Algorithmes génétiques

A. Bouvet, F. Collin, R. Velasco

Résumé : on étudie la recherche d'extremum global d'une fonction sur un intervalle donné. On propose ici une méthode basée sur le principe des algorithmes génétiques et une réalisation en Mathematica. **Mots clés :** optimisation, algorithme génétique, duplication, crossover.

Abstract: the search of a global extremum of a function on a given interval is investigated. A method using the principle of genetic algorithms and a Mathematica program is explained. **Keywords:** optimization, genetic algorithm, duplication, crossover.

Introduction

On utilise principalement l'ouvrage *Algorithmes Génétiques* de *David E. Golberg* pour comprendre les notions avec lesquelles on va travailler. On dispose du langage de programmation nécessaire pour la réalisation du projet. La programmation des algorithmes génétiques revient tout d'abord à utiliser un langage binaire pour coder la population choisie aléatoirement et ensuite à sélectionner les individus les mieux adaptés pour améliorer la population grâce à un processus de reproduction particulier.

Principe de fonctionnement

Les algorithmes génétiques (AG) sont basés sur des mécanismes de sélection inspirés de la sélection naturelle et de la génétique. Par sélection naturelle, on entend survie des structures les mieux adaptées ; par génétique, on entend échange d'information pseudo-aléatoire. En quelque sorte, "la nature fait un meilleur travail que l'ingénieur".

A chaque étape, les AG exploitent l'information obtenue précédemment, dans l'espoir permanent de l'améliorer. Ils sont simples d'un point de vue calculatoire mais très performants dans leur recherche d'amélioration.

Objectifs des AG : l'optimisation

"Le désir humain de perfection trouve son expression dans la théorie de l'optimisation. Elle étudie comment décrire et atteindre ce qui est meilleur, une fois que l'on connaît comment mesurer et modifier ce qui est bon et mauvais... La théorie de l'optimisation comprend l'étude quantitative des optimums et les méthodes pour les trouver" (Beightler, Phillips, Wilde). Pour les systèmes complexes, on n'a pas besoin de trouver l'optimum, on se contente d'améliorer à volonté.

Originalité des AG

Les AG :
– utilisent un codage des paramètres (et non les paramètres eux-mêmes) ;
– travaillent sur une population de points (et non un point unique) ;
– utilisent les valeurs de la fonction étudiée (et non la dérivée par exemple...) ;
– utilisent des règles de transition probabilistes (et non déterministes).

Les opérateurs

Il y en a 2 principaux :

– la duplication, à savoir la copie des chaînes en fonction des valeurs de la fonction d'adaptation ;
– la reproduction qui consiste à donner plus d'importance dans l'algorithme aux chaînes donnant la valeur la plus grande à la fonction ;

– le crossover qui est l'échange de morceaux de chaînes entre 2 chaînes distinctes, conduisant à la création de nouveaux individus.

Ces mécanismes utilisent un générateur de nombres aléatoires, la copie de chaînes et les échanges partiels de chaînes (on entrevoit ici les programmes à mettre en œuvre avec Mathematica pour programmer un AG).

■ En résumé

En résumé, la mise en œuvre d'un algorithme génétique repose sur :

– une population de n chaînes ;
– la copie des n chaînes en pondérant leur importance ;
– la constitution de paires ;
– l'échange de sous-chaînes.

■ Analyse du problème

Compte tenu de la méthode de programmation employée pour trouver l'extremum d'une fonction, à savoir celle s'appuyant sur les algorithmes génétiques, certaines contraintes doivent être mentionnées. Elles feront office de cahier des charges du projet. Le programme ne pourra donner en réponse finale qu'une valeur dont le nombre de décimales est fixé. Ainsi, la valeur en laquelle la fonction présentera un maximum ne pourra être qu'approchée. L'utilisation du codage binaire impose à l'intervalle de départ d'être de longueur l, avec l=2^n où n entier naturel. Le programme ne peut être utilisé que pour des fonctions continues ou présentant des discontinuités de première espèce.

■ Conception du programme

Le principe de ce programme est de faire évoluer une population initiale aléatoire représentant des nombres d'un intervalle souhaité. Cette évolution, empruntée à la nature, privilégie les meilleurs individus lors de la reproduction.

Mathematica peut avoir des variables parasites en mémoire : c'est pourquoi nous commençons par les effacer.

```
ClearAll["Global`*"]
```

Ensuite, l'utilisateur choisit ses paramètres : f sera la fonction étudiée, a la valeur inférieure de l'intervalle, n la valeur définissant le nombre de subdivisions (2^n) et la longueur de l'intervalle (2^n/10^p), d la moitié du nombre d'individus de la population, p le nombre de décimales de la solution et r le nombre d'itérations.

Pour dupliquer les meilleurs individus, on utilisera le programme ci-dessous : à partir d'une liste préalablement triée, il élimine le premier élément de cette liste et duplique son dernier élément, ajouté en fin de liste.

```
duplication[l_] := Join[Rest[l], {Last[l]}]
```

Le programme suivant agit sur des nombres codés en binaire : à partir de deux nombres binaires, il joint les n premiers bits du premier nombre au second nombre privé de ses n premiers bits, n étant choisi aléatoirement.

```
crossover[{x_, y_}] :=
 With[{n = Random[Integer, {1, Length[x]}]},
    {Join[Take[x, n], Drop[y, n]],
     Join[Take[y, n], Drop[x, n]]}]
```

Le programme qui suit sépare une liste (de longueur paire) en deux parties de même taille. Associé à la fonction `Transpose`, ce programme créera des paires d'individus.

```
splitList[l_] := {Take[l, Length[l] / 2], Take[l, -Length[l] / 2]}
```

Arrive ensuite le programme d'initialisation : il génère une population de nombres codés en binaire.

```
sample[n_, d_] :=
    Table[
         IntegerDigits[Random[Integer, {0, 2^n - 1}], 2, n],
    {2 d}]
```

A chaque itération, on obtient une nouvelle liste d'individus. Pour réitérer les opérations de classement, de duplication et de crossover, on programme la fonction `loop` (basée sur `Nest`), laquelle classe la liste par ordre croissant des valeurs prises par la fonction f en ces points. On duplique ensuite plusieurs fois le dernier élement de la liste, qui correspond à l'individu donnant la valeur la plus élevée à la fonction f. On apparie alors les premiers élements aux derniers et on fait agir le crossover sur ces paires.

```
loop[l_, {f_, a_, d_, p_}] :=
 Flatten[Map[crossover,
    Transpose[splitList[
     Nest[duplication,
        Sort[l,
          (f[(FromDigits[#1, 2] / (10^p) + a)] ≤
          f[(FromDigits[#2, 2] / (10^p) + a)] &)
        ],
     d - 1]
    ]]
 ], 1]
```

Enfin, après avoir fait agir `loop` plusieurs fois sur les listes d'individus successives, on convertit l'ultime liste en nombre rationnel. On aboutit ainsi au programme principal.

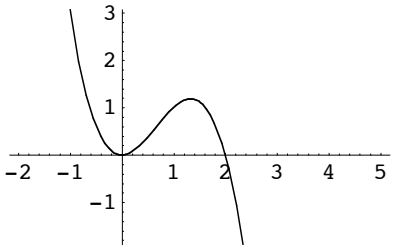
```
GeneticAlgorithm[f_, a_, n_, d_, p_, r_] :=
 Map[
    (FromDigits[#, 2] / (10^p) + a) &,
    Nest[loop[#, {f, a, d, p}] &, sample[n, d], r]
 ]
```

■ Exemple

Prenons l'exemple d'une fonction f présentant un maximum en x=4/3=1.333...

```
f[x_] := -x^3 + 2 x^2
```

```
Plot[f[x], {x, -2, 5}];
```




```
GeneticAlgorithm[f, 0, 12, 30, 3, 100];
```

```
Apply[Equal, %]
```

```
True
```

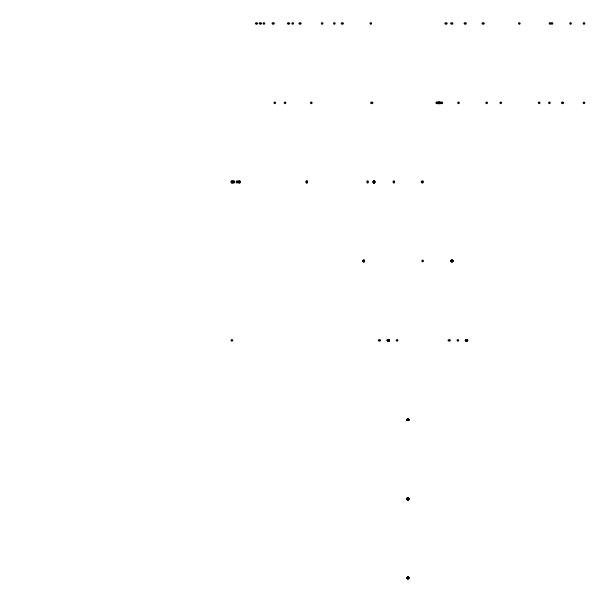
```
First[%%]
```

$$\frac{1333}{1000}$$

```
N[GeneticAlgorithm[f, 0, 12, 10, 3, 5]]
```

```
{1.333, 1.333, 1.333, 1.333, 1.333, 1.333,
 1.333, 1.333, 1.333, 1.333, 1.333, 1.333, 1.333,
 1.333, 1.333, 1.333, 1.333, 1.333, 1.333, 1.333}
```

```
Do[
 ListPlot[Map[{#, 0} &, N[GeneticAlgorithm[f, 0, 12, 10, 3, k]]],
   Axes -> None, AspectRatio -> 1 / 10];, {k, 0, 7}]
```



```
GeneticAlgorithm[f, -1, 12, 20, 3, 50];
```

```
Apply[Equal, %]
```

```
True
```

```
N[First[%%]]
```

```
-0.996
```

L'algorithme détecte un maximum en -1, extrêmité de l'intervalle.

■ Utilisation du programme

Le codage binaire ne permet pas d'utiliser des nombres négatifs : nous ramenons alors la fonction sur un intervalle de borne inférieure 0 et de longueur 2^n lors de toute l'étude. La dernière opération du programme permet de se ramener à l'intervalle initial. Ce subterfuge passe donc inaperçu dans l'utilisation du programme.

Pour utiliser le programme de façon optimale, il est conseillé de suivre les recommandations suivantes : on commence par chercher la valeur donnant le maximun de f en nombre entier (ici 0) ; il est judicieux de choisir d=30 et n= 7 pour cette première étape. L'étude se fera alors sur un intevalle de longueur 128 à partir

de a. Donc, suivant l'intervalle désiré, il faut harmoniser en choisissant un a qui va bien. Pour les décimales suivantes, la valeur finale que l'on obtiendra sera comprise dans l'intervalle [a, a+(2^n-1)/10^p].

Il convient donc de modifier a, p, d et surtout n (qui doit rester raisonnable c'est-à-dire inférieur à 15) pour que la solution entière précédemment approchée se trouve dans l'intervalle mentionné ; par exemple, pour p=3, on prend n=11, d=30 et a=(solution entière-1), l'intervalle est alors [a, a+2,047]. Si la liste finale contient des éléments distincts, il suffit d'augmenter le nombre d'itérations (fixé dans le premier exemple à 100).

■ Conclusion

La conception du programme emprunte le principe de la sélection naturelle. En effet, les nombres testés sont regoupés dans une liste qui représente une population. Au départ, les nombres sont choisis au hasard mais le programme va permettre de privilégier les individus (nombres) les mieux adaptés (ceux se rapprochant du maximum de la fonction). Ainsi, au cours des générations successives, il ne subsistera que les meilleurs individus.Toutefois, dans des cas extrêmes, il est indispensable de faire intervenir des mutations afin d'éviter la convergence des élements de la liste vers une solution erronée. La mutation va alors faire apparaître de nouveaux allèles (changement de valeur d'un bit). Une amélioration du point de vue de ces mutations pourrait encore être apportée au programme et une optimisation pourrait en être également faite en étudiant les valeurs des paramètres à choisir pour permettre une convergence rapide vers une solution correcte.

Ce micro-projet nous a amené à la réflexion suivante : "La conscience professionnelle des ingénieurs, c'est de passer 30 heures sur un projet pour le rendre performant quand il leur était conseillé d'y passer 12 heures."

■ Bibliographie

[1] GOLBERG E. D. *Algorithmes Génétiques*, Addison-Wesley France, 1994

[2] BARRÈRE R. *Mathematica, calcul formel et programmation symbolique pour l'informatique scientifique : version abrégée*, polycopié ensmm, 2001

[3] BARRÈRE R. *Calcul scientifique avec Mathematica, projets de modélisation - simulation - conception*, polycopié ensmm, 2001-02