ProSim

EXEMPLE D'APPLICATION PROSIMPLUS

EXEMPLE D'UTILISATION D'UN ALGORITHME D'OPTIMISATION EXTERNE

INTERET DE L'EXEMPLE
Cet exemple présente l'utilisation d'un algorithme d'optimisation externe dans l'environnement de simulation de ProSimPlus. Les interfaces de communication avec l'algorithme externe sont détaillées ainsi qu'un exemple basique.

DIFFUSION	☑ Libre Internet	☐ Réservé clients ProSim	☐ Restreinte	☐ Confidentiel

FICHIER PROSIMPLUS CORRESPONDANT	*PSPS_E26_FR – Exemple optimiseur externe.pmp3*

TABLE DES MATIÈRES

1. INTRODUCTION 3

2. FONCTIONNEMENT DE L'OPTIMISEUR DANS L'ENVIRONNEMENT PROSIMPLUS 3

2.1. Principe 3

2.2. Gestion des courants coupés 3

2.3. Gestion des informations 4

2.4. Paramètres 5

3. INTERFACES DE COMMUNICATION AVEC L'ALGORITHME EXTERNE 5

3.1. Interface d'appel pour la résolution 5

3.2. Interface d'appel pour les impressions 7

3.3. Gestion des fonctions objectif 8

3.4. Gestion des contraintes 8

3.4.1. Contraintes inégalité 8

3.5. Impressions intermédiaires 8

3.6. Interface graphique 9

3.7. Rapport de simulation 11

4. EXEMPLE D'IMPLEMENTATION : LE SOLVEUR MIDACO 11

ANNEXE 1 – CODE SOURCE DE LA ROUTINE OPTI EXTERNE 16

ANNEXE 2 – CODE SOURCE DE LA ROUTINE IOPTI EXTERNE 19

1. INTRODUCTION

L'environnement de simulation de ProSimPlus dispose en standard de deux algorithmes d'optimisation, l'un basé sur un algorithme SQP (pour Sequential Quadratic Programming), le second sur un algorithme évolutionnaire (algorithme génétique). Tous deux ne fonctionnent qu'avec des variables continues correspondant à des paramètres opératoires du procédé. Dans le but offrir plus de souplesse à l'utilisateur, une nouvelle fonctionnalité a été ajoutée dans la version 3.5.16 de ProSimPlus ; elle permet de faire appel à un algorithme d'optimisation externe, fourni sous la forme d'une bibliothèque dynamique et pouvant traiter n'importe quel type de problème (multi-objectifs, variables continues, discrètes ou booléennes). Ce document présente un exemple d'utilisation de cette fonctionnalité.

Dans un premier temps, le mode de fonctionnement de l'optimiseur standard de ProSimPlus est rappelé, puis une seconde partie détaille les interfaces de communication, enfin un exemple de mise en place de l'algorithme d'optimisation MIDACO (www.midaco-solver.com) est détaillé sur un exemple basique. MIDACO utilise un algorithme évolutionnaire de type « Ant Colony Optimization » étendu pouvant traiter des problèmes multicritères en variables mixtes.

2. FONCTIONNEMENT DE L'OPTIMISEUR DANS L'ENVIRONNEMENT PROSIMPLUS

2.1. Principe

Le principe de fonctionnement de l'optimiseur de ProSimPlus repose sur une succession d'allers-retours (appelés « passages dans le RCM ») entre le module lui-même et les modules faisant partie de la boucle d'optimisation. Ceci repose notamment sur la nécessité de connaître les valeurs des fonctions à minimiser et des contraintes à différentes étapes du processus d'optimisation :

- A l'initialisation du processus de convergence
- Lors de l'évaluation de la fonction et des contraintes pour un ensemble de variables donné à l'itération courante
- Lors du calcul des sensibilités de la fonction et des contraintes entre deux itérations par perturbations numériques

La description du problème à résoudre ainsi que les paramètres associés à l'algorithme SQP sont décrits dans le manuel utilisateur auquel on se réfèrera pour tout complément d'information notamment sur les approches **Feasible Path** et **Unfeasible Path** ainsi que l'optimisation multi-périodes.

C'est donc à l'algorithme d'optimisation que revient la gestion de la phase courante de calcul et selon le type d'algorithme utilisé, le nombre de phases de calcul peut être variable. Un algorithme stochastique par exemple ne réalise que des évaluations pour différentes populations et après mutations et croisements, recommence le processus d'évaluation jusqu'à atteindre un nombre de générations fixé à l'avance.

2.2. Gestion des courants coupés

Sans entrer dans la description détaillée du mode **Unfeasible Path**, il faut savoir que la présence de courants coupés dans la boucle d'optimisation impose à l'optimiseur de gérer les variables associées à ces courants à la fois comme variables d'optimisation et sous forme de contraintes égalités de la forme :

$$x - g(x) = 0$$

Avec : x Valeur courante de la variable

$g(x)$ Valeur de la variable après calcul des modules du RCM.

Les variables d'un courant coupé sont constituées des débits molaires partiels des différents constituants pris en compte dans la simulation et éventuellement de la température et de la pression du courant selon le choix de l'utilisateur. Généralement la température est sélectionnée tandis que la pression n'est utilisée que si le module dont est issu le courant calcule automatiquement la pression du courant (module de calcul de pertes de charges par exemple).

Par ailleurs, les variables associées aux courants coupés doivent impérativement se trouver en premier dans le vecteur des variables d'optimisation (variables d'action) et dans l'ordre suivant :

Débit partiel du constituant 1 pour le courant coupé n°1

Débit partiel du constituant 2 pour le courant coupé n°1

…

Débit partiel du constituant NC pour le courant coupé n°1

Température du courant coupé n°1 (si la température est choisie comme variable)

Pression du courant coupé n°1 (si la pression est choisie comme variable)

….

Les variables d'optimisation (courants d'information sortants du module OPTI) viennent ensuite.

L'utilisation d'un algorithme d'optimisation externe n'échappe pas à cette règle à partir du moment où c'est l'approche **Unfeasible Path** qui est sélectionnée.

L'approche **Feasible Path**, qui impose l'utilisation d'un module de gestion des recyclages, permet de s'affranchir de cette contrainte bien que la séquence de calcul doive alors être fournie manuellement (cf. Manuel utilisateur de ProSimPlus).

2.3. Gestion des informations

Les différentes informations nécessaires à l'utilisation du module OPTI sont véhiculées par l'intermédiaire de courants d'information selon le schéma et l'ordre suivant :

- Informations entrantes
    1. Fonction objectif
    2. Contrainte égalité n°1
    3. …
    4. Contrainte égalité N°**NE**
    5. Contrainte inégalité n°1
    6. …
    7. Contrainte inégalité n°**NI**

- Informations sortantes
    1. Variable d'action n°1
    2. …
    3. Variable d'action n°**NA**

Dans la version standard de l'optimiseur (algorithme SQP), une seule fonction objectif est autorisée, ce qui signifie que l'optimisation multicritère n'est pas possible. Néanmoins dans un algorithme externe cette possibilité peut exister et la possibilité de connecter plusieurs fonctions objectif a été implémentée. Pour l'algorithme SQP standard, cette limite est toujours d'actualité, tandis que pour un algorithme externe, elle a été étendue à 10 fonctions objectif simultanées.

2.4. Paramètres

Afin de conserver le maximum de souplesse dans l'utilisation d'un algorithme externe, de nombreux paramètres du module OPTI existant ont été conservés et sont utilisables conjointement avec un algorithme externe. Il s'agit notamment des paramètres suivants :

- Sélection des courants coupés
- Sélection des paramètres associés aux courants coupés : bornes et incréments pour une éventuelle analyse de sensibilité
- Paramètres relatifs aux variables d'optimisation (variables d'action) : bornes, incréments et type d'incréments pour l'analyse de sensibilité
- Paramètres numériques généraux : nombre maximum de passages dans le RCM, nombre maximum d'itérations, indicateur d'impressions intermédiaires, nombre de pas de repos, nombre de contraintes égalité, nombre de périodes

Concernant les paramètres spécifiques à chaque algorithme externe, ils sont disponibles sous forme de grille de saisie de type « tableur » permettant d'adapter ceux-ci à chaque algorithme particulier et d'offrir le maximum de souplesse d'utilisation.

3. INTERFACES DE COMMUNICATION AVEC L'ALGORITHME EXTERNE

Deux interfaces de communication ont été implémentées, respectivement pour la résolution proprement dite et pour les impressions ; elles sont présentées ci-après.

3.1. Interface d'appel pour la résolution

L'interface d'appel de l'algorithme d'optimisation externe est détaillée ci-dessous :

```
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)
```


La convention d'appel utilisée est **STDCALL** et les arguments sont passés par référence. Le nom du point d'entrée est par ailleurs imposé et doit être impérativement **OPTI** (en majuscules).

La signification des arguments de la fonction est donnée dans le tableau ci-dessous :

Nom	Type	Taille (octets)	E/S	Dimension	Description
NPT	Entier	4	E	1	Nombre de paramètres utilisateur
ITEMAX	Entier	4	E	1	Nombre maximum d'itérations
NAPM	Entier	4	E	1	Nombre maximum de passages dans le RCM
IALP	Entier	4	E	NVA	Types d'incrément pour l'analyse de sensibilité IALP(i)=1 : Absolu, IALP(i)=2 : Proportionnel
NCO	Entier	4	E	1	Nombre total de contraintes
NCI	Entier	4	E	1	Nombre de contraintes inégalité
NVA	Entier	4	E	1	Nombre de variables d'optimisation
NF	Entier	4	E	1	Nombre de fonctions objectif
ITER	Entier	4	E/S	1	Numéro de l'itération en cours
KOPT	Entier	4	E/S	1	Indicateur de phase de calcul
INFO	Entier	4	S	1	Information relative à l'arrêt des calculs
PAR	Réel	8	E/S	NPT	Vecteur de paramètres pour l'algorithme externe
F	Réel	8	E	NF	Vecteur des fonctions objectif
C	Réel	8	E/S	NCO	Vecteur des contraintes
XL	Réel	8	E	NVA	Vecteur des bornes min des variables d'action
XU	Réel	8	E	NVA	Vecteur des bornes max des variables d'action
AC	Réel	8	E	NVA	Vecteur des incréments pour l'analyse de sensibilité
X	Réel	8	E/S	NVA	Vecteur des variables d'optimisation
G	Réel	8	E/S	(NF,NVA)	Tableau de stockage des dérivées des fonctions par rapport aux variables d'optimisation
CN	Réel	8	E/S	(NCO,NVA)	Tableau de stockage des dérivées des contraintes par rapport aux variables d'optimisation

Note : Les tableaux **G()** et **CN()** sont fournis comme espace de stockage pour les dérivées des fonctions et des contraintes par rapport aux variables d'optimisation et peuvent ne pas être utilisés en fonction de l'algorithme externe utilisé.

A chaque appel, la fonction renvoie un indicateur de l'état des calculs (code de retour de la fonction) qui peut prendre les valeurs suivantes :

OPTI_RUNNING	-1	Optimisation en cours
OPTI_CONVERGED	0	Optimisation terminée
OPTI_FAILED	2	Echec de l'optimisation

Par ailleurs, en cours de calcul, la variable **KOPT** est utilisée par la fonction **OPTI** pour connaitre la phase en cours de calcul. La variable **KOPT** prend la valeur 0 au premier appel (ce qui permet d'effectuer certaines initialisations par exemple ou de réaliser des traitements ne devant être effectués qu'une seule fois) et doit ensuite être modifiée par la fonction avec une valeur différente de 0. Les valeurs sont libres et laissées au choix du développeur. En fin de calcul la variable **KOPT** doit être remise à zéro (c'est-à-dire lorsque le code de retour de la fonction est **OPTI_CONVERGED** ou **OPTI_FAILED**).

La variable **INFO** est utilisée pour connaitre les raisons de l'arrêt des calculs. Cette variable peut prendre les valeurs suivantes :

INFO	Description
1	Optimisation terminée avec succès
≠ 1	Echec de l'optimisation

3.2. Interface d'appel pour les impressions

L'interface d'appel pour les impressions de l'algorithme d'optimisation externe est détaillée ci-dessous :

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

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

La convention d'appel utilisée est **STDCALL** et les arguments sont passés par référence. Le nom du point d'entrée est par ailleurs imposé et doit être impérativement **IOPTI** (en majuscules).

La signification des arguments de la fonction est la même que pour l'interface principale de résolution.

Note : Cette interface est optionnelle, si elle n'est pas présente dans la bibliothèque dynamique, elle ne sera pas appelée. Dans le cas où elle est implémentée, les impressions devront être réalisées par l'intermédiaire d'un fichier temporaire nommé **IMPO_OPTI.tmp** devant se trouver dans le répertoire temporaire de l'utilisateur courant. Ce fichier doit être ouvert dans la routine **IOPTI** puis fermé en sortie afin que le programme appelant puisse y lire les informations stockées et le détruire ensuite. Ce fichier peut permettre, par exemple, de transférer des impressions réalisées par l'algorithme d'optimisation dans un fichier particulier vers le rapport de simulation. Si il est nécessaire d'accéder aux paramètres utilisateur de l'algorithme, le vecteur **PAR** est transmis en argument mais ne doit pas être modifié (lecture seule).

3.3. Gestion des fonctions objectif

Le vecteur **F()** contient les valeurs des fonctions objectif du problème d'optimisation courant. Il contient les valeurs transmises au module OPTI par l'intermédiaire des courants d'information qui sont connectés aux ports intitulés « Fonction objectif i ». Le vecteur **F()** est actualisé à chaque passage avec les valeurs des fonctions objectif courantes. Dans le cas d'un algorithme d'optimisation nécessitant l'évaluation du gradient des fonctions objectif, il est donc nécessaire de sauvegarder les valeurs du vecteur **F()** à chaque nouvelle itération dans un vecteur local.

3.4. Gestion des contraintes

Le vecteur **G()** contient les valeurs des contraintes du problème d'optimisation courant. Il contient les valeurs des contraintes égalité, suivies de celles des contraintes inégalité. Dans le cas où le module OPTI gère les courants coupés, c'est à dire en approche **Unfeasible Path**, les contraintes égalité sont composées en premier lieu des contraintes relatives aux courants coupés, suivies des autres contraintes égalités transmises au module par l'intermédiaire des courants d'information connectés aux ports intitulés « Contrainte i ».

Les ports de connexion « Contrainte 1 » à « Contrainte NE » devront impérativement être utilisés pour les **NE** contraintes égalités, les ports « Contrainte NE+1 » à « Contrainte NE+NI » par celles correspondant aux contraintes inégalité.

A l'instar du vecteur des fonctions objectif, le vecteur **G()** est actualisé à chaque passage avec les valeurs des contraintes courantes. Dans le cas d'un algorithme d'optimisation nécessitant l'évaluation du gradient des contraintes, il est donc nécessaire de sauvegarder les valeurs du vecteur **G()** à chaque nouvelle itération dans un vecteur local.

3.4.1. Contraintes inégalité

Les contraintes inégalités doivent être écrites de manière à ce que la valeur de la contrainte soit négative lorsqu'elle est satisfaite, c'est-à-dire qu'elles doivent suivre le même fonctionnement que pour le module OPTI standard. Par exemple si une contrainte est exprimée sous la forme :

$$a \leq f_i \leq b$$

Alors le vecteur **G()**, contiendra les expressions suivantes :

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

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

Dans le cas où l'algorithme externe impose qu'elles soient positives (comme par exemple avec l'algorithme MIDACO), leur signe devra être changé avant l'appel à la routine de résolution.

3.5. Impressions intermédiaires

En cours de calcul, il est possible d'imprimer certains résultats intermédiaires dans le fichier historique (extension .HIS). Ces impressions sont réalisées par l'intermédiaire d'un fichier temporaire nommé **IMPI_OPTI.tmp** devant se trouver dans le répertoire temporaire de l'utilisateur courant. Ce fichier doit être ouvert à chaque appel de la routine **OPTI** puis fermé en sortie afin que le programme appelant puisse y lire les informations stockées et le détruire ensuite.

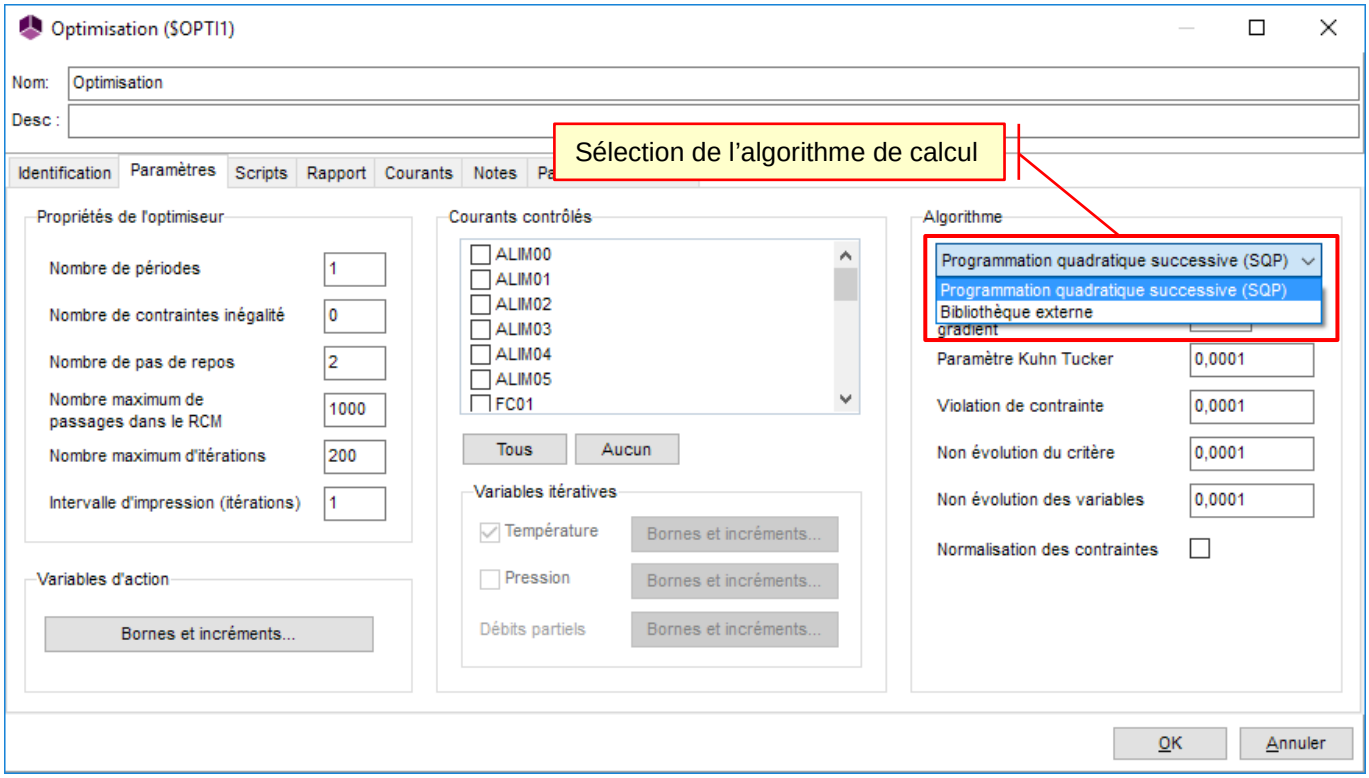
Si aucune information n'est requise, il n'est pas nécessaire que ce fichier soit créé. Ceci peut être le cas si par exemple l'algorithme externe génère lui-même des fichiers intermédiaires, néanmoins ces fichiers ne pourront pas être insérés ni dans le rapport de simulation ni dans le fichier historique des calculs.

3.6. Interface graphique

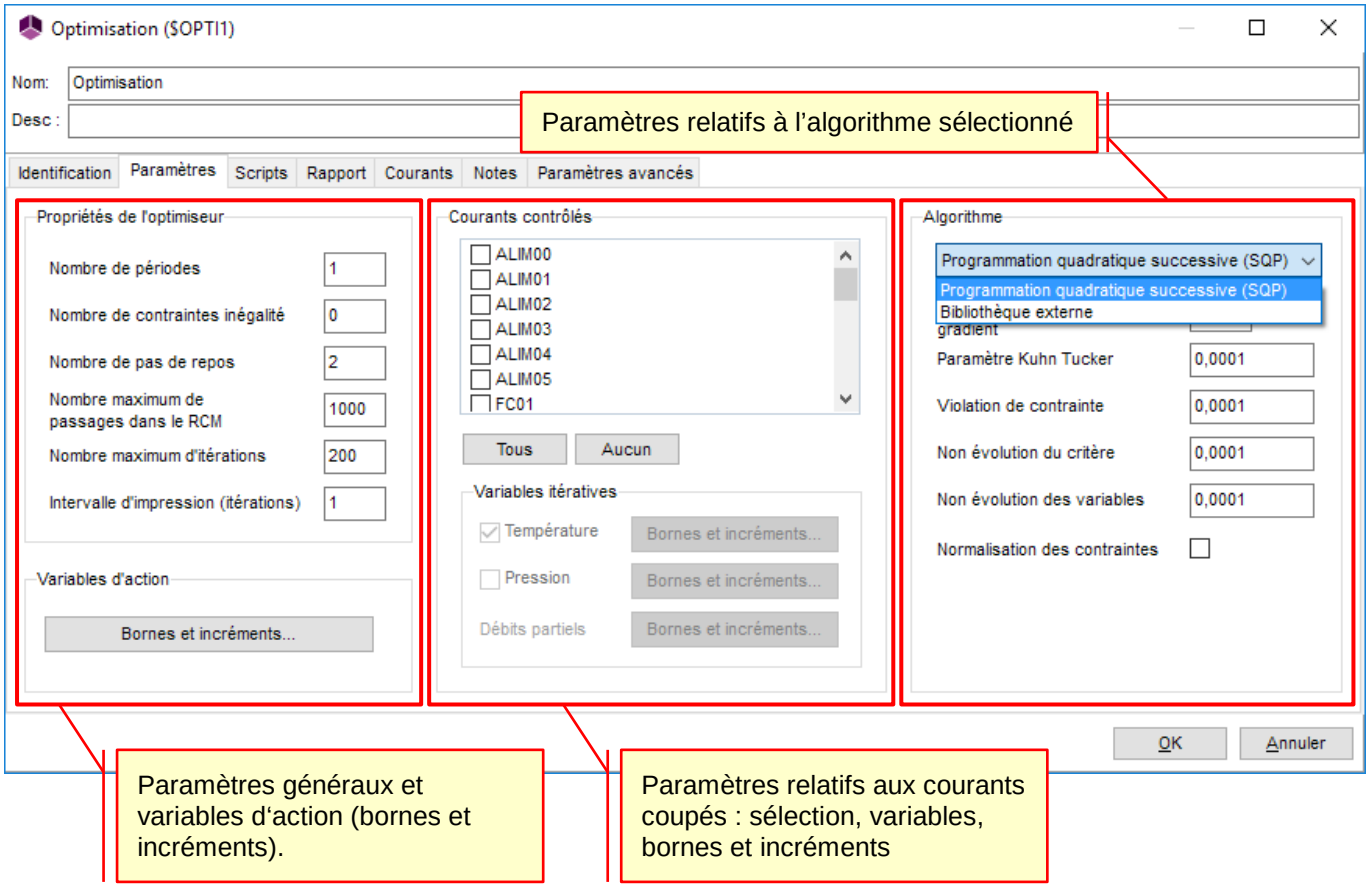
Les copies d'écrans ci-dessous présentent les modifications réalisées dans le module « Optimiseur » de ProSimPlus afin de rendre accessible un algorithme d'optimisation externe.

Pour cela, l'onglet principal contenant les paramètres a été réagencé afin de séparer plus clairement les paramètres généraux (paramètres numériques et paramètres associés aux variables d'action) des paramètres liés aux courants coupés (approche **Unfeasible Path**) et des paramètres propres à l'algorithme d'optimisation choisi.

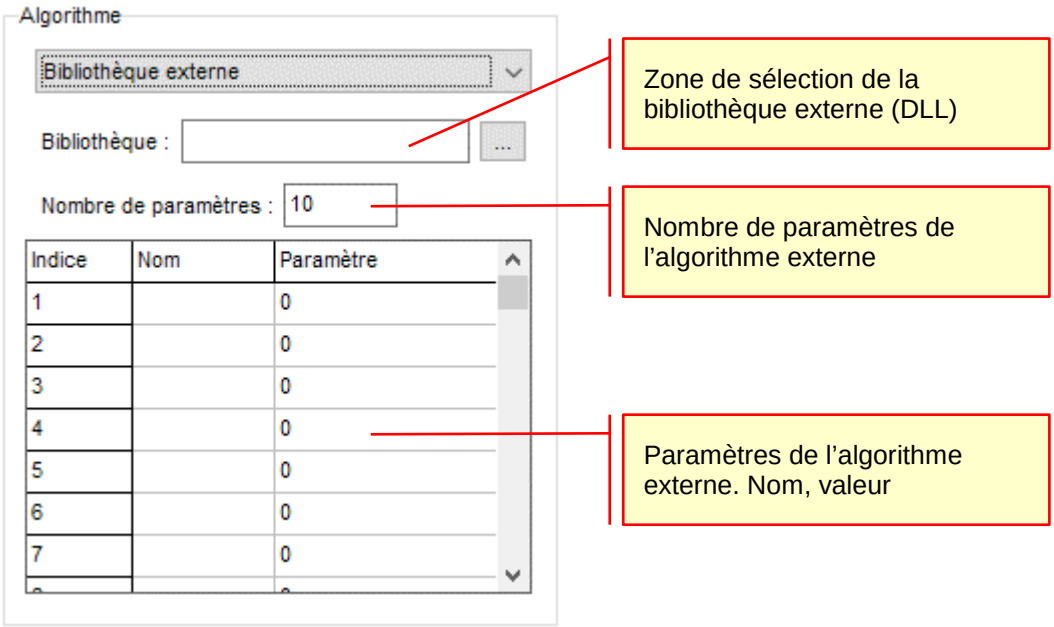
Sélection de l'algorithme d'optimisation



Autres paramètres du module « Optimiseur »



Paramètres relatifs à l'algorithme externe



3.7. Rapport de simulation

Le rapport de simulation reprend les éléments génériques du module d'optimisation tels que les données relatives aux fonctions objectif, aux contraintes et aux variables d'optimisation. Il reprend également les éléments relatifs aux paramètres numériques standards du module : variables d'action et bornes, courants coupés et bornes. Enfin il reprend les paramètres utilisateur décrits dans la fenêtre ci-dessus avec leurs noms et valeurs numériques. Si une routine d'impression est implémentée, les sorties réalisées via celle-ci sont également reprises dans le rapport à la suite des impressions « standard ».

4. EXEMPLE D'IMPLEMENTATION : LE SOLVEUR MIDACO

Ce paragraphe détaille un exemple d'utilisation d'un algorithme externe. Le solveur sélectionné est celui développé par M. Schlueter et M. Munemoto de l'université d'Hokkaido. Il s'agit d'un algorithme d'utilisation générale pour des problèmes mono et multi-objectifs initialement développé pour la résolution de problèmes non linéaires en variables mixtes (MINLP). Dans sa version actuelle, il est basé sur un algorithme évolutionnaire connu sous le nom « **Ant Colony Optimization** », d'où il tire son nom : **M**ixed **I**nteger **D**istributed **A**nt **C**olony **O**ptimization.

Une version d'évaluation, limitée à quatre variables d'optimisation est disponible sur le site web www.midaco-solver.com. C'est cette version qui a été testée dans le cadre de cet exemple.

Le problème ayant servi de test est un exemple d'optimisation simple pour lequel on dispose de la solution analytique et de celle fournie par l'optimiseur standard de ProSimPlus basé sur un algorithme de type SQP (**S**uccessive **Q**uadratic **P**rograming).

Il s'agit d'un exemple purement théorique qui est le suivant : on dispose de 4 alimentations dont on cherche à maximiser la fonction suivante qui dépend des débits des 4 alimentations.

$$f = -(f_1 + f_2 + f_3 + f_4)$$

Avec :

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

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

Et

x_i Débit molaire total de l'alimentation n°i

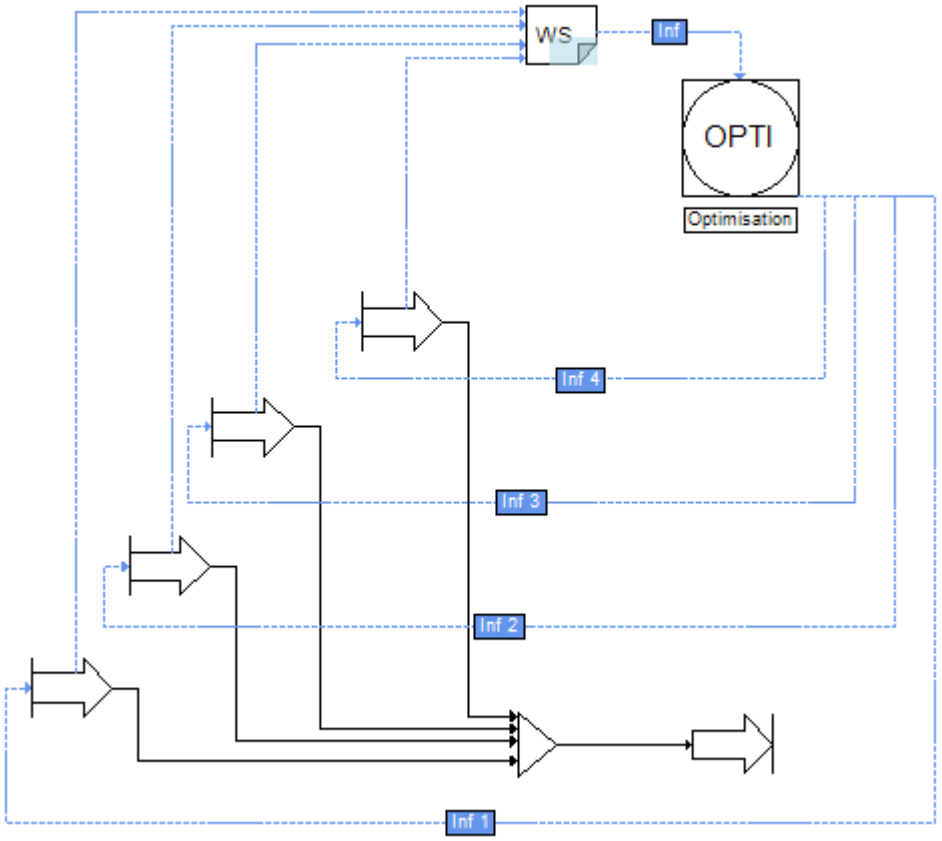
La solution analytique est :

$$f = -40$$

Et $x_1 = 1.111, \quad x_2 = 2.222$

$x_3 = 3.333, \quad x_4 = 4.444$

La fonction objectif est calculée à l'aide d'un script qui récupère les valeurs des débits des différentes alimentations et le module OPTI ajuste ces mêmes débits. Il s'agit d'un problème sans contrainte dont le flowsheet ProSimPlus est le suivant :



Les variables sont bornées entre 0.1 kmol/h et 10 kmol/h

L'utilisation de l'algorithme SQP de ProSimPlus avec les paramètres par défaut (hormis les bornes) conduit aux résultats suivants :

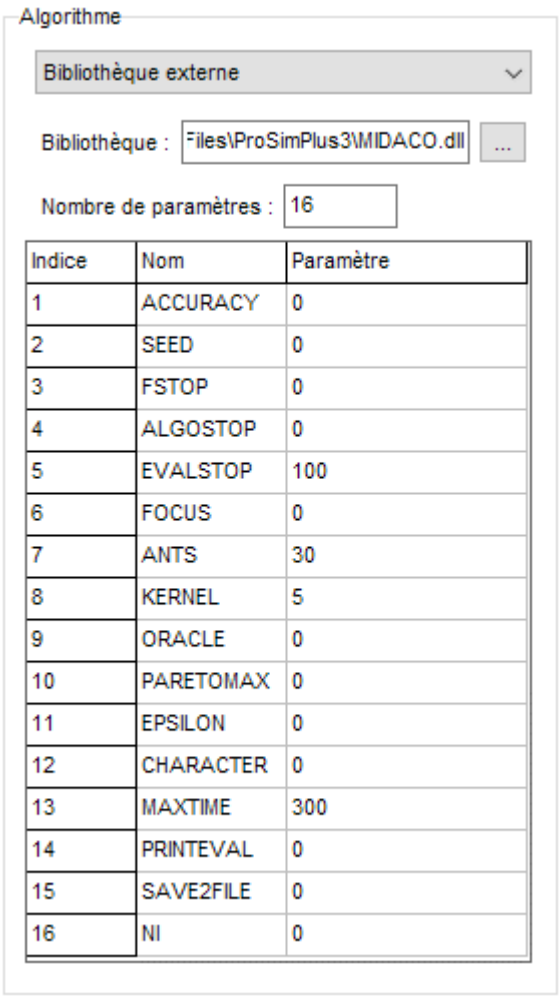
```
      ALGORITHME DE PROGRAMMATION QUADRATIQUE SUCCESSIVE
      ************************************************


NOMBRE DE VARIABLES DU PROBLEME.........................    4
     DONT   0 CORRESPONDENT A DES COURANTS CONTROLES
       ET   4 SONT DES PARAMETRES D'OPTIMISATION

NOMBRE DE CONTRAINTES DU PROBLEME.......................    0
     NOMBRE DE CONTRAINTES DE TYPE EGALITE..............    0
     NOMBRE DE CONTRAINTES DE TYPE INEGALITE............    0

LE GRADIENT DU CRITERE EST CALCULE:
     - PAR DIFFERENCES FINIES D'ORDRE 1

PARAMETRES NUMERIQUES :
     NOMBRE MAXIMUM D'ITERATIONS MAJEURES..............  200
     NOMBRE MAXIMUM DE PASSAGES SEQUENTIELS.............. 1000
     TOLERANCE SUR L'ERREUR DE KUHN-TUCKER.............. 1.000000E-04
     TOLERANCE DE NON-EVOLUTION DES VARIABLES........... 1.000000E-04
     TOLERANCE DE NON-EVOLUTION DE LA FONCTION.......... 1.000000E-04
```



```
RESULTATS :
     NOMBRE D'ITERATIONS.............................   23
     NOMBRE DE PASSAGES DANS LE PROCEDE..............  130
     VALEUR DE LA FONCTION OBJECTIF:
       - Inf........................................... -39.9992
     ERREUR DE KUHN-TUCKER........................... 1.505471E-05
     PARAMETRE DE VIOLATION DES CONTRAINTES.............  0.00000
     NOMBRE DE CONDITIONNEMENT.......................  1.04448


--> LA DERNIERE ITERATION FOURNIT UNE SOLUTION OPTIMALE


                              VALEUR     BORNE INF.   BORNE SUP.   INCREMENT       TYPE

PARAMETRES D'OPTIMISATION :
---------------------------
 1: Inf 1                    1.110       0.1000        10.00       1.0000E-03  PROPORTIONNEL
 2: Inf 2                    2.221       0.1000        10.00       1.0000E-03  PROPORTIONNEL
 3: Inf 3                    3.331       0.1000        10.00       1.0000E-03  PROPORTIONNEL
 4: Inf 4                    4.442       0.1000        10.00       1.0000E-03  PROPORTIONNEL
```

Le code source de la routine d'appel à l'algorithme MIDACO est donné en annexe 1.

La copie d'écran ci-après présente les paramètres utilisateur de l'algorithme, les 12 premiers correspondent au vecteur **PARAM()** de la routine MIDACO, les suivants sont utilisés pour accéder aux autres paramètres de l'algorithme et sont détaillés dans la notice d'utilisation du solveur (MAXTIME, PRINTEVAL, SAVE2FILE et NI).



Indice	Nom	Paramètre
1	ACCURACY	0
2	SEED	0
3	FSTOP	0
4	ALGOSTOP	0
5	EVALSTOP	100
6	FOCUS	0
7	ANTS	30
8	KERNEL	5
9	ORACLE	0
10	PARETOMAX	0
11	EPSILON	0
12	CHARACTER	0
13	MAXTIME	300
14	PRINTEVAL	0
15	SAVE2FILE	0
16	NI	0

Dans l'exemple étudié, les résultats obtenus dépendent fortement de la valeur donnée aux paramètres **ANTS** et **KERNEL** qui avec les valeurs par défaut (0 et 0) conduisent à une solution non optimale.

Avec les paramètres ci-dessus les résultats obtenus sont présentés ci-après.

```
  ALGORITHME D'OPTIMISATION EXTERNE
  *********************************


NOMBRE DE VARIABLES DU PROBLEME.........................    4
     DONT   0 CORRESPONDENT A DES COURANTS CONTROLES
       ET   4 SONT DES PARAMETRES D'OPTIMISATION

NOMBRE DE CONTRAINTES DU PROBLEME.......................    0
     NOMBRE DE CONTRAINTES DE TYPE EGALITE..............    0
     NOMBRE DE CONTRAINTES DE TYPE INEGALITE............    0

PARAMETRES DE L'ALGORITHME EXTERNE:
     ACCURACY.............................................  0
     SEED.................................................  0
     FSTOP................................................  0
     ALGOSTOP.............................................  0
     EVALSTOP.............................................  100.000
     FOCUS................................................  0
     ANTS.................................................  30.0000
     KERNEL...............................................  5.00000
     ORACLE...............................................  0
     PARETOMAX............................................  0
     EPSILON..............................................  0
     CHARACTER............................................  0
     MAXTIME..............................................  300.000
     PRINTEVAL............................................  0
     SAVE2FILE............................................  0
     NI...................................................  0

PARAMETRES NUMERIQUES :
     NOMBRE MAXIMUM D'ITERATIONS MAJEURES..............  200
     NOMBRE MAXIMUM DE PASSAGES SEQUENTIELS.............. 1000

RESULTATS :
     NOMBRE D'ITERATIONS..................................  499
     NOMBRE DE PASSAGES DANS LE PROCEDE.................  501
     VALEUR DE LA FONCTION OBJECTIF:
       - Inf.............................................. -39.9954


--> LA DERNIERE ITERATION FOURNIT UNE SOLUTION OPTIMALE


                              VALEUR     BORNE INF.   BORNE SUP.   INCREMENT       TYPE

PARAMETRES D'OPTIMISATION :
---------------------------
 1: Inf 1                    1.114       0.1000        10.00       1.0000E-03  PROPORTIONNEL
 2: Inf 2                    2.218       0.1000        10.00       1.0000E-03  PROPORTIONNEL
 3: Inf 3                    3.329       0.1000        10.00       1.0000E-03  PROPORTIONNEL
 4: Inf 4                    4.441       0.1000        10.00       1.0000E-03  PROPORTIONNEL
```

On notera que les paramètres décrits dans l'interface graphique du module sont repris dans le rapport de simulation.

Pour cet exemple, une routine d'impression additionnelle est implémentée et transfère dans le rapport de simulation le contenu du fichier MIDACO_SOLUTION.TXT lorsque la variable SAVE2FILE est différente de zéro. La routine d'impression est fournie en annexe 2.

ANNEXE 1 – CODE SOURCE DE LA ROUTINE OPTI EXTERNE

```
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
    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     CCCCCCCCCCCCCCCCCCCCCCCC
    !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     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```



```
!CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
!
!     Step 1.A : Problem dimensions
!     CCCCCCCCCCCCCCCCCCCCCCCCCCCC
O  = NF             ! Number of objectives
N  = NVA            ! Number of variables (in total)
NI = nint(PAR(16))  ! Number of integer variables (0 <= NI <= N)
M  = NCO            ! Number of constraints (in total)
ME = NCO - NCI      ! Number of equality constraints (0 <= ME <= M)

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

!CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
!CC Step 2: Choose stopping criteria and printing options    CCCCCCCCCCC
!CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
!
! Step 2.A : Stopping criteria
! CCCCCCCCCCCCCCCCCCCCCCCCCCC
MAXEVAL = NAPM              ! Maximum evaluation budget (e.g. 1000000)
MAXTIME = nint(PAR(13))     ! Maximum time limit (e.g. 300 = 10 Min)
!
! Step 2.B : Printing options
! CCCCCCCCCCCCCCCCCCCCCCCCCC
PRINTEVAL = nint(PAR(14))   ! Print-Frequency for current best solution (e.g. 1000)
SAVE2FILE = nint(PAR(15))   ! Save SCREEN and SOLUTION to TXT-files [0=NO/1=YES]

!CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
!CC Step 3: Choose MIDACO parameters (FOR ADVANCED USERS)    CCCCCCCCCCC
!CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
!
!PARAM( 1) = 0.D0  ! ACCURACY
!PARAM( 2) = 0.D0  ! SEED
!PARAM( 3) = 0.D0  ! FSTOP
!PARAM( 4) = 0.D0  ! ALGOSTOP
!PARAM( 5) = 0.D0  ! EVALSTOP
!PARAM( 6) = 0.D0  ! FOCUS
!PARAM( 7) = 0.D0  ! ANTS
!PARAM( 8) = 0.D0  ! KERNEL
!PARAM( 9) = 0.D0  ! ORACLE
!PARAM(10) = 0.D0  ! PARETOMAX
!PARAM(11) = 0.D0  ! EPSILON
!PARAM(12) = 0.D0  ! CHARACTER
do I=1,12
   PARAM(I) = PAR(I)
end do

ITER = -1

! Print MIDACO headline with basic information
call MIDACO_PRINT(1,PRINTEVAL,SAVE2FILE,IFLAG,ISTOP,F,C1,X,XL, &
                  XU,O,N,NI,M,ME,RW,PF,MAXEVAL,MAXTIME,PARAM,1,0,KEY)

KOPT = 15

!CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
!
!       Call MIDACO by Reverse Communication
```



```
    !
    !CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

100 continue
    ! Incrémentation du numéro d'itération
    ITER = ITER + 1
    ! Changement de signe des contraintes inégalité
    if (M /= 0) then
        C1(1:M) = C(1:M)
        do I=ME+1,M
            C1(I) = -C1(I)
        end do
    end if

    ! Ouverture du fichier historique spécifique
    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,O,N,NI,M,ME,RW,PF,MAXEVAL,MAXTIME,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
```


ANNEXE 2 – CODE SOURCE DE LA ROUTINE IOPTI EXTERNE

```
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
```

