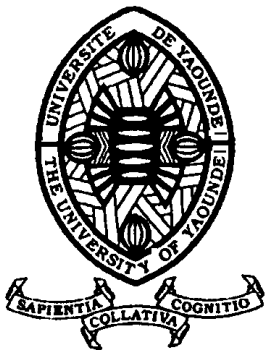
Expose Groupe xx INFO 4127 TECHNIQUE D'OPTIMISATION
THEME:
MÉTHODE DES SUITES DE FIBONACCI



Université de Yaoundé I
University of Yaounde I

Département D'Informatique
Department of Computer Science

OUSSAH SIGHA Sony Nelson
FEZEU GHOMSI Eugene Clotaire 17Q2864

27 Décembre 2020

Contents

INTRODUCTION

Avant d'entreprendre l'étude de problèmes d'optimisation, il est bon de bien définir ce qu'est un problème d'optimisation. Dans toute sa généralité, le problème d'optimisation consiste à déterminer la plus petite (grande) valeur possible qu'une fonction réelle f: E $\longrightarrow \mathbb{R}$ nommée fonction objectif puisse prendre dans l'ensemble E nommé ensemble réalisable. Sous forme mathématique, ceci s'exprime (pour le cas de la minimisation)

$$f^*(x) = \inf f(x) x \in E \ .$$

Il existe plusieurs techniques d'optimisations classifiés selon que certains problèmes a optimiser sont avec ou sans contraintes. Nous étudierons ici une méthode d'optimisation sans contrainte notamment la **Méthode de suite Fibonacci**.

1 MÉTHODE DE SUITE DE FIBONACCI

1.1 Définition

La méthode de Fibonacci est une technique d'optimisation sans contraintes directe (recherche de la solution sans dériver f) appliqué aux fonctions monodimensionnelle strictement convexe. en d'autre terme elle s'applique sur des fonctions a une variable . L'approche de Fibonacci est une variante plus optimal de la méthode de dichotomie étant donnée que le choix du nouvel intervalle $[a^k; b^k]$ se fait suivant les nombres issue de la suite de Fibonacci, elle est aussi très adapté lorsque trouver la dérivée de la fonction à optimiser s'avère difficile.

1.2 Condition d'utilisation

Une méthode de recherche directe est une méthode qui repose uniquement sur l'évaluation de $f(x)$ sur une séquence $x_1, x_2, ..., x_n$ et comparer des valeurs afin de trouver un optimiseur de f. Les méthodes directes sont généralement appliquées dans les circonstances suivantes:

- la fonction $f(x)$ n'est pas différentiable;
- les dérivées de f sont compliquées à calculer ou même inexistantes;
- la fonction a peu de variables;
- l'emplacement d'une solution optimale est à peu près connu.

.

1.3 Principe

De façon générale le principe de la méthode de suite de Fibonacci est basé sur le lemme suivant:

Lemme. *Soit f: [a;b] $\longrightarrow \mathbb{R}$ une fonction uni-modale et x^D, x^G tels que $a < x^G < x^D < b$ alors:*

- $f(x^G) < f(x^D) \Longrightarrow \widehat{x} \in [a; x^D]$
- $f(x^G) > f(x^D) \Longrightarrow \widehat{x} \in [x^G; b]$
- $f(x^G) = f(x^D) \Longrightarrow \widehat{x} \in [x^G; x^D]$, *$\widehat{x}$ représente l'optimum (minimum ou maximum) rechercher.*

La spécificité de cette méthode réside dans le choix des points x^G et x^D. En effet ces points sont déterminés par les nombre de la suite de Fibonacci. Pour rappel la suite de Fibonacci se définit comme suit:

$$F_0 = F_1 = 1$$

$$F_{k+2} = F_{k+1} + F_k$$

$$\lim_{k\to\infty} \frac{F_{k+1}}{F_k} \longrightarrow \frac{1+\sqrt{5}}{2}$$

les point x^G et x^D sont donc définis de manière suivante:

$$x_k^G = a_k + \frac{F_{N-k}}{F_{N+2-k}}(b_k - a_k), x_k^D = a_k + \frac{F_{N+1-k}}{F_{N+2-k}}(b_k - a_k)$$

on utilise ensuite le lemme précédent pour trouver l'intervalle dans lequel le minimum peut se trouver et on s'arrête après un nombre N d'itérations.

1.4 Performance et Limites

Étant donnée que les nombres de la suite de Fibonacci croissent rapidement vers l'infini, les distances des intervalles $[a_k; b_k]$ tendent plus rapidement vers 0 ce qui fait de la méthode des suites de Fibonacci la méthode qui converge plus rapidement que toutes autre méthode d'optimisation sans contraintes mono-linéaire. La méthode des suites de Fibonacci requiert un nombre N d'itérations, ainsi lorsque N $\longrightarrow \infty$ la complexité de l'algorithme s'en trouve lourdement impacté. La méthode de section dorée vient donc compléter celle des suites de Fibonacci en se basant sur le fait que

$$\lim_{N\to\infty} \frac{F_{N+1}}{F_N} \longrightarrow \frac{1+\sqrt{5}}{2}$$

en posant donc $\tau = \frac{1+\sqrt{5}}{2}$ (Le nombre d'or, d'où le nom de **section dorée**) on peut écrire

$$\lim_{N\to\infty} \frac{F_N}{F_{N+1}} \longrightarrow \frac{1}{\tau}$$

. Et donc les points x^G et x^D peuvent se réécrire indépendamment de N comme suit:

$$x_k^G = a_k + \frac{1}{\tau}(b_k - a_k), x_k^D = a_k + \frac{1}{\tau}(b_k - a_k)$$

1.5 Algorithme

Lorsqu'on connaît le principe de la méthode , l'algorithme de Fibonacci se déduit facilement.

1.5.1 Simulateur ou évaluation de la fonction aux point ak et bk

- on se donne N = nombre total de fois ou l'on évaluera f en un point
- initialisation : $a_1 = a, b_1 = b$
- itérations :

  $$x_k^G = a_k + \frac{F_{N-k}}{F_{N+2-k}}(b_k - a_k)$$

  et

  $$x_k^D = a_k + \frac{F_{N+1-k}}{F_{N+2-k}}(b_k - a_k)$$

- Si:
  - $f(x_k^G) < f(x_k^D) \Longrightarrow a_{k+1} = a_k et b_{k+1} = x_k^D$
  - $f(x_k^G) > f(x_k^D) \Longrightarrow a_{k+1} = x_k^G et b_{k+1} = b_k$
  - $f(x_k^G) = f(x_k^D) \Longrightarrow a_{k+2} = x_k^G et b_{k+2} = x_k^D$
- arrêt : après N itérations.

1.5.2 Optimiseur ou choix des points ak+1 et bk+1

Calcul de x_{k+1}^D et x_{k+1}^G :

▶ cas $f(x_k^G) < f(x_k^D)$

- $a_{k+1} = a_k$ et $b_{k+1} = x_k^D$
- $x_{k+1}^D = a_{k+1} + \frac{F_{N+1-(k+1)}}{F_{N+2-(k+1)}}(b_{k+1} - a_{k+1})$
- $= a_k + \frac{F_{N-k}}{F_{N+1-k}}(x_k^D - a_k)$
- $= a_k + \frac{F_{N-k}}{F_{N+1-k}} \times \frac{F_{N+1-k}}{F_{N+2-k}}(b_k - a_k)$
- $= a_k + \frac{F_{N-k}}{F_{N+2-k}}(b_k - a_k)$
- $= x_k^G$

Donc seul $f(x_{k+1}^G)$ doit être évaluer à l'itération k+1. Longueur de l'intervalle : $b_{k+1} - a_{k+1} = x_k^D - a_k = \frac{F_{N+1-k}}{F_{N+2-k}}(b_k - a_k)$.

► Cas $f(x_k^G) > f(x_k^D)$

Ce déduit du cas précédent par symétrie. Seul $f(x_{k+1}^D)$ doit être évaluer à l'itération k+1, et longueur de l'intervalle : $b_{k+1} - a_{k+1} = x_k^D - a_k = \frac{F_{N+1-k}}{F_{N+2-k}}(b_k - a_k)$.

► Cas $f(x_k^G) = f(x_k^D)$

$f(x_{k+2}^G)$ et $f(x_{k+2}^D)$ doivent être évalués $b_{k+2} - ak + 2 \leq \frac{F_{N-k}}{F_{N+2-k}}(b_k - a_k)$.

A la sortie de l'algorithme la longueur de l'intervalle

$$(b_N - a_N) \leq \prod_{k=1}^{N-1} \frac{F_{N+1-k}}{F_{N+2-k}}(b-a) = \frac{(b-a)}{F_{N+1}}$$

2 CHOIX DES FONCTION POUR L'IMPLÉMENTATION DE LA MÉTHODE DES SUITES DE FIBONACCI

Les fonction uni-modales choisies pour l'application de la méthode des suites de Fibonacci sont les suivantes:
$f(x) = \exp^{x(x-1)}$ sur $[-1;1]$ et $g(x) = x^2(1-\cos(x))$ sur $[-1;\frac{1}{2}]$.

2.1 Vérification de L'hypothèse d'Uni-modalité

Définition. *Une fonction f, partout définie sur I, est dite* ***uni-modale*** *sur I si :*

- *elle admet un unique minimum $\widehat{x}$ dans I*
- *elle est strictement décroissante sur $I \cap]-\infty;\widehat{x}[$ et strictement croissante sur $I \cap]\widehat{x},+\infty[$*

► Pour la fonction $f(x) = \exp^{x(x-1)}$ sur $[-1;1]$ on a:

$$f'(x) = 0 \Leftrightarrow (2x-1)\exp^{x(x-1)} = 0$$

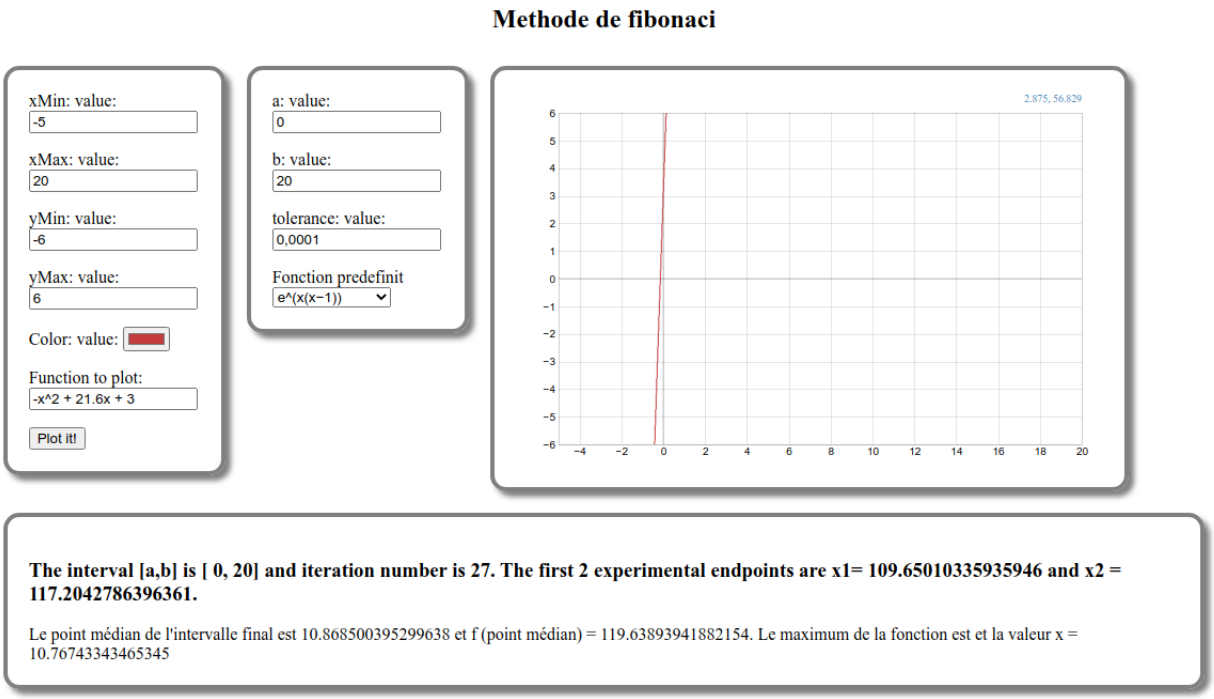
L'unique solution est $\frac{1}{2} \in [-1;1]$. La première condition est donc vérifier. De plus $f'(x)$ est strictement décroissante sur $[-1;\frac{1}{2}[= [-1;1] \cap]-\infty;\frac{1}{2}[$ et strictement croissante sur $]\frac{1}{2};1] = [-1;1] \cap]\frac{1}{2},+\infty[$, ainsi les deux propriétés sont vérifiées. Donc $f(x) = \exp^{x(x-1)}$ sur $[-1;1]$ est uni-modale.

► Pour la fonction $g(x) = x^2(1-\cos(x))sur[-1;\frac{1}{2}]$ on a:

$$g'(x) = 0 \Leftrightarrow 2x(1-\cos(x)) + x^2\sin(x) = 0$$

L'unique solution est $0 \in [-1;\frac{1}{2}]$. La première condition est donc vérifier. De plus $g'(x)$ est strictement décroissant sur $[-1;0[= [-1;\frac{1}{2}] \cap]-\infty;0[$ et strictement croissante sur $]0;\frac{1}{2}] = [-1;\frac{1}{2}] \cap]\frac{1}{2},+\infty[$, ainsi les deux propriétés sont vérifiées. Donc $g(x) = x^2(1-\cos(x))sur[-1;\frac{1}{2}]$ est uni-modale.

3 IMPLÉMENTATION DE LA MÉTHODE DES SUITES DE FIBONACCI



l'implementation de l'algorithme de fibonacci est la suivante

```
function fib(n) {
  const preCalculated = Array(n + 1).fill(0)
  preCalculated[1] = 1

  for (var i = 0; i <= n; i++) {
    preCalculated[i + 2] = preCalculated[i + 1] + preCalculated[i]
  }

  return preCalculated[n]

}
```

la genaraton de valeur de fait lors de la phase d'initialisation et celle d'itération.

```
    function FIBSearch(f,a,b,T){
      var L, M, j, C, x1, x2, N, val;
      L = (b-a)/T;
      M = Math.round(L);
      j=0;
      do{
        C = fib(j);
        if( C<L ){
          j = j+1;

        }
      }while(C<L);
      N = j;
      x1 = a+(fib(N-2)/fib(N))*(b-a);
      x2 = a+(fib(N-1)/fib(N))*(b-a);
      iterate(f,a,b,N,x1,x2);
    }
        function iterate(f,a,b, N,x1,x2){
      var x1n = [],x2n = [],an = [],bn = [],i,fx1=[],fx2 = [],j,fmid;


     i = 1;
     x1n[1] = x1;
     x2n[1] = x2;
     an[1] = a;
     bn[1] = b;
     i = 1;

    for(var j=1; j < N + 1; j++){
```

```
30
31        fx1[i] = evalf(f,x1n[i]);
32        fx2[i] = evalf(f,x2n[i]);
33        if (fx1[i]<=fx2[i]){
34          an[i+1] = x1n[i];
35          bn[i+1] = bn[i];
36          x1n[i+1] = x2n[i];
37          x2n[i+1] = an[i+1]+(fib(N-i-1)/fib(N-i))*(bn[i+1]-an[i+1]);
38
39        }else{
40          an[i+1] = an[i];
41          bn[i+1] = x2n[i];
42          x2n[i+1] = x1n[i];
43          x1n[i+1] = an[i+1]+(fib(N-i-2)/fib(N-i))*(bn[i+1]-an[i+1]);
44
45        }
46        i = i+1;
47
48          mdpt = (an[i] + bn[i])/2;
49          if((i+2)==N){
50            if(evalf(f, an[i]) > evalf(f,bn[i]) || evalf(f, an[i]) > evalf(f,mdpt)){
51              fkeep = evalf(f,an[i]);
52              xkeep = an[i];
53            }else{
54              if(evalf(f,bn[i]) > evalf(f,mdpt)){
55                fkeep = evalf(f,bn[i]);
56                xkeep = evalf(f,bn[i]);
57              }else{
58                fkeep =  evalf(f,mdpt);
59                xkeep = mdpt;
60              }
61            }
62          }
63      }
64
65
66    }
```

le code qui nous intéresse ici est:

```
1         x1 = a+(fib(N-2)/fib(N))*(b-a);
2         x2 = a+(fib(N-1)/fib(N))*(b-a);
```

on génère ainsi les x a tester. lors de la phase d'itération les x sont généré a partir des nouvelles bornes de l'intervalle

```
1           x1n[i+1] = x2n[i];
2           x2n[i+1] = an[i+1]+(fib(N-i-1)/fib(N-i))*(bn[i+1]-an[i+1]);
```

```
1           x2n[i+1] = x1n[i];
2           x1n[i+1] = an[i+1]+(fib(N-i-2)/fib(N-i))*(bn[i+1]-an[i+1]);
```

ces deux portions de code permettent donc de générer les valeurs a tester. l'optimiseur est constituer d'une condition qui nous permet de réduire l'intervalle de valeur a chaque itération.

```
1           if (fx1[i]<=fx2[i])
```

CONCLUSION

La méthode de Fibonacci est une technique d'optimisation sans contraintes directe appliqué au fonction uni-modale. dû au fait que le découpage de l'intervalle en sous intervalle pour rechercher la solution dépend des nombres de la suite de Fibonacci cette méthode converge très rapidement vers la solution et fait d'elle la méthode la plus efficace partie les technique de son genre, néanmoins un inconvénient de cette méthode est qu'il faut fixer au préalable un nombre **N** d'itération donc si **N** est mal fixé (N trop petit) on pourrait en ressortir avec une solution qui n'est pas optimale. Pour palier a ce problème une autre méthode dite de section dorée base sur celle de Fibonacci a été mise au point. La méthode de Fibonacci a un principe des plus simple et son algorithme est assez simple a implémenter, ceci fait de cette méthode une technique assez peu pratique devant des problèmes réels de l'informatique qui ont parfois beaucoup de contraintes et plusieurs solutions optimales.