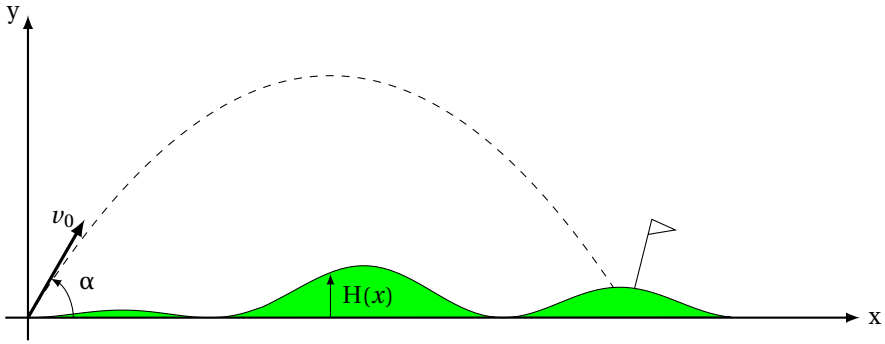
Algorithmes génétiques

1 Introduction

Dans cette séance, nous allons étudier la trajectoire d'une balle de golf, et utiliser un algorithme génétique pour estimer l'angle et la vitesse de rotation optimaux pour envoyer la balle le plus loin possible.

On se restreint à une trajectoire dans le plan, $\vec{e}_x$ étant dirigé horizontalement, $\vec{e}_y$ verticalement, vers le haut.

Le sol n'est pas horizontal, mais présente des collines. Par ailleurs, du vent souffle au-dessus de celui-ci, et ce vent est affecté par la forme du terrain.



Pour obtenir les informations sur le terrain et le vent, on dispose de différentes fonctions :

- `terrain.h(x)` retourne l'altitude du sol à l'abscisse `x` ;
- `terrain.vent(x, y)` retourne un vecteur (v_x, v_y) correspodant à la force du vent au point de coordonnées (x, y).

Pour des raisons de simplicité, il est ici acceptable de considérer que le nom `terrain` est un nom global, que l'on pourra utiliser dans toutes les fonctions sans avoir à passer `terrain` en argument.

2 Calcul de la trajectoire

Une balle de golf a une masse $m = 45$ g, et un moment d'inertie autour d'un axe quelconque passant par son centre $J = 8{,}7 \times 10^{-6}$ kg.m^2.

La balle, frappée, débute sa trajectoire avec une vitesse $v_0 = 70$ m/s (une vitesse obtenue avec un excellent joueur de golf et un bois, c'est-à-dire un club destiné aux frappes longues) dans le plan $(\vec{e}_x, \vec{e}_y)$, depuis l'origine du repère, dans une direction faisant un angle α avec l'horisontale.

Par ailleurs, la frappe communique à la balle une vitesse de rotation propre $\vec{\omega}$. On supposera que cette rotation est uniquement autour d'un axe $\vec{e}_z$, afin que le mouvement reste plan, et on notera $\vec{\omega} = \omega \vec{e}_z$.

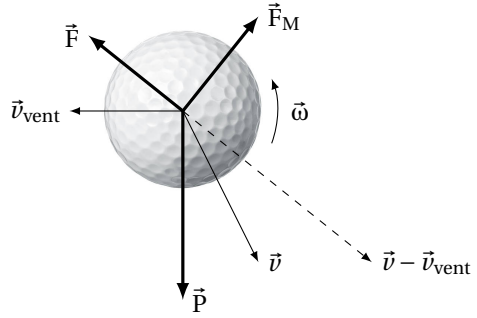
La balle est ensuite soumise :

- à son poids $\vec{P} = m\vec{g}$;
- à une force de frottement [1] $\vec{F} = -c_f \, |\vec{v} - \vec{v}_{\text{vent}}| \, (\vec{v} - \vec{v}_{\text{vent}})$ où $\vec{v}$ est la vitesse de la balle, $\vec{v}_{\text{vent}}$ la vitesse du vent au voisinage de la balle, et c_f un coefficient de frottements (on prendra [2] $c_f = 2{,}7 \times 10^{-4}$ SI) ;
- à une force dûe à la rotation de la balle, dite force de Magnus, $\vec{F}_M = c_a \vec{\omega} \otimes (\vec{v} - \vec{v}_{\text{vent}})$, où c_a est un coefficient constant (on prendra $c_a = 4{,}0 \times 10^{-5}$ SI).

Cette dernière force est très importante dans beaucoup de sports. C'est cette force qui est à l'origine des « effets » observés sur la balle au tennis, au ping-pong, au baseball ou sur la trajectoire d'un ballon de football. Au golf, elle permet à la balle d'avoir de la portance (une force dirigée vers le haut) durant la trajectoire, ce qui permet d'aller plus loin.

Toutefois, une rotation propre trop importante de la balle n'est pas souhaitable, la trajectoire montant alors trop haut, et la portée étant réduite. On pourrait même observer une trajectoire en forme de boucle (ce qui arrive parfois au baseball). Pour une frappe avec un bois, la vitesse de rotation idéale se situe aux environs de 2500 rotations par minute.

La balle est par ailleurs soumise à un moment résistant $\overrightarrow{\mathcal{M}} = -c_\omega \omega \vec{e}_z$ dus aux frottements, avec $c_\omega = 5{,}0 \times 10^{-7}$ SI.



1. Les balles de golf sont alvéolées pour diminuer l'effet de la force de frottement. Initialement, les balles étaient lisses, mais on a rapidement remarqué que des balles usées avaient une portée nettement supérieure. Le réglement impose des alvéoles régulièrement réparties, car il est possible de fabriquer des balles anisotropes qui corrigent d'elles même leur rotation.
2. On a $c_f = \frac{1}{2}\rho_{\text{air}} \pi R^2 C_x$ avec $C_x \simeq 0{,}3$ pour une balle alvéolée.

L'état de la balle à un instant t est défini par la donnée d'un vecteur à cinq composantes, $\big(x(t), y(t), v_x(t), v_y(t), \omega(t)\big)$.

1. Déterminer l'expression de la dérivée du vecteur précédent, en fonction du vecteur proprement dit, des données du problème et de la vitesse du vent $v_{\text{vent}}\big(x(t), y(t)\big)$ au niveau de la balle.

2. En déduire une fonction `der(etat, t)` prenant en argument une liste de cinq flottants correspondant au vecteur état $\big(x(t), y(t), v_x(t), v_y(t), \omega(t)\big)$ et l'instant[3] t et retournant la dérivée du vecteur état.

3. En déduire une fonction `trajectoire(v0, alpha, w0)` prenant en paramètre la vitesse initiale v_0, l'angle α et la vitesse de rotation initiale ω_0 et utilisant la fonction `odeint` de `scipy` afin de retourner un tableau `numpy` à 201 lignes et 5 colonnes correspondant aux valeurs prises par le vecteur état pour des instants t_i entre $t = 0$ et $t = 20$ s séparés par un intervalle de temps $\mathrm{d}t = 0{,}1$ s.

On rappelle que la fonction `odeint` du module `scipy.integrate` attend au moins trois paramètres, dans l'ordre suivant :

- une fonction, prenant en argument un vecteur décrivant l'état d'un système (ici le quintuplet $\big(x(t), y(t), v_x(t), v_y(t), \omega(t)\big)$) et un instant `t` et retourne la dérivée du vecteur état ;
- un vecteur décrivant les conditions initiales ;
- la liste des instants t_i pour lequel il faudra calculer l'état du système, le premier des t_i correspondant aux conditions initiales précédentes.

Le résultat alors fourni est un tableau (précisément un `numpy.ndarray`) dont le nombre de lignes correspond au nombre d'instants t_i et le nombre de colonnes à la taille du vecteur état, chacune des lignes décrivant l'état du système pour le t_i correspondant.

La simulation montre que l'angle optimal ($\pi/4$) même en l'absence de frottements et de rotation n'est plus le meilleur angle possible dans le cas présent, et que la rotation doit être présente, mais modérée. La forme de la trajectoire idéale, quasiment triangulaire, est très caractéristique de la trajectoire d'un mobile soumis à des frottements et à un effet Magnus.

3 Détection du point d'impact

On s'attache à présent à déterminer la position du point d'impact avec le sol, à partir du tableau de taille 201×5 obtenu avec la fonction précédente.

Dans un premier temps, on cherche le plus petit entier `i` tel que la balle se situe au-dessus du niveau du sol (ou au niveau du sol) à l'instant t_i, et strictement au-dessous du niveau du sol à l'instant t_{i+1}.

4. Comment doit-on écrire cette condition en fonction de $x(t_i)$, $y(t_i)$, $x(t_{i+1})$, $y(t_{i+1})$ et de la fonction $y = \mathrm{H}(x)$ décrivant le sol ?

5. Proposer une fonction `etape_sol(tab)` prenant en argument[4] un tableau de taille $n \times 5$ et retournant le `i` recherché. On utilisera une recherche linéaire : en effet, si une recherche dichotomique pourrait sembler plus intéressante, il est possible que la balle « passe temporairement » sous le sol (par exemple à cause d'une colline), et on veut impérativement le *premier* contact avec le sol. La fonction peut retourner un résultat quelconque si la trajectoire est intégralement au-dessus du sol, ce qui ne devrait pas arriver en pratique.

On suppose qu'entre les instants t_i et t_{i+1}, la trajectoire de la balle est approximativement rectiligne.

6. Proposer une fonction `Intersect(x1, y1, x2, y2, eps)` déterminant et retournant l'intersection (supposée unique) (x, y) entre un segment $\mathrm{M_1M_2}$ où $\mathrm{M_1}$ est le point de coordonnées (x_1, y_1) et $\mathrm{M_2}$ le point de coordonnées (x_2, y_2), et la courbe $y = \mathrm{H}(x)$ (fournie par `terrain`) avec une erreur d'au plus `eps` sur la valeur de x. On travaillera par dichotomie, les points $\mathrm{M_1}$ et $\mathrm{M_2}$ étant supposés de part et d'autre de la courbe $y = \mathrm{H}(x)$.

7. En déduire une fonction `portee(tab, eps)` qui retourne la portée (abcisse du contact avec le sol) d'une trajectoire décrite dans le tableau `tab` avec une précision `eps`.

8. Ecrire une fonction `traj_air(v0, alpha, w0, eps)` qui utilise la fonction `Trajectoire` pour construire un tableau de taille $n \times 5$ représentant la trajectoire, détermine le numéro de ligne `i` du dernier point de la trajectoire au-dessus du sol, les coordonnées exactes du contact avec le sol (à une précision `eps` près), et construit un tableau de taille $i + 2 \times 5$ dont les $i + 1$ premières lignes sont celles du tableau retourné par `Trajectoire` et la dernière ligne correspond au point de contact avec le sol (on pourra prendre 0 pour les valeurs de v_x, v_y et ω pour cette dernière ligne).

4 Optimisation

À présent qu'il nous est possible de déterminer la portée $x_{\max}(\alpha, \omega_0)$ d'un swing connaissant les conditions initiales, nous allons nous efforcer de trouver les valeurs de α et ω_0 qui permettront d'obtenir la plus grande portée possible.

Il serait possible d'utiliser un outil de minimisation classique pour ce faire, mais nous allons profiter de l'occasion pour mettre en œuvre une méthode dite « génétique ».

Ces méthodes gèrent une population d'« individus », correspondant à des paramètres possibles (dans notre cas, l'angle α et la vitesse de rotation initiale ω_0). À chaque étape de l'algorithme, les individus les moins « performants » sont éliminés, et d'autres individus sont créés, soit par mutation d'un individu existant, soit par reproduction d'un couple d'individus.

3. Qui ne servira pas ici, mais qui est requis pour l'utilisation de `odeint`.

4. On rappelle que `terrain` est une variable globale, dont on peut se servir dans la fonction

Nous allons représenter les individus par le triplet $(\alpha, \omega_0, x_{\max})$, où $x_{\max}$ représente la portée obtenue avec les paramètres initiaux α et ω_0, et fait office de mesure de la « performance » d'un individu, puisque l'on souhaite envoyer la balle le plus loin possible.

La population sera une liste d'individus, et on s'assurera qu'à tout instant les individus sont rangés dans la liste par ordre décroissant de performance.

Le module `random` contient une fonction `random()` ne prenant pas d'argument et retournant un réel choisi pseudo-aléatoirement entre 0 et 1.

9. Proposer une fonction `cree_individu()` ne prenant aucun argument et retournant un triplet $(\alpha, \omega_0, x_{\max})$, où α a été choisi aléatoirement entre 0 et $\pi/2$, et ω_0 entre -200 rad/s et 700 rad/s.

10. Écrire une fonction `insere_individu(population, individu)` prenant en argument une liste d'individus rangés par ordre décroissant de performance et un triplet représentant un individu, et insérant ce dernier dans la liste, de façon à préserver le caractère trié de la liste.

11. En déduire une fonction `initialise_population(n)` créant une population de `n` individus générés aléatoirement.

Afin de faire évoluer cette population, il nous faut créer des fonctions permettant de créer de nouveaux individus. On commence par la mutation. Pour un individu $(\alpha, \omega_0, x_{\max})$, le résultat d'une mutation est un individu $\left(\alpha + \delta\alpha, \omega_0 + \delta\omega_0, x'_{\max}\right)$.

$\delta\alpha$ est choisi selon une loin normale de moyenne $\mu = 0$ et d'écart-type $\pi/12$. Pour ce faire, on utilisera la fonction `normal` du module `random`. De même, $\delta\omega_0$ est choisi selon une loin normale de moyenne $\mu = 0$ et d'écart-type 50 rad/s.

12. Proposer une fonction `mute(individu)` prenant un individu et retournant une mutation de cet individu.

Une autre façon de créer de nouveaux individus est la reproduction. Elle consiste à prendre deux individus $(\alpha, \omega_0, x_{\max})$ et $\left(\alpha', \omega'_0, x'_{\max}\right)$, et construit un nouvel individu $\left(\alpha'', \omega''_0, x''_{\max}\right)$ où α'' et ω''_0 sont déterminés par $\alpha'' = k\alpha + (1-k)\alpha'$ et $\omega''_0 = k\omega_0 + (1-k)\omega'_0$ où k suit une loi uniforme sur $[0, 1]$.

13. Proposer une fonction `reproduit(individu1, individu2)` construisant un individu résultat de la reproduction des deux individus fournis en paramètre.

Il s'agit à présent de choisir l'individu qui sera muté, parmi la population, ou le couple d'individus qui vont se reproduire. On souhaite privilégier les meilleurs parents, tout en laissant une chance à tous les individus d'être sélectionner.

Pour sélectionner un individu, on procédera de la façon suivante :

- on assigne au nom `candidat` la valeur **None** ;
- on considère chaque individu de la population un par un, par performance *croissante* ; avec une probabilité p, l'individu en question devient le nouveau `candidat`
- une fois tous les candidats considérés, si `candidat` n'est pas **None**, on le sélectionne, et sinon on recommence l'étape précédente.

14. Justifier que la probabilité de choisir le i^{e} individu de la population (i étant sa place dans la liste) à l'issue d'une passe dans la liste est $p(1-p)^i$, et que la probabilité est d'autant plus grande que i est petit.

15. Quelle est la probabilité qu'à l'issue de la boucle sur les individus, aucun n'ait été sélectionné ?

16. En déduire la probabilité de choisir le i^{e} individu de la population à l'issue de l'algorithme proposé.

17. Proposer une fonction `choisit_1(population, p)` retournant un individu choisi aléatoirement dans la liste, où la probabilité p est fournie en paramètre.

18. Á partir de la fonction `choisit_1` précédente, construire une fonction `choisit_2(population, p)` retournant deux individus *distincts* dans la population.

Il ne nous reste à présent qu'à assembler les différents éléments. L'algorithme d'optimisation va donc :

- créer une population de n individus choisis aléatoirement
- puis, `nb_iter` fois,
  - on choisit un individu aléatoirement que l'on mute, et on ajoute le nouvel individu à la population ;
  - on choisit un couple d'individus distincts aléatoirement que l'on reproduit, et on ajoute le nouvel individu à la population ;
  - on supprime les deux individus de la population les moins efficaces.

19. Écrire une fonction `evolue(n, p, nb_iter)` effectuant le travail précédent, et retournant de listes :

- la liste des $x_{\max}$ de l'individu le plus performant à l'issue de chaque itération de l'algorithme
- la liste contenant la population à l'issue de l'algorithme.