DS Python

Sergio Peignier

Cet examen aurra pour but de vous évaluer :-(et de vous présenter plusieurs algorithmes utiles et interessants :-) . Les algorithmes ont été simplifiés pour pouvoir être codés plus rapidement tout en conservant leur principe de base. Dans le premier exo on va s'amuser à refroidir des métaux, dans le deuxiéme on va jouer avec un essain de "boids" et on finira par faire évoluer des bestioles. Lisez le sujet en entier et en particulier lisez chaque exo en entier avant de commencer à coder!

1 Recuit simulé

Le recuit simulé est un algorithme d'optimization inspiré de la métalurgie: En controlant et en ralentissant le refroidissement de certains métaux on peut atteinde une configuration plus solide et stable (Énergie plus basse). On peut utiliser l'analogie avec ce type de phénomènes pour optimiser des fonctions (analogie entre la fonction à optimiser et l'énergie qui est minimizée en métalurgie). Les différentes équations proviennent de modéles de physique statistique. Dans notre cas on cherchera à utiliser le recuit simlé pour minimiser une fonction $F : \mathbb{R} \to \mathbb{R}$ (e.g. si $F(s) = s^2$ alors l'algorithme devrait converger vers $s_{min} = 0$ avec une énérgie minimale $F(s_{min}) = 0$). L'algorithme est très simple:

Variables et fonctions

- Soit $s \in \mathbb{R}$ une variable associée à la configuration du métal et $s_0 \in \mathbb{R}$ la configuration initiale du métal (ici $s_0 = \mathcal{N}(10, 0)$ valeur tirée selon loie gaussiéne centrée en 10 et d'écart type 1).
- Soit $s_n \in \mathbb{R}$ une variable associé la configuration suivante que le métal peut prendre.
- Soit $F : \mathbb{R} \to \mathbb{R}$ la fonction à optimiser qui est analogue à la fonction énergie qui associe à une configuration du métal son état énergétique (ici on prendra $F(s) = s^2$ pour simplifier)
- Soit $N : \mathbb{R} \to \mathbb{R}$ une fonction qui associe à une configuration s une configuration voisine. Dans notre cas cette fonction va juste renvoyer la valeur $N(s) = s + \mathcal{N}(0, 1)$ ou $\mathcal{N}$ est une variable aléatoire tirée selon une loie centrée réduite.
- Soit $e = F(s)$ l'énergie associée à la configuration s et $e_n = F(s_n)$ l'énergie associée à la configuration s_n
- Soit T la température du systéme. T_0 correspond à la température initiale et on prend $T_0 = 10$.
- Soit γ un paramétre du systéme (ici on prend $\gamma = 0.9$).
- Soit p la probabilité que la nouvelle configuration remplace l'ancièNne (i.e. $s \leftarrow s_n$). $p = 1$ si $e_n < e$ (état plus stable atteint) et si $e_n > e$ alors $p = e^{-\frac{e_n - e}{T}}$

Algorithme

- Initialiser $s \leftarrow s_0$, $e \leftarrow F(s)$, $T \leftarrow T_0$
- Pendant t itérations répéter les opérations suivantes:
  - Générer une nouvelle configuration : $s_n = N(s)$, $e_n = F(s_n)$
  - Calculer p et mettre à jour le systéme.
  - Faire: $T \leftarrow T * \gamma$

1.1 Questions

- Codez une classe Recuit, dont le constructeur prendra en paramétres: Une instance de la fonction à utiliser pour initialiser s_0 ainsi que les paramétres associés à cette fonction (dans un tuple), une instance de la fonciton à optimiser ainsi que ses éventuels paramétres (dans un tuple), la température initiale et la valeur du paramétre γ.
- L'opérateur étoile de python (*) pourra vous être utile: Imaginons une fonction *coucou* qui prend en parametres deux variables a et b : normalement pour appeler la fonction il faut faire $coucou(a, b)$. Que se passe t'il si j'ai un tuple c qui contient a et b (i.e. $c = (a, b)$) et j'écris: $coucou(*c)$?

- À quoi sert le paramétre γ? (pas besoin de coder quoi que ce soit ... juste regarder l'équation)
- Faire un fichier contenant les colonnes suivantes : *iteration s e T* pour chaque itération de l'algo.
- Depuis Python visualiser s en fonction des *iterations* (Gnuplot, matplotlib ...)
- Vous pouvez coder le recuit sans respecter les contraintes des points 1 et 2 mais dans ce cas là vous n'aurez pas tous les points ;-)

2 Optimisation par essaims particulaires

L'optimisation par essaims particulaires est un algorithme qui s'inspire du comportement collectif de certains animaux (oiseaux ...). Dans notre cas on cherchera à utiliser cet algorithme pour maximiser une fonction $F : \mathbb{R} \to \mathbb{R}$. Pour ce faire on lance un groupe de *boids* dans $\mathbb{R}$, chaque *boid* i aurra une position x_i dans $\mathbb{R}$ et on va faire en sorte que les *boids* se déplacent de telle sorte à ce qu'ils trouvent le point s_{max} ayant la valeur $F(x_{max})$ la plus élevée (une position ou il y a plein de nourriture par exemple). Voici l'algorithme:

Variables et fonctions

- Soit $x_{i,0} \in \mathbb{R}$ une variable associée à la position initiale du *boid* i (ici $x_{i,0} = \mathcal{U}(-100, 100)$ valeur tirée selon une loie uniforme entre -100 et 100.
- Soit $x_i \in \mathbb{R}$ une variable associée à la position du *boid*.
- Soit $F : \mathbb{R} \to \mathbb{R}$ la fonction à optimiser qui est analogue à une mesure de qualité associé aux positions du *boid* (e.g., présence de nouriture) on prendra ici i $F(x) = -x^2$
- Soit $p_i \in \mathbb{R}$ une variable associée à la meilleure position (vis-à-vis de F) que le *boid* i a rencontré au cours de sa vie, sa mesure de qualité est $ep_i = F(p_i)$.
- Soit $v_{i,0} \in \mathbb{R}$ une variable associée à la vitesse initiale du *boid* i (ici $v_{i,0} = \mathcal{N}(0, 1)$ valeur tirée selon une loie gaussiéne centrée réduite).
- Soit v_i la vitesse du boid i
- Soit $e_i = F(x_i)$ la qualitée associée à la position x_i du *boid* i,
- Soit g la meilleur position rencontrée par tous les *boids* au cours de leurs vies, sa mesure de qualité est $eg = F(g)$.
- Soient ϕ_p, ϕ_g et ω trois paramétres de l'algorithme (on prendra $\phi_p = 0.3$, $\phi_g = 0.5$ et $\omega = 0.1$)

Algorithme

- Pour chaque *boid* i de la population : initialiser $x_{i,0}$, $v_{i,0}$, $p_i \leftarrow x_{i,0}$, $g = \text{argmax}_{p_i}(E(p_i))$
- Pendant t itérations répéter: les opérations suivantes:
  - Pour chaque *boid* i faire:
    - générer deux nombres aléatoires r_g et r_p uniformement entre 0 et 1
    - mettre à jour la vitesse : $v_i \leftarrow \omega v_i + \phi_p r_p(p_i - x_i) + \phi_g r_g(g - x_i)$
    - mettre à jour la position du boid : $x_i \leftarrow x_i + v_i$
    - si $ep_i < e_i$: $p_i \leftarrow x_i$
    - si $g < e_i$: $g \leftarrow x_i$

2.1 Questions

- Codez l'algorithme, plusieurs choix d'architecture sont valables (une seule classe, une classe boids et une classe essaim ... à vous de voir). Expliquez très rapidement le choix (je n'attends pas une architecture en particulier, je veux juste que vous puissiez expliquer le choix rapidement).
- À quoi servent les paramétres ϕ_p, ϕ_g et ω ? (pas besoin de coder quoi que ce soit ... juste regarder l'équation)
- Faire un fichier contenant les colonnes suivantes : *iteration g eg* pour chaque itération de l'algo.
- Depuis Python visualiser eg en fonction des *iterations* (Gnuplot, matplotlib ...)
- Comment est-ce qu'on modifierait l'algo pour optimiser des fonctions de $\mathbb{R}^n \to \mathbb{R}$?

3 Algorithme évolutionniste

Les algorithme évolutionnistes s'inspirent de la théorie de l'évolution pour résoudre divers problèmes. Dans ce contexte un individu correspond à une solution à un problème doné, une population d'individus correspond à un ensemble de solutions possibles à un même problème. Ces individus sont soumis à un procéssus de sélection afin de générer la génération suivante, puis la nouvelle génération est soumise à un procéssus de mutation. À travers la création de nouveaux individus (mutation) et la séléction des meilleures solutions, il est possible de générer des solutions de plus en plus adaptées à un probléme donné. Dans notre cas nous allons utiliser un algorithme évolutionniste pour optimiser une fonction $F : \mathbb{R} \to \mathbb{R}$. Dans notre cas un individu et son génôme sont confondus et correspond à une position dans l'axe des abscisses (une valeur x de $\mathbb{R}$) et sa fitness correspond à la fonction qu'on cherche à optimiser ($F(x) \in \mathbb{R}$). Voici l'agorithme:

Variables et fonctions

- Soit $x_i \in \mathbb{R}$ l'individu i
- Soit $F : \mathbb{R} \to \mathbb{R}$ la fitness qui attribue a un individu sa fitness. Cette fonction correspond à la fonction qu'on cherche à optimiser. (Dans notre cas on prend $F(x) = -x^2$)
- Soit $f_i = F(x_i)$ la fitness de l'individu i.
- Soit $M : \mathbb{R} \to \mathbb{R}$ la fonction de mutation qui associe à un individu x son image aprés mutation $M(x)$. Dans notre cas on pose $M(x) = x + \mathcal{N}(0, 1)$ ou $\mathcal{N}(0, 1)$ correspond à une valeur tirée selon une loie gaussiéne centrée réduite.
- Soit $S : \mathbb{R}^2 \to \mathbb{R}$: la fonction de séléction par tournoi. Cette fonction prend en entrée deux individus, x et x' et renvoie celui dont la fitness est la plus grande (et si leurs fitnesses sont identiques il prend l'un des deux au hasard).
- Soit x_b le meilleur individu de la population et soit $F(x_b)$ sa fitness.

Algorithme

- Initialiser chaque individu x_i de la population selon une loi uniforme entre -100 et 100.
- Pendant t itérations répéter: les opérations suivante:
  - Pour chaque individu i de la population :
    - $x_i \leftarrow S(x_i, M(x_i))$
    - Si $F(x_i) > F(x_b)$ mettre à jour x_b.

3.1 Questions

- Codez l'algorithme, plusieurs choix d'architecture sont valables. Expliquez très rapidement le choix (je n'attends pas une architecture en particulier, je veux juste que vous puissiez expliquer le choix rapidement).
- Faire un fichier contenant les colonnes suivantes : *iteration* x_b $F(x_b)$pour chaque itération de l'algo.
- Depuis Python visualiser $F(x_b)$ en fonction des *iterations* (Gnuplot, matplotlib ...)

4 Algorithme évolutionniste plus réaliste

Cette façon de représenter un individu est un peu trop simple non? On va complexifier un peu cette représentation. Vos superbes cours de bio vous ont appris les conceptes de genotype et phenotype. Le genotype est associé à l'information héréditaire de l'individu alors que le phenotype est associé aux propriétés "observables" de l'individu (morphologie, comportement ...). Nous allons donc rajouter à l'algo la notion de mapping $genotype \mapsto phenotype$. Dans cette partie on ne va pas s'interresser à la boucle évolutive mais uniquement au modèle du génôme de l'individu et au mapping.

Variables Dans ce cas nous allons considérer qu'un génôme est une liste binaire ($G = [0, 1, ...1, 1, 0]$) de longueur variable notée n qui peut subir les opérations suivantes:

- Mutation ponctuelle: l'éléments g_i du génôme est switché.
- Inversions: deux éléments du génôme g_a et g_b sont choisis aléatoirement comme points de coupure et les positions des éléments entre les deux points de coupure sont inversés (g_a échange de position avec g_b, g_{a+1} échange de position avec g_{b-1} et ainsi de suite)
- Translocation: trois éléments du génôme g_a, g_b et g_c sont choisis aléatoirement comme points de coupure et la séquence entre g_a et g_b est coupée et collée entre g_c et g_{c+1}

- Translocation: trois éléments du génôme g_a, g_b et g_c sont choisis aléatoirement comme points de coupure et la séquence entre g_a et g_b est copiée et collée entre g_c et g_{c+1}
- Délétion: deux éléments du génôme g_a et g_b sont choisis aléatoirement comme points de coupure et la séquence entre g_a et g_b est éliminée.
- Pour éviter les effets de bords nous allons considérer que le génôme est circulaire le dernier élément de G est donc voisin du premier (g_0 et g_n sont adjacents).
- Le mapping est très simple: Soit x le phenotype de l'individu, alors $x = \sum_i G_i$ (i.e. somme des 1 du génôme).
- Le nombre de fois que chaque opérateur de mutation sera utilisé est déterminé par tirage selon une loi binomiale $\mathcal{B}(n, p)$ ou n correpond à la taille du génôme et p au taux de mutation, un paramétre du modèle. (aide: numpy.random.binomial)

Algorithme

- Initialiser le génôme en fonction d'un paramétre c qui correspond à la probabilité d'avoir un 1 dans le génôme initial et en fonction d'une taille initiale n. (ici on prendra $c = 0.5$ et n variable)
- Pour chaque opérateur de mutation faire:
  - Générer le nombre de mutations à faire (loi binomiale $\mathcal{B}(n, p)$)
- Appliquer toutes les mutations dans un ordre aléatoire.

4.1 Questions

- Implémentez le modèle.
- Faire muter des génômes ayant des tailles initiales différentes et faire un fichier contenant une colonne "taille avant mutation" et une "taille aprés mutation".
- Visualiser les résultats.