Introduction à l'analyse numérique
Année : 2018-2019
Formation : L2 Mathématiques

TP noté 2 : Théorie de l'évolution en informatique et balistique

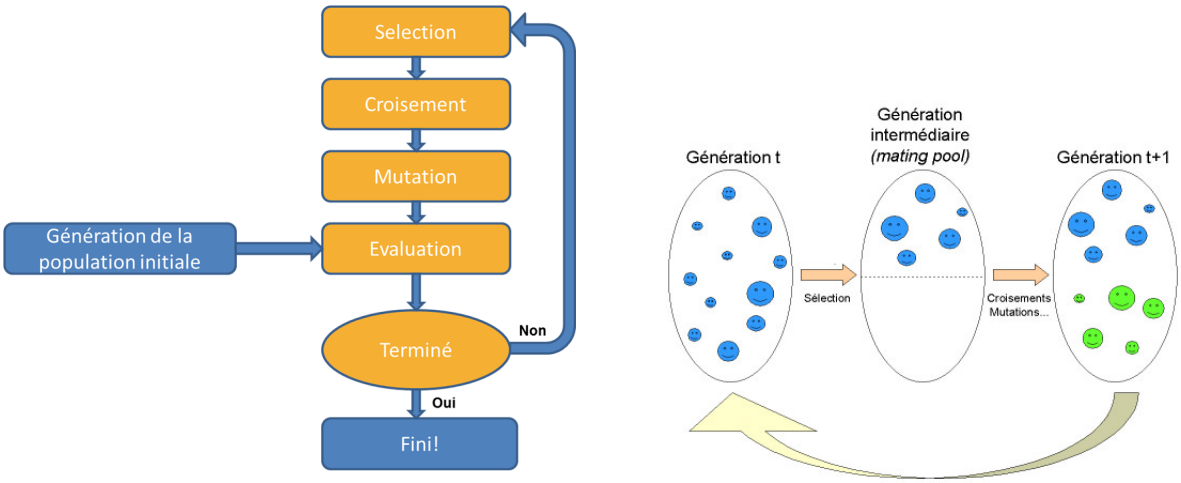
Ce sujet est composé de deux parties indépendantes : la première partie présente une méthode originale de résolution de système d'équations linéaires qui se base sur les principes de la théorie de l'évolution. La seconde partie traite du calcul de la trajectoire d'un projectile.

Pour chaque partie il faudra créer deux fichiers dans lesquels vous enregistrerez votre code : un fichier en .sci qui contiendra toutes les fonctions de la partie et un fichier en .sce qui contiendra les autres commandes. A la fin du TP vous devez envoyer vos fichiers par mail à l'adresse suivante : florian.le-manach@math.u-bordeaux.fr. N'oubliez pas d'indiquer votre nom et prénom (et celui de votre binôme si vous faites le TP à deux) dans le mail.

1 Algorithme évolutionniste de résolution d'équations linéaires

On cherche à résoudre numériquement, grâce à un algorithme dit évolutionniste, une équation de la forme $Ax = b$ avec A une matrice $n \times n$, b et x des vecteurs colonnes à n lignes.

Un algorithme évolutionniste a pour but d'obtenir une solution approchée à un problème d'optimisation pour le résoudre en un temps raisonnable. Il utilise la notion de sélection naturelle et l'applique à une population de solutions potentielles au problème donné. Cela se fait en 5 étapes comme indiqué sur les schémas suivants



(1) Génération de la population initiale (initialisation) : cette étape consiste à générer aléatoirement une population (un ensemble) de nbPop vecteurs colonnes x_i. Cette population correspond à la première génération des solutions (potentielles) de l'équation $Ax = b$. A noter que comme les vecteurs sont générés aléatoirement, ces solutions sont à priori très mauvaises.

(2) Évaluation : on vérifie lors de cette étape qu'aucun individu de la population ne résout de manière satisfaisante le problème (ici l'équation $Ax = b$). Si un des vecteurs x_i vérifie la condition $\|Ax_i - b\|_\infty < \varepsilon$ alors on aura trouvé une solution approchée de l'équation et on s'arrête. Sinon on continue et on fait jouer la sélection naturelle.

(3) Sélection : dans notre population de nbPop individus (solutions potentielles) on selectionne les nbPop/2 meilleures solutions au problème, c'est-à-dire les nbPop/2 solutions x_i qui minimisent $\|Ax_i - b\|_\infty$. En continuant l'analogie avec la théorie de l'évolution on peut considérer que les moins bonnes solutions ne survivent pas.

(4) Croisement : les indivus sélectionnés à l'étape précédente (ceux qui ont survécu) se reproduisent 2 à 2. A partir de deux vecteurs x_1 et x_2, on génère les deux vecteurs suivants : $x_3 = \frac{x_1+x_2}{2}$ et $x_4 = \frac{3x_1-x_2}{2}$.

(5) Mutation : on modifie légèrement et de manière aléatoire (avec une probabilité taux de mutation) les nouveaux individus de notre population. On applique à chaque vecteur muté x la transformation suivante : $x \mapsto x + 0.01 \times \|x\|_\infty \times e$ avec e un vecteur aléatoire dont tous les coefficients ont des valeurs comprises entre -1 et 1. Après cette étape on répète tout le procédé depuis l'étape (2).

Le but final de cette partie est de créer une fonction function x=algoEvo(A,b,eps,nbPop,taux) qui prend en paramètre une matrice A de taille $n \times n$, un vecteur colonne b de taille n, $esp > 0$, un entier $nbPop$ qui correspond à la taille de la population et $taux \in [0, 1]$. La fonction algoEvo renvoie un vecteur colonne x qui vérifie $\|Ax - b\|_\infty < eps$.

1. Définissez la fonction function s=normInf(x) qui prend en paramètre un vecteur colonne et qui renvoie $\|x\|_\infty$ (c'est-à-dire la maximum des valeurs absolues des coefficients de x). Dans cette question la seule fonction Scilab autorisée est la fonction abs.

2. Définissez la fonction function L=ini(nbPop,n) qui renvoie une liste de nbPop vecteurs colonnes de taille n générés aléatoirement, dont tous les coefficients ont des valeurs comprises entre -1 et 1.

3. Définissez la fonction function y=croisement(x1,x2) qui prend en paramètre deux vecteurs colonnes et qui renvoie y=[x3,x4] généré par le croisement de x_1 et x_2 (cf étape (4)).

4. Définissez la fonction function M=mutation(L,taux) qui prend en paramètre une liste L de vecteurs colonnes et qui renvoie la liste M des vecteurs mutés (cf étape (5)). Si on note n la longeur de la liste L, on a que les $n/2$ premiers éléments de la liste M sont identiques à ceux de L (ce sont les anciens individus qui se sont reproduis) et on modifie les $n/2$ derniers éléments de la liste avec une probabilité taux. Pour chacun de ces éléments on pourra tirer un nombre au hasard dans $[0, 1]$ avec la fonction rand et si rand() $<$ taux alors on fait muter le vecteur.

5. Définissez la fonction function c=critere(A,b,x) qui prend en paramètre une matrice A de taille $n \times n$ et des vecteurs colonnes b et x de taille n et qui renvoie la valeur $\|Ax - b\|_\infty$.

6. Définissez la fonction function s=evaluation(A,b,L,eps) qui prend en paramètre une matrice A, un vecteur colonne b, une liste L de vecteurs colonnes et $eps > 0$ et qui renvoie l'indice i s'il existe un vecteur x_i de la liste L qui vérifie $\|Ax_i - b\|_\infty < \varepsilon$. La fonction renvoie 0 sinon.
On rappelle que pour acceder à x_i en scilab on utilise la commande L(:,i)

7. Définissez la fonction function T=triBulle(L,f) qui prend en paramètre une liste L de vecteurs colonnes et une fonction f (qui à un vecteur colonne associe un nombre) et qui renvoie la liste triée T selon la fonction f, c'est-à-dire qu'un vecteur x sera dit plus petit que y si $f(x) \leq f(y)$. Pour cela on utilise l'algorithme de tri à bulle : on parcours la liste L avec une boucle for et lorsque l'on a x_i vérifiant $f(x_i) > f(x_{i+1})$, on intervertit x_i et x_{i+1}. En parcourant la liste n fois (avec n la taille de la liste) on s'assure d'obtenir une liste triée (on peut aussi parcourir la liste jusqu'à qu'il n'y ait plus de permutation à faire).
On pourra s'assurer du bon fonctionnement de cette fonction en utilisant pour la fonction f la fonction Scilab max ou la fonction normInf de la question 1.

8. A l'aide des deux dernières questions, définissez la fonction function S=selection(L,A,b) qui prend en paramètre une liste L de vecteurs colonnes, une matrice A et un vecteur colonne b et qui renvoie la liste S des $n/2$ éléments x_i de la liste L qui minimisent la quantité $\|Ax_i - b\|_\infty$ (avec n correspondant à la taille de la liste L).
On pourra utiliser la commande deff('s=f(x)','s=critere(A,b,x)') pour définir la fonction f utilisée dans la fonction TriBulle.

9. Avec toutes les fonctions précédentes, définissez la fonction `function x=algoEvo(A,b,eps,nbPop,taux)` qui répond au problème. Pour le croisement on fera se croiser les vecteurs qui sont côte à côte et on ajoutera le résultat à la fin. Par exemple si $L = [x_1, x_2, x_3, x_4, ..., x_{nbPop/2}]$ alors on croisera x_1 avec x_2, x_3 avec x_4, ... et on obtiendra $L = [x_1, ..., x_{nbPop/2}, x'_1, x'_2, ..., x'_{nbPop}]$.

10. Testez votre fonction sur des exemples de votre choix et comparez le résultat avec celui de la fonction Scilab `linsolve` (voir l'aide Scilab). On pourra par exemple prendre :
`A=2*rand(3,3)-ones(3,3)`, `b=2*rand(3,1)-ones(3,1)`, $nbPop = 4$, $eps = 0.01$ et $taux = 0.7$.

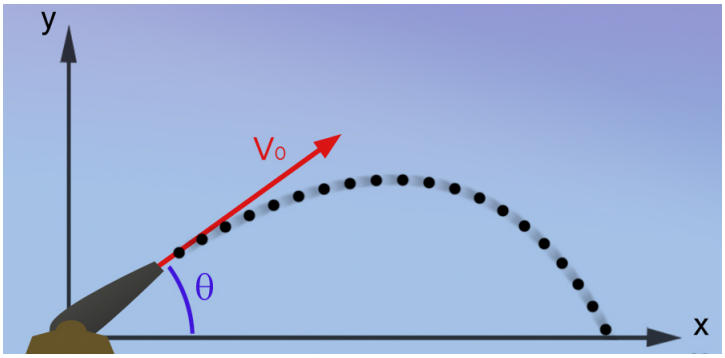
11. En modifiant (légèrement) la fonction `algoEvo`, définissez la fonction `function S=algoEvo2(A,b,eps,nbPop,taux)` qui renvoie la liste des meilleures solutions de chaque étape. Affichez sur un graphe l'évolution de l'erreur (c'est-à-dire $\|Ax - b\|_\infty$) pour l'individu le plus performant de chaque étape, en fonction du nombre d'étapes.

12. (BONUS) : Codez une fonction qui résout l'équation $Ax = b$ par la méthode de Jacobi et comparez l'efficacité de cette méthode avec la méthode précédente sur une matrice à diagonale strictement dominante. On prendra le premier élément de la suite de Jacobi $x^{\mathbf{0}} = 0$ et on rappelle la formule de récurrence

$$x_i^{\mathbf{k+1}} = \frac{1}{a_{ii}}\Big(b_i - \sum_{j=1, j\neq i}^{n} a_{ij} x_j^{\mathbf{k}}\Big).$$

2 Balistique

Le but de cette partie et d'étudier la trajectoire d'un projectile lancer par un canon.



Le projectile part avec une vitesse $v_0 = 950$ (m/s) et le canon est orienté selon un angle θ que l'on peut faire varier. L'objectif final de cette exercice est de déterminer l'angle θ afin d'atteindre une cible à une distance fixée. Toute la difficulté de ce problème physique est de considérer la force de frottement de l'air qui rend complexe la résolution exacte de l'équation de la trajectoire du projectile, ce qui motive l'utilisation d'une méthode de résolution numérique.

Si on considère $M(t) = (x(t), y(t))$ la position du projectile au temps t, on obtient l'équation différentielle suivante

$$\begin{cases} x''(t) &= -\mu/m \sqrt{x'(t)^2 + y'(t)^2}\, x'(t) \\ y''(t) &= -g/m - \mu/m \sqrt{x'(t)^2 + y'(t)^2}\, y'(t) \end{cases}$$

avec pour condition initiale $M(0) = (0, 0)$ et $M'(0) = (v_0 \cos(\theta), v_0 \sin(\theta))$. Dans toute la suite on considère $m = 45$ (kg), $g = 9.8$ (m/s^2) et $\mu = 0,05$ (kg/s).

1. En considérant le vecteur $Y(t) = (x(t), y(t), x'(t), y'(t))$, déterminez et définissez en scilab la fonction `function R=f(Y)` vérifiant $Y'(t) = f(Y(t))$. On pourra au choisir de considérer Y comme un vecteur colonne. On rappelle que pour accéder au premier (resp. ième) élément de `Y` on utilise `Y(1)` (resp. `Y(i)`) et qu'on utilise le point-virgule pour définir des vecteurs colonnes.

2. Définissez la fonction `function L=RK2(f,theta,dt)` qui prend en paramètre la fonction f de l'équation $Y'(t) = f(Y(t))$, l'angle θ du problème et un pas de temps dt. Cette fonction renvoie la liste des Y_n vérifiant la formule de la méthodes de Runge-Kutta d'ordre 2, jusqu'à que le projectile touche le sol (c'est-à-dire tant que $y(t) \geq 0$).
Si vous ne vous rappelez pas de la méthode RK2 utilisez la méthode d'Euler explicite (cela rapporte moins de points).

3. Testez la fonction précédente pour les angles $\theta = \pi/6$, $\theta = \pi/4$ et $\theta = \pi/3$. Dessinez sur un même graphique la trajectoire des trois projectiles. On pourra prendre un pas de temps $dt = 0,1$

On veut maintenant atteindre une cible à 3 000 m de distance avec une précision de 100 m.

4. Écrivez un script (ou une fonction) qui détermine l'angle θ pour atteindre la cible. Pour cela on testera tous les angles de 0 à $\pi/2$ avec un pas de $0,01$.