Metaheuristic algorithms.

Lab 1: Optimization problems
Introduction into Scilab
_____

1. Optimization problems

Optimization problems represent one of the most frequently encountered classes of problems arising in applications which require:
- *Planning* (scheduling and/or establishing the execution order of some activities, resource allocation, timetabling etc)
- *Modelling* (constructing a model which fits well some experimental data)
- *Adaptation* (changing the behavior of a system by training based on some training data)

All optimization problems are characterized by: *solution search space*, *optimization criterion* (*objective function*) and *constraints*.

1.1. ***Solution search space.*** There are two main cases:
- Discrete search space (usually a finite but of large size set).
- Continuous search space

Depending on the search space there are two main classes of optimization problems:
- Combinatorial optimization problems
  - Assignment/allocation/selection problems (e.g. knapsack problem, bin packing problem)
  - Routing problems (e.g. travelling salesman problem)
- Continuous optimization problems
  - Parameter estimation (e.g. neural networks training, searching cluster prototypes in data clustering)

1.2. ***Optimization criterion (objective function).*** It represents the value which should be optimized (minimized or maximized). Depending on the number of optimization criteria there are:
- Single-objective optimization
- Multi-objective optimization

Depending on the available information on the objective function, the optimization problems can be:
- *White-box:* the objective function is explicitly defined (e.g. known analytical expression) and its properties can be analyzed and *exploited* (linear/non-linear, continuous/discontinuous, smoothness etc).
- *Black-box:* the objective function is not known explicitly – it can be only evaluated (e.g. in the case of objective functions which are based on simulations).

1.3 ***Constraints.*** The constraints define the space of feasible solutions and can be of several types:
- *Bound constraints*: $a <= x <= b$
- *Equality constraints*: $g(x) = 0$
- *Inequality constraints*: $h(x) >= 0$
where g and h are (nonlinear) functions

Depending on their importance, the constraints can be:

- *hard*: they have to be satisfied
- *soft*: it is preferable that they are satisfied but they are not mandatory (in such a case the constraints can be transformed in optimization criteria).

2. Steps in solving optimization problems

Step 1: problem analysis and identification of the following elements:

- *Structure* (encoding) of candidate solutions; this allows to establish the properties and the size of the search space. The problem can be of one of the following classes:
    - o Combinatorial optimization
    - o Continuous optimization

    of size
    - o small (less than 10 variabile)
    - o medium (between 10 and 100 variables)
    - o large (more than 100 de variables)

- *Constraints:* establish which constraints are hard and which are soft; transform the constraints such that they are in a canonical form (e.g. the inequalities are all >= or all <=)
- *Objective function* and its properties:
    - o Linear, quadratic, arbitrary (black box)?
    - o continuous/discontinuous, differentiable/non-differentiable?
    - o one objective function/ several objective functions, unimodal/multimodal?
    - o noisy or dynamic?

Step 2: Method selection:

It depends on the properties of the objective function and of the constraints.

- Methods for objective functions/constraints which are linear/ quadratic (e.g. techniques from linear or quadratic programming)
- Derivatives-based local optimization (e.g. variants of the gradient method, variants of the Newton method)
- Derivatives-free local optimization (e.g. Nelder Mead)

and of the aim of the optimization
- Find one global optimum (global optimization)
- Find all global/local optima (multi-modal optimization)
- Find a set of reciprocally non-dominated solutions (multi-objective optimization)
- Follow a moving optimum (dynamic optimization)

3.   **Optimization in SciLab**

In SciLab there are several functions for different optimization problems (see the following table [M. Baudin, V. Couvert, Optimization with SciLab, 2011])

Objective	Bounds	Equality	Inequalities	Size	Gradient Needed	Solver
Linear	yes	linear	linear	medium	-	linpro
Quadradic	yes	linear	linear	medium	-	quapro
Quadratic	yes	linear	linear	large	-	qpsolve
Quadratic	yes	linear	linear	medium	-	qld
Non-Linear	yes			large	yes	optim
Non-Linear				small	no	fminsearch
Non-Linear	yes			small	no	neldermead
Non-Linear	yes			small	no	optim_ga
Non-Linear				small	no	optim_sa
N.Li.Lea.Sq.				large	optional	lsqrsolve
N.Li.Lea.Sq.				large	optional	leastsq
Min-Max	yes			medium	yes	optim/nd
Multi-Obj	yes			small	no	optim_moga
Semi-Def.			lin. (spectral)	large	no	semidef
L.M.I.		lin. (spectral)	lin. (spectral)	large	no	lmisolve

3.1. Optimization using optim

Types of problems which can be solved: unconstrained nonlinear optimization problems.

Methods: quasi-Newton (local methods which require an initial approximation and the computation of the gradient)

Calls:
  [fopt,xopt]=optim(objfct, x0)   // x0 = initial approximation
  [fopt,xopt]=optim(objfct,"b",liminf,limsup, x0)   // liminf, limsup = bounding vectors

  [fopt,xopt]=optim(objfct,x0, alg)   //alg=algorihtm
              //"qn": quasi-Newton (default) based on BFGS (Broyden-Fletcher-Goldfarb-Shanno)
              //"gc": BFGS with limited memory (for a large number of variables)
              //"nd": for non-differentiable objective functions

Remark: the SciLab function which implements the objective function should return both the value of the objective function and the corresponding value of the gradient.

Example: Find an optimum of the function, f:RxR->R, given by

$f(x1,x2)=100*(x2-x1^2)^2 +(1-x1)^2$. (This function is known as Rosenbrock function and is considered difficult to be optimized in the case of a large number of variables)

It is supposed that the optimum is in the neighborhood of (-1,1.5).

Scilab implementation:
```
function [f, g, ind]=rosenbrock(x, ind)
  f = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2          // objective function
  g(1) = - 400*(x(2)-x(1)^2)*x(1) - 2*(1-x(1)) // gradient
                                               //components
  g(2) = 200*(x(2)-x(1)^2)
endfunction
x0 = [-1.0 1.5];     // initial approximation
[ fopt , xopt ] = optim (rosenbrock , x0)
```
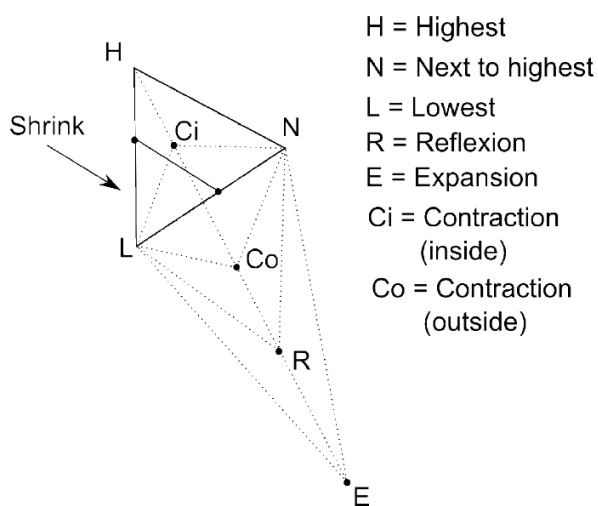
Exercise 1.

a) Estimate the optimum of the Rosenbrock function using various methods ("qn", "gc", "nd") and compare the results
b) Analyze the case when the derivatives are not computed analytically but numerically: g=numderivative(rosenbrockS,x). In the call of the function numderivative, the parameter rosenbrockS denotes a function which returns only the value (it does not provide the gradient) - see file rosenbrock.sci in the folder \examples
c) Change the function in order to be used for more than two variables.

3.2. Optimization using fminsearch

Types of problems which can be solved: unconstrained nonlinear optimization problems for non-differentiable objective functions.

Methods: Nelder-Mead (the search is based on the construction of new solutions using a simplex structure and a set of transformations – see the following figure and Lecture 2)

H = Highest
N = Next to highest
L = Lowest
R = Reflexion
E = Expansion
Ci = Contraction (inside)
Co = Contraction (outside)

Call:
   [xopt,fopt]=fminsearch(objfct, x0)   // x0 = initial approximation

[xopt,fopt]=fminsearch(objfct, x0, options)

The options are set using optimset: options= optimset(OptionName,OptionValue)
Types of options:
- MaxIter - maximal number of iterations (default: 200*number of variables)
- MaxFunEvals – maximal number of objective function evaluations (default: 200*number of variables)
- TolFun – tolerance on the objective function value (implicit: 0.0001)
- TolX – tolerance on the variables (implicit: 0.0001)
- PlotFcns – plot of the objective function value vs the number of iterations (default: empty – no visualization)

Example: Rosenbrock function

SciLab:
```
function f=rosenbrockS(x)
  f = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2          // objective function

endfunction
x0 = [-1.0 1.5];     // initial approximation
[ fopt , xopt ] = fminsearch ( rosenbrockS , x0)
```

Exercise 2. Implement and test the following variants:

a)   Visualization of the minimization process:
```
options = optimset ( "PlotFcns" , optimplotfval )
x0=[-1,1.5]
[ fopt , xopt ] = fminsearch ( rosenbrockF , x0, options)
```
b)   Output of the transformations applied at each iteration:
```
options = optimset ( "Display" , "iter" )
x0=[-1,1.5]
[ fopt , xopt ] = fminsearch ( rosenbrockS , x0, options)
```

c)   Test the case with a larger number of variables

3.3 Other functions/packages for optimization in Scilab

a. *Linear optimization with constraints.* Scilab contains a function (called karmakar) which implements the Karmakar algorithm (belonging to the class of interior point methods) to solve linear optimization problems with equality/non-equality constraints. In the simplest case (only equality constraints) the call [xopt,fopt]=karmakar(A,b,c) allows to solve the problem:

Min $c^T x$
Ax=b
x>=0

Remark: to solve linear problems using the simplex method one can use the package lpsolve
(http://lpsolve.sourceforge.net/5.1/Scilab.htm)

b. *ATOMS*

The Scilab packages (similar to Matlab toolboxes) can be loaded by Console -> Applications->Module Manager –>ATOMS

The optimization section contains:

- Quadratic programming questions (QuaPro)
- Direct search methods (Derivative-free)
- Metaheuristics: CMA-ES (Covariance Matrix Adaptation), PSO (Particle Swarm Optimization)

Homework.

1. Find $x=(x_1,x_2,\ldots,x_{10})$ which maximizes $F(x_1,x_2,\ldots,x_{10})= p_1*x_1+p_2*x_2+\ldots+p_{10}*x_{10}$ subject to the constraints $c_1*x_1+c_2*x_2+\ldots+c_{10}*x_{10}=C$, x_i in [0,1]

   Test data: p=[92; 57; 49; 68; 60; 43; 67; 84; 87; 72], C=165

   Compare the result with the optimal one (obtained by applying a greedy technique).

   Remarks: this is the continuous variant of the knapsack problem; the data set is from http://people.sc.fsu.edu/~jburkardt/datasets/knapsack_01/knapsack_01.html

   Hint: you can use the function karmakar fromScilab

Appendix 1: Scilab - Open source software for numerical computation
(http://www.scilab.org/)

Scilab is an interpreted programming language which offers support for computational tasks arising in linear algebra, polynomials operations, interpolation and approximation, linear and quadratic optimization, differential equations, signal processing, statistics and graphics.

In Scilab the basic object is the matrix, both the vectors and the scalars being particular cases of matrices.

Some general aspects:
- Scilab is case-sensitive
- Being an interpreter, the variables do not have to be declared but they should have a value assigned; the assignment operator is =
- The predefined constants have names with the prefix % (e.g. %pi, %i, %e, %t (true), %f (false))
- The relational operators are: == (equal), ~= or <> (unequal), <=, >=
- The logical operators are: ~ (not), & (and), | (or)
- The result of an evaluation which is not explicitly assigned to a variable is implicitly assigned to the object ans which can be used in the following command
- The commands specified on the same line should be separated by ; (this separator has also the effect of inhibiting the visualization of the last evaluation result).
- The strings are specified using double quotes (") and their concatenation can be specified by the plus operator (+)
- The line comments can be specified by //

Specifying matrices:
- *Explicitly*, by specifying all elements (the elements on the same row should be separated by , or space and the rows should be separated by ; or enter):
  A=[a11,a12,…a1n; a21,a22,…,a2n; …;am1, am2,…,amn]

- *Implicitly*, by using functions which generate matrices:
  - zeros(m,n) : matrix with m rows and n columns and elements equal to 0
  - ones(m,n): matrix with m rows and n columns and elements equal to 1
  - rand(m,n): matrix with m rows and n columns and elements randomly generated in (0,1)

Operations with matrices.
- Finding the size: size(mat) returns [nr rows, nr columns]
- Reorganizing a matrix: matrix(mat, nr rows, nr columns) returns a matrix with the specified size and elements taken row by row from the object specified as the first parameter
- Accessing the elements: mat(row index,column index).
  Remarks:
  - The indices can be individual values or ranges specified as inf:sup. The last index of a row or column can be specified by $. For instance mat($,$-1) specifies the element on the last row and the column before the last one.
  - The indices start with 1
- Changing a matrix:

- o Change an element: mat(i,j)=val
  - o Add a row: mat=[mat; el1, el2,…,eln]
  - o Add a column: mat=[mat'; el1, el2,…,elm]'   (the operator ' denotes the transpose)
  - o Remove a row: mat(i,:)=[]
  - o Remove a column: mat(:,j)=[]
- Arithmetical operations: all arithmetical operations are vectorized; in order to specify operations at the level of elements the operators should be prefixed by . (dot). For instance, A*B returns the algebraic product of matrices A and B and A.*B returns the matrix which is obtained by multiplying the corresponding elements from the matrices.

Other types of objects in Scilav:
- Structures:
  - o struct(fieldname1,value1, fieldname2,value2,…., fieldnamen,valuen)
  - o Example: date=struct('day',10,'month','October','year',2017)
  - o Element specification: StructureName.Fieldname(e.g.: date.day is 3)

- Heterogeneous lists:
  - o Simple list: list(Element1,Element2,…,Elementn)
    Remark: the elements are specified using indices
  - o Typed list: tlist(Names,Element1,Element2,…,Elementn) ;
    Example: d=tlist(['date','day','month','year'],10,'October',2017)
    Remark: the elements can be specified both by indexing and by qualification:
    d(2) is identical to d.day

Instructions
- **If statement:**

  if (condition) then
        <statements 1>
  else
        < statements 2>
  end

  Variant:

  if (condition1) then
        < statements 1>
  elseif (condition 2)
        < statements 2>
  else
        < statements 3>
  end

- **Select statement:**

  select < selector>
  case < val 1>
        < statements 1>

```
        case <val 2>
                < statements 2>
        …
        case <val n>
                < statements n>
        else
                <other statements >
        end
```

- **for statement**

```
for contor=inf:step:sup
        < statements >
end
```

Remark: If step is 1 the it can be ommitted. The value of the step can be less than 0. The iteration can be done over the elements of a vector (one row matrix):

```
for contor=vector
        < statements >
end
```

- **while statement**

```
while(conditie)
        < statements >
end
```

Functions

Scilab functions can be used to compute several results (specified by output variables in the function header):

```
function [output1, …,outputm]=functionName(input1,…,inputn)
        <function body>
endfunction
```

Graphics

Several graphics functions:
- plot - one-dimensional functions
- fplot3d and contour - surfaces
- paramfplot2d – curves given by parametric equations
- polarplot – polar coordinates representation
Example:
// one-dimensional function to be plotted
```
function y=f(x)
    y=x*x/10+sin(x)
endfunction
```

```
x=-2*%pi:0.1:2*%pi
clf
plot(x,f)

// surface
function y=f2arg(x1, x2) // function with 2 arguments
    y = x1 **2 + x2 **2;
endfunction
x1data = linspace ( -1 , 1 , 100 ); // this is equivalent with x1data = -1:0.1:1;
x2data = linspace ( -1 , 1 , 100 );
contour ( x1data , x2data , f2arg , 10) // contour plot
pause
clf                              // clean the screen
fplot3d( x1data , x2data , f2arg)      // surface plot
```