



OUTIL DE MODÉLISATION & D'OPTIMISATION POUR LES PROBLÈMES *MIP*
(*MIXED INTEGER PROGRAMMING*)

Yacine Gaoua
DRT/LITEN/DTBH/SSETI/LSED
13/02/2020

ETAT DE L'ART SUR LES OUTILS DE MODÉLISATION


- Outils de modélisation *MILP/MINLP*: **interfaçage avec tous les solveurs**



- Outils de modélisation *MILP*: **propre solveur**



- Outils de modélisation *MILP* gratuit: **interfaçage avec les solveurs *MILP***



Rehearse: (C++ API) Dépendance de l'interface Coin-Osi & temps élevé pour la construction des grands problèmes d'optimisation

FlopC++: (C++ API) Modélisation modulaire compliquée

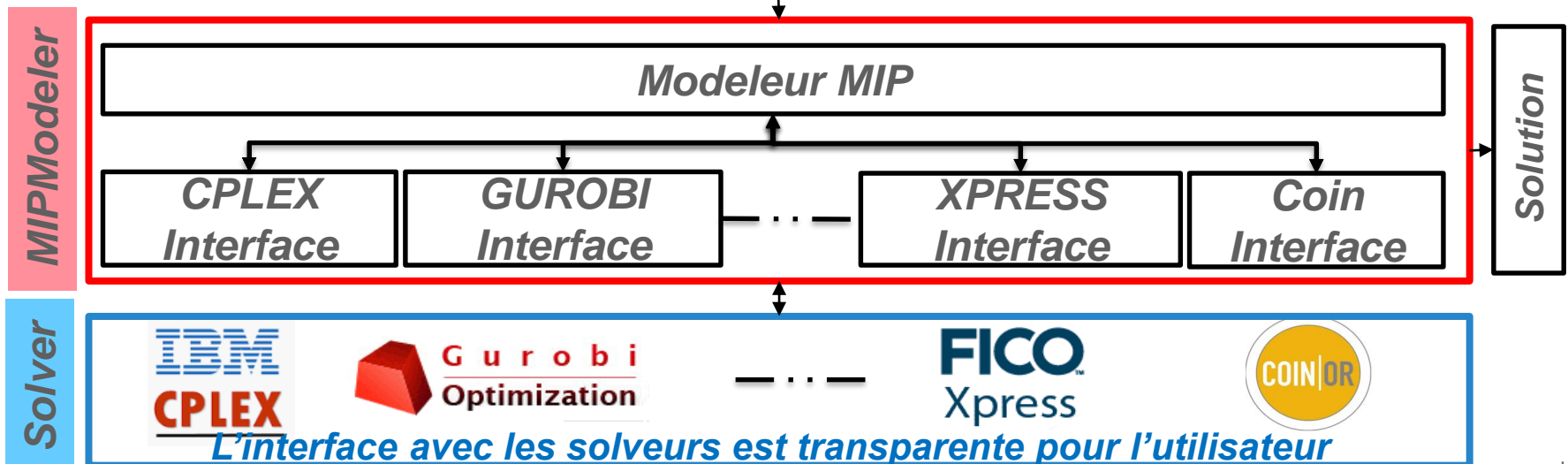
Cmpl: (C++ API) Licence GPLv3

Pulp: (Python API)

C'EST QUOI MIPMODELER ?

- Outil de modélisation *MILP*
 - C++ orienté Object *API*
 - Ecriture des problèmes d'optimisation facile et rapide
 - Efficacité lors de la construction des problèmes d'optimisation (même avec des millions de variables & de contraintes)
 - Adapté pour la modélisation modulaire
 - Intègre une bibliothèque de fonctions génériques *MILP*
 - Débogage facile et rapide des modèles d'optimisation
 - Interfaçage avec tous les solveurs *MILP* (commerciaux & open-source)

Problème d'optimisation



LES VARIABLES

- **Classe *MIPVariableND***

- **Dimension:** seulement pour les variables de dimension $N > 0$
- **LowerBound:** borne *INF* sur la variable (par défaut *-MIP_INFINITY*)
- **UpperBound:** borne *SUP* sur la variable (par défaut *MIP_INFINITY*)
- **Type:** type de la variable qui peut être soit une variable continue *MIP_FLOAT* soit une variable entière *MIP_INT* (par défaut *MIP_FLOAT*)

- **Méthodes utiles**

- **fix(.):** fixer la variable ou une partie de la variable à une valeur constante
- **lower(.):** changer la borne *INF* de la variable ou une partie de la variable
- **upper(.):** changer la borne *SUP* de la variable ou une partie de la variable
- **extractValue(.):** retourner la valeur de la variable après la résolution

- **Bonne pratique de modélisation**

- *MIPModeler* offre la possibilité de modéliser les problèmes d'optimisation d'une façon efficace avec le moins de variables/contraintes

***MIPVariable1D* $x(T, 0, \text{MIP_INFINITY})$**
***model.add*(x)**

***model.add*($x(t) \geq ub(t) \forall t \in T'$)**
***model.add*($lb(t) \leq x(t) \forall t \in T'$)**

Ajout de 2. T' contraintes au modèle

***MIPVariable1D* $x(T, 0, \text{MIP_INFINITY})$**
***model.add*(x)**

$x(t).$ *upper*($ub(t) \forall t \in T'$)
 $x(t).$ *lower*($lb(t) \forall t \in T'$)

Rien à rajouter au modèle

LES VARIABLES SOS

- **Classe *MIPSpecialOrderedSet***
 - MIPModeler permet de définir des variables de type *SOS1* & *SOS2* supporter par la plupart des solveurs d'optimisation à l'exception de *GLPK* (solveur gratuit), *MOSEK* (solveur commercial)
- **Méthodes utiles**
 - ***add(Variable, Type)***: ajouter une variable d'optimisation comme étant une variable SOS de type *MIP_SOS1* ou *MIP_SOS2* à l'ensemble des variables SOS
 - ***close()***: effacer le contenu de *MIPSpecialOrderedSet*
- **C'est quoi une variable SOS**
 - ***SOS1***: Sur un ensemble de variables *V* de type *SOS1*, une seule variable prend une valeur non-nulle, les *V-1* autres variables sont nulles
 - ***SOS2***: Sur un ensemble de variables *V* de type *SOS2*, deux variables consécutives prennent des valeurs non-nulles, les *V-2* autres variables sont nulles
 - Les variables *SOS1* & *SOS2* permettent de modéliser plus facilement certaines situations (ex. la linéarisation par morceaux, ...)

LES EXPRESSIONS

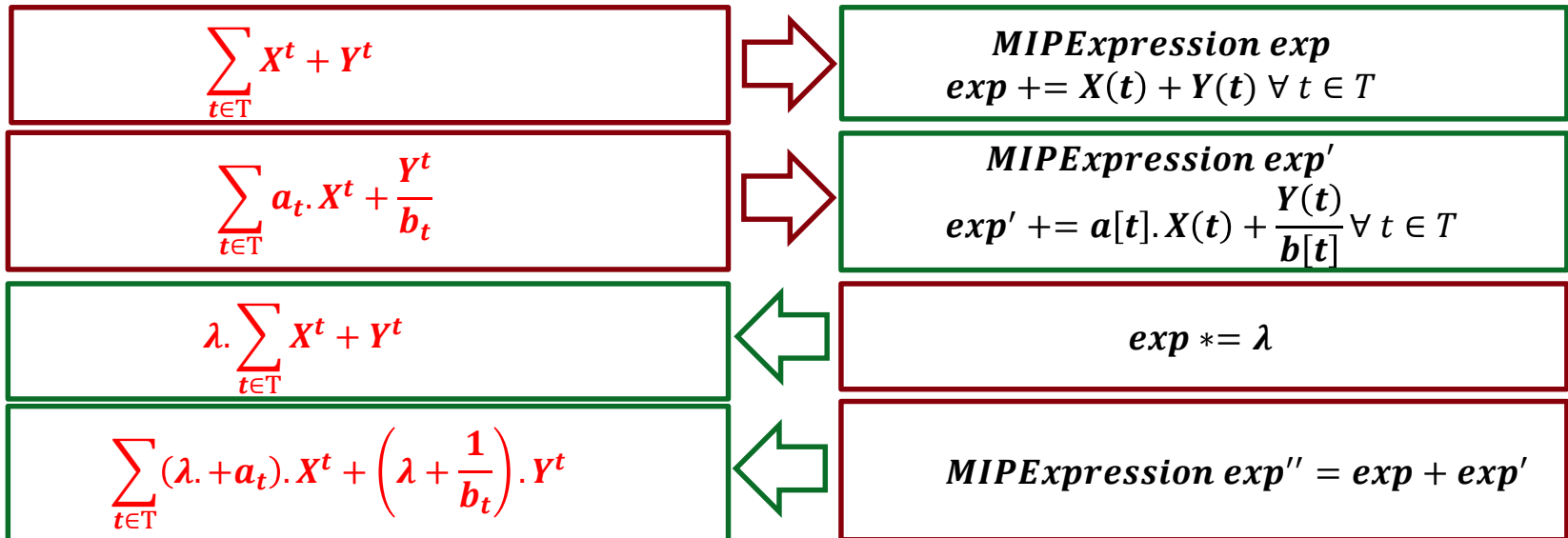
- **Classe *MIPExpression***

- Une expression est une combinaison linéaire entre les variables de décision du problème d'optimisation.
- Les opérateurs (+, -, *, /, +=, -=, *=, /=) sont définis dans *MIPModeler* afin de réaliser toutes les opérations possibles sur des variables, des expression, et/ou des constantes

- **Méthodes utiles**

- ***evaluate(.)***: évaluer l'expression après la résolution du problème MILP
- ***close()***: effacer le contenu de *MIPExpression*

- **Quelques exemples**



LES CONTRAINTES

- **Classe *MIPConstraint***

- L'écriture des contraintes avec *MIPModeler* n'a pas de règle précise i.e., pas d'obligation de respecter le format standard:

Expression* { \leq , $=$, \geq } ***Constante**

- **Quelques exemples**

- ***Constante* { \leq , $=$, \geq } ***Expression*****
- ***Expression* { \leq , $=$, \geq } ***constante*****
- ***Variable* { \leq , $=$, \geq } ***Constant*****
- ***Constant* { \leq , $=$, \geq } ***Variable*****
- ***Expression* { \leq , $=$, \geq } ***Variable*****
- ***Variable* { \leq , $=$, \geq } ***Expression*****
- ***Expression* { \leq , $=$, \geq } ***Expression'*****
- ***Variable* { \leq , $=$, \geq } ***Variable'*****

$$a_t.X^t \leq M.(1 - Y^t) \forall t \in T$$



model.add ($a[t].X(t) \leq M.(1 - Y(t)) \forall t \in T$)

SOLUTION INITIALE POUR UN DÉMARRAGE À CHAUD

- **Classe *MIPWarmStart***

- Définition d'une solution initiale (ou une partie de la solution) pour un démarrage à chaud lors de la résolution. La solution initiale permet de définir une borne supérieure sur le problème d'optimisation (diminution de l'espace de recherche et le nombre de nœuds dans le *Branch-and-Bound*: accélérer la convergence vers l'optimum).

- **Méthodes utiles**

- ***add(Variable, Value)***: initialiser une variable d'optimisation à une valeur
- ***close()***: effacer le contenu de *MIPWarmStart*

- **Exemple d'utilisation (Problème de sac à dos « *NP-Hard* »):**

```

MIPVariable1D select(N, 0, 1, MIP_INT)
model.add(select)

model.add( Sum( i in N,      select(i) * weight(i) ) ≤ Capacity)
model.setObjective( Sum( i in N,      select(i) * price(i)), MIP_MAXIMIZE)

MIPWarmStart sol
sol.add(select(i),      val(i)) ∀ i ∈ N' ⊆ N
model.add(sol)
sol.close()
    
```


- **Classe *MIPModel***

- **Name:** nom du modèle (option)

- **Méthodes utiles**

- ***add(MIPVariableND, Name):*** ajouter une variable d'optimisation au modèle. Il est aussi possible de définir son nom (en option) pour faciliter le débogage
 - ***add(MIPConstraint, Name):*** ajouter une contraintes au modèle. Il est aussi possible de définir son nom (en option) pour faciliter le débogage
 - ***add(MIPSpecialOrderedSet):*** ajouter les variables d'optimisation de type SOS au modèle
 - ***add(MIPWarmStart):*** ajout d'une ou plusieurs solutions initiales pour un démarrage à chaud lors de la résolution
 - ***isMIP():*** vérifier si le problème d'optimisation est un *MILP*
 - ***setObjective(MIPExpression, MIPDirection):*** définir la fonction objective écrite sous forme d'une expression (par défaut expression nulle). La direction de l'optimisation peut être soit *MIP_MAXIMIZE* ou *MIP_MINIMIZE* (par défaut *MIP_MINIMIZE*)
 - ***setObjectiveDirection(MIPDirection):*** Modifier la direction de l'optimisation soit *MIP_MAXIMIZE* ou *MIP_MINIMIZE* (par défaut *MIP_MINIMIZE*)

- MIPModeler permet de déboguer les problèmes d'optimisation en renseignant le nom des variables et des contraintes lors de l'ajout au modèle via:
 - la génération d'un fichier *LP/MPS* lisible et compréhensible
 - le ciblage par nom, les variables ou les contraintes infaisable
- Exemple de génération d'un fichier *LP* (Problème *RO* de transport)**

$$\begin{aligned} \min \quad & \sum_{i \in Usine} \sum_{j \in Client} C_{ij} \cdot transp^{ij} \\ \sum_{j \in Client} transp^{ij} & \leq capacity_i \quad \forall i \in Usine \\ \sum_{i \in Usine} transp^{ij} & \geq demand_j \quad \forall j \in Client \\ transp^{ij} & \geq 0 \end{aligned}$$

MIPModeler: Génération
du LP par défaut

MIPModeler: Génération LP
avec l'option nommage

ENCODING=ISO-8859-1
Problem name: problem

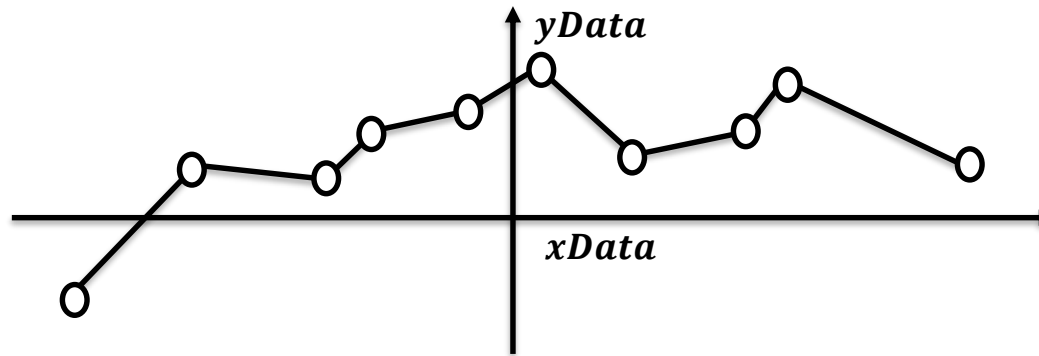
Minimize
obj: 0.225 x1 + 0.153 x2 + 0.162 x3 +
0.225 x4 + 0.162 x5 + 0.126 x6
Subject To
c1: x1 + x4 >= 325
c2: x2 + x5 >= 300
c3: x3 + x6 >= 275
c4: x1 + x2 + x3 <= 350
c5: x4 + x5 + x6 <= 600
End

ENCODING=ISO-8859-1
Problem name: Transport

Minimize
obj: 0.225 transp(0,0) + 0.153 transp(0,1) + 0.162 transp(0,2)
+ 0.225 transp(1,0) + 0.162 transp(1,1) + 0.126 transp(1,2)
Subject To
demand: transp(0,0) + transp(1,0) >= 325
demand: transp(0,1) + transp(1,1) >= 300
demand: transp(0,2) + transp(1,2) >= 275
capacity: transp(0,0) + transp(0,1) + transp(0,2) <= 350
capacity: transp(1,0) + transp(1,1) + transp(1,2) <= 600
End

BIBLIOTHÈQUE DE FONCTIONS GÉNÉRIQUES

- ***MIPPiecewiseLinearisation***: cette fonction est un sous problème *MILP* pour la linéarisation par morceaux des courbes 2D définies par un ensemble de N points $(xData_i, yData_i)$ avec $i \in N$

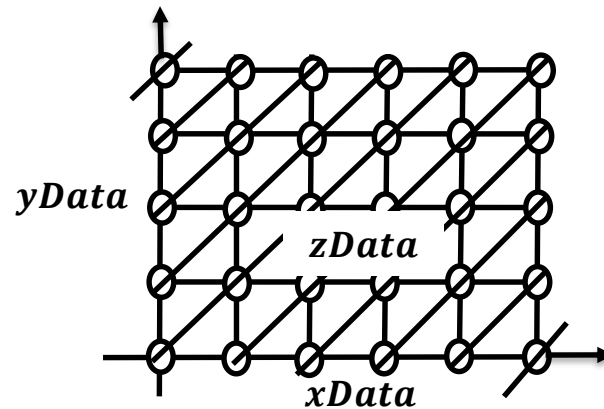


$Y = \text{MIPPiecewiseLinearisation}(\text{model}, X, xData, yData, \text{MIP_Linear}, \text{relaxedForm})$

- ***Model***: ajout de la linéarisation par morceaux au modèle
- ***MIP_Linear***:
 - ***MIP_SOS* (par défaut)**: formulation basée sur des variables SOS2 avec une complexité $o(N)$ variables binaires
 - ***MIP_LOG***: formulation innovante avec une complexité $o(\log(N))$ variables binaires
- ***RelaxedForm* (true / false)**: formulation sans variables binaires en cas de convexité de la courbe 2D

BIBLIOTHÈQUE DE FONCTIONS GÉNÉRIQUES

- ***MIPTriMeshLinearisation***: cette fonction est un sous problème *MILP* pour la linéarisation, par maillage triangulaire, des plans 3D définis par un ensemble de N points $(xData_i, yData_j, zData_{ij})$ avec $i \in I$ et $j \in J$

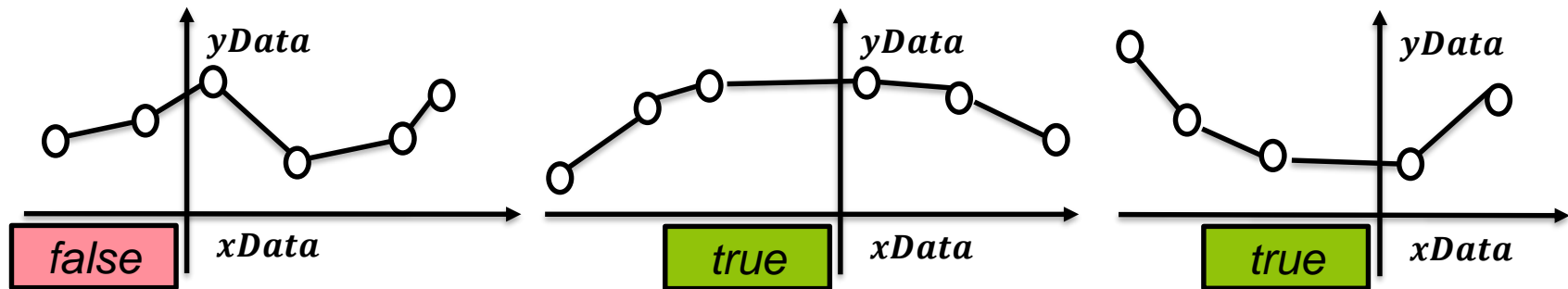


Z = MIPTriMeshLinearisation(model, X, Y, xData, yData, zData, MIP_Linear, relaxedForm)

- ***Model***: ajout de la linéarisation par maillage triangulaire au modèle
- ***MIP_Linear***:
 - ***MIP_SOS*** (par défaut): formulation basée sur des variables SOS2 avec une complexité $o(I)+o(J)+o(I+J)$ variables binaires
 - ***MIP_LOG***: formulation innovante avec une complexité $o(\log(I)+\log(J)+\log(I+J))$ variables binaires
- ***RelaxedForm (true / false)***: formulation sans variables binaires en cas de convexité du plan 3D

BIBLIOTHÈQUE DE FONCTIONS GÉNÉRIQUES

- **isConvexSet:** cette fonction permet de vérifier la convexité d'une courbe 2D ou d'un plan 3D afin de relaxer la linéarisation par morceaux (courbe 2D) ou la linéarisation par maillage triangulaire (plan 3D)



bool isConvexSet(xData, yData)

bool isConvexSet(xData, yData, zData) en cours

- Autres fonctions de linéarisation *MILP*: **en cours**
 - *Abs(expression)*
 - *Max(ensemble d'expression)*
 - *Min(ensemble d'expression)*
 - *Produit de deux variables (continue & binaire)*

INTERFACE AVEC LES SOLVEURS *MILP*

- *MIPModeler* s'interface avec tous les solveurs *MILP* existants
- **Classe *MIPXXSolver(MIPModeler::MIPModel)***
 - **Classe *MIPCpxSolver***: Interface *MIPModeler* avec *CPLEX*
 - **Classe *MIPGrbSolver***: Interface *MIPModeler* avec *GUROBI*
 - **Classe *MIPXprSolver***: Interface *MIPModeler* avec *FICO XPRESS*
 - **Classe *MIPMskSolver***: Interface *MIPModeler* avec *MOSEK*
 - **Classe *MIPCbcSolver***: Interface *MIPModeler* *CBC*
 - **Classe *MIPClpSolver***: Interface *MIPModeler* *CLP*
 - Etc...
- **Méthodes utiles**
 - ***setTimeLimit(.)***: paramétrage du critère d'arrêt du solveur en temps ()
 - ***setGap(.)***: paramétrage du critère d'arrêt en *Gap*
 - ***setThreads(.)***: paramétrage du nombre de *Threads* utilisés
 - ***writeLp()***: écriture du fichier *LP*
 - ***setSolverPrint(int)***: paramétrage de l'affichage des informations solveur lors de la résolution: (afficher tous les détails, afficher que les étapes importantes, etc.)
 - ***solve()***: résolution du problème d'optimisation

EXEMPLE DE MODÉLISATION AVEC MIPMODELER: STOCKAGE & PV

Data: T , dt , $SoeMin$, $SoeMax$, $SoeInit$, $rendement$, $SoeData$, $MaxPchData$, $MaxPdisData$

Modélisation Stockage avec MIPModeler:

MIPModel model

MIPVariable1D Pch (T , 0, *MIP_INFINITY*)

MIPVariable1D Pdis (T , 0, *MIP_INFINITY*)

MIPVariable1D Soe (T , 0, $SoeMax$)

model.add(Pch, "Storage_PCH")

model.add(Pdis, "Storage_PDIS")

model.add(Soe, "Storage_SOE")

MIPExpression1D MaxPch = *MIPPiecewiseLinearisation*(model, Pch, MaxPchData, SoeData)

MIPExpression1D MaxPdis = *MIPPiecewiseLinearisation*(model, Pdis, MaxPdisData, SoeData)

model.add($SOE(t) = SOE(t-1) - rend * Pdis(t) * dt + Pch(t) * dt / rend$, "StateOfCharge") forall t in $T \setminus \{0\}$

model.add($SOE(t) = SOEInit - rend * Pdis(t) * dt + Pch(t) * dt / rend$, "StateOfCharge") for $t = 0$

model.add($Pch(t) \leq MaxPch[t]$, "Storage_PCHLimitation") forall t in T

model.add($Pdis(t) \leq MaxPdis[t]$, "Storage_PDISLimitation") forall t in T

Data: T , MaxPowerForecasting

Modélisation PV avec MIPModeler:

MIPModel model

MIPVariable1D Power (T , 0, *MIP_INFINITY*)

model.add(Power, "PV_Power")

Power(t).upper(maxPowerForecasting[t]) forall t in T

- Enrichir la bibliothèque de fonctions génériques *MILP* (abs, max, min, produit de variables continue & binaire)
- *MIPModeler* (*MIP for Mixed Integer Programming*): future extension pour la modélisation des problèmes *MINLP* (*Mixed Integer Non Linear Programming*) et interfaçage avec les solveurs *MINLP* (*BONMIN, COUENNE, DICOPT, BARON, MINLP, LINGO, SCIP, KNITRO, etc.*)

Merci de me remonter vos suggestions & commentaires permettant de faciliter vos développements sous *MIPModeler* (lors de développement de nouveaux composants sous PERSEE par exemple)

- A titre exemple: Suite à la remarque d'Etienne & Alain pour le débogage de certains modèles d'optimisation → la fonctionnalité débogage a été intégrée à MIPModeler