Faculté sciences et ingénierie Université de Toulouse

LICENCE 3 DE MÉCANIQUE - ÉNERGÉTIQUE

Année 2022-2026

Projet suivi en Informatique Scientifique.

Etude de balistique, partie 1 : modèle analytique

Compétences et capacités visées

- Résoudre un problème scientifique par codage propre (suivant la PEP 8 et PEP 20)
- Comprendre un énoncé de problème
- Appliquer des méthodologies proposées tout en gardant un esprit l'initiative
- Coder en Python en autonomie
- Organiser son temps de travail pour accomplir un projet dans le délai imparti

1 Déroulement

Le projet est planifié sur 4 séances de TP (Tableau 1), mais un travail doit être accompli entre chaque séance et il sera vérifié.

n° de séance	Travail
8	Introduction aux classes, CC1
9	Modèle analytique
10	Modèle numérique
11	Modèle numérique, étude paramétrique

TABLE 1 – Déroulement du projet par séance de TP

⇨ Le projet est une bonne illustration de l'intérêt de la programmation orientée objet (POO)

⇨ Les séances sont cadencées, et donc il faut finir à la maison de travail d'une séance à l'autre. Ce sera évalué.

⇨ Il faut conserver les choix de noms fournis dans les morceaux de code, afin de faciliter la notation du travail.

⇨ Il faut respecter au mieux la recommandation d'écriture **PEP 8** et **PEP 20**

⇨ L'ensemble du projet se fera dans un IDE : spyder, vsc, ou pycharm. Jupyter notebook interdit.

⇨ Le travail peut être effectué en binôme ou seul. Mais la note restera individuelle, et il pourra y avoir des écarts importants entre membres d'un même binôme.

⇨ A chaque séance des objectifs à atteindre pour la prochaine séance sont définis.

⇨ Les séances permettent de présenter les différents éléments à implémenter, de discuter des les difficultés, de résoudre les problèmes rencontrés.

Phase	Contenu	Modèle	Temps (h)	Séance	Points
1	Apprentissage classe, modèle basique	1	1	8	0
2	Ajout graphiques, trajectoires en fonction de α	1	2	9	20
3	Etude paramétrique, portée optimale	2	2	9	10
4	Apprentissage : résolution ODE du problème basique, validation	1	2	10	0
5	Ajout portance et traînée, propulsion	3	2	10 -11	20
	Etude paramétrique, trajectoires, isocontours	2	2	10 -11	20
	Travail dans les séances (et présence)				20
	Travail hors séance		6	A	20
	Rapport		3	A	10
Total			**20**		100

TABLE 2 – Les phases du projet, les durées, exemple de notation ; A : en autonomie

2 Introduction

Les études de balistique c'est-à-dire l'étude du mouvement d'un objet lancé dans un champ de pesanteur sont très anciennes (voir wikipédia).

On se propose ici de coder quelques modèles, progressivement, et de sortir des résultats dans des graphiques en utilisant les modules spécifiques de Python.

Scientifiquement le problème est de niveau Licence 1, en mécanique du point. On peut se contenter de prendre les équations, mais il vaut mieux comprendre la physique, très simple ici.

3 Evaluation du projet

La durée estimée du projet est d'environ 20h, dont 7 h en présentiel, et le reste en autonomie.

L'évaluation prendra en compte les éléments suivants :

- les codes produits
- le travail en séance
- le travail hors séance, en autonomie
- le rapport

Le projet est découpé en phases, chacune étant plus ou moins évaluée en fonction des trois éléments suivants :

- les fonctionnalités ajoutées pour 60 %
- le suivi des règles du codage propre, rigueur, pertinence des choix pour 30 %
- les commentaires écrits dans le rapport et la présentation des figures.

Enfin on peut ajouter quelques remarques concernant l'évaluation :

- La non adéquation évidente entre le travail mené en séance et le travail rendu entraînera une forte pénalisation de la note.
- En cas de plagiat ou triche avérée, la note de 0 / 20 sera donnée.
- Enfin toute absence pénalisera fortement la note car l'activité effective ne pourra être vérifiée.

Le contenu du rapport fait l'objet d'une section spéciale en fin de la la partie 2 du projet.

Rappelons enfin que dans cette première partie, seul le problème avec une solution analytique sera traité.

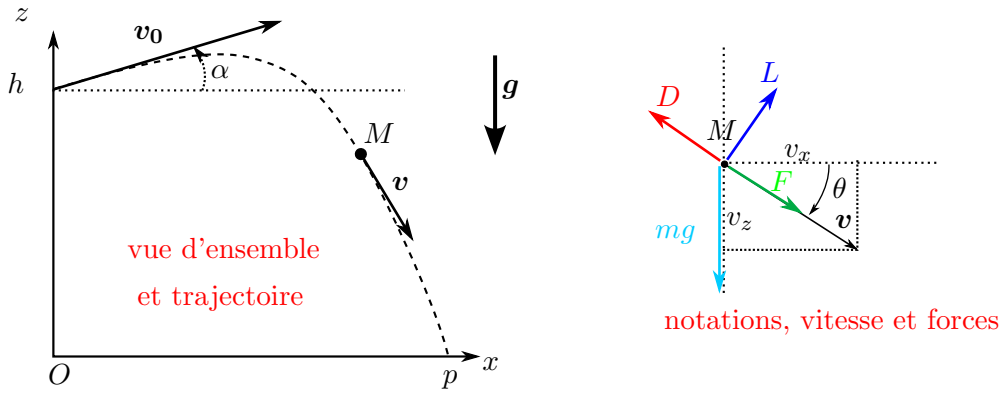

FIGURE 1 – Schéma et vecteur vitesse.

4 Généralités sur la configuration

La figure 1 représente d'un part la trajectoire avec les notations, d'autre par le vecteur vitesse et les forces en présence, certaines étant supposées nulles dans cette partie.

On considère le mouvement d'un objet de masse m, dans le plan (x, z), soumis à un champ pesanteur $\boldsymbol{g}$. Il est lancé à une altitude h, avec une vitesse initiale $\boldsymbol{v_0}$ orientée d'un angle α par rapport à $\boldsymbol{e_x}$. Il tombe sous l'effet de la pesanteur, avec une accélération $\boldsymbol{a}$ et une vitesse $\boldsymbol{v}$ qui sont calculées par application du principe fondamental de la dynamique (PFD) [1].

Cette description correspond au modèle de référence, dit aussi **modèle analytique**, ou modèle 1.

On peut y ajouter, dans la réalité les éléments suivants :

- une force de traînée
- une force de portance
- une force de propulsion

On parle alors du **modèle complet** qui n'a pas de solution analytique. Cela fera l'objet de la seconde partie du projet.

5 Objectifs pour le modèle analytique

On peut se donner à minima deux objectifs principaux, qui se traduisent par des données de sorties du code :

- calculer les données à l'impact (référées par l'indice i) : la portée $p = x_i$ qui est la distance franchie lorsque l'objet touche le sol, le le temps d'impact t_i, la vitesse d'impact $\boldsymbol{v_i}$, son module, l'angle d'impact θ_i, et enfin l'accélération $\boldsymbol{a_i}$.
- déterminer la hauteur maximale de la trajectoire $h_{\max} = h_1$ qui arrive au temps t_1 et la vitesse à cette hauteur $\boldsymbol{v_1}$.

6 Les paramètres d'entrée du problème

Le problème, pour être résolu correctement, nécessite la donnée des paramètres suivants :

- le module de la vitesse initiale v_0 et l'angle α
- l'altitude initiale h
- la valeur de la gravité g

1. connu sous le nom de la seconde loi de Newton.

7 Relations utiles pour le modèle analytique

Ce modèle est le **modèle 1**.

C'est le modèle classique de mécanique du point, où on néglige les forces de frottement, et seule l'accélération de la pesanteur est prise en compte. C'est un modèle purement analytique, simple, mais qui sert de référence dans tout le projet. On se contente de donner ici les résultats :

Equations

- L'accélération et les deux composantes de la vitesse sont données par :

$$a = -g, \qquad v_x = v_0\ \cos\alpha, \qquad v_z = v_0\ \sin\alpha - g\ t, \qquad v = \sqrt{v_x^2 + v_z^2}$$

- La position est répérée par :

$$z = -\frac{g}{2}\ t^2 + v_0\ \sin\alpha\ t + h, \qquad x = v_0\ \cos\alpha\ t$$

- En substituant $t = \dfrac{x}{v_0\ \cos\alpha}$ dans $z(t)$ on obtient la **trajectoire parabolique** :

$$z = -\frac{g}{2}\ \frac{x^2}{v_0^2\ \cos^2\alpha} + \tan\alpha\ x + h$$

- Le point d'impact est défini à partir de son temps t_i et de la portée $p = x_i$ avec $z_i = 0$:

$$t_i = \frac{v_0\ \sin\alpha + \sqrt{(v_0\ \sin\alpha)^2 + 2g\ h}}{g}, \qquad x_i = v_0\ \cos\alpha\ t_i, \qquad \theta_i = \arctan\frac{v_z(t_i)}{v_x(t_i)}$$

  On a l'angle de l'impact θ_i et on peut facilement avoir le module de la vitesse.

- La **portée maximale** peut s'écrite sous la forme :

$$p_{\max} = \frac{v_0^2}{g}\ \left(\frac{\sin 2\alpha}{2} + \cos\alpha\ \sqrt{\sin^2\alpha + \mathcal{L}}\right), \qquad \mathcal{L} = \frac{2g\ h}{v_0^2}$$

- On peut remarquer que la portée sera maximale pour la valeur de α qui vérifie :

$$f(\alpha) = 0 = \cos 2\alpha + \frac{\sin\alpha\ \ (\cos 2\alpha - \mathcal{L})}{\sqrt{\sin^2\alpha + \mathcal{L}}} \tag{1}$$

  On retrouve le résultat connu de l'angle optimal de 45° si $h = 0$.

  Sinon l'angle est inférieur à 45°.

- On sait que la trajectoire est une parabole et son apogée est atteinte quand $v_z = 0$. On a alors :

$$t_1 = \frac{v_0\ \sin\alpha}{g}, \qquad v(t_1) = v_x, \qquad h_1 = z(t_1)$$

Ces relations peuvent être très facilement obtenues à la main ou à partir du module **sympy** de Python [2].

2. Il n'est pas demandé de les vérifier dans le cadre du projet

8 Phase 1 : apprentissage des classes.

Durée : 1 heure

8.1 Quelques généralités

Les classes sont employées dans la **Programmation Orientée Objet** (POO). De nos jours il devient très difficile de s'intégrer dans un projet de recherche ou industriel, où on doit utiliser ou créer du code, sans connaître les bases de la POO.

Un classe est un objet Python qui sert de modèle pour créer d'autres objets de même type. C'est ce qu'on appelle un **conteneur**. Le contenant est constitué d'attributs et de méthodes qui sont discutés juste après.

Nous allons continuer l'apprentissage démarré dans la dernière séance de travaux pratiques de ce cours.

Ici on va définir une classe pour le modèle 1 avec 3 paramètres d'entrée `v_0`, `h` et `alpha`.

L'angle α est en degrés et les autres grandeurs sont dans les unités SI.

Préparation

Le premier travail, avant de venir à la première séance, est de lire ce document, de relire la partie du cours sur le Python, le POO, les recommandations PEP et de consulter les documents mis à votre disposition sur Moodle.

Notamment sur la partie Moodle affectée à ce projet, se trouve une fiche d'apprentissage sur les classes en python et la POO, du niveau lycée, très bien faite.

Important

- La classe est définie par un nom avec l'instruction `class`.
- Le mot `self` fait référence à la classe elle-même, il transporte (on parle d'encapsulation) toutes les variables (appelées attributs) et toutes les fonctions (appelées méthodes) qui sont incluses dans la classe.
- L'utilisation du mot `self` est très pratique puisque cela évite d'écrire à chaque fois un grand nombre d'arguments dans les méthodes et d'avoir accès très facilement aux divers éléments de la classe.
- Les attributs agissent comme des variables globales interne à la classe. Ils sont identifiés par un nom commençant par `self.nom_attribut`

8.2 Exemple de la création d'une classe

Dans le projet **balistique**, on a créé un module `Ballistic` (balistique en anglais) et dans celui-ci, le sous-module `analytical_model_example.py` contenant la classe `AnalyticalModelExample`

Attention

Le premier travail consiste à lire attentivement et dans le détail ce module.
Toutes les lignes de code doivent être comprises.

Il servira d'exemple tout au long du projet On rappelle ici les points importants à retenir :

- Le constructeur `__init__` sert à introduire les attributs lors de l'utilisation de la classe
- La méthode `message_initial` ne contient pas de paramètre de la classe. On dit que c'est une méthode statique.

- On saute 1 ligne entre les méthodes dans la classe et 2 lignes avant la commande `class`
- La méthode `set_velocity` permet de calculer la vitesse en fonction du temps t qui est un paramètre de la méthode mais qui n'est pas un attribut de la classe et elle retourne un tuple de 3 éléments.
- A quelques exceptions près, non discutées ici, il faut que le premier argument à la création d'une méthode soit `self`. On accède ainsi à tous les éléments de la classe dans cette méthode.
- Par contre quand on utilise la méthode on ne met pas le premier argument `self`. Il est compris dedans dans la programmation du langage.
- la méthode `set_impact_values` permet de calculer les variables d'impacts, qui ressortent sous la forme d'un dictionnaire.
- Pour appeler une méthode interne à la classe il faut ajouter `self.` devant.
- On remarque l'utilisation des fonctions `sqrt, sin, cos, arctan, deg2rad, rad2deg` du module `numpy` (mettre `np.` devant)
- Dans les classes on fait la différence entre les "setters" ou **mutateurs** en français, des méthodes qui fixent des choses (calculs, etc...) et les "getters" ou **accesseurs** en bon français, qui permettent de faire sortir des attributs ou des données en dehors de la classe, notamment pour les afficher ou les réemployer ailleurs.
- Très souvent les méthodes de type "setters" commencent par le mot `set_` et les méthodes de type "getters" commencent par le mot `get_`. Cela permet de déclarer **son intention** par rapport à la méthode considérée.
- Des doctrings sont ajoutées après la définition des méthodes. Normalement elles devraient être en anglais. Sans être obligatoire, c'est plutôt conseillé de la faire.

8.3 Exemple d'utilisation d'une classe

Dans le projet, les classes sont appelées dans des scripts (modules) principaux qui commencent toujours par le mot `main_`.

L'utilisation de la classe précédente se trouve dans dans le script `main_analytical_model_example.py`

L'ouvrir et le lire attentivement. Les commentaires permettent normalement de comprendre le sens des différentes lignes de code.

Quelques points sont encore à retenir quant à l'utilisation d'une classe :

1. Il faut d'abord l'importer
2. Pour modifier les attributs de la classe on a ici montré deux approches :
   (a) En modifiant directement l'attribut de la classe. Mais ce n'est pas ce qui est préconisé dans la PEP 20
   (b) En ajoutant une méthode dédiée pour mettre à jour les paramètres, via un dictionnaire (méthode `update_parameters`)

   La seconde approche est plus propre et plus claire dans l'**intention** pour l'utilisateur.

8.4 Premières conclusions

- Il faut bien faire la différence entre attributs et méthodes
- Il faut bien faire la différence entre écrire du code dans une classe et utiliser une classe
- Il faut faire la différence entre un "setters" et un "getters" (entre le calcul et l'affichage ou l'utilisation du résultat)
- Il faut toujours présenté des sorties formattées

- Dans l'écriture de la classe, il faut mieux rassembler les éléments de type SETTERS, GETTERS, et PLOTS ensemble, toute chose doit être à la bonne place.

9 Phase 2 : création de la classe `Model_1`

Durée : 2 à 4 heures

9.1 Copie de la classe `AnalyticalModel`

Cette classe est une extension de la classe précédente, mais pour faire simple, on doit recopier la classe précédente dans le nouveau module `model_1.py` et la renommée en `Model_1`. On s'inspirera du fichier principal qui existe déjà.

Une fois celà fait on peut la modifier et la tester dans le nouveau script principal `main_model_1.py` en implémentant progressivement les éléments qui suivent.

9.2 Implémentation de la trajectoire et tracé

Attention

Il faut penser à sauvegarder les figures au format png pour les inclure dans le rapport [a]

Toute copie d'écran de mauvaise qualité sera fortement pénalisante.

a. `plt.savefig()`.

1. Ajouter une méthode `set_trajectory` qui calcule la trajectoire entre les temps $t = 0$ et $t = t_{end}$ (`t_end`). Cette méthode génère 3 nouveaux attributs sous la forme de vecteurs `numpy` : `self.time, self.x, self.z`. Le vecteur temps est généré par `np.linspace` avec le paramètre `npt` pour définir le nombre de points. La méthode `set_trajectory` prend donc au moins trois paramètres : `self, t_end, npt`

   Il n'y a pas de sortie direct avec `return`, puisque les sorties sont des nouveaux attributs de la classe.

2. Ajouter une méthode `plot_trajectory` qui trace la trajectoire sur la base des attributs de classe `x` et `z`. La figure doit comporter un titre, les labels sur les axes, et une trajectoire de couleur rouge d'épaisseur 3 points (`linewidth=3` ou `lw=3`). On doit avoir pour les labels et les légendes `fontsize=12`.

   Ajouter une grille aussi. On pourra vérifier sur la figure que la portée est correcte.

   Les recommendations graphiques doivent être conservée pour toutes les figures du projet et ses morceaux de code pourront être recopiés plus tard..

3. Ajouter une méthode `plot_components` qui trace deux cadres l'un au dessus de l'autre, avec sur l'axe des abscisses le temps et sur l'axe des ordonnées $v_z(t)$ et le module $v(t)$. On utilisera les fonctions `subplots` et `fill_between` pour faire joli. Pour cela il faut probablement modifier la méthode `set_trajectory` pour récupérer $v_z(t)$ et $v(t)$.

   Le résultat doit correspondre plus ou moin à la figure 2 (qui est améliorable).

10 Phase 3 : tracé de trajectoires multiples et angle optimal

Suivant le temps ce travail s'inscrira en travail personnel ou dans la seconde séance.

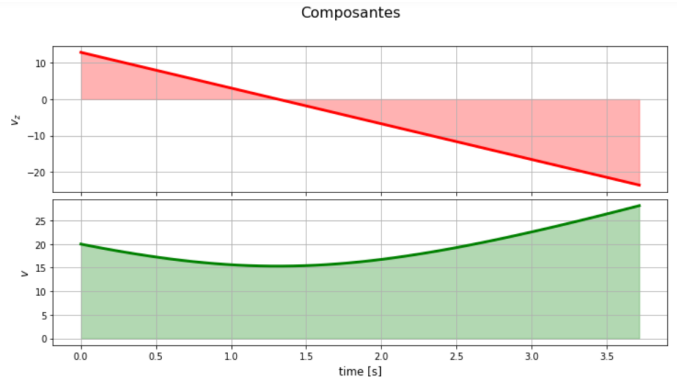

FIGURE 2 – Evolution de v_z et v en fonction du temps.

10.1 Graphiques de trajectoires

Cette étape consiste à représenter plusieurs trajectoires sur un même graphique en ayant modifier l'angle α qui sert de paramètre principal, et qui varie de 20 à 70 °par pas de 5°. Pour cela il faudra modifier le script principal `main_model_1.py`.

On peut ajouter dans ce script principal une variable locale `task` qui suivant sa valeur lancera différentes phases de cette partie du projet.

Il existe plusieurs façon de procéder. Ici il faudra suivre la démarche suivante [3].

- Créer une méthode `set_trajectories(alpha)`. Cette méthode permettra de récupérer une liste des trajectoires pour une liste de α. On y trouvera une boucle en `alpha` modifiant l'attribut de même nom dans la classe `Model_1`.
- Créer une méthode `plot_trajectories(alpha, x, z)`. Cette méthode permettra de tracer sur un même graphique, toutes les trajectoires avec une légende indiquant la valeur de l'angle α
- Ajouter le nécessaire dans le script pour utiliser correctement ces deux méthodes et modifier ou ajouter ce qu'il faut pour remplir l'objectif dans la classe `Model_1`.

On doit trouver quelque chose qui ressemble à la figure 3, améliorable.

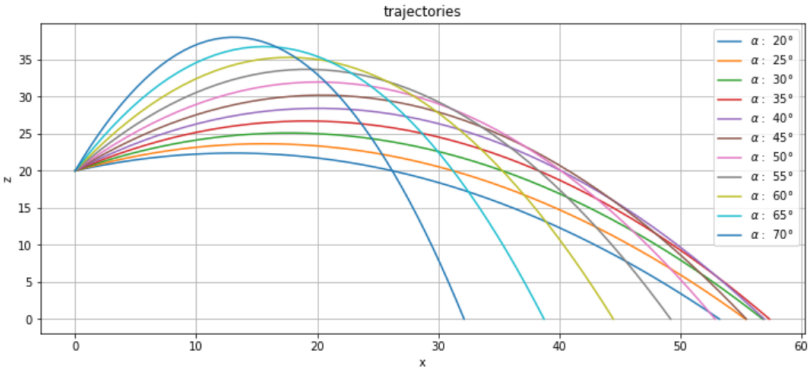

FIGURE 3 – Différentes trajectoires en fonction de α.

10.2 Portée maximale

Dans le même script principal, ajouter une autre méthode qui générère une seconde figure. Elle fournie la portée maximale $p_{\max}$ en fonction de l'angle initial α. Le résultat typique est donné sur la figure 4.

3. à modifier dans `main_model_1.py` et ne pas confondre avec des méthodes qui portent un nom voisin dans la classe `Model_1`.

On en déduira, par un affichage de qualité, la portée maximale pour une vitesse de $v_0 = 20$ m/s et $h = 20$ m ainsi que l'angle optimal α_{opt} déterminé simplement à partir des données générées.

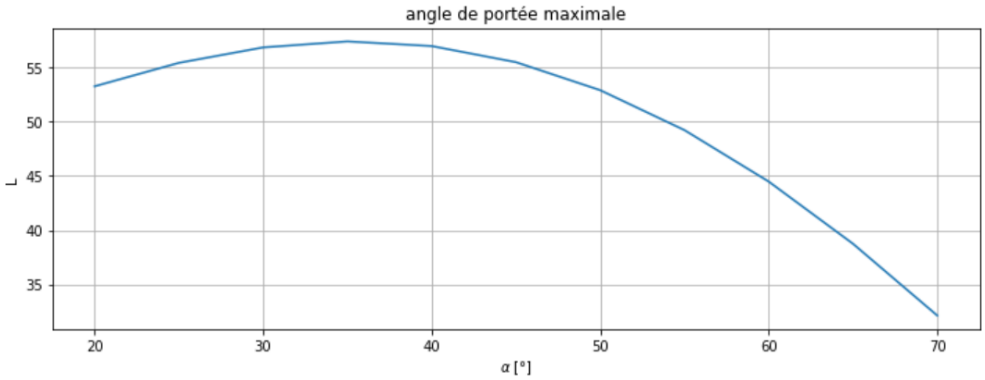

FIGURE 4 – Portée optimale en fonction de α

10.3 Calcul précis de l'angle optimal

Il faut maintenant travailler dans la classe `Model_1` :

1. Rajouter une méthode qui représente le calcul de la relation (1).
2. Rajouter une méthode qui résoud de façon assez précise $f(\alpha) = 0$ et retourner l'angle optimal α_{opt}. L'approche la plus simple consiste à générer dans un intervalle constitué de 1001 points, de faible longueur, les valeurs de $f(\alpha)$ et de rechercher la valeur de α qui fournit une fonction la plus proche de zéro [4].
3. rajouter une fonction qui permet de tracer $f(\alpha)$ entre deux bornes qui sont données en paramètres.

Rajouter dans le script principal précédent une tâche et des lignes de code qui permettent de tracer la fonction et l'affichage soigné de l'angle optimal, et la portée optimale associée à cet angle.

10.4 Altitude maximale

Il est probable que cette partie doive être faite en autonomie, en dehors des séances en présentiel.

Dans la classe, rajouter une méthode qui permet de calculer l'altitude maximale atteinte ainsi que le temps correspond et une méthode qui permet de calculer cette altitude maximale en fonction de α.

Rajouter une tâche et ce qu'il faut dans le script principal pour tester.

Optionnellement, les meilleurs étudiants créeront un module spécifique qui permet d'effectuer la phase 3, et un script principal simple qui permet d'utiliser ce module.

4. Il existe aussi une méthode dans numpy permettant de faire correctement le travail.