Mathieu MARÉCHAL

Simulation Numérique II

Nom de code : Dém***ez-vous !

Compte-rendu des projets

Encadrants : C. Desjouy, S. Raetz

L2 SPI

2018 - 2019

Introduction

Table des matières

Projet 1 La balle

L'objectif de ce projet est de modéliser la trajectoire d'une ou plusieurs balles en prenant en compte les rebonds. La consigne divise le travail en plusieurs étapes qui consistent à, tout d'abord, calculer et animer la trajectoire sans rebonds d'une balle, puis de prendre en compte les rebonds. Enfin, le programme nécessite d'être capable de calculer plusieurs trajectoires à la fois. J'ai donc décidé de procéder comme cela, en rajoutant des étapes : j'ai tout d'abord cherché à calculer une trajectoire simple et sans animation. Ensuite, j'ai créé les codes successifs dans des fichiers séparés, en rajoutant successivement : les animations, les rebonds, et enfin, plusieurs balles.

1.1 Calculer la trajectoire

Tout d'abord, les premières constantes dont va dépendre la trajectoire sont crées :
— l'angle initial, θ_0 nommé `a_0`
— la vitesse initiale v_0, notée `v_0`
— plus tard, le temps de départ t_0 (`t_0`) sera utilisé
Ensuite, on implémente les équations déterminant la position du projectile en fonction des constantes précédentes :

$$\begin{cases} d_x = & v_0 t cos(\theta) + d_{0_x} \\ d_y = & -\frac{1}{2} g t^2 + v_0 t sin(\theta) + d_{0_y} \end{cases}$$

Il est maintenant possible de calculer la trajectoire mais la première question qui se pose est : quand doit-on arrêter de tracer la trajectoire ? En effet, si on trace la trajectoire sur un temps `t_balle` fixé, on observera la trajectoire avoir une position d_y négative, avec une pente très élevé, ce qui n'a pas du tout la forme d'une parabole. C'est le premier problème que j'ai rencontré. C'est donc quand le projectile touche le sol qu'il faut arrêter de calculer la trajectoire, quand sa position en y vaut $0 \Leftrightarrow d_y = 0$. Pour cela, il faut calculer le temps de course du projectile, en résolvant l'équation $d_y = 0$. Il est alors possible de désigner une variable `t_max` qui calcule cette solution, ce qui permet de définir la limite du vecteur de temps `t_balle`.

```
t_max = (2 * v_0 * np.sin(a_0)) / g + t_0
t_balle = np.arange(t_0, t_max, 0.1)
```

Par la suite, on peut animer la trajectoire du projectile. Pour cela, c'est la `FuncAnimation` du module `matplotlib.animation` qui sera utilisée, car elle utilise une fonction pour animer les données calculées et non une boucle `for` comme utilise l'`ArtistAnimation`. Ce choix est justifié aussi par le fait que le programme sera mis sous la forme d'une classe par la suite.

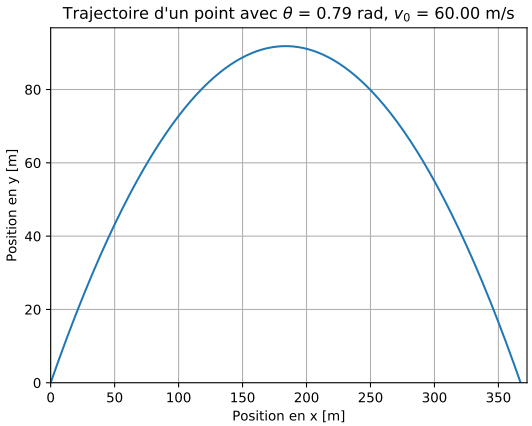

FIGURE 1 – Trajectoire fixe d'une balle

1.2 Ajouter les rebonds

Implémenter les rebonds, paraît assez simple et c'est le cas ! Il suffit de déterminer une nouvelle vitesse initiale à la fin du calcul de la trajectoire et d'en calculer une nouvelle, ce qui est très aisé avec une boucle `for` itérant pour le nombre de rebonds souhaités.

Ainsi, on met sous cette boucle la totalité de la section du code réservé au calcul de la trajectoire développé précédemment. A la fin de la boucle, la vitesse initiale est multipliée par un coefficient de restitution r, `v_0 = r*v_0` caractérisant la rigidité du sol, et dans l'ensemble, l'élasticité du rebond, c'est-à-dire la quantité d'énergie de la balle absorbée par le sol. La vitesse de la balle va donc diminuer au fur et à mesure des rebonds successifs. L'autre donnée à conserver lors du calcul du rebond est la position horizontal de la trajectoire précédente. On stocke donc la dernière position en x telle que : `d_0_x = d_x` et on ajoute `d_0_x` en offset au calcul de la position.

Bug rencontré

Néanmoins, dans cette partie du développement de ce programme, des difficultés ont été rencontrées. D'après la figure 2, on voit que la forme de la trajectoire correspond vaguement à la figure recherchée, mais l'amplitude de la trajectoire sur l'axe y diminue très rapidement et la balle part dans les y négatifs.

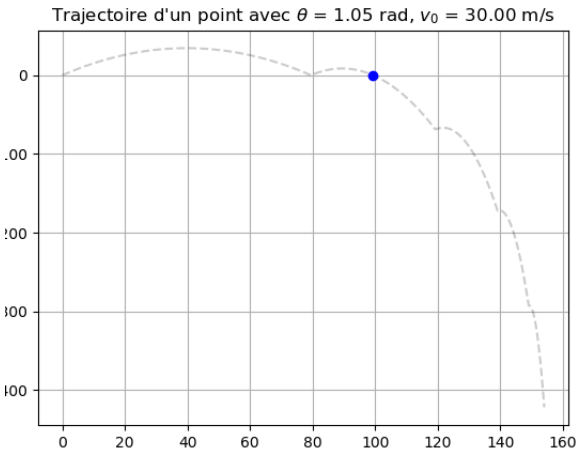

FIGURE 2 – Bug rebond balle

Cependant, il est possible de remarquer très vite le problème : le temps de la trajectoire `t_max` est le même pour chaque rebond. En effet, cette variable était déterminé au début du programme et faisait partie des conditions initiales. Cependant, comme vu précédemment, `t_max` dépend de l'angle initiale, mais surtout de la vitesse initiale. Comme la vitesse varie après chaque rebond, il faut recalculer `t_max` après chaque rebond.

Notre balle a donc obtenu la capacité de rebondir n fois, mais elle se sent toujours très seule. Nous allons donc adapter le programme pour pouvoir en afficher plusieurs en simultanée.

1.3 Plus de balles

Dans cette partie, l'objectif est de pouvoir modéliser la trajectoire de **n** balles en simultané en définissant des constantes initiales aléatoires. Pour cela, il d'initialiser toutes ces conditions en leur donnant une valeur aléatoire avec le module `random`.

Afin de simplifier le programme dans son ensemble et l'implémentation de cette fonctionnalité, le code sera mis sous la forme d'une classe. La première version de ce programme est `class_multiball.py` : ce programme contient une classe "Balle" avec une méthode `setup` qui calcule la trajectoire de toutes les balles et une méthode `draw` qui les anime sur une figure. Cette syntaxe n'est cependant pas très cohérente et

```
class Balle_1():
    def setup(self):
        # calcule les trajectoires de n balles
    def draw(self):
        # trace toutes les balles

b = Balle_1()
b.setup()
b.draw()

#==========

class Balle_2():
    def __init__(self):
        # conditions initiales
    def get_tmax(self):
        # retourne le temps de course de la balle
    def change_speed(self):
        # fait diminuer la vitesse au moment du rebond
    def get_x(self, t):
        # retourne la position en x de la balle    t secondes
    def get_y(self, t):
        # retourne la position en y de la balle    t secondes
    def get_parab(self):
        # retourne une liste contenant les positions pour une premi re parabole
    def get_rebonds(self):
        # retourne la trajectoire compl te de la balle
    def get_traject(self):
        # adapte la liste de la m thode get_rebonds pour que toutes les
        # instances de la classe Balle_2 aient la m me longueur

b = Balle_2()
b.get_traject()
# Animation
```

FIGURE 3 – Comparaison des 2 versions du programme animant plusieurs projectiles

n'optimise pas les possibilités d'une classe. De plus, chaque méthode exécute un grand nombre d'action au lieu d'une seule.

Ainsi, une deuxième version a été créée : `class_multiball_2.py`. Ce programme contient une classe qui renvoie seulement une liste contenant les positions décrivant la trajectoire d'une balle. Le tracé est effectué à l'extérieur de la classe et fait bien retourner à la classe un objet qui est une unique balle.

Ce second programme m'a permis de bien comprendre le fonctionnement d'une classe : ici, la classe représente le "comportement" d'une balle. Néanmoins, l'animation est identique pour les 2 programmes, le seule changement étant la lisibilité et la logique derrière le calcul.

Projet 2 Le tube de Kundt

Projet 3 L'interface du reste de ma vie

Pour ce troisième projet, nous allons créer une interface permettant de charger un fichier généré par le logiciel *INTAC* du CTTM et d'afficher les résultats. Pour cela, il s'agit de décomposer le travail en plusieurs étapes : créer les fonctions pour charger un fichier texte, puis le tracer, et ensuite créer une interface utilisateur avec des boutons et un espace d'affichage afin de pouvoir visualiser la courbe et interagir avec pour modifier par exemple l'affichage ou le type de tracé (temporel ou fréquentiel).

La première étape est assez simple : demander à l'utilisateur le chemin du fichier à charger et le tracer demande seulement quelques lignes en utilisant l'instruction `data = np.loadtxt(filename)` où `filename` est une chaîne de caractères contenant le chemin du fichier texte à charger. Ensuite, il suffit de créer une figure avec matplotlib pour la tracer.

La seconde et principale étape est plus délicate : il faut utiliser un module permettant de créer une interface graphique pour qui utilise ces fonctions. Il existe de nombreux modules qui permettent de créer des GUI (graphical user interface) tels que Tkinter, PyQt, Kivy ou beaucoup d'autres. Mon choix s'est porté sur le module PyQt pour sa simplicité à utiliser avec des classes, et pour le fait qu'il permette de faire plus simplement une belle interface que Tkinter, qui est assez brut. La difficulté de ce module reste tout de même que la documentation est en C et non en python, mais de nombreux exemples et tutoriels existent pour prendre en main ce module. Enfin, il a été choisi car il comprend une intégration plutôt simple de Matplotlib, que ce soit pour la figure ou la barre d'outils.

Le programme de l'interface est donc construit sous la forme d'une classe "GUI", initialisée avec des variables précisant la géométrie de la fenêtre, son titre, icône et des variables utiles au fonctionnement des différentes fonctions de l'interface. De façon générale, l'initialisation de cette classe doit comprendre tous les objets que l'on veut afficher dans l'interface. La création des objets et la disposition de ces derniers sont donc séparés dans 2 méthodes différentes appelées dans l'initialisation. Cette classe est de type `QDialog`. Il y a plusieurs types de classe pour créer une interface dont : `QWidget`, `QDialog`, `QMainWindow`. On utilise l'une ou l'autre de ces classes selon l'interface à créer :

- `QMainWindow` permet la création d'une barre de manu, d'une barre d'outils et d'autres objets dont une fenêtre principal peut avoir besoin.
- `QDialog` est plus utile pour une petite interface. Elle est basée sur `QWidget` et est designée pour apparaître comme une fenêtre. De plus, elle a des fonctions qui rendent la création de boutons assez basique simple et bien intégrée à la fenêtre.
- Enfin `QWidget` est la classe de base pour tout objet affichable sous Qt. Quand un objet widget n'a pas de fenêtre parent, il peut être affiché comme une fenêtre.

Le choix d'utiliser `QDialog` est relativement arbitraire mais est justifié par le fait que pour le développement de cette interface, il nous faut avoir la possibilité de faire afficher une fenêtre "enfant" pour afficher l'arborescence du disque et choisir un fichier à tracer au lieu de devoir chercher le chemin de chaque fichier, c'est l'objectif premier de cette interface. Il n'est pas possible d'ouvrir une boîte de dialogue avec `QWidget`. Enfin, `QMainWindow` paraît un peu exagéré : une barre d'outils/ de menus est inutile pour une interface de cette taille.

Projet 4 Traitement du signal et imagerie

Projet 5 Optique géométrique