Faculté sciences et ingénierie Université de Toulouse

LICENCE 3 de Mécanique - Énergétique
Année 2022-2026

Projet suivi en Informatique Scientifique.

Etude de balistique, partie 2

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.

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.

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

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.

Dans cette partie on résoud le problème qui se réduit à une solution numérique d'un système d'équations aux dérivées ordinaires (EDO en français, ODE en anglais). On va donc traiter le **problème complet**.

3 Présentation du problème complet

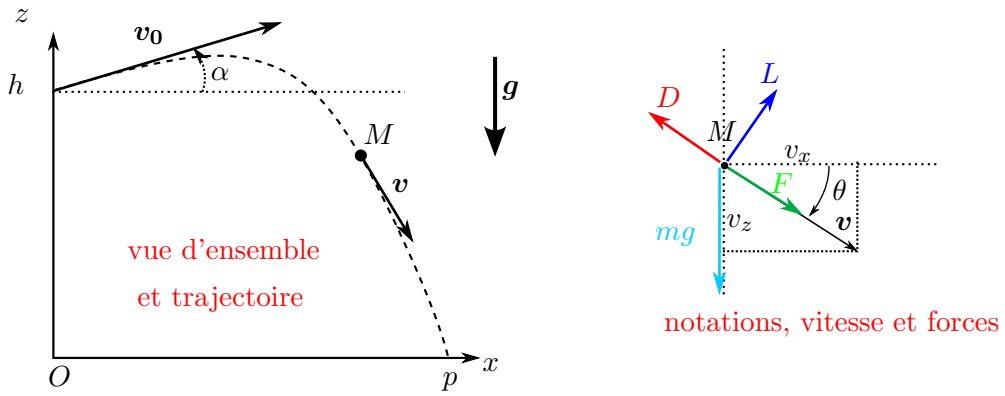


FIGURE 1 – Schéma et vecteur vitesse.

La figure 1 représente d'un part la trajectoire avec les notations, d'autre par le vecteur vitesse et les forces en présence.

On considère le mouvement d'un objet de masse m, dans le plan (x, z), soumis à un champ de gravité $\boldsymbol{g}$. 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 dans le champ de gravité avec une vitesse $\boldsymbol{v}$ et une accélération $\boldsymbol{a}$ qui sont calculées par le principe fondamental de la dynamique.

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. On en reparlera plus tard.

Pour valider la démarche on va d'abord travailler sur le cas identique au cas analytique, à savoir sans forces de traînée, de portance ou de propulsion.

Ce modèle est appelé le **modèle 2**.

4 Phase 4 : apprentissage du modèle sans force autre que la gravité, classe `Model_2`

4.1 Modèle vectoriel

Les équations de ce modèle sont écrites sous la forme d'un système différentiel :

Equations

$$\frac{d\boldsymbol{y}}{dt} = \boldsymbol{f}(t, \boldsymbol{y}) \qquad \Longrightarrow \qquad \frac{d}{dt}\begin{bmatrix} x \\ z \\ v_x \\ v_z \end{bmatrix} = \begin{bmatrix} v_x \\ v_z \\ 0 \\ -g \end{bmatrix}$$

On peut facilement identifier les composantes du vecteur $\boldsymbol{y}$ et de $\boldsymbol{f}$ en prenant les indices de vecteur conformément au Python en démarrant à 0 :

$$y_0 = x, \qquad y_1 = z, \qquad y_2 = v_x, \qquad y_3 = v_z$$

Les conditions initiales pour ce problème sont :

$$\boldsymbol{y}(0) = (0,\ h,\ v_0 \cos\alpha,\ v_0 \sin\alpha)^T$$

Ce modèle peut donc être intégré numériquement et il sera validé à partir de la solution analytique vue précédemment.

Attention

Il faut au préalable charger les fichiers complémentaires pour la partie 2 qui se trouvent sur Moodle, et mettre le fichier principal et le module de la nouvelle classe au bon endroit, en se basant sur ce qui a été fait pour la partie 1.

4.2 Création de la classe `Model_2`

On crée une nouvelle classe, `Model_2`.

On a besoin des trois paramètres d'entrée `v_0, h et alpha` mais aussi de paramètres numériques.

On va donc entrer les paramètres par un dictionnaire qui s'appelle `params`

On peut intégrer le problème numériquement avec différentes méthodes qui sont vues dans le cours de calcul scientifique. Ici on ne donnera pas les détails mais juste la méthode permettant de le faire. Principalement on peut intégrer avec un schéma d'Euler à l'ordre 1 ou un schéma de Runge-Kutta à l'ordre 4. On va utiliser une méthode Python `odeint` qui utilise le schéma de Runge-Kutta en une ligne de code.

Pour gagner du temps, le code est partiellement fourni.

La classe est alors utilisée dans un nouveau script principal `main_model_2.py`.

Il faut ouvrir les deux fichiers correspondant, les lire, les comprendre et les modifier pour que tout soit fonctionnel.

Quelques remarques s'imposent :

1. Le constructeur `__init__` prend en compte le dictionnaire des paramètres en paramètre de la méthode.
2. La méthode `solve_trajectory` intègre le système différentiel, avec les bonnes conditions initiales. Bien observer les sorties qui sont de nouveaux attributs de la classe, donc facilement accessibles.

   En entrée on a laissé l'angle α et on ajouté le temps de la simulation t_{end}.

   Le nom de cette méthode commence par `solve` pour bien indiquer qu'on résoud quelque chose.
3. On conserve la méthode `plot_trajectory`. Elle doit être recopiée de la partie 1.
4. Une méthode `validation` a été rajoutée afin de vérifier si la solution numérique est proche de la solution analytique. Cette méthode est pour le moment incomplète.
5. La simplicité du script principal est à relever.

4.3 Validation

En l'état la validation n'est pas effectuée. Il faut en fonction des paramètres d'entrée, calculer et afficher pour la durée finale de la trajectoire `t_end` la position de l'objet obtenue par la solution numérique et celle analytique, et donner un erreur sur les deux coordonnées.

Pour cela il faut modifier la méthode `validation` implémentée dans la classe et tester dans le script principal.

L'erreur **absolue** doit être inférieure à 10^{-7} pour la validation.

4.4 Amélioration de la portée, calcul précis à l'impact

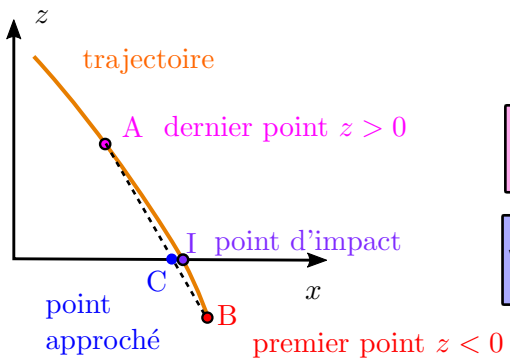
Pour déterminer le point d'impact, il faut d'abord déterminer le temps pour l'impact, t_i qui vérifie $z(t_i) = 0$. On procède par interpolation linéaire à partir de la solution numérique, au voisinage de $z = 0$. On peut se référer à la figure 2 pour la démarche et les relations utiles.

Travail à mener par les étudiants

- Dans la classe actuelle on ne calcule pas encore les valeurs à l'impact.
- La méthode est pourtant présente dans la classe : il faut donc la modifier afin de trouver les valeurs à l'impact puis tester ci-dessous.
- La validation se fera à partir de la solution analytique.
- Il faut vérifier que l'utilisateur demande un temps de simulation `t_end` tel que l'altitude à l'impact soit négative, sinon il n'y a pas d'impact.

  Dans ce cas sortir un message d'erreur avec `raise ValueError()`, ainsi on stoppe le script.

  Si `t_end` est trop grand il faut arrêter le dessin de la trajectoire dès que $z < 0$.
- On peut procéder par interpolation linéaire, entre le dernier point vérifiant $z > 0$ et le premier point avec $z < 0$.
- On pourrait utiliser une méthode d'interpolation `interp1d` présente dans le module `scipy.interpolate`, mais finalement c'est plus simple ici d'utiliser la méthode décrite dans la figure 2. Pour trouver les valeurs à l'impact il suffit alors de remplacer u par θ, v_x, v_z.
- Effectuer les implémentations nécessaires et ajouter les lignes de codes dans le script principal.



$$\frac{x_I - x_A}{x_B - x_A} = \frac{z_I - z_A}{z_B - z_A}$$

$$x_I - x_A = -z_A \frac{x_B - x_A}{z_B - z_A}$$

$$\forall u : \quad \frac{x_I - x_A}{x_B - x_A} = \frac{u_I - u_A}{u_B - u_A}$$

FIGURE 2 – Interpolation linéaire au voisinage de $z = 0$.

5 Phase 5 : modèle complet et classe `Model_3`

On étend le modèle précédent en supposant que l'objet est soumis à la force aérodynamique (portance et traînée) et à une force de propulsion T_0 constante qui est suivant la direction du vecteur vitesse $\boldsymbol{v}$.

5.1 Modèle vectoriel

- La force de portance (*lift*) et de traînée (*drag*) sont définies respectivement en fonction de la pression dynamique q par :

$$L = q\ S\ c_\ell, \qquad D = -q\ S\ c_d$$

  S est une section droite de l'objet et respectivement, c_ℓ et c_d sont les coefficients de portance et de traînée.
- La **pression dynamique** est fonction de la vitesse relative et de la masse volumique du gaz ρ (ici l'air) :

$$q = \frac{\rho}{2} v^2, \qquad v^2 = v_x^2 + v_z^2$$

- Comme les composantes de la force aérodynamique L et D sont définies dans le repère du vecteur vitesse, il faut les projeter dans le repère cartésien et on a :

$$R_z = q\ S\ c_z, \qquad R_x = q\ S\ c_x, \qquad c_z = c_\ell \cos\theta - c_d \sin\theta, \qquad c_x = -c_d \cos\theta - c_\ell \sin\theta$$

  avec

$$\theta = \arctan \frac{v_z}{v_x}$$

  Dans ces relations les coefficients c_ℓ et c_d sont comptés positifs et l'angle θ est négatif dès le franchissement du point le plus haut de la trajectoire.
- Dans la réalité, les coefficients c_ℓ et c_d sont compliqués à calculer, bien qu'il existe quelques corrélations. Pour ce modèle on les considérera comme des **paramètres d'entrée constants**.
- Si l'objet est soumis à une force de propulsion T_0 constante qui est suivant la direction du vecteur vitesse $\boldsymbol{v}$, il faut projeter dans le repère cartésien cette force.
- L'équation différentielle à résoudre est la même mais le second membre a changé :

Equations

$$\frac{d\boldsymbol{y}}{dt} = \boldsymbol{f}(t, \boldsymbol{y}) \qquad \Longrightarrow \qquad \frac{d}{dt}\begin{bmatrix} x \\ z \\ v_x \\ v_z \end{bmatrix} = \begin{bmatrix} v_x \\ v_z \\ \beta\ v^2\ c_x + c_t\ \cos\theta \\ -g + \beta\ v^2\ c_z + c_t\ \sin\theta \end{bmatrix} \qquad c_t = \frac{T_0}{m}, \qquad \beta = \frac{\rho\ S}{2\ m}$$

- On remarque qu'il faut maintenant tenir compte de la masse de l'objet m.
- On a introduit deux nouveaux coefficients β et c_t qui jouent un rôle important dans la trajectoire.
- Les conditions initiales restent inchangées.

5.2 Création de la classe `Model_3`

Le travail est le suivant :

1. Il faut d'abord recopier la classe `Model_2` dans le fichier `model_3.py` et renommée la classe.
2. dans le constructeur de la nouvelle classe, il faut :
   (a) Modifier le dictionnaire des paramètres d'entrée et ajouter les attributs nécessaires : `mass, rho, area, cd, cl` qui représentent les paramètres m, ρ, S, c_d, c_ℓ du problème.
   (b) Ajouter et calculer les attributs `beta` et `c_t`
3. Modifier le premier message
4. Rajouter les termes nécessaires dans la méthode `ode`
5. Modifier la méthode `get_parameters` car il y plus de paramètres d'entrée
6. Apporter toutes les autres petites modifications, notamment les titres ou légendes de figures qui indiquent éventuellement les valeurs des paramètres principaux `beta, c_z, c_x, c_t`.

Il faut faire les modifications progressivement et tester leur implémentation dans le script principal `main_model_3.py`
On prendra les valeurs numériques tests :
$S = 0.01$ m^2, $\rho = 1.3$ kg/m^3, $m = 100$ g, $c_\ell = 0.18$, $c_d = 0.01$
Si besoin, on prendra une force de propulsion de la forme :

$$T_0 = a \times mg, \qquad a \approx 0.3$$

En fixant $c_d = c_\ell = T_0 = 0$, on doit retrouver la valeur de la portée classique, en conservant le même jeu de paramètres h, v_0, α.

Maintenant on va créer une étude paramétrique dans le script principal qui contiendra alors 3 tâches :

- `task == 0` : tests d'implémentation de la classe
- `task == 1` : tracé de 4 trajectoires défnies par 3 cas :
  1. $c_d = c_\ell = 0$, cas classique sans friction
  2. $c_\ell = 0$, cas avec modèle de traînée uniquement
  3. $c_d \neq 0, c_\ell \neq 0$, cas avec portance et traînée
  4. $c_d \neq 0, c_\ell \neq 0$ et $c_t = 0.3g$.

On affichera aussi les valeurs des portées à chaque fois.

On prendra $\alpha = 36°$ pour tous les cas.

- `task == 2`, on trace les isocontours de la distance d'impact dans le plan $(\alpha - c_d)$, pour :

$$20 \leqslant \alpha \leqslant 60, \qquad 0 \leqslant c_d \leqslant 0.1$$

en prenant au moins 11 points sur chaque axe (sois 121 calculs).

Il faut utiler la méthode `contourf` avec la `colormap jet`.

Pour cette tâche on pourra modifier aussi le contenu de la classe.

Dans le script principal on doit introduire des fonctions, comme par exemple les fonctions `plot_trajectories` et `plot_contours`, des listes ou tuples, pour conserver les sorties des instances de classe pour chaque cas.

Au final on doit trouver des figures qui ressemblent aux figures 3 et 4.

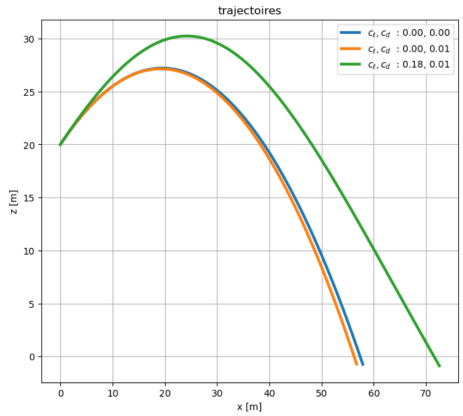


FIGURE 3 – Trajectoire avec le modèle 3 pour différents cas.

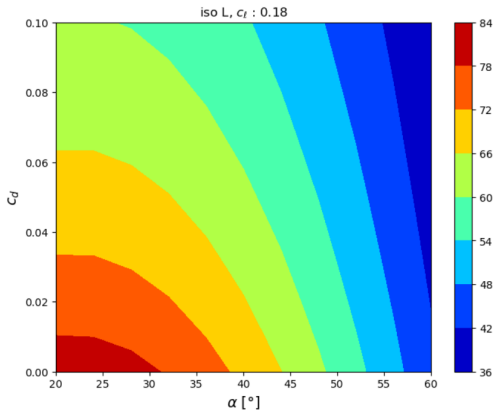


FIGURE 4 – Contours de la portée dans le plan $(\alpha - c_d)$.

6 Rendu du projet

6.1 Le rapport

Il faut penser à rédiger un rapport sur ce projet, contenant au moins un page de titre, une introduction, une conclusion, une bibliographie, et des sections relatives aux travails, et aux analyses et résultats obtenus.

Des petits morceaux de codes, de quelques lignes peuvent être mis en illustration dans un annexe.

Le rapport doit comporter de 8 à 10 pages, tout compris.

Latex est à privilégier mais un document word ou openoffice est acceptable.

Il n'est pas utile de recopier systématiquement les relations données dans le sujet.

La notation portera sur votre apport original dans le texte et sur la forme du rapport.

Il faudra le rendre au format pdf.

6.2 Le module `balistique`

Le projet reste largement améliorable, normalement il faudrait :

- ne conserver que deux classes, celle qui donne la solution analytique et celle qui donne la solution numérique
- Il faudrait rajouter deux classes pour les études paramètriques et réduire le nombre de script principaux à deux.
- Il faudrait sauvegarder les courbes dans un dossier "Outputs"
- Il faudrait améliorer les messages ou informations en ajoutant de la couleur dans le terminal.
- Il faudrait mettre les paramètres dans un fichier texte au format TOML.

Mais il faudrait plus de temps pour faire tout cela. Toute personne qui prendra du temps pour effectuer une ou la totalité des améliorations possibles obtiendra des points supplémentaires et l'estime des professeurs.

Il faut vérifier que si l'enseignant lance les scripts principaux, les codes fonctionnent sans erreurs.

Il faut aussi relire ses codes pour obtenir un rendu qui correspond à la PEP 8.

L'ensemble du dossier contenant les codes doit être nettoyé, en enlevant les dossiers du type `__pycache__`. Il ne doit rester que des fichiers *.py.

Il faut positioner le rapport au format pdf dans un dossier du module `balistique` qui s'appelle **Rapport**.

Ensuite il faut archiver le dossier "balistique" au format zip avec le nom des étudiants du binôme (par exemple : Dupond_Durand.zip).

Enfin il faut le déposer sur Moodle, dans la section Dépôt de projet, zone mécanique-énergétique, sur l'un des membres du binômes.

Il faut rappeler que le copiage de codes non compris sera particulièrement scruté.

La date limite sera indiqué sur Moodle le moment venu. Il n'y aura pas de temps supplémentaire accordé.