# Optimization

Optimization means finding the minimum value of a function.

We have to find the minimum of a function

$$f(\mathbf{x}), \mathbf{x} = (x_1, \ldots, x_N).$$

The function $f$ is called the *objective function*.

Very many problems can be expressed as optimization problems:

1 Solving an equation $f(x) = 0$. The root is also the minimum of the function $|f|$. Since this is a function with a lower bound, it has at least an infimum, even if the original equation has no solution. Particularly a set of nonlinear equations may be difficult to solve, in which case it can be handled as an optimization problem.

2 Least squares fit is found by minimizing the residual. In nonlinear cases there is no analytical solution, and the minimum has to be found directly by optimization. The form of the function has no real effect on the complexity of the problem.

3 If the criterion is something else than least squares, optimization of the residual is usually the only possible method.

4 Solution of regularized problems. The regularization parameter can be one of the variables to be determined.

5 Combinatorial problems (like the travelling salesman problem). Some may be NP complete: the time needed for the solution proportional to $2^n$ or $n!$.

If there are no additional requirements, the problem is unconstrained.

In a constrained problem the solution must also satisfy given constraints that can be
- inequalities: $g(\mathbf{x}) \leq 0$
- equations: $h(\mathbf{x}) = 0$.

If there are several variables and the method employs derivatives, we may need the Hessian formed from the second derivatives:

$$H(\mathbf{x}) = \begin{pmatrix} \frac{\partial^2 f(\mathbf{x})}{\partial x_1^2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_n} \\ \vdots & & \vdots \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_n} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_n^2} \end{pmatrix}.$$

If the matrix is positive/negative definite the point is a local minimum/maximum.

**Local optimization**

1. Derivatives of the objective function are known. E.g. conjugate gradient methods.

2. Derivatives are not known or difficult to calculate. E.g. polytope method.

**Global optimization**

There are no absolutely reliable algorithms.

- Find many local optima by starting from different points.
- Genetic algorithms.
- Simulated annealing.

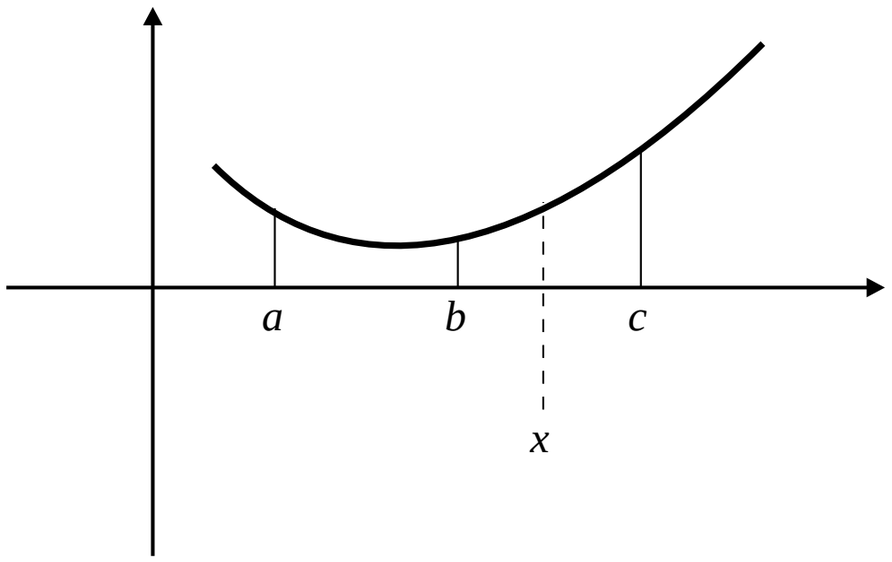
## Forking method

Onedimensional case, derivative is not known.

The funkction is known at three points $a < b < c$ such that $f(b) < f(a)$ and $f(b) < f(c)$.

Select a point $x$ e.g. in the interval $(b, c)$. If $f(x) < f(b)$, the new interval will be $(b, c)$, otherwise $(a, x)$.

Repeat the same operation until $f(x)$ decreases no more.



At the minimum $f'(x) = 0$; thus the change of $f$ is at most of the second order. Iteration can be terminated when the length of the interval is of the order $\sqrt{\epsilon}$, where $\epsilon$ is the machine constant. Smaller changes in $x$ will not affect the value of the function.

Golden section rule: the middle point is chosen so that

$$\frac{b-a}{c-a} = \frac{3-\sqrt{5}}{2} = 0.38197.$$

The next point is chosen in the longer subinterval so that

$$\frac{x-b}{c-b} = 0.38197.$$

Onedimensional case, derivative is known

Assume again that $a < b < c$ and $f(b) < f(a)$ and $f(b) < f(c)$.

Find $f'(b)$. If $f'(b) > 0$, the next point is chosen in the interval $(a, b)$, otherwise in $(b, c)$.

When the derivative has been calculated at two points, we can use e.g. the secant method to find the next point, where the derivative should be zero. If this point is outside the interval, e.g. the midpoint of the interval can be taken as the new point.

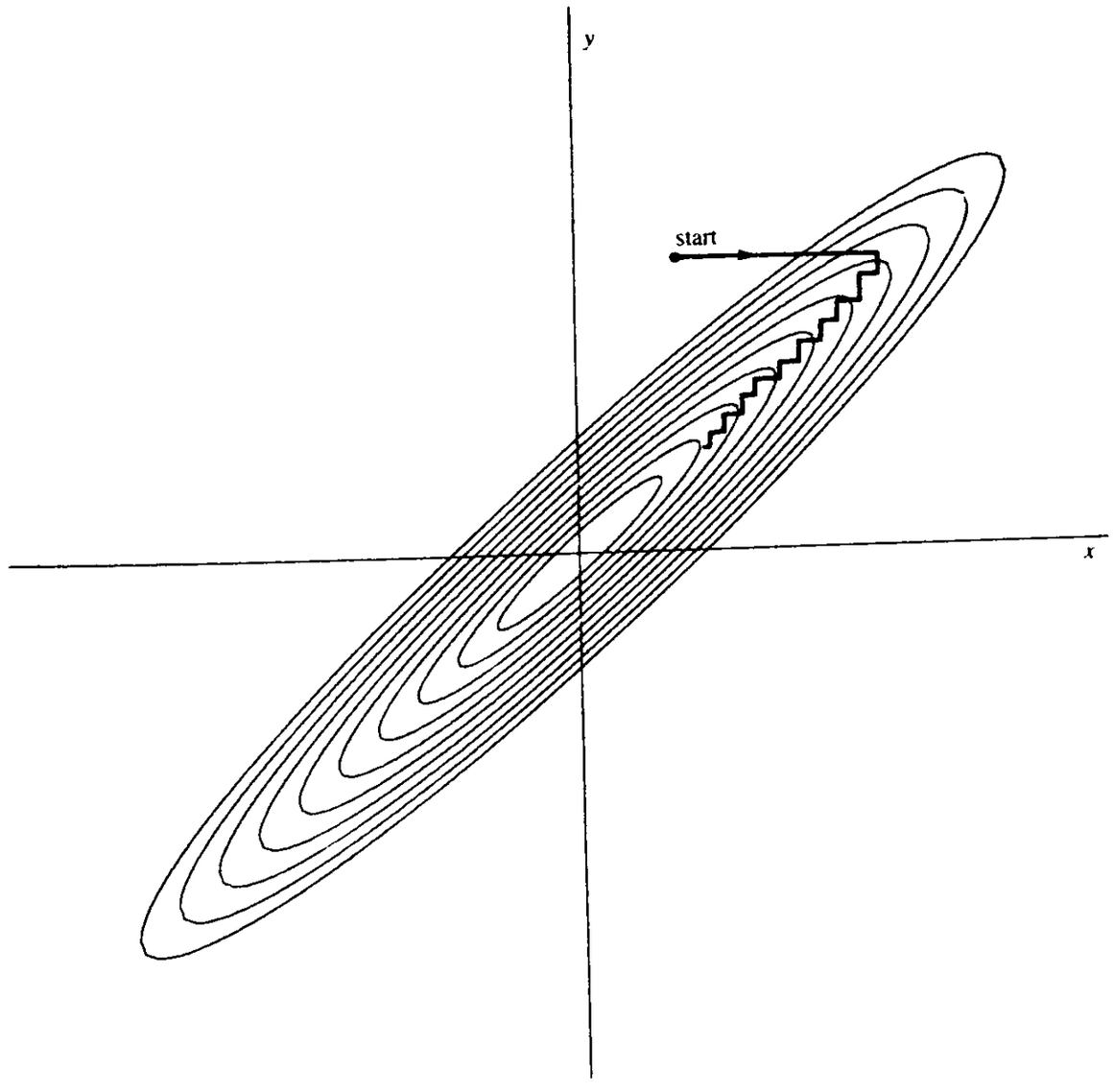
### $N$-dimensional case

Proceed in some direction as long as the objective function decreases, and then change the direction. This line search is repeated until no smaller values are found.

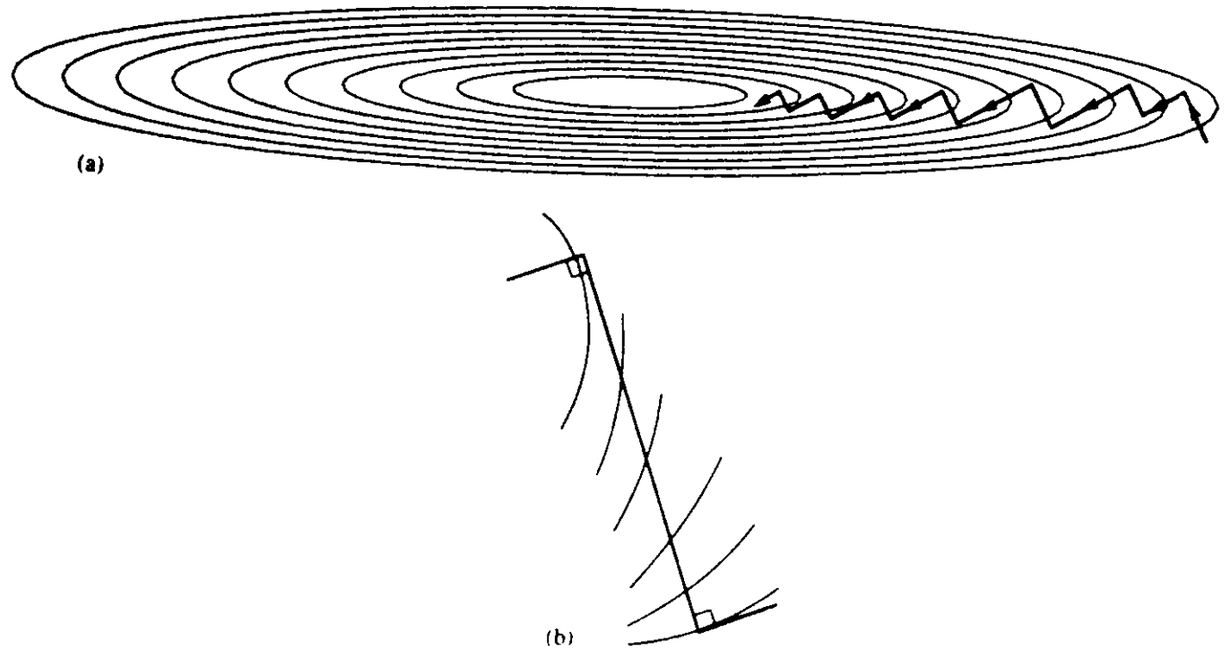
At each step a onedimensional optimization problem is solved. Advance in a given direction ends, when the gradient of the objective function has no nonzero component in that direction.

The direction can be chosen in different ways.

1. Proceed in the direction of coordinate axes only; change the coordinates $x_1, \ldots, x_N, x_1, \ldots$ alternatively. Often inefficient.



2. Proceed in the direction of the steepest descent (i.e the gradient of the objective function). After each line search turn 90°. This may not lead very directly towards the minimum.



3. Try to choose the next direction by taking into account the previous directions.

## Conjugate gradient method

Let $\mathbf{x}$ be the current point with respect to the location $\mathbf{p}$ of the best minimum found this far. then

$$f(\mathbf{x}) \approx f(\mathbf{p}) + \sum_i \frac{\partial f}{\partial x_i} x_i + \frac{1}{2} \sum_{i,j} \frac{\partial^2 f}{\partial x_i \partial x_j} x_i x_j$$

$$= c + \mathbf{b} \cdot \mathbf{x} + \frac{1}{2} \mathbf{x} \cdot \mathbf{A} \cdot \mathbf{x},$$

where $\mathbf{b}$ is the gradient of the function $f$ at $\mathbf{p}$ and $\mathbf{A}$ the Hessian at $\mathbf{p}$. Differentiating this we get an expression for the gradient (at $\mathbf{x}$)

$$\nabla f = \mathbf{A} \cdot \mathbf{x} + \mathbf{b}.$$

When we move from $\mathbf{x}$ in the direction of $\mathbf{v}$, the change of the gradient is $\mathbf{A} \cdot \mathbf{v}$.

Assume that the previous step was taken in the direction $\mathbf{u}$. To avoid losing this advantage the gradient must pe perpendicular to $\mathbf{u}$ also after the step $\mathbf{v}$:

$$\mathbf{u} \cdot \mathbf{A} \cdot \mathbf{v} = 0.$$

Such directions $\mathbf{u}$ and $\mathbf{v}$ are called conjugates.

1. Choose an initial value $\mathbf{x}_1$.

2. Repeat the following, $k = 1, 2, \ldots$, until the norm $\| \mathbf{g}_k \|$ is sufficiently small:
   - $\mathbf{g}_k = \nabla f(\mathbf{x}_k)$.
   - if $k = 1$, set $\beta_1 = 0$, $\mathbf{s}_0 = 0$; otherwise

$$\beta_k = \frac{\mathbf{g}_k \cdot \mathbf{g}_k}{\mathbf{g}_{k-1} \cdot \mathbf{g}_{k-1}}$$

   - $\mathbf{s}_k = -\mathbf{g}_k + \beta_k \mathbf{s}_{k-1}$.
   - use line search to find a point $\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda_k \mathbf{s}_k$. So we proceed in the direction of $\mathbf{s}_k$ until a minimum is found.

## Polytope method

Nelder–Mead polytope method; called also a simplex method (which is quite different from the simplex method of linear optimization problems). Derivatives of the objective function are not needed.

The simplex of an $N$-dimensional space is a polygon with $N + 1$ apices, $\mathbf{x}_1, \ldots, \mathbf{x}_{N+1}$. Denote $f_i = f(\mathbf{x}_i)$, $f_l = \min f_i$, $f_h = \max f_i$.

The centre of gravity of the face opposite the worst apex

$$\mathbf{x}_c = \frac{1}{n} \sum_{i \neq h} \mathbf{x}_i.$$

Reflection coefficient $\alpha = 1$, reduction coefficient $\beta = 0.5$, expansion coefficient $\gamma = 2$.

Four kinds of operations are used to transform the simplex:

1. Reflect the worst apex w.r.t the centre of gravity of the opposite face:

$$\mathbf{x}_r = (1+\alpha)\mathbf{x}_c - \alpha\mathbf{x}_h.$$

2. Expand the simplex:

$$\mathbf{x}_e = (1-\gamma)\mathbf{x}_c + \gamma\mathbf{x}_r.$$

3. Shrink the simplex by moving one apex:

$$\mathbf{x}_s = (1-\beta)\mathbf{x}_c + \gamma\mathbf{x}_h.$$

4. Shrink the simplex by moving one face.

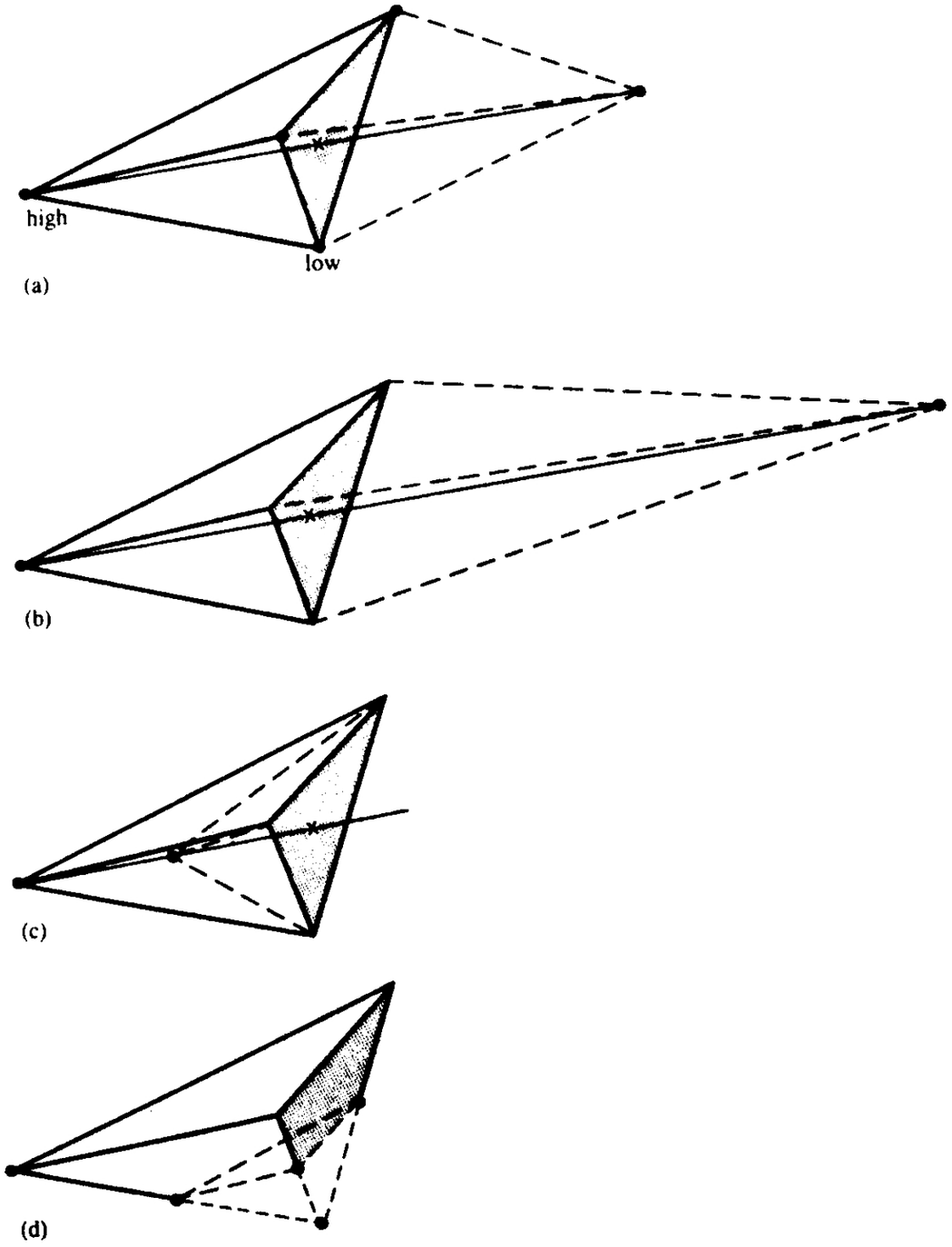



Figure 10.4.1. Possible outcomes for a step in the downhill simplex method. The simplex at the beginning of the step, here a tetrahedron, is drawn with solid lines. The simplex at the end of the step (drawn dashed) can be either (a) a reflection away from the high point, (b) a reflection and expansion away from the high point, (c) a contraction along one dimension from the high point, or (d) a contraction along all dimensions toward the low point. An appropriate sequence of such steps will always converge to a minimum of the function.

```
program amoeba
! Nelder-Mead polytope or simplex method
! adapted from a C program in Numerical recipes
 implicit none
 integer, parameter :: NMAX = 1000, &  ! max nr of steps
                ndim = 2, &    ! nr of dimensions
                mpts = 3      ! nr of apices of the simp,lex
 real, parameter :: FTOL = 0.001, &   ! final tolerance
              ALPHA = 1.0, &    ! reflection factor
              BETA = 0.5, &     ! shrinking factor
              GAMMA = 2.0       ! expansion factor
 integer i,j,ilo,ihi,inhi, nfunk
 real ytry, ysave, sum, rtol, psum(ndim), &
    p(ndim+1, ndim), &
    y(ndim+1)
! apices of the initial simplex
 p(1,:) = (/ 0.0, 0.0 /)
 p(2,:) = (/ 0.0, 1.0 /)
 p(3,:) = (/ 1.0, 0.0 /)
```

```
! function values at the apices
do i=1,3
 y(i) = funk(p(i,:))
end do

! sums of coordinates; the centre of gravity of the
! face opposite of apex i is (psum-p(i))/ndim
do j=1,ndim
 sum=0.0
 do i=1,mpts
  sum = sum+p(i,j)
 end do
 psum(j) = sum
end do
nfunk=0
```

```
main: do
  ! find the best (ilo), worst (ihi) and
  ! second worst (inhi) apex
  ilo=1
  if (y(1) > y(2)) then
    inhi=2; ihi=1
  else
    inhi=1; ihi=2
  end if
  do i=1,mpts
   if (y(i) < y(ilo)) ilo=i
   if (y(i) > y(ihi)) then
     inhi=ihi; ihi=i
  else if (y(i) > y(inhi)) then
    if (i /= ihi) inhi=i
  end if
end do
```

```
! if the worst and best almost same the minimum is found
rtol=2.0*abs(y(ihi)-y(ilo)) / (abs(y(ihi))+abs(y(ilo)))
if (rtol < FTOL) then
     write(6,’("rtol=",f10.5)’) rtol
     write(6,’("nfunk=",i5)’) nfunk
     exit main
end if
! stop if too many iterations
if (nfunk >= NMAX) then
     write(6,’("nfunk=",i5)’) nfunk
     exit main
end if
```

```
! reflection
ytry=amotry(-ALPHA)

! if the result improved, move the new apex even further
if (ytry <= y(ilo)) then
 ytry=amotry(GAMMA)
else if (ytry >= y(inhi)) then
 ! the new point is worst than the second worst;
 ! shrink the simplex
 ysave=y(ihi)
 ytry=amotry(BETA)
 if (ytry >= ysave) then
! still too big; shrink the simplex w.r.t the best point
do i=1,mpts
 if (i /= ilo) then
  do j=1,ndim
   psum(j)=0.5*(p(i,j)+p(ilo,j))
   p(i,j)=psum(j)
  end do
  y(i)=funk(psum)
 end if
```

```
      end do
      nfunk = nfunk+ndim
      ! update sums of coordinates
      do j=1,ndim
        sum=0.0
        do i=1,mpts
          sum = sum+p(i,j)
        end do
        psum(j) = sum
      end do
    end if
  end if
end do main

! print the points and corresponding function values;
! they should be almost equal since the simplex has
! shrunk to a small neighbourhood of the optimum
write(*,*) p(1,:), y(1)
write(*,*) p(2,:), y(2)
write(*,*) p(3,:), y(3)
```

```
contains

! objective function
real function funk(p)
 implicit none
 real p(ndim)
 funk=(p(1)-3.0)**2+(p(2)-2.0)**2+1.0
end function
```

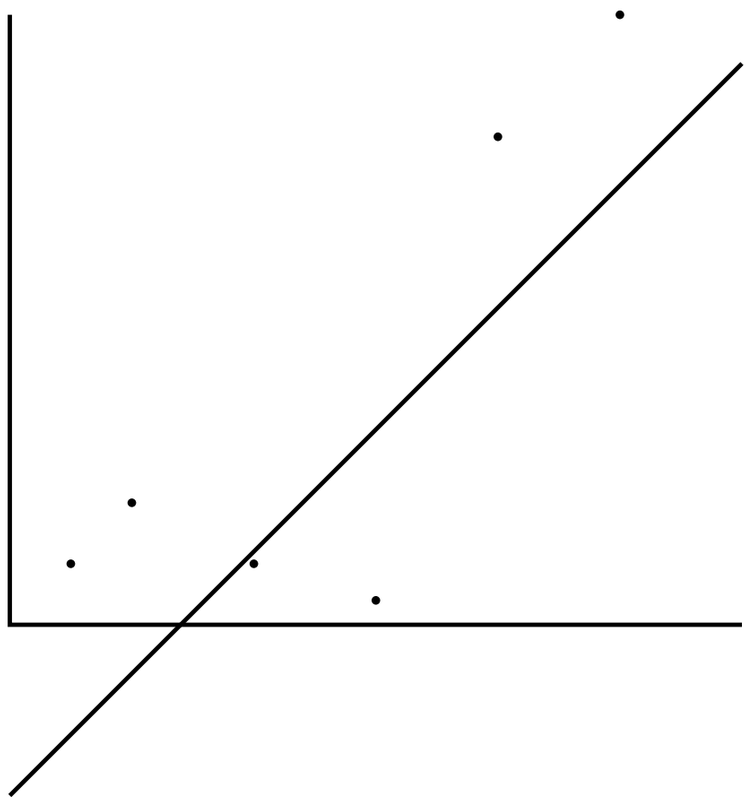
```
! modify the simplex
! the worst apex is reflected w.r.t. the opposite face
! and the distance is multiplied by fac
real function amotry(fac)
 implicit none
 real fac, fac2
 integer j
 real fac1, ytry, ptry(ndim)
 fac1=(1.0-fac)/ndim
 fac2=fac-(1-fac)/ndim
 do j=1,ndim
   ptry(j)=psum(j)*fac1+p(ihi,j)*fac2
end do
ytry=funk(ptry)
nfunk = nfunk+1
if (ytry < y(ihi)) then
 y(ihi)=ytry
 do j=1,ndim
  psum(j) = psum(j)+ptry(j)-p(ihi,j)
  p(ihi,j)=ptry(j)
 end do
```

```
   end if
   amotry=ytry
  end function
end program

rtol=  0.00068
nfunk=  31
  3.009915     2.024210     1.000684
  2.985695     2.014316     1.000410
  3.000000     2.000000     1.000000
```

```
program amotest
 use amoeba
 implicit none
 interface
  real function funk(p, ndim)
   integer, intent(in) :: ndim
   real, dimension(ndim) :: p
  end function
 end interface

 integer, parameter :: ndim = 2    ! dimension of the space
 real p(ndim+1, ndim), &
    y(ndim+1)

 ! initial simplex
 p(1,:) = (/ 0.0, 0.0 /)
 ...
 call simplex(funk, 2, p, y)
 write(*,*) p(1,:), y(1)
 ...
end program
```

```
real function funk(p, ndim)
! L_infty residual
 implicit none
 integer, intent(in) :: ndim
 real, dimension(ndim) :: p
 integer, parameter :: ndata=6
 real, dimension(ndata) :: &
     xx = (/ 0.5, 1.0, 2.0, 3.0, 4.0, 5.0 /), &
     yy = (/ 0.5, 1.0, 0.5, 0.2, 4.0, 5.0 /)
 integer i
 real s

 s=max(abs(yy-p(2)-p(1)*xx))

  funk=s
end function
```

```
rtol=  0.00094
nfunk=  44
  1.001739    -1.402117     1.403100
 0.9989247    -1.397929     1.403306
 0.9990266    -1.392659     1.404421
```



## Constraints

The solution vector must be updated in such a way that the constraints are always satisfied.

Programs for unconstrained problems can be used to solve constrained problems, if the constraints are replaced by penalty or barrier functions.

An extra term is added to the function to be optimized; this term is very big outside the allowed region. If we are optmizing $f$ with the constraint $g(x) \leq 0$, the objective function could be

$$f(x) + s \max\{0, g(x)\},$$

where the constant $s$ is a penalty parameter (the fine paid for crossing the boundary).

A barrier function is a function that will grow beyond all limits when we approach the boundary of the allowed region.

## Global optimization

The objective function may have many local minima. A minimum can be very narrow. Therefore, no method can guarantee that the global minimum will be found.

Take sufficiently many initial values and proceed by local optimization. One of the local minima may be the gloabal minimum.

Particularly in laborious combinatorial problems genetic algorithms and simulated annealing may be useful. They depend on the specific problem, and therefore we can here give only some general principles.

## Genetic algorithms

Mimic evolutionary process.

- Create a set of solution vectors, the initial population.
- Interbreed elements of the population and generate random mutations.
- The best solutions are taken as the new population.
- Repeat this process until the result will not improve any further.

Due to the mutations the solution will not get stuck in the first local minimum.

## Simulated annealing

Metropolis (1953).

Analogy: when a liquid is cooled sufficiently slowly, it will freeze to a regular crystal lattice having minimum energy.

The objective function corresponds to the energy of the system. The configuration is modified randomly. A temperature dependent probability is used to choose a new configuration.

When the temperature is high, even transitions to worse (higher energy) states are accepted with a moderate probability. This tends to prevent getting stuck in local minima.

When the temperature decreases, the probability for accepting a worse configuration will also decrease.

Suitable applications: e.g. combinatorial problems with no known fast solution methods, like the travelling salesman problem.

The method can usually find a reasonably good solution, but cannot guarantee its optimality.