Université Paris Nanterre

Licence MIAGE

Mémoire Stage Licence 3 MIAGE CLASSIQUE

Implémentation d'algorithmes pour les modèles de jeu stochastiques

Stage réalisé du 23 Mars 2020 au 1 Juin 2020

Avi ASSAYAG

Table des matières

Remerciements

Merci à Monsieur Hyon, maitre de conférence à l'Université de Nanterre et chercheur dans l'equipe SYSDEF du Lip6 , d'avoir accepter le poste de tuteur pour mon stage de Licence 3 MIAGE. Grâce a son accompagnement personnel j'ai pu solidifier mes compétences algorithmiques (Java et Python) mais aussi découvert d'autre aspect de la programmation linéaire.

Cette opportunité n'a été seulement possible que par la collaboration de Monsieur Emmanuel Hyon, mon tuteur ainsi que Monsieur François Delbot, responsable de la Licence 3, et les remercie de leur patience , de leur encouragement et de le encadrement tout au long de ce stage.

1 Introduction

Pendant ces semaines de stage, nous allons essayez d'implémenter des algorithmes pour résoudre des modèles de jeux stochastiques, plus précisément des jeux de gain à somme nul (que nous représenterons sous forme bimatricel).

L'objectif est dans un premier temps de concevoir une modélisation informatique de ces jeux puis dans un second temps implémenter un ou des algorithmes permettant de résoudre ces jeux.

Pour parfaire à ces attentes, nous allons utilisé le langage *Python*, non utilisé durant le cursus scolaire actuel, le solveur *Gurobi*, que nous utiliserons afin de résoudre des programmes linéaires et pour l'orienté objet *Python* et la plateforme *GitHub*, l'hébergeur de code, pour avoir accès a tout les codes sources, document qui m'ont aidé a réaliser ce stage.

L'utilisation de GitHub n'était pas obligatoire, mais elle plus que judicieuse pour que mon tuteur *Mr Emmanuel Hyon* puisse avoir accès en temps réel à mon code afin de m'orienter si je m'écarte du sujet. C'est donc à son initiative que nous avons utilisé *Github* tout au long de ce stage.

Dans les sections qui vont suivre nous allons expliciter différents concepts propre aux modèles stochastiques (notamment le principe même de la théorie des jeux) mais aussi les outils utilisés ; comment les installer et les utiliser.

2 GitHub

Github est un service d'hébergement web (un peu comme une sorte de Drive) et de gestion de développement de logiciel lancé en 2008. Ce dernier est codé principalement en Ruby et Erlang par différents programmeurs : Chris Wanstrath, PJ Hyett et Tom Preston-Werner.

Aujourd'hui cette plateforme compte plus de 15 millions d'utilisateurs et enregistre environ 40 millions de dépôts de fichiers, se plaçant donc en tête du plus grand hébergeur source code mondial.

Le fonctionnement de Git est assez simple, on créer un répertoire (un référentiel / requisitory) dans lequel on va stocker tout les fichiers que l'on désire et on peut soit rendre l'accès publique (au quel cas tout le monde peut rejoindre et consulter ces fichiers) ou alors le restreindre en accès privé (au quel cas c'est le créateur qui décide quels seront les collaborateurs ayant droit de consultation des fichiers).

Ensuite cela s'agit comme une sorte de réseau constitué de branches (branch) où chaque branches représentent un collaborateur ainsi que la master qui correspond au créateur du référentiel.

Une des caractéristiques de Git repose sur le fait que c'est un outil de verisonning (gestion de version) est donc permet de si le fichier à été modifié ; si oui par qui et quand a eu lieu la modification et quel fichiers ont été affectés. Cela permet notamment de pouvoir faire des travails de groupe sur le même sujet (un site ou une application par exemple) où chacun doit travaillé sa partie mais nécessite les parties des autres membres du groupes (mis à jours régulièrement).

Évidemment toutes les étapes (initialisation, dépôts, fusion et clonage) se font a l'aide de lignes de commandes sur le terminal (en bash) que j'expliquerai un peu plus loin ainsi que les commandes principales pour chaque étapes.

2.1 Initialisation

Pour créer un projet il suffit d'aller sur le site https ://github.com/ puis Repositories –> New et remplir les informations données avant de valider. Ensuite pour initialiser le Git (et que la branch master existe ; elle sera crée automatiquement a l'instantaciation du projet) il faut se placer dans le dossier (en ligne de commande cd) et tappez : git init

Ensuite il faudra taper : git remote add origin < lien donnée par git hub > puis git push –u origin master qui respectivement créerons le répertoire du projet et ensuite la zone de dépôt.

2.2 Branches

Comme nous l'avons dit plus haut le projet est contenu dans la branche principale la master et grâce à des copies de branches le projet acquiert une plus grande flexibilité qui permet d'incrémenter au fur et mesure le projet.

Pour ajouter une branche il suffira simplement de taper git branch < nom de la branche > et pour supprimer une branche il faut rajouter l'option -d a la commande soit : git branch -d < nom de la branche >.

Pour changer de branche (afin d'effectuer un dépôt ou autre) il faudra taper : git checkout < nom de la branche > et enfin pour visualiser l'ensemble des branches existantes on devra taper : git branch .

2.3 Dépôt et mises à jours

Avant toute chose il faut savoir sur quelle branche déposer le fichier puis il faudra taper les commandes suivantes pour les ajouter au fichier : git add < nom des fichier > (ou * pour tout ajouter) puis git commit -m message et enfin pour finir git push origin < nom de la branche>.

Pour récupérer des modifications faites sur le projet il suffit à l'inverse de taper : git pull origin master.

2.4 Clonage

Une fois les autres branches (celles des différents collaborateurs) crées il faut juste qu'il copie le lien du git pour pouvoir travailler dessus et effectuer les futurs dépôts. En premier lieu il faudra taper : git clone < lien du git > puis effectuer la commande git pull origin master pour récupérer les fichiers de la branche master et enfin faire les commandes relatives au dépôt (vu plus haut).

3 Python

Python est un langage de programmation à part entière dont la première version fut développe par *Guido van Rossum* et lancé en 1991. Ce langage est facile d'utilisation et ne possède pas forcément de syntaxe particulières seulement une indentation permettant au compilateur intégré de suivre les blocs d'instructions.

Ce langage permet donc une multitude de possibilité de code mais aussi d'action puisqu'il existe des bibliothèques déjà implémentés et il suffira seulement des les utiliser comme bon nous le semble (par exemple Matplotlib ou encore Networkx etc...). Malheureusement Python n'est pas le langage le plus rapide d'exécution contrairement au C ou C++ et Java mais il permet tout de même d'accéder à des fonctionnalités que d'autres langages ne peuvent proposer.

Contrairement au C, Python admets des types sophistiqués supplémentaires tel que les *Listes*, les *Dictionnaires*, les *Sets* et les *Tuples*. Il en va de soit que les types primitifs sont aussi présent *int,float,double,boolean etc...* Mais le réel avantage du langage repose sur le fait que l'on ne se soucie pas du type de retour d'une fonction ni de la déclaration du type du paramètre ainsi que le langage admet la possibilité d'être orienté objet.

Python est un langage interprété et donc n'a pas besoin de passer pas un compilateur comme GCC (GNU Complier Collection), tout se fait directement sur la console une fois l'environnement installé.

Quant à l'installation de Python, cette dernière est assez simple ; il suffit d'aller sur le site officiel et télécharger la version en question (aujourd'hui version 3.8.2) et ensuite de l'installer. Il existe différentes méthodes d'activation du langage, qui représente chacune l'environnement de la machine (Windows, Mac OS ou encore Linux).

A savoir que sur Mac Os et Linux, Python est déjà préinstallé et il faudra peut être seulement mettre à jours la version qui pourrai être obsolète ou dépassé.

3.1 Installation de Python

3.1.1 Méthode packages

Pour cela il faut allez télécharger les packages en question sur le site officiel de Python puis les interpréter c'est dire ouvrir la console (terminal python) et demander à python d'exécuter le fichier *.py* en question via la commande :

```
python setup.py install
```

3.1.2 Méthode module Pip

Il s'agit d'une des méthodes les plus simple, après avoir téléchargé les packages Python sur le site, on installe tout les modules externes (pip , Django etc ...) que l'on pourrai avoir besoin d'utiliser par la suite via le terminal :

```
pip install <nom_module>
```

3.2 Un petit code Python

Pour déclarer une variable il suffit seulement de la nommé, Python n'attend pas forcement le type de la variable ; tout comme pour une fonction il n'attend pas le type de retour de la fonction.Ensuite pour les boucles et les conditions il suffit d'utiliser le mot clé en question suivit de " : " et donnez les instructions de façon indenter

Rien de mieux qu'un petit code Python pour mieux comprendre la syntaxe et la facilité d'utilisation du langage. Ainsi je vais vous présenter un code source de la fonction *Tri à bulles* :

```
def tri_a_bulle(tab) :
    taille=len(tab)
    for i in range(taille) :
           for j in range(taille-1) :
                 if tab[j] > tab[j+1] :
                     tab[j] , tab[j+1] = tab[j+1] , tab[j]
    return tab
```

3.3 Python et les objets

Python permet aussi l'utilisation de l'orienté objet, c'est donc un des autres plus de ce langage puissant et aux vagues possibilité. Dans cette partie nous allons vous montrer comment coder un objet en Python et aussi le construire. Nous allons donc voir la syntaxe générale d'une *Classe* et celle d'un *Constructeur*. Enfin pour terminer cela nous implémenterons un objet "Pullover" avec différents attributs et son propre constructeur.

3.3.1 Création d'une Classe "Lambda"

Pour déclarer un objet il suffit simplement d'utiliser le mot Class suivit de " :" et ensuite déclarer des variables ou autres instructions.

```
Class Personne :
     name
     age
```

3.3.2 Constructeur de la Classe "Lambda"

Pour déclarer le constructeur d'un objet il faut utiliser la méthode *init()* au sein de la classe en passant en paramètre ceux de l'objet en question. La petite différence par rapport à d'autres langages de programmation orienté objet (C++ ou Java) est l'utilisation du paramètre (mais aussi mot clé) *self* au seins du constructeur. En réalité *self* n'est autre que la première référence de l'instance de l'objet que l'on va créer.

```
def __init__(self, name, age) :
     self.name = name
     self.age = age
```

3.3.3 Exemple d'objet : Pullover

Maintenant que nous avons une première approche de la syntaxe objet essayons de mettre cela en application avec quelque chose de plus concret. Nous allons créer un "Pullover" avec comme attribut : une marque, une taille, un nom de modèle, une couleur et un prix

```
Class Pullover :
     brand
     size
     model_name
     color
     price

     def __init (brand, size, model_,name, color, price) :       #constructeur
           self.brand=brand
           self.size=size
           self.model_name=model_name
           self.color=color
           self.price=price

  Pull1 = Pullover("ZARA", "XS", "AED934", "black", 19)         #instanciation
```

4 Gurobi

La plateforme Gurobi est un solveur mathématique autrement dit c'est une optimisation mathématique. Il traduit un problème commercial en un énoncé mathématique. Gurobi à été ecrit pour prendre en considération différentes interfaces sous différents langage : *C ,C++, Java, Python et R*.

Il y a deux méthodes d'installation soit directement avec une licence (payante ou gratuite) ou alors avec un la distribution *Anaconda* que nous allons tenter d'expliquer.

4.1 Installation de Gurobi

4.1.1 Méthode classique

Il est aussi possible d'installer *Gurobi* directement sur la machine en gardant notre environnement configuré par nos propre soins puisque l'environnement Python a pensé cela lors de sa conception.

Pour cela il suffira d'installer *gurobipy* via le terminal Python en tappant la commande suivante :

```
python setup.py install
```

4.1.2 Méthode via Anaconda

Via cette méthode, l'environnement est déjà préinstaller pour l'utilisateur et comporte une interface graphique *Spyder* ainsi qu'un éditeur de texte *Jupiter* .Pour cela il suffira simplement de télécharger les fichiers nécessaires sur https ://www.gurobi.com/get-anaconda/ puis lancer *Anaconda* via le terminal et enfin installer le package de *Gurobi*.

```
python | Anaconda
conda install gurobi
```

4.2 Test d'installation

4.3 Exemple Gurobi programmation linéaire (lp)

```
import sys
import gurobipy as gp
from gurobipy import GRB

if len(sys.argv) < 2:
    print('Usage: lp.py filename')
    quit()

# Read and solve model
model = gp.read(sys.argv[1])
model.optimize()
```

```
if model.status == GRB.INF_OR_UNBD:
    # Turn presolve off to determine whether model is infeasible
    # or unbounded
    model.setParam(GRB.Param.Presolve, 0)
    model.optimize()

if model.status == GRB.OPTIMAL:
    print('Optimal objective: %g' % model.objVal)
    model.write('model.sol')
    sys.exit(0)
elif model.status != GRB.INFEASIBLE:
    print('Optimization was stopped with status %d' % model.status)
    sys.exit(0)

# Model is infeasible - compute an Irreducible Inconsistent Subsystem (IIS)
print('Model is infeasible')
model.computeIIS()
model.write("model.ilp")
print("IIS written to file 'model.ilp'")
```

5 La théorie des jeux

Comme nous l'avons expliqué un peu plus haut, l'un des objectifs de ce stage est la modélisation d'algorithmes afin de résoudre des jeux stochastiques. Mais tout d'abord détaillons un peu le concept des *jeux*.

Définition 1 *Un jeu est une analyse des interactions stratégiques, intégrant des contraintes (si elles existent), sur les actions des différents acteurs (joueurs) au cour du jeu.*

Pour intégrer et comprendre ce concept, il y a d'autres notions à connaitre telles que *jeux statiques* , *jeux dynamiques* , *stratégie* ou encore *jeux bimatricel* et enfin *gains à somme nulle*.

5.1 Jeux statiques et dynamiques

Comme vous l'avez compris,un jeu nécessite la présence d'acteurs ; dans la suite de nos explications lorsque nous parlerons de *joueurs* nous ferons donc référence aux acteurs du jeu.

5.1.1 Statiques

Lorsque l'on parle de jeu statique il s'agit en réalité d'un jeu ou chaque joueur effectue une seule action en simultané de l'autre mais sans avoir accès aux informations de l'action de l'autre joueurs.

5.1.2 Dynamiques

A l'inverse, un jeu dynamique est un jeu qui se déroule en plusieurs étapes et non en simultané ; c'est-à-dire que chacun des joueurs a connaissance de

l'action de l'autre et donc peut établir une stratégie avant chaque futures actions.

5.2 Jeu bimatriciel

Un jeu bimatriciel se caractérise comme son nom l'indique par deux matrices. Ces dernières ne sont autres que les gains des joueurs. Autrement dit les joueurs jouent de manière simultané et on inscrit dans une matrice leurs gains (une matrice pour chaque joueur). Voici un exemple de deux matrices de gains pour deux joueurs A et B qui joue respectivement les lignes et les colonnes.

$$Joueur\ A \qquad \begin{pmatrix} 3 & 3 \\ 2 & 5 \\ 0 & 6 \end{pmatrix} \qquad\qquad Joueur\ B \qquad \begin{pmatrix} 3 & 2 \\ 2 & 6 \\ 3 & 1 \end{pmatrix}$$

5.3 Jeu à sommes nulles

Comme annoncé dans notre introduction nous essayerons de résoudre des jeux a sommes nulles via des algorithmes que nous allons implémenter par la suite. Mais qu'est ce qu'un jeu à somme nul ?

eifheien

6 Webographie

Références

[Python] https://docs.python.org/fr

[GitHub] https://help.github.com/en

[Gurobi] https://www.gurobi.com/documentation/9.0/quickstart_mac/py_python_interface.html

[Théorie des Jeux] https://fr.wikipedia.org/wiki/Th%C3%A9orie_des_jeux

[Théorie des Jeux] http://www.cril.univ-artois.fr/~konieczny/enseignement/TheorieDesJeux.pdf

[Théorie des Jeux] http://www.cril.univ-artois.fr/~konieczny/enseignement/TheorieDesJeux.pdf

7 Annexes

7.1 Syntaxe Python

La syntaxe est assez similaire aux autre langage puisque python utilise les mêmes types de variables, sauf les types sophistiqués. A la différence des autres langages de programmation (C,,C++,Java,php) la fin d'une instruction se termine par un caractère vide et non ; , avec python c'est l'indentation qui fait office d'instruction et donc de bloc de code.

7.1.1 Structure Conditionelle If

La condition est suivi par : puis vient ensuite l'instruction à effectuer, si le test est vérifié, qu'il faudra indenter (d'un cran).

```
if <condition> :
      <instruction>
```

7.1.2 Structure Conditionelle Else

La condition est suivi par : puis vient ensuite l'instruction à effectuer, si le premier test n'est pas vérifié, qu'il faudra indenter (d'un cran) au même niveau que le test If.

```
if <condition1> :
      <instruction1>
else :
      <instruction2>
```

7.1.3 Structure Conditionelle Elif

La condition est suivi par : puis vient ensuite l'instruction à effectuer, si le premier test n'est pas vérifié, qu'il faudra indenter (d'un cran) au même niveau que le test If.

```
if <condition1> :
      <instruction1>
elif <condition2> :
      <instruction2>
else :
      <instruction3>
```

7.1.4 Boucle For

La structure est composé de for puis de deux valeurs élément et sequence qui permette de suivre l'itération à effectuer. Le bloc est exécuté autant de fois de qu'il y a d' éléments dans la sequence et se termine par : .

```
for element in sequence :
     <instruction>
```

7.1.5 Boucle While

La structure est composé de while puis de la condition qui permet d'effectuer un test. Le bloc est exécuté tant que la condition est vérifié et se termine par :

```
while <condition> :
     <instruction>
```

7.1.6 Les fonctions

Quant au fonction la définition se fait de manière très simple il suffit d'utiliser le mot clé def et cela est terminer, en python on ne prend pas en compte le type de retour d'une fonction comme en C, C++ ou en Java (int, void, double, float etc ...).

```
def onction (param1 , param2) :
    <instruction1>
    <instruction2>
        if <test1> :
            <instruction3>
        else :
            <instruction4>
    return  <instruction5>
```

7.2 L'orienté objet en Python

Python est un langage résolument orienté objet, ce qui signifie que le langage tout entier est construit autour de la notion d'objets. Quasiment tous les types du langage String / Integer / Listes / Dictionnaires sont avant tout des objets tout comme les fonctions qui elles aussi sont des objets.

Pour créer une classe , donc un Objet il suffit d'utilise le mot clé class suivit de : et ne pas oublier l'indentation.

```
class < NomClasse> :
     attribut1
     attribut2
```

Ensuite il faudra définir un constructeur qui permettra d'instancier les objets dont nous auront besoins, il faut donc utiliser la méthode *init* au sein de la classe sans oublier le paramètre obligatoire (mot clé de python) self.

```
class < NomClasse> :
    attribut1
    attribut2

    def __init__ (self):
        self.attribut1 = ... (str)
        self.attribut2 =  ...  (int)
```

Si l'on défini une classe vide c'est a dire ou pour le moment il n'y aucune action à effectuer il faut rajouter le mot clé pass

```
class < NomClasse > :
    pass
```

Comme nous l'avons également vu ont une classe mère peut hérité d'une autre et donc de ses attributs et de ses méthodes. la syntaxe est simple, il suffit de mettre en paranthése la classe mère au moment de la déclaration de la classe fille. Voici un exemple avec < NomClasse > et < NomClasse2>

```
class < NomClasse> :                          #classe mère
    attribut1
    attribut2


class < NomClasse2> (< NomClasse >) :         #classe fille
    attribut1                                 #hérité
    attribut2                                 #hérité
    attribut3
    attribut4
```

A ce niveau on peut se demander comment Python gére ces héritages. Lorsqu'on tente d'afficher le contenu d'un attribut de données ou d'appeler une méthode depuis un objet, Python va commencer par chercher si la variable ou la fonction correspondantes se trouvent dans la classe qui a créé l'objet.

Si c'est le cas, il va les utiliser. Si ce n'est pas le cas, il va chercher dans la classe mère de la classe de l'objet si cette classe possède une classe mère. Si il trouve ce qu'il cherche, il utilisera cette variable ou fonction.

Si il ne trouve pas, il cherchera dans la classe mère de la classe mère si elle existe et ainsi de suite. Deux fonctions existent pour savoir si l'objet est seulemnent une instance d'une classe et pour savoir si la classe en question a eu recourt à de l'hériatge : isinstance() et issubclass().

7.3 GitHub

En résumé les commandes principales de Github sont :
— git init
— git remote add
— git clone
— git checkout
— git branch < branche > à l'inverse git branch -d < branche >
— git add < fichier > ou alors git add * (pour tous les fichiers)
— git commit -m ...
— git push origin master ou bien git push origin < branche >
— git pull origin master ou bien git pull origin < master >
— git merge

7.4 Gurobi

Table des figures