Déviation par la force de Coriolis

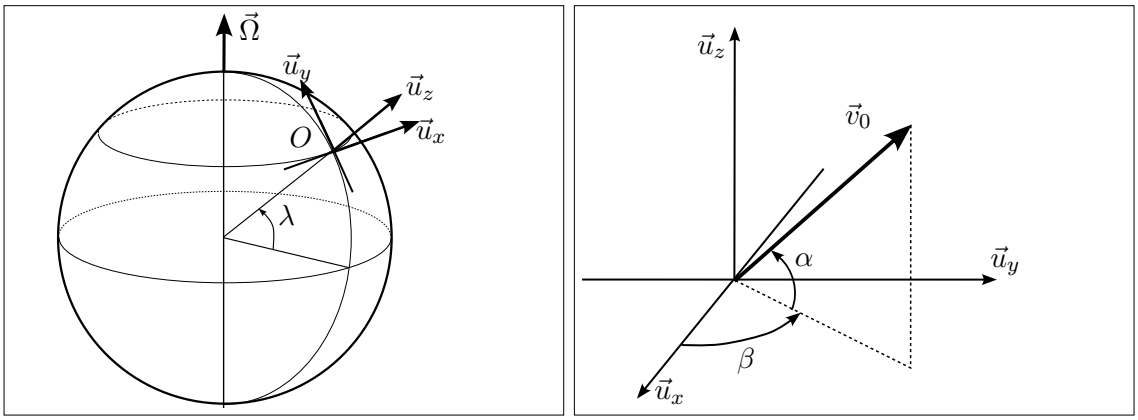
I Problème étudié et mise en équation

On considère un projectile de masse m lancé avec un vitesse $\vec{v}_0$ depuis un point M_0 du référentiel terrestre. On étudie sa trajectoire avec les hypothèses suivantes.

— Le référentiel géocentrique est galiléen mais le référentiel terrestre $\mathcal{R}_T$ ne l'est pas.

— La surface de la terre près de M_0 est confondue avec son plan tangent et le champ de pesanteur $\vec{g}$, qui prend en compte l'effet d'inertie d'entraînement, est uniforme le long de la trajectoire.

On utilise un repère cartésien $(O, \vec{u}_x, \vec{u}_y, \vec{u}_z)$ avec $\vec{u}_z$ dirigé selon la verticale ascendante, $\vec{u}_x$ tangent au parallèle passant par O et dirigé vers l'est, $\vec{u}_y$ tangent au méridien passant par O et dirigé vers le nord. Dans cette base, le vecteur rotation de la Terre s'écrit $\vec{\Omega} = \Omega \cos\lambda\, \vec{u}_y + \Omega \sin\lambda\, \vec{u}_z$.



Le vecteur vitesse initiale $\vec{v}_0$ est défini par sa norme v_0 et les angles (α, β). L'angle α désigne l'élévation du tir par rapport à l'horizontal et β donne son azimut. Par exemple, $\beta = 0$ correspond à un tir vers l'est et $\beta = \pi/2$ à un tir vers le nord.

Dans $\mathcal{R}_T$, le projectile est soumis à son poids $m\vec{g}$ et à la force de Coriolis $\vec{F}_{ic} = -2m\vec{\Omega} \wedge \vec{v}$. On tient compte d'une force de frottement aérodynamique $\vec{F}_a$ proportionnelle au carré de la vitesse : $\vec{F}_a = -Av\,\vec{v}$. En notant $B = A/m$, l'application du principe fondamental de la dynamique conduit aux équations couplées

$$\begin{cases} \dfrac{dv_x}{dt} = -2\Omega v_z \cos\lambda + 2\Omega v_y \sin\lambda - Bv\, v_x \\ \dfrac{dv_y}{dt} = -2\Omega v_x \sin\lambda - Bv\, v_y \\ \dfrac{dv_z}{dt} = 2\Omega v_x \cos\lambda - g - Bv\, v_z \quad . \end{cases} \tag{1}$$

qui déterminent de manière unique l'évolution de la vitesse $\vec{v}(t)$ puis, par intégration, la position $M(t)$ du projectile.

Dans le cas particulier où les frottements sont négligés ($B = 0$), le système d'équations différentielles (1) est linéaire et on peut le résoudre exactement par des techniques d'algèbre linéaire. Lorsque l'effet de la force de Coriolis est « assez petit », un calcul approximatif simplifie le travail comme nous l'avons vu sur l'exemple de la déviation d'une chute libre. Cependant, dans le cas plus réaliste où les frottements sont pris en compte, aucune résolution explicite les équations (1) n'est possible. On recourt alors à une approche numérique qui consiste à trouver des valeurs approchées de $\vec{v}$ et $\vec{OM}$ en certains instants t_i, $i \in \mathbb{N}$. Cette démarche figure explicitement au programme de physique de la classe de PCSI et nous avons ici une occasion de la réviser. Les explications qui suivent sont assez succinctes et je vous invite à consulter vos cours de première année pour plus de détails.

II Résolution numérique d'un système différentiel

On considère une équation différentielle du type $Y' = F(Y,t)$. À partir d'une valeur initiale $Y(0) = Y_0$ de la fonction inconnue, on calcule des valeurs approchées $Y_i \simeq Y(t_i)$ en prenant $\delta t = t_{i+1} - t_i$ assez petit. Chacune se déduit de la précédente en assimilant la courbe d'équation $Y(t)$ avec sa tangente au point (t_i, Y_i), ce qu'exprime la relation

$$Y(t_{i+1}) \simeq Y(t_i) + \delta t\ Y'(t_i) \quad \text{qui devient} \quad Y_{i+1} = Y_i + \delta t\ F(Y_i, t_i) \qquad . \tag{2}$$

Il est ici sous-entendu que la variable t est réelle. Par contre, $Y(t)$ et $F(Y,t)$ peuvent tout aussi bien être des réels, dans le cas d'une équation différentielle ordinaire, que des élements de $\mathbb{R}^n$ pour un système différentiel de n équations portant sur n fonctions inconnues. En dimension 3 par exemple, $Y(t) = (x(t), y(t), z(t))$ et $Y'(t)$ désigne le vecteur $(x'(t), y'(t), z'(t))$. La suite de l'énoncé se place dans ce cas « vectoriel » que vous avez également dû étudier en première année. Les vecteurs Y_i et $F(Y_i, t_i)$ sont représentés par des tableaux de numpy de n flottants, ce qui permet la « vectorisation » de certaines opérations, comme l'addition terme à terme de deux tableaux et la multiplication d'un tableau par un nombre.

1. Écrire la fonction `Euler(fonction_derive, Y_0, t)` qui exécute la méthode d'Euler. Ses paramètres sont :

- une fonction `fonction_derive(Y, t)` qui, à partir du tableau `Y` représentant Y_i, calcule et renvoie, sous forme de tableau numpy, le vecteur dérivé représentant $F(Y_i, t)$;
- le tableau numpy `Y_0` contenant les valeurs des fonctions inconnues à $t = 0$;
- le tableau numpy `t` de N valeurs équidistantes t_i pour $i \in [\![0, N-1]\!]$.

Cette fonction renvoie un tableau de taille $N \times n$ dans lequel la ligne d'indice i représente Y_i. En particulier la première ligne de ce tableau est `Y_0`.

La méthode d'Euler constitue le B-A BA des méthodes numériques de résolution des équations différentielles. Il existe d'autres méthodes plus élaborées et beaucoup plus précises, telles les célèbres méthodes de Runge et Kutta qui perfectionnent la méthode d'Euler en remplaçant dans la relation (2) la pente $F(Y_i, t_i)$ par une pente moyenne issue de plusieurs évaluations de sa valeur entre (t_i, Y_i) et (t_{i+1}, Y_{i+1}). La fonction `odeint` du module `scipy.integrate` met en œuvre des méthodes de ce type et permet à l'utilisateur d'obtenir sans effort, avec une grande précision, des valeurs approchées d'une fonction solution d'une équation différentielle. Elle s'utilise de la manière suivante.

```
solution = odeint(fonction_derive, Y_0, t)
```

Cette fonction prend les mêmes paramètres que `Euler` et renvoie le même genre résultat, à la différence près que les t_i peuvent ne pas être équidistants.

Avant de passer à la partie suivante, vérifions sur un exemple que la fonction `Euler` est correcte en comparant son résultat à celui de `odeint`. On résout le système

$$\begin{cases} x'(t) = -y(t) \\ y'(t) = 2x(t) \end{cases} \quad \text{avec} \quad \begin{cases} x(0) = 1 \\ y(0) = 1 \end{cases} \quad .$$

La solution exacte est connue ; elle s'écrit

$$\begin{cases} x(t) = \cos\sqrt{2}t\ - \dfrac{1}{\sqrt{2}} \sin\sqrt{2}t \\ y(t) = \cos\sqrt{2}t + \sqrt{2}\sin\sqrt{2}t \end{cases}$$

et on peut l'utiliser comme point de comparaison pour tester la validité des méthodes numériques.

2. Coder la fonction `derive_essai(Y,t)` qui prend en argument le tableau numpy `Y` à deux éléments contenant les valeurs $x(t)$ et $y(t)$, et qui renvoie le tableau de numpy contenant $x'(t)$ et $y'(t)$.

3. Résoudre une première fois le système avec la méthode d'Euler en prenant 100 dates t_i sur l'intervalle $[0, 4]$. Stocker les valeurs approchées de $x(t_i)$ et $y(t_i)$ dans des tableaux de numpy `x_E` et `y_E`.

4. Résoudre une seconde fois le système avec `odeint` sur les mêmes points t_i. Stocker les valeurs approchées de $x(t_i)$ et $y(t_i)$ dans des tableaux de numpy `x_odeint` et `y_odeint`.

5. Sur un premier graphe, tracer la solution exacte $x(t)$, la solution approchée fournie par la méthode d'Euler et la solution approchée fournie par `odeint`. Sur un second graphe, faire de même pour $y(t)$. Afin d'uniformiser la présentation des courbes et de faciliter leur lecture par le professeur, utiliser les lignes de code suivantes.

```
plt.figure(0)
plt.clf()
plt.subplot(2, 1, 1)
plt.title('Comparaison␣des␣résultats␣numériques␣au␣résultat␣exact')
plt.plot(t, x_E, color = 'red', label = 'Euler')
plt.plot(t, x_odeint, label = 'odeint')
plt.plot(t, x_exact, color = 'black', linestyle = 'dashed', linewidth = 2,
         label = 'exact')
plt.ylabel('x(t)')
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(t, y_E, color = 'red', label = 'Euler')
plt.plot(t, y_odeint, label = 'odeint')
plt.plot(t, y_exact, color = 'black', linestyle = 'dashed', linewidth = 2,
         label = 'exact')
plt.ylabel('y(t)')
plt.xlabel('t')
plt.legend()
```

III Calcul d'une trajectoire

Nous allons écrire un code Python qui traite comme variables globales toutes les constantes du problème, c'est à dire toutes les grandeurs physiques introduites dans la partie I, en particulier $\Omega = 7,29.10^{-5}\,\text{rad.s}^{-1}$ et $g = 9,81\,\text{m.s}^{-2}$. On introduit aussi deux variables globales booléennes `coriolis` et `frottements`. La première indique si l'on souhaite ou non prendre en compte la force de Coriolis, la seconde indique si l'on souhaite prendre en compte les frottements. Le vecteur vitesse à un instant donné est représenté par un tableau de numpy de trois flottants.

1. Coder la fonction `derive_v(v, t)` qui prend en argument le vecteur vitesse et une date t, et qui renvoie le vecteur accélération à cette même date. Dans nos exemples, t ne joue aucun rôle mais ce format assure la compatibilité avec `odeint`. On envisagera différents cas selon les valeurs de `coriolis` et `frottements`.

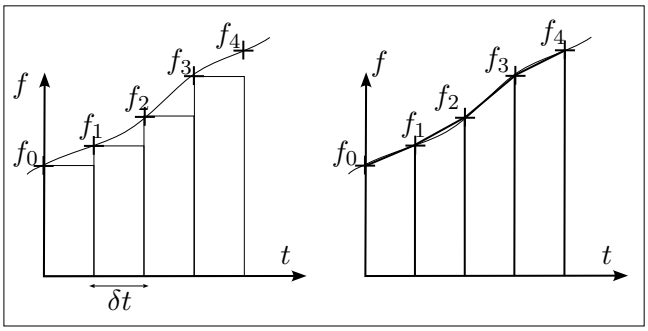
2. La résolution numérique de l'équation du mouvement fournit des valeurs approchées du vecteur vitesse. La position du projectile s'en déduit par intégration :

$$\vec{OM}(t) = \vec{OM}_0 + \int_0^t \vec{v}(t)\,dt \qquad x(t) = x_0 + \int_0^t v_x(t)\,dt \qquad \text{et de même pour } y(t) \text{ et } z(t) \quad .$$

À partir de valeurs numériques approchées, l'intégration peut se faire selon la méthode des rectangles qui approxime l'aire sous la courbe par celle de rectangles de largeur δt et de hauteurs f_i (se reporter au dessin et au programme de PCSI) :

$$\int_0^{t_i} f(t)dt \simeq \sum_{k=0}^{i-1} f(t_i)\delta t \qquad \text{pour} \quad i \geqslant 1 \qquad .$$

Coder la fonction `integre_rectangle(f, t)` prenant en paramètre un tableau de numpy `f` contenant les N valeurs d'une fonction à valeurs réelle f aux instant t_i contenus dans le tableau t, et renvoyant les N valeurs de l'intégrale calculée par la méthode des rectangles, c'est à dire N valeurs approchées aux instants t_i de la primitive de f qui s'annule en $t_0 = 0$. On demande une complexité en $O(N)$.



3. Tout comme la méthode d'Euler, la méthode des rectangles est assez imprécise [1]. La méthode des trapèzes (partie droite du dessin) permet une amélioration sensible et s'appuie sur la relation

$$\int_0^{t_i} f(t)dt \simeq \delta t \sum_{k=1}^{i} \frac{f_{k-1}+f_k}{2} \quad \text{pour } i \geqslant 1 \quad .$$

On considère ici que la fonction f est vectorielle en dimension n, et sa représentation numérique est un tableau `f` de N lignes et n colonnes. Son intégrale jusqu'à t_i est donc un vecteur de n composantes. Coder la fonction `integre_trapeze(f, t)` qui renvoie le tableau de N lignes et n colonnes contenant les N valeurs approchées de la primitive de f qui s'annule en $t_0 = 0$. Dans la suite, vous pourrez vous assurer que votre fonction est correcte en comparant son résultat à celui de `integr.cumtrapz(f, t, axis = 0, initial = t[0])`.

4. Coder la fonction `trouve_position_vitesse(M0, V0, t)` qui prend en paramètres la vitesse et la position initiales (tableaux de numpy de trois flottants) ainsi que le tableau `t` des dates t_i, et qui renvoie dans un tableau de N lignes et 6 colonnes les valeurs successives du vecteur position et du vecteur vitesse. Pour concaténer selon leurs colonnes deux tableaux bidimensionnels possédant le même nombre de lignes, on utilisera `np.concatenate((T1, T2), axis=1)`.

IV Applications

Dans toute la suite, on fixe $\Omega = 7,29.10^{-5}\,\text{rad.s}^{-1}$, $g = 9,81\,m.s^{-2}$, $\lambda = 45\,°$, $m = 23,6\,\text{kg}$. Les autres paramètres seront modifiés au fil des exemples.

1. Dans un premier temps, on exclut la force de Coriolis et les frottements. Étudier « à la main » la trajectoire et exprimer la portée du tir. Procéder aux applications numériques dans le code Python et imprimer les résultats dans la console.

2. Pour $v_0 = 100\,\text{m.s}^{-1}$, $\alpha = 45\,°$, $\beta = 0$, $\lambda = 45\,°$, utiliser les fonctions précédentes et obtenir numériquement la trajectoire. Tracer les graphes de $x(t)$, $z(t)$ et $z(x)$ et vérifier la valeur de la portée. Pour les représentations graphiques, s'inspirer des lignes ci-dessous en les modifiant et complétant au besoin. La figure 1 permet d'obtenir une vue en trois dimensions.

```
plt.figure(0)
plt.clf()
plt.subplot(3, 1, 1)
plt.plot(t, x)
plt.ylabel('x')
plt.subplot(3, 1, 2)
plt.plot(t, y)
plt.ylabel('y')
plt.xlabel('t')
plt.subplot(3, 1, 3)
plt.plot(x, z)
plt.ylabel('z')
plt.xlabel('x')
plt.hlines(0, 0, np.max(x), color =
    'black')

plt.figure(1)
ax = plt.axes(projection = '3d')
ax.set_zlim(0, 120)
ax.plot3D(x, y, z)
ax.scatter3D([x0], [y0], [z0], color
    = 'black', s = 20)
```

1. Appliquer la méthode des rectangles revient à résoudre $F' = f$ par la méthode d'Euler

3. À la latitude $\lambda = 45\,°$, on abandonne un objet en chute libre depuis une hauteur de 100 m. En négligeant les frottements de l'air, représenter sa trajectoire et déterminer son point de chute. Cet exemple doit vous permettre de valider vos codes par comparaison au résultat établi en cours (déviation de 1,55 cm vers l'est).

4. On considère un obus de masse $m = 23,6\,\text{kg}$ tiré plein sud ($\beta = -90\,°$) avec une élévation $\alpha = 40\,°$ et une vitesse $v_0 = 870\,\text{m.s}^{-1}$. En tenant compte des frottements avec $A = 2,04.10^{-3}$ SI, calculer sa trajectoire avec ou sans l'influence de la force de Coriolis. Tracer en trois dimensions les deux trajectoires côte à côte. Trouver le point d'impact au sol et le faire apparaître (avec `scatter3D`). Quel écart la force de Coriolis provoque-t-elle ? Pour trouver le point d'impact et le représenter, on utilisera les lignes ci-dessous.

```
i_portee = np.argmin(np.abs(z[100:])) # [100:] élimine la solution triviale i=0
t_portee = t[i_portee]
y_portee = y[i_portee]
x_portee = x[i_portee]
print('t_portee : ', t_portee, 'y_portee', y_portee, 'x_portee', x_portee)
ax.scatter3D(x[i_portee], y[i_portee], z[i_portee], color = 'black', s = 20)
ax.text(x[i_portee], y[i_portee], z[i_portee], 'impact x = ' +
    str(x[i_portee])[0:5] + 'm')
```

V Obtention simultanée de la vitesse et de la position

L'approche exposée ci-dessus considère le vecteur vitesse comme fonction inconnue dans le système différentiel (1). On le trouve numériquement, puis la position s'en déduit dans un second temps par intégration numérique. Une autre approche consiste à considérer que l'inconnue est le vecteur position $(x(t), y(t), z(t))$ et à réécrire le système (1) sous la forme

$$\begin{cases} \dfrac{d^2x}{dt^2} = -2\Omega\dfrac{dz}{dt}\cos\lambda + 2\Omega\dfrac{dy}{dt}\sin\lambda - Bv\dfrac{dx}{dt} \\ \dfrac{d^2y}{dt^2} = -2\Omega\dfrac{dx}{dt}\sin\lambda - Bv\dfrac{dy}{dt} \\ \dfrac{d^2z}{dt^2} = 2\Omega\dfrac{dx}{dt}\cos\lambda - g - Bv\dfrac{dz}{dt} \end{cases} \quad \text{avec} \quad v = \sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2 + \left(\frac{dz}{dt}\right)^2} \quad . \qquad (3)$$

Dans le problème étudié ici, seules les dérivées de la position interviennent dans le membre de droite et ce changement de variable ne semble pas très pertinent. Mais si on doit traiter des problèmes où forces dépendant de la position, les coordonnées (x, y, z) interviennent dans le membre de droite et cette approche est inévitable. Dans notre exemple, ce serait par exemple le cas si on voulait considérer un champ de pesanteur g dépendant de x, y et z.

Le système (3) est ordre 2, alors qu'il était d'ordre 1 pour la vitesse. Pour le résoudre, on se ramène à un système d'ordre 1 selon une technique qui figure au programme de PCSI. On introduit le vecteur inconnu E à 6 composantes $(E_0 = x(t), E_1 = y(t), E_2 = z(t), E_3 = \dot{x}(t), E_4 = \dot{y}(t), E_5 = \dot{z}(t))$. Nous le nommerons « vecteur d'état » puisqu'il regroupe la position et la vitesse du projectile. Il est solution du système d'ordre 1 suivant :

$$\begin{cases} \dfrac{dE_0}{dt} = E_3 \\ \dfrac{dE_1}{dt} = E_4 \\ \dfrac{dE_2}{dt} = E_5 \\ \dfrac{dE_3}{dt} = -2\Omega E_5\cos\lambda + 2\Omega E_4\sin\lambda - Bv\,E4 \\ \dfrac{dE_4}{dt} = -2\Omega E_3\sin\lambda - Bv\,E_4 \\ \dfrac{dE_5}{dt} = 2\Omega E_3\cos\lambda - g - Bv\,E_5 \end{cases} \quad \text{avec} \quad v = \sqrt{E_3^2 + E_4^2 + E_5^2} \quad . \qquad (4)$$

1. Coder la fonction `derive_etat(E, t)` prenant en argument le tableau de numpy à 6 éléments représentant l'état E à l'instant t, et renvoyant le tableau de numpy à 6 éléments contenant les composantes de dE/dt.

2. On reprend l'exemple du tir d'un obus. Utiliser `odeint` pour résoudre le système différentiel (3). Tracer la trajectoire avec ou sans force de Coriolis et calculer à nouveau la déviation que cette force provoque.