Matthieu Boyer
Planche A

Méthode d'Euler discrète pour l'approximation de trajectoires ballistiques

Considérez un projectile lancé à partir du sol avec une vitesse initiale v_0 et un angle de lancement θ_0.

1. Montrez que la trajectoire du projectile est régie par les équations suivantes (avec des notations usuelles) :

$$\frac{dx}{dt} = v_x = v_0 \cos(\theta_0), \quad \frac{dy}{dt} = v_y = v_0 \sin(\theta_0) - gt$$

et

$$\frac{dv_x}{dt} = 0, \quad \frac{dv_y}{dt} = -g$$

2. À quel temps le projectile aura-t-il atteint le sol ?

Vous devez implémenter une approximation de la trajectoire du projectile par la méthode d'Euler discrète. Le programme doit prendre un angle d'entrée et afficher les valeurs de la position $x(t)$ et $y(t)$ à chaque itération jusqu'à ce que $y(t)$ atteigne zéro (le projectile touche le sol). On définit le type `struct { float x, y; } vecteur;`.

1. Implémentez `void euler_step(vecteur* pos, vecteur* vit, float dt);` : Effectue une étape de la méthode d'Euler pour mettre à jour la position et la vitesse.
2. Implémentez `void afficher_trajectoire(vecteur* pos, int n);` qui affiche les positions du vecteur.
3. Implémentez un point d'entrée pour prendre l'angle et la vitesse initiale en entrée, et affichez les résultats jusqu'à ce que le projectile atteigne le sol.
4. Que se passe-t-il si on ajoute des frottements de l'air modélisés par une force $f = -\alpha v$?

Matthieu Boyer
Planche B

Bibliothèque de nombres complexes pour la transformée de Fourier discrète

La transformée de Fourier discrète (TFD) nécessite des calculs avec des nombres complexes. Avant d'implémenter la TFD, il vous est demandé de créer une petite bibliothèque pour manipuler les nombres complexes en C. On définit le type `struct { float re, im; } complexe;` Implémentez les fonctions suivantes :

1. `complexe addition(complexe a, complexe b);` : Addition de deux nombres complexes.
2. `complexe multiplication(complexe a, complexe b);` : Multiplication de deux nombres complexes.
3. `complexe exponentielle_complexe(float angle);` : Retourne l'exponentielle complexe $e^{i\theta}$ où θ est un angle en radians.
4. `float module(complexe z);` : Retourne le module du nombre complexe z.
5. `float argument(complexe z);` : Retourne l'argument du nombre complexe z.

La TFD est définie par la formule :

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-i2\pi \frac{kn}{N}}$$

où $x(n)$ est l'élément de l'array d'entrée et $X(k)$ le spectre de fréquence de sortie.

6. Implémentez `complexe fourier_discret(float* x, int N, int k);` qui calcule un seul terme de la transformée de Fourier discrète pour un indice k.
7. Implémentez `void fourier_transform(float* x, int N, complexe* X);` Implémentez la fonction qui calcule la transformée de Fourier discrète complète pour un tableau d'entrée de taille N. Remplissez un tableau `X` de taille N avec les coefficients de la TFD.
8. Implémentez `void inverse_fourier_transform(complexe* X, int N, float* x);` la fonction qui calcule l'inverse de la transformée de Fourier discrète. On vous laisse le soin de prouver la correction de votre fonction. Cette fonction doit reconstituer le signal d'origine à partir de ses coefficients de Fourier. N'oubliez pas de diviser les résultats par N.
9. Donnez un exemple d'usage en implémentant un programme principal qui lit un tableau d'entrées de type float, applique la transformée de Fourier discrète, puis reconstruit le signal d'origine avec la fonction inverse.

Bibliothèque de matrices pour les rotations en 3D

Les rotations en 3D peuvent être représentées à l'aide de matrices. Implémentez une bibliothèque pour manipuler des matrices en C, spécifiquement pour les rotations en 3D. On définit `struct { float m[3][3]; } matrice;` et `struct { float x, y, z; } vecteur;`.

1. Implémentez `matrice rotation_x(float angle);` : Retourne la matrice de rotation autour de l'axe x pour un angle θ en radians. On rappelle l'expression de la matrice.

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix} \tag{1}$$

2. Implémentez `matrice multiplication_matrices(matrice* m1, matrice* m2);` : Multiplie deux matrices de rotation.
3. Implémentez `void appliquer_rotation(matrice* m, vecteur* v);` : Applique la matrice de rotation à un vecteur donné.
4. Montrez que pour tout vecteur v $R_x(\theta)R_x(\theta') = R_x(\theta + \theta')$. Proposez une autre méthode pour représenter des matrices comme trois angles. Réimplémentez `appliquer_rotation` pour cette notation.

Utilisez cette bibliothèque pour appliquer des rotations successives à un vecteur donné (par exemple, $(1, 0, 0)$).

Bibliothèque statistique

Dans cet exercice, vous allez implémenter une bibliothèque en C pour effectuer des calculs statistiques sur des tableaux de données. Le type de données de base est un tableau de flottants. Vous devez implémenter les fonctions suivantes pour calculer des statistiques de base :

1. `float somme(float* tab, int N);` qui calcule la somme des éléments d'un tableau de N nombres réels (floats).
2. `float moyenne(float* tab, int N);` qui calcule la moyenne des éléments du tableau en utilisant la formule.
3. `float variance(float* tab, int N);` qui calcule la variance des éléments du tableau.
4. `float ecart_type(float* tab, int N);` qui calcule l'écart-type des éléments du tableau.
5. `float amplitude(float* tab, int N);` qui calcule l'amplitude du tableau, c'est-à-dire la différence entre la valeur maximale et la valeur minimale. On ne fera qu'un passage sur le tableau.
6. `float mediane(float* tab, int N);` qui calcule la médiane du tableau. On pourra au préalable implémenter une fonction qui trie le tableau.
7. `void afficher_statistiques(float* tab, int N);` : Implémentez une fonction qui affiche toutes les statistiques calculées précédemment : somme, moyenne, médiane, écart-type, variance, amplitude.
8. Si le temps : donnez un algorithme qui calcule la médiane du tableau en ne parcourant le tableau qu'une seule fois.