PROSIMPLUS APPLICATION EXAMPLE

EXAMPLE OF USE OF AN EXTERNAL OPTIMIZATION ALGORITHM

EXAMPLE PURPOSE

This example illustrates the use of an external optimization algorithm in the ProSimPlus simulation environment. The communication interfaces with the external algorithm are detailed along with a short application example.

ACCESS

✔ Free Internet
☐ Restricted to ProSim clients
☐ Restricted
☐ Confidential

CORRESPONDING PROSIMPLUS FILES

PSPS_E26_EN – External optimizer example.pmp3

Reader is reminded that this use case is only an example and should not be used for other purposes. Although this example is based on actual case it may not be considered as typical nor are the data used always the most accurate available. ProSim shall have no responsibility or liability for damages arising out of or related to the use of the results of calculations based on this example.

Copyright © 2017 ProSim, Labège, France - All rights reserved
# TABLE OF CONTENTS

1. **INTRODUCTION** ....................................................................................................................................... 3

2. **OPERATING MODE OF THE PROSIMPLUS STANDARD OPTIMIZER** ................... 3
   2.1. Principle ............................................................................................................................................... 3
   2.2. Management of “tear” streams ............................................................................................................ 3
   2.3. Management of information ................................................................................................................. 4
   2.4. Parameters .......................................................................................................................................... 5

3. **COMMUNICATION INTERFACES WITH THE EXTERNAL ALGORITHM** ........... 5
   3.1. Calling interface for the solving ........................................................................................................... 5
   3.1. Calling interface for the printings ......................................................................................................... 7
   3.2. Management of objective functions ..................................................................................................... 8
   3.3. Management of constraints ................................................................................................................. 8
       3.3.1. Inequality constraints ..................................................................................................................... 8
   3.4. Intermediate printings .......................................................................................................................... 8
   3.5. Graphical User Interface ..................................................................................................................... 9
   3.6. Simulation report ............................................................................................................................... 11

4. **USE CASE: THE MIDACO SOLVER** .............................................................................. 11

**APPENDIX 1 - SOURCE CODE OF THE EXTERNAL OPTI ROUTINE** ......................... 15

**APPENDIX 2 - SOURCE CODE OF THE EXTERNAL IOPTI ROUTINE** ............................ 18
1. INTRODUCTION

The ProSimPlus simulation environment contains in its standard library two optimization algorithms, the first one is based on a SQP method (Sequential Quadratic Programming) and the second is implementing an evolutionary algorithm (genetic algorithm). Both of them are only working with continuous variables corresponding to the operating parameters of the process to be modeled. With the aim to offer more flexibility to the end-user, a new feature has been developed in version 3.5.16 of ProSimPlus; it allows to use an external optimization algorithm that can handle any type of optimization problem (multi-objective with continuous, integer or logical variables) through an external dynamic library. This document presents an example of use of this feature.

In a first step, the operating mode of the standard ProSimPlus optimizer is reminded, then the second part is dedicated to the presentation of the communication interfaces and finally an example based on the MIDACO optimization algorithm (www.midaco-solver.com) is detailed. MIDACO uses an extended “Ant Colony Optimization” evolutionary algorithm and is able to deal with multi-objective problems with mixed variables.

2. OPERATING MODE OF THE PROSIMPLUS STANDARD OPTIMIZER

2.1. Principle

The main operating principle of the ProSimPlus optimizer rely on a sequence of back to back runs (called « runs in the MCN ») between the module itself and the unit operations belonging to the optimization loop. This operating mode rely on the fact that the optimizer needs to know the values of the objective functions and the constraints at different steps of the optimization process:

- At the start of the convergence process
- When the objective functions and the constraints are being evaluated for a set of iterative variables at the current iteration
- When the derivatives of the objective functions and of the constraints are being calculated through numerical perturbations between two successive iterations

The description of the problem to be solved and the parameters associated to the SQP algorithm are detailed in the user’s manual to which one’s will refer for any additional information, especially concerning the Feasible Path and Unfeasible Path approaches as well as for the multi-periods optimization.

Thus, it’s the role of the optimization algorithm to deal with the current calculation phase and depending on the type of algorithm used, the number of calculation phases can be variable. An evolutionary algorithm, for example, does only functions evaluations for different populations and after mutation and crossing, restarts the evaluation process until it reaches the predefined maximum number of generations.

2.2. Management of “tear” streams

Without describing in full details the Unfeasible Path approach, one’s must know that the presence of “tear” streams in the optimization loop impose to the optimizer to handle the variables associated with these “tear” streams as optimization variables and to write equality constraints related to these variables, in form of:

\[ x - g(x) = 0 \]
With: \( x \) Current value of the variable
\[ g(x) \] Value of the variable after a run in the MCN.

The variables associated to a “tear” stream are: the partial molar flowrates of the components involved in the simulation and possibly the temperature and the pressure of the stream. Generally the temperature is taken into account whereas the pressure is added only if the unit operation from which the stream is exiting is automatically adjusting the pressure (for example the “Pipe segment” unit).

Furthermore, the variables associated to the “tear” streams must be the first variables of the optimization variables array (action variables) and in the following order:

- Partial molar flowrate of the component #1 in the “tear” stream #1
- Partial molar flowrate of the component #2 in the “tear” stream #1
- …
- Partial molar flowrate of the component #NC in the “tear” stream #1
- Temperature of the “tear” stream #1 (if the temperature is selected as an optimization variable)
- Pressure of the “tear” stream #1 (if the pressure is selected as an optimization variable)
- …

The other optimization variables (outlet information streams from the “Optimization” unit) come after.

The use of an external optimization algorithm does not deviate from this rule as soon as it’s the Unfeasible Path approach which is used.

The Feasible Path approach, which imposes the use of a “Constraints and recycles” unit, allows to ignore this rule, even if in this case the calculation sequence must then be manually supplied (see ProSimPlus user’s manual).

### 2.3. Management of information

The various information required by the “Optimization” unit, are transmitted thanks to information streams connected to the unit and in the following sequence:

- **Inlet information**
  1. Objective function
  2. Equality constraint #1
  3. …
  4. Equality constraint \#NE
  5. Inequality constraint #1
  6. …
  7. Inequality constraint \#NI

- **Outlet information**
  1. Action variable #1
  2. …
  3. Action variable \#NA
In the standard version of the optimizer (SQP algorithm), only one objective function is allowed, this means that the multi-objective optimization is not possible. However, an external algorithm might be able to handle several objective functions, thus the possibility to connect several objective functions has been implemented. For the standard SQP algorithm, this limitation is still in use, whereas for an external algorithm up to 10 objective functions (i.e. inlet information streams) can be connected to perform a multi-objective optimization.

### 2.4. Parameters

In order to offer the best user-friendliness in the use of an external algorithm, most of the standard parameters of the “Optimization” unit have been kept and can be used along with the external algorithm. These parameters are the following ones:

- Selection of the “tear” streams
- Selection of the parameters associated to the « tear » streams: bounds and increments for a possible sensitivity analysis
- Parameters relating to the optimization variables (action variables) : bounds, increments and types of increments for the sensitivity analysis
- General numerical parameters: maximum number of runs in the MCN, maximum number of iterations, intermediate printings step, number of rest steps, number of equality constraints, number of periods

Concerning the external algorithm-specific parameters, they are available in a « spreadsheet » like grid, allowing to adapt to the specificity of each external algorithm by offering the best flexibility of use.

### 3. COMMUNICATION INTERFACES WITH THE EXTERNAL ALGORITHM

Two communication interfaces are available, respectively for the calculation itself and for the printing of the results. They are detailed hereafter.

#### 3.1. Calling interface for the solving

The calling interface used during the solving, is presented below:

```fortran
integer(4) function OPTI(PAR,NPT,ITEMAX,NAPM,IALP,ITER,KOPT,INFO, & NCO,NCI,NVA,NF,F,C,X,XL,XU,AC,G,CN) result(rc)

!dec$ attributes stdcall, reference, alias:'OPTI', dllexport :: OPTI

integer(4), intent(in) :: NPT, ITEMAX, NAPM, IALP(NVA), NCO, NCI, NVA, NF
integer(4), intent(inout) :: ITER, KOPT, INFO
real(8), intent(inout) :: PAR(NPT)
real(8), intent(in) :: F(NF), C(NCO), XL(NVA), XU(NVA), AC(NVA)
real(8), intent(inout) :: X(NVA), G(NF,NVA), CN(NCO,NVA)
```

The calling convention used is **STDCALL** and the arguments are passed by reference. The name of the entry point is imposed and must be **OPTI** (in uppercase).

The meaning of the function arguments is given in the following table:
### Example of use of an external optimization algorithm

#### Table of Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Size</th>
<th>I/O</th>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPT</td>
<td>Integer</td>
<td>4</td>
<td>I</td>
<td>1</td>
<td>Number of user parameters</td>
</tr>
<tr>
<td>ITEMAX</td>
<td>Integer</td>
<td>4</td>
<td>I</td>
<td>1</td>
<td>Maximum number of iterations</td>
</tr>
<tr>
<td>NAPM</td>
<td>Integer</td>
<td>4</td>
<td>I</td>
<td>1</td>
<td>Maximum number of runs in the MCN</td>
</tr>
<tr>
<td>IALP</td>
<td>Integer</td>
<td>4</td>
<td>I</td>
<td>NVA</td>
<td>Types of increment of the sensitivity analysis</td>
</tr>
<tr>
<td>NCO</td>
<td>Integer</td>
<td>4</td>
<td>I</td>
<td>1</td>
<td>Number of constraints</td>
</tr>
<tr>
<td>NCI</td>
<td>Integer</td>
<td>4</td>
<td>I</td>
<td>1</td>
<td>Number of inequality constraints</td>
</tr>
<tr>
<td>NVA</td>
<td>Integer</td>
<td>4</td>
<td>I</td>
<td>1</td>
<td>Number of optimization variables</td>
</tr>
<tr>
<td>NF</td>
<td>Integer</td>
<td>4</td>
<td>I</td>
<td>1</td>
<td>Number of objective functions</td>
</tr>
<tr>
<td>ITER</td>
<td>Integer</td>
<td>4</td>
<td>I/O</td>
<td>1</td>
<td>Current iteration count</td>
</tr>
<tr>
<td>KOPT</td>
<td>Integer</td>
<td>4</td>
<td>I/O</td>
<td>1</td>
<td>Index of the calculation phase</td>
</tr>
<tr>
<td>INFO</td>
<td>Integer</td>
<td>4</td>
<td>O</td>
<td>1</td>
<td>Information relative to the calculation stop</td>
</tr>
<tr>
<td>PAR</td>
<td>Float</td>
<td>8</td>
<td>I/O</td>
<td>NPT</td>
<td>Array of parameters for the external algorithm</td>
</tr>
<tr>
<td>F</td>
<td>Float</td>
<td>8</td>
<td>I</td>
<td>NF</td>
<td>Array of the objective functions</td>
</tr>
<tr>
<td>C</td>
<td>Float</td>
<td>8</td>
<td>I/O</td>
<td>NCO</td>
<td>Array of the constraints</td>
</tr>
<tr>
<td>XL</td>
<td>Float</td>
<td>8</td>
<td>I</td>
<td>NVA</td>
<td>Array of the lower bounds of the action variables</td>
</tr>
<tr>
<td>XU</td>
<td>Float</td>
<td>8</td>
<td>I</td>
<td>NVA</td>
<td>Array of the upper bounds of the action variables</td>
</tr>
<tr>
<td>AC</td>
<td>Float</td>
<td>8</td>
<td>I</td>
<td>NVA</td>
<td>Array of the increments for the sensitivity analysis</td>
</tr>
<tr>
<td>X</td>
<td>Float</td>
<td>8</td>
<td>I/O</td>
<td>NVA</td>
<td>Array of the optimization variables</td>
</tr>
<tr>
<td>G</td>
<td>Float</td>
<td>8</td>
<td>I/O</td>
<td>(NF,NVA)</td>
<td>Work array for the storage of the derivatives of the objective functions</td>
</tr>
<tr>
<td>CN</td>
<td>Float</td>
<td>8</td>
<td>I/O</td>
<td>(NCO,NVA)</td>
<td>Work array for the storage of the derivatives of the constraints</td>
</tr>
</tbody>
</table>

Note: The $G()$ and $CN()$ arrays are provided as storage arrays for the derivatives of the objective functions and derivatives of the constraints with respect to the optimization variables and may not be used depending on the optimization algorithm used.

At each call, the function must return an integer value indicating the current state of the calculation.
The return code can take the following values:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTI_RUNNING</td>
<td>Optimization running</td>
</tr>
<tr>
<td>OPTI_CONVERGED</td>
<td>Optimization ended</td>
</tr>
<tr>
<td>OPTI_FAILED</td>
<td>Optimization failed</td>
</tr>
</tbody>
</table>

Furthermore, during the calculation, the KOPT variable is used by the OPTI function to know and manage the current calculation state. The KOPT variable is set to 0 at first call (which allows to perform some initializations or to execute specific code only once) and must be set to any non-zero value by the function. The values are let free to the developer’s needs. At the end of the optimization process, the KOPT variable must be reset to zero (i.e. when the return code has been set to OPTI_CONVERGED or OPTI_FAILED).

Then, the INFO variable is used to know the reason of the calculation stop. This variable can take the following values:

<table>
<thead>
<tr>
<th>INFO</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Optimization successful</td>
</tr>
<tr>
<td>≠ 1</td>
<td>Failure of the optimization</td>
</tr>
</tbody>
</table>

### 3.1. Calling interface for the printings

The calling interface used for the printings, is presented below:

```fortran
subroutine IOPTI(PAR,NPT)
!dec$ attributes stdcall, reference, alias:'IOPTI', dlexport :: IOPTI

  integer(4), intent(in) :: NPT
  real(8),    intent(in) :: PAR(NPT)
```

The calling convention used is STDCALL and the arguments are passed by reference. The name of the entry point is imposed and must be IOPTI (in uppercase).

The meaning of the arguments is the same as for the main calling interface (OPTI function above).

**Note:** The entry point is optional, if it is not available in the dynamic link library, it will not be called but this will not prevent the optimization process to run. In the case it is available, the printings must be made into specific file, named IMPO_OPTI.tmp and being located in the temporary folder of the current user. This file must be open in the IOPTI subroutine and closed before leaving it in order to the calling process to be able to read its content and to destroy it after. This file can be used, for example, to transfer the contents of the output file(s) generated by the external algorithm to the simulation report. If needed, the external algorithm-specific parameters are available in the arguments list but must not be modified (read only).
3.2. **Management of objective functions**

The $F()$ array contains the value(s) of the objective function(s) for the current optimization process. It contains the values transmitted to the "Optimization" unit through information stream(s) connected to the connection ports labelled "Objective function i". The $F()$ array is updated at each call with the current value(s) of the objective function(s). In the case of an optimization algorithm requiring to evaluate the derivatives of the objective function(s) with respect to the optimization variables, it will be necessary to save the contents of this array into a local array at the beginning of every new iteration (i.e. once the evaluation of the objective function(s) and the constraints has been performed).

3.3. **Management of constraints**

The $G()$ array contains the values of the constraints for the current optimization process. It contains the values of the equality constraints followed by those of the inequality constraints. In the case the "Optimization" unit handles by itself the “tear” streams, i.e. in the case of the **Unfeasible Path** approach, the equality constraints are first made up with the constraints relating to the « tear » streams followed by the constraints transmitted to the "Optimization" unit through the information streams connected to the connection ports labelled “Constraint i”.

Furthermore, the connection ports, labelled “Constraint 1” to “Constraint NE” must compulsorily be used for the NE equality constraints and the ports “Constraint NE+1” to “Constraint NE+NI” by those associated to the inequality constraints.

As for the objective function(s), the $G()$ array is updated at each call with the current values of the constraints. In the case of an optimization algorithm requiring to evaluate the derivatives of the constraints with respect to the optimization variables, it will be necessary to save the contents of this array into a local array at the beginning of every new iteration (i.e. once the evaluation of the objective function(s) and the constraints has been performed).

3.3.1. **Inequality constraints**

The inequality constraints must be written in such a way that the value of the constraint is negative when it is satisfied, which is the normal behavior used in the standard optimization algorithm (SQP). For example, if a constraint is expressed as:

$$ a \leq f_i \leq b $$

Then, the $G()$ array will contains the following expressions:

$$ G(k) = a - f_i $$

$$ G(k + 1) = f_i - b $$

In the case where the external algorithm uses the reverse way (like for the MIDACO algorithm), their sign must be changed before each call to the external algorithm solving routine.

3.4. **Intermediate printings**

During the calculation phase, it is possible to print some intermediate results into the ProSimPlus history file (extension .HIS). These printings are performed through the use of a temporary file, named **IMPI_OPTI.tmp** being located in the temporary folder of the current user. This file must be open at each call to the **OPTI** function and closed before leaving it so that the calling process can read its content and destroy it after.
If no information is required, it is not necessary to create this file. This can be the case if the external algorithm manages its own output files, nevertheless in such a case the content of these files could not be inserted neither in the simulation report nor in the history file.

3.5. **Graphical User Interface**

The following screenshots present the modifications realized in the ProSimPlus “Optimization” unit in order to make possible the use on an external optimization algorithm.

The main tab, containing the parameters has been redesigned in order to clearly split the general parameters from the parameters relating to the “tear” streams (Unfeasible Path approach) from those relating to the selected optimization algorithm.

**Selection of the optimization algorithm**

![Selection of the optimization algorithm](image-url)
Other parameters of the “Optimization” unit

Parameters relative to the selected algorithm

General parameters and action variables (bounds and increments).

Parameters relative to the “tear” streams: selection, variables, bounds and increments

Parameters relative to the external algorithm

Selection field of the external library (DLL)

Number of parameters for the external algorithm

Parameters of the external algorithm: name, value
3.6. Simulation report

The simulation report displays the generic elements of the “Optimization” unit, like the data relative to the objective function(s), to the constraints and to the optimization variables. It displays also the items relative to the standard numerical parameters of the unit: action variables and bounds, “tear” streams and bounds. Finally it displays the user-defined parameters as they are defined in the grid with their names and numerical values. If a printing subroutine is implemented the outputs made in this procedure are included at the end of the “standard” printings.

4. USE CASE: THE MIDACO SOLVER

This section details a use case based on the implementation of an external algorithm. The solver selected has been developed by M. Schluter and M. Munemoto from the Hokkaido University. It implements a general purpose algorithm for solving mono and multi-objective problems, initially developed for solving nonlinear problems with mixed variables (MINLP). In its current version, it is based on an evolutionary algorithm known as « Ant Colony Optimization », from which it takes its name: Mixed Integer Distributed Ant Colony Optimization.

An evaluation version, limited to up to 4 optimization variables is available on the following web-site www.midaco-solver.com. It is the version which has been used in this example.

The problem used as a test case is a simple optimization problem for which the analytical solution is known as well as the solution provided by the ProSimPlus standard SQP algorithm.

It is a purely theoretical problem described as follows: a process has 4 feeds which are mixed and the aim is to maximize the following function which depends on the flowrates of these feeds.

\[ f = -(f_1 + f_2 + f_3 + f_4) \]

With:

\[ f_1 = \frac{1}{(x_1 - 1.111)^2 + 0.1}, \quad f_2 = \frac{1}{(x_2 - 2.222)^2 + 0.1} \]

\[ f_3 = \frac{1}{(x_3 - 3.333)^2 + 0.1}, \quad f_4 = \frac{1}{(x_4 - 4.444)^2 + 0.1} \]

And

\[ x_i \quad \text{Total molar flowrate of the feed \#i} \]

The analytical solution is:

\[ f = -40 \]

And \[ x_1 = 1.111, \quad x_2 = 2.222 \]

\[ x_3 = 3.333, \quad x_4 = 4.444 \]

The objective function is calculated thanks to a script unit which takes the values of the feeds flowrates (inlet information streams) and the “Optimization” unit calculates them. It is a problem without constraints and the ProSimPlus flow diagram is presented hereafter:
The variables are bounded between 0.1 kmol/h and 10 kmol/h

The use of the ProSimPlus standard SQP algorithm with the default numerical parameters (except for the bounds) gives the following results:

SUCCESSIVE QUADRATIC PROGRAMMING ALGORITHM

NUMBER OF VARIABLES OF THE PROBLEM................. 4
AMONG WHICH 0 ARE TEAR STREAM VARIABLES
AND 4 ARE OPTIMIZATION VARIABLES

NUMBER OF CONSTRAINTS OF THE PROBLEM............... 0
NUMBER OF EQUALITY CONSTRAINTS.................... 0
NUMBER OF INEQUALITY CONSTRAINTS................... 0

DERIVATES OF THE FUNCTION ARE COMPUTED
- BY 1ST ORDER FINITE DIFFERENCES

NUMERICAL PARAMETERS:
MAXIMUM NUMBER OF MAJOR ITERATIONS................. 200
MAXIMUM NUMBER OF SEQUENCE CALCULATIONS.......... 1000
TOLERANCE ON KUHN-TUCKER ERROR.................... 1.000000E-04
TOLERANCE FOR NON-EVOLUTION OF THE VARIABLES.... 1.000000E-04
TOLERANCE FOR NON-EVOLUTION OF THE FUNCTION..... 1.000000E-04

RESULTS:
NUMBER OF ITERATIONS.............................. 23
NUMBER OF RUNS IN THE PROCESS.................... 130
VALUE OF THE OBJECTIVE FUNCTION:
- Inf............................................ -39.9992
Example of use of an external optimization algorithm

KUHN-TUCKER ERROR................................. 1.505471E-05
PARAMETER FOR CONSTRAINTS VIOLATION........... 0.00000
CONDITION NUMBER................................. 1.04448

--> THE LAST ITERATION GIVES AN OPTIMAL SOLUTION

<table>
<thead>
<tr>
<th>VALUE</th>
<th>LOW. BOUND</th>
<th>UPP. BOUND</th>
<th>PERTURBATION</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Inf 1</td>
<td>1.110</td>
<td>0.1000</td>
<td>10.00</td>
<td>1.0000E-03</td>
</tr>
<tr>
<td>2: Inf 2</td>
<td>2.221</td>
<td>0.1000</td>
<td>10.00</td>
<td>1.0000E-03</td>
</tr>
<tr>
<td>3: Inf 3</td>
<td>3.331</td>
<td>0.1000</td>
<td>10.00</td>
<td>1.0000E-03</td>
</tr>
<tr>
<td>4: Inf 4</td>
<td>4.442</td>
<td>0.1000</td>
<td>10.00</td>
<td>1.0000E-03</td>
</tr>
</tbody>
</table>

The source code of the procedure calling the MIDACO algorithm is provided in appendix 1.

The screenshot below shows the specific algorithm-parameters, the first 12 of them correspond to the PARAM() array of the MIDACO routine, the following ones are used to access to other parameters and are detailed in the MIDACO user's guide (MAXTIME, PRINTEVAL, SAVE2FILE and NI).

![Parameter screenshot]

In the studied example, the results obtained strongly depend on the values given to the parameters ANTS and KERNEL which with the default values (0 and 0) lead to a non-optimal solution.

With the above parameters, the results obtained are presented hereafter.
EXTERNAL OPTIMIZATION ALGORITHM

NUMBER OF VARIABLES OF THE PROBLEM ...................... 4
AMONG WHICH 0 ARE TEAR STREAM VARIABLES
AND 4 ARE OPTIMIZATION VARIABLES

NUMBER OF CONSTRAINTS OF THE PROBLEM .................... 0
NUMBER OF EQUALITY CONSTRAINTS ..................... 0
NUMBER OF INEQUALITY CONSTRAINTS ................... 0

PARAMETERS OF THE EXTERNAL ALGORITHM:
ACCURACY ........................................... 0
SEED ............................................... 0
FSTOP .............................................. 0
ALGOSTOP ........................................... 0
EVALSTOP ........................................... 100.000
FOCUS ............................................. 30.0000
KERNEL ............................................. 5.00000
ORACLE ............................................. 0
PARETOMAX ......................................... 0
EPSILON ............................................. 0
CHARACTER .......................................... 0
MAXTIME ............................................ 300.000
PRINTeval .......................................... 1.00000
SAVE2FILE .......................................... 2.00000
NI ................................................. 0

NUMERICAL PARAMETERS:
MAXIMUM NUMBER OF MAJOR ITERATIONS ..................... 200
MAXIMUM NUMBER OF SEQUENCE CALCULATIONS ............ 1000

RESULTS:
NUMBER OF ITERATIONS ......................... 499
NUMBER OF RUNS IN THE PROCESS ................. 501

VALUE OF THE OBJECTIVE FUNCTION:
- Inf ............................................ -39.9954

--> THE LAST ITERATION GIVES AN OPTIMAL SOLUTION

<table>
<thead>
<tr>
<th>VALUE</th>
<th>LOW. BOUND</th>
<th>UPP. BOUND</th>
<th>PERTURBATION</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Inf 1</td>
<td>1.114</td>
<td>0.1000</td>
<td>10.00</td>
<td>1.0000E-03</td>
</tr>
<tr>
<td>2: Inf 2</td>
<td>2.218</td>
<td>0.1000</td>
<td>10.00</td>
<td>1.0000E-03</td>
</tr>
<tr>
<td>3: Inf 3</td>
<td>3.329</td>
<td>0.1000</td>
<td>10.00</td>
<td>1.0000E-03</td>
</tr>
<tr>
<td>4: Inf 4</td>
<td>4.441</td>
<td>0.1000</td>
<td>10.00</td>
<td>1.0000E-03</td>
</tr>
</tbody>
</table>

One can note that the parameters described in the graphical user interface are printed in the simulation report.

For this example, an additional printing routine is implemented, which transfers in the simulation report the content of the MIDACO_SOLUTION.TXT file when the variable SAVE2FILE has a non-zero value. The source code of the printing routine is provided in appendix 2.
APPENDIX 1 - SOURCE CODE OF THE EXTERNAL OPTI ROUTINE

integer(4) function OPTI(PAR,NPT,ITEMAX,NAPM,IALP,ITER,KOPT,INFO, &
NCO,NCI,NVA,NF,F,C,X,XL,XU,AC,G,CN) result(rc)
!dec$ attributesstdcall, reference, alias:'OPTI', dllexport :: OPTI
use IFWIN
implicit none
integer(4), intent(in)    :: NPT, ITEMAX, NAPM, IALP(NVA), NCO, NCI, NVA, NF
integer(4), intent(inout) :: ITER, KOPT, INFO
real(8),    intent(inout) :: PAR(NPT)
real(8),    intent(in)    :: F(NF), C(NCO), XL(NVA), XU(NVA), AC(NVA)
real(8),    intent(inout) :: X(NVA), G(NF,NVA), CN(NCO,NVA)
integer(4), parameter :: OPTI_CONVERGED = 0
integer(4), parameter :: OPTI_FAILED    = 2
integer(4), parameter :: OPTI_RUNNING   = -1

! Dimensions of the optimization problem
integer(4) O, N, NI, M, ME

! MIDACO information and stop flags
integer(4) IFLAG, ISTOP
! MIDACO parameter
real(8) PARAM(12)
! MIDACO integer 'IW' and real'RW' workspace and pareto front 'PF'
integer(4), parameter :: LIW = 10000
integer(4), parameter :: LRW = 10000
integer(4), parameter :: LPF = 10000
integer(4) IW(LIW)
real(8) RW(LRW),PF(LPF)
! Parameter for stopping criteria, printing and license
integer(4) MAXTIME, MAXEVAL, PRINTEVAL, SAVE2FILE, I
character(len=60) :: KEY = 'MIDACO_LIMITED_VERSION___[CREATIVE_COMMONS_BY-NC-ND_LICENSE]'
!
real(8), allocatable :: C1(:)
integer(BOOL) rcDel
logical Found
integer(4) IOS, rcTemp, szTempPath
character(len=256) TempPath
if (KOPT /= 0) goto 100

!CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
!CC Step 0: Deleting possible output files     CCCCCCCCCCCCCCCCCCCCCCCCC
!CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
inquire(FILE='MIDACO_HISTORY.TXT', exist=Found)
if (Found) rcDel = DeleteFile('MIDACO_HISTORY.TXT')
inquire(FILE='MIDACO_SOLUTION.TXT', exist=Found)
if (Found) rcDel = DeleteFile('MIDACO_SOLUTION.TXT')
inquire(FILE='MIDACO_SCREEN.TXT', exist=Found)
if (Found) rcDel = DeleteFile('MIDACO_SCREEN.TXT')
inquire(FILE='MIDACO_PARETOFRONT.TXT', exist=Found)
if (Found) rcDel = DeleteFile('MIDACO_PARETOFRONT.TXT')

!CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
!CC Step 1: Problem definition     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC


Example of use of an external optimization algorithm

Step 1.A: Problem dimensions

\[
\begin{align*}
O &= NF \quad \text{! Number of objectives} \\
N &= NVA \quad \text{! Number of variables (in total)} \\
NI &= \text{nint}(PAR(16)) \quad \text{! Number of integer variables (0} \leq NI \leq N) \\
M &= NCO \quad \text{! Number of constraints (in total)} \\
ME &= NCO - NCI \quad \text{! Number of equality constraints (0} \leq ME \leq M)
\end{align*}
\]

if (M /= 0) then
    if (allocated(C1)) deallocate(C1)
    allocate(C1(M))
else
    if (allocated(C1)) deallocate(C1)
    allocate(C1(1))
end if

C1(:) = 0.D0

Step 2: Choose stopping criteria and printing options

Step 2.A: Stopping criteria

\[
\begin{align*}
\text{MAXEVAL} &= \text{nint}(PAR(13)) \quad \text{! Maximum evaluation budget (e.g. 1000000)} \\
\text{MAXTIME} &= \text{nint}(PAR(15)) \quad \text{! Maximum time limit (e.g. 300 = 10 Min)}
\end{align*}
\]

Step 2.B: Printing options

\[
\begin{align*}
\text{PRINTEVAL} &= \text{nint}(PAR(14)) \quad \text{! Print-Frequency for current best solution (e.g. 1000)} \\
\text{SAVE2FILE} &= \text{nint}(PAR(15)) \quad \text{! Save SCREEN and SOLUTION to TXT-files [0=NO/1=YES]}
\end{align*}
\]

Step 3: Choose MIDACO parameters (FOR ADVANCED USERS)

\[
\begin{align*}
\text{PARAM( 1)} &= 0.D0 \quad \text{! ACCURACY} \\
\text{PARAM( 2)} &= 0.D0 \quad \text{! SEED} \\
\text{PARAM( 3)} &= 0.D0 \quad \text{! FSTOP} \\
\text{PARAM( 4)} &= 0.D0 \quad \text{! ALGOSTOP} \\
\text{PARAM( 5)} &= 0.D0 \quad \text{! EVALSTOP} \\
\text{PARAM( 6)} &= 0.D0 \quad \text{! FOCUS} \\
\text{PARAM( 7)} &= 0.D0 \quad \text{! ANTS} \\
\text{PARAM( 8)} &= 0.D0 \quad \text{! KERNEL} \\
\text{PARAM( 9)} &= 0.D0 \quad \text{! ORACLE} \\
\text{PARAM(10)} &= 0.D0 \quad \text{! PARETOMAX} \\
\text{PARAM(11)} &= 0.D0 \quad \text{! EPSILON} \\
\text{PARAM(12)} &= 0.D0 \quad \text{! CHARACTER}
\end{align*}
\]

do I=1,12
    PARAM(I) = PAR(I)
end do

ITER = -1

Print MIDACO headline with basic information

call MIDACO_PRINT(I,PRINTEVAL,SAVE2FILE,IFLAG,ISTOP,F,C1,X,XL, &
XU,O,N,NI,N,M,ME,RW,PF,MAXEVAL,MAXTIME,PARAM,1,0,KEY)

KOPT = 15

Call MIDACO by Reverse Communication
Example of use of an external optimization algorithm

100 continue
! Incrementation of the iteration counter
ITER = ITER + 1
! Sign change for the inequality constraints
if (M /= 0) then
  C1(1:M) = C(1:M)
  do I=ME+1,M
    C1(I) = -C1(I)
  end do
end if

! Opening the specific history file (if SAVE2FILE >= 2)
if (SAVE2FILE >= 2) then
  szTempPath = len(TempPath)
  rcTemp = GetTempPath(szTempPath, TempPath)
  if (rcTemp > szTempPath .or. rcTemp == 0) then
    TempPath = '.\'
    rcTemp = 2
  end if
  open(UNIT=888,FILE=TempPath(1:rcTemp)//'IMPI_OPTI.tmp',STATUS='REPLACE',IOSTAT=IOS)
  write(888,1)
  1 format('#########################################################',/, &
    '### SOLUTION FORMAT:     F(1:O)    G(1:M)    X(1:N)   ###',/, &
    '#########################################################')
  write(888,2) O,M,N
  2 format('#',/, &
    '#        O         M         N  ',/, &
    '#',/, &
    3I10),
  call SAVE_HISTORY(1,O,N,M,F,C1,X,888,ISTOP)
  close(888)
end if

call MIDACO(1,O,N,NI,M,ME,X,F,C1,XL,XU,IFLAG, &
  ISTOP,PARAM,RW,LRW,IW,LIW,PF,LPF,KEY)

! Call MIDACO printing routine
call MIDACO_PRINT(2,PRINTEVAL,SAVE2FILE,IFLAG,ISTOP,F,C1,X, &
  XL,XU,0,N,NI,M,ME,RW,PF,MASEVAL,MATEVAL,PARAM,1,0,KEY)

if (ISTOP == 0) then
  rc = OPTI_RUNNING
else
  select case (IFLAG)
  case (1:7)
    rc = OPTI_CONVERGED
  case default
    rc = OPTI_FAILED
  end select
  select case (IFLAG)
  case (1, 3, 5, 7)
    INFO = 1
  case default
    INFO = IFLAG
  end select
  KOPT = 0
  if (allocated(C1)) deallocate(C1)
end if

end function OPTI
APPENDIX 2 - SOURCE CODE OF THE EXTERNAL IOPTI ROUTINE

```fortran
subroutine IOPTI(PAR,NPT)
!
dec$ attributes stdcall, reference, alias:'IOPTI', dllexport :: IOPTI
use IFWIN

implicit none

integer(4), intent(in) :: NPT
real(8),    intent(in) :: PAR(NPT)

integer(4) IOS, rc, SAVE2FILE, szTempPath

character(len=256) TempPath, LINE

SAVE2FILE = nint(PAR(15))

if (SAVE2FILE /= 0) then
  open(UNIT=889,FILE='MIDACO_SOLUTION.TXT',STATUS='OLD',IOSTAT=IOS)
  if (IOS /= 0) return
  szTempPath = len(TempPath)
  rc = GetTempPath(szTempPath, TempPath)
  if (rc > szTempPath .or. rc == 0) then
    TempPath = '.\'
    rc = 2
  end if
  open(UNIT=888,FILE=TempPath(1:rc)//'IMPO_OPTI.tmp',STATUS='REPLACE',IOSTAT=IOS)
  do while (.not.eof(889))
    read(889,'(A)') LINE
    write(888,'(A)') LINE
  end do
  close(888)
  close(889)
end if

end subroutine IOPTI
```