



Licence MIASHS parcours MIAGE

Rapport de stage

L3 MIAGE, parcours classique

Implémentation d'algorithmes pour modèles de jeux 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, Maître de conférence à l'Université de Nanterre et chercheur dans l'équipe SYSDEF du Lip6, d'avoir accepté le poste de tuteur pour mon stage de Licence 3 MIAGE. Grâce à son accompagnement personnel j'ai pu solidifier mes compétences algorithmiques (Java et Python) mais aussi découvrir d'autres aspects 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 je les remercie pour leur patience, pour leurs encouragements et pour l'encadrement tout au long de ce stage.

Merci aussi à tous mes autres professeurs qui ont contribué tout au long de l'année à parfaire mes compétences tant 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 décidez de vous lancer dans le même thème.

Résumé

Pendant ce stage, j'ai eu pour objectifs d'implémenter des algorithmes pour modèles de jeux stochastiques. Pour cela, j'ai manipulé Python (POO), la programmation linéaire (PL) et **gurobi**. J'ai aussi travaillé pendant de nombreuses heures sur les concepts de la théorie des jeux (jeux bimatriciel à sommes nulles, stratégies pures et mixtes, équilibre de Nash ...).

Au terme de ces dix semaines je suis parvenu à modéliser un jeu à somme nulle : **matching pennies** ainsi que trouver les programmes linéaires associés aux stratégies respectives des deux joueurs (JA & JB). J'ai aussi résolu des programmes linéaires via **gurobi**, que nous avons vu en cours, avant de résoudre ceux de **matching pennies**.

J'ai aussi codé, en python à l'aide du module **gurobipy** une résolution de programme linéaire (.py) mais aussi un fichier de lecture permettant à **gurobi** de lire un modèle et donner si elle existe, la solution optimale (grâce au fichier "lp")

Dans les semaines à venir, je vais implémenter des modèles pour **jeux stochastiques** dans le but de trouver l'**équilibre de Nash** mais aussi automatiser les résolutions afin de pouvoir donner le type et les stratégies d'un jeu passé en paramètre.

During this internship, my goals were to implement algorithms for stochastic game models. For this, I manipulated Python (POO), linear programming (LP) and **gurobi**. I also worked for many hours on game theory concepts (zero-sum bimatrix games, pure and mixed strategies, and Nash equilibrium).

At the end of these ten weeks, I managed to model a zero-sum game : **matching pennies** as well as finding the linear programs associated with the respective strategies of the two players (JA & JB). I also solved linear programs through **gurobi**, which we had seen in class, before solving the **matching pennies**.

I also coded in python using the **gurobipy** extension, a linear program resolution but also a reading file which allows **gurobi** to read a model and give, if it exists, the optimal solution (thanks to "lp" file).

In the upcoming weeks, I will implement models for **stochastic games** in order to find the **Nash equilibrium** but also automate the resolutions in order to give the type and strategies of a game passed in parameters.

Table des matières

1	Introduction	6
2	Contexte du Stage	7
2.1	Présentation de l'entreprise	7
2.2	Présentation du service et de l'équipe	8
2.3	Mission proposée	9
2.4	Cahier des charges	9
3	Outils utilisés	10
3.1	GitHub	10
3.1.1	Initialisation	10
3.1.2	Branches	10
3.1.3	Dépôt et mises à jour	11
3.1.4	Clonage	11
3.2	Python	11
3.2.1	Procédure d'installations de Python	12
3.2.2	Un exemple générique de code Python	13
3.2.3	La programmation objet en Python	13
3.3	Gurobi	15
3.3.1	Installation de Gurobi	15
3.3.2	Utilisation de gurobi	16
3.3.3	Exemple de programmation linéaire avec Gurobi	16
4	Théorie des jeux	19
4.1	Jeux statiques	19
4.1.1	Principes et modèles des jeux statiques	19
4.2	Jeux bimatrixiel	21
4.2.1	Exemple de jeu bimatrixiel et illustration des concepts	21
4.3	Jeux à sommes nulles	22
4.3.1	Un exemple de jeu à somme nulle : "Matching pennies"	23
4.3.2	Un autre exemple de jeu à somme nulle : "La bataille des réseaux"	24
4.4	Jeux bimatrixiel à sommes nulles et résolution par PL	25
4.4.1	Explication de la démarche	25
4.4.2	Programme linéaire du Joueur A : maximiser v	26
4.4.3	Programme linéaire du Joueur B : maximiser w	26
4.4.4	L'équilibre de Nash dans Matching Pennies par PL	26
4.5	Jeux Stochastiques	27
4.5.1	Jeux dynamiques	27
4.5.2	Fonction de valeur	27
4.5.3	Modèles et principes de jeux stochastiques	28
5	Conclusions	29
5.1	Contributions effectuées	29
5.2	Difficultés rencontrées	29
5.3	Perspectives et améliorations	30
6	Webographie	31

7	Annexes	32
7.1	Nanterre	32
7.2	Python en général	33
7.2.1	Structure Conditionnelle If	33
7.2.2	Structure Conditionnelle Else	33
7.2.3	Structure Conditionnelle Elif	33
7.2.4	Boucle For	33
7.2.5	Boucle While	33
7.2.6	Les fonctions	34
7.3	L'orienté objet en Python	34
7.3.1	Création de class	34
7.3.2	Création du constructeur	34
7.3.3	Le mot clé pass	34
7.3.4	L'héritage en python	35
7.4	GitHub	35
7.5	Gurobi	36
7.6	CV	37

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 sommes nulles (que nous représenterons sous forme bimatrice).

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

Pour parfaire à ces attentes, nous allons utiliser le langage **Python**, non abordé cette année, le solveur **Gurobi**, que nous manipulerons 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, documents qui m'ont aidé à réaliser ce projet.

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 mes codes afin de superviser mes avancées. 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 **jeux à sommes nulles** , c'est à dire un jeu où le gain d'un des acteurs représente la perte exacte des autres , mais nous l'expliquerons en détail dans une prochaine section cf[4.1.3].

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

Dans les semaines de ce stage nous avons décidé 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 profondeur 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 second semestre de la troisième année de ma licence Miage (S6), je suis tenu 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, afin de me former et de compléter mes connaissances. En attendant ces réponses, j'ai effectué une demande (avec l'accord de mon professeur référent **Monsieur Delbot**) auprès de l'un de enseignants de l'université.

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

2.1 Présentation de l'entreprise

L'université Paris Nanterre, également appelée **Paris X** est une université française fondée le 20 Octobre 1964 et spécialisée 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 de nombreux espaces verts , un terrain d'athlétisme, des cours de tennis et une piscine. On y dénombre environ 35 000 étudiants (sans compter ceux inscrits à distance) répartis dans 8 UFR (Unité de Formation et de Recherche) et 5 instituts, pour près de 2 200 enseignants et chercheurs.



FIGURE 1 – Le campus de Nanterre

Comme nous l'avons expliqué l'Université est organisée autour 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)

Il existe une hiérarchie au sein l'Université. Monsieur **Jean-François Balaudé** Président Directeur depuis 2012. Chaque UFR a un président et chaque département 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 celui-ci au sein de l'UFR **SEGMI** (Sciences Économiques, Gestion, Mathématiques et Informatique).

La présidence de l'UFR est géré par Monsieur **Yann Demichel**, enseignant chercheur en mathématique. La formation MIAGE (celle dans laquelle je me trouve), est dirigée par Monsieur **Jean-François Pradat-Peyre**, enseignant chercheur et responsable du département mathématique et informatique. Les master sont respectivement suivis par Monsieur **Pascal Poizat** pour le classique et par Madame **Marie Pierre Gervais** pour l'apprentissage. Enfin les licences sont présidées respectivement par Monsieur **François Delbot** pour le parcours classique et par Madame **Sonia Saadaoui** pour l'apprentissage.

Mon tuteur Monsieur **Emmanuel Hyon** est enseignant chercheur mais également maître de conférence à l'Université de Nanterre (Paris X) depuis 2004. Il enseigne l'algorithme et programmation C lors du premier semestre de la licence MIAGE.

Il travaille au LIP6 (Département Desir) en recherche opérationnel (RO) et sur modélisation mathématique et informatique à l'université de Nanterre (Paris X). Il a choisi comme thème de stage : **Implémentation d'algorithmes pour modèles de jeux stochastiques**.

Dans le contexte de crise sanitaire auquel nous sommes confronté j'ai effectué mon stage à domicile sans la présence de mon tuteur, ce qui à rendu celui ci plus complexe dans la mesure où nos échanges étaient hebdomadaire et je n'ai pu le solliciter comme j'aurais pu le faire en présentiel. Pour remédier à cela et à l'initiative de Monsieur **Hyon** 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 questions, de me donner les axes que je devais améliorer ainsi que les points sur lesquels je devais travailler pour la semaine à venir.

2.3 Mission proposée

Il m'a été confié de modéliser un jeu stochastique sur un plan informatique (via python) mais surtout de trouver un moyen, par un algorithme de le résoudre.

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.

J'ai du également 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]).

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

Aimant les mathématiques et l'informatique depuis jeune le sujet, de ce stage me correspondait parfaitement et complétait mon envie de poursuivre mes études avec un master MIAGE.

2.4 Cahier des charges

Il m'est un peu compliqué de donner des dates exactes et précises de chaque tâche réalisées ou effectuées mais je peux détailler la liste des charges accomplies :

- Mise en place d'un lien **GitHub** (pour avoir accès en temps réel aux codes sources et 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)

Pendant 10 semaines et depuis le 23 mars, je travaille quotidiennement de le but d'implémenter ces 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 / requisitory) dans lequel on va stocker tout les fichiers que l'on désire et on peut, soit rendre l'accès public (auquel cas tout le monde peut rejoindre et consulter ces fichiers) ou alors, le restreindre en accès privé (auquel cas c'est le créateur qui décide quels seront les collaborateurs ayant droit de consultation des fichiers).

Ensuite cela agit comme une sorte de réseau constitué de branches (branch) où chaque branche représente un collaborateur et la "master" correspond au créateur du référentiel.

Une des caractéristiques de Git est que c'est un outil de verisonning (gestion de version) qui permet de savoir si le fichier à été modifié. Si c'est le cas, par qui, quand 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 tout en ayant besoin des parties des autres membres du groupe (mises 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 tout comme les commandes principales pour chaque étape.

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

Il faudra ensuite 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.

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 flexi-

bilité qui permet d'incrémenter au fur et à mesure le projet.

Pour ajouter une branche il suffit simplement de taper `git branch < nom de la branche >`, pour supprimer une branche il faut rajouter l'option `-d` à la commande soit : `git branch -d < nom de la branche >`.

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

3.1.3 Dépôt et mises à jour

Avant toute chose il faut savoir sur quelle branche déposer le fichier puis il faut 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'ils copient le lien du git pour pouvoir travailler dessus et effectuer les futurs dépôts. En premier lieu il faut 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 a été 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 possibilités de code mais aussi d'actions puisqu'il existe des bibliothèques déjà implémentées et il suffit seulement des les utiliser comme bon nous 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 va de soit que les types primitifs sont aussi présents `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 de paramètres. Il donne 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 3.8.3) 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 jour la version qui pourrait être obsolète ou dépassé.

3.2.1 Procédure d'installations de Python

3.2.1.1 Méthode packages

Pour cela il faut aller 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 simples. Après avoir téléchargé les packages Python sur le site, on installe tout les modules externes (pip , Django , Gurobi etc ...) dont on a besoin d'utiliser via le terminal :

```
pip install <nom_module>
```

3.2.1.3 Environnement virtuel python

On parle d'environnement virtuel pour certaines utilisations bien propres à **python**. Il arrive parfois que certaines versions ne permettent pas d'utiliser certains modules, 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 nécessaires 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 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. pour les boucles et les conditions il suffit d'utiliser le mot clé en question suivit de " : " et donner les instructions de façon avec l'indentation requise

Rien de mieux qu'un petit code Python pour mieux comprendre la syntaxe et la facilité d'utilisation du langage. Je vous présenter ci-dessous 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 nombreux plus de ce langage puissant et aux multiples possibilités. Dans cette partie je vais vous montrer comment coder un objet en Python et aussi le construire. Je vais vous montrer la syntaxe générale d'une Classe et celle d'un Constructeur. Enfin pour terminer, je vais implémenter 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 suivit 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` dans le 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. Je vais 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 langages : 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 je vais vous d'expliquer.

Travaillant sur MacOS, j'ai opté pour l'installation de **Gurobi** en privé sur ma machine et donc en gérant l'installation sur mon environnement Python, et j'ai utilisé le module *gurobipy*. 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 propres soins puisque l'environnement Python y a pensé cela lors de sa conception.

Pour cela il faut au préalable télécharger le solveur sur le site web de **Gurobi** (le lien est en annexe) et attendre le téléchargement. Une fois terminé il faut d'exécuter le fichier télécharger (en double cliquant dessus) pour démarrer l'installation. Le système d'exploitation nous demande dans quel dossier stocker les packages nécessaires à **Gurobi**. Ensuite il faudra se rendre à cet emplacement, via un terminal et exécuter la commande suivante :

```
python setup.py install
```

Après avoir créé son compte afin d'obtenir une licence, il faut l'enregistrer sur la machine afin d'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 ceci effectué et donc paramétré il faut, pour utiliser le solveur, faire un **import** du module et donc de la bibliothèque **Gurobi** au début du script python qu'on élabore.

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

3.3.1.2 Méthode via Anaconda

Anaconda est une solution libre office, c'est à 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**.

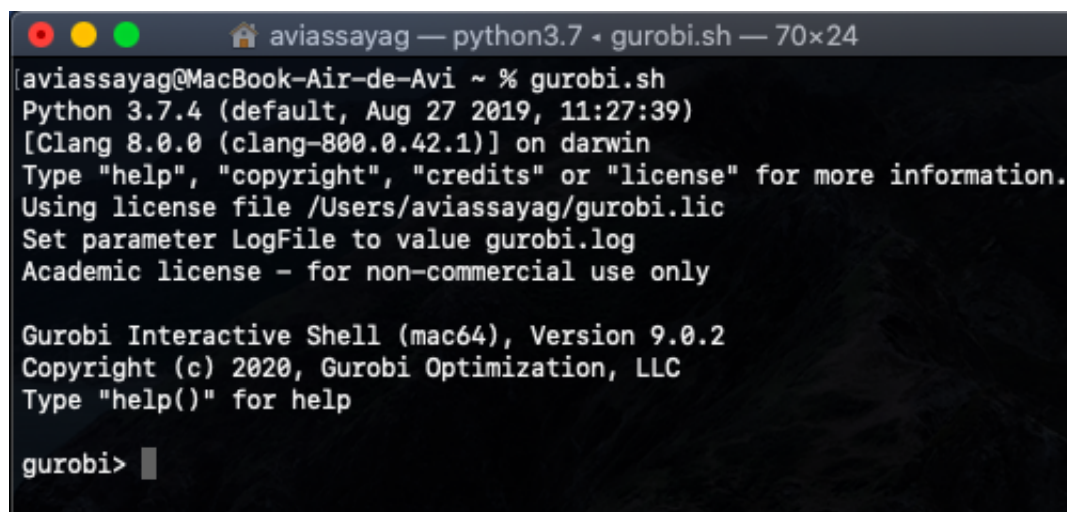
```
via Python :      pip install <module>
via Anaconda :    conda install <module>
```

Via cette méthode, l'environnement est déjà préinstallé pour l'utilisateur et comporte une interface graphique **Spyder** ainsi qu'un éditeur de texte **Jupyter**. Pour cela il suffit 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 fourni avec le téléchargement du solveur. En en le lançant, c'est à dire en le tapant à la console le terminal lance une console **gurobi** où il faut directement écrire le code à exécuter. Ainsi un interpréteur **Gurobi** s'ouvre et attend 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

Pour résoudre un programme linéaire avec **gurobi** il existe deux méthodes d'implémentation soit en le codant directement en utilisant le module **gurobipy** et ses méthodes (fonctions et constructeurs) cf[3.3.3.1] soit en utilisant un fichier de lecture déjà implémenté par le solveur (lp.py) . Il reste à écrire le fichier d'ouverture cf[3.3.3.2].

Je vais résoudre un programme linéaire avec les deux techniques et utiliser le même énoncé.

Soit x_1 et x_2 les quantités (en volume) respectives produites pour les bières b1 et b2. Les quantités sont soumises à des contraintes (3) pour chaque ingrédient 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 : $\max 15 x_1 + 25 x_2$

Solutions :

- $x_1 = 12$
- $x_2 = 28$

Le programme apparaît déjà sous forme canonique pure mais pour une meilleur visibilité je modifie quelques contraintes. Je vais diviser C1 par 2,5. Multiplier C2 par 100 et le diviser par 125. Enfin multiplier par C3 par 10 et le diviser par 25.

$$\left\{ \begin{array}{lcl} \text{Maximiser} & 15x_1 + 25x_2 & \\ & sc & \\ & x_1 + 3x_2 & \leq 96 \\ & x_1 + x_2 & \leq 40 \\ & 7x_1 + 4x_2 & \leq 238 \\ & x_1, x_2 & \geq 0 \end{array} \right.$$

3.3.3.1 Résolution en implémentant un code gurobi : "biere.py"

Il faut fait appel aux modules de **gurobi** et les utiliser, créer le modèle grâce au constructeur associé **Model()**, ajouter les variables de décisions **addVar()**, donner la fonction objective et son action associé **setObjective()**, informer les contraintes **addConstr()**, la résolution **optimize()** et enfin gérer l'affichage de la solution avec **getVars()**. Pour être plus méthodique et consciencieux il est faut vérifier les exceptions et les erreurs. Ce qui donne :

```
import gurobipy as gp
from gurobipy import *

try :
    m = gp.Model("Biere")

    x1 = m.addVar(vtype=GRB.INTEGER, name="x1")
    x2 = m.addVar(vtype=GRB.INTEGER, name="x2")

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

    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")

    m.optimize()
```

```
for v in m.getVars():
    print('%s %d' % (v.varName, v.x))

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

Ci dessus le code python utilisé pour permettre a **Gurobi** de trouver la solution de maximisation, soit : **x1= 12 et x2 = 28** en exécutant la commande **python3 biere.py**

3.3.3.2 Résolution via le fichier de lecture LP : "biere.lp"

La syntaxe est différente du code précédent puisque ici j'utilise un fichier **lp.py** qui demande des informations sans syntaxe particulière. Le fichier se compose de "quatre" parties et se finit toujours par **end** annonçant la fin de lecture. La structure est la suivante : en premier lieu l'objectif ensuite viennent les contraintes des variables puis les limites des variables (positivité) et enfin le types des variables de décision. Ce qui donne :

```
Maximise
    15 x1 + 25 x2

Subject To
    c0: 2.5 x1 + 7.5 x2 <= 240
    c1: 0.125 x1 + 0.125 x2 <= 5
    c2: 17.5 x1 + 10 x2 <= 595

Bounds
    x1 >= 0
    x2 >= 0

Integers
    x1 x2

End
```

Ci dessus le code utilisé pour permettre a **Gurobi** de trouver la solution de maximisation en lisant un fichier soit : **x1= 12 et x2 = 28** en exécutant la commande **python3 lp.py biere.lp**

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 je l'ai expliqué précédemment, un des objectifs de ce stage est la modélisation d'algorithmes afin de résoudre des jeux stochastiques. Détaillons 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.

Un jeu nécessite la présence d'acteurs, dans le langage courant cela fait référence à des **joueurs**. Dans mon projet le nombre d'acteur sera fixé à deux.

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_i . Ce vecteur est noté π^1 pour le joueur 1 et π^2 pour le joueur 2. 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.*

C'est maintenant 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. On peut donc distinguer deux grandes catégories de **stratégies** soit pures soit mixtes

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 π^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 π^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 π^1 est une probabilité on a obligatoirement $\sum_i \pi^1(i) = 1$.*

4.1.1.3 Equilibre de Nash

Maintenant que l'on connaît un peu plus le concept fondamental de la **théorie des jeux** j'ai compris que résoudre un jeu (dynamique bien évidemment) ne revient non pas à "gagner" la partie mais à trouver la meilleure 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 optimale de stratégies mixtes (x, y) qui sont meilleures réponses l'une en 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 stratégie spécifiant pleinement pour tout joueur, toutes les actions dans le jeu. Il comporte une seule et unique stratégie pour chaque joueur.

Définition 7 *Soit π^i une stratégie du joueur i et π^{-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 σ^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 σ^* un profil de stratégie mixte. σ^* est un équilibre de Nash si, pour tout joueur i on a :*

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

4.2 Jeux bimatrixiel

Un jeu bimatrixiel 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 je vais alors donc utiliser strictement des jeux à deux joueurs JA et JB.

Ainsi un jeu bimatrixiel 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) G1 pour le joueur JA $[M \times N]$
- Une matrice de gain (pay-off) G2 pour le joueur JB $[N \times M]$
- Une stratégie pure par joueurs composé des actions des joueurs respectifs

4.2.1 Exemple de jeu bimatrixiel et illustration des concepts

$$\text{Jeu total} \quad \begin{pmatrix} 3, 3 & 3, 2 \\ 2, 2 & 5, 6 \\ 0, 3 & 6, 1 \end{pmatrix}$$

$$\text{Joueur A} \quad \begin{pmatrix} 3 & 3 \\ 2 & 5 \\ 0 & 6 \end{pmatrix}$$

$$\text{Joueur B} \quad \begin{pmatrix} 3 & 2 \\ 2 & 6 \\ 3 & 1 \end{pmatrix}$$

4.2.1.1 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.

4.2.1.2 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 $(x_1, x_2, y_1, y_2 > 0)$.

Considérons alors 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 :

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

pour un PayOff total respectif :

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

4.2.1.3 Vérification de l'équilibre de Nash

Pour vérifier qu'il s'agit bien d'équilibre de Nash il faut veiller à ce qu'aucun joueur ne change de stratégie; regardons donc notre situation.

Stratégie $x = (1,0,0)$, $(1,0)$ Dans ce cas ci, A joue 3 et B joue 3 personne de domine personne cela pourrait être un équilibre.

Stratégie $x = (0,1,0)$, $(1,0)$: Dans ce cas ci, A joue 2 et B joue 3 alors B domine A puisque $2 < 3$.

Stratégie $x = (0,0,1)$, $(1,0)$: Dans ce cas ci, A joue 0 et B joue 3 alors B domine A puisque $0 < 3$.

Stratégie $x = (1,0,0)$, $(0,1)$: Dans ce cas ci, A joue 3 et B joue 2 alors A domine B puisque $3 > 2$.

Stratégie $x = (0,1,0)$, $(0,1)$: Dans ce cas ci, A joue 2 et B joue 6 alors B domine A puisque $2 < 6$.

Stratégie $x = (0,0,1)$, $(0,1)$: Dans ce cas ci, A joue 0 et B joue 6 alors B domine A puisque $0 < 6$.

Ainsi on remarque bien que seule la première solution permet un équilibre de Nash en stratégie pure puisque c'est la situation parfaite pour que personne n'ai envie de change puisque si l'un des deux joueurs A ou B change d'action il donne l'avantage à son adversaire.

4.3 Jeux à sommes nulles

Comme annoncé dans notre introduction, je vais résoudre des jeux à sommes nulles via des algorithmes que je vais implémenter par la suite.

Un jeu à somme nulle est un jeu où le gain d'un des acteurs (JA, par exemple) représente la perte équivalente des autres acteurs ((JB, par exemple). Pour faciliter la compréhension et la résolution de ces modèles, le nombre d'acteurs sera fixé à 2.

Ainsi on peut définir un jeu à somme nulle par :

- La somme des gains des acteurs s'annule : $\forall a \in A \sum_{j \in J} R_j(a) = 0$
- Deux stratégies π_1 et π_2 appartenant réciproquement à JA et JB
- Une fonction de gain $R(\pi_1, \pi_2)$ pour JA et $-R(\pi_1, \pi_2)$ pour JB
- Une valeur du jeu V^* déterminable par le théorème du minimax

$$V^* = \max_{\pi_1 \in \Pi_1} \min_{\pi_2 \in \Pi_2} R(\pi_1, \pi_2) = \min_{\pi_2 \in \Pi_2} \max_{\pi_1 \in \Pi_1} R(\pi_1, \pi_2)$$

Il existe plusieurs jeux ou situations "connus" que l'on peut assimiler à un jeu à somme nulle, comme le jeu de pile ou face mais je vais m'intéresser à un autre jeu moins étudié, Matching pennies et je vais essayer de comprendre ce qu'est un jeu à somme nulle et trouver la solution optimale.

4.3.1 Un exemple de jeu à somme nulle : "Matching pennies"

C'est un jeu simple de la théorie des jeux, il se joue à deux joueurs JA : Pair et JB : Impair. Chaque acteur possède une pièce et choisit simultanément et secrètement une face., En fonction du choix de chacun, on obtient des résultats suivant :

- Les deux faces des pièces sont identiques et dans ce cas Pair garde les deux pièces (+1 Pair, -1 Impair)
- Les deux faces des pièces sont différentes et dans ce cas Impair garde les deux pièces (+1 Impair, -1 Pair)

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

FIGURE 3 – Représentation matricielle de "Matching Pennies"

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 joueur 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]).

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

Matrice de gains des joueurs A et B

Ainsi comme je le développe depuis plusieurs paragraphes, le propre même du jeu bimatriciel repose sur l'existence de la matrice de "gain" ou de "payoff". Je vais 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 ($B = -A$).

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

4.3.2 Un autre exemple de jeu à somme nulle : "La bataille des réseaux"

Considérons deux sociétés A et B se disputant des parts sur les réseaux sociaux sur deux thèmes différents : le **sport** et les **séries**. Lorsqu'une des sociétés gagne des parts de marché l'autre en perd automatiquement la même somme ; on peut donc assimiler notre situation à un jeu à somme nulle.

Matrice de gains des sociétés A et B

$$\text{Matrice A} \quad \begin{pmatrix} 10 & 4 \\ 0 & -10 \end{pmatrix} \quad \text{Matrice B} \quad \begin{pmatrix} -10 & -4 \\ 0 & 10 \end{pmatrix}$$

		<u>Société B</u>	
		SPORT	SERIE
<u>Société A</u>	SPORT	(10 , -10)	(4 , -4)
	SERIE	(0 , 0)	(-10 , 10)

FIGURE 4 – Représentation matricielle de "Bataille de réseaux"

On peut remarquer que la situation la plus avantageuse pour A, augmentant ses gains est : 10 ou 4 quand à B il s'agit de -4 et 10. on en conclure que l'équilibre de Nash peut se trouver, dans cette situation via la stratégie pure : $\mathbf{x} = (4, -4)$.

4.4 Jeux bimatriciel à sommes nulles et résolution par PL

Maintenant que j'ai abordé la notion de stratégie, je m'intéresse 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 permettra de maximiser les gains du jeu.

Définition 9 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.

4.4.1 Explication de la démarche de résolution

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

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

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

π est une probabilité relative à une stratégie donc π_i^1 est la stratégie relative 1 en fonction de i (etc...), quand à a_{ij} il s'agit de la matrice (ligne i \times 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 :

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

π_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 (que nous savons résoudre). Pour cela je vais le transformer en posant une fonction v à maximiser et ce 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 seule contrainte :

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

Dans un jeu à somme nul 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 pour 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 :

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

4.4.2 Programme linéaire du Joueur A : maximiser v

Pour trouver la stratégie de JA il suffit de trouver une fonction à maximiser v tel que v soit plus petite que toutes les sommes que nous cherchons à minimiser, soit :

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

4.4.3 Programme linéaire du Joueur B : maximiser w

D'après le paragraphe précédent nous venons de calculer la stratégie du JA, afin de trouver celle du JB il suffit simplement de résoudre le problème dual soit :

$$\begin{aligned} & \text{minimiser } w \\ \text{s.c. } & w \geq \sum_j \pi_j^2 a_{ij} \quad \forall i \\ & \sum_j \pi_j^2 = 1 \\ & \pi_j^2 \geq 0 \quad \forall j \end{aligned}$$

4.4.4 L'équilibre de Nash dans Matching Pennies par PL

4.4.4.1 Matrice de gains des joueurs A et B

Le propre du jeu bimatriciel repose sur l'existence de la matrice de "gain" ou de "payoff". Je vais 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 ($B = -A$).

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

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

Dans un jeu à **somme nulle** étant donné que le gain de JA représente la perte équivalente de JB il n'existe pas d'équilibre de Nash) en stratégie pure mais donc en stratégie mixte.

4.4.4.2 Expression concrète du programme linéaire du Joueur A

Explications

4.4.4.3 Expression concrète du programme linéaire du Joueur B

Puisqu'il s'agit d'un jeu à somme nulle, le programme linéaire cherché n'est autre que le problème dual du programme linéaire précédent.

4.5 Jeux Stochastiques

4.5.1 Jeux dynamiques

Définition 10 *Un jeu est dit 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 joueurs JA et JB, ils vont effectuer tour à tour des actions en différé en ayant accès aux informations de l'autre.

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

4.5.2 Fonction de valeur

Le critère d'évaluation d'une stratégie π 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 de manière différenciée la valeur des gains futur. En notant r_t la récompense reçue à l'instant $t \in K$ et s^0 l'état initial, on définit la **fonction de valeur** du Joueur B pour tout $s \in S$ pour une stratégie π :

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

4.5.3 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.

4.5.3.1 Modèles des jeux stochastiques

On définit Φ un jeu stochastique fini à deux joueurs tel que :

- Un ensemble Ω 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 $\rho : I \times J \times \Omega \rightarrow \Delta(\Omega)$

4.5.3.2 Principes des jeux stochastiques

On part d'un état ω_1 donné et connu des deux joueurs (JA et JB), pour chaque étape $t \in \mathbb{N}$ on observe :

- JA et JB observe ω_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 ω_{t+1} attribué selon $\rho(i_t, j_t, \omega_t)$

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. En notant Π_j l'ensemble des stratégies possibles du joueur j , on peut adapter la notion d'équilibre de Nash aux jeux stochastiques.

4.5.3.3 Équilibre de Nash en jeux stochastiques

Définition 11 Une stratégie conjointe π^* 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 effectuées

Dans le cadre de mon stage, ayant pour but d'implémenter des modèles de jeux stochastiques, je suis parvenu au bout de dix semaines à résoudre, via la programmation linéaire, un jeu statique à somme nulle : **matching pennies**.

Le but était de me familiariser avec l'utilisation du solveur **gurobi** que ce soit pour écrire un script ou pour utiliser un fichier permettant la lecture d'un modèle (les deux ayant des syntaxes différentes) et assimiler les différents concepts de la **théorie des jeux** pour les résoudre.

Mon tuteur m'a conseillé de commencer par les **jeux à sommes nulles** puisque ces derniers sont plus faciles d'accès et de résolution que les autres types de jeux. Dans les semaines à venir je mettrai en place de le mettre en place, comme le titre de mon stage l'indique, des algorithmes pour résoudre des **jeux stochastiques**.

5.2 Difficultés rencontrées

Au cours de ces dix dernières durant lequel j'ai effectué mon stage et mon mémoire j'ai dû faire face à différentes difficultés et contraintes. Néanmoins je ne regrette pas du tout cette expérience puisqu'elle m'a été très bénéfique et enrichissante.

La crise sanitaire que nous traversons n'a pas du tout aidée. Le simple fait de ne pas pouvoir être au côté de mon tuteur pour lui poser des questions sur les outils utilisés, sur les différents concepts des **jeux** ou encore le simple fait d'être isolé et de ne pas avoir une dynamique de groupe ont été les réels difficultés que j'ai rencontrées. Je tiens à remercier tout particulièrement Monsieur **Hyon** pour sa disponibilité, ses précieux conseils et son accompagnement. Notre rendez-vous téléphonique hebdomadaire m'a permis de faire le point régulièrement et ainsi d'avancer.

Travaillant sur un système d'exploitation MacOS, les installations des outils que j'ai dû utiliser **Python** et **Gurobi** n'ont pas été aussi rapides et faciles comme indiqué sur les sites officiels respectifs. J'ai dû modifier les packages d'installation ainsi que les répertoires où sont enregistrés les différents fichiers (ou modules) pour y parvenir. De plus la version précédente de **gurobi** (9.0.1) n'était pas encore compatible avec ma version de **python** (3.8.3) qui évolue souvent. J'ai rencontré certaines erreurs lors de compilation de mes programmes python. Quelque temps plus tard la nouvelle version est sortie (9.0.2) et tous les "bugs" ont été corrigés.

Enfin l'utilisation d'un nouveau langage orienté objet **python** encore ainsi que le solveur **gurobi** permettant l'optimisation et la résolution de programme linéaire, m'ont demandé plusieurs heures de documentation (cf webographie [7]), mais aussi plusieurs heures d'exercices et de programmation. Néanmoins, cela m'a permis d'étendre mes horizons informatiques et m'a donné une meilleure approche sur mes connaissances acquises durant mon second semestre en recherche opérationnelle (RO).

5.3 Perspectives et améliorations

Après dix semaines de recherche et de travail, le projet ne peut être totalement terminé. Ainsi avec l'accord et la participation de mon maître de stage Monsieur Hyon je vais poursuivre la mission pour un résultat optimal et plus intéressant.

Dans un premier temps nous allons résoudre un autre exemple de **jeu à somme nulle**, autrement dit trouver, l'**équilibre de Nash** en stratégie mixte. Il faut pour cela modéliser la structure du jeu en **python**, puis à l'aide de **gurobi**, résoudre les programmes linéaires associés aux joueurs (à l'aide du théorème du *minimax*). Cela nous permettra de comparer les résultats avec ceux que nous avons obtenu avec notre jeu à somme nulle : **matching pennies**.

Nous réaliserons les mêmes travaux mais cette fois ci sur un autre type de jeux : les jeux **stochastiques**. Ici l'**équilibre de Nash** se trouvera différemment puisque nous utiliserons l'**algorithme de Shapley** qui avec l'utilisation de la programmation linéaire, une fois le code **gurobi** implémenté trouvera les valeurs de la stratégie mixtes des deux joueurs.

Finalement nous automatiserons les solutions des différents jeux. En fonction du jeu utilisé on aura la solution de l'**équilibre de Nash** pour chaque joueur. L'avantage principal de cette automatisation sera qu'en possession d'un code **python** implémentant un modèle de jeu (avec toutes les classes et fonctions nécessaires), notre programme sera à même de déterminer le type de jeu (**somme nulle, stochastique, bimatriciel** mais surtout des pouvoir le résoudre.

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

[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 Nanterre

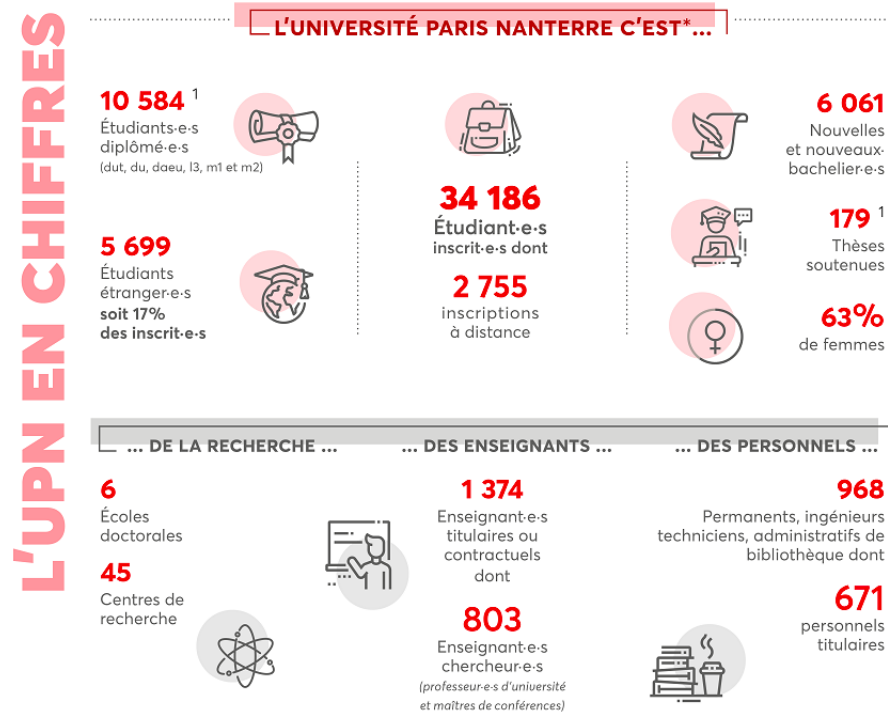


FIGURE 5 – L'Université Paris X en chiffres

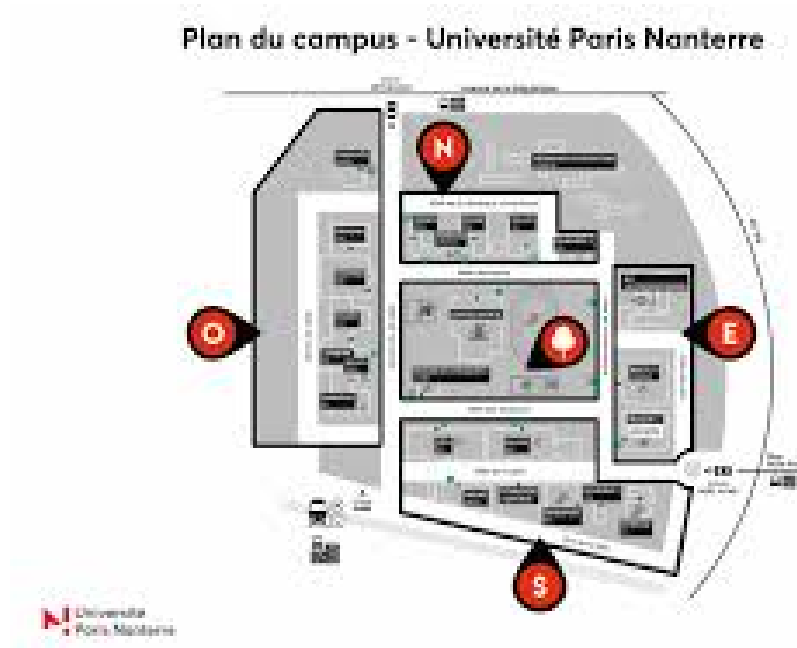


FIGURE 6 – Le plan de Paris X

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, une classe mère peut hériter d'une autre et donc de ses attributs et de ses méthodes. La syntaxe est simple, il suffit de mettre en parenthèse 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. Lorsque l'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.

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

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 LP nommé "biere.py" (voir exemple [3.3.3]) :

```

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 7 – 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 8 – 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 MIAGE** à l'Université **Paris Nanterre** en constante **recherche de connaissances et d'évolution** »

FORMATIONS

2019-2020 **Licence 3 MIAGE**

- Université Nanterre Paris Ouest

2018-2019 **Licence 2 MIASHS / MIAGE**

- 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 9 – Curriculum Vitae Avi ASSAYAG L3 MIAGE

Table des figures

1	Le campus de Nanterre	7
2	Shell interactive Gurobi	16
3	Represenation matricielle de "Matching Pennies"	23
4	Représentation matricielle de "Bataille de réseaux"	24
5	L'Université Paris X en chiffres	32
6	Le plan de Paris X	32
7	Résolution via shell gurobi (gurobi.sh)	36
8	Résolution via module gurobi (gurobipy)	36
9	Curriculum Vitae Avi ASSAYAG L3 MIAGE	37