

Licence MIASHS parcours MIAGE

## Rapport de stage

L3 MIAGE, parcours classique

# Implémentation d'algorithmes pour modèles de jeu stochastiques

Entreprise d'accueil : Université Paris Nanterre  
Stage réalisé du 23 mars 2020 au 22 mai 2020

*présenté et soutenu par*

**Avi ASSAYAG**

le 25 mai 2020

### Jury de la soutenance

M. François Delbot,  
M. François Delbot,  
M. Emmanuel Hyon ,

Maître de conférences  
Maître de conférences  
Maître de conférences

Responsable du L3 MIAGE  
Tuteur enseignant  
Maître de stage

## Remerciements

Merci à Monsieur Hyon, maitre de conférence à l'Université de Nanterre et chercheur dans l'équipe 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.

Merci aussi aux autres professeurs qui ont contribué tout a long de l'année à parfaire toutes nos compétences autant sur le plan théorique que techniques.

Enfin je tiens aussi à remercier mes relecteurs qui ont permis de rendre ce mémoire aussi précis et grammaticalement correct , mais aussi d'être un document accessible et utile à tous si un jour vous décider de vous lancer dans le même sujet que moi.

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>5</b>
<b>2</b>	<b>Contexte du Stage</b>	<b>6</b>
2.1	Présentation de l'entreprise . . . . .	6
2.2	Présentation du service et de l'équipe . . . . .	7
2.3	Mission proposée . . . . .	8
2.4	Cahier des charges . . . . .	8
<b>3</b>	<b>Outils utilisés</b>	<b>9</b>
3.1	GitHub . . . . .	9
3.1.1	Initialisation . . . . .	9
3.1.2	Branches . . . . .	9
3.1.3	Dépôt et mises à jours . . . . .	10
3.1.4	Clonage . . . . .	10
3.2	Python . . . . .	10
3.2.1	Procédure d'installations de Python . . . . .	11
3.2.2	Un exemple générique de code Python . . . . .	12
3.2.3	La programmation objet en Python . . . . .	12
3.3	Gurobi . . . . .	14
3.3.1	Installation de Gurobi . . . . .	14
3.3.2	Utilisation de gurobi . . . . .	15
3.3.3	Exemple de programmation linéaire avec Gurobi . . . . .	15
<b>4</b>	<b>Théorie des jeux</b>	<b>17</b>
4.1	Jeux statiques . . . . .	17
4.1.1	Principes et modèles des jeux statiques . . . . .	17
4.1.2	Jeux bimatriceiel . . . . .	18
4.1.3	Jeux à somme nulle . . . . .	20
4.2	Jeux bimatriceiel a sommes nulles et résolution par PL . . . . .	20
4.2.1	Un exemple de jeu : "Matching pennies" . . . . .	21
4.2.2	Matrice de gains des joueurs A et B . . . . .	22
4.2.3	Résolution via programmation linéaire . . . . .	22
4.2.4	L'équilibre de Nash dans Matching Pennies . . . . .	23
4.3	Jeux Stochastiques . . . . .	23
4.3.1	Jeux dynamique . . . . .	23
4.3.2	Fonction de valeur . . . . .	23
4.3.3	Modèles et principes de jeux stochastiques . . . . .	24
<b>5</b>	<b>Conclusions</b>	<b>26</b>
5.1	Contributions apportées . . . . .	26
5.2	Difficultés rencontrées . . . . .	26
5.3	Axes d'amélioration . . . . .	26
<b>6</b>	<b>Webographie</b>	<b>27</b>

<b>7</b>	<b>Annexes</b>	<b>28</b>
7.1	Nanterre . . . . .	28
7.2	Python en général . . . . .	29
7.2.1	Structure Conditionnelle If . . . . .	29
7.2.2	Structure Conditionnelle Else . . . . .	29
7.2.3	Structure Conditionnelle Elif . . . . .	29
7.2.4	Boucle For . . . . .	29
7.2.5	Boucle While . . . . .	29
7.2.6	Les fonctions . . . . .	30
7.3	L'orienté objet en Python . . . . .	30
7.3.1	Création de class . . . . .	30
7.3.2	Création du constructeur . . . . .	30
7.3.3	Le mot clé pass . . . . .	30
7.3.4	L'héritage en python . . . . .	31
7.4	GitHub . . . . .	31
7.5	Gurobi . . . . .	32
7.6	CV . . . . .	33

# 1 Introduction

Pendant ces semaines de stage, nous allons essayer 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 utiliser 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 à tout les codes sources, document qui m'ont aidé à réaliser ce stage.

L'utilisation de **GitHub** n'était pas obligatoire, mais elle était plus que judicieuse afin 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 chapitres qui suivront nous allons expliciter différents concepts relatifs aux **modèles stochastiques** (notamment le principe même de la théorie des jeux) mais aussi les outils utilisés ; comment les installer et les utiliser.

Enfin nous tenterons de rédiger et de résoudre différents modèles de **jeu à sommes nulles**, c'est à dire un jeu où le gain d'un des acteurs représente la perte exacte des autres acteurs de ce jeu, mais nous l'expliquerons en détails dans une prochaine section [4.3.1].

Les **jeux à sommes nulles** sont nombreux vous connaissez surement le **jeu pierre-feuille-ciseau**, mais aussi le **dilemme du prisonnier**, ou encore le **dilemme du voyageur** et tant d'autres.

Dans les semaines de ce stage nous avons décider d'implémenter un **jeu à somme nulle** moins connu que les précédents mais tout aussi intéressant il s'agit : du jeu "**Matching Pennies**"

Enfin nous étudierons plus en profondeurs qu'est ce que sont les **jeux stochastiques** mais surtout comment définir un modèle informatique (en **python**) et sa résolution grâce à la programmation linéaire (via **gurobi**).

## 2 Contexte du Stage

Durant le seconde semestre de la troisième année de notre licence Miage (S6), nous sommes tenus d'effectuer un stage en entreprise d'une durée minimum de 10 semaines. Pour cela j'ai donc recherché activement depuis le mois de février une entreprise prête à m'accueillir en stage afin de me former et de compléter mes connaissances. En attendant des réponses positives, j'avais quand même envisagé d'effectuer une demande (avec l'accord de mon professeur référant **Monsieur Delbot** au près de l'un de enseignants de l'université en cas de refus de candidatures potentielles.

Avec la crise sanitaire que nous traversons et en l'absence de réponse positive la direction de la formation a fait tout son possible pour que le stage se passe dans les meilleurs conditions possible (tant sur le plan sanitaire que sur le plan logistique) et ainsi j'ai pu obtenir un sujet de stage, dirigé par mon professeur **Monsieur Hyon** au seins de la faculté.

### 2.1 Présentation de l'entreprise

L'université Paris Nanterre, de nom officiel **Paris X** est une université française fondé le 20 Octobre 1964 et spécialisé dans les sciences économiques et sociales, le droit, la psychologie et l'informatique. Elle se situe à Nanterre (92000) , à deux pas de l'un des centre économique de Paris , La Défense.

Le campus s'étend sur plus de 32 hectares et compte plusieurs restaurants (dans le restaurant universitaire Croos), de nombreux espaces verts , un terrain d'athlétismes, des cours de tennis et une piscine. On y dénombre environs 35 000 étudiants (sans compter ceux inscrit à distance) repartis dans 8 UFR (Unité de Formation et de Recherche) et 5 instituts, pour un un total d'environs 2 200 enseignants chercheurs (respectivement 1800 et 400).



FIGURE 1 – Le campus de Nanterre

Comme nous l'avons expliqué l'Université est organisé autours de 8 UFR :

- UFR Langues et cultures étrangères (LCE)
- UFR Littérature, langages, philosophie et arts du spectacle (PHILLIA)
- UFR Droit et science politique (DSP)
- UFR Sciences psychologiques et sciences de l'éducation (SPSE)
- UFR Sciences sociales et administratives (SSA)
- UFR Sciences et techniques des activités physiques et sportives (STAPS)
- UFR Systèmes industriels et techniques de communication (SITEC)
- UFR Sciences économiques, gestion, mathématiques et informatique (SEGMI)

Évidemment il subsiste une hiérarchie au seins l'Université, à sa tête on y trouve Monsieur **Jean-François Balaudé** (depuis 2012) puis chaque UFR à un président et chaque parcours universitaire (mathématiques, informatique, droits, économie, gestion etc...) a un responsable par niveau (licence et master) et par voie d'études (classique ou apprentissage).

## 2.2 Présentation du service et de l'équipe

Effectuant mon stage à l'université de Nanterre et étant étudiant en troisième année de licence MIAHS parcours MIAGE, il était logique d'effectuer mon stage au seins de l'UFR SEGMI (Sciences économiques, gestion, mathématiques et informatique).

A sa direction on trouve Monsieur **Yann Demichel**, enseignant chercheur mathématique au seins de l'université de Nanterre (Paris X) puis différents responsables par formation. Concernant le parcours MIAGE, nous avons Monsieur **Jean-François Pradat-Peyre** enseignant chercheur et responsable du département mathématique et informatique puis respectivement Monsieur **Pascal Poizat** et Madame **Marie Pierre Gervais** responsable du master MIAGE classique et alternance. Quant aux licences elles sont gérer respectivement par Monsieur **François Delbot** et Madame **Sonia Saadaoui** responsable de la licence MIAGE classique et alternance.

Quant à mon tuteur Monsieur **Emmanuel Hyon** il est enseignant chercheur mais aussi maître de conférence à l'université de Nanterre (Paris X) depuis 2004 et nous à donner cours d'algorithme et programmation C lors du premier semestre de la licence MIAGE.

Travaillant principalement au LIP6 (Département Desir ) en recherche opérationnel (RO) et sur modélisation mathématique et informatique à l'université de Nanterre (Paris X) il parrait logique de choisir comme thème de stage : **Implémentation d'algorithmes pour modèles de jeux stochastiques**.

Au vue de la situation peu évidente et inhabituelle, et dans le but de préserver un maximum notre santé j'ai effectué mon stage à domicile donc sans la présence de mon tuteur, ce qui aurai rendu le stage encore plus intéressant et plus évident à gérer (d'un point de vue logistique et pour des questions potentielles). Pour remédier à cela et à l'initiative de Monsieur **Hyon** mon tuteur, nous avons mis en place un lien **GitHub** mais aussi un rendez vous téléphonique hebdomadaire en début de semaine permettant de faire le point, de poser mes questions, de me donnez les axes que je devais améliorer ainsi que les points sur lesquels je devais travailler pour la semaine à venir.

## 2.3 Mission proposée

Ainsi il m'a été confié de modéliser un jeu stochastique sur un point de vue informatique (via `python`) mais surtout de trouver un moyen, par un algorithme de le résoudre c'est à dire de trouver la solution la plus optimale (nous reviendrons plus en détails dans les sections futures cf [4.2], [4.3]).

Pour cela j'ai du apprendre les bases, la syntaxe et le fonctionnement du langage `python` mais aussi me familiariser avec le concept orienté objet (déjà étudié durant mon cursus avec `C++` et `Java`) sur un nouveau langage : `python`.

De plus j'ai du apprendre utiliser un solveur mathématique `gurobi` afin de trouver la solution optimale, autrement maximiser la fonction objective (représentant la solution du jeu en question cf [RO]).

Ainsi tout ça à eu pour but de me permettre de modéliser et résoudre tout d'abord le jeu de **Matching Pennies**, un jeu bimatriciel à somme nulle puis dans un second temps la modélisation et la résolution d'un jeu bilatéral.

Aimant les mathématiques et l'informatique depuis jeune le sujet de ce stage me correspond parfaitement et comblait mon envie de poursuivre mes études avec un master MIAE.

## 2.4 Cahier des charges

Il m'est un peu compliqué de donner des dates exactes et précise de chaque tâches à réaliser ou effectués mais je peux énumérer la liste des charges à accomplir :

- Mise en place d'un lien **GitHub** (pour avoir accès en temps réel au codes sources mais aussi au rapport)
- Installation de `python` (version 3.8)
- Apprentissage du langage `python` et des propriétés orienté objet (POO)
- Installation du solveur `gurobi`
- Apprentissage de la syntaxe `gurobi` et du module `gurobi` pour la modélisation des jeux (de tout type)
- S'informer sur les concepts fondamentaux de la théorie de jeux
- Définir les notions de jeu bimatriciel, jeu stochastique, stratégie, gain
- Approche mathématique de l'équilibre de Nash
- Création d'un objet "Jouet" en `python`
- Modéliser et résoudre un programme linéaire (`biere.py`)
- Modéliser et résoudre le jeu **Matching Pennies**
- Modéliser et résoudre un jeu **stochastique bilatéral**
- Rédaction du mémoire pour la soutenance de stage (semaine du 25 mai)

Ainsi pendant 10 semaines et ce depuis le 23 mars, je travaille quotidiennement de le but d'implémenter des algorithmes pour modèles de jeux stochastiques.



## 3 Outils utilisés

### 3.1 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 crée un répertoire (un référentiel / repository) dans lequel on va stocker tout les fichiers que l'on désire et on peut soit rendre l'accès public (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 versionning (gestion de version) est donc permet de savoir si le fichier a été modifié ; si oui par qui et quand a eu lieu la modification et quels fichiers ont été affectés. Cela permet notamment de pouvoir faire des travaux de groupe sur le même sujet (un site ou une application par exemple) où chacun doit travailler 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 à 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.

#### 3.1.1 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 créée automatiquement à l'instanciation du projet) il faut se placer dans le dossier (en ligne de commande cd) et taper : git init

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

#### 3.1.2 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` .

### 3.1.3 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`.

### 3.1.4 Clonage

Une fois les autres branches (celles des différents collaborateurs) créées 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).

## 3.2 Python

Python est un langage de programmation à part entière dont la première version fut développée par **Guido van Rossum** et lancée 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 implémentées 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 admet 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 par un compilateur comme GCC (GNU Compiler 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.

### 3.2.1 Procédure d'installations de Python

#### 3.2.1.1 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
```

#### 3.2.1.2 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>
```

#### 3.2.1.3 Environnement virtuel python

On parle d'environnement virtuel pour certaines utilisation bien propre à **python**. Il arrive parfois que certaines versions ne permettent pas d'utiliser certains module, ou alors il existe des "bugs" dans la version.

**Définition 1** *Un environnement virtuel est un environnement d'exécution isolé (en mode coopératif) qui permet aux utilisateurs de Python et aux applications d'installer et de mettre à jour des paquets sans interférer avec d'autres applications Python fonctionnant sur le même système.*

Ainsi la création d'un environnement virtuel permet d'avoir les bibliothèques nécessaire ainsi que les modules adéquats pour la réalisation d'un projet ou d'une application. Pour cela il faut créer l'environnement **venv.venv** ,dans le dossier que l'on souhaite via le terminal et enfin l'activer.

```
#Création d'un environnement  
python3 -m venv tutorial-env
```

```
#Activation sous Windows  
tutorial-env\Scripts\activate.bat
```

```
#Activation sous Unix et Mac  
source tutorial-env/bin/activate
```

### 3.2.2 Un exemple générique de code Python

Pour déclarer une variable il suffit seulement de la nommer, Python n'attend pas forcément 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 suivi 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
```

### 3.2.3 La programmation objet en Python

Python permet aussi l'utilisation de l'orienté objet, c'est donc un des autres plus de ce langage puissant et aux vagues possibilités. 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.

#### 3.2.3.1 Code générique Classe "Lambda"

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

```
Class Personne :  
    name  
    age
```

#### 3.2.3.2 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 sein 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
```

### 3.2.3.3 Exemple de class : 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
```

### 3.3 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 a été écrit 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 la distribution **Anaconda** que nous allons tenter d'expliquer.

Travaillant sur MacOS, j'ai opter pour l'installation de **Gurobi** en privé sur ma machine et donc en gérant l'installation sur mon environnement Python, et utiliserait donc le module *gurobipy*. De plus il m'a fallu créer un compte chez **Gurobi** pour utiliser une licence académique gratuite bien évidemment.

#### 3.3.1 Installation de Gurobi

##### 3.3.1.1 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 faudra au préalable télécharger le solveur sur le site web de **Gurobi** (le lien est en annexe) et attendre le téléchargement. Une fois terminé il suffit d'exécuter le fichier télécharger (en double cliquant dessus) pour démarrer l'installation. Durant cette dernière le système d'exploitation nous demandera dans quel dossier stocker les packages nécessaire a **Gurobi**. Ensuite il faudra se rendre à cette emplacement, via un terminal et exécuter la commande suivante :

```
python setup.py install
```

Après avoir créer son compte afin d'obtenir un licence il faudra l'enregistrer sur la machine afin de pouvoir utiliser le solveur sans souci, pour cela il faudra ouvrir le terminal et exécuter la commande suivante :

```
grbgetkey 4fd46a16-7d9c-11ea-809f-020d093b5256
```

Une fois tout ceci effectué et donc paramétré il faudra, pour utiliser le solveur, faire un **import** du module et donc de la bibliothèque **Gurobi** au début du script python que l'on élabore.

```
import gurobipy as gp
from gurobipy import *
```

##### 3.3.1.2 Méthode via Anaconda

Pour essayer de faire simple, Anaconda est une solution libre office, c'est a dire à téléchargement gratuit, qui permet une installation rapide et simple de **Python** avec un interpréteur (IDE) ainsi que de nombreuses bibliothèques (les plus utilisées et les plus utiles), **gurobi** par exemple. Par la suite si on veut ajouter des modules ou bibliothèques supplémentaires on le fait comme avec python sauf qu'au lieu d'écrire **pip** on écrit **conda**.

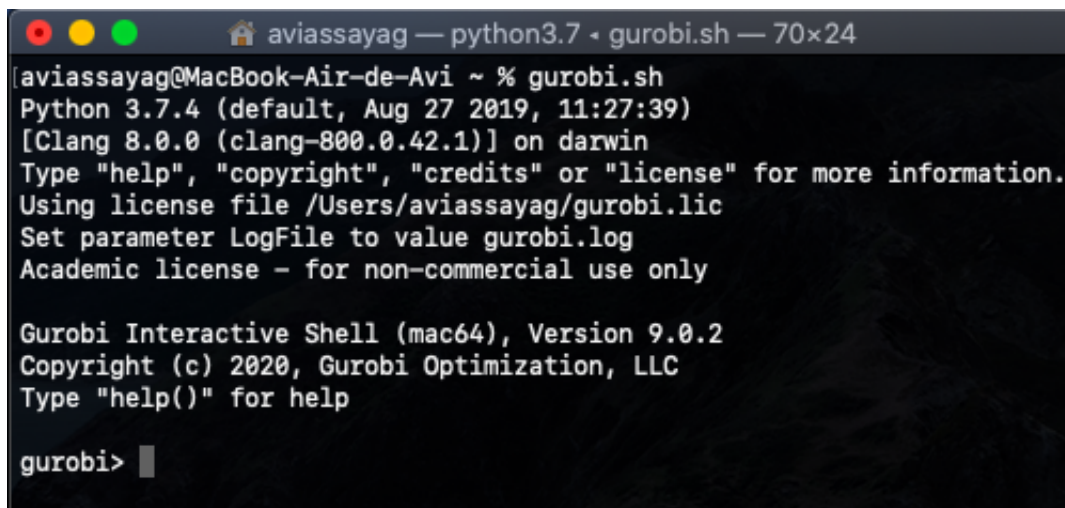
via Python :        `pip install <module>`  
via Anaconda :     `conda install <module>`

Via cette méthode, l'environnement est déjà préinstaller pour l'utilisateur et comporte une interface graphique **Spyder** ainsi qu'un éditeur de texte **Jupyter**. 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
```

### 3.3.2 Utilisation de gurobi

Lorsque que **Gurobi** mentionne son "shell interactive" il s'agit en fait d'un script (fichier ".sh") qui est fournit avec le téléchargement du solveur. En en le lançant, c'est à dire en le tapant à la console le terminal lancera une console **gurobi** ou il faudra directement écrire le code à exécuter. Ainsi un interpréteur **Gurobi** sera ouvert et attendra des instructions, au même titre qu'un interpréteur **Python**.



```
aviassayag@MacBook-Air-de-Avi ~ % gurobi.sh  
Python 3.7.4 (default, Aug 27 2019, 11:27:39)  
[Clang 8.0.0 (clang-800.0.42.1)] on darwin  
Type "help", "copyright", "credits" or "license" for more information.  
Using license file /Users/aviassayag/gurobi.lic  
Set parameter LogFile to value gurobi.log  
Academic license - for non-commercial use only  
  
Gurobi Interactive Shell (mac64), Version 9.0.2  
Copyright (c) 2020, Gurobi Optimization, LLC  
Type "help()" for help  
  
gurobi> █
```

FIGURE 2 – Shell interactive Gurobi

### 3.3.3 Exemple de programmation linéaire avec Gurobi

Soit  $x_1$  et  $x_2$  les quantités (en volume) respectives produite pour les bières  $b_1$  et  $b_2$ . Les quantités sont soumises à des contraintes (3) pour chaque ingrédients utilisé :  
Contraintes :

- Contrainte C1 :  $2,5 x_1 + 7,5 x_2 \leq 240$  (pour le maïs)
- Contrainte C2 :  $0,125 x_1 + 0,125 x_2 \leq 5$  (pour le houblon)
- Contrainte C3 :  $17,5 x_1 + 10 x_2 \leq 595$  (pour le malt)
- Contrainte de positivité :  $x_1, x_2 > 0$

Objectif :

- Maximiser :  $\max 15 x_1 + 25 x_2$

```
#Appel et utilisation de Gurobi et de ses modules
import gurobipy as gp
from gurobipy import *

try :
    # Création du model
    m = gp.Model("Biere")

    # Déclaration des Variables
    x1 = m.addVar(vtype=GRB.INTEGER, name="x1")
    x2 = m.addVar(vtype=GRB.INTEGER, name="x2")

    # Maximisation
    m.setObjective(15*x1 + 25*x2, GRB.MAXIMIZE)

    # Contraintes des Variables
    m.addConstr(2.5 * x1 + 7.5 * x2 <= 240, "c1")
    m.addConstr(0.125 * x1 + 0.125 * x2 <= 5, "c2")
    m.addConstr(17.5 * x1 + 10 * x2 <= 595, "c3")

    # Résoud la solution objective
    m.optimize()

    #Affichage de la réponse
    for v in m.getVars():
        print('%s %d' % (v.varName, v.x))

    print('Obj: %s' % m.objVal)

    #Vérification des exceptions
except gp.GurobiError as e:
    print('Error code ' + str(e.errno) + ': ' + str(e))

except AttributeError:
    print('Encountered an attribute error')
```

Ci dessus le code python utilisé pour permettre a Gurobi de trouver la solution de maximisation, soit : **x1= 12 et x2 = 28**



## 4 Théorie des jeux

### 4.1 Jeux statiques

#### 4.1.1 Principes et modèles des jeux statiques

##### 4.1.1.1 Principe des jeux statiques

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.

Pour intégrer et comprendre ce concept, il y a d'autres notions à connaître telles que jeux statiques, jeux dynamiques, stratégie, concurrent ou encore jeux bimatriciel et enfin gain.

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.

##### 4.1.1.2 Modèle des jeux statiques

**Définition 2** *Un jeu est dit statique lorsque le jeu se déroule en une seule étape et de manière simultanée sans avoir accès aux informations de l'action de ou des autres joueurs.*

Ainsi un jeu statique peut être défini par :

- Un nombre fini  $J$  de joueurs :  $1, 2, \dots, J$
- Pour chaque joueur  $i$ , un ensemble de stratégies  $\Pi^i = \{ \pi^1, \dots, \pi_j \}$
- Pour chaque joueur  $i$  une fonction de valeur  $v_i$  tel que :  $\Pi^i \times \dots \times \Pi^j \rightarrow \mathbf{R}$

**Définition 3** *On appelle stratégie la manière dont un joueur choisit l'action qu'il effectue. C'est un vecteur de taille  $|A_i^1|$  où  $|A_i^1|$  est l'ensemble des actions du joueur  $A_i$ ,  $|A_i|$  est le cardinal de l'ensemble  $A$ . Ce vecteur est noté  $\pi^1$  pour le joueur 1 et  $\pi^2$  pour le joueur 2.*

Lorsque l'on parle de stratégie pure il s'agit d'une stratégie déterministe c'est à dire une stratégie dans laquelle une seule et unique action est effectuée. Elle détermine en particulier l'action qu'un acteur (joueur) réalisera devant toutes les situations auxquelles ce dernier sera confronté.

**Définition 4** *On note par  $\pi^1(i)$  la coordonnée d'indice  $i$  du vecteur  $\pi^1$  représentant le choix d'un joueur, ce choix est déterministe. Par convention :  $\pi^1(i) = 0$  l'action n'est pas choisie et  $\pi^1(i) = 1$  l'action est choisie.*

En parallèle aux stratégies pures, il existe aussi des stratégies dites mixtes c'est à dire où chaque action d'un joueur  $i$  dépend d'une probabilité (connue ou non) provenant de la stratégie pure du même joueur.

**Définition 5** L'action est choisie en fonction d'une probabilité. Le vecteur  $\pi^1$  est maintenant une distribution de probabilité sur les actions du joueur.  $\pi^1(i)$  est la probabilité que l'action  $i$  soit choisie. De plus comme  $\pi^1$  est une probabilité on a obligatoirement  $\sum_i \pi^1(i) = 1$ .

#### 4.1.1.3 Equilibre de Nash

Maintenant que l'on en connaît un peu plus sur le concept fondamental de la **théorie des jeux** nous avons bien compris que résoudre un jeu (dynamique bien évidemment) revient non pas à "gagner" la partie mais trouver la meilleur stratégie permettant de maximiser ses gains et donc a fortiori minimiser ses pertes (puisqu'elles correspondent à gain réciproque des autres acteurs).

**Définition 6** Une action conjointe  $a^*$  est un équilibre de Nash si et seulement si :  $\forall j, \forall a_j R_j(a^*) \geq R_j(a_j, a_{-j}^*)$ .

Autrement dit, trouver un équilibre de Nash dans un jeu revient à trouver la solution optimal de stratégies mixtes (x,y) qui sont meilleurs réponses l'une fonction de l'autre, c'est à dire :

- x est la meilleur réponse dans la stratégie de y (J2) qui maximise la matrice de gain du J1 donc x.
- y est la meilleur réponse dans la stratégie de x (J1) qui maximise la matrice de gain du J1 donc x

**Nash en stratégie pure** Un profil de stratégie est un ensemble de spécifiant pleinement pour tout joueur, toutes les actions dans le jeu. Il comporte une et une seule stratégie pour chaque joueur.

**Définition 7** Soit  $\pi^i$  une stratégie du joueur  $i$  et  $\pi^{-i}$  la stratégie des autres joueurs. Un profil de stratégies  $\pi^* = \pi^i$  est un équilibre de Nash si et seulement si :

$$\forall i, \forall \pi'^i \in \Pi^i, v_i(\pi^i, \pi^{-i}) \geq v_i(\pi'^i, \pi^{-i})$$

**Nash en stratégie mixte** On appelle  $\sigma^i$  une stratégie mixte (ou profil de stratégie mixte) une distribution de probabilités sur les actions. L'ensemble des stratégies mixtes du joueur  $i$  est noté  $S^i$ .

**Définition 8** Soit  $\sigma^*$  un profil de stratégie mixte.  $\sigma^*$  est un équilibre de Nash si, pour tout joueur  $i$  on a :

$$\forall \sigma'^i \in S^i, v_i(\sigma^{*i}, \sigma^{*-i}) \geq v_i(\sigma'^i, \sigma^{*-i})$$

#### 4.1.2 Jeux bimatriceiel

Un jeu bimatriceiel 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 simultanée et on inscrit dans une matrice leurs gains (une matrice pour chaque joueur). Pour la suite du mémoire et du stage nous allons donc utiliser strictement des jeux à deux joueurs JA et JB.

Ainsi un jeu bimatriceiel est défini par :

- Un nombre fini  $J$  de joueurs :  $\{1, 2, \dots, J\}$
- Un nombre fini  $M$  d'actions pour le joueur 1 (JA) :  $\{1, 2, \dots, M\}$
- Un nombre fini  $N$  d'actions pour le joueur 2 (JB) :  $\{1, 2, \dots, N\}$
- Une matrice de gain (pay-off)  $G_1$  pour le joueur JA  $[M \times N]$
- Une matrice de gain (pay-off)  $G_2$  pour le joueur JB  $[N \times M]$
- Une stratégie pure par joueurs composé des actions des joueurs respectifs

#### 4.1.2.1 Exemple de jeu bimatriciel

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

#### 4.1.2.2 Équilibre en stratégie pure

D'après les matrices précédentes le seul équilibre pur (obtenu par le théorème du minimax) est  $\mathbf{x} = \{(1,0,0), (1,0)\}$ .

Pour vérifier cela il semble logique que si JA choisi (1,0,0) soit 3 et JB choisi (1,0) soit 3 il apparait un équilibre pur puisque le gain de JA serait de 3 et JB 3 aussi. C'est bien la seule solution possible pour équilibre.

#### 4.1.2.3 Équilibre en stratégie mixte

On cherche à savoir si il existe un support de la forme  $\mathbf{x}=(x_1, x_2, 0)$  et  $\mathbf{y}=(y_1, y_2)$  correspondant à une stratégie mixte tel que  $(x_1, x_2, y_1, y_2 > 0)$ .

insi d'après les données précédentes,  $\mathbf{x} = (x_1, x_2)$  et  $\mathbf{y} = (y_1, y_2)$  impliquent respectivement que  $(A\mathbf{y}_1)$  et  $(A\mathbf{y}_2)$  ont la même valeur tout comme  $(B\mathbf{x}_1)$  et  $(B\mathbf{x}_2)$ . D'où le système suivant :

$$\begin{cases} 3x_1 + 2x_2 = v \\ 2x_1 + 6x_2 = v \\ x_1 + x_2 = 1 \end{cases} \quad \begin{cases} 3y_1 + 3y_2 = w \\ 2y_1 + 5y_2 = w \\ y_1 + y_2 = 1 \end{cases}$$

ce qui nous donne :

$$\begin{cases} x_1 = 4x_2 \\ x_1 + x_2 = 1 \end{cases} \quad \begin{cases} y_1 = 2y_2 \\ y_1 + y_2 = 1 \end{cases}$$

Ainsi il apparait rapidement les stratégies mixtes suivantes :

$$\mathbf{x} = \left(\frac{4}{5}, \frac{1}{5}, 0\right) \quad \mathbf{y} = \left(\frac{2}{3}, \frac{1}{3}\right).$$

donnant les vecteur de PayOff respectif :

$$A\mathbf{y} = (3, 3, 2) \quad B\mathbf{x} = \left(\frac{14}{5}, \frac{14}{5}\right).$$

pour un PayOff total respectif :

$$\mathbf{x}^t A\mathbf{y} = (3) \quad \mathbf{x}^t B\mathbf{y} = \left(\frac{14}{5}\right).$$

### 4.1.3 Jeux à somme nulle

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

Un jeu à somme nul est un jeu où le gain d'un des acteurs (J1, par exemple) représente la perte équivalente réciproque des autres acteurs (J2, par exemple). Pour faciliter la compréhension et la résolution de ces modèles, le nombre d'acteurs autrement dit de joueurs sera fixé à 2.

Il existe plusieurs jeux ou situations "connus" que l'on pourrait assimiler à un jeu à somme nulle comme le jeu de pile ou face mais nous allons nous plonger sur un autre jeu un peu moins connu **Matching pennies** et nous allons essayer de mieux comprendre qu'est ce qu'un jeu à somme nulle et surtout comment le gagner ou du moins trouver la meilleure solution.

### 4.1.4 Un exemple de jeu : "Matching pennies"

Si l'on essaie de traduire mot - a - mot le nom du jeu on obtient : "jeux d'appariement de sous"; évidemment cela ne nous avance pas plus. Alors comment pourrions nous définir et mieux intégrer le jeu **matching pennies** ?

C'est un jeu simple et l'un plus connu dans le concept de la théorie des jeux, il se joue seulement à deux joueurs JA : **Pair** et JB : **Impair**. Chaque acteurs (Pair & Impair) possèdent une pièce et choisissent simultanément et secrètement une des deux faces **Pile** ou **Face**, puis en fonction des résultats obtenu on assiste des scénarios de gains différents :

- Les deux faces des pièces correspondent et dans ce cas Pair garde les deux pièces (+1 Pair, -1 Impair)
- Les deux faces des pièces ne correspondent pas et dans ce cas Impair garde les deux pièces (+1 Impair, -1 Pair)

Ainsi d'après ce que nous avons déjà vu dans les sections précédentes on peut comprendre que **matching pennies** est un jeu à somme nulle puisque le gain d'un des deux joueurs (par exemple Pair) est bien la perte réciproque de l'autre joueur (ici Impair).

De même vous aurez compris que ce jeu permet aussi d'illustrer le concept de stratégies et de l'équilibre de Nash cependant cette configuration n'admet pas de stratégie pure pour l'équilibre de Nash mais plutôt une stratégie mixte reposant sur le choix probabiliste de chaque joueurs à choisir une des deux faces de la pièce.

Puisque c'est un jeu bi-matriciel à somme nulle, profitons de sa définition même pour avoir une vue d'ensemble sur le jeu ; c'est à dire les actions de chaque joueurs mais aussi les gains ou pertes respectifs en fonction des choix effectués (cf [4.1.1]).

		<u><b>Joueur B : <i>Impair</i></b></u>	
		<b>PILE</b>	<b>FACE</b>
<u><b>Joueur A : <i>Pair</i></b></u>	<b>PILE</b>	( 1 , -1 )	( -1 , 1 )
	<b>FACE</b>	( -1 , 1 )	( 1 , -1 )

FIGURE 3 – Représentation matricielle de "Matching Pennies"

Pour la représentation du jeu nous allons estimer que le gain ou perte est de 1 puisque c'est avec une seule pièce que nous jouons, que le joueur JA *Pair* joue sur les lignes tandis que le joueur JB *Impair* joue les colonnes.

#### 4.1.4.1 Matrice de gains des joueurs A et B

Ainsi comme nous l'expliquons depuis plusieurs paragraphes, le propre même du jeu bimatriciel repose sur l'existence de la matrice de "gain" ou de "payoff". Nous allons donc donner la matrice de gain A de JA *Pair* et celle de JB *Impair* qui n'est autre que l'opposé de celle de A ( $B = -A$ ).

$$\text{Matrice A} \quad \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \quad \text{Matrice B} \quad \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}$$

## 4.2 Jeux bimatriciel a sommes nulles et résolution par PL

C'est alors qu'intervient la notion de **stratégie** (ou encore, règles de décision) pour qu'un acteur puisse prendre une décision afin d'effectuer une future action dans le jeu.

**Définition 9** *La stratégie d'un acteur est l'une des options qu'il choisit dans un contexte où son choix dépend non seulement de ses propres actions, mais également de celles des autres.*

On peut donc distinguer deux grandes catégories de **stratégies** soit pures soit mixtes, nous rentrerons en détails dans les sections suivantes (respectivement [4.2.1] et [4.2.2]).

Maintenant que nous avons abordé la notion de stratégie, intéressons nous à la programmation linéaire. La "victoire" ou "réussite" d'un jeu se traduit mathématiquement par le **gain** le plus élevé. Ainsi le but est donc de d'utiliser la stratégie la plus optimale qui permettrait de maximiser les **gains** du jeu.

**Définition 10** *La programmation linéaire (PL) est un problème d'optimisation où la fonction objectif et les contraintes sont toutes linéaires. Le but de résoudre un PL est de trouver les variables optimales qui maximisent la fonction objective.*

C'est donc grâce à la **programmation linéaire** (via une implémentation sur Gurobi) que nous arriverons à trouver la solution optimale et dans le meilleur des cas l'équilibre de Nash[4.2.3].

D'une approche plus mathématique on peut résumer la résolution d'un jeu statique par **programmation linéaire** comme ceci :

$$\max \min \sum_{i=0} \pi_i^1 a_{ij} = \max \min \sum_i \sum_j \pi_i^1 a_{ij} \pi_j^2$$

Expliquons un peu plus cette formule ;  $\pi$  est une probabilité relative à une stratégie donc  $\pi_i^1$  est la stratégie relative 1 en fonction de  $i$  (etc...), quand à  $a_{ij}$  il s'agit de la matrice (ligne  $i$  x colonne  $j$ ) de gain ou récompense de  $A$  (donc du premier joueur). Le but même est de trouver, pour  $A$ , la stratégie mixte répondant à l'objectif de maximiser les gains de  $A$ .

Ainsi le joueur  $A$ , doit résoudre :

$$\begin{aligned} \max \min \sum_i \pi_i^1 a_{ij} \\ s.c \quad \sum_i \pi_i^1 = 1 \end{aligned}$$

Puisque  $\pi_i^1$  étant une probabilité, sa somme ne peut pas être supérieure à 1. Cependant nous ne sommes pas en présence encore d'un programme linéaire (dont nous avons connaissance de technique de résolution), pour cela nous allons le transformer en posant une fonction  $v$  à maximiser et ceux pour chacune des sommes dont on cherche le minimum de gain (du joueur adverse, ici  $B$ ).

Le problème devient donc sous forme linéaire (PL) avec pour fonction objective  $v$  à maximiser avec pour seules contraintes :

$$\begin{aligned} v &\leq \sum_i \pi_i^1 a_{ij} \\ \text{et toujours : } \sum_i \pi_i^1 &= 1 \end{aligned}$$

Comme nous l'avons expliqué plus haut dans un jeu à somme nulle trouver l'équilibre de Nash n'est pas forcément possible il faut donc raisonner autrement et utiliser le théorème du minimax. Pour résoudre cela, il faut tenter de maximiser la stratégie pure de  $JA$  Pair et donc trouver :

$$\text{maximiser } \min_{1 \leq j \leq m} \sum_i \pi_i^1 a_{ij}$$

Cependant le problème n'est pas tout à fait encore sous forme linéaire et donc cela il suffit de trouver la fonction à maximiser  $v$  tel que  $v$  soit plus petite que toutes les sommes que nous cherchons à minimiser. D'où :

$$\begin{aligned} & \text{maximiser } v \\ \text{s.c } & v \leq \sum_i \pi_i^1 a_{ij} \\ & \sum_i \pi_i^1 = 1 \\ & \pi_i^1 \geq 0 \quad \forall i \end{aligned}$$

#### 4.2.1 Matrice de gains des joueurs A et B

Ainsi comme nous l'expliquons depuis plusieurs paragraphes, le propre même du jeu bimatrixel repose sur l'existence de la matrice de "gain" ou de "payoff". Nous allons donc donner la matrice de gain A de JA Pair et celle de JB Impair qui n'est autre que l'opposé de celle de A ( $B = -A$ ).

$$\text{Matrice A} \quad \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \qquad \text{Matrice B} \quad \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}$$

#### 4.2.2 L'équilibre de Nash dans Matching Pennies

Supposons maintenant que le JA Pair joue Pile avec une probabilité  $p$  (donc il joue Face avec une probabilité de  $(1-p)$ ) et que le JB Impair joue Pile avec une probabilité  $q$  (donc il joue Face avec une probabilité de  $(1-q)$ ) le jeu se donc sous cette forme :

$$\begin{pmatrix} (pq) & p(q-1) \\ q(p-1) & (1-p)(1-q) \end{pmatrix}$$

En utilisant le théorème du minimax on arrive à la proposition suivante (pour JA Pair) :

$$\begin{aligned} pq + (1-p)(1-q) - p(1-q) - q(1-p) &= 4pq - 2p - 2q + 1 \\ &= (1-2p)(1-2q) \end{aligned}$$

On peut tout de suite voir que pour chaque "membre" soit positif autrement dit que JA Pair maximise ses gains et minimise sa perte (ou inversement avec JB Impair la seule situation stable s'apparente que si :  $p = \frac{1}{2}$  et que  $q = \frac{1}{2}$ .

### 4.3 Jeux Stochastiques

#### 4.3.1 Jeux dynamique

**Définition 11** *Un jeu est dynamique lorsqu'il se déroule en plusieurs étapes non simultanées, c'est à dire que les joueurs jouent plusieurs fois mais en ayant connaissance des actions des autres joueurs et donc peuvent établir des stratégies.*

Prenons pour hypothèse un jeu dynamique à deux J1 et J2, les deux joueurs vont effectuer tours à tours des actions en différé en ayant accès aux informations des actions du joueur précédent.

Ainsi un jeu dynamique peut être défini par :

- Un nombre fini J de joueurs :  $\{1, 2, \dots, J\}$
- Un nombre identique d'actions K par joueurs :  $\{1, 2, \dots, K\}$
- Un ensemble infini E d'états du jeu qui sont indicé par les actions des joueurs
- Une stratégie pure par joueurs composé des actions des joueurs respectifs

### 4.3.2 Fonction de valeur

Le critère d'évaluation d'une stratégie  $\pi$  correspond à l'espérance mathématique de la somme des gains obtenus en suivant cette stratégie et en partant d'un sommet initial  $s$ , cette espérance est appelée **fonction de valeur**.

Rappelons le tout de même une stratégie  $\pi_i(s)$  dépend de l'action choisi par le joueur  $i$  à l'état  $s$  en stratégie pure et  $\pi_i(s, a)$  en stratégie mixte avec une probabilité  $a$  pour l'état  $s$ .

L'une des variantes les plus connues est de pondérer les gains par un facteur  $\gamma \in [0; 1[$  afin de prendre en compte la valeur des gains futur. En notant  $r_t$  la récompense reçue à l'instant  $t \in \mathbb{K}$  et  $s^0$  l'état initial, on définit la **fonction de valeur** pour tout  $s \in S$  pour une stratégie  $\pi$  :

$$V^\pi(s) = E^\pi \left( \sum_{t=0}^{\infty} \gamma^t r^t \mid s^0 = s \right)$$

### 4.3.3 Modèles et principes de jeux stochastiques

Au début de chaque étape, le jeu est dans un certain état . Les joueurs sélectionnent des actions et chaque joueur reçoit un gain cela dépend de l'état actuel et des actions choisies. Le jeu passe alors à un nouvel état aléatoire dont la distribution dépend de l'état précédent et des actions choisies par les joueurs. La procédure est répétée au nouvel état et le jeu continue pour un nombre fini ou infini d'étapes. Le gain total pour un joueur est souvent considéré comme la somme actualisée des gains de l'étape ou la limite inférieure des moyennes des gains de l'étape.

#### 4.3.3.1 Modèles des jeux stochastiques

On définit  $\Phi$  un jeu stochastique fini à somme nulle tel que :

- Un ensemble  $\Omega$  fini d'états
- I l'ensemble fini d'action du joueur A
- J l'ensemble fini d'action du joueur B
- Une fonction de paiement  $g : I \times J \times \Omega \rightarrow [-M, M]$  (max JA et min JB)
- Une probabilité de transition d'état  $\rho : I \times J \times \Omega \rightarrow \Delta(\Omega)$

#### 4.3.3.2 Principes des jeux stochastiques

On part d'un état  $\omega_1$  donné et connu des deux joueurs (JA et JB), pour chaque étape  $t \in \mathbb{N}$  on observe :



- JA et JB observe  $\omega_t$  l'état courant et se rappelle des états précédents
- Simultanément et respectivement JA choisit une action mixte  $x_t$  dans  $X = \Delta(I)$  et JB choisit une action mixte  $y_t$  dans  $Y = \Delta(J)$
- Respectivement JA effectue une action  $i_t$  en fonction de  $x_t$  et JB effectue une action  $j_t$  en fonction de  $y_t$
- Une étape de paiement  $t : g_t = g(i_t, j_t, \omega_t)$
- L'état suivant  $\omega_{t+1}$  attribué selon  $\rho(i_t, j_t, \omega_t)$

Ainsi, dans un jeu stochastique, chaque action conjointe mène tous les joueurs à un nouvel état où se joue l'équivalent d'un jeu statique avec des gains particuliers à chaque état. En théorie, dans un jeu stochastique, les probabilités de transition peuvent dépendre de l'historique de tous les états passés ; lorsque les probabilités de transition ne dépendent que de l'état courant, il s'agit au sens strict d'un jeu de Markov. Durant le stage, on considérera uniquement des jeux où les probabilités de transition ne dépendent que de l'état courant.

À la différence des jeux statiques, les jeux stochastiques ne disposent pas de solutions optimales qui soient indépendantes des joueurs. Par conséquent, il faut définir la notion de stratégie optimale de manière analogue à celle d'un jeu classique. Ainsi, en notant  $\Pi_j$  l'ensemble des stratégies possibles du joueur  $j$ , on peut adapter la notion d'équilibre de Nash aux jeux stochastiques.

#### 4.3.3.3 Équilibre de Nash en jeux stochastiques

**Définition 12** Une stratégie conjointe  $\pi^*$  est un équilibre de Nash si et seulement si :

$$\forall s \in S, \forall j \in J, \forall \pi_j \in \Pi_j, V_j^{\pi^*}(s) \geq V_j^{\pi_j, \pi_{-j}^*}(s)$$

## **5 Conclusions**

### **5.1 Contributions apportées**

### **5.2 Difficultés rencontrées**

### **5.3 Axes d'amélioration**

## 6 Webographie

### Références

[Paris X (Nanterre)] <https://www.parisnanterre.fr/>

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

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

[Gurobi] <https://www.gurobi.com/downloads/gurobi-software/>

[Gurobi] [https://www.gurobi.com/documentation/9.0/refman/py\\_model.html](https://www.gurobi.com/documentation/9.0/refman/py_model.html)

[Théorie des Jeux] [https://fr.wikipedia.org/wiki/Th%C3%A9orie\\_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 Nanterre

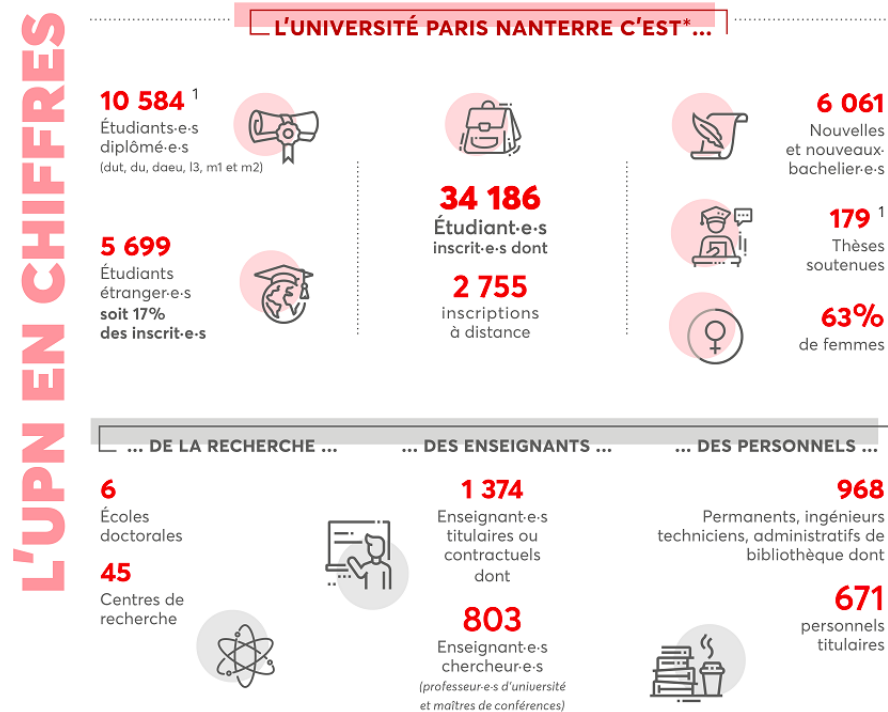


FIGURE 4 – L'Université Paris X en chiffres

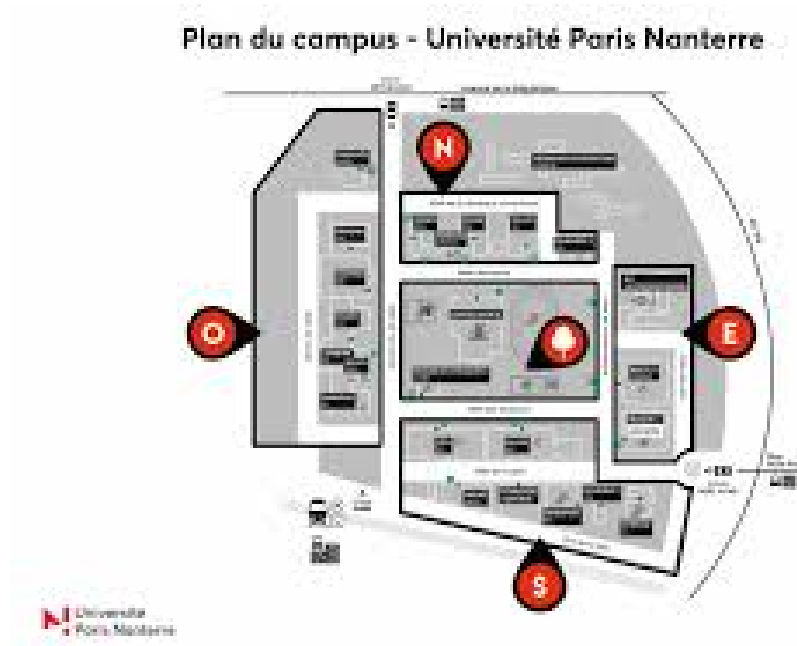


FIGURE 5 – Le plan de Paris X

## 7.2 Python en général

La syntaxe est assez similaire aux autres langages 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.

### 7.2.1 Structure Conditionnelle If

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

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

### 7.2.2 Structure Conditionnelle Else

La condition est suivie 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>
```

### 7.2.3 Structure Conditionnelle Elif

La condition est suivie 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>
```

### 7.2.4 Boucle For

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

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

### 7.2.5 Boucle While

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

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

### 7.2.6 Les fonctions

Quant à la fonction la définition se fait de manière très simple il suffit d'utiliser le mot clé `def` et cela est terminé, 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 fonction (param1 , param2) :  
    <instruction1>  
    <instruction2>  
    if <test1> :  
        <instruction3>  
    else :  
        <instruction4>  
    return <instruction5>
```

## 7.3 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.

### 7.3.1 Création de class

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

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

### 7.3.2 Création du constructeur

Ensuite il faudra définir un constructeur qui permettra d'instancier les objets dont nous aurons besoin, 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)
```

### 7.3.3 Le mot clé pass

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

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

### 7.3.4 L'héritage en python

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 paranthèse 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 gère 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 seulement une instance d'une classe et pour savoir si la classe en question a eu recours à de l'héritage : `isinstance()` et `issubclass()`.

## 7.4 GitHub

En résumé les commandes principales de Github.

git init	git remote add
git clone	git checkout
git branch < branche >	git branch -d < branche >
git add < fichier >	git add * (pour tous les fichiers)
git commit -m ".."	git merge
git push origin < branche >	git pull origin < master >

## 7.5 Gurobi

Voici les deux méthodes (via `gurobi.sh` ou alors via le module `gurobipy`) que l'on peut utiliser pour résoudre un programme MIP nommé "biere.py" (voir exemple [5.3]) :

```

aviassayag@MacBook-Air-de-Avi ~ % gurobi.sh biere.py
Using license file /Users/aviassayag/gurobi.lic
Academic license - for non-commercial use only
Gurobi Optimizer version 9.0.2 build v9.0.2rc0 (mac64)
Optimize a model with 3 rows, 2 columns and 6 nonzeros
Model fingerprint: 0x31aa2bc2
Variable types: 0 continuous, 2 integer (0 binary)
Coefficient statistics:
  Matrix range [1e-01, 2e+01]
  Objective range [2e+01, 2e+01]
  Bounds range [0e+00, 0e+00]
  RHS range [5e+00, 6e+02]
Found heuristic solution: objective 510.0000000
Presolve time: 0.00s
Presolved: 3 rows, 2 columns, 6 nonzeros
Variable types: 0 continuous, 2 integer (0 binary)

Root relaxation: objective 8.800000e+02, 2 iterations, 0.00 seconds

  Nodes | Current Node | Objective Bounds | Work
Expl Unexpl | Obj Depth IntInf | Incumbent BestBd Gap | It/Node Time
* 0 0 | 0 | 880.0000000 880.00000 0.00% - 0s

Explored 0 nodes (2 simplex iterations) in 0.00 seconds
Thread count was 4 (of 4 available processors)

Solution count 2: 880 510

Optimal solution found (tolerance 1.00e-04)
Best objective 8.800000000000e+02, best bound 8.800000000000e+02, gap 0.0000%
x1 12
x2 28
Obj: 880.0
aviassayag@MacBook-Air-de-Avi ~ %

```

FIGURE 6 – Résolution via shell gurobi (`gurobi.sh`)

```

Last login: Tue Apr 28 11:12:22 on ttys001
aviassayag@MacBook-Air-de-Avi ~ % python3 biere.py
Using license file /Users/aviassayag/gurobi.lic
Academic license - for non-commercial use only
Gurobi Optimizer version 9.0.2 build v9.0.2rc0 (mac64)
Optimize a model with 3 rows, 2 columns and 6 nonzeros
Model fingerprint: 0x31aa2bc2
Variable types: 0 continuous, 2 integer (0 binary)
Coefficient statistics:
  Matrix range [1e-01, 2e+01]
  Objective range [2e+01, 2e+01]
  Bounds range [0e+00, 0e+00]
  RHS range [5e+00, 6e+02]
Found heuristic solution: objective 510.0000000
Presolve time: 0.00s
Presolved: 3 rows, 2 columns, 6 nonzeros
Variable types: 0 continuous, 2 integer (0 binary)

Root relaxation: objective 8.800000e+02, 2 iterations, 0.00 seconds

  Nodes | Current Node | Objective Bounds | Work
Expl Unexpl | Obj Depth IntInf | Incumbent BestBd Gap | It/Node Time
* 0 0 | 0 | 880.0000000 880.00000 0.00% - 0s

Explored 0 nodes (2 simplex iterations) in 0.00 seconds
Thread count was 4 (of 4 available processors)

Solution count 2: 880 510

Optimal solution found (tolerance 1.00e-04)
Best objective 8.800000000000e+02, best bound 8.800000000000e+02, gap 0.0000%
x1 12
x2 28
Obj: 880.0
aviassayag@MacBook-Air-de-Avi ~ %

```

FIGURE 7 – Résolution via module gurobi (`gurobipy`)

Il est logique que le résultat produit est le même sauf la commande utilisé n'est pas la même. L'avantage de la deuxième méthode est que l'on peut importer le module `gurobipy` dans n'importe quelle future création Python.



## 7.6 CV



**Avi**  
**ASSAYAG**



**OUTILS**  
**INFORMATIQUES**

SUITE OFFICE ●●●●●●

HTML CSS ●●●●●●

LANGAGE C ●●●●●●

SQL PHP ●●●●●●

C ++ JAVA ●●●●●●

PYTHON ●●●●●●

BASH/SHELL ●●●●●●



**LANGUES**

FRANCAIS ●●●●●●

ANGLAIS ●●●●●●



**CENTRES D'INTERET**

 SOUTIEN SCOLAIRE
  CUISINE
  VOYAGES
  SPORT

06.21.91.94.18

24 ans , Permis B

avi\_assayag22@hotmail.fr

Villeneuve La Garenne 92390

**DEVELOPPEUR JUNIOR**

« Jeune étudiant **dynamique sérieux** et **ambitieux**, en **Licence 3 MIAE** à l'Université **Paris Nanterre** en constante **recherche de connaissances et d'évolution** »

**FORMATIONS**

**2019-2020** **Licence 3 MIAE**  
• Université Nanterre Paris Ouest

**2018-2019** **Licence 2 MIAHS / MIAE**  
• Université Nanterre Paris Ouest

**2015-2016** **BTS Opticien Lunetier**  
• ORT Daniel Mayer (ORT Montreuil)

**EXPERIENCES PROFESSIONNELLES**

**Mars 2020 -10 Semaines-** **Stage Université Paris Nanterre**  
• Implémentation d'algorithmes pour modèles de jeux stochastique. (Python et Gurobi)

**2017-2018** **Responsable Magasin Optic Express**  
• Formation d'un alternant  
• Gestion des stocks et achats  
• Ventes montages et tiers payants

**2016-2017** **Opticien colaboreur Optic 2000**  
• Ventes, montages et SAV  
• Gestion des tiers payants

**PROJETS D'ETUDES**

**LICENCE 3** Générateur de Trombinoscope (Latex & C)

**LICENCE 3** Site internet (PHP & HTML & CSS)

**LICENCE 2** Visionneuse Photo (Qt Creator & C++)

**LICENCE 2** Bataille Navale (Langage C)

FIGURE 8 – Curriculum Vitae Avi ASSAYAG L3 MIAE

## Table des figures

1	Le campus de Nanterre . . . . .	6
2	Shell interactive Gurobi . . . . .	15
3	Represenation matricielle de "Matching Pennies" . . . . .	22
4	L'Université Paris X en chiffres . . . . .	28
5	Le plan de Paris X . . . . .	28
6	Résolution via shell gurobi (gurobi.sh) . . . . .	32
7	Résolution via module gurobi (gurobipy) . . . . .	32
8	Curriculum Vitae Avi ASSAYAG L3 MIAGE . . . . .	33