# Evolutionary Algorithms

## Evolutionary Strategies

# Continuous representations

For some optimization problem the natural selection for representation are floating point numbers. In this case the genotype is an $(x_1, \ldots, x_k) \in \mathbb{R}^k$ vector. For example:

- if the variables have physical meaning (like angle, weight, length)
- finding global optimum for continuous functions
- finding optimal weights in a neural network

For continuous functions the gradient methods works fine, if the fitness function is differentiable and convex.

# Simulated Annealing

Let us suppose we have an $f : \mathbb{R}^n \to \mathbb{R}$ function, and we are looking for a global minimum.

We can use simulated annealing if $f$ is non convex or doesn't have a gradient or has plateaus.

We start with one random point, generate several new ones. If there is a better function value amongst the new points, we select the one with the best value for the next round, otherwise we select one randomly according to a distribution, where better points (that is with lower function values) have better chances to be selected.

# Simulated Annealing

The Simulated Annealing starts with one random point in $\mathbb{R}^n$, and generates new ones according to a distribution, which gives far point at the beginning of the iteration with high probability, although the excepted value of the difference in each coordinate is 0.

During the running of the algorithm we change the parameters, to lower the expected difference between the old and the new points, this assures the convergence.

# Generating new points

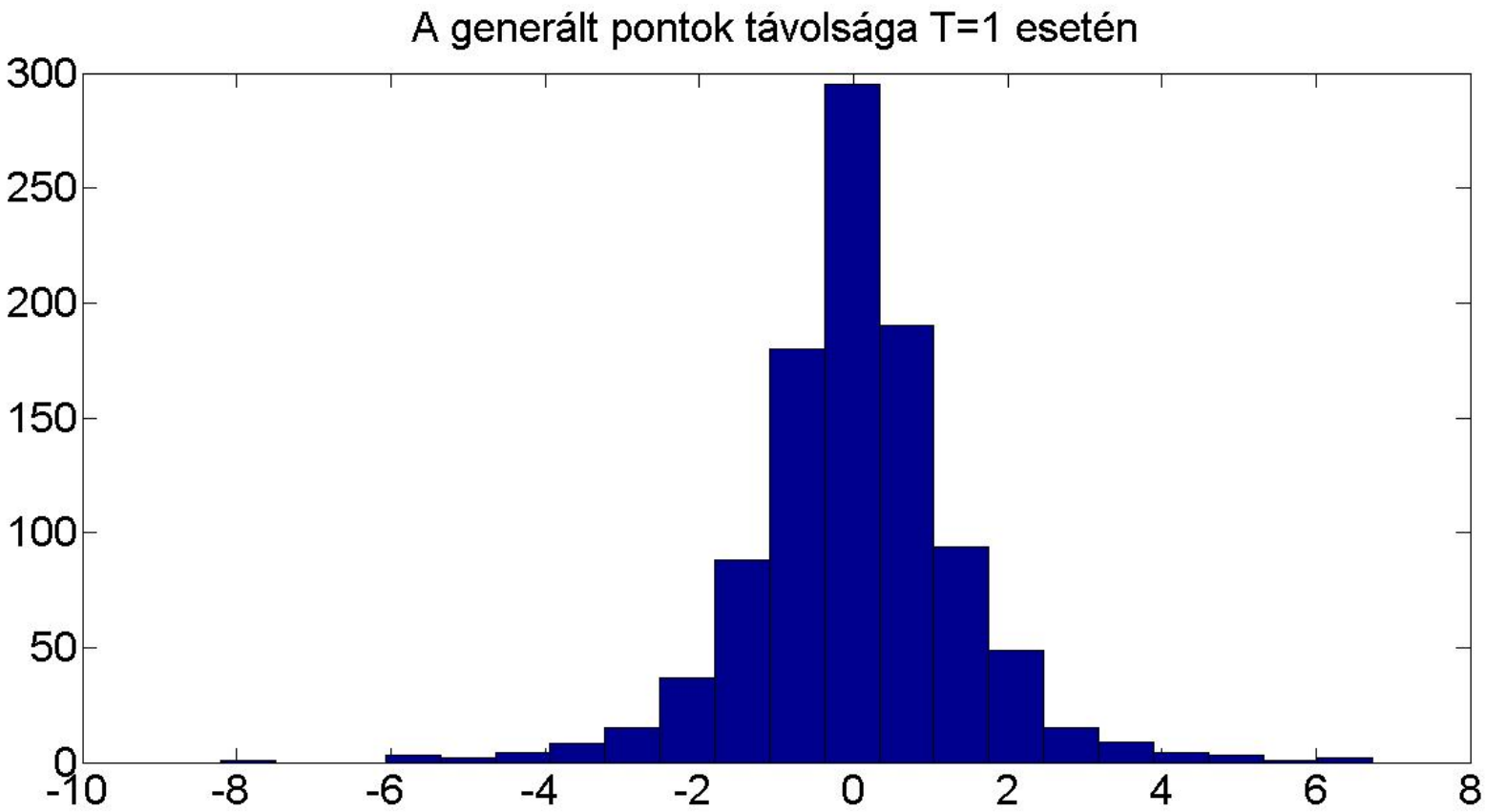
Let us denote our starting point by $x$, its coordinates are $x_1, x_2, \ldots, x_n$. Also, let us denote the new points by $u^1, u^2, \ldots u^m$. We can get the coordinates of the new points by generating two uniformly distributed random variables on $[0, 1]$, let's say rand1 and rand2, and then:

$$u_i^j = x_i + T \cdot (\ln(rand1) - \ln(rand2)),$$

where $T$ is a constant.

We can calculate the probability density function of the distribution of the new coordinates. It tells us that the coordinates of the new point falls in the $(x_i - k, x_i + k)$ interval with the probability $1 - exp\left(\frac{-k}{T}\right)$.

We decrease the value of $T$ in later iterations in order to achieve convergence (that is, the new points are close to their 'parent').


A generált pontok távolsága T=1 esetén
300
250
200
150
100
50
0
-10
-8
-6
-4
-2
0
2
4
6
8

# Selecting the new parent

If there is a new point, where the value of the function is less than for the parent point, then we choose that point as the new parent (if there is more than one, we choose the best).

If there is no such point (that is, the value of the function is greater in every new point than it was in the parent point), then let us calculate the quantities

$$g(j) = \exp\left(\frac{f(x) - f(u_j)}{T}\right)$$

for each $u^j$ point. We make a probability distribution from the $g(j)$ values by scaling, and select the new parent according to this distribution (like we did with the roulette-wheel method).

Since if $g(j)$ is bigger if the function value is smaller, we give a higher probability for better looking directions.

# Parameters

It's recommended to choose $T$ that has the same magnitude as the distance between the starting point and the optimum (if we have such information). This $T$ is sometimes called the temperature.

We decrease the temperature in every 10 iterations by multiplying it with a constant factor (around 0.9 is recommended). If we decrease it too fast we might end up with an early convergence, if we decrease too slow we might not converge at all.

It's not recommended to generate not too many new points, not only because it takes a lot of computation time, but also increases the danger of getting stuck in a local optimum.

# Comparison to genetic algorithms

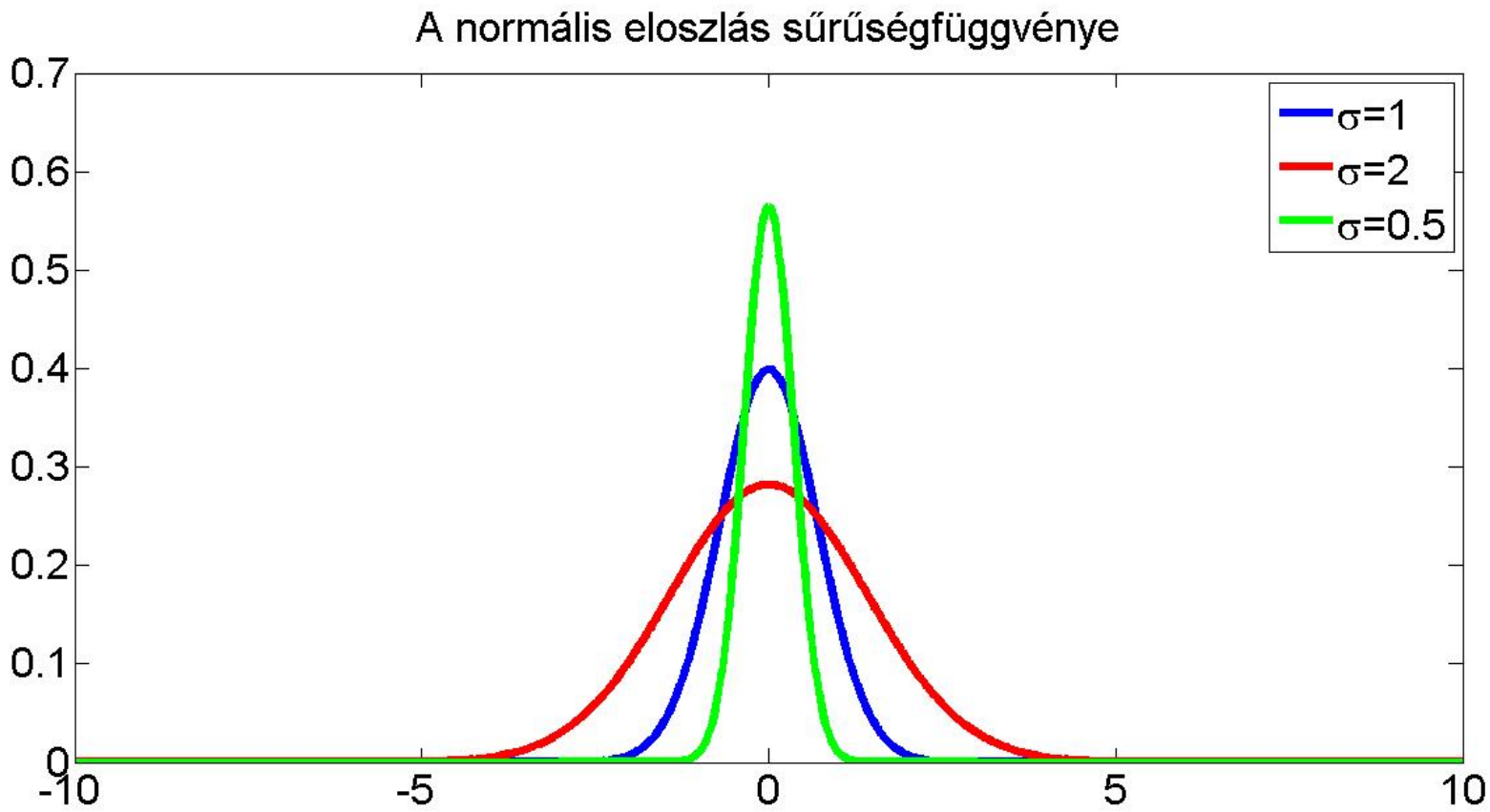
Simulated Annealing has some of the features of the genetic algorithms: we are getting closer to the optimum by changing the possible solution ('mutation'). We also create children similar to their parents, and try to select the fittest.

On the other hand we don't work on a population, only on one individual. And the choosing of the right parameters is also a hard problem, we will see an elegant solution to that when dealing with the next algorithm.

# Rechenberg's algorithm

Another option for functions without a smooth gradient is to design an evolutionary algorithm. Strictly speaking, this algorithm is not an evolutionary algorithm, but has many similarities to them. It differs from simulated annealing because

- It generates only one new point by adding to each coordinate of the parent a normally distributed random number (the expected value is always 0, the standard deviation is changing)
- If the new point is better, then it discards the old point, if not, then it discards the new one (elitism)
- The changes in standard deviation (step-size) is adapted to the function


A normális eloszlás sűrűségfüggvénye
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-10
-5
0
5
10
σ=1
σ=2
σ=0.5

# Rechenberg's 1/5 rule

The bigger the standard deviation ($\sigma$), the bigger the expected step-size of the Rechenberg's algorithm is.

In the beginning a greater value of $\sigma$ is recommended in order to explore the solution space. Later a smaller one is desirable to be able to have a precise estimate for the optimum.

If there are many successful steps, then we are heading in the right direction, in this case we should increase the value of $\sigma$. This way we get a quicker convergence.

If in most of the steps we discard the new point, then we are close to the optimum (since in every direction the function increases), so we have to decrease $\sigma$.

# Rechenberg's 1/5 rule

Let $0.817 < c < 1$ be fixed, and after 20 iterations the ratio of successful steps is denoted by $p$. After every 20 iterations, we change the value of $\sigma$:

$$\sigma := \begin{cases} \frac{\sigma}{c} & p > \frac{1}{5} \\ \sigma \cdot c & p < \frac{1}{5} \\ \sigma & p = \frac{1}{5} \end{cases}$$

# Evolutionary strategies

An evolutionary algorithm for a continuous problem has many similarities with Rechenberg's algorithm. We use the generation mechanism of the Rechenberg algorithm as mutation.

The representation of the solution will be an $\mathbb{R}^n$ vector (if the function has $n$ variables) *supplemented with a parameter vector storing the parameters of the algorithm*.

From now on an attribute of an individual is not only the function value itself, but also the expected step size (that is, it tells us something about the fitness of its offsprings). The algorithm not only optimizes the function value, but also the step size. In this way our algorithm is self-adaptive.

# Mutation

In continuous problems, mutation means the changing of every coordinate at once.

The most commonly used mutation operator is adding a normally distributed random number to each coordinate, whose expected value is 0, standard deviation is some $\sigma$, which is the parameter stored in the genotype of the individual. Hence the probability density distribution is:

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

With low probability it might give as an offspring that is very far away from its parent. If we wish to avoid stepping out of a bounded interval we might use a uniform distribution instead.

If we want to use a distribution which takes big steps with greater probability then the normal distribution we might consider using Cauchy-distribution. It's less concentrated on 0, it doesn't even have an expected value.

# Uniform stepsize

If we work with a uniform stepsize, then we have a genotype $(x_1, \ldots, x_n, \sigma)$, that has $n+1$ coordinates. We generate $n$ random normally distributed number independently with standard deviation $\sigma$ and add these numbers to each coordinates.

Let us denote the random variable with expected value 0, standard deviation $\sigma$ by $N(0, \sigma)$. Then

$$\sigma' = \sigma \mathrm{e}^{\tau N(0,1)}, \qquad \sigma' < \varepsilon_0 \to \sigma' = \varepsilon_0,$$

and

$$x_i' = x_i + N(0, \sigma)$$

We evaluate the quality of the solution two times: first when selecting parents (that is, how good the actual function value is), second when creating the offspring (how good offspring it's able to generate). Therefore it's important to change $\sigma$ first, and generate the offspring only after that.

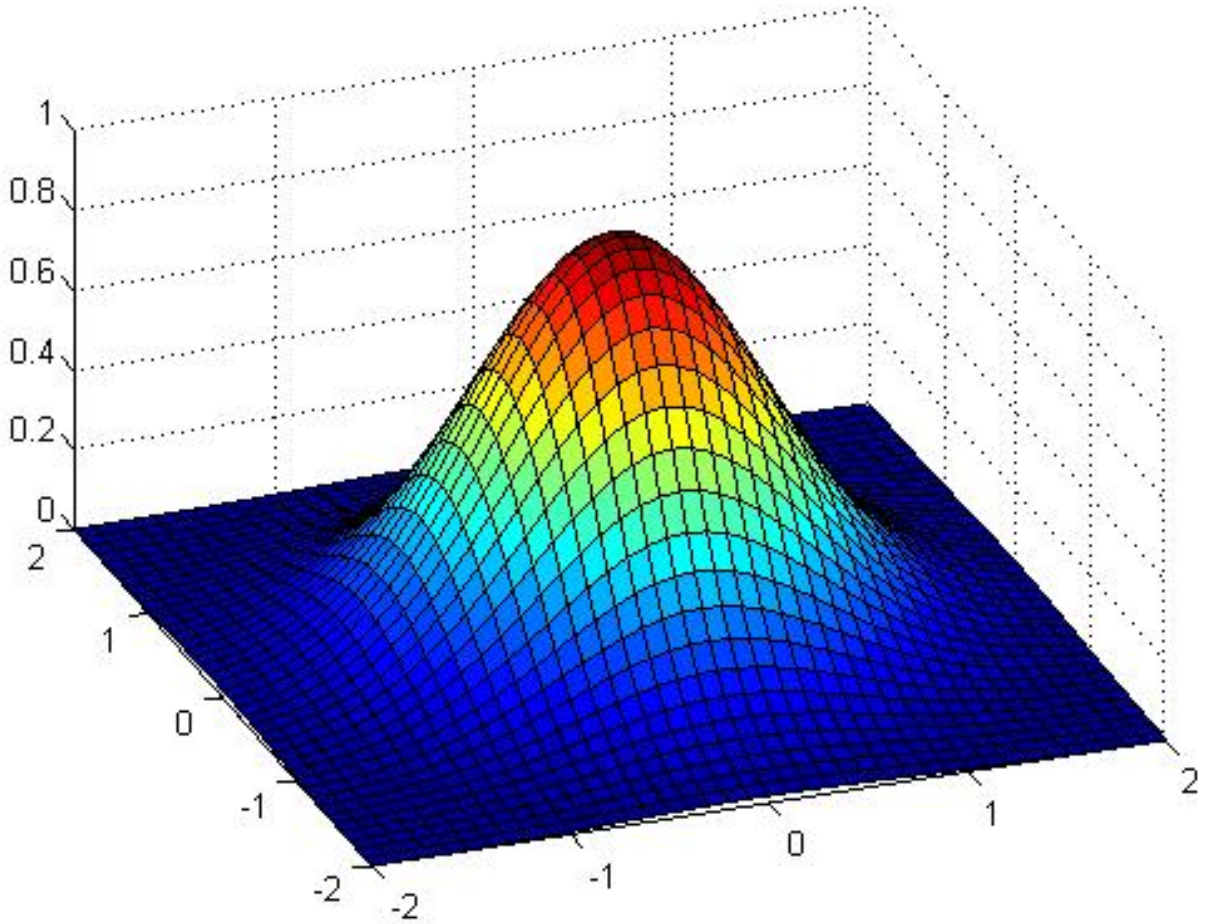
# Log-normal distribution

The distribution of $\mathrm{e}^{\tau N(0,1)}$ is called log-normal. We use this distribution to mutate the standard deviation because:

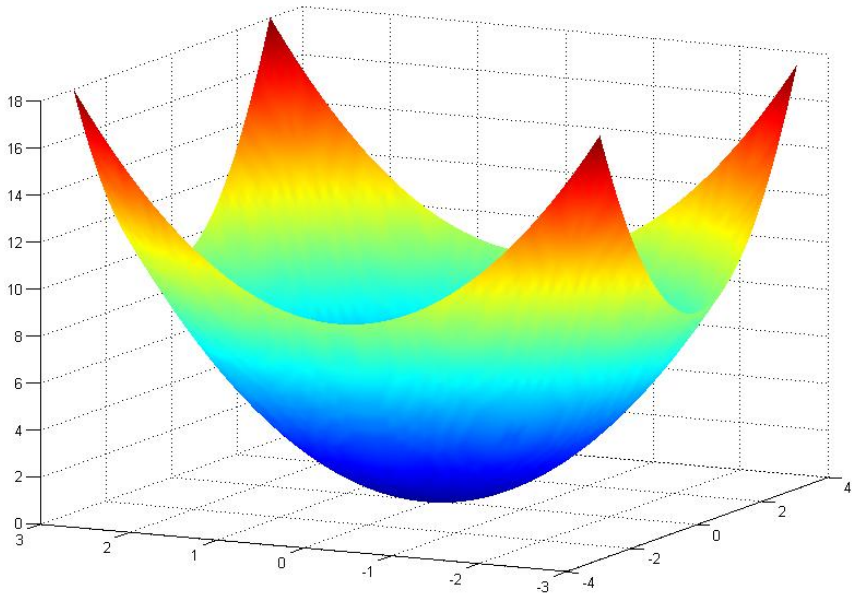
- mostly small changes are appropriate
- the median is 1, that is, increasing the standard deviance has the same probability as decreasing
- the mutation has to be neutral in the long term, and the expected value is 1.

Here the parameter $\tau$ is called the learning rate, it's recommended to be chosen $\tau \sim \frac{1}{\sqrt{n}}$.

# Uncorrelated normal distribution with the same variance

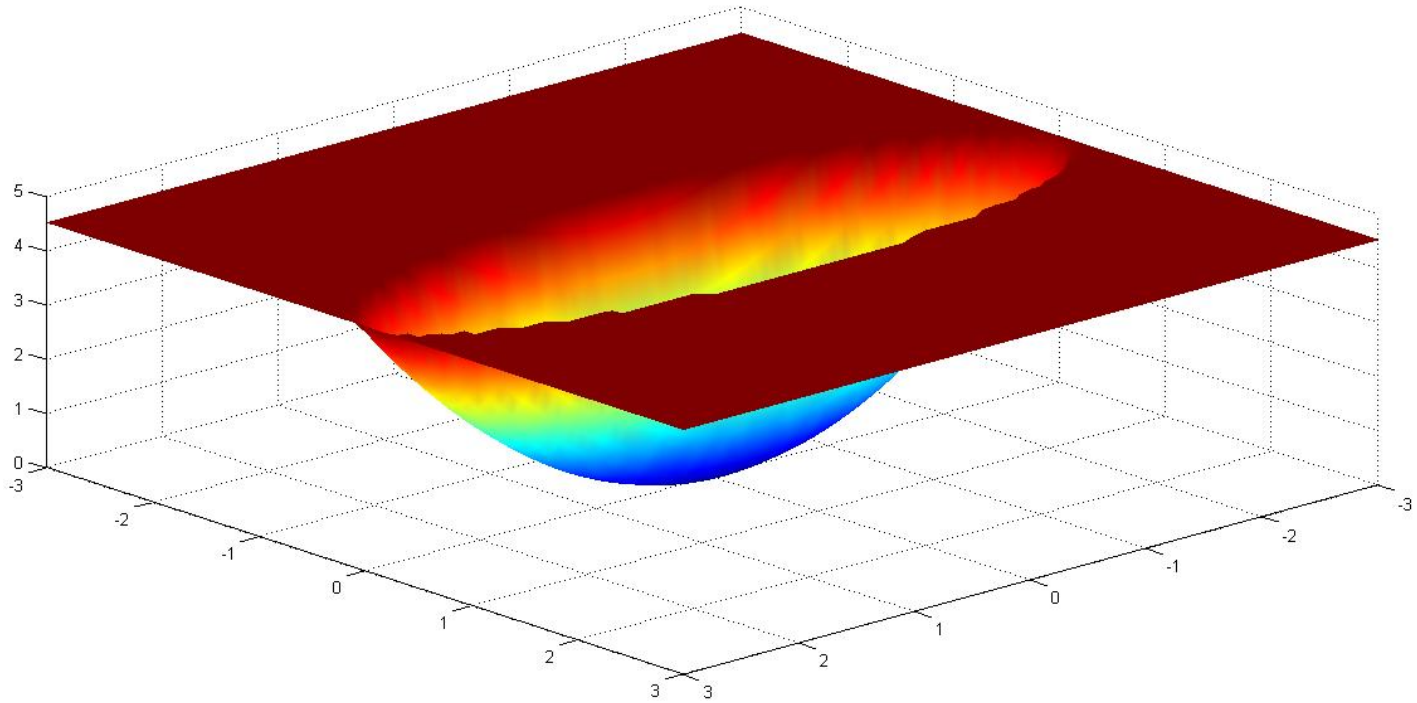


# 2 dimensional problems




This is a function of the type $a\xi^2 + b\eta^2$ . It's known, that every differentiable function look like this around its local optima, because the gradient is 0. Here $a \sim b$, and $a, b > 0$.

# 2 dimensional problems




Here $a >> b > 0$, this is called the long valley problem. It causes difficulties, because taking same sized step in each coordinate is no longer efficient.

# Uncorrelated mutation

We change the genotype to $(x_1, x_2, \ldots, x_n, \sigma_1, \sigma_2, \ldots, \sigma_n)$. Our algorithm now learns the appropriate step size (standard deviation) in each coordinate.
In this case

$$\sigma_i' = \sigma_i e^{\tau N(0,1) + \tau' N_i(0,1)}, \qquad \sigma' < \varepsilon_0 \to \sigma' = \varepsilon_0,$$

and

$$x_i' = x_i + N_i(0, \sigma_i)$$

Where the index $i$ means, that we generate the random number coordinate-wise.

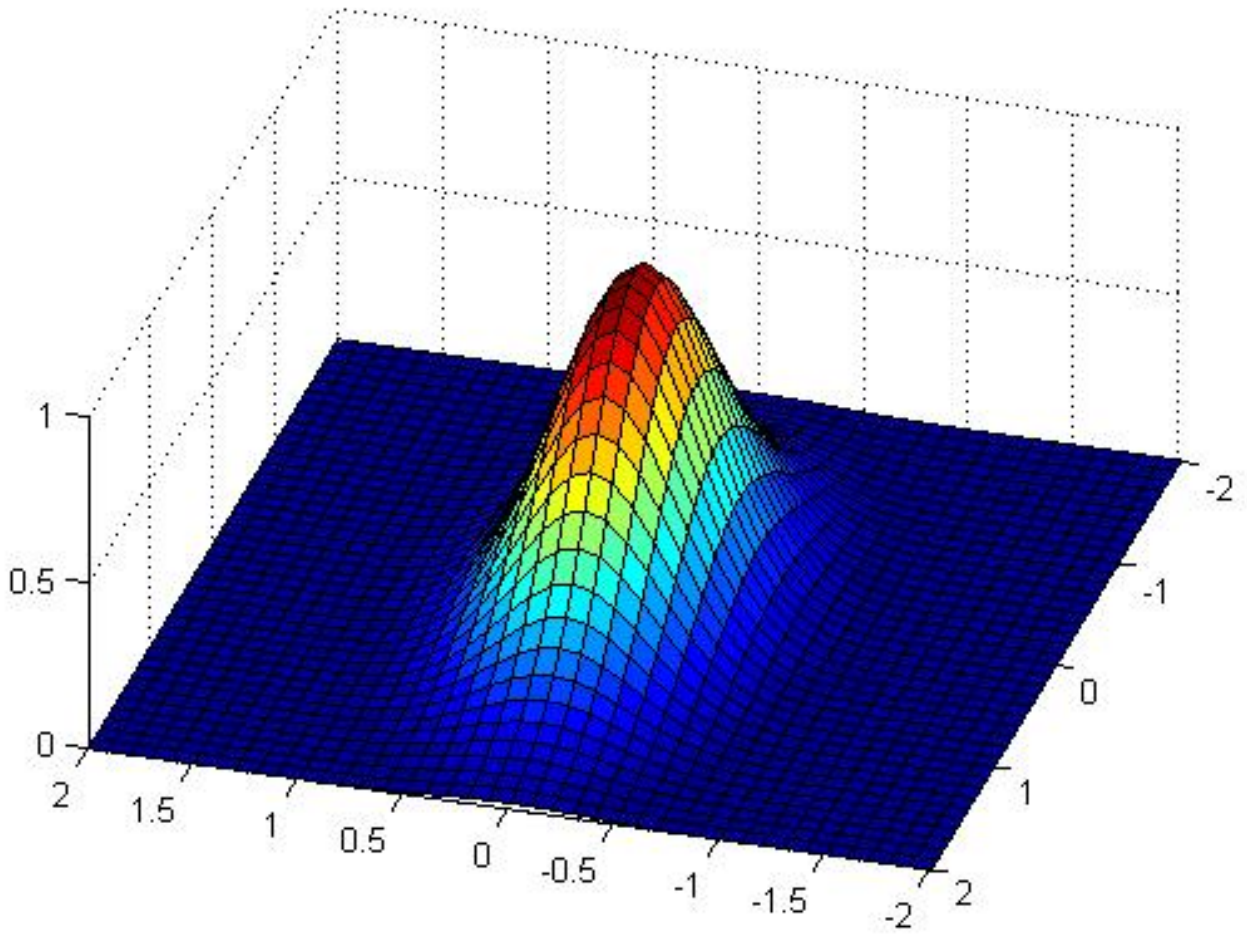
# Mutation of $\sigma$

The sum of two independent normally distributed random variables is also a normally distributed random variable (the new expected value is the sum of the expected values, the new standard deviation is $\sqrt{\sigma_1^2 + \sigma_2^2}$). It means that the multiplier of the mutation is also a log-normally distributed random variable with expected value 1.

Instead of a single $\tau$, we have $\tau$ and $\tau'$ parameters now. One that doesn't depend on the coordinate $\tau N(0,1)$, that controls the overall convergence rate. The other one, that depends on the coordinate $\tau' N_i(0,1)$, that makes us possible to use different step sizes in each coordinate.

It's recommended to choose $\tau \sim \frac{1}{\sqrt{2n}}$ and $\tau' \sim \frac{1}{\sqrt{2\sqrt{n}}}$.

# Uncorrelated normal distribution with different variance



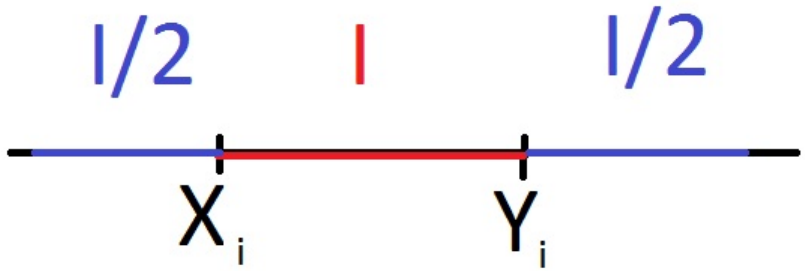
# Crossover operators

There are several options for contentious problems:

- discrete crossover, where the allele of the offspring is simply copied from one of the parents. It is recommended on the part of the vector where the coordinates are stored in order to maintain the diversity.
- arithmetic crossover: let $0 < \alpha < 1$ one of the offspring is $\alpha x + (1 - \alpha)y$, the other is $(1 - \alpha)x + \alpha y$. It is recommended on the part where the parameters are stored.
- Blend: instead of the previous $\alpha$ we choose a random number on $[-.5, 1.5]$ to avoid early convergence.

# Blend crossover

Let $\alpha$ be $2u - 0.5$, where $u$ is uniform random number on $[0, 1]$. In this case the offspring is between the parent with probability $1/2$ and its distribution is uniform on $[1.5x_i - y_i, 1.5y_i - 0.5x_i]$. We maintain the diversity, hence this operator is suitable to be used on the coordinates part of the genome.




The crossover is global, it is possible to have different parents in each coordinates.

# Selecting survivors

For continuous problems, we typically generate many more offspring in one generation (let's say $\lambda$), and then select the best $\mu$ (according to the function value). The selection pressure is the ratio of $\mu$ and $\lambda$, it's usually between $\frac{1}{7}$ and $\frac{1}{4}$.

It is recommended to select the new generation's individuals only from the offspring (the parents not included), because it reduces the chance that a single individual dominates the population with a good function value but an inappropriate step-size.

For this reason the elitism isn't recommended.