Challenge Monte-Carlo 2:

Angry Birds 2.0

MAP556 - Novembre 2021

Version courante: 6 novembre 2021

En résumé. Tout le monde connait le jeu de balistique Angry Birds. Il s'agit de propulser des oiseaux vers des cibles représentées par des bâtiments s'effondrant sur leurs ennemis cochons. Ce challenge va reprendre ce principe dans une version d'oiseaux 2.0 (appelés `AG 2.0`), qui pourront ajuster leur vitesse le long de leur trajectoire. Mais attention, leur tâche sera rendue difficile par des conditions météorologiques exécrables, avec des vents tourbillonnants imprévisibles qui viendront perturber leur trajectoire.

L'objectif du challenge sera de fournir une fonction de contrôle qui permette au mieux d'atteindre la cible en présence de vent aléatoire. La construction numérique de cette fonction pourra être réalisée via des simulations Monte Carlo du vent et des techniques d'optimisation et d'apprentissage sur ces trajectoires.

Les fonctions les plus performantes seront récompensées par les meilleures notes.

Mise en équation. Nous partons du principe que l'`AG 2.0` obéit au principe fondamental de la dynamique (2ème loi de Newton). Quelques notations :

i) m : masse de l'Angry Bird (en $\mathtt{kg}$). Dans la suite, on choisit des oiseaux calibrés à $m = 1\mathtt{kg}$.

ii) X_t : sa position en mètre ($\mathtt{m}$) dans le plan $\mathbb{R}^2$ avec $X_t = \begin{pmatrix} X_{1,t} \\ X_{2,t} \end{pmatrix}$ et $X_0 = 0$.

iii) $\mathbf{g}$: les `AG 2.0` évoluent sur la *Red Planet* dont la force de gravité est de coordonnées $\mathbf{g} = \begin{pmatrix} 0 \\ -4 \end{pmatrix} \mathtt{ms}^{-2}$.

iv) λ : coefficient de résistance de l'air (en $\mathtt{kg\ s}^{-1}$) (en prenant pour simplifier une force de résistance linéaire en la vitesse). Sur la *Red Planet*, $\lambda = 0.01$.

v) V_t : le vent (aléatoire, dont le modèle sera précisé plus tard en (0.2)), c'est un vecteur à 2 composantes exprimés en $\mathtt{ms}^{-1}$.

vi) u_t : le contrôle de vitesse de l'`AG 2.0`.

Partant de $m\frac{\mathrm{d}^2 X_t}{\mathrm{d}t^2} = m\mathbf{g} - \lambda\frac{\mathrm{d}X_t}{\mathrm{d}t} + \frac{\mathrm{d}V_t}{\mathrm{d}t}$, en intégrant en temps et en prenant $m = 1\mathtt{kg}$, on obtient :

$$\dot{X}_t := \frac{\mathrm{d}X_t}{\mathrm{d}t} = \dot{X}_0 + \mathbf{g}t - \lambda X_t + V_t$$

en absence de contrôle de la vitesse. Avec la composante de contrôle u_t, cela devient

$$\boxed{\begin{cases} \dot{X}_t &:= \frac{\mathrm{d}X_t}{\mathrm{d}t} = \dot{X}_0 + \mathbf{g}t - \lambda X_t + V_t + u_t, \\ X_t &= \dot{X}_0 t + \mathbf{g}\frac{t^2}{2} - \lambda \int_0^t X_s \mathrm{d}s + \int_0^t V_s \mathrm{d}s + \int_0^t u_s \mathrm{d}s. \end{cases}}$$

Nous utiliserons une version discrétisée aux temps $t_i = i\Delta_T$ avec $\Delta_T = 0.1\mathtt{s}$:

$$\boxed{X_{t_{i+1}} = X_{t_i} + \dot{X}_0 \Delta_T + \mathbf{g}\left(\frac{t_{i+1}^2}{2} - \frac{t_i^2}{2}\right) - \lambda X_{t_i}\Delta_T + V_{t_i}\Delta_T + u_{t_i}\Delta_T.} \tag{0.1}$$

Le vent aléatoire V obéit à une équation différentielle stochastique

$$\boxed{V_t = \begin{pmatrix} V_{1,t} \\ V_{2,t} \end{pmatrix} := -\int_0^t \begin{pmatrix} 0.9 \times V_{1,s} + 0.2 \times V_{2,s} \\ V_{1,s} + 1.1 \times V_{2,s} \end{pmatrix} \mathrm{d}s + 5 \begin{pmatrix} W_{1,t} \\ W_{2,t} \end{pmatrix}} \tag{0.2}$$

où W_1, W_2 sont deux mouvements browniens standards indépendants.

Paramètres de condition initiale. Il s'agit d'atteindre l'objectif de coordonnées $X^{\mathrm{obj}} = (D, 0)$ avec $D = 200\mathtt{m}$. L'`AG 2.0` est programmé pour se désagréger sur sa cible au bout de $T = 10\mathtt{s}$ précisément. On choisit de relier vitesse initiale $\dot{X}_0$ et D pour que sans vent, sans contrôle, et sans frottement, on ait $X_{10} = X^{\mathrm{obj}}$. Cela conduit à

$$\begin{cases} \dot{X}_{1,0} = \frac{D}{T} = 20\mathtt{ms}^{-1}, \\ \dot{X}_{2,0} = \frac{T}{2}|\mathbf{g}| = 20\mathtt{ms}^{-1}. \end{cases}$$

Dans tout le challenge, ces paramètres conserveront les mêmes valeurs numériques.

Forme générale de la fonctionnelle de coût. La performance du contrôle tiendra compte de 2 facteurs :

- la proximité de X_{10} avec X^{obj}, ce sera calculé à travers une fonction de perte L (dont la forme exacte sera donnée plus tard),
- la quantité de contrôle utilisée pour parvenir à ses fins.

La fonction performance sur un scénario de vent s'écrit alors

$$j(u) := \int_0^{10} |u_t|^2 \,\mathrm{d}t + L\left(X_{10}\right),$$

alors qu'en moyenne sur les trajectoires, cela s'écrit

$$J(u) := \mathbb{E}\left[j(u)\right].$$

Objectif étudiant. Trouver une représentation numérique du contrôle u_t comme une fonction du temps courant t et de la position seulement, c'est-à-dire sous la forme $u_t = u(t, X_t)$ pour une certaine fonction u qui pourra être calculée dans un espace d'approximation linéaire (espace engendré linéairement par des fonctions de base) ou non-linéairement (réseaux de neurones).

En pratique, la commande ne sera appliquée qu'une fois par seconde, i.e. au temps $t = 0, 1, \ldots, 9$, les équations intégrales de X seront discrétisées avec un pas de $\Delta_T = 0.1\mathtt{s}$ (comme en (0.1)).

L'apprentissage de $u(i, .), i = 0, \ldots, 9$ pourra être faite sur des simulations de vents V.

Evaluation en fonction de la meilleure performance sur M tirages de vent

$$J(u) \approx \frac{1}{M} \sum_{m=1}^{M} \left(j(u)\right)^{(m)}.$$

Détails

Contrôle. Pour chaque groupe d'étudiant, il s'agit de proposer un contrôle de la forme

$$u_t = u(t, X_t) \in \mathbb{R}^2$$

qui sera appliqué une fois par seconde, i.e. au temps $t = 0, 1, \ldots, 9$. Attention, la fonction u ne doit pas dépendre du vent. S'inspirant du cours, on pourra par exemple prendre u dans un espace d'approximation linéaire, c'est-à-dire de la forme

$$u(t,x) = \sum_{k=1}^{K} \alpha_k \varphi_k(t,x)$$

avec des fonctions *features* $\varphi_k : (t,x) \in [0,T] \times \mathbb{R}^2 \mapsto \begin{pmatrix} \varphi_{1,k}(t,x) \\ \varphi_{2,k}(t,x) \end{pmatrix} \in \mathbb{R}^2$ et pour des coefficients réels $(\alpha_k : 1 \leq k \leq K)$. Ou bien, prendre u dans un espace non-linéaire (réseaux de neurones), c'est-à-dire de la forme

$$u(t,x) = \begin{pmatrix} \texttt{NN}_1(\theta, t, x) \\ \texttt{NN}_2(\theta, t, x) \end{pmatrix}.$$

Le nombre de paramètres de modélisation (le nombre K de features, ou le nombre de paramètres θ pour les réseaux de neurones) n'est pas spécifié et est laissé au choix de chaque groupe d'étudiant.

Pour cadrer la complexité de u (et éviter d'utiliser des milliards de paramètres), nous imposons une contrainte sur la mémoire requise pour stocker cette fonction contrôle : voir les détails dans le paragraphe **Méthodologie et collecte des challenges.**

Fonction de cout. La fonction performance sur un scénario de vent est définie par

$$j(u) := \sum_{i=0}^{9} |u_i|^2 + L\left(X_{10}\right),$$

alors qu'en moyenne sur les trajectoires de vent, cela s'écrit

$$J(u) := \mathbb{E}\left[j(u)\right]. \tag{0.3}$$

La fonction de perte terminale est [1]

$$L(x) = \left(\frac{x_1 - D - x_2}{\sqrt{2}} + \frac{(x_1 - D - x_2)_+}{\sqrt{2}}\right)^2 + \left(\frac{x_1 - D + x_2}{\sqrt{2}}\right)^2 + (x_1 + x_2 - (D-15))_-^2. \tag{0.4}$$

1. y_+ et y_- désignent respectivement la partie positive et négative de y

Les deux premiers carrés représentent une mesure de distance quadratique, aggravée dans la direction $(1,-1)$, pour représenter qu'un dépassement de la cible dans cette direction n'est pas souhaitable (imaginer un obstacle par exemple). Le troisième terme pénalise les trajectoires qui seront trop loin du point cible $(D,0)$ dans la direction (1,1) (imaginer un ravin dans la zone avant la cible et toute arrivée dans le ravin est à éviter).

Algorithm 1: Pseudo-Code MCC2

Input: Une trajectoire de V
Output: Le cout trajectoriel $j(u)$

for $i = 0, \dots, 9$ **do**
  $u \leftarrow u(i, X_i)$ (à coder par les éudiants)
  for $t_j = i : i+1$, avec un pas $\Delta_T = 0.1\mathtt{s}$ **do**
    calculer $X_{1,t_{j+1}}, X_{2,t_{j+1}}$ à l'aide de $X_{1,t_j}, X_{2,t_j}, V_{1,t_j}, V_{2,t_j}$
  $j(u) \leftarrow j(u) + |u|^2$
$j(u) \leftarrow j(u) + L(X_{10})$
return $j(u)$

Code de calcul. Un code `controle_etudiant.py` est fourni aux étudiant.e.s :

- il lit un scénario de vent (fichier .csv), un exemple `ventETU.csv` est donné à titre d'illustration ;
- la fonction `dynamique_position_sur_un_pas_de_temps(...)` implémente l'équation (0.1) entre t_i et $t_i + \Delta_T$ (avec un pas de temps $\Delta_T = 0.1\mathtt{s}$). Dans cette équation, le contrôle est mis à jour à chaque seconde (et non à chaque pas de temps) : $u_{t_i} = u(j, X_j)$ pour $j \leq t_i < j+1$ pour $j = 0, 1, \dots, 9\mathtt{s}$;
- la fonction `perte_terminale(...)` qui code $L(.)$ donnée en (0.4) ;
- une boucle sur les temps $t = 0.0, 0.1, 0.2, \dots, 9.9$ qui implémente la discrétisation (0.1) soumise au vent et contrôlée par u, puis finalement calcule la performance trajectorielle $j(u)$ et l'écrit dans un fichier.

Ce code peut bien sûr être réutilisé pour apprendre la fonction u.

Ce même code sera utilisé (par l'équipe pédagogique) pour évaluer la performance de la fonction u proposée par chaque groupe d'étudiant, en calculant $J(u)$ (défini en (0.3)) par une moyenne sur $M \gg 1$ scénarii de vents (choisis au hasard par l'équipe pédagogique selon le modèle (0.2)) :

$$J(u) \approx J^M(u) := \frac{1}{M} \sum_{m=1}^{M} \left(j(u)\right)^{(m)}.$$

Plus petite sera $J^M(u)$, meilleure sera la note à ce challenge.

Méthodologie et collecte des challenges. Le challenge est par groupe d'un ou deux étudiants (nous recommandons le travail en binôme). Pour apprendre la fonction u, chaque groupe d'étudiant

- pourra utiliser autant de scenarii de vents qu'il le souhaite (qu'il devra lui-même simuler) ;

— devra paramétrer la fonction u avec la méthode de son choix, et optimiser ses paramètres pour minimiser $J(u)$.

Chaque groupe d'étudiant

— devra transmettre sa fonction u obtenue dans un fichier appelé `mon_controle.py`. Attention l'évaluation de la fonction u au point $(t, x) =$ (`seconde`, `position`) doit impérativement être la sortie de la fonction `main(seconde, position)` qui est appelée dans `controle_etudiant.py`.

Il est possible que les paramètres de la fonction u soient mis dans un autre fichier. L'ensemble de ces fichiers ne devra pas dépasser 10Ko dans moodle et devra être compressé dans un `.zip`. Pour information, 600 paramètres (correspondant à un espace linéaire de dimension 600, ou bien environ à un réseau de neurones avec 100 neurones sur une couche cachée) constitue avec un fichier `mon_controle.py` de 100 lignes un volume d'environ 4Ko (au format `.zip`).

— devra déposer sur moodle un document pdf de 3 pages maximum expliquant sa démarche de résolution.

Le date limite de rendu est le mardi 30 novembre 20h. Le dépot est ouvert dès maintenant pour que chaque groupe d'étudiant puisse tester la contrainte de 10Ko sur sa solution zippée.