PROJETS d'ALGORITHMIQUE, 2016

Vous avez le choix entre plusieurs projets. Vous pouvez réaliser le projet seul ou par deux. Bien sûr, il en sera tenu compte dans votre note de projet.

PROJET 1 : Optimisation par la méthode JAYA

Soit une fonction f de deux vecteurs :
P, un vecteur de paramètres, dont la valeur ne peut être modifiée,
X, un vecteur de coordonnées, dont les valeurs sont modifiables.
Le but est de trouver X tel que f(P, X) soit minimale.
On suppose que f(P, X) est infini quand la norme de X est infinie. C'est le cas si f est une somme de termes carrés.

Exemple :

$$f(x,p)= (x0-p0)^2 + (x1-p1)^2 + (x2-p2)^2$$

est minimale pour x=p.

Le vecteur p peut être vide. Votre programme doit marcher pour un nombre quelconque de paramètres et d'inconnues.

La méthode Jaya, ainsi que des exemples de fonction à minimiser, est présentée dans l'article :
http://www.growingscience.com/ijiec/Vol7/IJIEC_2015_32.pdf

C'est cet article qui fait foi. Il est assez mal écrit, mais ce sera votre lot quotidien dans votre vie professionnelle.

Présentons le principe de Jaya. Une population, ou nuage, de N points X[0]...X[N-1] est généré. Chaque point est en dimension D. L'utilisateur doit pouvoir fixer la valeur de N. Le nuage initial est, par exemple, généré au hasard dans une boîte (carré, cube, etc) centrée en un X donné et de demi-côté donné.

On procède ensuite à plusieurs étapes. Dans une étape, la valeur de la fonction f en chaque point du nuage est calculée. Le meilleur point du nuage, A (avec la plus petite valeur de f) et le pire point du nuage Z (avec la plus grande valeur de f) sont calculés. Ensuite pour chaque point X[n] du nuage, vous calculez un point X2[n] tel que :

```
pour d de 0 à D-1  faire  (D est la dimension de chaque point X)
        R1[ d ] :=  random(0., 1.)
        R2[ d ] :=  random(0., 1.)
pour n de 0 à N-1 faire   (N est le nombre de points du nuage)
        pour chaque coordonnée d de  0 à D-1  faire
                X2[ n ][ d]  := X[ n ][ d ] +  R1[ d ] ( A[ d ]  - | X[ n ][ d ]  | )
                                            -  R2[ d ] (  Z[ d ]  - | X[ n ][ d ]  | )
```

Variante : vous pouvez essayer de générer un nombre aléatoire chaque fois que vous en avez besoin : ainsi, vous n'avez pas besoin de les stocker dans des tableaux.

Quand tous les points X2[n] sont calculés : pour chaque n de 0 à N-1, le point X2[n] remplace X[n] si et seulement si X2[n] est meilleur que X[n] (la valeur de f est plus petite). Cette mise à jour de X conclut l'étape. Après plusieurs étapes, le nuage de points converge vers un minimum local (s'il en existe un), autrement dit tous les points X sont proches les uns des autres. Le programme est terminé soit après un nombre maximal spécifié d'étapes, soit quand tous les points du nuage courant sont égaux à une tolérance près (utilisez les deux conditions !)

Vous programmerez plusieurs exemples de l'article original. Vous afficherez les 2 premières coordonnées (il y en a D) des points X du nuage (sauf si D=1). Vous tracerez le chemin suivi par chaque point du nuage. Faites en sorte que l'affichage permette d'avoir une intuition de la méthode. Par exemple vous pouvez tracer d'une couleur plus pâle les points X2[n] qui ne remplacent pas leur point X[n].

Pour mettre au point votre programme, testez d'abord des exemples simples.

Jaya permet de maximiser une fonction g : il suffit de minimiser f(P, X)= - g(P, X).

Jaya permet de résoudre des systèmes d'équations, linéaires ou non linéaires : pour résoudre f1(x1, x2)=f2(x1, x2)=0, il suffit de minimiser $f1(X)^2 + f2(X)^2$. La généralisation à un nombre quelconque d'équations et d'inconnues est évidente.

La méthode est intéressante car elle est simple, massivement parallèle, et générale, autrement dit, elle ne suppose rien sur la fonction à optimiser : elle n'a pas besoin d'être dérivable , convexe, analytique, etc... Pour que Jaya fonctionne, il vaut mieux que la fonction soit continue.

Jaya peut être utilisé pour trouver le point X de R^3 tel que la somme de la distance de X au cube $[0, 1]^3$ et à une droite ou un plan donné soit minimale. L'équation de la droite ou du plan peut être mise sous la forme AX-B=0, avec A une matrice et B un vecteur donnés. La distance de X au cube est la distance entre X et sa projection P sur le cube $[0, 1]^3$:
pour i de 0 à 2, P[i] = X[i] si X[i] est entre 0 et 1,
P[i] = 0 si X[i] est négatif,
P[i] = 1 si X[i] est plus grand que 1
Remarque : si X est à l'intérieur du cube, alors P=X. La généralisation à des cubes quelconques parallèles aux axes est évidente (stocker dans le paramètre P ce genre de données).

Vous programmerez la résolution du problème précédent en 2D : trouver le point X tel que la somme de la distance de X au carré $[0, 1]^2$ et à une droite passant par deux points donnés est minimale. Pour garantir l'unicité de la solution, vous considérerez f(X)= dcarre(X) + ddroite(X) + $(dcarre(X) - ddroite(X))^2$ où dcarre(X) est la distance au carré unité et ddroite est la distance à la droite. Attention : ne confondez pas nombres flottants et nombres entiers. Il vous faut faire une règle de 3 (ou une « mise à l'échelle) entre les coordonnées dans « le monde » et les coordonnées de l'écran.

Si vous résolvez ce problème en 3D (en java3D par exemple), vous pouvez, exceptionnellement, vous mettre à 3 étudiants.

PROJET 2 : Dessin de courbes définies par leurs équations en notation postfixe

Le cercle de centre x0, y0 et de rayon R a comme équation, en notation postfixe :
x x0 OPP + CARRE y y0 OPP + CARRE + R CARRE OPP
avec x0 et y0 et R remplacés par des valeurs flottantes, OPP est l'opération unaire opposé (qui à x associe -x), CARRE est l'élévation au carré (opération unaire qui à x associe x^2).

N'utilisez pas de – ou de / car l'ordre des paramètres pose des problèmes. Ils sont éviter en utilisant OPP (et INV pour l'inverse).

Vous pouvez éventuellement permettre des variables locales, disons avec la syntaxe :
expression postfixe QUOTE nomvar
par exemple
1 2 3 * + QUOTE tmp
qui signifie que 7 est stockée dans la variable tmp, et ensuite tmp pourra être utilisée comme x et y.
Ceci permet de ne calculer qu'une fois les expressions partagées. Bien sûr vous pouvez utiliser un seul caractère pour les identifieurs OPP, QUOTE, tmp...

Votre programme doit prendre en entrée une expression postfixe (une chaîne de caractères), un cadre, et doit tracer la courbe à l'intérieur du cadre spécifié : un point (x, y) est sur la courbe si l'expression s'annule en (x, y). Vous pouvez re-utiliser le programme dans les archives qui utilise une arithmétique d'intervalles pour tracer une courbe. Il se trouve à l'URL:
http://ufrsciencestech.u-bourgogne.fr/licence2/Info31/CORRECTIONS/DAG2/

Pour des équations de courbes implicites :
https://fr.wikipedia.org/wiki/Lima%C3%A7on_de_Pascal
https://fr.wikipedia.org/wiki/Cardio%C3%AFde
https://fr.wikipedia.org/wiki/Astro%C3%AFde
et d'autres accessibles depuis : https://fr.wikipedia.org/wiki/Courbe_plane#Courbes_planes_classiques

En 2013, des TP notés concernaient l'évaluation des expressions postfixes. Attention, ici x et y sont des intervalles.

Variante 1 : l'équation implicite définit une région intérieure, et une région extérieure, que vous pouvez remplir avec des couleurs différentes. Vous pouvez utiliser des opérations booléennes (inter, union, différence) entre les régions. Vous pouvez aussi translater, changer la taille ou tourner des régions. Il est alors commode de pouvoir définir des régions. Par exemple, pour définir le cercle unité cu ainsi :
: cu x x * y y * 1 NEG .
Et ensuite pour translater le cercle :
: cu2 cu 1 2 trans .
Le reste est laissé à votre imagination. Pour vous aider, vous pouvez consulter les pages traitant de Forth ou de Postscript, deux langages à pile.

Variante 2 : faire des animations, sauver les images au format .ppm (c'est facile par programme), les convertir dans un format tiff (par convert) pour visualiser les animations. En ce cas, une variable t (pour temps) peut être commode, en plus de la variable x ou y.