Balistique

T. Camus, J. Dopeux(chasseur.fou@wanadoo.fr), L. Jardoux

Résumé: on étudie la trajectoire de mobiles en chute libre ou de projectiles du point de vue du calcul formel. On propose une modélisation en *Mathematica*, accompagnée de quelques applications à titre d'illustrations et de tests.
Mots-clés: balistique, projectile, trajectoire, calcul formel.

Abstract: the trajectory of mobiles is investigated from the computer algebra. A *Mathematica* implementation is put forward, with a few applications as illustration and test. **Keywords:** ballistics, projectile, trajectory, computer algebra.

Introduction

Nous nous proposons de mener l'étude de la trajectoire d'un projectile. On établira pour cela l'équation du mouvement que l'on résoudra de façon exacte ou par une méthode d'approximation. Le programme prend en compte les conditions initiales (vitesse, position) et la modification possible des différents paramètres entrant en jeu dans l'équation de mouvement (masse du projectile, résistance à l'air). On créera des fonctions annexes pour le calcul du point d'impact, la durée de la trajectoire, le temps mis par le projectile pour atteindre sa hauteur maximale, la hauteur maximale atteinte par le projectile...

Etude préliminaire

■ Hypothèses

■ Influence de l'attraction terrestre

Si aucune force d'attraction n'était exercée sur le projectile celui-ci continuerait sa course dans l'axe du tir et serait donc rectiligne. Mais l'attraction terrestre (la pesanteur) exerce son action, et tend à ce que le projectile chute. A partir de la vitesse initiale, de la position initiale du projectile et de l'accélération de la pesanteur nous pouvons savoir la trajectoire du projectile dans le vide, ce dernier subissant l'attraction de pesanteur dès la sortie du "canon". Par la suite on désignera par la lettre g l'accélération gravitationnelle ou due à la gravitation. La valeur approchée de g est 9,80 m/s^2, Cette valeur subit des variations à la surface de la Terre, elle augmente légèrement quand on va de l'équateur aux pôles [1, 6, 7].

■ Influence de la résistance de l'air

Dans la pratique un objet n'est pas seulement soumis à son poids mais aussi à d'autres forces. Une catégorie importante est celle des forces qui tentent à s'opposer au mouvement. On appelle ces forces, qui existent généralement lors du mouvement dans un milieu tel que l'air ou l'eau, les forces résistantes ou d'amortissement ou dissipatives.

L'air est un fluide, il oppose une certaine résistance à un objet s'y déplaçant qui dépend de la section perpendiculaire par rapport au déplacement, de la forme et particulièrement de la vitesse instantanée. La résistance d'un fluide est opposée et proportionnelle à la vitesse dans le cas de faible vitesse (on utilisera cette règle lors de l'étude préliminaire) ou au carré de la vitesse (on distinguera cette règle de la précédente lors de la programmation de la trajectoire). Donc, plus notre projectile va vite plus l'air lui oppose de résistance [1, 7].

■ Planéité de la Terre

Dans la pratique, l'hypothèse de la Terre plate est très adéquate pour décrire le mouvement local d'objets près de la surface de la Terre et nous l'utiliserons dans cette étude. Toutefois, pour décrire le mouvement d'objets éloignés de la surface terrestre il convient de se ramener à l'utilisation des forces centrales [1, 7].

■ Analyse du problème

On se place dans un repère $\left(O, \vec{i}, \vec{j}, \vec{k}\right)$ supposé être galiléen, $\vec{k}$ étant vertical ascendant et $(O, \vec{i}, \vec{j})$ formant le plan de la Terre. Soit $\vec{r}$ le vecteur position du projectile, de masse m, à un instant quelconque. Le projectile est soumis à une force de résistance de l'air égale à $-\beta \times \vec{v}$ (dans le cas d'une étude simplifiée) où β désigne une constante positive. Alors d'après la loi de Newton, la force de Coriolis étant négligée, on a :

$$m \times \frac{d^2 \vec{r}}{dt^2} = -m \times \vec{g} - \beta \times \vec{v}$$

ce qui équivaut à:

$$m \times \frac{d \vec{v}}{dt} + \beta \times \vec{v} = -m \times g \times \vec{k}$$

On peut résoudre cette équation différentielle en projetant le vecteur vitesse sur les axes du repère. On obtient ainsi un système de trois équations différentielles.

- Résolution suivant la composante $\vec{i}$

$$m \times \frac{d v_x}{dt} + \beta \times v_x = 0$$

on obtient donc :

$$v_x = A \times e^{\frac{-\beta t}{m}}$$

à t=0, $\vec{v_0} = v_0 \times \cos(\alpha) \times \vec{j} + v_0 \times \sin(\alpha) \times \vec{k}$, ce qui donne $A = 0$ et donc $v_x = 0$.

- Résolution suivant la composante $\vec{j}$

$$m \times \frac{d v_y}{dt} + \beta \times v_y = 0$$

on obtient donc :

$$v_y = B \times e^{\frac{-\beta t}{m}}$$

à t=0, $v_y = v_0 \times \cos(\alpha)$, ce qui donne $B = v_0 \times \cos(\alpha)$ et donc $v_y = v_0 \times \cos(\alpha) \times e^{\frac{-\beta t}{m}}$

- Résolution suivant la composante $\vec{k}$

$$m \times \frac{d v_z}{dt} + \beta \times v_z = -m \times g$$

on obtient donc :

$$v_z = C \times e^{\frac{-\beta \times t}{m}} - \frac{m \times g}{\beta}$$

à t=0, $v_z = v_0 \times \sin(\alpha) = C - \frac{m \times g}{\beta}$; d'où $C = v_0 \times \sin(\alpha) + \frac{m \times g}{\beta}$ et $v_z = (v_0 \times \sin(\alpha) + \frac{m \times g}{\beta}) \times e^{\frac{-\beta \times t}{m}} - \frac{m \times g}{\beta}$

- Finalement

$$\vec{v} = \left(v_0 \times \cos(\alpha) \times \vec{j} + v_0 \times \sin(\alpha) \times \vec{k}\right) \times e^{\frac{-\beta \times t}{m}} - \frac{m \times g}{\beta} \times \left(1 - e^{\frac{-\beta \times t}{m}}\right) \times \vec{k}$$

on obtient $\vec{r}$ par intégration :

$$\vec{r} = \frac{\left(1 - e^{-\frac{t\,\beta}{m}}\right) m\, v0\, \text{Cos}[\alpha]}{\beta} \times \vec{j} + \frac{g\,m^2 - g\,m\,t\,\beta + h\,\beta^2 + m\,v0\,\beta\,\text{Sin}[\alpha] - e^{-\frac{t\,\beta}{m}}\, m\,(g\,m + v0\,\beta\,\text{Sin}[\alpha])}{\beta^2} \times \vec{k}$$

Conception d'un programme

■ Calcul pas à pas de la trajectoire du projectile

Nous allons développer dans une première partie un calcul pas-à-pas de la trajectoire d'un projectile, puis nous regrouperons les différentes fonctions pour former un programme permettant de traiter aussi bien les conditions initiales que les différents paramètres entrant en jeu dans l'équation de la trajectoire [3, 5]. Nous pourrons vérifier la compatibilité des différentes étapes de programmation et la cohérence des résultats en traçant les courbes de trajectoire et en comparant les résultats du programme aux calculs théoriques réalisés précédemment.

On enregistre l'équation différentielle déterminant la trajectoire du projectile, puis on la projette sur les axes du repère. On aboutit ainsi à un système de trois équations différentielles linéaires à coefficients constants.

```
theEquation := m * D[X[t], {t, 2}] + β * (D[X[t], t]) == m * G
X[t_] = Through[{x, y, z}[t]]; G = {0, 0, -g};
theSystem = Thread[theEquation]
```

```
{β x'[t] + m x''[t] == 0, β y'[t] + m y''[t] == 0, β z'[t] + m z''[t] == -g m}
```

On fixe les conditions initiales du système.

```
initialPosition = Thread[X[0] == {0, 0, h}];
initialVelocity = Thread[X'[0] == {0, v0 * Cos[α], v0 * Sin[α]}];
```

On résout le systéme d'équations suivant les axes du repère en prenant en compte les conditions initiales du système.

```
theSolution = FullSimplify[
  DSolve[Flatten[{theSystem, initialPosition, initialVelocity}], X[t], t]]
```

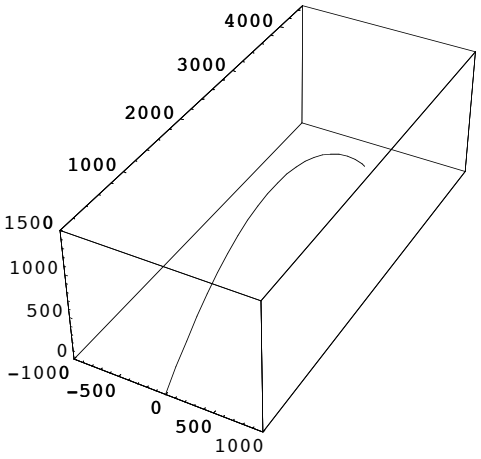
$$\left\{\left\{x[t] \to 0,\ y[t] \to \frac{\left(1 - e^{-\frac{t\,\beta}{m}}\right) m\, v0\, \text{Cos}[\alpha]}{\beta},\right.\right.$$
$$\left.\left. z[t] \to \frac{g\,m^2 - g\,m\,t\,\beta + h\,\beta^2 + m\,v0\,\beta\,\text{Sin}[\alpha] - e^{-\frac{t\,\beta}{m}}\, m\,(g\,m + v0\,\beta\,\text{Sin}[\alpha])}{\beta^2}\right\}\right\}$$

Le résultat obtenu correspond effectivement à celui mené dans l'étude préliminaire, la force de Coriolis étant négligée.

La fonction suivante permet de donner et/ou de modifier les valeurs des différents paramètres et des conditions initiales du système pour avoir ainsi une programmation plus ergonomique. On fixera par la suite du développement du programme les mêmes valeurs numériques puis nous tracerons la courbe représentative de la trajectoire du projectile à titre d'illustrations et de tests de vérification de l'étendue de la programmation.

```
numericalValues =
  {h → 1., v0 → 200., α → π/4, β → 10^-3, g → 9.80, m → 10.};

ParametricPlot3D[
  Evaluate[X[t] /. Flatten[theSolution] /. numericalValues],
  {t, 0, 100}, PlotRange → {{-1000, 1000}, {0, 4500}, {0, 1500}}];
```



Nous pouvons ne pas nous satisfaire de cette résolution particulière. En effet, il existe de nombreux problèmes de balistique, que ce soit en chute libre ou en trajectoire parabolique (arc de parabole ou une droite que l'on peut considérer comme une parabole dégénérée) qui font appel à des calculs du même type. Il pourrait être intéressant de généraliser ce programme pour qu'il s'applique à toutes les configurations possibles. Le programme général consistera à rassembler en une fonction unique toutes les étapes successives du calcul, admettant pour variables les différents paramètres du système.

■ Programme général

On exécute un `ClearAll["Global`*"]` pour effacer les définitions utilisées précédemment. On regroupe les différentes fonctions utilisées dans le calcul pas à pas de l'étude de la trajectoire en une fonction unique en vue de simplifier l'utilisation du programme [3, 5].

```
ProjectileTrajectory[X_, t_, X0_, v0_, α_, β_, G_, m_] :=
 DSolve[Flatten[{
    Thread[m * D[X, {t, 2}] + β * D[X, t] == m * G],
    Thread[(X /. t → 0) == X0],
    Thread[(D[X, t] /. t → 0) == {0, v0 * Cos[α], v0 * Sin[α]}]}], X, t]
X[t_] = Through[{x, y, z}[t]];
theSolution = FullSimplify[
  ProjectileTrajectory[X[t], t, {0, 0, h}, v0, α, β, {0, 0, -g}, m]]
```

$$\left\{\left\{x[t] \to 0,\ y[t] \to \frac{\left(1 - e^{-\frac{t\,\beta}{m}}\right)\, m\, v0\, \text{Cos}[\alpha]}{\beta},\right.\right.$$

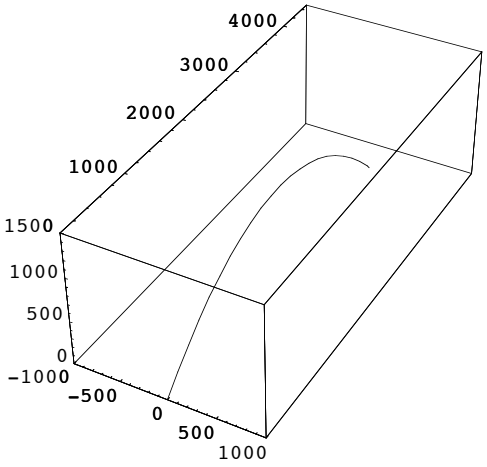
$$\left.\left. z[t] \to \frac{g\,m^2 - g\,m\,t\,\beta + h\,\beta^2 + m\,v0\,\beta\,\text{Sin}[\alpha] - e^{-\frac{t\,\beta}{m}}\, m\,(g\,m + v0\,\beta\,\text{Sin}[\alpha])}{\beta^2}\right\}\right\}$$

On fixe les valeurs des différents paramètres et des conditions initiales du système.

```
numericalValues =
  {h → 1., v0 → 200., α → π/4, β → 10^-3, g → 9.80, m → 10.};
```

On vérifie la conformité de la programmation en traçant la trajectoire du projectile avec les mêmes valeurs numériques des différents paramètres et des conditions initiales que dans la première partie du développement.

```
ParametricPlot3D[
  Evaluate[X[t] /. Flatten[theSolution] /. numericalValues],
  {t, 0, 100}, PlotRange → {{-1000, 1000}, {0, 4500}, {0, 1500}}];
```



Cependant la multitude des paramètres rend ce programme peu confortable. Pour remédier à ce problème nous pouvons introduire une fonction avec des options munies de valeurs par défaut. Nous développerons cette programmation dans une annexe (annexe 1) pour une question de fluidité du rapport.

■ Résolution par une méthode d'approximation

Nous avons vu dans l'étude préliminaire concernant l'influence de la résistance de l'air que celle-çi est proportionnelle à la vitesse dans le cas de faible vitesse ou à une puissance de la vitesse — généralement au carré de la vitesse — en cas de chute libre ou dans le cas où la vitesse est importante. Cependant la puissance de la vitesse peut être quelquefois supérieure à deux, comme le montrent des études plus complexes et approfondies [1]. Nous nous limiterons dans cette étude à la résolution de l'équation de la trajectoire d'un projectile pour lequel l'influence de la résistance de l'air est proportionnelle au carré de la vitesse instantanée.

Il existe sur Mathematica une méthode de résolution par approximation qui calcule une solution numérique d'équation différentielle, qu'elle soit linéaire ou non (NDSolve). Cette fonction donne des résultats en tant que fonction interpolée (InterpolatingFunction). Nous verrons l'utilisation de cette fonction en annexe (annexe 2).

Une autre méthode d'approximation importante pour résoudre des équations différentielles non-linéaires est la méthode de perturbation. Cette méthode est utilisée s'il existe un paramètre petit et si l'équation différentielle peut être résolue dans le cas où ce paramètre est nul ; il est alors possible de faire un développement — dit de perturbation — de la solution en les puissances de ce petit paramètre [4].

■ Méthode de perturbation

Nous développerons la méthode de perturbation par rapport au paramètre β car celui-ci est très petit devant 1. On enregistre l'équation différentielle déterminant la trajectoire du projectile, puis on la projette sur les axes du repère. On aboutit ainsi à un système de trois équations différentielles.

```
theEquation := m * D[XApproximate[t], {t, 2}] + β * (D[XApproximate[t], t])^2 == m * G
XApproximate[t_] = Through[{xAp, yAp, zAp}[t]]; G = {0, 0, -g};
theSystem = Thread[theEquation];
```

On définit x[t], y[t] et z[t] comme des polynômes en β d'ordre 4 dont les coefficients sont des fonctions de t.

```
xAp[t_] = Sum[β^i * a[i][t], {i, 0, 4}]; yAp[t_] = Sum[β^i * b[i][t], {i, 0, 4}]; zAp[t_] = Sum[β^i * c[i][t], {i, 0, 4}];
```

Les fonctions $a_0(t)$, $a_1(t)$,…$b_0(t)$, $b_1(t)$…$c_0(t)$, $c_1(t)$… sont à déterminer pour que la solution approchée satisfasse l'équation différentielle. La solution approchée doit également satisfaire les conditions initiales : X(0)={0, 0, h} et X'(0)={0, $v_0 \times \cos(\alpha)$, $v_0 \times \sin(\alpha)$}. Nous devons choisir les conditions initiales pour chaque $a_n(t)$, $b_n(t)$ et $c_n(t)$ et leurs possibles dérivées [5]. Nous prenons $a_0(0) = h$, $a_n(0) = 0$ pour n>0, $b_n(0) = 0$ pour n≥0, $c_n(0) = 0$ pour n≥0, $a_n'(0)=0$ pour n≥0, $b_0'(0)=v_0 \times \cos(\alpha)$, $b_n'(0)=0$ pour n>0, $c_0'(0)=v_0 \times \sin(\alpha)$ et $c_n'(0)=0$ pour n>0.

Après avoir substitué la solution approchée dans l'équation de la trajectoire nous obtenons un système de trois équations polynomiales en β dont les coefficients sont à déterminer. Pour cela on sélectionne les coefficient de même puissance en β, qui forment une séquence d'équations différentielles que nous pouvons résoudre successivement. Chaque coefficient de même puissance de β obtenu est nul. Nous parvenons ainsi à plusieurs conditions sur les coefficients de la solution approchée.

- Résolution de x(t)

La fonction suivante permet de résoudre les différentes équations différentielles formant les coefficients du polynôme en β ; elle prend en compte les conditions initiales sur les $a_n(t)$.

```
SolOnX[i_Integer] :=
 DSolve[{CoefficientList[Collect[Subtract @@ theSystem[[1]], β], β][[i + 1]] == 0,
     {a[i][0] == 0, (D[a[i][t], t] /. t → 0) == 0}} // Flatten, a[i][t], t][[1, 1]]
```

Les deux fonctions suivantes permettront plus tard de rendre la programmation moins "lourde". Elles servent à ce que les $a_n(t)$ soient correctement substitués et dérivés.

```
SubstituteX[i_Integer] := a[i][t] = a[i][t] /. SolOnX[i]
DerivativeX[i_Integer] := a[i]'[t] = D[a[i][t], t]
```

On peut désormais calculer les différents coefficients de x(t).

```
For[i = 0, i < 5, SolOnX[i]; SubstituteX[i]; DerivativeX[i]; i++]
```

On affiche maintenant la solution approchée de x(t).

```
xAp[t]
```

```
0
```

- Résolution de y(t)

On effectue le même raisonnement sur y(t).

```
SolOnY[i_Integer] :=
 DSolve[{CoefficientList[Collect[Subtract @@ theSystem[[2]], β], β][[i + 1]] == 0,
     {b[i][0] == 0, (D[b[i][t], t] /. t → 0) == If[i == 0, v0 * Cos[α], 0]}} //
    Flatten, b[i][t], t][[1, 1]]
SubstituteY[i_Integer] := b[i][t] = b[i][t] /. SolOnY[i]
DerivativeY[i_Integer] := b[i]'[t] = D[b[i][t], t]

For[i = 0, i < 5, SolOnY[i]; SubstituteY[i]; DerivativeY[i]; i++]
```

On affiche maintenant la solution approchée de y(t).

```
yAp[t]
```

$$t\, v0\, \text{Cos}[\alpha] + \frac{t^3\, v0^3\, \beta^2\, \text{Cos}[\alpha]^3}{3\, m^2} - \frac{t^4\, v0^4\, \beta^3\, \text{Cos}[\alpha]^4}{4\, m^3} + \frac{t^5\, v0^5\, \beta^4\, \text{Cos}[\alpha]^5}{5\, m^4} - \frac{t^2\, \beta\, (v0^2 + v0^2\, \text{Cos}[2\, \alpha])}{4\, m}$$

▪ Résolution de z(t)

On effectue le même raisonnement sur z(t).

```
SolOnZ[i_Integer] :=
 DSolve[{CoefficientList[Collect[Subtract @@ theSystem[[3]], β], β][[i + 1]] == 0,
     {c[i][0] == If[i == 0, h, 0], (D[c[i][t], t] /. t → 0) == If[i == 0, v0 * Sin[α], 0]}} //
    Flatten, c[i][t], t][[1, 1]]
SubstituteZ[i_Integer] := c[i][t] = c[i][t] /. SolOnZ[i]
DerivativeZ[i_Integer] := c[i]'[t] = D[c[i][t], t]

For[i = 0, i < 5, SolOnZ[i]; SubstituteZ[i]; DerivativeZ[i]; i++]
```

On affiche maintenant la solution approchée de z(t).

```
zAp[t]
```

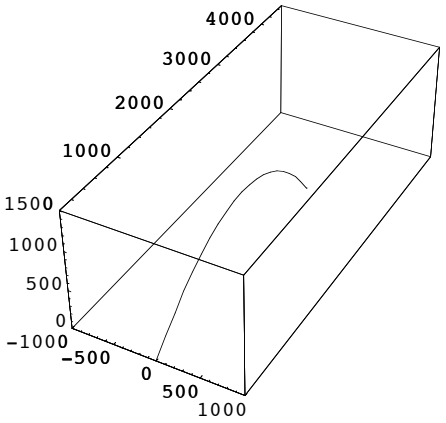
$$\frac{1}{2}\, (2\, h - g\, t^2 + 2\, t\, v0\, \text{Sin}[\alpha]) + \frac{\beta\, (-g^2\, t^4 + 4\, g\, t^3\, v0\, \text{Sin}[\alpha] - 6\, t^2\, v0^2\, \text{Sin}[\alpha]^2)}{12\, m} + \frac{\beta^2\, (-g^3\, t^6 + 6\, g^2\, t^5\, v0\, \text{Sin}[\alpha] - 15\, g\, t^4\, v0^2\, \text{Sin}[\alpha]^2 + 15\, t^3\, v0^3\, \text{Sin}[\alpha]^3)}{45\, m^2} + \frac{1}{2520\, m^3} (\beta^3\, (-17\, g^4\, t^8 + 136\, g^3\, t^7\, v0\, \text{Sin}[\alpha] - 476\, g^2\, t^6\, v0^2\, \text{Sin}[\alpha]^2 + 840\, g\, t^5\, v0^3\, \text{Sin}[\alpha]^3 - 630\, t^4\, v0^4\, \text{Sin}[\alpha]^4)) + \frac{1}{14175\, m^4} (\beta^4\, (-31\, g^5\, t^{10} + 310\, g^4\, t^9\, v0\, \text{Sin}[\alpha] - 1395\, g^3\, t^8\, v0^2\, \text{Sin}[\alpha]^2 + 3465\, g^2\, t^7\, v0^3\, \text{Sin}[\alpha]^3 - 4725\, g\, t^6\, v0^4\, \text{Sin}[\alpha]^4 + 2835\, t^5\, v0^5\, \text{Sin}[\alpha]^5))$$

On fixe les valeurs des différents paramètres et des conditions initiales du système.

```
numericalValues =
  {h → 1., v0 → 200., α → π/4, β → 10^-3, g → 9.80, m → 10.};
```

On peut maintenant exécuter le tracé de la trajectoire du projectile.

```
ParametricPlot3D[
  Evaluate[XApproximate[t] /. numericalValues],
  {t, 0, 100}, PlotRange → {{-1000, 1000}, {0, 4500}, {0, 1500}}];
```



Discussion et perspectives

■ Discussion

Nous avons pu développer au cours de cette étude la programmation de méthodes de calcul de la trajectoire d'un projectile. La dissociation des cas où la résistance de l'air dépend de la vitesse ou du carré de la vitesse a nécessité le développement d'un programme de résolution. En effet, il n'existe pas d'algorithme capable de résoudre de façon exacte toutes les équations différentielles non-linéaires. Il existe cependant sur Mathematica une méthode d'approximation numérique (NDsolve) qui semble répondre aux attentes de l'utilisateur. Une résolution par approximation avec la méthode de perturbation sur un petit paramètre a été développée au cours de l'étude. Par comparaison des deux tracés nous pouvons valider les résultats des deux méthodes d'approximation. Au cours de nos recherches, nous nous sommes rendus compte que certaines caractéristiques de la trajectoire du projectile avaient de l'importance, telles que la hauteur maximale atteinte par le projectile ou bien encore la portée de tir. Il a été ainsi développé plusieurs sous programmes en annexe (annexe 2) répondant a ces attentes. Nous avons également développé en annexe (annexe 1) un programme simplifié muni de valeurs par défaut dans le cas où la résistance de l'air dépend de la vitesse. Il s'agissait ici de faire un programme dans le cas d'une étude simple plus ergonomique et plus pratique d'utilisation.

■ Perspectives

La programmation a été développée en supposant que les forces de Coriolis agissant sur le projectile étaient négligeables. Nous avons tout de même essayé en annexe de résoudre l'équation différentielle en prenant en compte cette force. La complexité de l'équation implique une résolution par approximation, c'est pourquoi nous avons opté pour l'utilisation de la fonction NDsolve. Cependant la forme du système différentiel (trois équations différentielles couplées) implique une surabondance de calcul donc une utilisation très importante de la mémoire que Mathematica n'arrive pas à gérer. Nous avons tout de même tenu à mettre cette approche de programmation en annexe (annexe 4), mais il serait utile de développer une nouvelle méthode par approximation pour répondre à ce problème. Une fois ce programme réalisé il ne resterait plus qu'à mettre en place un programme d'approximation pouvant gérer à la fois la force de Coriolis et une résistance de l'air dépendant du carré de la vitesse. On obtiendrait ainsi un modèle aussi réaliste que possible, physiquement et mécaniquement parlant.

■ Conclusion

Le développement du programme général dans le cas où la résistance de l'air dépend de la vitesse est largement inspiré du programme publié dans le rapport sur la spectroscopie ionique. Il a fallu adapter ce programme à notre système mécanique et en comprendre le mécanisme d'exécution. Bien que le sujet sur la résolution d'équations différentielles non-linéaires par la méthode de perturbation ait déjà été développée, nous en avons proposé une autre mise en forme en résolvant les trois équations indépendemment. Mais nous aurions pu aller plus loin en réalisant une résolution vectorielle de l'équation, ce qui aurait rendu le programme plus ergonomique. Nous nous sommes rendus compte que malgré la puissance de résolution de Mathematica, la résolution de systèmes complexes implique parfois un problème de saturation de la mémoire.

Bibliographie

[1] M. Spiegel: Mécanique, Mc Graw-Hill
[2] V. Bourges: La physique avec Maple, Ellipses, 2000
[3] R. Barrère: Mathematica, Vuibert, 2002
[4] Mini-projet Mathematica: 047 Perturbations
[5] Mini-projet Mathematica: 075 Spectroscopie ionique
[6] http://www.royal-chasse.com
[7] http://www.physicclassroom.com

Annexes

■ Annexe 1 : ergonomie du programme

Nous avons vu que la multitude des paramètres rendait le programme peu confortable. C'est pourquoi nous avons décidé de le munir de valeurs par défaut au moyen des options. On exécute `ClearAll["Global`*"]` pour effacer les définitions utilisées précédemment. On fixe des valeurs par défauts aux différents paramètres et aux conditions initiales du système.

```
Options[ProjectileTrajectoryWithOptions] =
  {InitialPositionWithOptions → {0, 0, 1}, InitialVelocityWithOptions → 100,
   ShootingAnglWithOptions → π/4, AirRubbingWithOptions → 10^(-3),
   GravitationalForceWithOptions → 9.8, MassWithOptions → 10.};
```

La fonction suivante permet de mettre en forme l'équation de la trajectoire du projectile avec la possibilité de modifier ou de laisser inchanger les valeurs enregistrées par défauts dans `Options[ProjectileTrajectoryWithOptions]`.

```
ProjectileTrajectoryWithOptions[XOptions_, t_, theOptions___] :=
 ReleaseHold[Hold[DSolve[
      Flatten[{
        Thread[MassWithOptions × D[XOptions, {t, 2}] + AirRubbingWithOptions × D[XOptions
             t] == MassWithOptions × {0, 0, -GravitationalForceWithOptions}],
        Thread[(XOptions /. t → 0) == InitialPositionWithOptions],
        Thread[(D[XOptions, t] /. t → 0) ==
          {0, InitialVelocityWithOptions × Cos[ShootingAnglWithOptions],
           InitialVelocityWithOptions × Sin[ShootingAnglWithOptions]}]}], XOptions,
      t]] /. {theOptions} /. Options[ProjectileTrajectoryWithOptions]]
XOptions[t_] = Through[{xOp, yOp, zOp}[t]];
```

On peut maintenant résoudre l'équation de la trajectoire primo en gardant les valeurs munies par défaut et secondo en modifiant les valeurs (on prendra à titre d'illustration et de vérification les mêmes valeurs que dans la première partie du sujet).

```
theSolutionWithOptions = FullSimplify[
   ProjectileTrajectoryWithOptions[XOptions[t], t]];

theSolutionWithoutOptions = FullSimplify[
  ProjectileTrajectoryWithOptions[XOptions[t], t,
   InitialPositionWithOptions → {0, 0, hOp}, InitialVelocityWithOptions → v0Op,
   ShootingAnglWithOptions → αOp, AirRubbingWithOptions → βOp,
   GravitationalForceWithOptions → gOp, MassWithOptions → mOp]]
```

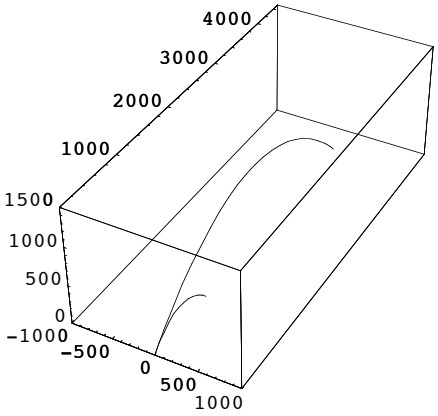
$$\left\{\left\{\text{xOp}[t] \to 0,\ \text{yOp}[t] \to \frac{\left(1 - e^{-\frac{t\,\beta Op}{mOp}}\right)\ mOp\ v0Op\ \text{Cos}[\alpha Op]}{\beta Op},\right.\right.$$
$$\text{zOp}[t] \to \frac{1}{\beta Op^2}\left(gOp\ mOp^2 - gOp\ mOp\ t\ \beta Op + hOp\ \beta Op^2 +\right.$$
$$\left.\left.\left.mOp\ v0Op\ \beta Op\ \text{Sin}[\alpha Op] - e^{-\frac{t\,\beta Op}{mOp}}\ mOp\ (gOp\ mOp + v0Op\ \beta Op\ \text{Sin}[\alpha Op])\right)\right\}\right\}$$

On fixe les valeurs des différents paramètres et des conditions initiales du système.

```
numericalValues =
  {hOp → 1., v0Op → 200., αOp → π/4, βOp → 10^-3, gOp → 9.80, mOp → 10.};
```

On superposera sur le même graphe la trajectoire du projectile avec les valeurs pré-enregistées dans le programme et le graphe avec les valeurs de numericalValues. Pour des raisons de règles de programmation en Mathematica il est nécessaire d'évaluer et d'enregistrer dans deux variables différentes les solutions obtenues.

```
ParamCurve1 = Evaluate[XOptions[t] /. Flatten[theSolutionWithOptions]];
ParamCurve2 =
  Evaluate[XOptions[t] /. Flatten[theSolutionWithoutOptions] /. numericalValues];
ParametricPlot3D[
  {{ParamCurve1[[1]], ParamCurve1[[2]], ParamCurve1[[3]]},
   {ParamCurve2[[1]], ParamCurve2[[2]], ParamCurve2[[3]]}},
  {t, 0, 100}, PlotRange → {{-1000, 1000}, {0, 4500}, {0, 1500}}];
```



■ Annexe 2 : utilisation de l'approximation numérique NDSolve de Mathematica

On exécute `ClearAll["Global`*"]` pour effacer les définitions utilisées précédemment puis on fixe les valeurs des différents paramètres et des conditions initiales du système.

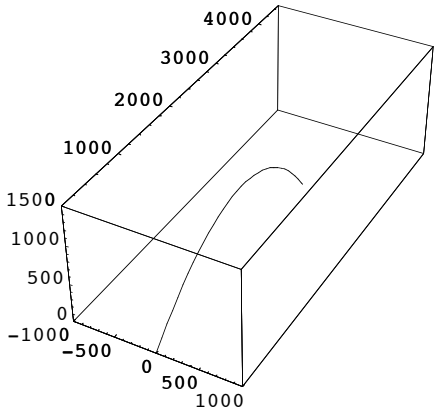
```
numericalValues =
  {h → 1., v0 → 200., α → π/4, β → 10^-3, g → 9.80, m → 10.};
```

On utilise le programme utilisé dans la sous section Programme général adaptée à l'utilisation de NDSolve, c'est à dire que les coefficients de l'équation différentielle et que les conditions initiales du système sont donnés sous forme numérique.

```
ProjectileTrajectory[Xsol_, t_, X0_, v0_, α_, β_, G_, m_] :=
 NDSolve[Flatten[{
     Thread[m * D[Xsol, {t, 2}] + β * (D[Xsol, t])^2 == m * G /. numericalValues],
     Thread[(Xsol /. t → 0) == X0 /. numericalValues],
     Thread[(D[Xsol, t] /. t → 0) == {0, v0 * Cos[α], v0 * Sin[α]}]} /. numericalValues],
  Xsol, {t, 0, 60}]
Xsol[t_] = Through[{x, y, z}[t]];
theSolution = FullSimplify[
   ProjectileTrajectory[Xsol[t], t, {0, 0, h}, v0, α, β, {0, 0, -g}, m]];
```

On peut maintenant exécuter le tracé de la trajectoire du projectile.

```
ParametricPlot3D[
  Evaluate[Xsol[t] /. Flatten[theSolution] /. numericalValues],
  {t, 0, 60}, PlotRange → {{-1000, 1000}, {0, 4500}, {0, 1500}}];
```

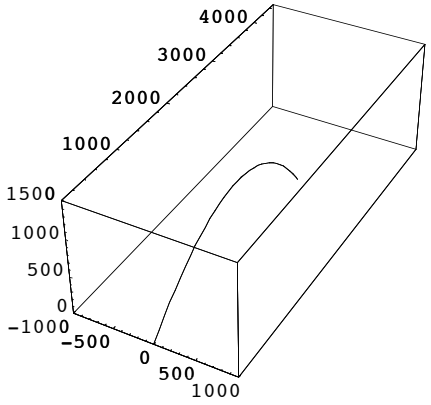


Avec des paramètres et des conditions initiales identiques dans le cas où la résistance de l'air est proportionnelle à la vitesse et dans le cas où la vitesse est proportionnelle au carré de la vitesse on peut déjà remarquer que le résultat de la courbe représentative de la trajectoire du projectile semble être cohérent avec l'étude préliminaire mené sur l'influence de la résistance de l'air. En effet la portée du tir et la hauteur maximale atteinte sont plus faibles dans le cas présent.

On superpose sur le même graphe la trajectoire du projectile avec la résolution du système différentielle par la fonction NDSolve de Mathematica et le graphe avec la résolution du système différentielle par la méthode de la perturbation d'un paramètre petit. Pour des raisons de règles de programmation sur Mathematica il est nécessaire d'évaluer et d'enregistrer dans deux variables différentes les solutions obtenues.

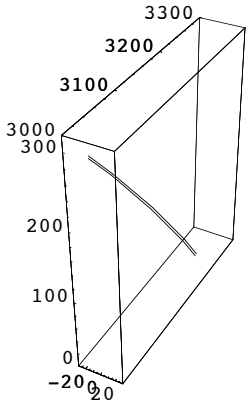
$$\text{XApproximate}[t] = \Big\{0,\ t\,v0\,\text{Cos}[\alpha] + \frac{t^3\,v0^3\,\beta^2\,\text{Cos}[\alpha]^3}{3\,m^2} - \frac{t^4\,v0^4\,\beta^3\,\text{Cos}[\alpha]^4}{4\,m^3} + \frac{t^5\,v0^5\,\beta^4\,\text{Cos}[\alpha]^5}{5\,m^4} - \frac{t^2\,\beta\,(v0^2 + v0^2\,\text{Cos}[2\,\alpha])}{4\,m},$$
$$\frac{1}{2}\,(2\,h - g\,t^2 + 2\,t\,v0\,\text{Sin}[\alpha]) + \frac{\beta\,(-g^2\,t^4 + 4\,g\,t^3\,v0\,\text{Sin}[\alpha] - 6\,t^2\,v0^2\,\text{Sin}[\alpha]^2)}{12\,m} +$$
$$\frac{1}{45\,m^2}\,(\beta^2\,(-g^3\,t^6 + 6\,g^2\,t^5\,v0\,\text{Sin}[\alpha] - 15\,g\,t^4\,v0^2\,\text{Sin}[\alpha]^2 + 15\,t^3\,v0^3\,\text{Sin}[\alpha]^3)) +$$
$$\frac{1}{2520\,m^3}\,(\beta^3\,(-17\,g^4\,t^8 + 136\,g^3\,t^7\,v0\,\text{Sin}[\alpha] - 476\,g^2\,t^6\,v0^2\,\text{Sin}[\alpha]^2 + 840\,g\,t^5\,v0^3\,\text{Sin}[\alpha]^3 - 630\,t^4\,v0^4\,\text{Sin}[\alpha]^4)) +$$
$$\frac{1}{14175\,m^4}\,(\beta^4\,(-31\,g^5\,t^{10} + 310\,g^4\,t^9\,v0\,\text{Sin}[\alpha] - 1395\,g^3\,t^8\,v0^2\,\text{Sin}[\alpha]^2 + 3465\,g^2\,t^7\,v0^3\,\text{Sin}[\alpha]^3 - 4725\,g\,t^6\,v0^4\,\text{Sin}[\alpha]^4 + 2835\,t^5\,v0^5\,\text{Sin}[\alpha]^5))\Big\};$$

```
ParamCurve1 = Evaluate[XApproximate[t] /. numericalValues];
ParamCurve2 = Evaluate[Xsol[t] /. Flatten[theSolution] /. numericalValues];
ParametricPlot3D[
  {{ParamCurve1[[1]], ParamCurve1[[2]], ParamCurve1[[3]]},
   {ParamCurve2[[1]], ParamCurve2[[2]], ParamCurve2[[3]]}},
  {t, 0, 60}, PlotRange → {{-1000, 1000}, {0, 4500}, {0, 1500}}];
```



Il semble que le résultat obtenu avec la première méthode d'approximation (NDSolve de Mathematica) est très proche du résultat de la méthode de la perturbation, cependant on peut apercevoir une légère différence de la trajectoire sur la fin de celle-ci. Il pourrait être intéressant de faire un grossissement sur la fin de la trajectoire pour apprécier les résultats des deux méthodes d'approximation.

```
ParametricPlot3D[
  {{ParamCurve1[[1]], ParamCurve1[[2]], ParamCurve1[[3]]},
   {ParamCurve2[[1]], ParamCurve2[[2]], ParamCurve2[[3]]}},
  {t, 0, 60}, PlotRange → {{-30, 30}, {3000, 3300}, {0, 320}}];
```



Nous pouvons calculer la hauteur maximale, le temps mis par le projectile pour atteindre celle-ci, la durée du vol et la portée du projectile pour obtenir une vérification concrète de l'influence de la résistance de l'air sur un projectile en mouvement (il est nécessaire d'avoir validé les fonctions de l'annexe 3 avant d'exécuter les calculs à suivre).

```
XApproximate = {x[t] → 0, y[t] → t v0 Cos[α] + (t^3 v0^3 β^2 Cos[α]^3)/(3 m^2) -
    (t^4 v0^4 β^3 Cos[α]^4)/(4 m^3) + (t^5 v0^5 β^4 Cos[α]^5)/(5 m^4) - (t^2 β (v0^2 + v0^2 Cos[2 α]))/(4 m),
  z[t] → 1/2 (2 h - g t^2 + 2 t v0 Sin[α]) + (β (-g^2 t^4 + 4 g t^3 v0 Sin[α] - 6 t^2 v0^2 Sin[α]^2))/(12 m) +
    1/(45 m^2) (β^2 (-g^3 t^6 + 6 g^2 t^5 v0 Sin[α] - 15 g t^4 v0^2 Sin[α]^2 + 15 t^3 v0^3 Sin[α]^3)) +
    1/(2520 m^3) (β^3 (-17 g^4 t^8 + 136 g^3 t^7 v0 Sin[α] -
        476 g^2 t^6 v0^2 Sin[α]^2 + 840 g t^5 v0^3 Sin[α]^3 - 630 t^4 v0^4 Sin[α]^4)) +
    1/(14175 m^4) (β^4 (-31 g^5 t^10 + 310 g^4 t^9 v0 Sin[α] - 1395 g^3 t^8 v0^2 Sin[α]^2 +
        3465 g^2 t^7 v0^3 Sin[α]^3 - 4725 g t^6 v0^4 Sin[α]^4 + 2835 t^5 v0^5 Sin[α]^5))};

TimeForMaxHeight[XApproximate, t,
 {h → 1., v0 → 200., α → π/4, β → 10^-3, g → 9.80, m → 10., a → 0}]

13.5544

MaxHeight[XApproximate, t, {h → 1., v0 → 200., α → π/4, β → 10^-3, g → 9.80, m → 10., a → 0}]

929.595

FlightTime[XApproximate, t, {h → 1., v0 → 200., α → π/4, β → 10^-3, g → 9.80, m → 10., a → 0}]

-0.00706898

ShootingRange[XApproximate, t,
 {h → 1., v0 → 200., α → π/4, β → 10^-3, g → 9.80, m → 10., a → 0}]

-0.999755
```

Nous remarquons que la résistance de l'air a peu d'influence sur la durée et sur la hauteur maximale atteinte par le projectile, par contre elle montre toute son influence sur la portée du projectile. Nous pouvons donc considérer que la résistance de l'air porte son influence sur le projectile lorsque celui-ci chute et que sa vitesse augmente de nouveau après que celle-ci se soit annulée su l'axe Oz.

■ Annexe 3 : options de programmation

Cette annexe a pour objet le développement de sous-programmes caractéristiques à l'étude de la trajectoire d'un projectile, en particulier du calcul du point d'impact, la durée du vol du projectile, la hauteur maximale atteinte par le projectile ainsi que le temps mis pour atteindre celle-ci. On exécute `ClearAll["Global`*"]` pour effacer les définitions utilisées précédemment.

■ Détermination du temps necessaire pour atteindre le point le plus élevé

Le point le plus élevé de la trajectoire correspond à une composante nulle de la vitesse dans la direction $\vec{k}$, soit v_z=0, c'est-à-dire $(v_0 \times \sin(\alpha) + \frac{m \times g}{\beta}) \times e^{\frac{-\beta \times t}{m}} - \frac{m \times g}{\beta}$=0, la force de Coriolis étant négligée. Nous en déduisons t, le temps nécessaire pour atteindre l'altitude maximale [1] :

$$t=\frac{m}{\beta}\times\log\left(1+\frac{v_0\times\beta\times\sin(\alpha)}{m\times g}\right)$$

Nous allons tout d'abord déterminer le vecteur vitesse générant la trajectoire du projectile, puis nous allons résoudre sa composante sur $\vec{z}$ de telle sorte à déterminer le temps nécessaire pour atteindre le point le plus élevé.

```
TimeForMaxHeight[SolutionPositionVector_, t_, numericalValues_] :=
 t /. Flatten[FullSimplify[
     Solve[z'[t] == 0 /. Flatten[D[SolutionPositionVector, t]], t]]] /. numericalValues
```

■ Détermination de la hauteur maximale atteinte

Nous avons déterminer précédemment le temps nécessaire au projectile pour atteindre sa hauteur maximale. Il suffit de réinjecter la valeur de t trouver précédemment dans la composante normale du vecteur position à la surface de la Terre (z[t]) [1].

```
MaxHeight[SolutionPositionVector_, t_, numericalValues_] :=
 z[t] /. Flatten[FullSimplify[SolutionPositionVector]] /. t → t /. Flatten[FullSimplify[
     Solve[z'[t] == 0 /. Flatten[D[SolutionPositionVector, t]], t]]] /. numericalValues
```

■ Détermination de la durée du vol jusqu'à l'impact

Nous considérons que le point d'impact du projectile a lieu en z=a qui ne correspond pas forcément à la surface de la Terre (z=0).En cas de dénivellation de terrain il est possible de modifier l'altitude du point d'impact en modifiant le paramètre a. La durée du vol du projectile jusqu'à la Terre est donc la valeur de t pour laquelle on a une composante égale à a sur z[t][1].

```
FlightTime[SolutionPositionVector_, t_, numericalValues_] :=
 t /. Flatten[FullSimplify[Solve[z[t] == a /. Flatten[SolutionPositionVector], t]]] /.
  numericalValues
```

■ Détermination de la portée

Nous avons déterminé précédemment la durée du vol du projectile jusqu'à l'impact. Il suffit de réinjecter la valeur de t trouver précédemment dans la composante tangentielle à la surface terrestre du vecteur position y[t] [1].

```
ShootingRange[SolutionPositionVector_, t_, numericalValues_] :=
 y[t] /. Flatten[FullSimplify[SolutionPositionVector]] /. t → t /. Flatten[FullSimplify[
     Solve[z[t] == a /. Flatten[SolutionPositionVector], t]]] /. numericalValues
```

■ Applications numériques

Nous pouvons à titre d'illustrations et de tests faire une application numérique des quatres fonctions précédentes et comparer les résultats à la courbe représentative de la trajectoire du projectile.

```
TimeForMaxHeight[{x[t] → 0, y[t] → (1 - e^(-t β/m)) m v0 Cos[α] / β,
  z[t] → 1/β^2 (g m^2 - g m t β + h β^2 + m v0 β Sin[α] - e^(-t β/m) m (g m + v0 β Sin[α]))},
 t, {h → 1., v0 → 200., α → π/4, β → 10^-3, g → 9.80, m → 10., a → 0}]
```

```
14.4203
```

```
MaxHeight[{x[t] → 0, y[t] → (1 - e^(-t β/m)) m v0 Cos[α]/β,
   z[t] → 1/β^2 (g m^2 - g m t β + h β^2 + m v0 β Sin[α] - e^(-t β/m) m (g m + v0 β Sin[α]))},
 t, {h → 1., v0 → 200., α → π/4, β → 10^-3, g → 9.80, m → 10., a → 0}]
```

```
1020.43
```

```
FlightTime[{x[t] → 0, y[t] → (1 - e^(-t β/m)) m v0 Cos[α]/β,
   z[t] → 1/β^2 (g m^2 - g m t β + h β^2 + m v0 β Sin[α] - e^(-t β/m) m (g m + v0 β Sin[α]))},
 t, {h → 1., v0 → 200., α → π/4, β → 10^-3, g → 9.80, m → 10., a → 0}]
```

```
28.8547
```

```
ShootingRange[{x[t] → 0, y[t] → (1 - e^(-t β/m)) m v0 Cos[α]/β,
   z[t] → 1/β^2 (g m^2 - g m t β + h β^2 + m v0 β Sin[α] - e^(-t β/m) m (g m + v0 β Sin[α]))},
 t, {h → 1., v0 → 200., α → π/4, β → 10^-3, g → 9.80, m → 10., a → 0}]
```

```
4074.79
```

En comparaison à la courbe les résultats numériques concernant la hauteur maximale et la portée du projectile semblent du même ordre de grandeur, par conséquent les durées sont cohérentes avec l'étude.

■ Annexe 4 : action de la force de Coriolis sur la trajectoire d'un projectile

Cette annexe n'a qu'une valeur de tentative de programmation. Nous avons essayé de résoudre l'équation différentielle en prenant en compte cette force. La complexité de l'équation implique une résolution par approximation, c'est porquoi nous avons opté pour l'utilisation de la fonction NDsolve. D'après la loi de Newton, on a :

$$m \times \frac{d^2 \vec{r}}{dt^2} = -m \times \vec{g} - \beta \times \vec{r} - m \times pv\left[\frac{d \vec{r}}{dt}, \vec{r}\right]$$

où pv[a,b] définit le produit vectoriel entre a et b. On exécute `ClearAll["Global`*"]` pour effacer les définitions utilisées précédemment. On enregistre l'équation différentielle gérant la trajectoire du projectile sous l'action de la force de Coriolis, puis on la projette sur les axes du repère.

```
theEquationWithCoriolis :=
 m * D[VCoriolis[t], t] + β * VCoriolis[t] + 2 * m * Cross[VCoriolis[t], XCoriolis[t]] == m * G
XCoriolis[t_] = Through[{xCor, yCor, zCor}[t]];
VCoriolis[t_] = Through[{vxCor, vyCor, vzCor}[t]];
G = {0, 0, -g};
theSystemWithCoriolis = Thread[theEquationWithCoriolis]
```

```
{β vxCor[t] + 2 m (-vzCor[t] yCor[t] + vyCor[t] zCor[t]) + m vxCor'[t] == 0,
 β vyCor[t] + 2 m (vzCor[t] xCor[t] - vxCor[t] zCor[t]) + m vyCor'[t] == 0,
 β vzCor[t] + 2 m (-vyCor[t] xCor[t] + vxCor[t] yCor[t]) + m vzCor'[t] == -g m}
```

On fixe les conditions initiales et les différentes valeurs des paramètres du système.

```
initialPositionWithCoriolis = Thread[XCoriolis[0] == {0, 0, h}];
initialVelocityWithCoriolis = Thread[VCoriolis[0] == {0, v0 * Cos[α], v0 * Sin[α]}];
numericalValues = {h → 1., v0 → 20., α → π/4, β → 0.001, g → 9.80, m → 10.};
```

On définit la fonction Deri qui définit la vitesse comme la dérivée de la position.

```
Deri = Thread[VCoriolis[t] == D[XCoriolis[t], t]];
```

On résout le système d'équations suivant les axes du repère en prenant en compte les conditions initiales et les valeurs des différents paramètres du système.

```
theSolutionWithCoriolis = FullSimplify[
  NDSolve[Flatten[{theSystemWithCoriolis, Deri,
      initialPositionWithCoriolis, initialVelocityWithCoriolis} /. numericalValues],
   {VCoriolis[t][[1]], VCoriolis[t][[2]], VCoriolis[t][[3]],
    XCoriolis[t][[1]], XCoriolis[t][[2]], XCoriolis[t][[3]]},
   {t, 0, 100}, WorkingPrecision → 8, MaxSteps → 1000]]
```

```
No more memory available.
Mathematica kernel has shut down.
Try quitting other applications and then retry.
```

```
ParametricPlot3D[
  Evaluate[XCoriolis[t] /. Flatten[theSolutionWithCoriolis] /. numericalValues],
  {t, 0, 10.88}, PlotRange → {{-50, 50}, {0, 50}, {0, 50}}];
```

Peut-être la forme du système différentiel (trois équations différentielles couplées) implique-t-elle une surabondance de calculs donc une utilisation très importante de la mémoire que Mathematica n'arrive pas à gérer ; ou alors, il y a un erreur de programme…