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

1 Environnement Utilisés

Durant ce stage, dans le but d'implémenter des algorithmes pour modèles de jeu stochastiques nous allons utiliser divers outils (environnements) et langages informatiques.

Parmi eux se trouvent le langage Python, la plateforme GitHub ainsi que le solveur Gurobi que nous utiliserons pour la programmation linéaire et le python orienté objet.

1.1 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.

1.1.1 Installation sous forme de 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
```

1.1.2 Installation via le 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
```

1.1.3 Un petit code Python

Rien de mieux qu'un petit code Python pour mieux comprendre la syntaxe et la facilité d'utilisation du langage. Voici donc une fonction *Puissance récurssive* et une fonction *Tri à bulles* :

```
def puissance_rec(x, n):
    if not n:
          return 1
    if n == 1:
          return x
    return x * puissance_rec(x, n - 1)


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
```

1.2 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.

1.2.1 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.

1.2.2 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 .

1.2.3 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.

1.2.4 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).

1.3 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.

1.3.1 Installation classique de Gurobi

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
```

1.3.2 Installation 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
```

2 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

3 Annexes

3.1 Python

3.1.1 Syntaxe

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.

Nous avons déjà vu plus haut la syntaxe des types dit sophistiqués donc je ne m'attarderai pas plus dessus, et nous allons donc voir la syntaxe des structures conditionnels (If / Else / Elif) et celle des boucles (For / While).

A) 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>
```

B) 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>
```

C) 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>
elif <condition2> :
      <instruction2>
else :
      <instruction3>
```

D) 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>
```

E) 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>
```

F) 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>
```

3.1.2 Orienté Objet (POO

Comme nous l'avons expliqué un peu plus haut les langages de programmation peuvent avoir deux voies statiques comme le C et Python par exemple ou alors orienté objet comme le Java, C++ ainsi que le Python.

Les objectifs principaux de la programmation orientée objet sont de nous permettre de créer des scripts plus clairs, mieux structurés, plus modulables et plus faciles à maintenir.

L'orienté objet (comme dans tout langage) repose sur quatre grandes notions :

- Le concept d'objet
- Le principe d'encapsulation
- Le polymorphisme
- L'héritage

A) Le concept orienté objet
En POO (programmation orienté objet) nous allons concevoir notre programme (script) non pas comme plusieurs fonction ayant un but complémentaire mais plutôt comme un ensemble d'objet (définit via des classes) interagissant les uns avec les autres. Un objet est un concept qui représente un ensemble de données et qui en contrôle l'accès où chaque objet a un comportement propre. Au sein d'un objets on y retrouve les membres qui le compose : ses attributs (structures de données définies dans l'objet) et ses méthodes (procédures et fonctions définies dans l'objet).

Ainsi pour répondre au besoin d'abstraction d'objet apparait une nouvelle notion celle des classes, c'est une structure (un model) dans laquelle on déclare l'ensemble des membres (attributs et méthodes) de l'objet en question. La deuxième notion qui apparait alors est donc la création / construction de cet objet ; nous utilisons donc un constructeur (et respectivement un destructeur). Ainsi le constructeur permet de créer des instances dont les caractéristiques (les membres) sont décris par la classe.

B) L'encapsultaion
Ce concept désigne le fait de dissimuler certaines informations contenue dans un objet et de proposer (à l'utilisateur) la modification que de certains membres. Il faudra spécifier des membres publiques (visible de l'objet) et privés (non visible de l'objet).

C) Le polymorphisme
Ce concept permet de redéfinir (le nom et corps) une méthode au sein d'une classe et donc de la spécifier. Mais aussi la possibilité de définir des comportements différents pour la même méthode selon les arguments donnés en paramètres. Pour résumer, le polymorphisme est un concept qui fait référence à la capacité d'une variable, d'une fonction ou d'un objet à prendre plusieurs formes, c'est-à-dire à sa capacité de posséder plusieurs définitions différentes.

D) L'héritage Ce concept permet à une classe d'hériter (des membres donc des attributs et des méthodes) d'une autre. On parlera alors de la classe mère (celle dont on hérite) et des classes filles (celles qui héritent). Ce n'est pas un principe unitaire il peut s'appliquer plusieurs fois et suivre un schéma on parlera alors d'héritage multiple.

3.2 Le POO 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)
```

Expliquons un peu le mot clé self qui est toujours présent soir dans une fonction soit dans une définition de classe. Ainsi les fonctions d'une classe ne font pas exception : ce sont également objets. C'est donc pour cela que la fonction possédera toujours un paramètre de plus,le fameux self. 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().

3.3 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

3.4 Gurobi

Table des figures