Installation et utilisation de *Cplex* avec Python

en local et sur les serveurs de calcul

Hélène Toussaint, septembre 2024

Table des matières

1 Installation de l'environnement de travail sur son PC

1.1 Installation de *Cplex*

Les étudiants et chercheurs peuvent bénéficier d'une licence gratuite du logiciel IBM ILOG CPLEX Optimization Studio grâce au programme *IBM Academic Initiative*. La marche à suivre pour le télécharger est la suivante :

- se rendre sur le site https://www.ibm.com/academic/

- cliquer sur le bouton *Acces Software download*

- se connecter (ou créer un compte)

- une fois connecté, aller sur la page https://academic.ibm.com/a2mt/downloads#/ et cliquer sur la rubrique « *Data Science* » puis sur ILOG CPLEX Optimization Studio. Plusieurs versions de *Cplex* sont disponibles, choisir la version souhaitée, par exemple : IBM ILOG CPLEX Optimization Studio V22.1.1 Multiplatform Multilingual eAssembly et choisir dans la liste déroulante la plateforme souhaitée : IBM ILOG CPLEX Optimization Studio V22.1.1 for Windows x86-64. Il suffit ensuite de suivre les instructions pour lancer l'installation de *Cplex*.

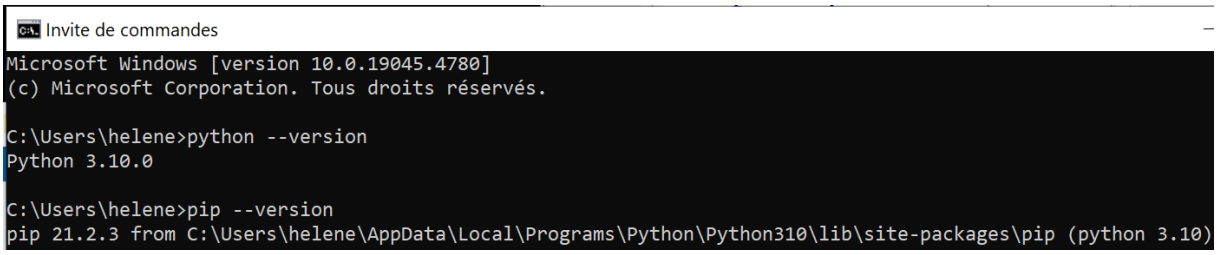
***NB**. D'une année sur l'autre les liens peuvent changer, si les liens donnés ici ne fonctionnent plus, faire une recherche sur internet avec les mots clés « IBM Academic Initiative ».*

Dans la suite de ce document, nous supposons que c'est la version 22.1.1 de *Cplex* qui est installée et que le répertoire d'installation est celui par défaut.

1.2 Installation de Python

On donne ici la marche à suivre sous Windows (facilement adaptable à Mac ou Linux).

1. Vérifier que Python et pip sont bien installés sur l'ordinateur : ouvrir l'invite de commandes (taper **`cmd`** dans la barre de recherche Windows) et exécuter les commandes **`python --version`** et **`pip --version`**

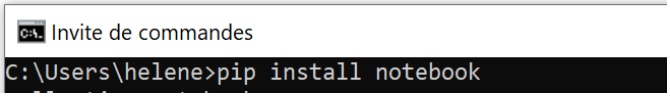


Attention, à ce jour (17 septembre 2024), les versions de Python supérieures à 3.10 ne sont pas compatibles avec la dernière version de *Cplex*.

Si le numéro de version s'affiche alors Python est correctement installé. Dans le cas contraire (ou si la version est supérieure à 3.10), télécharger la version souhaitée depuis le site officiel : https://www.python.org/downloads/.

NB. Cocher l'option « *Add Python to PATH* » lors de l'installation pour faciliter l'utilisation de pip.

2. Installer un éditeur de texte pour Python (par exemple Visual Studio Code https://code.visualstudio.com/ ou Jupyter Notebook).
   Dans ce document nous utiliserons Jupyter Notebook.
   Pour l'installer, taper la commande **`pip install notebook`** dans l'invite de commandes.



3. Exécuter la commande **`jupyter notebook`** dans l'invite de commandes : l'interface de Jupyter Notebook s'ouvre dans le navigateur. La liste des fichiers et répertoires du répertoire courant s'affichent. Pour créer un nouveau notebook Python, cliquer sur le bouton New en haut à droite de cette liste et sélectionner Python 3.

4. Vérifier la version de Python utilisée et le répertoire courant à l'aide du code ci-dessous

```
import os
import sys

# Obtenir le répertoire courant
current_directory = os.getcwd()
print(f"Répertoire courant : {current_directory}")

# Obtenir la version de Python
python_version = sys.version
print(f"Version de Python : {python_version}")
```

Dans Jupyter Notebook, le code est exécuté en cliquant sur la flèche entourée en rouge (voir Figure 1).

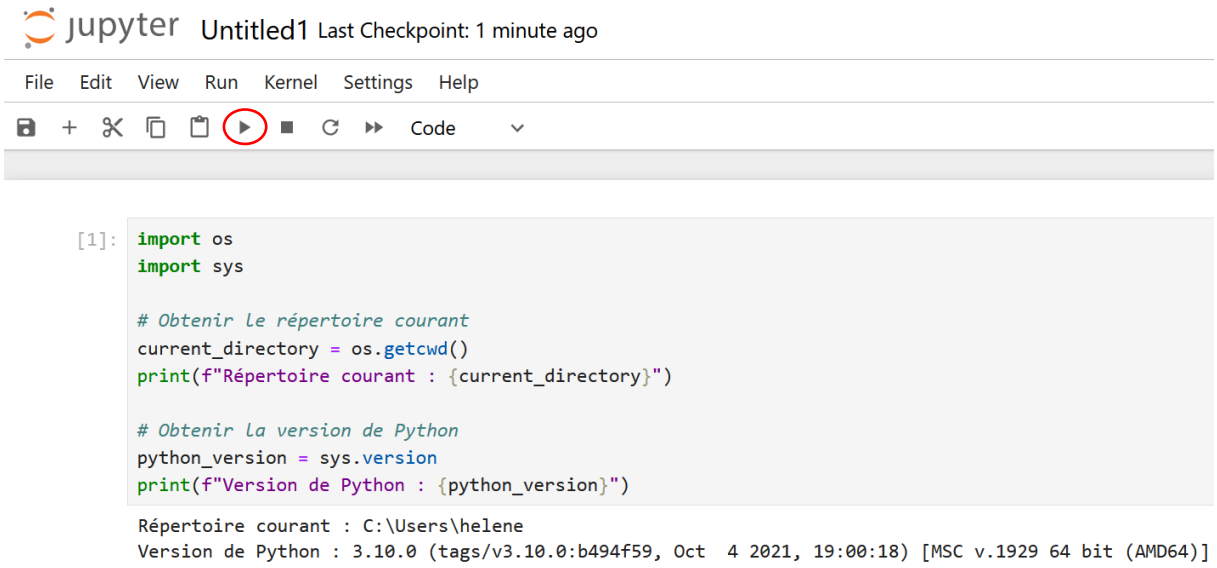


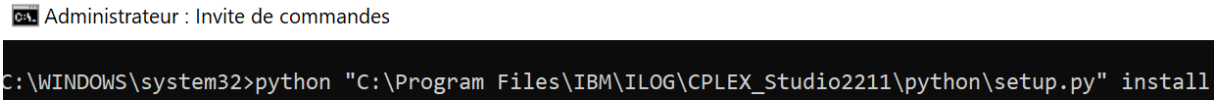
Figure 1. Obtenir la version de Python et le répertoire courant depuis Jupyter Notebook

1.3 Lier Python et Cplex

Pour pouvoir utiliser les moteurs CPLEX ou CP Optimizer à travers leur API Python, il faut indiquer à Python où les trouver ([1]). Pour ce faire :

- ouvrir une invite de commandes en mode administrateur (dans le menu déroulant Windows faire un clic droit sur « Invite de commandes » et choisir « Exécuter en tant qu'administrateur »).
- taper la commande :

```
python "C:\Program Files\IBM\ILOG\CPLEX_Studio2211\python\setup.py" install
```



L'environnement de travail est prêt.

2 Ecrire et résoudre un programme linéaire depuis un script Python

2.1 Documentation en ligne et exemples fournis avec Cplex

Pour se familiariser avec l'API Python pour Cplex, il y a un tutoriel de prise en main rapide disponible en ligne sur le site d'IBM ([2]). Il est également possible de découvrir les utilisations de l'API Python au travers d'exemples : de nombreux exemples de scripts Python sont disponibles dans le répertoire d'installation de Cplex :

C:\Program Files\IBM\ILOG\CPLEX_Studio2211\cplex\examples\src\python par défaut.

2.2 Exemple d'implémentation et résolution d'un programme linéaire

Nous donnons ici un petit exemple pour tester la bonne installation des bibliothèques.

$$\begin{aligned} &\text{Minimiser } 3x_1 + 2x_2 \\ &\text{sous } x_1 + x_2 \geq 3 \\ &\quad 2x_1 + x_2 \geq 2 \end{aligned}$$

Le code pour implémenter ce modèle est le suivant (la Figure 3 montre le résultat d'exécution) :

```
import cplex

# ==== 1. CREATION DU MODELE =====
model = cplex.Cplex()
model.objective.set_sense(model.objective.sense.minimize)

# ==== 2. AJOUT DES VARIABLES DE DECISION ====
# nom des variables, coefficients dans la fonction objectif et borne inf
names = ["x1", "x2"]
objective = [3, 2]
```

```
lower_bounds = [0, 0]
model.variables.add(obj=objective, lb=lower_bounds, names=names)

# ==== 3. AJOUT DES CONTRAINTES ====
# 3.1. on commence par définir les coefficients
constraints = [
    [["x1", "x2"], [1.0, 1.0]],  # x1 + x2
    [["x1", "x2"], [2.0, 1.0]],  # 2x1 + x2
]
# 3.2. puis le second membre
rhs = [3.0, 2.0]
# 3.3 et  le sens de la contrainte (ici >= )
constraint_senses = ["G", "G"]

# 3.4 enfin on ajoute les contraintes au modèle
model.linear_constraints.add(lin_expr=constraints, senses=constraint_senses,
rhs=rhs)

# ==== 4. résolution et affichage de la solution ====
model.solve()

# 4.1. Afficher le résultat
print("Valeur optimale de l'objectif :", model.solution.get_objective_value())

# 4.2. Afficher les valeurs optimales des variables de décision
solution_values = model.solution.get_values()
for var_name, value in zip(names, solution_values):
    print(f"Valeur optimale de {var_name} : {value}")
```

3 Utilisation de l'API Python pour *Cplex* sur les serveurs de calcul du LIMOS

Cplex 22.1.1 et Python 3.8.10 sont déjà installés sur les serveurs, ainsi que le package cplex pour Python. Il n'y a donc rien à faire au niveau de l'installation logicielle : tout est déjà en place et prêt à fonctionner.

Il suffit alors de produire un fichier .py (script Python) à partir du code écrit sur Jupyter Notebook. Pour cela dans l'onglet File de Jupyter Notebook, cliquer sur « Save and export Notebook as » puis « executable script ». Un fichier .py est créé et enregistré par défaut dans le répertoire « Téléchargement ».

Copier ce fichier dans votre espace de travail sur le cluster (l'utilisation du cluster est documenté sur [3]) et l'exécuter avec la commande **`python <nomDuFichier>.py`**

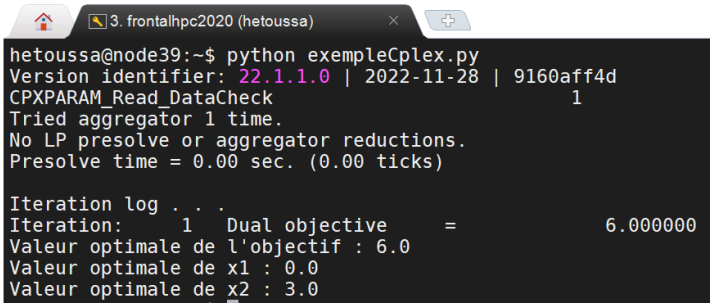


Figure 2. Exécution d'un script Python qui appelle Cplex (utilisation d'un shell interactif sur le cluster du LIMOS)

```python
import cplex

# ==== 1. CREATION DU MODELE =====
model = cplex.Cplex()
model.objective.set_sense(model.objective.sense.minimize)

# ==== 2. AJOUT DES VARIABLES DE DECISION ====
# nom des variables, coefficients dans la fonction objectif et borne inf
names = ["x1", "x2"]
objective = [3, 2]
lower_bounds = [0, 0]
model.variables.add(obj=objective, lb=lower_bounds, names=names)

# ==== 3. AJOUT DES CONTRAINTES ====
# 3.1. on commence par définir les coefficients
constraints = [
    [["x1", "x2"], [1.0, 1.0]],  # x1 + x2
    [["x1", "x2"], [2.0, 1.0]],  # 2x1 + x2
]
# 3.2. puis le second membre
rhs = [3.0, 2.0]
# 3.3 et  le sens de la contrainte (ici >= )
constraint_senses = ["G", "G"]

# 3.4 enfin on ajoute les contraintes au modèle
model.linear_constraints.add(lin_expr=constraints, senses=constraint_senses, rhs=rhs)

# ==== 4. résolution et affichage de la solution ====
model.solve()

# 4.1. Afficher le résultat
print("Valeur optimale de l'objectif :", model.solution.get_objective_value())

# 4.2. Afficher les valeurs optimales des variables de décision
solution_values = model.solution.get_values()
for var_name, value in zip(names, solution_values):
    print(f"Valeur optimale de {var_name} : {value}")
```

```
Version identifier: 22.1.1.0 | 2022-11-27 | 9160aff4d
CPXPARAM_Read_DataCheck                         1
Tried aggregator 1 time.
No LP presolve or aggregator reductions.
Presolve time = 0.00 sec. (0.00 ticks)

Iteration log . . .
Iteration:     1   Dual objective     =             6.000000
Valeur optimale de l'objectif : 6.0
Valeur optimale de x1 : 0.0
Valeur optimale de x2 : 3.0
```

Figure 3. Exemple d'implémentation et d'exécution d'un programme linéaire dans Jupyter Notebook

4 Quelques fonctions utiles pour débugger le modèle

On suppose dans les exemples de code de cette section que model est un modèle au sens de l'API Python pour Cplex et qu'il a déjà été créé via l'instruction model = cplex.Cplex() (voir l'exemple de code de la section 2).

4.1 Afficher le modèle

Avant même de lancer l'optimisation il est conseillé de s'assurer que le modèle programmé est bien celui qu'on croit. Pour cela il faut l'afficher :

- exporter le modèle dans un fichier texte à l'aide la fonction write de l'API
- lire et afficher le fichier avec Python

```
# écrit le modèle au format .lp dans le fichier texte model.lp
model.write("model.lp")

# lit et affiche le fichier créé
with open("model.lp", "r") as file:
    model_content = file.read()
print(model_content)
```

4.2 Afficher des informations sur la solution

On accède à la solution via le champ solution de l'objet model. Plusieurs fonctions peuvent être appliquées à la solution. On peut notamment récupérer son statut (optimale, non faisable, non bornée, etc…).

La fonction get_status_string permet de récupérer cette information au format texte :

```
# récupère et affiche le statut de la solution
status_string = model.solution.get_status_string()
print(f"Statut de la solution : {status_string}")
```

On peut également afficher les coûts réduits et les valeurs des variables duales.

Dans les exemples de code suivant, variables et constraints sont les objets contenant les variables et les contraintes du modèle tels que définis dans l'exemple de code de la section 2.

```
# récupère et affiche les coûts réduits
reduced_costs = model.solution.get_reduced_costs()
for var, cost in zip(variables, reduced_costs):
    print(f"Coût réduit de {var}: {cost}")
```

```
# récupère et affiche les valeurs des variables duales
dual_values = model.solution.get_dual_values()
for constraint, dual in zip(constraints, dual_values):
    print(f"Valeur duale de {constraint}: {dual}")
```

Beaucoup d'autres informations peuvent être obtenues à partir de la solution. Dans Jupyter Notebook, on peut utiliser l'auto-complétion (touche *Tabulation*) pour voir les fonctions qui s'appliquent à model.solution.

4.3 Rediriger les messages d'erreur ou warning vers la sortie standard

Dans Jupyter Notebook les messages d'erreur générés par Cplex peuvent ne pas s'afficher par défaut. Pour s'assurer qu'ils sont bien affichés, il faut explicitement les rediriger vers la sortie standard (sys.stdout) :

```
import sys
import cplex

# Créer un modèle CPLEX
model = cplex.Cplex()

# Rediriger les flux d'erreur, d'avertissement et de log vers sys.stdout
model.set_error_stream(sys.stdout)
model.set_warning_stream(sys.stdout)
model.set_log_stream(sys.stdout)
```

5 Références

[1] IBM Documentation. Setting up the Python API of CPLEX. https://www.ibm.com/docs/en/icos/22.1.1?topic=cplex-setting-up-python-api

[2] IBM Documentation. Python tutorial. https://www.ibm.com/docs/en/icos/22.1.1?topic=tutorials-python-tutorial

[3] Hélène Toussaint. Plateforme HPC LIMOS. https://hpc.isima.fr/