OPEN SOURCE ENGINEERING

A SCILAB PROFESSIONAL PARTNER

powered by ENGIN SOFT

MULTIOBJECTIVE OPTIMIZATION AND GENETIC ALGORITHMS

In this Scilab tutorial we discuss about the importance of multiobjective optimization and we give an overview of all possible Pareto frontiers. Moreover we show how to use the NSGA-II algorithm available in Scilab.

Level 

www.openeering.com

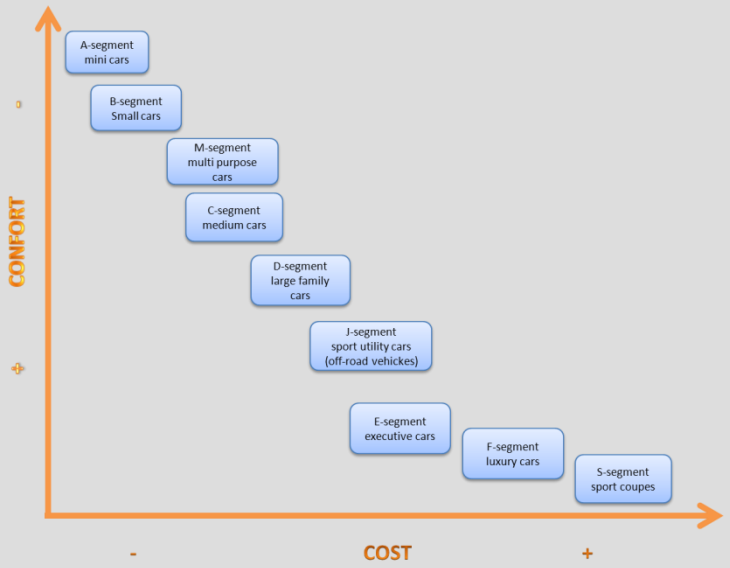
Step 1: Purpose of this tutorial

It is very uncommon to have problems composed by only a single objective when dealing with real-world industrial applications. Generally ***multiple, often conflicting, objectives*** arise naturally in most practical optimization problems.

Optimizing a problem means finding a set of ***decision variables*** which satisfies constraints and optimizes simultaneously a vector function. The elements of the vector represent the objective functions of all decision makers. This vector optimization leads to a non-unique solution of the problem.

For example, when selecting a vehicle that maximizes the comfort and minimizes the cost, not a single car, but a segment of cars may represent the final optimal selections (see figure).

After a general introduction on multiobjective optimization, the final aim of this tutorial is to introduce the reader to ***multiobjective optimization in Scilab*** and particularly to the use of the ***NSGA II*** algorithm.



(Example of car classification)

Step 2: Roadmap

In the first part of the tutorial we review some concepts on multiobjective optimization, then we show how to use NSGA-II algorithm in Scilab.

Steps 14 to 16 present some examples and exercises.

Step 17 shows how to call external (black-box) functions in Scilab.

Descriptions	Steps
Multiobjective optimization	3-5
NSGA 2	6-13
Examples and exercises	14-16
Calling external functions	17
Conclusion and remarks	18-19

Step 3: Multiobjective scenario

Here we consider, without loss of generality, the ***minimization of two objectives*** all equally important, where no additional information about the problem is available.

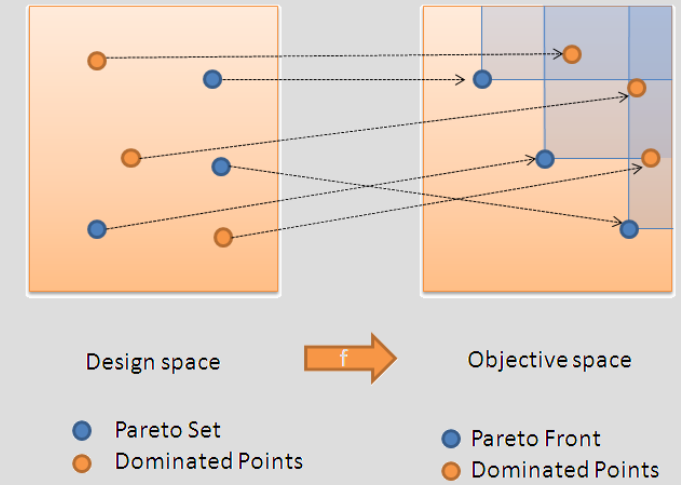
A solution of the problem can be described by a "***decision vector***" $\vec{x}$ of the form $\vec{x} = (x_1, \dots, x_n)$ lying in the design space X. The evaluation of the two ***objective functions*** on $\vec{x}$ produces a solution $\vec{y} = (y_1, y_2)$ in the objective space Y, i.e. f is a vector map of the form: $f: X \to Y$.

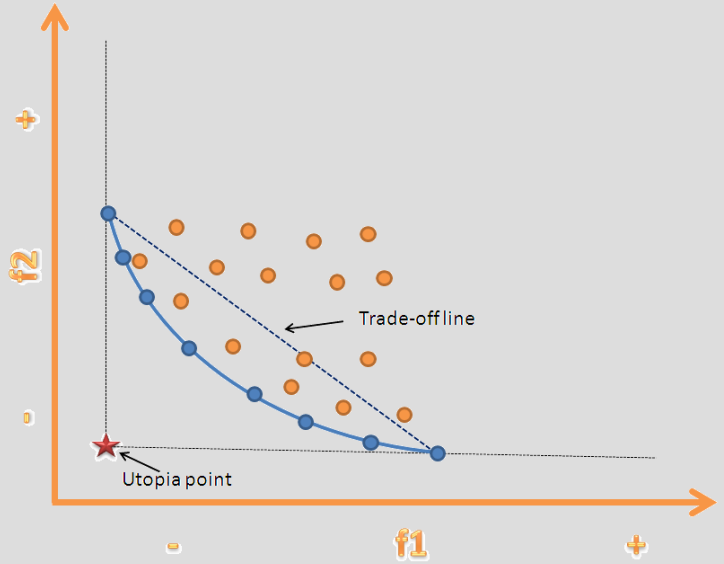
Comparing two solutions $\vec{x}_1$ and $\vec{x}_2$ requires to define a ***dominance criteria***. In modern multiobjective optimization the ***Pareto*** criteria is the most used. This criteria states:

- An objective vector $\vec{y}_1$ is said to dominate another objective vector $\vec{y}_2$ (i.e., $\vec{y}_1 < \vec{y}_2$) if no component of $\vec{y}_1$ is greater than the corresponding components of $\vec{y}_2$ and at least one component is greater;
- accordingly, the solution $\vec{x}_1$ dominates $\vec{x}_2$, if $f(\vec{x}_1)$ dominates $f(\vec{x}_1)$;
- all ***non-dominated solutions*** are the optimal solutions of the problem, solutions not dominated by any others. The set of these solutions is named Pareto set while its image in objective space is named ***Pareto front***.

A generic multiobjective optimization solver searches for non-dominated solutions that correspond to trade-offs between all the objectives.

The utopia (or ideal) point corresponds to the minimal of all the objectives and typically is not a real and feasible point.





Step 4: Type of Pareto fronts

The computation of the Pareto front can be a very difficult task. Many obstacles can make the problem complex: non-continuous design space, high-dimensionality and clustered solutions. Moreover, in a similar manner as local optimal points can trap algorithms in single objective problems, local Pareto frontiers can cause bad convergence of the multiobjective optimization approaches.

Two very typical Pareto fronts can arise when solving multiobjective optimization problems:

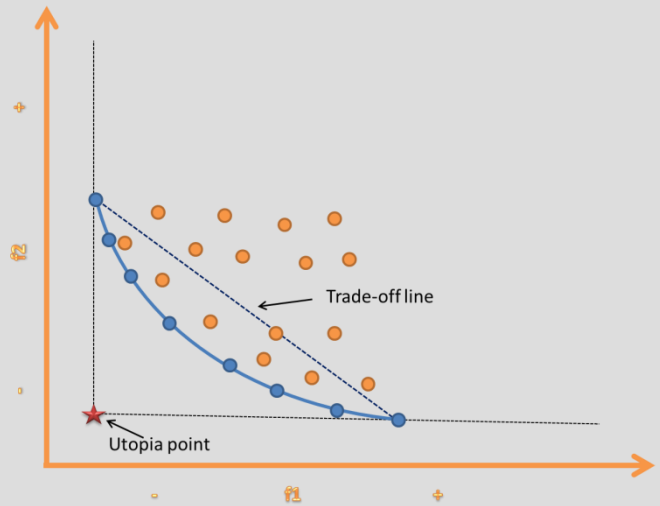
- ***Convex front***

  This is the most interest case for decision makers. When the Pareto front has this shape, the decision makers can negotiate, fighting for their own objective and they can more easily agree for a trade-off point. In this situation, the trade-off is much better than the linear combination of the original objectives. This means, practically, that if a decision maker gives up a percentage of its target, say 20%, another decision maker may have an improvement of more than 20% on his personal target.

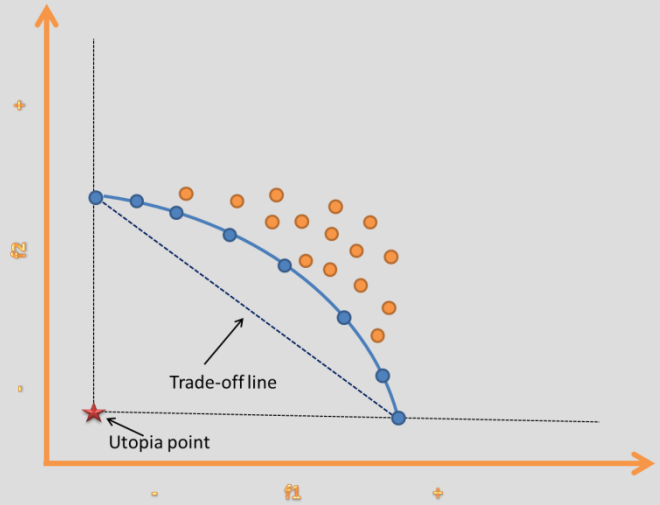
- ***Non-convex front***:

  This is the opposite of the previous situation, negotiation between decision markers is harder. Here, a decision maker should give up more than 20% of his goal to give at least 20% advantage to another decision maker. The final solution depends more on the influence of the decision maker rather than on a "democratic" negotiation.

Discontinuous fronts are common and more complex to analyze, any piece of a discontinuous front may be reduced to the two previous cases.



(Convex front)



(Non-convex front)

Step 5: Evolution algorithms

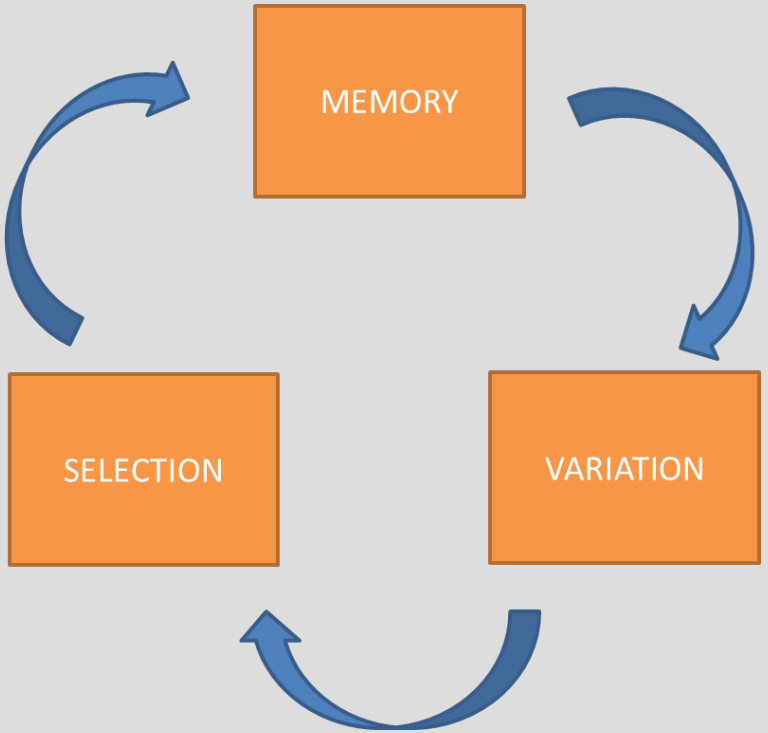
Many algorithms are based on a stochastic search approach such as evolution algorithm, simulating annealing, genetic algorithm.

The idea of these kind of algorithms is the following:

1. Define a ***memory*** that contains current solutions;
2. Define a ***selection module*** that determines which of the previously solutions should be kept in memory. Two types of selection are available:
    - ***Mating*** selection which consists of a fitness selection phase where promising solutions are picked for variation;
    - ***Environmental*** selection that determines which of stored solutions are kept into the memory.
3. Define a ***variation module*** that takes a set of solutions and systematically, or randomly, modifies these solutions to generate potentially better solutions using specific operators such as:
    - ***Crossover*** which produces new individuals combining the information of two or more parents;
    - ***Mutation*** which alters individuals with low probability of survival.

When we consider an evolution algorithm, by analogy to natural evolution, we call solutions as ***candidates*** and the set of candidates as ***population***.

The ***fitness function*** is a particular objective function that characterizes the problem measuring how close a given solution is to achieve the target, considering also all problem constraints.



Step 6: NGSA-II

NSGA-II is the second version of the famous "***Non-dominated Sorting Genetic Algorithm***" based on the work of Prof. Kalyanmoy Deb for solving ***non-convex*** and ***non-smooth*** single and multiobjective optimization problems.

Its main features are:

- A sorting non-dominated procedure where all the individual are sorted according to the level of non-domination;
- It implements elitism which stores all non-dominated solutions, and hence enhancing convergence properties;
- It adapts a suitable automatic mechanics based on the crowding distance in order to guarantee diversity and spread of solutions;
- Constraints are implemented using a modified definition of dominance without the use of penalty functions.

The Scilab function that implements the NSGA-II algorithm is:

`optim_nsga2`

which is directly available with Scilab installation.

Scilab syntax:

```
[pop_opt,fobj_pop_opt,pop_init,fobj_pop_init] =
optim_nsga2(ga_f,pop_size,nb_generation,p_mut,p_cross,Log,param)
```

Input arguments:

- ***ga_f***: the function to be optimized;
- ***pop_size***: the size of the population of individuals;
- ***nb_generation***: the number of generations to be computed;
- ***p_mut***: the mutation probability;
- ***p_cross***: the crossover probability;
- ***Log***: if %T, we will display to information message during the run of the genetic algorithm;
- ***param***: a list of parameters:
  - ***'codage_func'***: the function which will perform the coding and decoding of individuals;
  - ***'init_func'***: the function which will perform the initialization of the population;
  - ***'crossover_func'***: the function which will perform the crossover between two individuals;
  - ***'mutation_func'***: the function which will perform the mutation of one individual;
  - ***'selection_func'***: the function which will perform the selection of individuals at the end of a generation;
  - ***'nb_couples'***: the number of couples which will be selected so as to perform the crossover and mutation;
  - ***'pressure'***: the value the efficiency of the worst individual.

Output Parameters:

- ***pop_opt***: the population of optimal individuals;
- ***fobj_pop_opt***: the set of objective function values associated to pop_opt;
- ***pop_init***: the initial population of individuals;
- ***fobj_pop_init***: the set of objective function values associated to pop_init (optional).

Step 7: Problem ZDT1

The ZDT1 problem consists of solving the following multiobjective optimization problem:

$$\mathbf{min}\{f_1, f_2\}$$

where the object functions are

$$f_1 = x_1$$

$$f_2 = g \cdot \left(1 - \sqrt{\frac{f_1}{g}}\right)$$

and

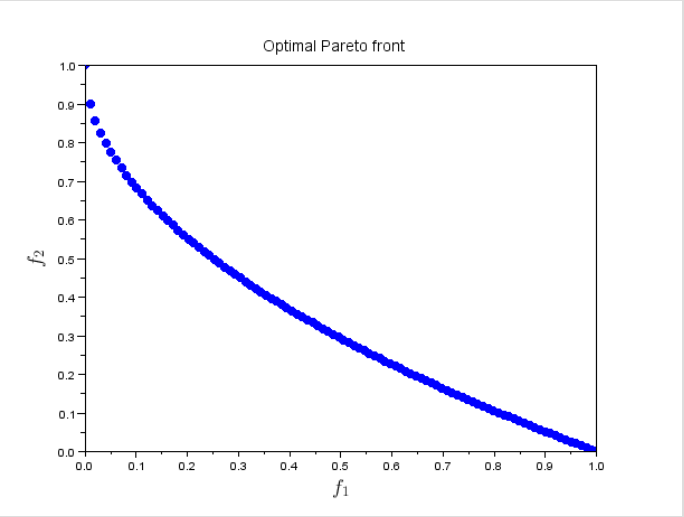
$$\mathbf{g}(\mathbf{x}_2, \ldots, \mathbf{x}_n) = 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i$$

$$0 \leq x_i \leq 1$$

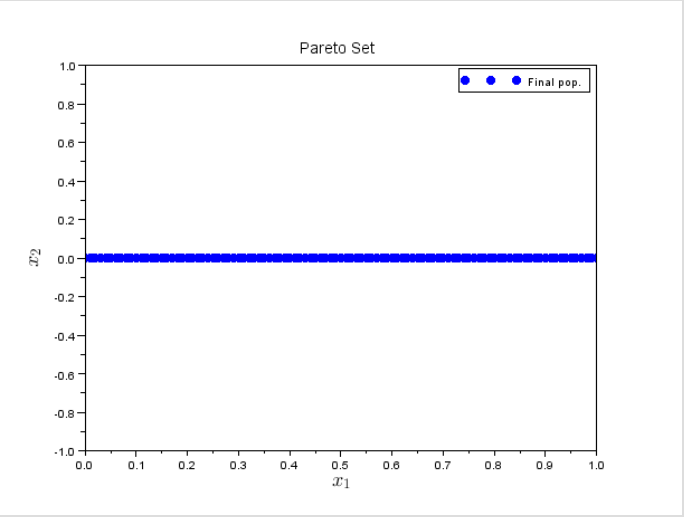
On the left we report the ***optimal Pareto front*** defined by

$$\left(f_1, 1 - \sqrt{f_1}\right)$$

This function has a ***continuous optimal Pareto front***. Moving along the frontier, from left to right, we improve the value of f_2 making the objective function f_1 worse.



(Optimal Pareto front for n=2)



(Pareto Set Points for n=2)

Step 8: Creating the ZDT1 function

In this first step, we create the "`zdt1`" function and its boundaries.

Please note that the function ZDT1 returns a ***horizontal vector of multiobjective functions evaluations***.

In the boundary functions we even define the problem dimension.

```
// ZDT1 multiobjective function
function f=zdt1(x)
f1 = x(1);
g  = 1 + 9 * sum(x(2:$)) / (length(x)-1);
h  = 1 - sqrt(f1 ./ g);
f = [f1, g.*h];
endfunction

// Min boundary function
function Res=min_bd_zdt1(n)
Res = zeros(n,1);
endfunction

// Max boundary function
function Res=max_bd_zdt1(n)
Res = ones(n,1);
endfunction
```

Step 9: Set NSGA-II parameters

In this step, we set algorithm parameters and problem dimension.

```
// Problem dimension
dim = 2;

// Example of use of the genetic algorithm
funcname    = 'zdt1';
PopSize     = 500;
Proba_cross = 0.7;
Proba_mut   = 0.1;
NbGen       = 10;
NbCouples   = 110;
Log         = %T;
pressure    = 0.1;
```

Step 10: Set NSGA-II main functions

Here, we set the NSGA-II main functions. In our case, since the problem is continuous we use the default NSGA functions.

In case of more complex mathematical optimization problem, the user can easily change the NSGA-II operators. For example, the user may define his own "mutation_func" function describing a mutation operation that perfectly fits with the problem at hand. The same can be done for "crossover_func" function or for the internal coding "codage_func".

```
// Setting parameters of optim_nsga2 function
ga_params = init_param();
// Parameters to adapt to the shape of the optimization
problem
ga_params =
add_param(ga_params,'minbound',min_bd_zdt1(dim));
ga_params =
add_param(ga_params,'maxbound',max_bd_zdt1(dim));
ga_params = add_param(ga_params,'dimension',dim);
ga_params = add_param(ga_params,'beta',0);
ga_params = add_param(ga_params,'delta',0.1);
// Parameters to fine tune the Genetic algorithm.
// All these parameters are optional for continuous
optimization.
// If you need to adapt the GA to a special problem.
ga_params =
add_param(ga_params,'init_func',init_ga_default);
ga_params =
add_param(ga_params,'crossover_func',crossover_ga_default
);
ga_params =
add_param(ga_params,'mutation_func',mutation_ga_default);
ga_params =
add_param(ga_params,'codage_func',coding_ga_identity);
ga_params = add_param(ga_params,'nb_couples',NbCouples);
ga_params = add_param(ga_params,'pressure',pressure);

// Define s function shortcut
deff('y=fobjs(x)','y = zdt1(x);');
```

Step 11: Performing optimization

The multiobjective optimization is performed using the command "optim_nsga2". Other methods are available, see for example:

- ***optim_moga***: multiobjective genetic algorithm;
- ***optim_ga***: A flexible genetic algorithm;
- ***optim_nsga***: A multiobjective Niched Sharing Genetic Algorithm.

```
// Performing optimization
printf("Performing optimization:");
[pop_opt, fobj_pop_opt, pop_init, fobj_pop_init] =
optim_nsga2(fobjs, PopSize, NbGen, Proba_mut,
Proba_cross, Log, ga_params);
```

Step 12: Plot the Pareto front

The Pareto front is obtained using the "pareto_filter" Scilab command, which automatically extracts non dominated solutions from a set of multiobjective and multidimensional solutions.

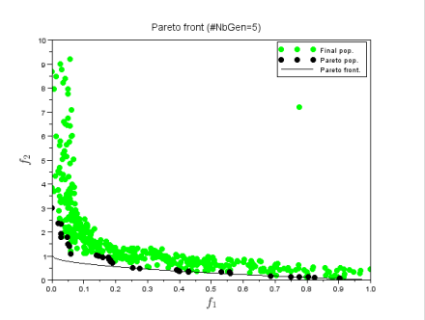
In order to plot the "population" it is necessary to convert the list "pop_pareto" to a vector using the command "list2vec".

```
// Compute Pareto front and filter
[f_pareto,pop_pareto] =
pareto_filter(fobj_pop_opt,pop_opt);

// Optimal front function definition
f1_opt = linspace(0,1);
f2_opt = 1 - sqrt(f1_opt);

// Plot solution: Pareto front
scf(1);
// Plotting final population
plot(fobj_pop_opt(:,1),fobj_pop_opt(:,2),'g.');
// Plotting Pareto population
plot(f_pareto(:,1),f_pareto(:,2),'k.');
plot(f1_opt, f2_opt, 'k-');
title("Pareto front","fontsize",3);
xlabel("$f_1$","fontsize",4);
ylabel("$f_2$","fontsize",4);
legend(['Final pop.','Pareto pop.','Pareto front.']);

// Transform list to vector for plotting Pareto set
npop = length(pop_opt);
pop_opt = matrix(list2vec(pop_opt),dim,npop)';
nfpop = length(pop_pareto);
pop_pareto = matrix(list2vec(pop_pareto),dim,nfpop)';

// Plot the Pareto set
scf(2);
// Plotting final population
plot(pop_opt(:,1),pop_opt(:,2),'g.');
// Plotting Pareto population
plot(pop_pareto(:,1),pop_pareto(:,2),'k.');
title("Pareto Set","fontsize",3);
xlabel("$x_1$","fontsize",4);
ylabel("$x_2$","fontsize",4);
legend(['Final pop.','Pareto pop.']);
```

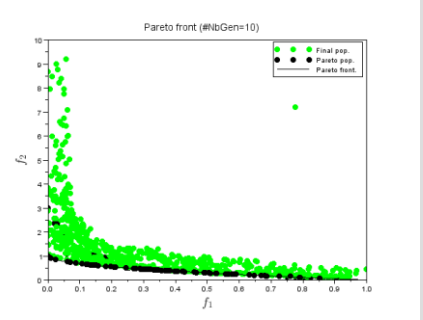
Step 13: Some results

Here, we report some results obtained running NSGA-II with the ZDT1 and changing the number of generations (parameter "`NbGen`").

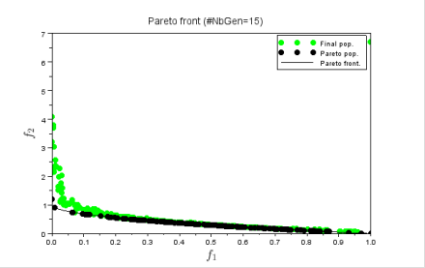
NbGen takes values from the set `(5,10,15,20)`.



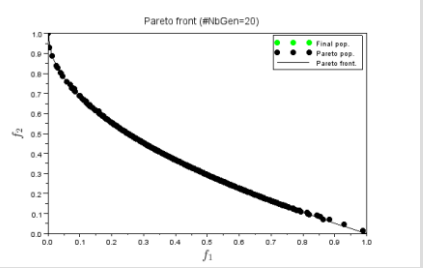
(NbGen = 5)



(NbGen = 10)



(NbGen = 15)



(NbGen = 20)

Step 14: Exercise #1: ZDT2 problem

Solve for the ZDT2 problem consists of solving the following multiobjective optimization problem:

$$\min\{f_1, f_2\}$$

where the two object functions are

$$f_1 = x_1$$

$$f_2 = g \cdot \left(1 - \left(\frac{x_1}{g}\right)^2\right)$$

and

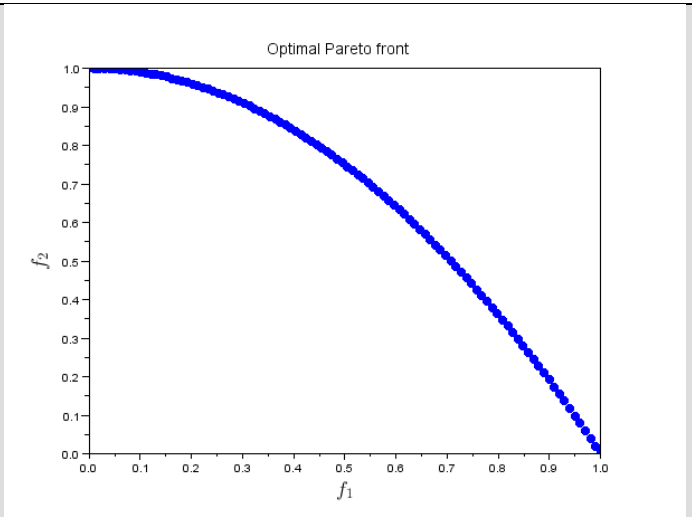
$$g(\vec{x}) = 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i$$

$$0 \le x_i \le 1$$

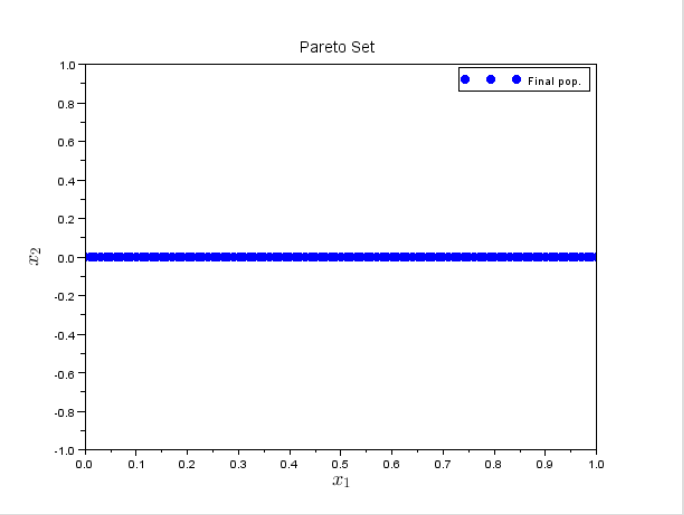
In the left we have reported the ***optimal Pareto front*** defined by

$$\left(f_1, 1 - {f_1}^2\right)$$

This function presents a ***continuous non-convex optimal Pareto front***.



(Optimal Pareto front for n=2)



(Pareto Set Points for n=2)

Step 15: Exercise #2: ZDT3 problem

Solve for the ZDT3 problem consists of solving the following multiobjective optimization problem:

$$\mathbf{min}\{f_1, f_2\}$$

where the two object functions are

$$f_1 = x_1$$

$$f_2 = g \cdot \left(1 - \sqrt{\frac{x_1}{g}}\right) - \frac{x_1}{g}\sin(10\pi x_1)$$

and

$$\mathbf{g}(\vec{x}) = 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i$$

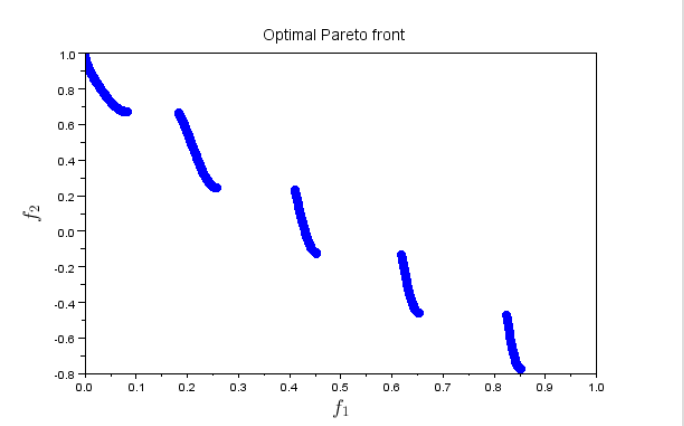
$$0 \le x_i \le 1$$

In the left we have reported the ***optimal Pareto front*** defined by

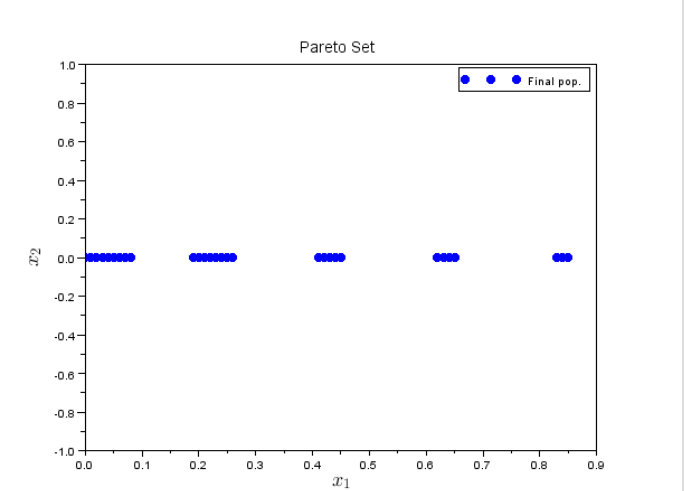
$$\left(f_1, 1 - \sqrt{f_1} - f_1 \cdot \sin(10\pi f_1)\right) \textbf{ with } f_1 \in F \textbf{ defined as}$$

$$F = [0.0, 0.0830015349] \cup (0.1822287280, 0.2577623634] \cup (0.4093136748, 0.4538821041] \cup (0.6183967944, 0.6525117038] \cup (0.8233317983, 0.8518328654]$$

This function has a ***discontinuous optimal Pareto front***.



(Optimal Pareto front)



(Pareto Set Points for n=2)

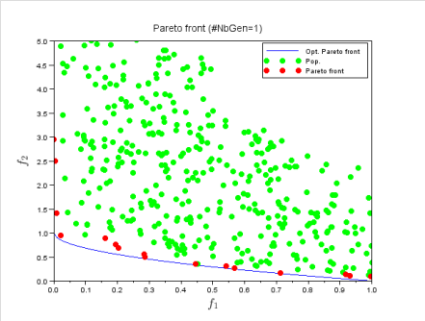
Step 16: Exercise #3

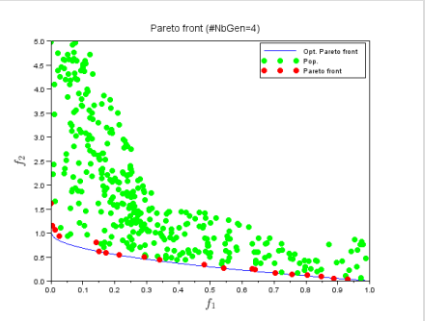
Modify the ZDT1 program in order to plot the population at each step.

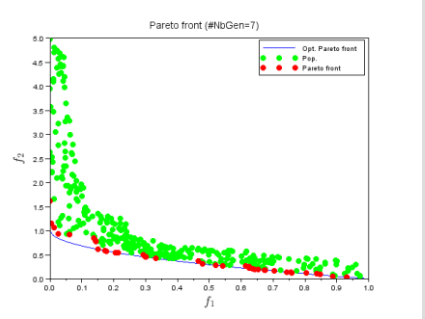
Hints:

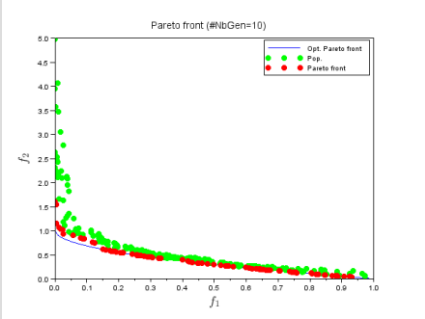
- Create a global variable named "`currPop`" where to save the population at each time step;
- Create an initial function for the optim_nsga2 named "`init_ga_previous`" that loads the computed population at each step except the initial step where the original init function must be called.
- Add plot at each time step
- If you want to produce a video or animate png save each plot using the command "`xs2png`" (for example an animated png image can be created using the program ***JapngEditor***).

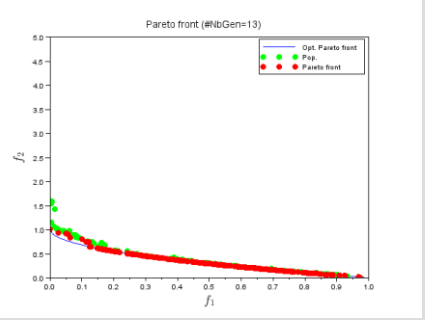
```
// Create a global variable
global currPop;

// Create a function for initialize the global variable
function Pop_init=init_ga_previous(popsize, param)
    global currPop;
    Pop_init = currPop;
endfunction

// Performing optimization
for i=1:NbGen
    // Call optim_nsga2 with a local number of generation equals
to 1
    if i>1
        // Change init generation function
    end
    // Save population
    currPop = pop_opt;
    // filtering and Plotting data
end
```

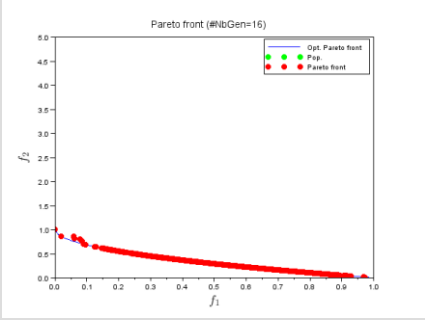












Step 17: Calling external functions

Scilab, with its Input/Output functions, enables coupling to any external functions and tools (even CAD, CAE, CFD) that can be called from external commands.

This can be done in a very easy way as shown in several examples in “Made with Scilab”:

http://www.openeering.com/made_with_scilab

Scilab can even deal with parallel computing. This represents an enormous advantage when combining optimization together with engineering simulations. This approach can speed-up time-consuming optimization problems that are very typical in industrial applications.

Moreover, if simulation time is still too demanding, Scilab can take advantage of several meta-modeling techniques such as Kriging, DACE, neural networks, etc.

If you are interested in integrating your simulation code into Scilab for solving specific optimization problem, please do not hesitate to contact the Openeering team.

The most useful commands to integrate Scilab with your external code are:

- dos — shell (cmd) command execution (Windows only);
- unix — shell (sh) command execution;
- unix_g — shell (sh) command execution, output redirected to a variable;
- unix_s — shell (sh) command execution, no output ;
- unix_w — shell (sh) command execution, output redirected to scilab window;
- unix_x — shell (sh) command execution, output redirected to a window;
- host — Unix or DOS command execution.

```
// Windows only example
[s,w] = dos('dir');
// general syntax to run my simulation code with options
in Windows systems
//[s, w] = dos('mysimulation.exe /option')

//Run list or dir according to operating systems
if getos() == 'Windows' then
  unix_w("dir "+'"'"'+WSCI+"\modules"+'"'");
else
  unix_w("ls $SCI/modules");
end
```

Step 18: Concluding remarks and References

In this Scilab tutorial we have shown how to use the NSGA-II within Scilab.

On the right-hand column you may find a list of interesting references for further studies.

1. Scilab Web Page: Available: www.scilab.org.
2. Openeering: www.openeering.com.
3. ZDT1, ZDT2 and ZDT3 documentation: http://www.tik.ee.ethz.ch/sop/download/supplementary/testproblems/
4. JapngEditor : https://www.reto-hoehener.ch/japng/

Step 19: Software content

To report bugs or suggest improvements please contact the Openeering team.

www.openeering.com.

Thank you for your attention,

Manolo Venturin and Silvia Poles

```
----------------
NSGA 2 IN SCILAB
----------------

--------------
Main directory
--------------
example_steps.sce            : Exercise 1
example_ZDT1.sce             : Example for the ZDT1 function
example_ZDT2.sce             : Example for the ZDT2 function
example_ZDT3.sce             : Example for the ZDT3 function
front plots.sce              : Plots Pareto fronts and sets
license.txt                  : The license file
```