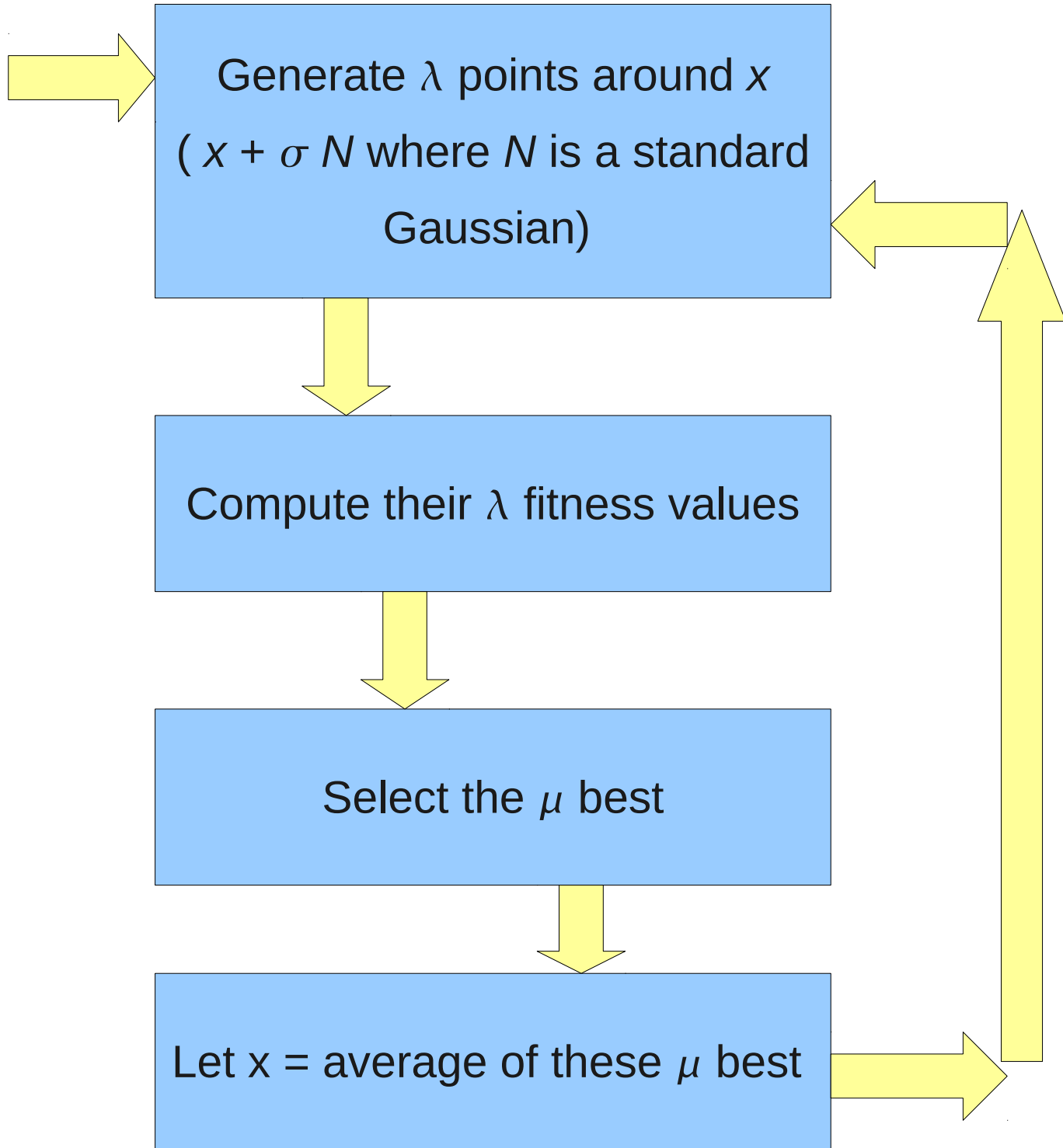
t

Let  $x =$  average of these  $\mu$  best



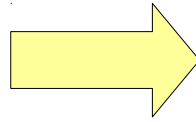
*Obviously  $\lambda$  parallel  
Really simple.*

Parameters:  
 $x, \sigma$

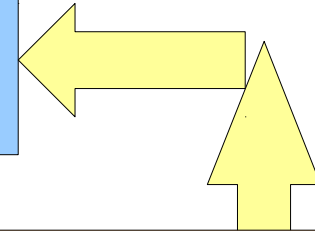


Parameters:

$x, \sigma$



Generate  $\lambda$  points around  $x$   
( $x + \sigma N$  where  $N$  is a standard  
Gaussian)



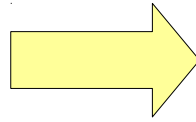
*Obviously  $\lambda$  parallel*  
*Really simple.*

Not a negligible advantage.  
When I accessed, for the 1st time,  
to a crucial industrial  
code of an important  
company, I believed  
that it would be  
clean and bug free.

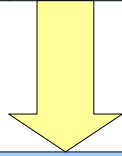
(I was young)

Parameters:

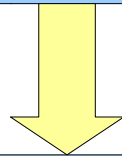
$x, \sigma$



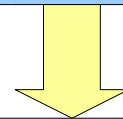
Generate 1 point  $x'$  around  $x$   
( $x + \sigma N$  where  $N$  is a standard  
Gaussian)



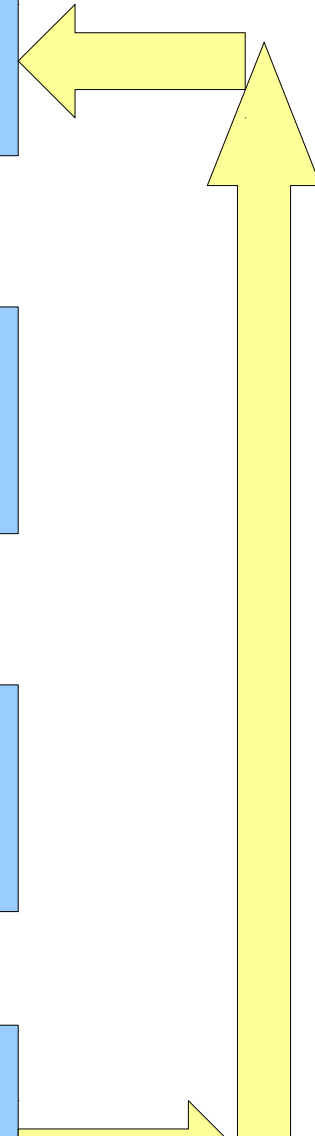
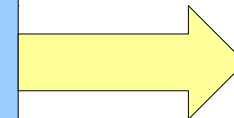
Compute its fitness value



Keep the best ( $x$  or  $x'$ ).

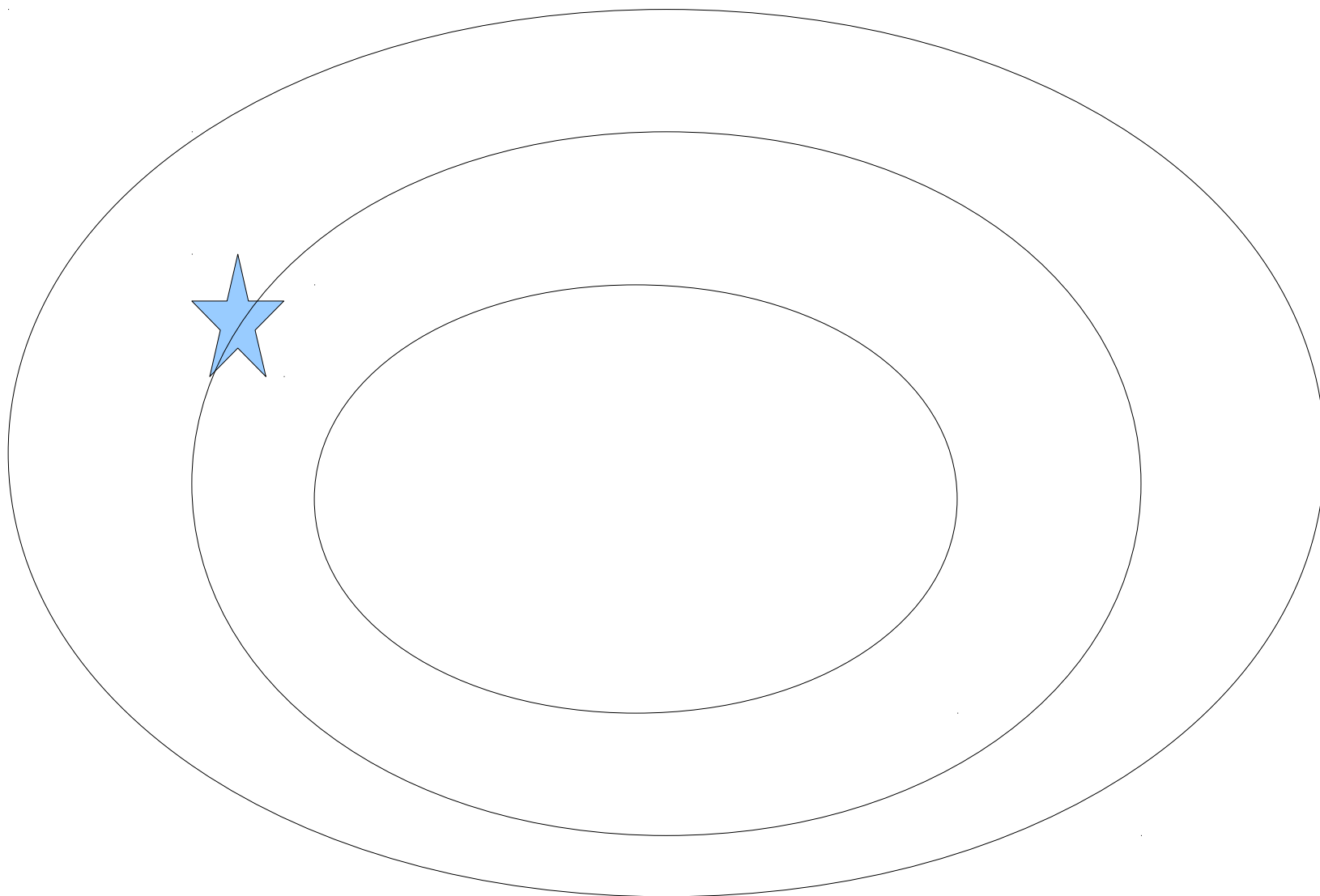


$x = \text{best}(x, x')$   
 $\sigma = 2\sigma$  if  $x'$  best  
 $\sigma = 0.84\sigma$  otherwise

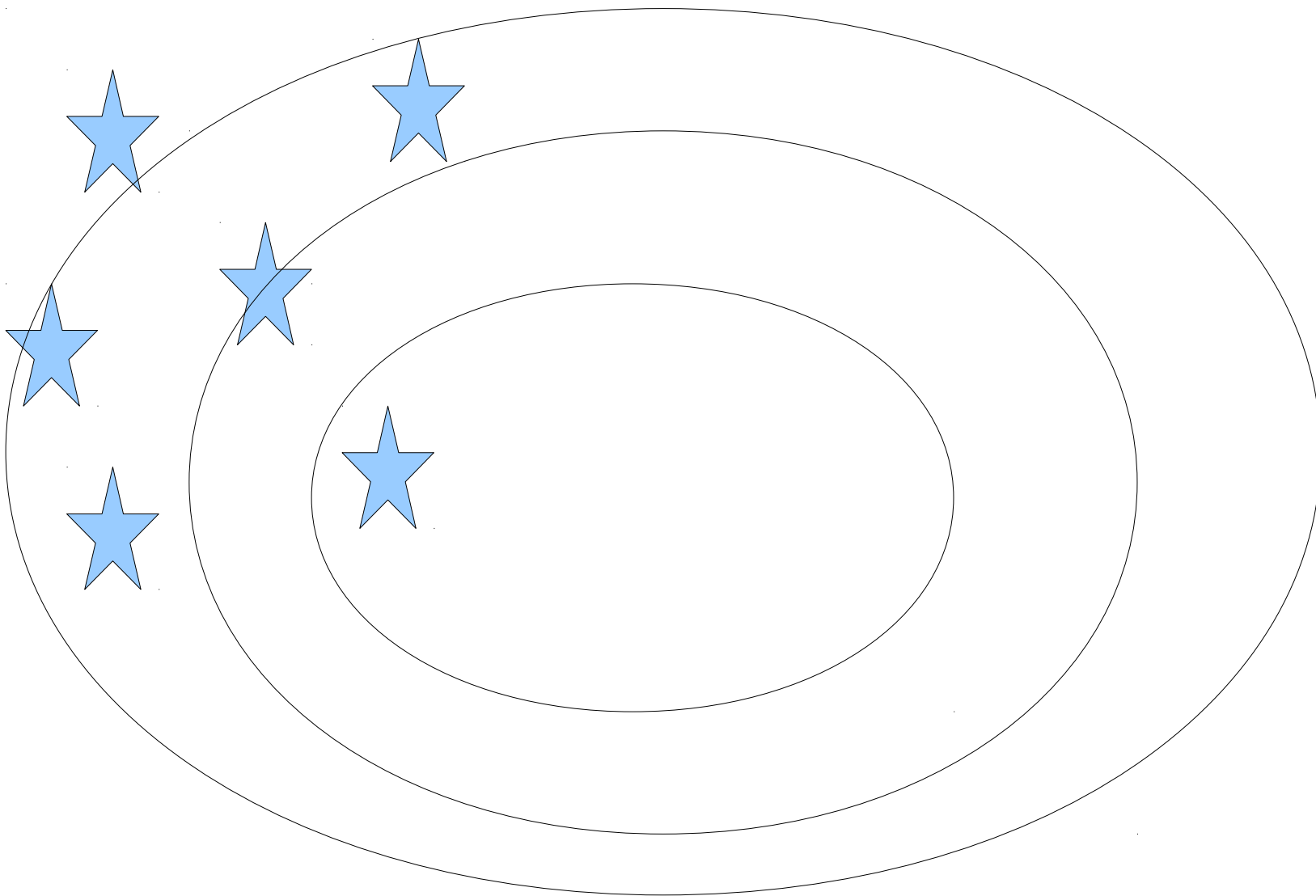


*The (1+1)-ES  
with 1/5<sup>th</sup> rule*

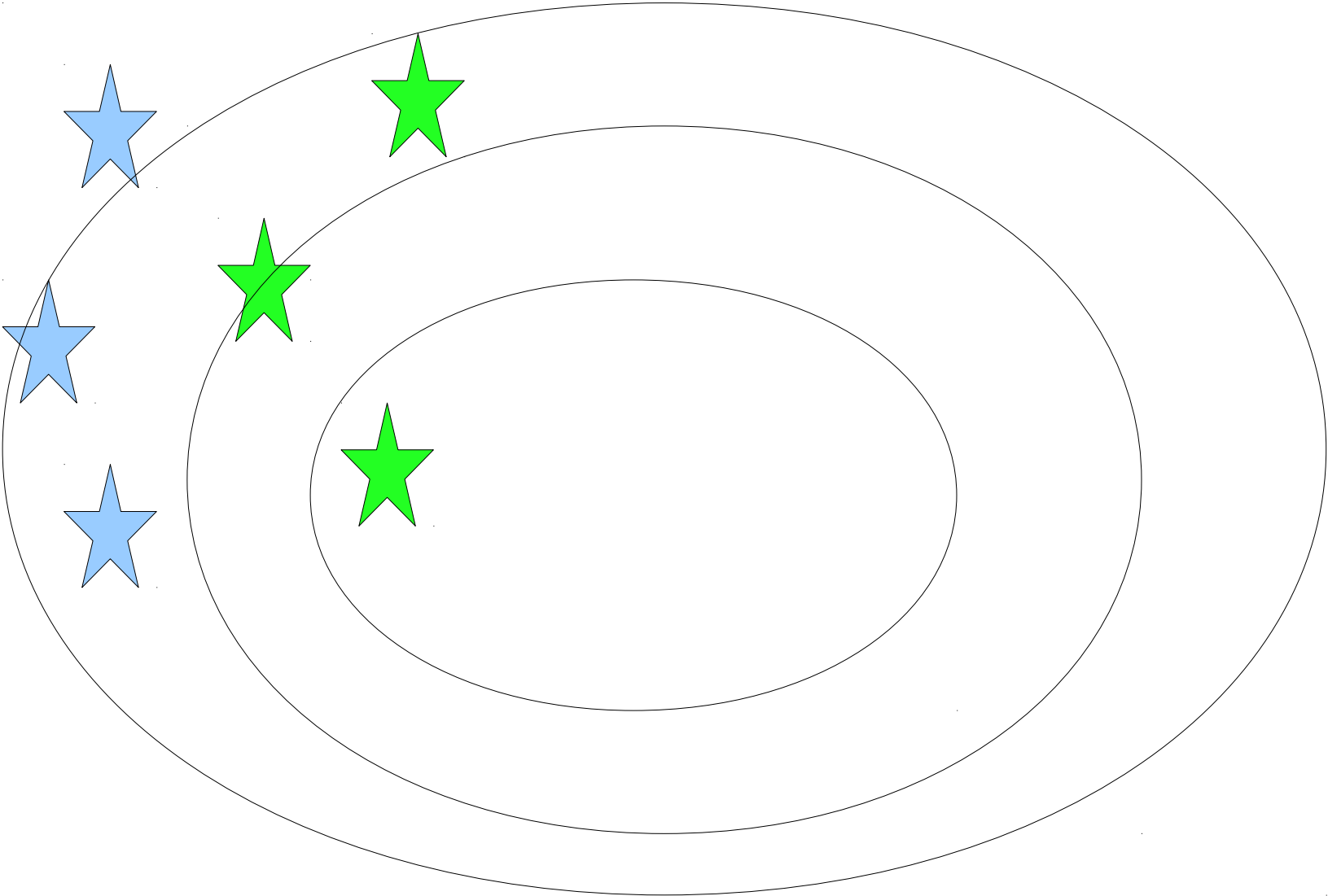
*This is x...*



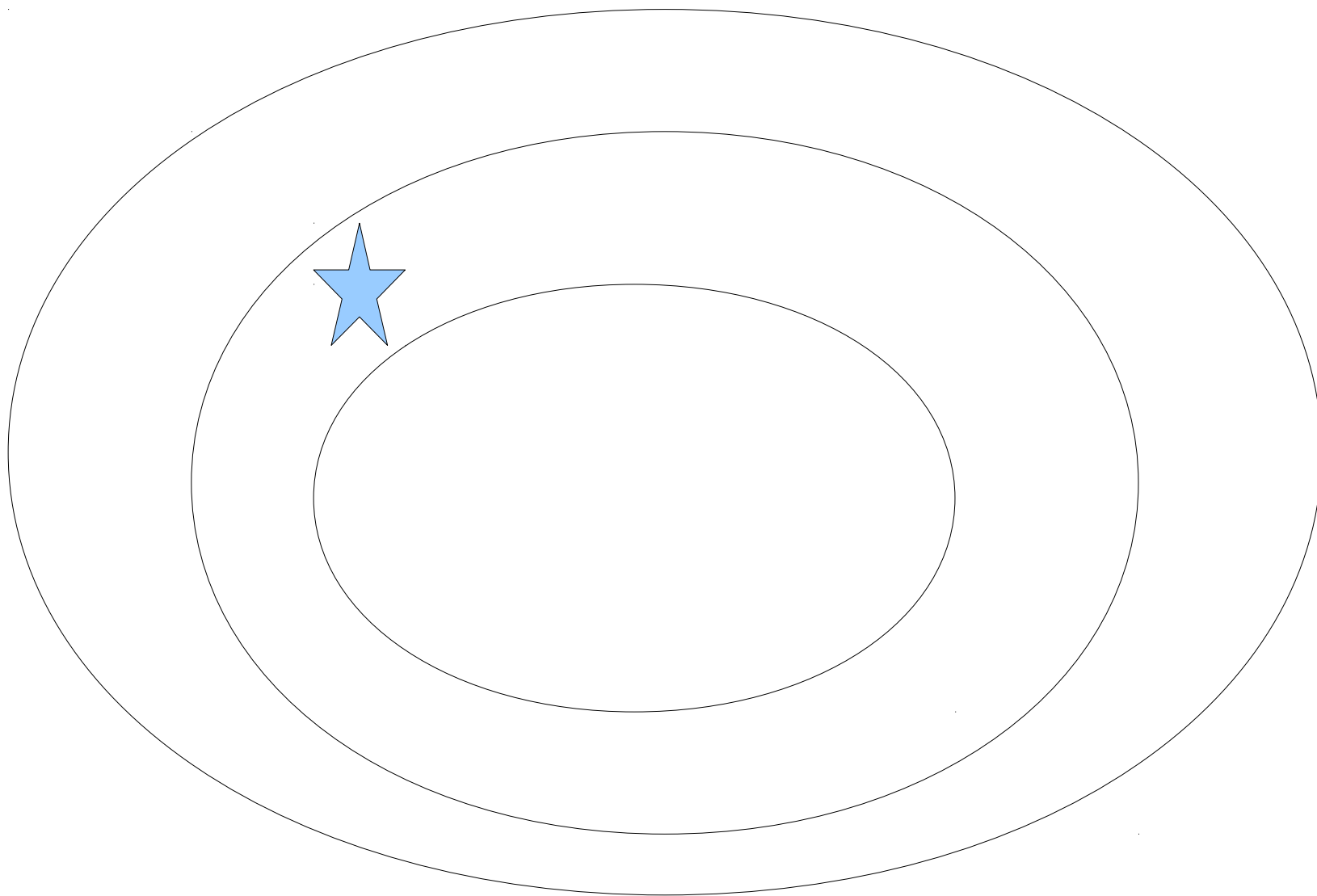
*I generate  $\lambda=6$  points*



*I select the  $\mu=3$  best points*



*$x = \text{average of these } \mu = 3 \text{ best points}$*



Ok.

Choosing an initial

$x$  is as in any algorithm.

But how do I choose  $\sigma$  ?

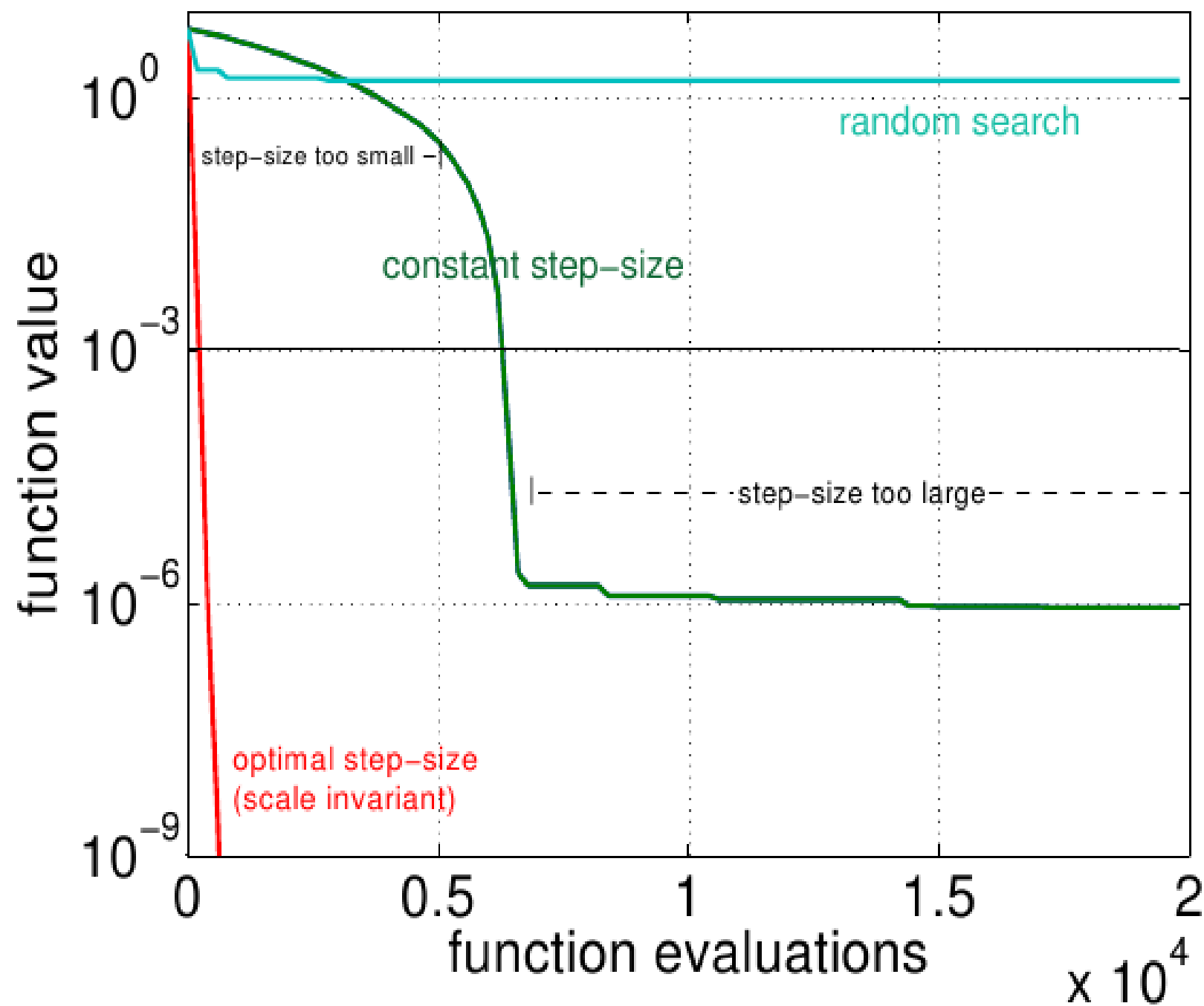
Ok.

Choosing  $x$  is as in any algorithm.

But how do I choose sigma ?

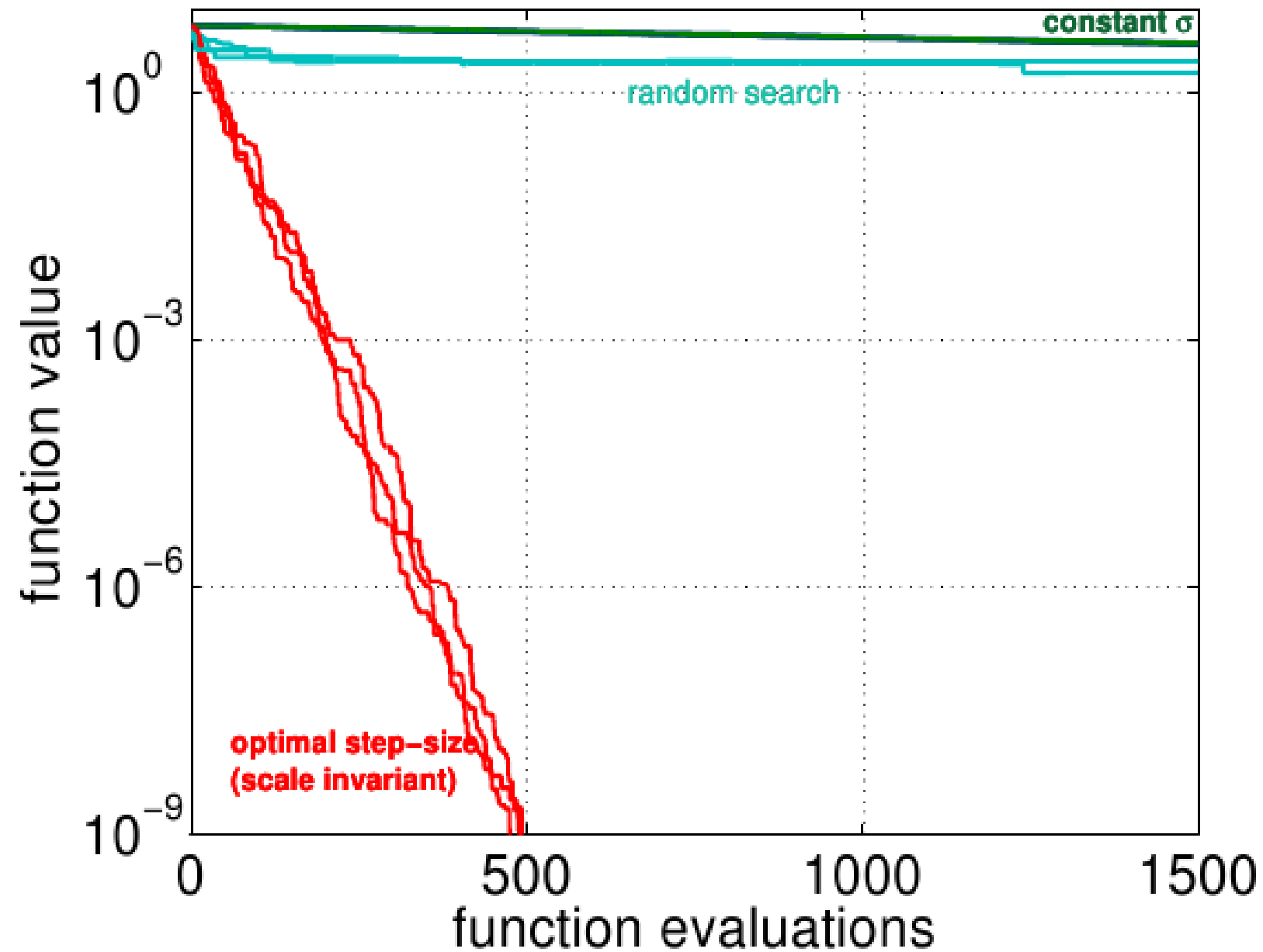
Sometimes by human guess.

But for large number of iterations,  
there is better.



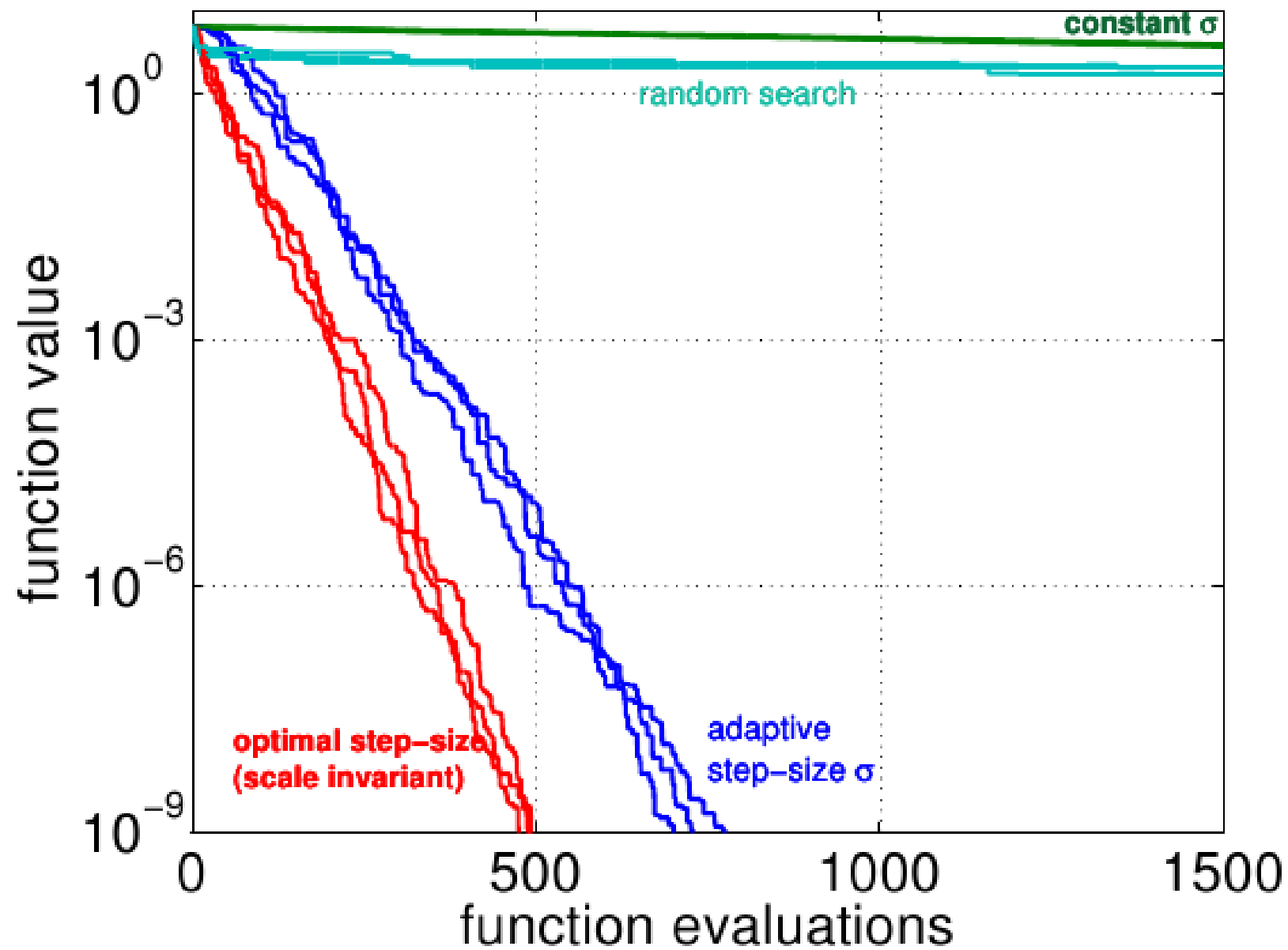
$$f(\mathbf{x}) = \sum_{i=1}^n x_i^2$$

in  $[-0.2, 0.8]^n$   
for  $n = 10$



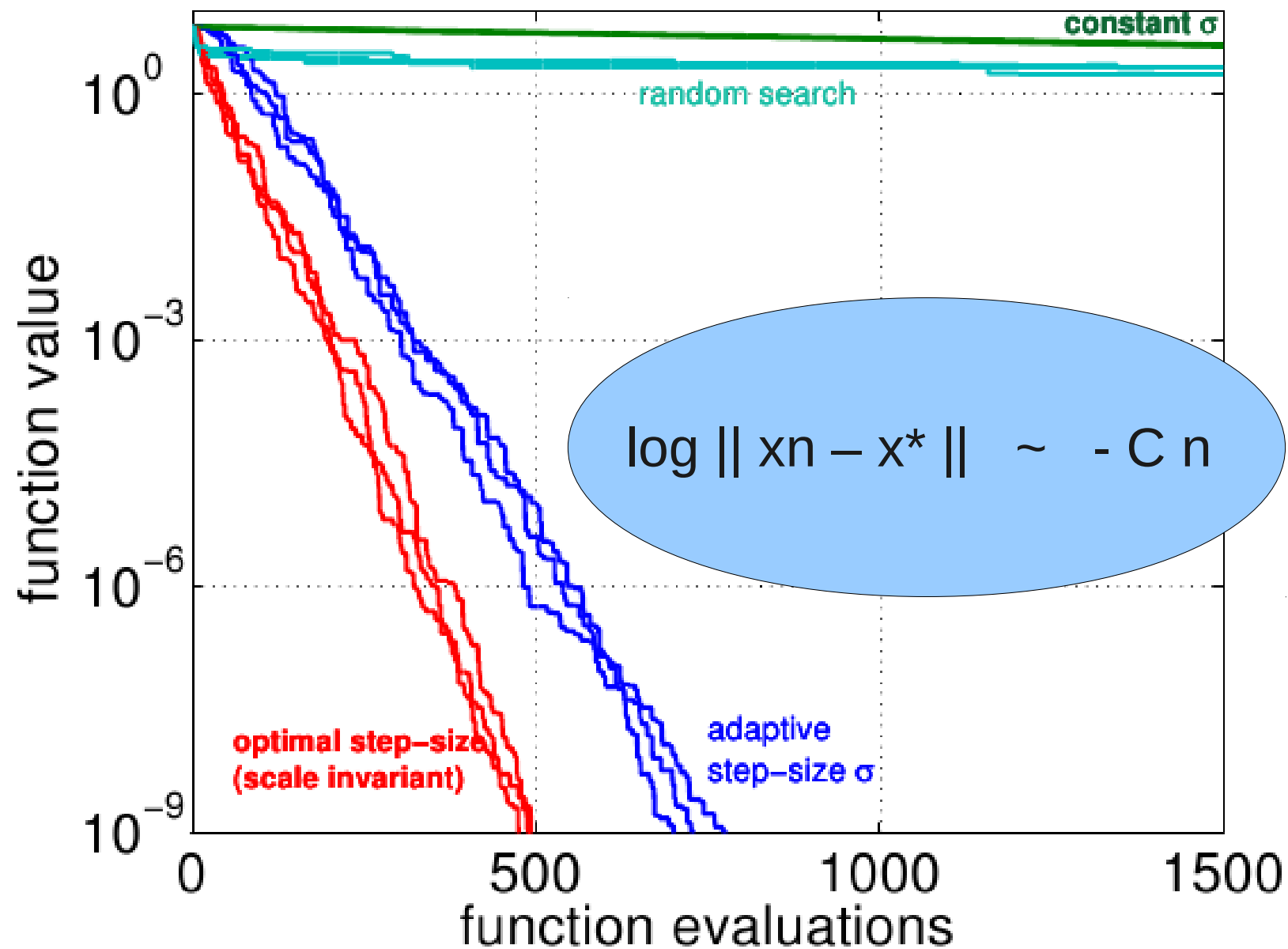
$$f(\mathbf{x}) = \sum_{i=1}^n x_i^2$$

in  $[-0.2, 0.8]^n$   
for  $n = 10$



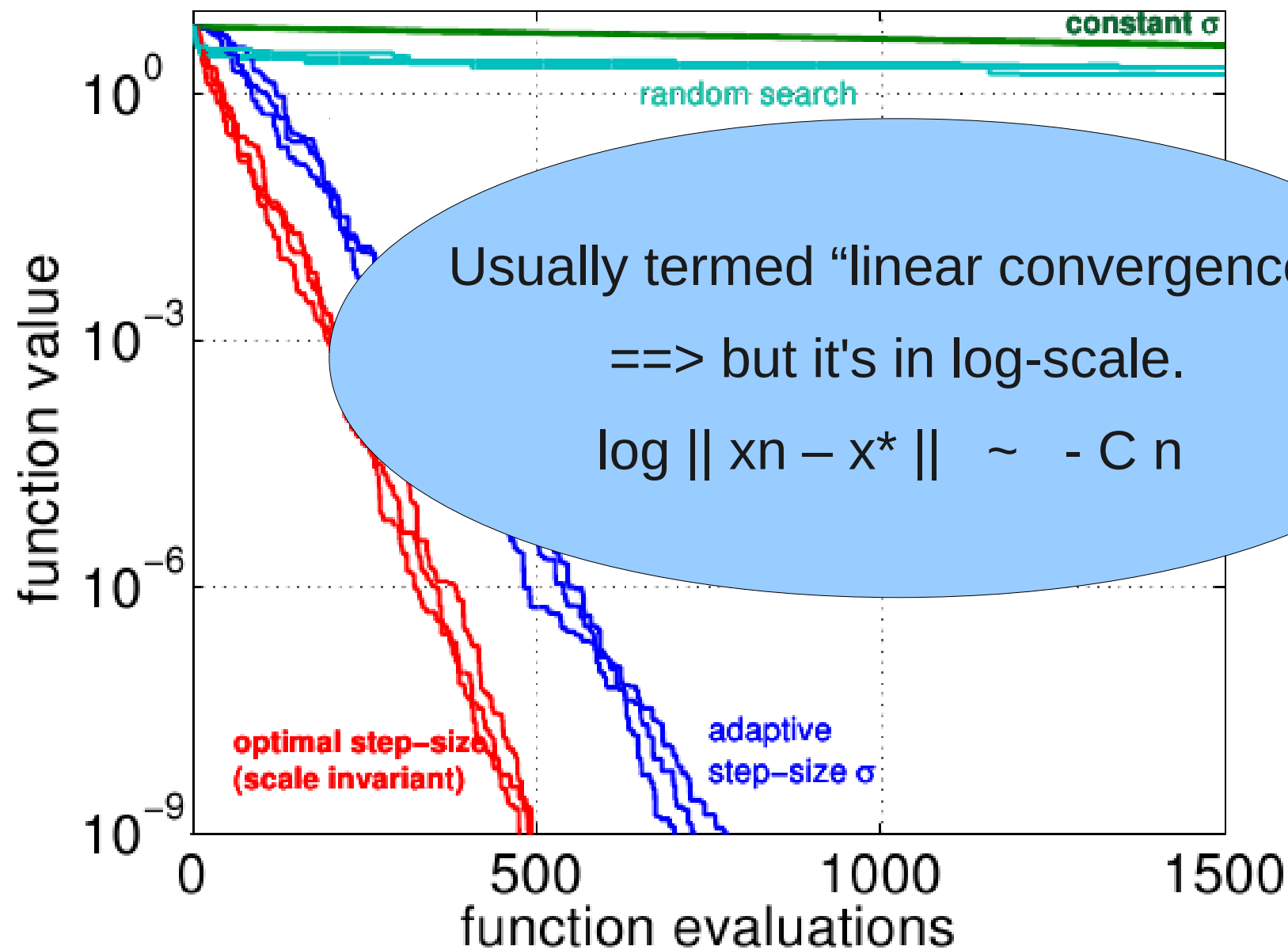
$$f(\mathbf{x}) = \sum_{i=1}^n x_i^2$$

in  $[-0.2, 0.8]^n$   
for  $n = 10$



$$f(\mathbf{x}) = \sum_{i=1}^n x_i^2$$

in  $[-0.2, 0.8]^n$   
for  $n = 10$



Usually termed “linear convergence”,

==> but it's in log-scale.

$$\log \|x_n - x^*\| \sim -C n$$

$$f(x) = \sum_{i=1}^n x_i^2$$

in  $[-0.2, 0.8]^n$

for  $n = 10$

# Examples of evolutionary algorithms

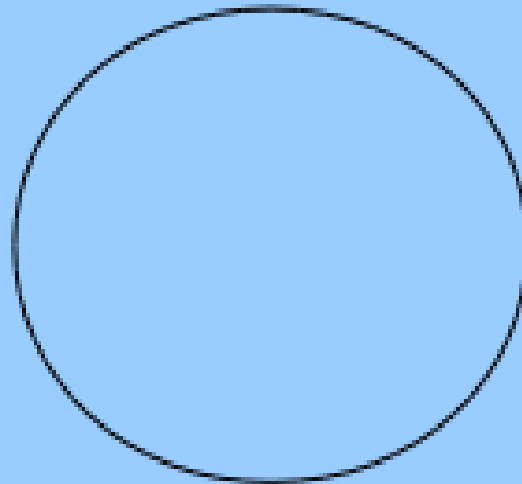
```
While ( I have time )  
{  
    Generate points  $(x_1, \dots, x_\lambda)$  distributed as  $N(x, \sigma)$   
    Evaluate the fitness at  $x_1, \dots, x_\lambda$   
    Update  $x$ , update  $\sigma$   
}
```

Main trouble: choosing  $\sigma$

- Cumulative step-size adaptation
- Mutative self-adaptation
- Estimation of Multivariate Normal Algorithm

# Estimation of Multivariate Normal Algorithm

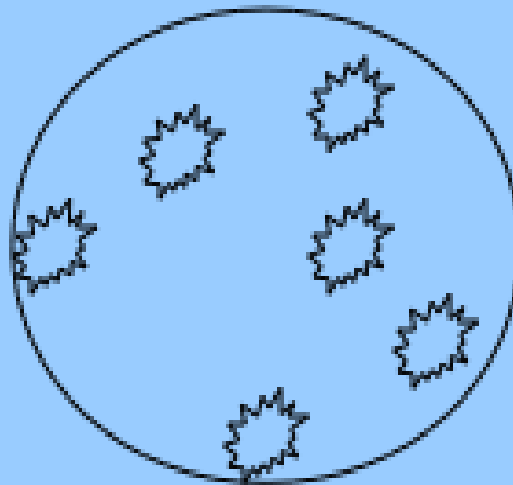
```
While ( I have time )  
{  
    Generate points  $(x_1, \dots, x_\lambda)$  distributed as  $N(x, \sigma)$   
    Evaluate the fitness at  $x_1, \dots, x_\lambda$   
     $X = \text{mean } \mu \text{ best points}$   
     $\sigma = \text{standard deviation of } \mu \text{ best points}$   
}
```



I have a Gaussian...

# Estimation of Multivariate Normal Algorithm

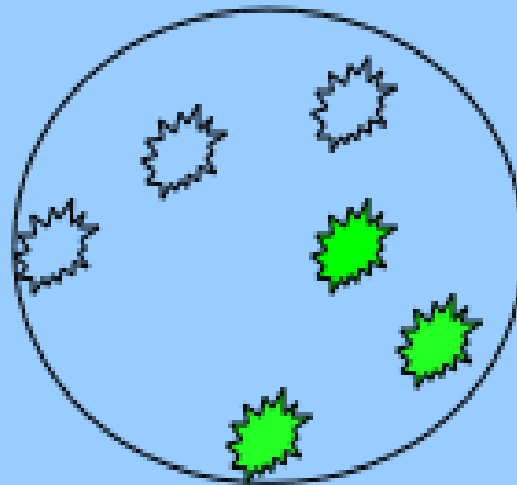
```
While ( I have time )  
{  
    Generate points  $(x_1, \dots, x_\lambda)$  distributed as  $N(x, \sigma)$   
    Evaluate the fitness at  $x_1, \dots, x_\lambda$   
     $X = \text{mean } \mu$  best points  
     $\sigma = \text{standard deviation of } \mu$  best points  
}
```



I generate 6 points

# Estimation of Multivariate Normal Algorithm

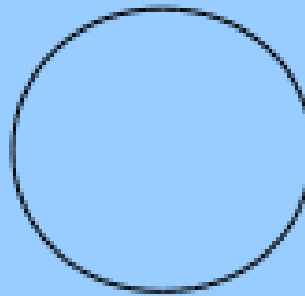
```
While ( I have time )  
{  
    Generate points  $(x_1, \dots, x_\lambda)$  distributed as  $N(x, \sigma)$   
    Evaluate the fitness at  $x_1, \dots, x_\lambda$   
     $X = \text{mean } \mu \text{ best points}$   
     $\sigma = \text{standard deviation of } \mu \text{ best points}$   
}
```



I select the three best

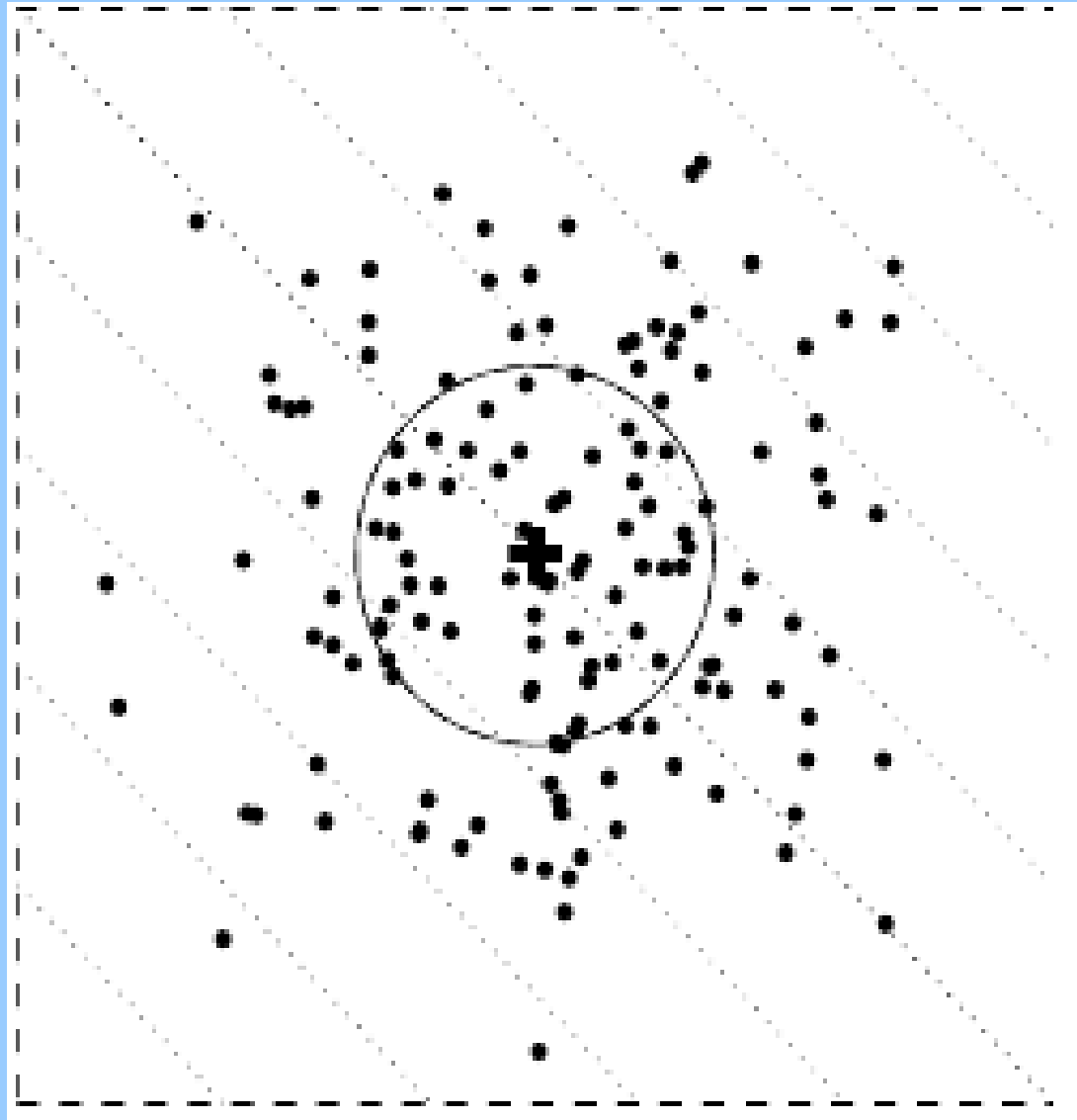
# Estimation of Multivariate Normal Algorithm

```
While ( I have time )  
{  
    Generate points  $(x_1, \dots, x_\lambda)$  distributed as  $N(x, \sigma)$   
    Evaluate the fitness at  $x_1, \dots, x_\lambda$   
     $X = \text{mean } \mu \text{ best points}$   
     $\sigma = \text{standard deviation of } \mu \text{ best points}$   
}
```



Obviously 6-parallel

# EMNA is usually non-isotropic

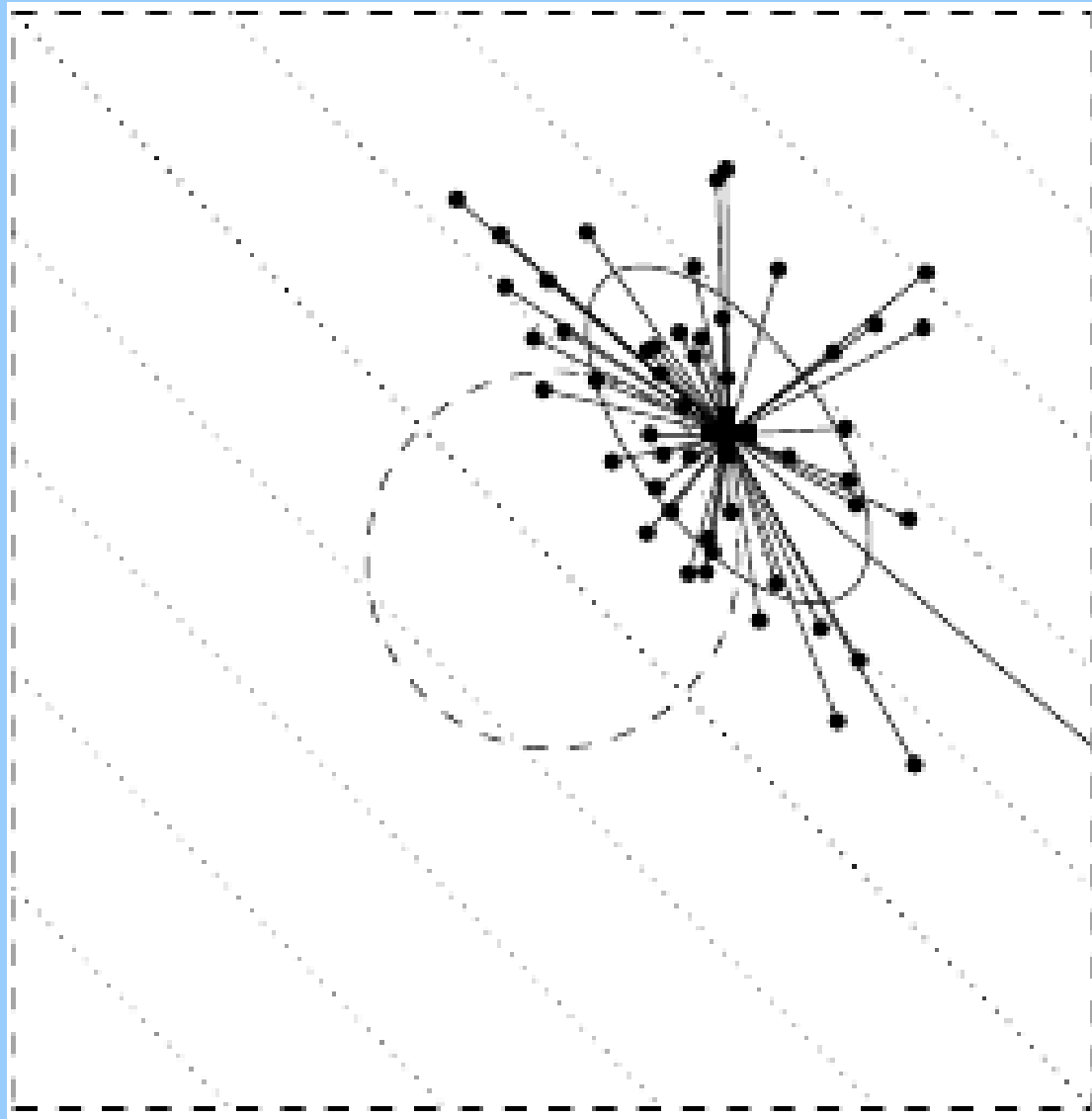


distributed as  $N(x, \sigma)$   
 $x\lambda$

best points

Obviously 6-parallel

# EMNA is usually non-isotropic



distributed as  $N(x, \sigma)$   
( $\lambda$ )

best points

obviously 6-parallel

# Self-adaptation (works in many frameworks)

```
 $\mu = \lambda / 4$   
While ( I have time )  
{  
    Generate  $(\sigma_1, \dots, \sigma_\lambda)$  as  $\sigma \times \exp(-k.N)$   
    Generate points  $(x_1, \dots, x_\lambda)$  distributed as  $N(x, \sigma)$   
    Select the  $\mu$  best points  
    Update  $x$  (=mean), update  $\sigma$  (=log. mean)  
}
```

# Self-adaptation (works in many frameworks)

```
 $\mu = \lambda / 4$   
While ( I have time )  
{  
  Generate  $(\sigma_1, \dots, \sigma_\lambda)$  as  $\sigma \times \exp(-k.N)$   
  Generate points  $(x_1, \dots, x_\lambda)$  distributed as  $N(x, \sigma)$   
  Select the  $\mu$  best points  
  Update  $x$  (=mean), update  $\sigma$  (=log. mean)  
}
```

Can be used for non-isotropic  
multivariate Gaussian  
distributions.

Let's generalize.

We have seen algorithms which work as follows:

- we keep one search point in memory  
(and one step-size)
- we generate individuals
- we evaluate these individuals
- we regenerate a search point and a step-size

Maybe we could keep more than one search point ?

Let's generalize.

We have seen algorithms which work as follows:

- we keep

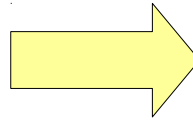
==>  $\mu$  search points

- ==>  $\lambda$  generated individuals

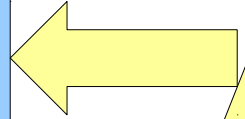
- we regenerate  $\lambda - \mu$  individuals of step-size

Maybe we could keep more than one search point ?

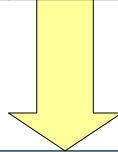
Parameters:  
 $x_1, \dots, x_\mu$



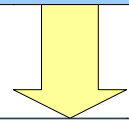
Generate  $\lambda$  points  
around  $x_1, \dots, x_\mu$   
e.g. each  $x$  randomly generated  
from two points



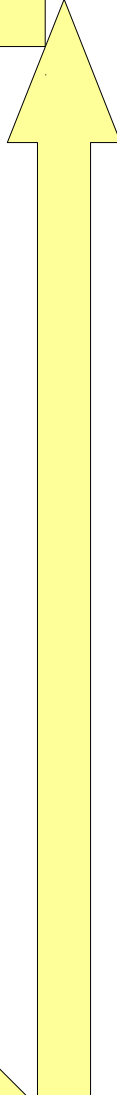
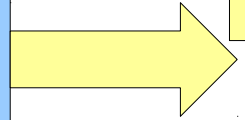
Compute their  $\lambda$  fitness values



Select the  $\mu$  best



Don't average...



*Obviously  $\lambda$  parallel*  
*Really simple.*

Generate  $\lambda$  points

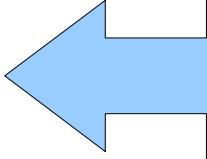
around  $x_1, \dots, x_\mu$

e.g. each  $x$  randomly generated

from two points

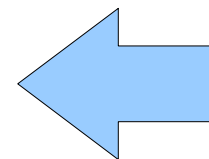
Generate  $\lambda$  points  
around  $x_1, \dots, x_\mu$   
e.g. each  $x$  randomly generated  
from two points

This is a  
cross-over



Generate  $\lambda$  points  
around  $x_1, \dots, x_\mu$   
e.g. each  $x$  randomly generated  
from two points

This is a  
cross-over



### Example of procedure for generating a point:

- Randomly draw  $k$  parents  $x_1, \dots, x_k$   
(truncation selection: randomly in selected individuals)

- For generating the  $i$ th coordinate of new individual  $z$ :

$$u = \text{random}(1, k)$$

$$z(i) = x(u)_i$$

## Let's summarize:

We have seen a general scheme for optimization:

- generate a population (e.g. from some distribution, or from a set of search points)
- select the best = new search points

==> Small difference between an Evolutionary Algorithm (EA) and an Estimation of Distribution Algorithm (EDA).

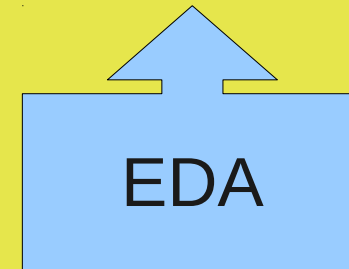
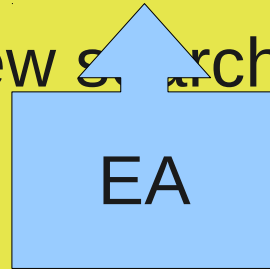
==> Some EA (older than the EDA acronym) are EDAs.

## Let's summarize:

We have seen a general scheme for optimization:

- generate a population (e.g. from some distribution, or from a set of search points)

- select the best = new search points



==> Small difference between an

Evolutionary Algorithm (EA) and an

Estimation of Distribution Algorithm (EDA).

==> Some EA (older than the EDA acronym) are EDAs.

## Gives a lot freedom:

- choose your representation  
and operators (depending on the problem)
- if you have a step-size, choose adaptation rule
- choose your population-size (depending on your  
computer/grid )
- choose  $\mu$  (carefully) e.g.  $\min(\text{dimension}, \lambda / 4)$

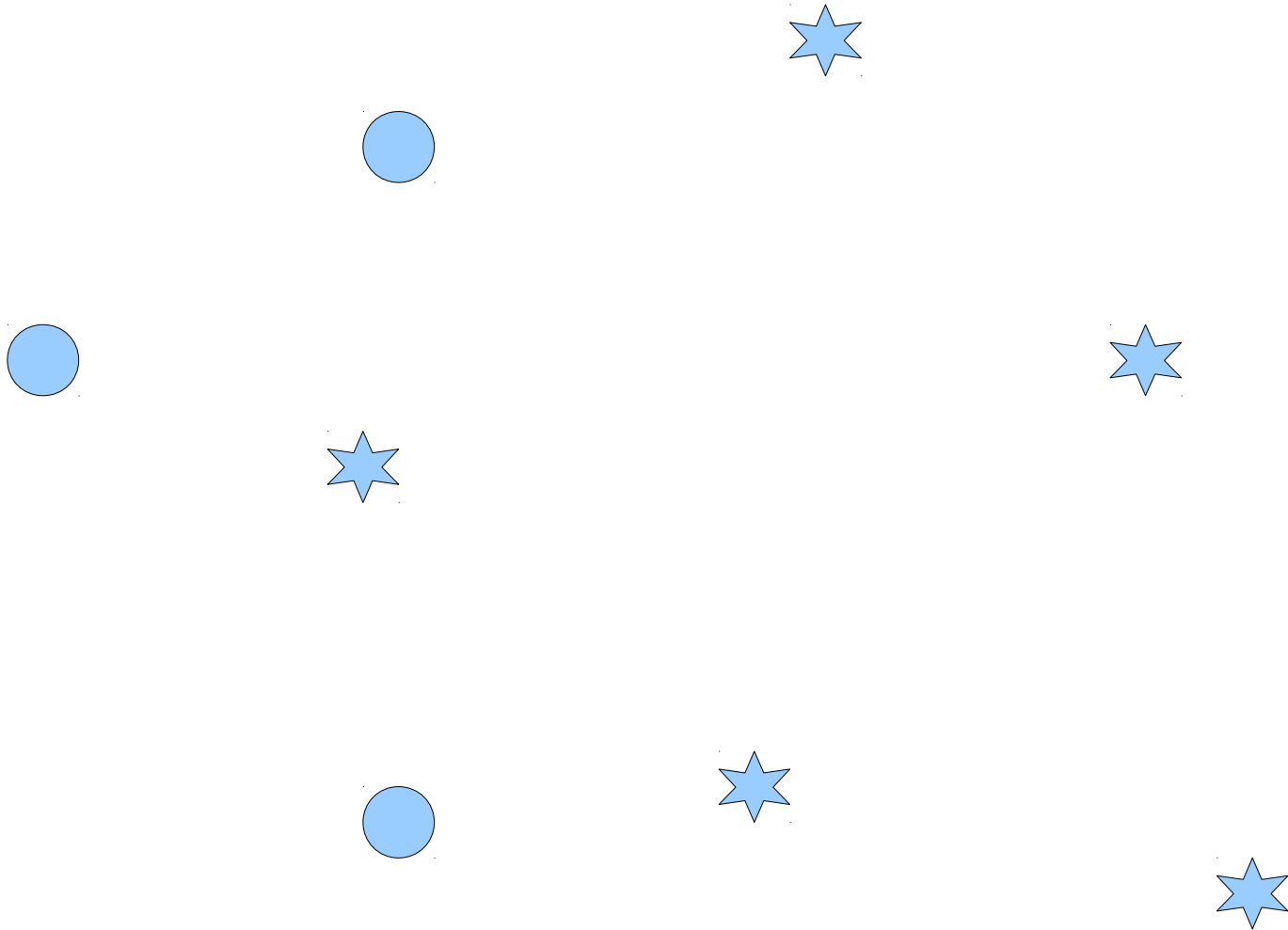
## Gives a lot freedom:

- choose your operators (depending on the problem)
- if you have a step-size, choose adaptation rule
- choose your population-size (depending on your computer/grid )
- choose  $\mu$  (carefully) e.g.  $\min(\text{dimension}, \lambda / 4)$

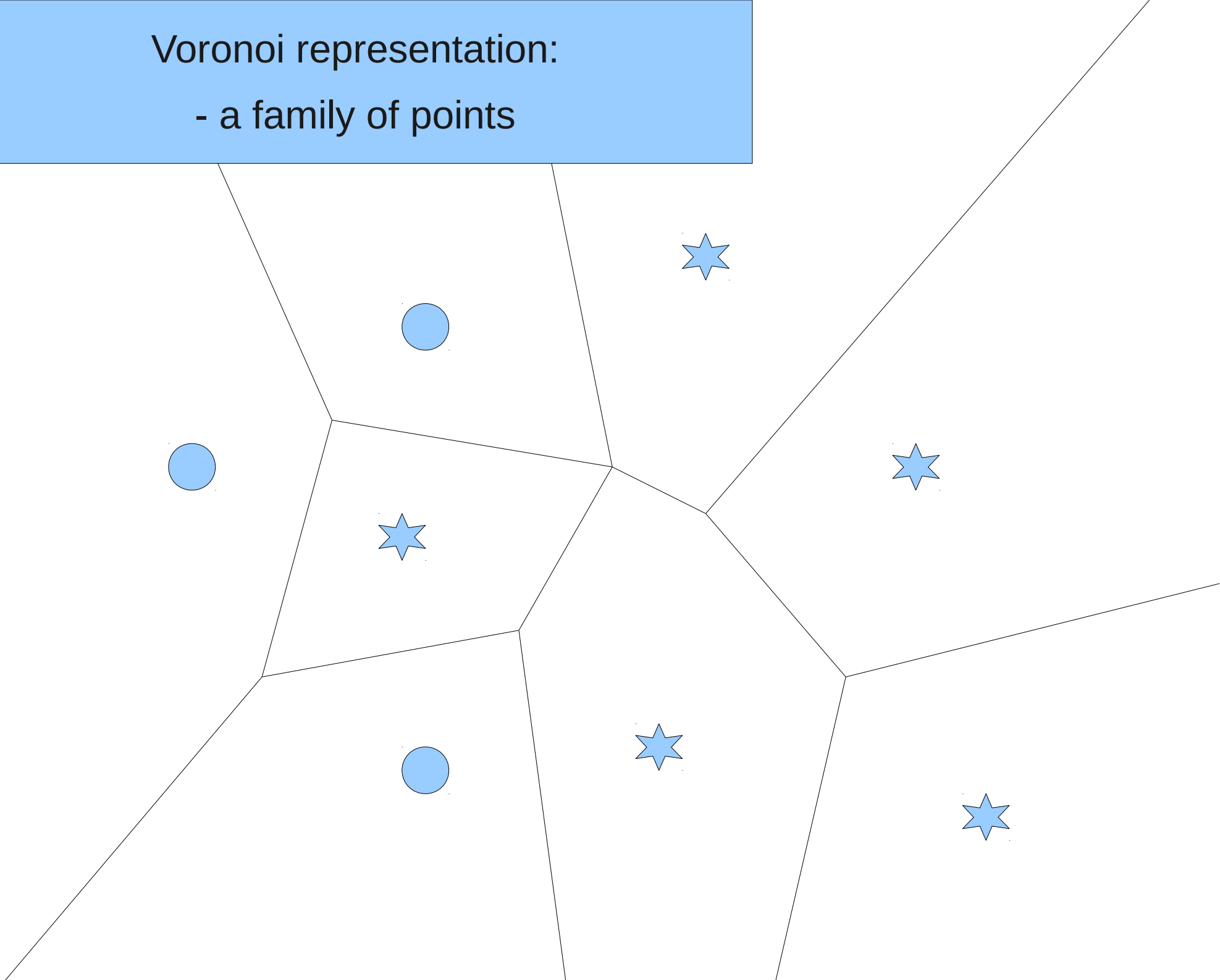
## Can handle strange things:

- optimize a physical structure ?
- structure represented as a Voronoi
- cross-over makes sense, benefits from local structure
- not so many algorithms can work on that

Voronoi representation:  
- a family of points

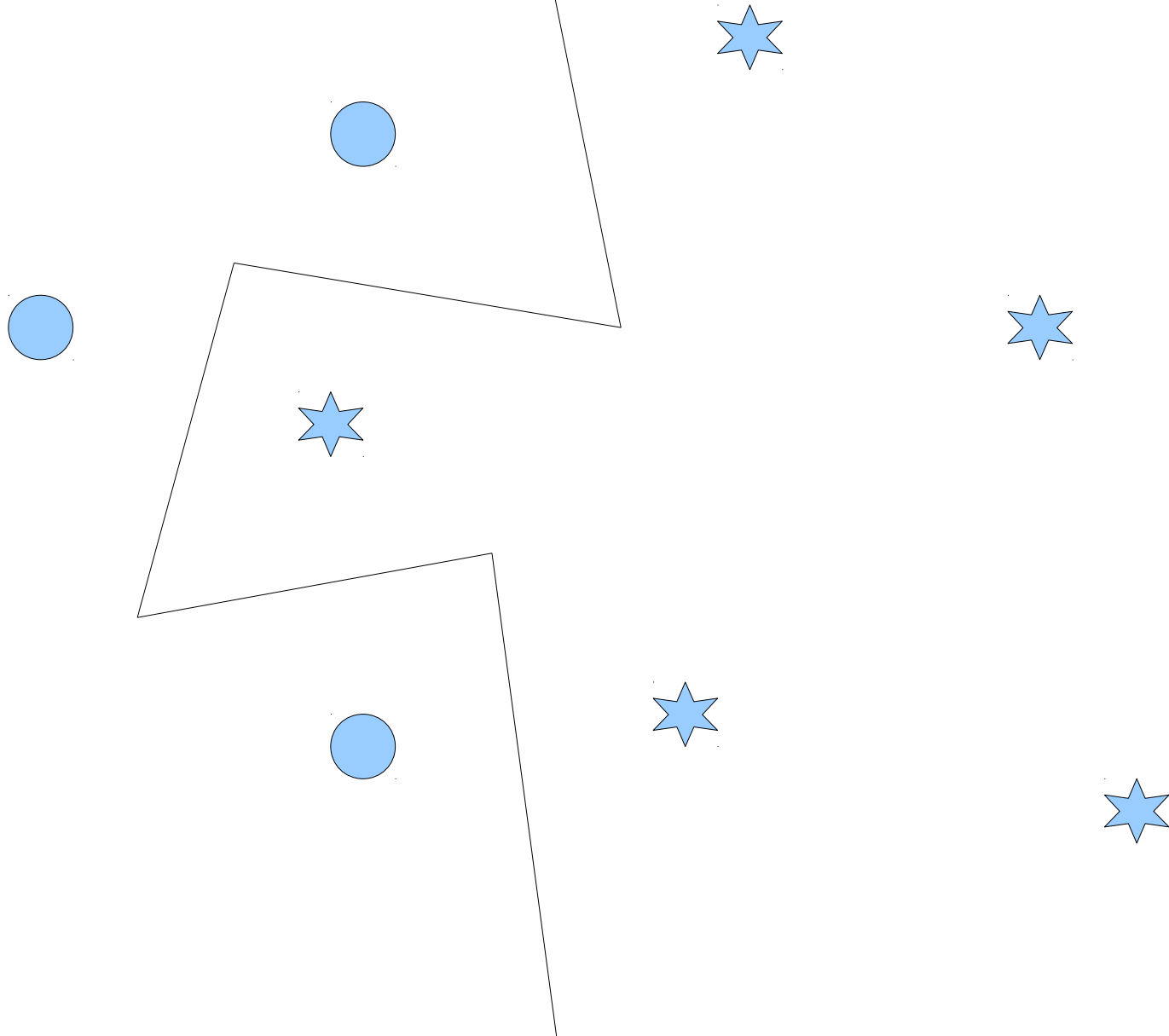


Voronoi representation:  
- a family of points



## Voronoi representation:

- a family of points
- their labels

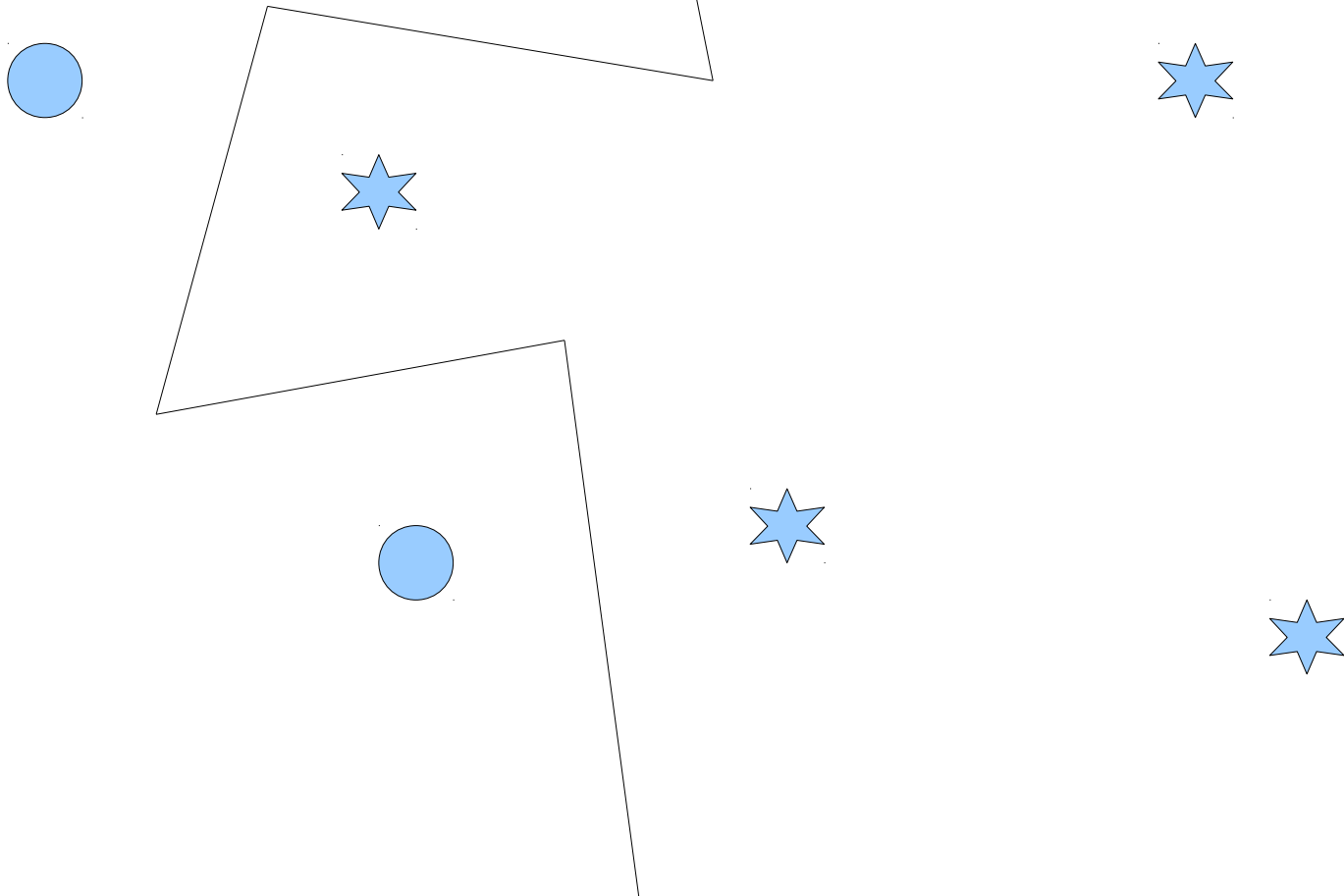


## Voronoi representation:

- a family of points
- their labels

==> cross-over makes sense

==> you can optimize a shape



Voronoi representation:

- a family of points
- their labels

==> cross-over makes sense

==> you can optimize a shape

==> not that mathematical;  
but really useful

**Mutations:** each label is changed with proba  $1/n$

**Cross-over:** each point/label is randomly drawn from one of  
the two parents

Voronoi representation:

- a family of points
- their labels

==> cross-over makes sense

==> you can optimize a shape

==> not that mathematical;  
but really useful

**Mutations:** each label is changed with proba  $1/n$

**Cross-over:** randomly pick one split in the representation:

- left part from parent 1
- right part from parent 2

==> related to biology

## Gives a lot freedom:

- choose your operators (depending on)
- if you have a step-size, choose ada
- choose your population-size (depen  
computer/grid
- choose  $\mu$  (carefully) e.g. min(dimer



## Can handle strange things:

- optimize a physical structure ?
- structure represented as a Voronoi
- cross-over makes sense, benefits from local structure
- not so many algorithms can work on that

## **II. Evolutionary algorithms**

a. Fundamental elements

b. Algorithms

***c. Math. Analysis***

Consider the (1+1)-ES.

$$x(n) = x(n-1) \text{ or } x(n-1) + \sigma(n-1)N$$

We want to maximize:

$$- E \log \| x(n) - f^* \|$$

Consider the (1+1)-ES.

$$x(n) = x(n-1) \text{ or } x(n-1) + \sigma(n-1)N$$

We want to maximize:

$$- E \log \| x(n) - f^* \|$$

-----

$$- E \log \| x(n-1) - f^* \|$$

Consider the (1+1)-ES.

$$x(n) = x(n-1) \text{ or } x(n-1) + \sigma(n-1)N$$

We want to maximize

$$- E \log || x(n) -$$

-----

$$- E \log || x(n-1)$$

We don't know  $f^*$ .

How can we optimize this ?

We will observe

the acceptance rate,

and we will deduce if  $\sigma$

is too large or too small..

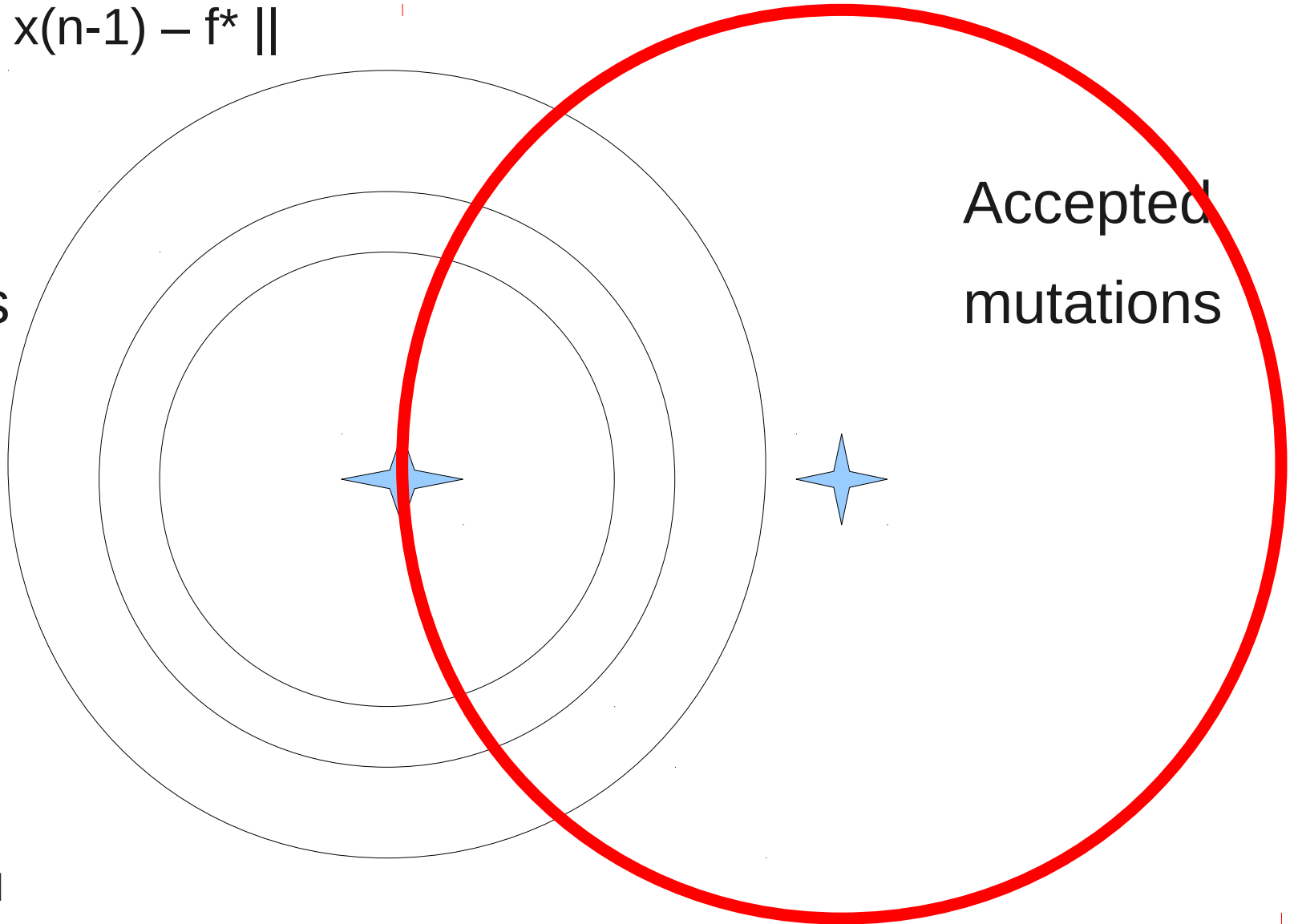
$$- E \log \| x(n) - f^* \|$$

***ON THE NORM FUNCTION***

---

$$- E \log \| x(n-1) - f^* \|$$

Rejected  
mutations



Accepted  
mutations

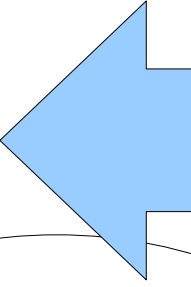
Olivier Teytaud

Inria Tao, en visite dans la belle ville de Liège

$$- E \log \| x(n) - f^* \|$$

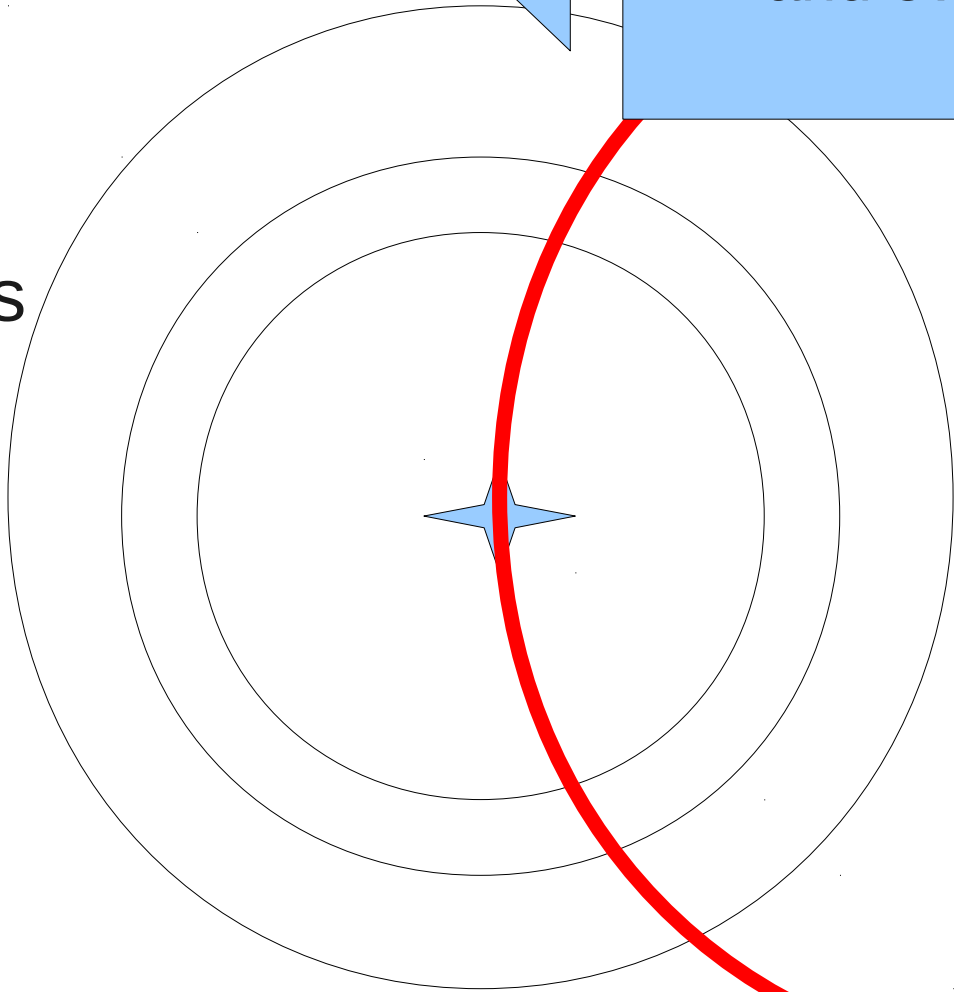
-----

$$- E \log \| x(n-1) - f^* \|$$

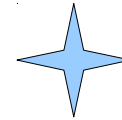


For each step-size,  
evaluate this “expected progress rate”  
and evaluate “P(acceptance)”

Rejected  
mutations



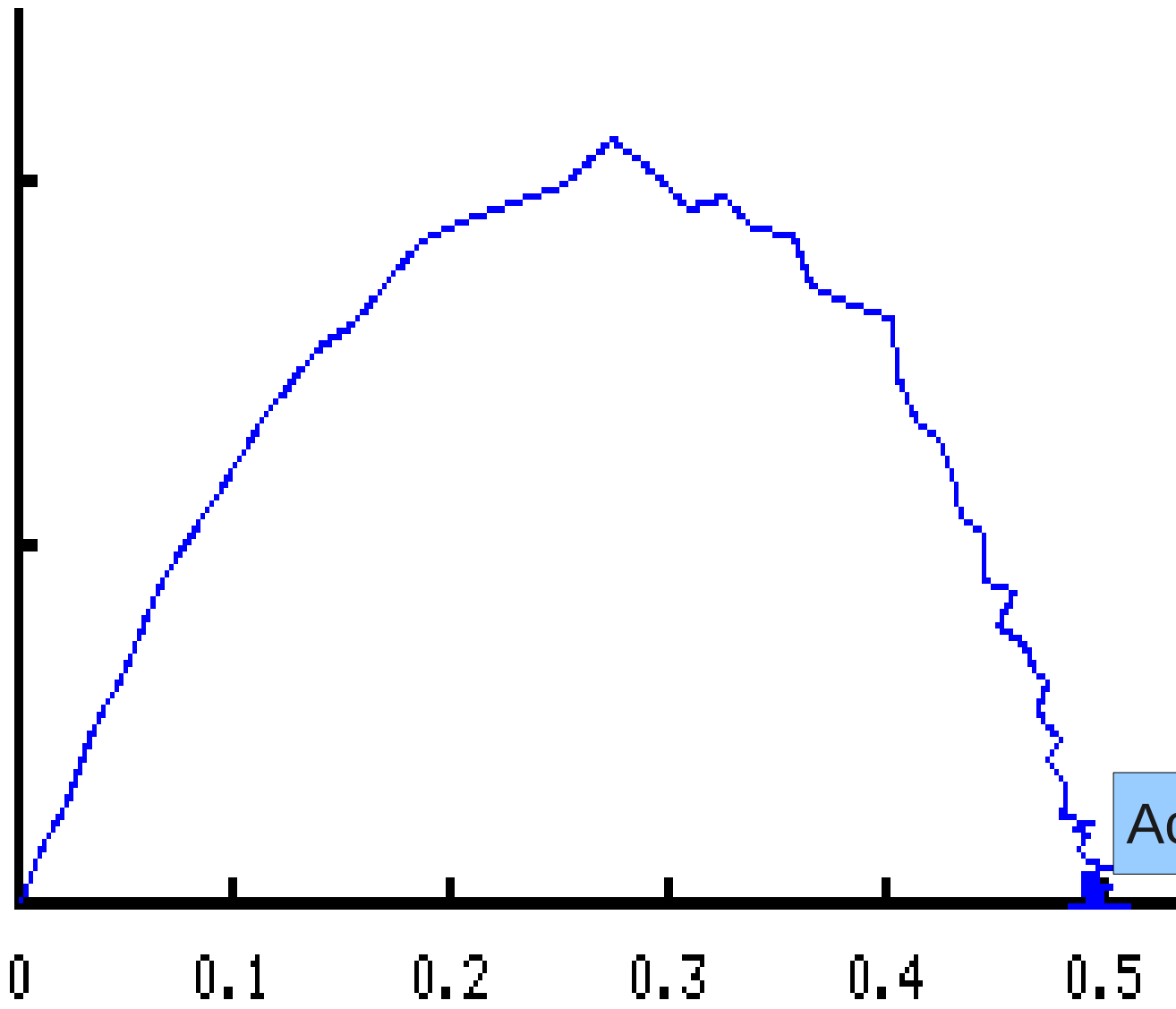
Accepted  
mutations



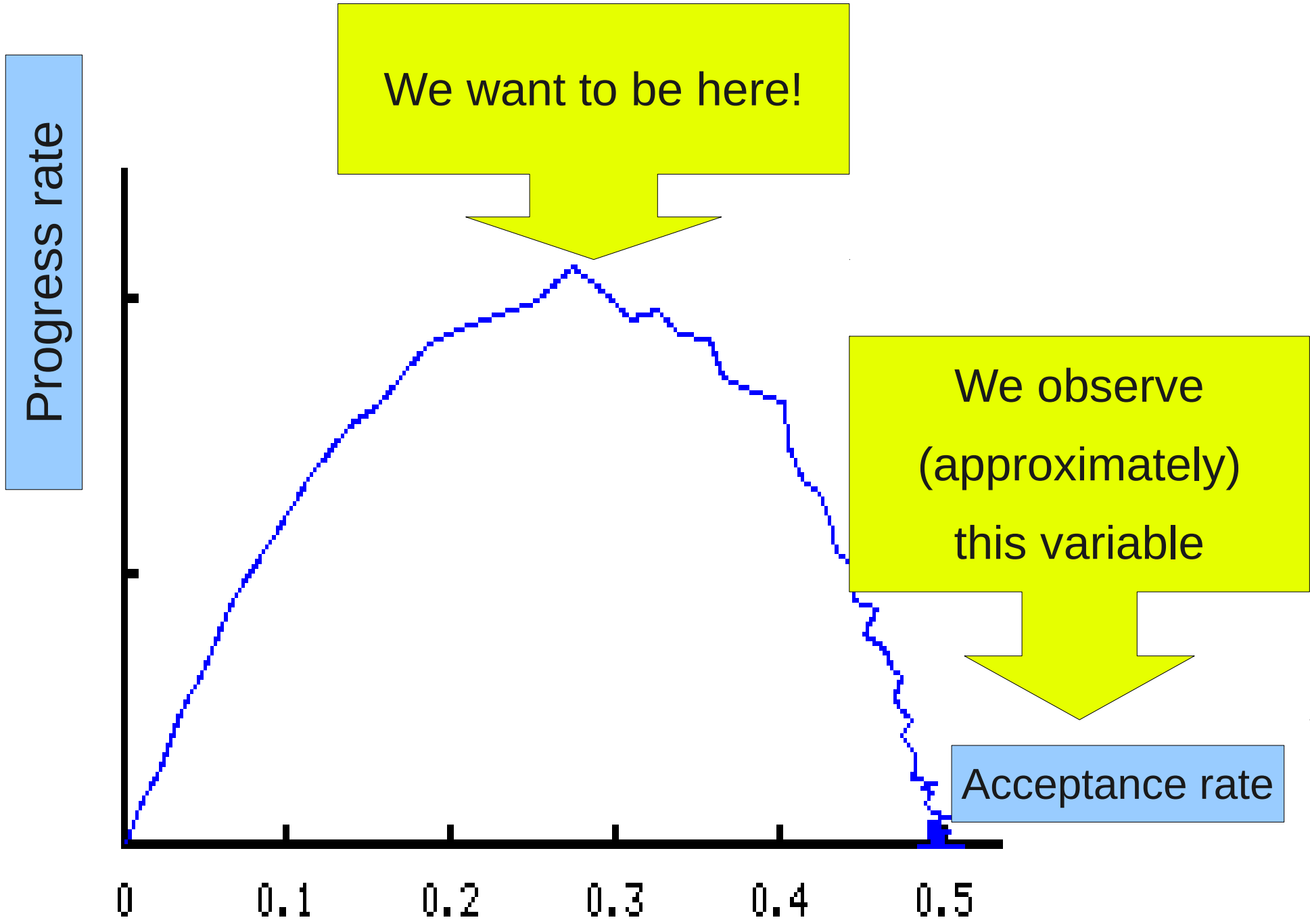
Olivier Teytaud

Inria Tao, en visite dans la belle ville de Liège

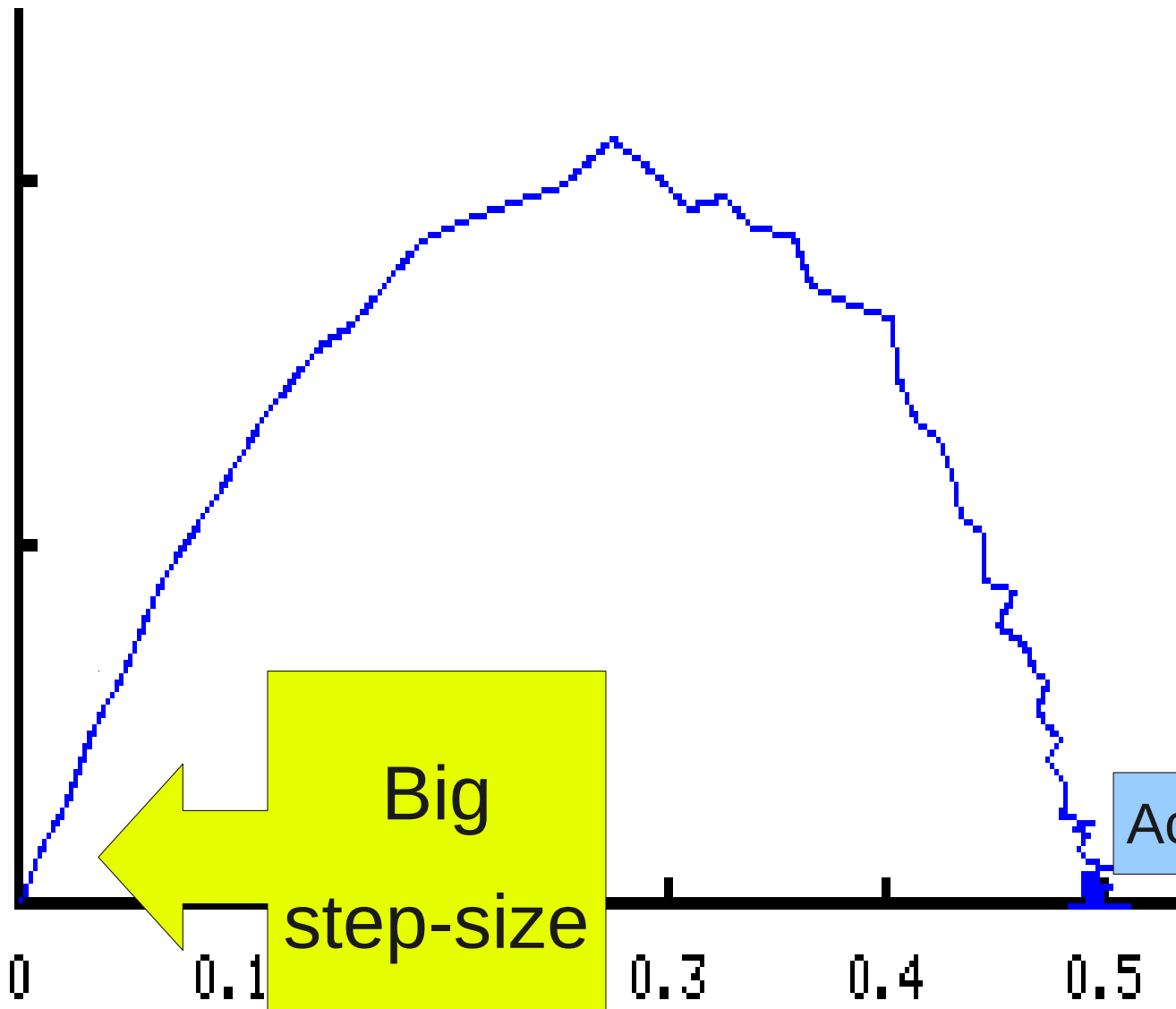
Progress rate



Acceptance rate



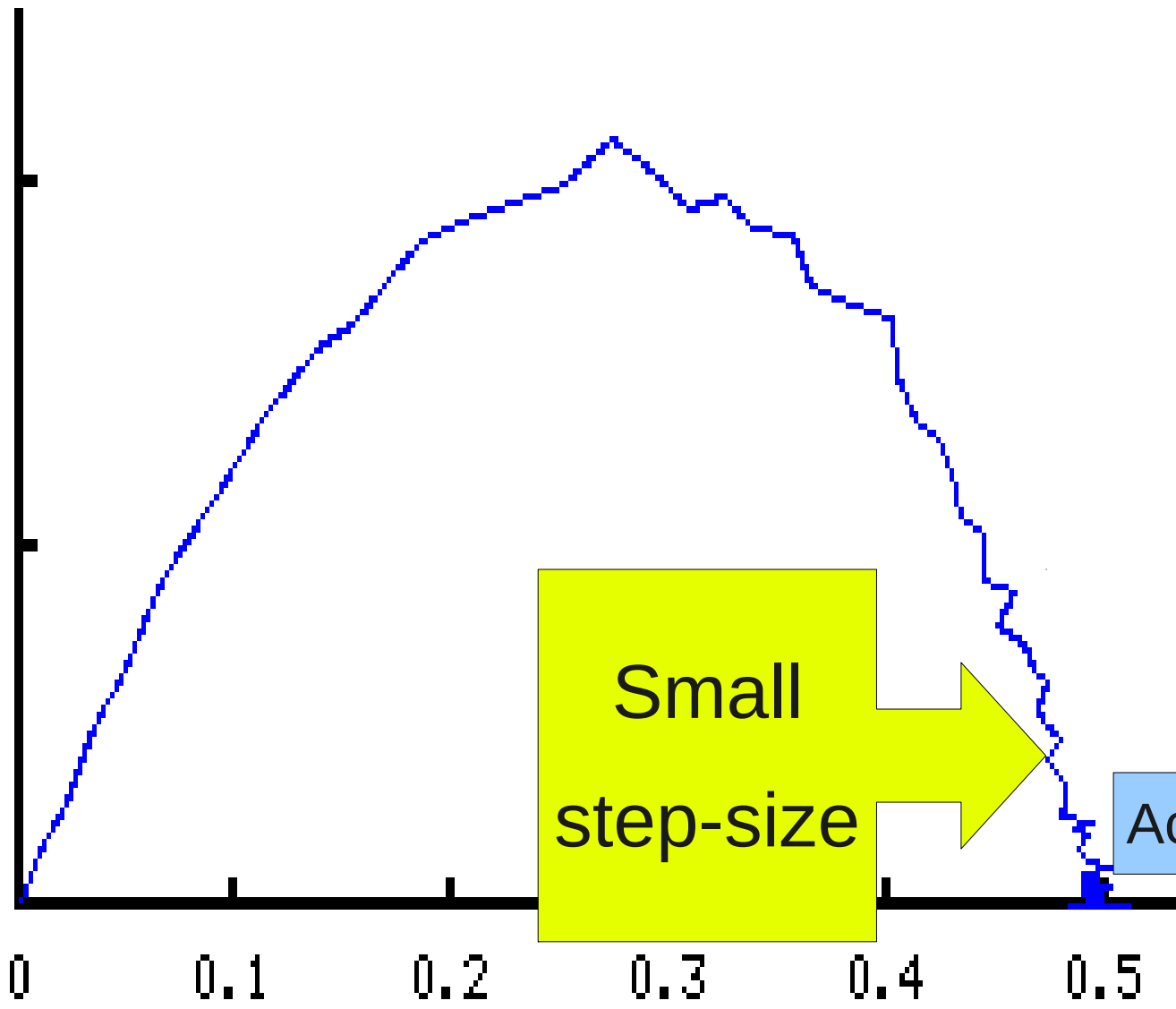
Progress rate



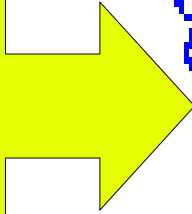
Big  
step-size

Acceptance rate

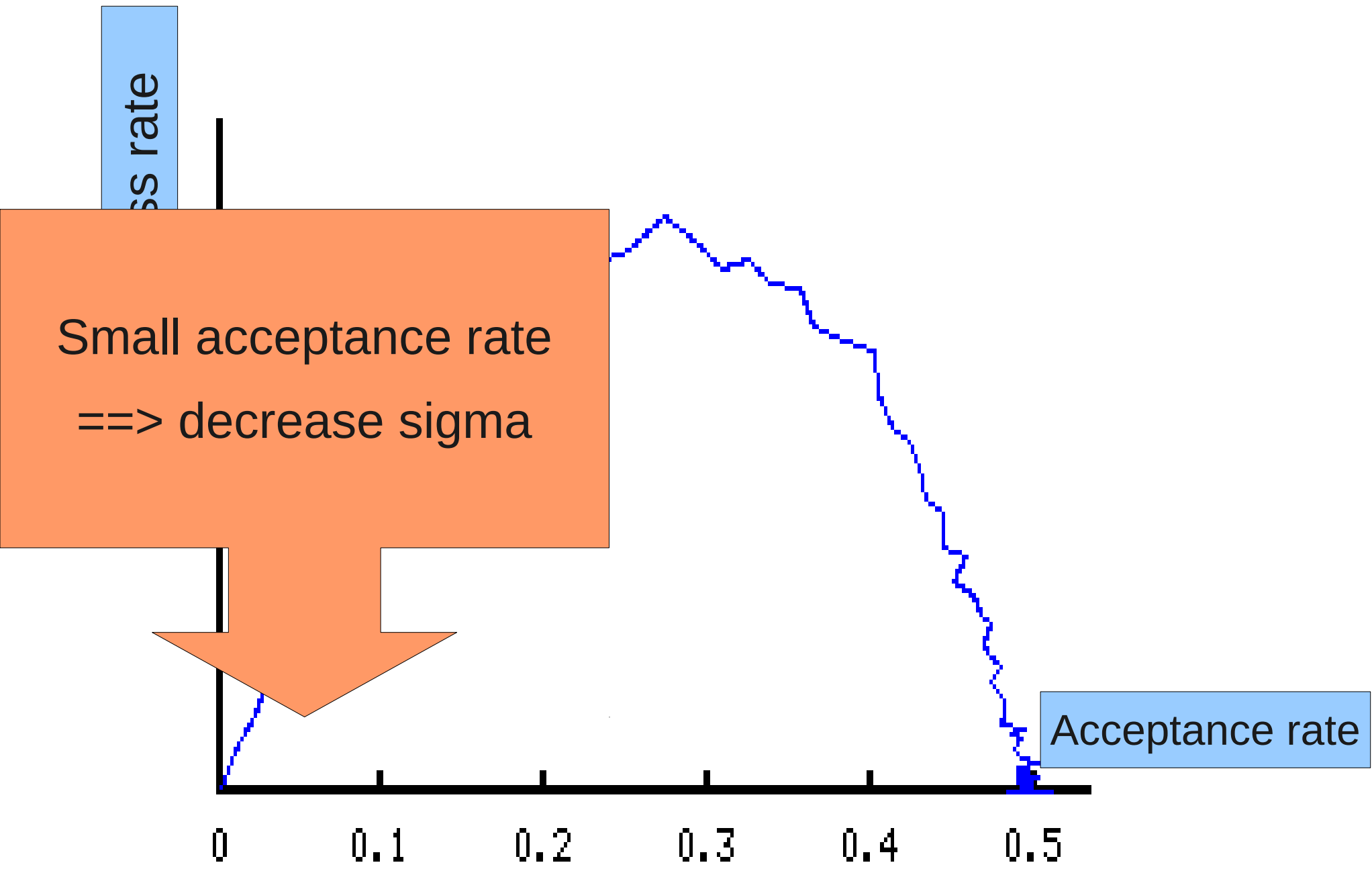
Progress rate



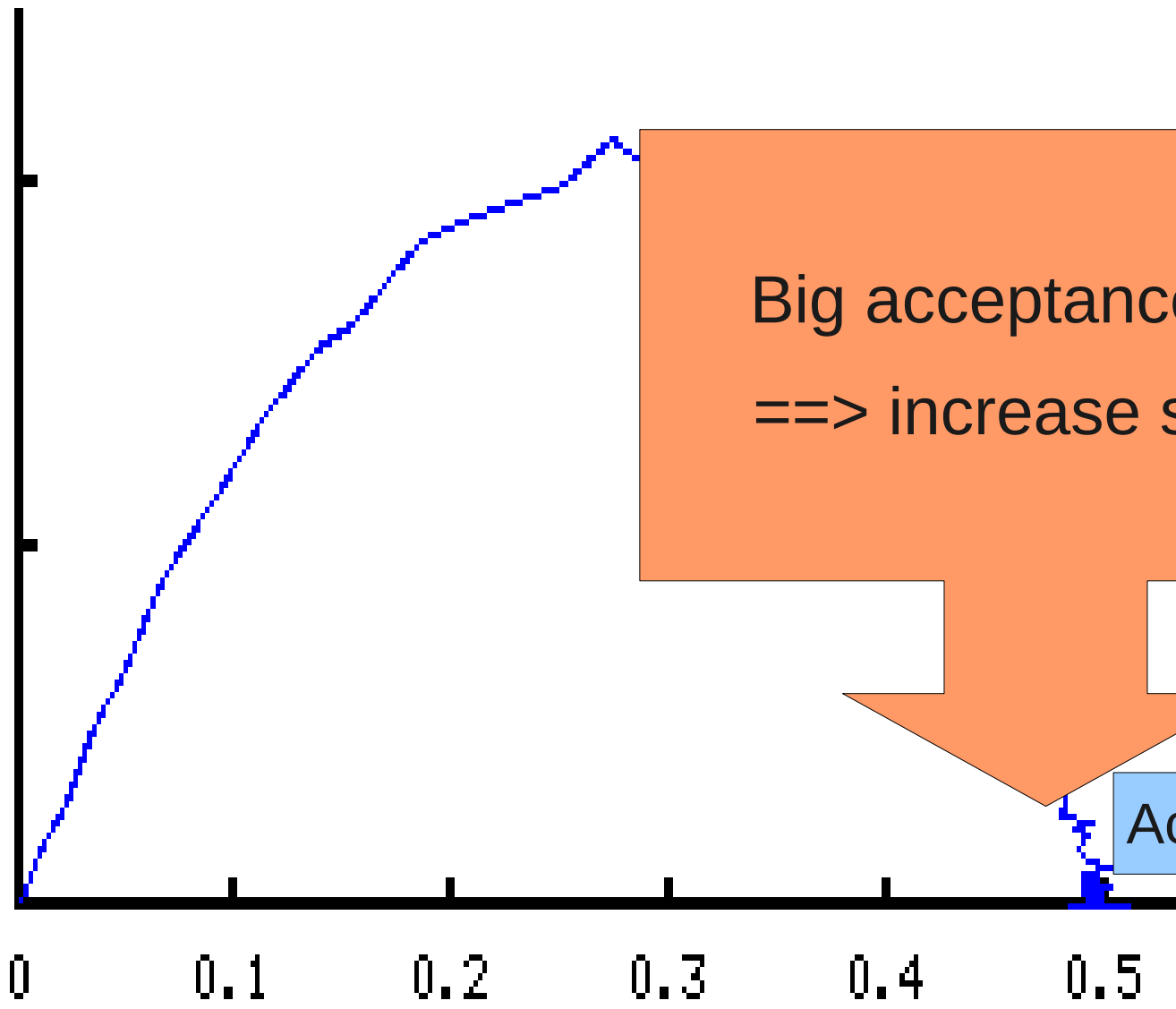
Small  
step-size



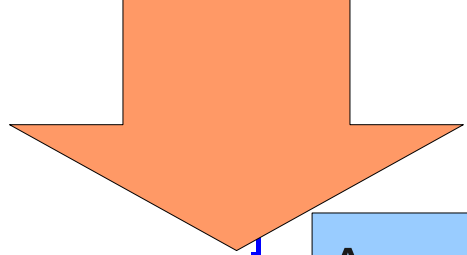
Acceptance rate



Progress rate



Big acceptance rate  
==> increase sigma



Acceptance rate

Based on maths showing  
that good step-size  
<==> success rate  $\approx 1/5$

$p_s$ : # of successful offspring / # offspring (per generation)

$$\sigma \leftarrow \sigma \times \exp\left(\frac{1}{3} \times \frac{p_s - p_{\text{target}}}{1 - p_{\text{target}}}\right)$$

Increase  $\sigma$  if  $p_s > p_{\text{target}}$

Decrease  $\sigma$  if  $p_s < p_{\text{target}}$

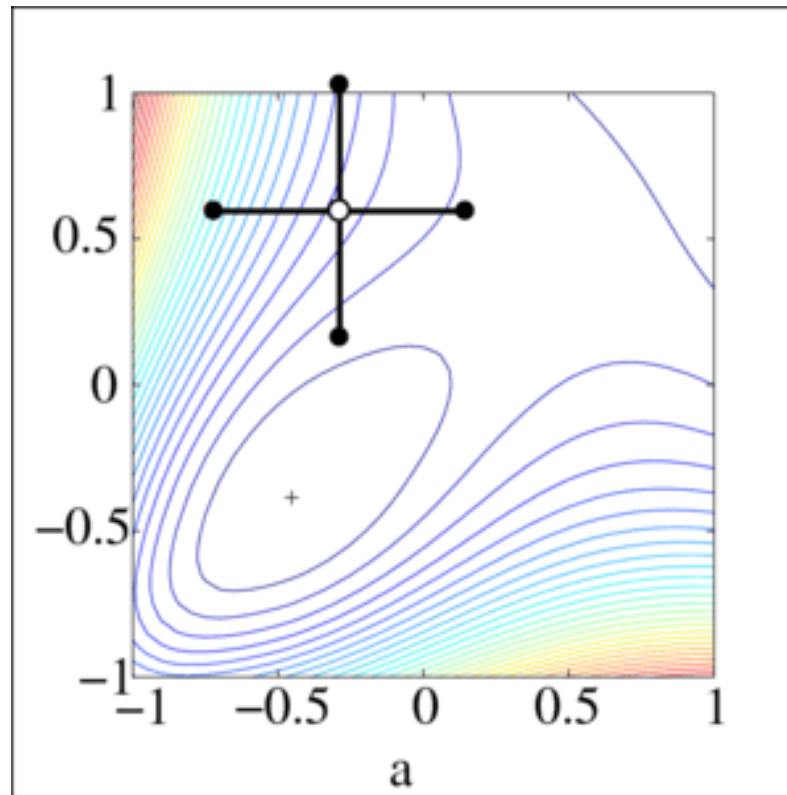
- I. Optimization and DFO
- II. Evolutionary algorithms
- III. From math. programming**
- IV. Using machine learning
- V. Conclusions

Olivier Teytaud

Inria Tao, en visite dans la belle ville de Liège

# III. From math. programming

==> pattern search method



## Comparison with ES:

- code more complicated
- same rate
- deterministic
- less robust

Olivier Teytaud

Inria Tao, en visite dans la belle ville de Liège

# III. From math. programming

Also:

- Nelder-Mead algorithm (similar to pattern search, better constant in the rate)

# III. From math. programming

Also:

- Nelder-Mead algorithm (similar to pattern search, better constant in the rate)
- NEWUOA (using value functions and not only comparisons)

- I. Optimization and DFO
- II. Evolutionary algorithms
- III. From math. programming
- IV. Using machine learning**
- V. Conclusions

Olivier Teytaud

Inria Tao, en visite dans la belle ville de Liège

# IV. Using machine learning

What if computing  $f$  takes days ?

$\implies$  parallelism

$\implies$  and “learn” an approximation of  $f$

# IV. Using machine learning

$$x_{n+1} = \text{Opt}(x_1, f(x_1), \dots, x_n, f(x_n)).$$

Statistical tools:  $f'(x)$  = approximation

$$(x, x_1, f(x_1), x_2, f(x_2), \dots, x_n, f(x_n))$$

$$y(n+1) = f'(x(n+1))$$

e.g.  $f'$  = quadratic function closest to  $f$  on the  $x(i)$ 's.

## IV. Using machine learning

==> keyword “surrogate models”

==> use  $f'$  instead of  $f$

==> periodically, re-use the real  $f$

- I. Optimization and DFO
- II. Evolutionary algorithms
- III. From math. programming
- IV. Using machine learning
- V. Conclusions**

Olivier Teytaud

Inria Tao, en visite dans la belle ville de Liège

## ***Derivative free optimization is fun.***

==> nice maths

==> nice applications + easily parallel algorithms

==> can handle really complicated domains  
(mixed continuous / integer, optimization  
on sets of programs)

Yet,

often suboptimal on highly structured problems (when  
BFGS is easy to use, thanks to fast gradients)

## *Keywords, readings*

==> cross-entropy (so close to evolution strategies)

==> genetic programming (evolutionary algorithms for automatically building programs)

==> H.-G. Beyer's book on ES = good starting point

==> many resources on the web

==> keep in mind that representation / operators are often the key

==> we only considered isotropic algorithms; sometimes not a good idea at all