An optimization problem consists of finding extrema of a function. This occurs frequently in engineering problems, for instance when the place of maximum deflection of a structure is to be determined, the position of a point with minimum error is looked for in a surveying adjustment, or to find the highest contamination of samples in a water quality test.

Different procedures are invented to solve such problems. These mostly involve minimization, therefore when the type of the extremum is maximum, the idea is to find minimum of the negative of the function, i.e. \( \max f(x) = \min (-f(x)) \). Extrema are always considered in a given interval or domain. Therein local minima may be found, i.e. values of the function are larger in an arbitrarily small local neighborhood of these minima (points \( P_i \) in figure below). When there are several local minima, the smallest one is called the global minimum (point \( P_2 \) in figure below).

Optimization may be unconditional or conditional (unconstrained or constrained). In constrained optimization problems those minima are searched that also meet an additional constraint or condition. One or several of such conditions may be given, they may be equalities or inequalities, linear or nonlinear. Different procedures are applied in different cases (e.g. Lagrange method, penalty method, Karush-Kuhn-Tucker conditions, linear programming). Due to time constraints only unconditional optimization problems will be considered. There are various solution methods for such problems as well.
OPTIMIZATION OF FUNCTIONS OF ONE VARIABLE

It is important for natural waters loaded by wastewater discharges to possibly minimize environmental impact. Therefore we need to know how contaminants are spread and concentrated. Let us consider minimal dissolved oxygen concentration in a river discharged by treated wastewater. There is a minimum standard for water ecosystems safety.

In this figure concentration is shown as a function of pollutant residence time. Concentration \( c(t) \) of dissolved oxygen [mg/L] as function of time is:

\[
c(t) = c_s - \frac{k_d L_0}{k_d + k_s - k_a} \left( e^{-k_a t} - e^{-k_d t} \right) - \frac{S_b}{k_a} \left( 1 - e^{-k_a t} \right)
\]

where \( t \) denotes residence time [day], \( c_s \) denotes saturation concentration (in this case: 10 mg/L), \( L_0 \) is biochemical oxygen demand (BOD) at inflow (50 mg/L), \( k_d \) denotes degradation rate [0.1 1/day], \( k_s \) is deposition rate [0.05 1/day], \( k_a \) denotes airing rate [0.6 1/day] and \( S_b \) denotes rate of oxygen demand for deposition [1 mg/L/day].

Let us determine minimal oxygen concentration near wastewater discharge. First make a plot of the function.

```matlab
% wastewater discharge
% clear all; clc; close all;
> cs = 10; L0 = 50; kd = 0.1; ks = 0.05; ka = 0.6; Sb = 1;
> c = @(t) cs - kd*L0/(kd+ks-ka)*(exp(-ka*t)-exp(-(kd+ks)*t)) - Sb/ka*(1-exp(-ka*t))
```

Function of the concentration was saved into the file concentration.mat, so we can load the function from that file to avoid typos:

```matlab
> load concentration;
> c
> % c = @(t)cs-kd*L0/(kd+ks-ka)*(exp(-ka*t)-exp(-(kd+ks)*t)) - Sb/ka*(1-exp(-ka*t))
```

Let us plot concentration in the range of 0-25 days:

```matlab
> figure(1)
> ezplot(c,[0 25])
```
We will consider two ways to find minimum of a function of one variable.

TERNARY SEARCH

Ternary search resembles to closed interval methods we learned for solution of nonlinear equations. Similarly to start, an interval $[a, b]$ must be specified that contains not a zero but a minimum of the function. In this interval the function is unimodal: monotonically decreases until it reaches the minimum and after that monotonically increases. The starting interval then should somehow be shrunk, similar to closed interval methods, till we find the solution. To proceed, let us choose two internal points $(x_1, x_2)$ and evaluate our function there.

Due to monotonicity of the function the place of the minimum must be in the interval between the point with the least function value and of its two neighbors. Hence if $f(x_1) < f(x_2)$, the minimum must be in the interval $[a, x_2]$, or if $f(x_1) > f(x_2)$ it must be in the interval $[x_1, b]$. See figure. Next we choose again two points in the new interval and repeat this process until interval length drops below a specified threshold.
Convergence is assured when the function is unimodal inside the given interval. The problem is how to choose \( x_1 \) and \( x_2 \) for a minimum number of iterations. One option is ternary search algorithm, i.e. uniform placement at 1/3 and 2/3 of the interval length. Matlab implementation of this algorithm is in file ternary.m.

\[
\text{function } [x, i] = \text{ternary}(f, a, b, \text{tol})
\]

\[
i = 1;
\]

\[
x_1 = a + \frac{1}{3}(b-a);
\]

\[
x_2 = b - \frac{1}{3}(b-a);
\]

\[
\text{while } \text{abs}(x_2-x_1) > \text{tol}
\]

\[
\text{if } f(x_1) < f(x_2)
\]

\[
b = x_2;
\]

\[
\text{else}
\]

\[
a = x_1;
\]

\[
i = i+1;
\]

\[
x_1 = a + \frac{1}{3}(b-a);
\]

\[
x_2 = b - \frac{1}{3}(b-a);
\]

\[
\text{end}
\]

\[
x = (x_1+x_2)/2;
\]

\[
\text{end}
\]

Let us find minimum oxygen concentration in river by this method. The initial unimodal interval is specified from figure as \([0, 5]\).

\[
\% \text{ ternary search - uniform placement of points}
\]

\[
[x1, i1] = \text{ternary}(c,0,5,1e-6)
\]

\[
\% x1 = 3.3912; i1 = 37
\]

\[
c1 = c(x1) \% 3.3226
\]

We see that altogether 37 iterations were required to find the minimum. Minimal concentration of oxygen was found to be 3.32 mg/L.

**GOLDEN-SECTION SEARCH**

Ternary search with uniform point placement, however, can be improved. Let us use golden ratio. This ratio is ubiquitous in nature and arts. By using golden ratio an interval \( L \) can be divided \((L = L_1 + L_2)\) such that the ratio of the longer section to the whole is the same as the ratio of the shorter section to the longer one.

\[
R = \frac{L_2}{L} = \frac{L_1}{L_2}
\]
Solve for L1 and L2 as a function of R and L: \( L2 = R \cdot L \); \( L1 = L2 \cdot R = L \cdot R^2 \). Substitute into the equation \( L = L1 + L2 \):

\[
L = L \cdot R^2 + R \cdot L
\]

Dividing by L and after rearranging we get:

\[
R^2 + R - 1 = 0.
\]

The required ratio is the only one positive root of this quadratic equation. This yields the golden ratio, where the ratio of the longer section to the whole is the same as the ratio of the shorter section to the longer one:

\[
R = \frac{\sqrt{5} - 1}{2} \approx 0.618
\]

Let us use this ratio for improved minimum search by modifying selection of the internal points within our interval.

Place internal points symmetrically such both have distances \( 0.618 \cdot L \) from the interval endpoints. Shrink our interval depending on function values. Place of the minimum must as well be between the two neighbors of the point with the least function value. But now we have a difference. Thanks to the special property of golden ratio one internal point of the new interval will be the same as one of the internal points of the old interval. We see on the figure that point \( x_2 \) of the new interval coincides with point \( x_1 \) of the old interval. Hence it is not necessary to evaluate function value at this point again, only this is necessary for the other point. For complicated functions this can be a considerable speedup. Let us look at the Matlab implementation of this procedure (golden.m)
function [x, i] = golden(f, a, b, tol)
    i = 1;
    R = (sqrt(5)-1)/2;
    x1 = b - R*(b-a);
    x2 = a + R*(b-a);
    f1 = f(x1); f2 = f(x2);
    while abs(x2-x1)>tol
        if f1 < f2
            b = x2;
            x2 = x1; f2 = f1; % taken from previous iteration
            x1 = b - R*(b-a);
            f1 = f(x1); % this must be evaluated
        else
            a = x1;
            x1 = x2; f1 = f2; % taken from previous iteration
            x2 = a + R*(b-a);
            f2 = f(x2); % this must be evaluated
        end
        i = i+1;
    end
    x = (x1+x2)/2;

During the first iteration our function is evaluated at both points \( x_1, x_2 \), but thereafter only one function evaluation is necessary, one of the old values may always be reused.

(Remark: Golden section search may also be implemented by a recursive algorithm)

Let us find and plot minimum oxygen concentration also by this method.

% golden-section search
[x2 i2] = golden(c,0,5,1e-6)
% x2 = 3.3912; i2 = 31
% x = c(x2) % 3.3226
hold on;
pplot(x2, c(x2), 'r*')

A first improvement is that the number of iterations reduced from 37 to 31. A second, even greater improvement is in the number of function evaluations. In ternary search there are 2 function evaluations per iteration, hence 37*2=74 function evaluations were needed. In golden section search only in the first iteration there were required 2 evaluations, thereafter only one per iteration, giving a total of 32 function evaluations. This is a great advantage over ternary search for complicated functions.

NEWTON'S METHOD

When computing derivative of the function poses no problem, optimization is straightforward by finding zeros of the first derivative. We have already seen an example of this when considered solving nonlinear equations in the problem of finding maximum deflection of a beam.
Let us use Newton's method for optimization. Instead of solving equation \( f(x) = 0 \) for the roots we will solve equation \( f'(x) = 0 \).

Iteration formula of Newton's method for root finding: \( x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \)

The same formula is good for optimization when we replace \( f(x) \) by \( f'(x) \):

\[
x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)}
\]

Both first and second derivatives of the function are required for optimization by Newton's method. Let us solve our problem by Newton's method. Specify one endpoint of the interval as initial value for comparison, e.g. 5. (Of course a better initial value, e.g. 4 could be selected from the figure). The `matlabFunction` procedure is used for converting symbolic derivatives into functions.

```matlab
>> % Newton's method
>> syms t
>> df = diff(c(t),t) % df = (5*exp(-(3*t)/20))/3 - (23*exp(-(3*t)/5))/3
>> ddf = diff(c(t),t,2) % ddf = (23*exp(-(3*t)/5))/5 - exp(-(3*t)/20)/4
>> % Convert symbolic expressions into functions
>> df = matlabFunction(df)
>> ddf = matlabFunction(ddf)
>> % solution by Newton's method
>> [cn in] = newton(df, ddf, 5, 1e-6, 100) % cn = 3.3912; in = 6
```

Now only 6 iterations were needed for the solution. This shows that Newton's method in case of convergence converges much more quickly. Try to change the initial value, give first the other interval endpoint 0, then try a better approximation 4 and finally use a more distant value, e.g. 10. Check the results.

### USING MATLAB’S BUILT-IN FUNCTION

Of course there is a built-in function of Matlab for optimization, e.g. `fminsearch`. This function uses the Nelder-Mead simplex method.

```matlab
>> % built-in Matlab function - fminsearch
>> cmin = fminsearch(c,5) % 3.3912
```

With detailed output:

```matlab
>> [x,fval,exitflag,output] = fminsearch(c,5)
>> i = output.iterations % i = 17
>> % Algorithm: Nelder-Mead simplex direct search
```

### FINDING MAXIMA

We have used ternary and golden-section search, which are good only for minimization. With Newton's method points with zero slope of the function are found, that is good for both minimization and maximization. Built-in function `fminsearch`, 1 for homework

as its name indicates, is good for minimization. A slight modification of ternary/golden-section search algorithm would make them capable of finding maxima; however, it is easier to find the minimum of the negative of the function. Consider an example. Let us find the maximum of the function \( f(x) = -x^2 + 6x + 1 \).

```matlab
> clear all; clc; close all;
> f = @(x) -x^2 + 6*x + 1
> ezplot(f)
> fm = @(x) -f(x)
> xm = fminsearch(fm,4) % 3.0000
> fmax = f(xm) % 10.0000
> hold on; plot(xm, fmax, 'ro')
```

Please keep in mind that for finding the value of maximum make substitution into the original function.

## Optimization of a Function in Several Variables

Optimization is frequently required of a function of not only one but of several variables. Such are problems of finding extremal points of a surface, finding maximum deflections of a 3D truss in x,y directions, optimal placing of crossings of a traffic network by minimizing ranges, etc. We have several methods for solving unconstrained multivariate optimization problems as well. For example Newton method in several variables, gradient method, Nelder-Mead simplex methods can be used.

## Positioning in Multivariate Case

Let us now consider an optimization problem in two variables. Revisit our former problem of mobile phone positioning by intersection using distances. We had to solve a system of two nonlinear equations for the two unknowns, but in practice we frequently have additional measurements. For 3 or more distances, when we have small measurement errors, there are misfits in positioning and we need adjustments. Very similar to overdetermined linear systems here also residuals (square sum of residuals) are minimized by least squares. This problem can be solved by linearization as well as by multivariate optimization algorithms. Now there are distance measurements for 4 mobile masts and our task is to get our most probable location.

<table>
<thead>
<tr>
<th>Mobile mast number</th>
<th>Coordinate X (x_i) [m]</th>
<th>Coordinate Y (y_i) [m]</th>
<th>Mast-terminal distance (r_i) [m]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>561</td>
<td>1487</td>
<td>2130</td>
</tr>
<tr>
<td>2</td>
<td>5203</td>
<td>4625</td>
<td>5620</td>
</tr>
<tr>
<td>3</td>
<td>5067</td>
<td>-5728</td>
<td>6040</td>
</tr>
<tr>
<td>4</td>
<td>1012</td>
<td>5451</td>
<td>5820</td>
</tr>
</tbody>
</table>
Locii of measured distances are circles with the following implicit equation:

$$(x-x_i)^2 + (y-y_i)^2 - r_i^2 = 0,$$

where $x$, $y$ denote coordinates of mobile masts, $x, y$ are that of the unknown position.

Plot circles in Matlab and check their intersections. Place measurements into vectors and plot mobile mast positions.

```matlab
> clear all; clc; close all;
> xt = [561; 5203; 5067; 1012]
> yt = [487; 4625; -5728; 5451]
> rm = [2130; 5620; 6040; 5820]
> % centers of circles
> figure(1); hold on;
> plot(xt, yt, 'r*')
> next define a generic function of a circle and by using symbolic variables $x,y$ plot all circles.

```matlab
> % Generic function of a circle
> eq = @(x,y,xt,yt,rm) (x-xt).^2 + (y-yt).^2 - rm.^2
> % plot circles
> syms x y
> E = eq(x,y,xt,yt,rm)
> for i = 1:4
>      ezplot(E(i),[-5000 12000])
> end
> axis equal
> title('Positioning by adjustment')
> % Area of intersection
> figure(2); hold on;
> for i = 1:4
>      ezplot(E(i),[2000 3000 -500 100])
> end
> axis equal
> title('Positioning by adjustment')
```

We realize that these four circles do not have a common intersection point but embrace an area where our assumed position is. This most probable position can be computed by minimizing sum of squared residuals. The function $f$ to be minimized is the sum of squared residuals

$$f(x, y) = \sum_{i=1}^{n} (x-x_i)^2 + (y-y_i)^2 - r_i^2$$
Let us define our cost function and make contour and 3D mesh plots of it in Matlab.

```matlab
> % cost function to be minimized
> f = sum((eq(x,y,xt,yt,rm)).^2)
> F = matlabFunction(f) % convert symbolic f into Matlab function
> ezcontour(f,[2000 3000 -500 100]);
> figure(3)
> ezsurf(f, [2000 3000 -500 100])
```

The minimum of this function is our most probable position. How to find it? Let us use Newton's method in several variables.

NEWTON'S METHOD IN SEVERAL VARIABLES

Let is recall the formula of Newton's iteration in one variable for optimization:

\[ x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)} \]

Generalization of this formula for several variables is possible by replacing first derivative with gradient vector (\( \nabla f \)) and second derivative with Hessian matrix (\( H \)). Variables must be specified as a vector (\( x \)). Our formula is

\[ x_{i+1} = x_i - H^{-1}(x_i) \cdot \nabla f(x_i) \]

where components of the Hessian are second partial derivatives of \( f(x) \), components of the gradient vector are first partial derivatives. In case of a function of two variables \( f(x,y) \), gradient vector and the Hessian are:

\[
\nabla f(x,y) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}; H(x,y) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} 
\]

We have already used the Jacobian matrix of first partial derivatives of equations/functions stored as vectors for solving systems of nonlinear equations. The Hessian can also be computed as the Jacobian matrix of the gradient vector of a function.
MULTIVARIATE NEWTON’S METHOD IN MATLAB

```matlab
function [x i X] = gradmulti(grad, hesse, x0, eps, nmax)
  x1 = x0 - pinv(hesse(x0))*grad(x0);
  i = 1;
  X = [x0 x1];
  while and(norm(x1 - x0) > eps, i < nmax)
    x0 = x1;
    x1 = x0 - pinv(hesse(x0))*grad(x0);
    i = i + 1;
    X = [X x1];
  end
  x = x1;
```

This function (gradmulti.m) implements multivariate Newton's method in Matlab. Its input arguments are the gradient vector, the Hessian, an initial guess \( x_0 \), tolerance \( \varepsilon \) for stopping iteration and maximum iteration number \( n_{max} \). Outputs of the function are the solution \( x_1 \), iteration number \( i \) and iteration steps are stored in matrix \( X \).

POSITIONING BY MULTIVARIATE NEWTON’S METHOD

Recall that both gradient vector and the Hessian are required in multivariate case instead of first and second derivatives that are used in univariate optimization. Let us compute them with Matlab. Matlab function `gradient` is suitable for both numeric and symbolic computation of the gradient vector. Let us plot numerically computed gradient vectors for better visualization. First we use `meshgrid` to create a grid and for grid points we then plot computed gradient vectors with the `quiver` function.

```matlab
% Visualization of numerically computed gradient vectors
[X,Y] = meshgrid(2000:100:3000, -500:50:100);
Z = F(X,Y);
[px,py] = gradient(Z); % numerical computation of gradient vectors
figure(2)
quiver(X,Y,px,py)
```

Optimization by multivariate Newton's method requires gradient vectors and the Hessian not numerically but given as functions of vector variables. These functions can be determined symbolically with the commands `gradient` and `hessian` applied for the symbolic function \( f \).

```matlab
% gradient vector symbolically
G = gradient(f)
% Hessian computed symbolically
H = hessian(f)
% Rewrithe H and G as functions of a vector variable
G = matlabFunction(G) % convert symbolic expression of G into a function
H = matlabFunction(H) % convert symbolic expression of H into a function
G = @(x) G(x(1),x(2)) % rewrite function with vectorial argument
H = @(x) H(x(1),x(2)) % rewrite function with vectorial argument
```
Find initial value from the figure and call gradmulti.m.

```Matlab
> x0 = [2400; -300]
> [p i pp] = gradmulti(G,H,x0,1e-6,100)
> plot(p(1),p(2),'r*')
```

Our computation converged to the minimum within specified tolerance very quickly in only 4 iterations.

**USING MATLAB'S BUILT-IN FMINSEARCH FUNCTION**

There are built-in Matlab functions for multivariate optimization, namely `fminsearch`, which uses Nelder-Mead simplex method and `fminunc`, which uses quasi-Newton minimization. Simplex method is preferred in the case when it is hard to compute derivatives of the function. This method works by starting from an initial polyhedron (simplex), which is a triangle in 2 dimensions. Then 3 vertices of this simplex is changed (by stretching, shrinking, mirroring) to follow the shape of the function and eventually it shrinks to the minimum point. To grasp the idea the following animation can help: [https://en.wikipedia.org/wiki/File:Nelder-Mead_Himmelblau.gif](https://en.wikipedia.org/wiki/File:Nelder-Mead_Himmelblau.gif)

Let us solve this problem by simplex method. We must rewrite function $F$ in terms of a vector variable.

```Matlab
> F = @(x) F(x(1),x(2)) % rewrite function with vectorial argument
> sol = fminsearch(F,x0)
> plot(sol(1),sol(2),'ks')
```

Let us check residuals of measured distances and distances between mobile masts and adjusted position.

```Matlab
> ex = xt - sol(1);
> ey = yt - sol(2);
> er = rm - sqrt(ex.^2+ey.^2)
> residual
> % 99.0847
> % 26.3188
> % -31.7595
> % -57.7440
```

Plot solution in 3D:

```Matlab
> figure(3); hold on;
> plot3(sol(1),sol(2),F(sol),'r*')
```

**Remark:** We considered local optimization algorithms that are designed to find local optimum near a given initial guess. To find global minimum several local minima must be searched and finally one with the smallest value have to be chosen. There are direct methods for global optimization over a specified domain (e.g. genetic algorithms) as well, but we do not have the time to deal with them.