TP/Cours : Résolution d'équations différentielles

1 Intégration numérique

L'intégration numérique consiste à obtenir de façon approchée l'aire sous la courbe d'une fonction f sur un intervalle borné $[a, b]$, à partir d'un calcul de $f(x)$ en un nombre fini de points x_k.

La répartition des points en abscisse est généralement uniforme. Cela revient à choisir un pas d'échantillonnage constant ($x_{k+1} - x_k = h$) mais il existe des méthodes à pas variable, ou encore à pas adaptatif.

L'intégration des polynômes étant très simple, l'opération consiste généralement à construire une interpolation polynomiale par morceaux (de degré plus ou moins élevé) puis d'intégrer le polynôme sur chaque morceau.

La précision de l'intégration numérique peut s'améliorer en augmentant le nombre de points n (autrement dit, en diminuant le pas d'échantillonnage h) ou en augmentant le degré de l'interpolation polynomiale (sous réserve de bonnes propriétés de continuité de la courbe).

1.1 Méthode d'intégration des rectangles (interpolation de degré 0)

La méthode des rectangles consiste à considérer la fonction interpolante *constante* sur chaque intervalle de largeur $x_{k+1} - x_k$ et égale à la valeur $f(x_k)$ prise par la fonction f au début de cet intervalle.

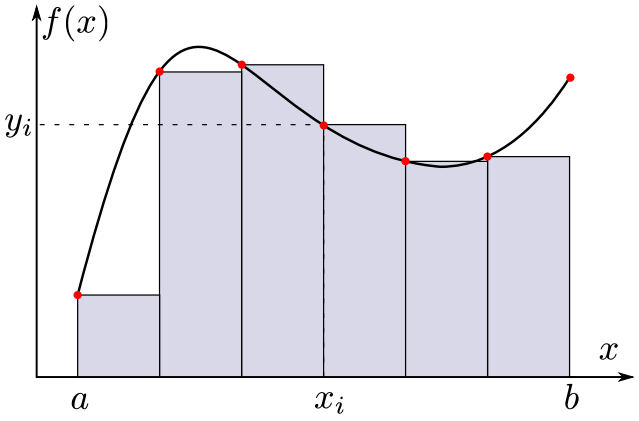

FIGURE 1 – Méthode des rectangles.

La valeur approchée de l'intégrale est alors :

$$I = \int_a^b f(x)dx \approx \sum_{k=1}^{n-1} y_k \left(x_{k+1} - x_k\right)$$

Lorsque la répartition des points en abscisse est uniforme (pas d'intégration constant) tel que $x_{k+1} - x_k = h$, on obtient la formule simplifiée :

$$I \approx h \sum_{k=1}^{n-1} y_k$$

À noter que pour n points, il y a $n - 1$ intervalles à sommer. Par conséquent, la dernière valeur $f(x_n)$ n'est pas prise ne compte.

Q 1. Écrire une fonction en Python prenant en argument deux listes X et Y de taille n contenant les abscisses x_k et les ordonnées $f(x_k)$ associées d'une fonction f à intégrer et qui retourne une valeur approchée de son intégrale entre x_1 et x_n par la méthode des rectangles.

Q 2. Écrire une fonction en Python prenant en argument une fonction f, deux abscisses a et b définissant l'intervalle sur lequel on veut intégrer la fonction f, un entier n représentant le nombre d'intervalles d'intégration et qui retourne une valeur approchée de son intégrale entre a et b.

Q 3. En déduire une valeur approchée de l'intégrale de $f : x \longmapsto \dfrac{\ln\left(\dfrac{x}{4}\right)}{x} + x^2$ entre $a = 0.1$ et $b = 5$ pour $n = 10, 50, 100, 1000$ et 5000 points.

1.2 Méthode d'intégration des trapèzes (interpolation de degré 1)

La méthode des trapèzes s'appuie sur une interpolation linéaire sur chaque intervalle de largeur $x_{k+1} - x_k$. Cela revient à dire que sur chaque intervalle, on approxime f par la fonction affine $\tilde{f}$ tel que :

$$\tilde{f}(x) = y_k + \frac{y_{k+1} - y_k}{x_{k+1} - x_k}(x - x_k) \quad \text{pour} \quad x \in \left[x_k, x_{k+1}\right]$$

L'aire I_k de chaque trapèze étant la somme de l'aire d'un rectangle de base $x_{k+1} - x_k$ et de hauteur y_k et de l'aire d'un triangle rectangle de base $x_{k+1} - x_k$ et de hauteur $y_{k+1} - y_k$, on en déduit la valeur approchée de l'intégrale :

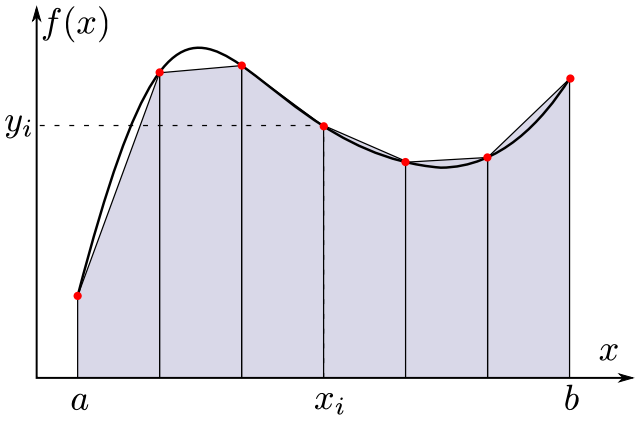

FIGURE 2 – Méthode des trapèzes.

$$I = \int_a^b f(x)dx \approx \sum_{k=1}^{n-1} \frac{y_i + y_{i+1}}{2} \left(x_{k+1} - x_k\right)$$

Lorsque la répartition des points en abscisse est uniforme (pas d'intégration constant) tel que $x_{k+1} - x_k = h$, on obtient la formule simplifiée :

$$I \approx h\frac{y_1}{2} + h\sum_{k=2}^{n-1} y_k + h\frac{y_n}{2}$$

Q 4. Écrire une fonction en Python prenant en argument deux listes X et Y de taille n contenant les abscisses x_k et les ordonnées $f(x_k)$ associées d'une fonction f à intégrer et qui retourne une valeur approchée de son intégrale entre x_1 et x_n par la méthode des trapèzes.

Q 5. En utilisant les fonctions fournies dans `comparaison_integration.py`, compléter les quelques lignes de code manquantes afin de comparer les vitesses de convergence de ces deux méthodes d'intégration. Laquelle est la plus rapide ? Estimer graphiquement une borne de $\|I - I_{approx}(n)\|$ dans les deux cas.

2 Dérivation numérique

La dérivation numérique consiste à obtenir de façon approchée la pente de la courbe représentant la fonction f sur un intervalle borné $[a, b]$, à partir d'un calcul de $f(x)$ en un nombre fini de points x_k.

La répartition des points en abscisse est généralement uniforme. Cela revient à choisir un pas d'échantillonnage constant $x_{k+1} - x_k = h$) mais il existe des méthodes à pas variable, ou encore à pas adaptatif.

La dérivation des polynômes étant très simple, l'opération consiste généralement à construire une interpolation polynomiale par morceaux (de degré plus ou moins élevé) puis de dériver le polynôme sur chaque morceau.

2.1 Dérivée première

L'estimation de la dérivée la plus simple (et la plus courante) consiste à calculer la pente à partir du point courant x_k et du point précédent x_{k-1} ou suivant x_{k+1} en faisant une approximation linéaire de la fonction f sur l'intervalle d'intérêt : $[x_{k-1}, x_k]$ pour la dérivée « arrière » et $[x_k, x_{k+1}]$ pour la dérivée « avant ».

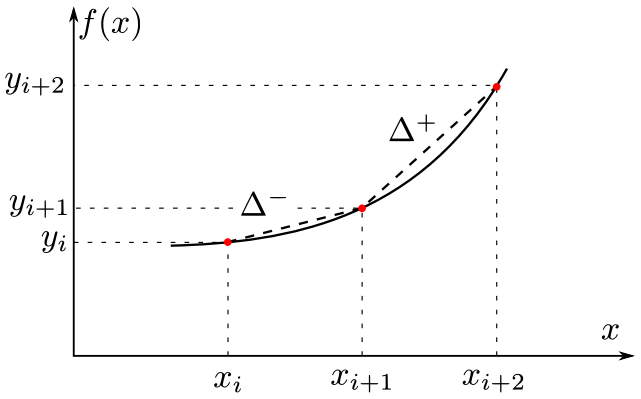

FIGURE 3 – Dérivée première.

L'estimation de la dérivée au point x_{i+1} peut alors se calculer par :

– différence avant : pente de la droite Δ^+

$$D_i = \frac{y_{i+2} - y_{i+1}}{x_{i+2} - x_{i+1}}$$

– différence arrière : pente de la droite Δ^-

$$D_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$$

Notons aussi que le calcul de la dérivée à partir de n points conduit à un tableau de valeur de dimension $n-1$.

Évidemment, lorsqu'il s'agit de dériver une fonction temporelle « en temps réel », le point suivant x_{i+2} n'est pas encore connu donc seule la différence arrière peut être calculée.

Q 6. Écrire une fonction en Python prenant en argument deux listes X et Y de taille n contenant les abscisses x_k et les ordonnées $f(x_k)$ associées d'une fonction f à dériver et qui retourne la liste des valeurs approchées de sa dérivée aux points x_k en utilisant un schéma de dérivation « arrière ».

Q 7. Calculer numériquement la dérivée de la fonction $f : x \longmapsto \dfrac{\ln\left(\dfrac{x}{4}\right)}{x} + x^2$ entre $a = 0.1$ et $b = 5$ pour $n = 10, 50, 100$ et 1000 points.

Q 8. En utilisant la librairie `matplolib.pyplot`, comparer graphiquement la dérivée numérique obtenue avec le résultat théorique pour les différentes valeurs de pas de dérivation précédentes.

Remarque Si la fonction à dériver présente une discontinuité, c'est à dire une variation brutale entre deux points (puisque la fonction est donnée par une liste de point), il faut bien noter que d'un point de vue théorique, une discontinuité conduit à une distribution de Dirac après dérivation, et d'un point de vue numérique, la discontinuité conduit à un pic (positif ou négatif) dont la valeur dépend du saut et du pas h de calcul. La valeur atteinte est donc d'autant plus grande que h est petit, et doit donc être interprétée avec la plus grande vigilance...

2.2 Influence du bruit de mesure

Lorsque la courbe est issue d'une mesure expérimentale, elle est généralement entachée d'un léger bruit, qui peut devenir catastrophique pour l'évaluation numérique de la dérivée. En effet, bien que les points de mesure restent « en moyenne » au voisinage de la valeur mesurée, il existe des fluctuations rapides (à la fréquence d'échantillonnage) entre les points successifs (voir zoom sur la FIGURE 4).

Or, puisque le calcul numérique de la dérivée conduit à déterminer la pente entre deux points successifs, le signal dérivé est fortement perturbé et cache les évolutions lentes du signal (i.e. lentes devant la période d'échantillonnage).

Deux solutions faciles à réaliser en pratique sont possibles :

– Filtrer (ou moyenner) le signal d'origine pour supprimer l'essentiel du bruit, puis dériver.

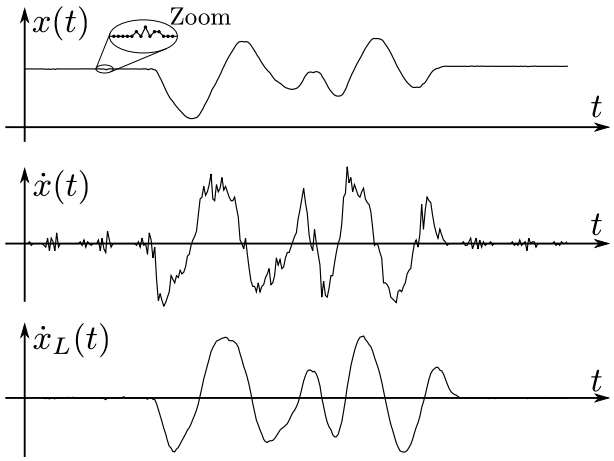

FIGURE 4 – Mesure d'une position au cours du temps $x(t)$ par un capteur potentiométrique, dérivée arrière $\dot{x}(t)$ et dérivée arrière lissée en effectuant la différence sur 10 pas.

– Calculer la dérivée sur un temps plus long que le temps d'échantillonnage, en calculant la pente entre deux points espacés de k pas (solution adoptée pour $\dot{x}_L(t)$ sur la FIGURE 4, avec $k = 10$).

Q 9. Modifier la fonction Python précédente pour qu'elle puisse calculer la dérivée d'une fonction f avec un pas variable k à partir de deux listes X et Y de taille n contenant les abscisses x_k et les ordonnées $f(x_k) + b_k$ associées (où b_k représente le bruit de mesure) en utilisant un schéma de dérivation arrière.

Q 10. Compléter le fichier Python `derivation_influence_bruit_mesure.py`, pour afficher la dérivée numérique de f à partir des listes `mesure_X.npy` et `mesure_Y.npy` fournies pour différents pas $k = 1$, 5 et 10.

Dans les deux cas, le signal dérivé sera entaché d'un retard sur le signal d'origine, ce qui oblige à trouver un compromis entre la qualité du signal dérivé et le retard. Usuellement, k est de l'ordre de 5 à 20 pas selon le niveau de bruit, le pas d'échantillonnage, les fréquences à observer dans le signal et le retard admissible.

3 Intégration des équations différentielles

3.1 Mise en place du problème

Une grande part des problèmes scientifiques se modélisent par des équations différentielles dont on cherche la solution pour dimensionner ou comprendre le phénomène.

Quand les équations sont simples (linéaires à coefficients constants d'ordre 1 ou 2), la résolution analytique est aisée, mais la plupart des modélisations conduisent à des équations différentielles plus complexes, voire non linéaires. Il faut alors les résoudre numériquement en utilisant des méthodes pour en déterminer une solution approchée.

Dans le cadre du programme de CGPE, nous nous limiterons aux *équations différentielles ordinaires explicites*, communément appelées ODE (Ordinary Differential Equation en anglais). Ce sont des équations différentielles dont la fonction inconnue $y(t)$ ne dépend que d'une seule variable (généralement t représente le temps) et s'écrit sous la forme :

$$F\left(t,\ y(t),\ y'(t),\ y''(t),\ \cdots\right) = 0$$

À noter que F peut être une fonction scalaire ou vectorielle (il s'agit alors d'un système d'équations différentielles scalaires). De même y peut être une fonction scalaire ou vectorielle du temps.

Remarque Les ODE se distinguent des *équations aux dérivées partielles* (EDP ou PDE en anglais) dont les solutions sont des fonctions dépendant *de plusieurs variables*, souvent le temps et l'espace, vérifiant des conditions limites sur leurs dérivées partielles. Ces équations seront vues en écoles d'ingénieurs, pour résoudre modéliser des problèmes en mécanique des milieux continus par exemple.

Les ODE les plus courantes, qui constituent le cœur des problèmes rencontrés en CPGE peuvent se mettre sous la forme :

$$y'(t) = F(t, y(t))$$

Cela signifie qu'il est possible d'isoler le terme dérivé d'ordre le plus élevé et de l'exprimer en fonction des dérivées d'ordre inférieure. Résoudre ce type d'équation revient à trouver numériquement la solution approchée d'un problème de Cauchy.

3.2 Résolution numérique du problème de Cauchy

Un problème de Cauchy consiste à trouver les fonctions Y de $[0, T] \to \mathbb{R}^N$, telles que :

$$\begin{cases} \dfrac{dY}{dt} = F(t, Y) \\ Y(t_0) = Y_0 \end{cases}$$

où $t_0 \in [0, T]$ et $Y_0 \in \mathbb{R}^N$ sont données, aussi appelées *conditions initiales* si $t_0 = 0$.

Le principe fondamental que l'on a vu aux paragraphes précédents est qu'une dérivée peut être approchée par un **taux de variation**, et inversement, que l'évolution d'une quantité peut être approchée à l'aide de la dérivée.

$$Y'(t) \approx \frac{Y(t + \Delta t) - Y(t)}{\Delta t} \quad \Longleftrightarrow \quad Y(t + \Delta t) \approx Y(t) + \Delta t \times Y'(t)$$

Ainsi, la résolution numérique du problème de Cauchy s'effectue en deux étapes :

– **Discrétisation** : On effectue une discrétisation de l'intervalle $[t_0, T]$ que l'on découpe régulièrement en t_0,..., t_n, en posant :

$$\Delta t = \frac{T - t_0}{n} \quad \text{et} \quad t_k = t_0 + k \times \Delta_t$$

– **Intégration** : En partant de $Y(t_0) = Y_0$, on calcule itérativement les solutions approchées $Y(t_k)$ du problème de Cauchy avec le schéma d'intégration numérique suivant, appelé **schéma d'Euler explicite** :

$$\forall k \in [\![1, n]\!], \quad Y_{k+1} = Y_k + \Delta t \times F(t_k, Y_k)$$

Q 11. Écrire un script Python permettant de résoudre numériquement l'équation différentielle $y'(t) = y(t)$ entre $[0, 5]$ où $Y_0 = 1$ pour différentes discrétisation $n = 10, 50, 100$ et 1000 avec le schéma d'Euler explicite.

Q 12. En utilisant la librairie `matplolib.pyplot`, comparer graphiquement la solution approchée obtenue avec le résultat théorique $y(t) = \exp(t)$ pour les différentes discrétisations précédentes.

4 Formulation matricielle pour les équations d'ordre supérieur

L'exemple du paragraphe précédent a permis d'implémenter une méthode de résolution utile pour obtenir numériquement la solution approchée d'une équation différentielle ordinaire d'ordre 1, qu'elle soit linéaire ou non.

Nous allons voir au travers de deux exemples supplémentaires comment le schéma d'Euler explicite se formule dans le cas des équations différentielles d'ordre supérieur à 1.

Remarque Ces deux exemples seront facilement transposables pour proposer un volet numérique à votre projet de TIPE, indispensable en filière PSI notamment.

4.1 Pendule simple non-linéaire

Soit un pendule de rayon R et de masse m. On recherche la loi d'évolution $\theta(t)$ en tenant compte du frottement visqueux et sans linéariser l'équation de la dynamique.

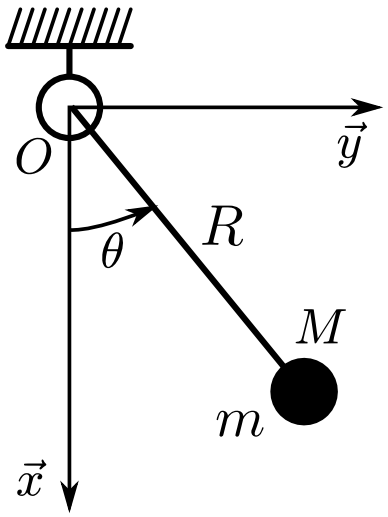

FIGURE 5 – Paramétrage du pendule.

L'équation du principe fondamental de la dynamique et les conditions initiales s'écrivent :

$$mR^2\ddot{\theta}(t) = -mgR\sin(\theta(t)) - R^2 f\dot{\theta}(t)$$
$$\text{avec} \quad \dot{\theta}(0) = \omega_0 \quad \text{et} \quad \theta(0) = \theta_0$$

Il s'agit d'une équation différentielle du second ordre, mais qui peut être réécrite d'un système de deux équations du premier ordre :

$$\begin{cases} \dfrac{d\theta}{dt} &= \dot{\theta} \\ \dfrac{d^2\theta}{dt^2} &= \dfrac{1}{mR^2}\left[-mgR\sin(\theta) - R^2 f\dot{\theta}\right] \end{cases}$$

On reconnaît alors une ODE sous la forme d'un problème de Cauchy en posant la variable Y :

$$Y = \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix} \quad \Longrightarrow \quad \frac{dY}{dt} = \begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = F(t,Y) = \begin{bmatrix} Y[1] \\ \dfrac{1}{mR^2}\left[-mgR\sin(Y[0]) - R^2 f Y[1]\right] \end{bmatrix}$$

Q 13. Compléter le script Python `euler_pendule.py` afin d'obtenir la loi horaire du pendule précédent.

Q 14. Commenter la trajectoire du point M pour $\omega_0 = 25$ rad/s et $\theta_0 = 0$ rad puis pour $\omega_0 = 0$ rad/s, $\theta_0 = \pi$.

Remarque La fonction `Euler` présente dans le script précédent est très générale et peut être utilisée pour toutes les ODE d'ordre supérieure à 1 dès lors que l'on défini proprement la fonction F avec des `array` de la librairie `numpy`.

4.2 Balistique d'une balle de tennis

On cherche à simuler la trajectoire d'une balle de tennis, connaissant la position M_0, la vitesse de frappe $\overrightarrow{V_0}$, ainsi que l'effet donné à la balle $\overrightarrow{\Omega}$ (rotation propre de la balle).

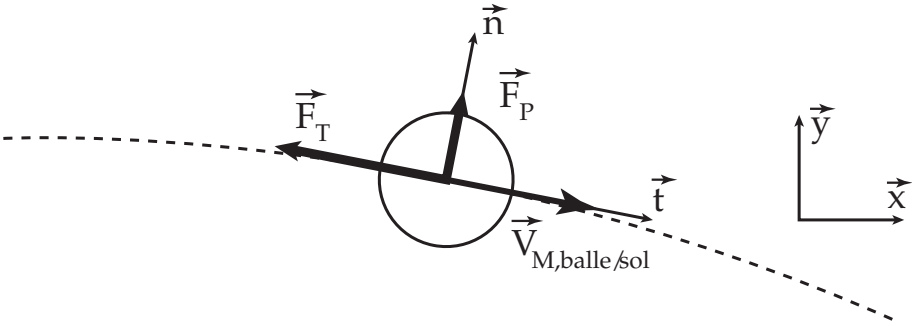

FIGURE 6 – Forces de portance et de trainée.

La balle est soumise à :

- la gravité $\overrightarrow{P} = m\vec{g}$,
- la force de trainée modélisée comme un frottement quadratique opposé à la vitesse $\vec{F}_T = -\frac{1}{2} f \|V\| \vec{V}$,
- une force de portance due à la rotation propre de la balle (effet Magnus), perpendiculaire à la vitesse $\vec{F}_P = \mu \vec{\Omega} \wedge \vec{V}$.

Le principe fondamental appliqué à la balle conduit à l'équation vectorielle :

$$m \overrightarrow{\Gamma_{M,\text{balle}/sol}} = m\vec{g} - \frac{1}{2} f \|V\| \vec{V} + \mu \vec{\Omega} \wedge \vec{V}$$

Il s'agit d'une équation vectorielle pouvant se projeter comme deux équations différentielles scalaires du second ordre. Ce système différentiel peut s'écrire comme un système de 4 équations différentielles du premier ordre, de manière similaire à l'exemple précédent.

Or, puisque le problème est posé sous forme vectorielle dans le plan, il est préférable de tirer parti d'une écriture vectorielle du programme.

La position $\overrightarrow{OM}(t)$ de la balle n'est pas vue comme deux coordonnées scalaires x et y mais comme un vecteur $\overrightarrow{OM}$. De même pour la vitesse $\vec{V}(t)$, de sorte que le vecteur $Y(t)$ des fonctions d'intégration est composé de deux vecteurs $\overrightarrow{OM}$ et $\vec{V}$:

$$\overrightarrow{OM} = \begin{bmatrix} x \\ y \end{bmatrix} \quad \vec{V} = \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} \quad Y = \begin{bmatrix} \overrightarrow{OM} \\ \vec{V} \end{bmatrix}$$

Et l'équation différentielle se met alors sous la forme d'un problème de Cauchy :

$$\frac{dY}{dt} = \begin{bmatrix} \dot{\overrightarrow{OM}} \\ \dot{\vec{V}} \end{bmatrix} = F(t,Y) = \begin{bmatrix} \vec{V} \\ \frac{1}{m}(m\vec{g} - \frac{1}{2} f \|V\| \vec{V} + \mu \vec{\Omega} \wedge \vec{V}) \end{bmatrix}$$

Q 15. Modifier la fonction F précédente pour résoudre cette nouvelle ODE. L'expression du PFD utilise une norme et un produit vectoriel qui peuvent être exprimés en Python avec les fonctions `linalg.norm()` et `cross()` de la librairie `numpy`. La fonction `np.concatenate((`$\overrightarrow{OM}$`,` $\vec{V}$`))` permet de construire Y.

Q 16. On se demande si le coup effectué par un joueur passe le filet ou non, celui-ci frappe la balle en $t_0 = 0$ en $M_0 = (-10, 0.5)$ mètres avec une vitesse initiale $\overrightarrow{V_0} = (120, 18)$ km/h et un « lift » $\overrightarrow{\Omega} = (0, 0, -100)$ rad/s. Le filet est positionné en $x = 0$ et a une hauteur $h = 0.91$ mètre. Résoudre le problème avec un schéma d'Euler explicite, tracer la trajectoire de la balle et conclure.