UNIVERSITÉ DE NANTES

L2 X22I050

Algorithmique numérique

Projet

2020_2021

ATALLA Salim

L2 – Informatique_Groupe_485k

Algorithme de Remez :

Le but de ce projet est de déterminer l'approximation du polynôme donnée par la méthode de Remez en plusieurs étapes :

1. On choisit des points de départ dans un intervalle donné.
2. Après avoir choisir ces points on considère qu'entre chaque deux points de cet intervalle on a un sous-intervalle et on détermine le maximum de ces intervalles pour la fonction f donnée.
3. Ensuite, on créer la matrice A de la forme (E1), et le vecteur de droite.
4. On aura des matrices de la forme (A z = b), où A c'est la matrice de (E1), et b le vecteur de droite, ensuite, on va déterminer le vecteur z qui représente les coefficients du polynôme !
5. En fin, pour trouver z on utilise la méthode de pivot de gauss pour résoudre le système linéaire pour obtenir les coefficients qui se déterminent ce polynôme.

Tâches 1 et 8 :

<u>Rôle</u> : Déterminer l'approximation d'un polynôme par la méthode de Remez.

Fonction *remez* (pointeur vers une fonction <u>d</u> f,

réel <u>d</u> a, réel <u>d</u> b, entier <u>d</u> n) : pointeur vers une tableau de réels

// pré : n >= 3 et b > a

<u>Variables</u> : `pointeur vers une tableau de réels : x_i, z_i, A, C, B, r`

<u>Début</u> :

```
x_i ← allocation(tableau de réels[1:n])
z_i ← allocation(tableau de réels[1:n-1])
A   ← allocation(tableau de réels[1:n*n])
C   ← allocation(tableau de réels[1:n])
B   ← allocation(tableau de réels[1:n])

x_i ← Div_Intervalle (a, b, n)
z_i ← Z_i_Max (f, x_i, n)
A   ← construire_Matrice_A (n, n, x_i)
C   ← construire_Vecteur_C (f, z_i, n)
B   ← gauss (A, C, n)

désallocation(x_i)
désallocation(z_i)
désallocation(A)
désallocation(C)

retourner B
```

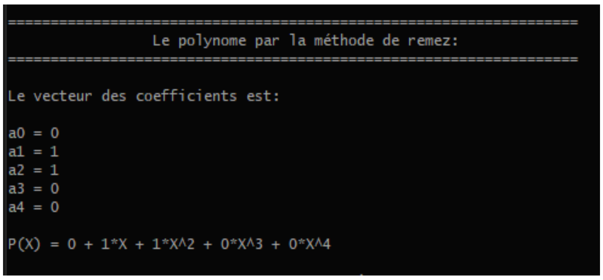
<u>Fin</u>

<u>Jeux de tests :</u>

Pour une fonction donnée : $f1(x) = x^2 + x$

pour un intervalle : [2, 6] et pour n = 5

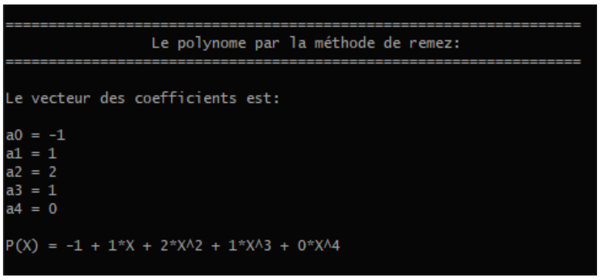
On aura le résultat ci-contre :



Pour une fonction donnée :

$f2(x) = x^3 + 2x^2 + x - 1$

On aura le résultat ci-contre :



On observe qu'on a bien le vecteur des coefficients qui se détermine le polynôme P(x) pour chaque fonction f1 et f2 .

Tâches 2 :

Rôle : construir un tableau de n points équidistants sur une intervalle [a, b] donnée.

Fonction *Div_Intervalle* (réel d a, réel d b, entier d n) : pointeur vers une tableau de réels

// pré : n >= 3 et b > a

Variables : `pointeur vers une tableau de réels : x_i,  entier : i`

Début :

```
x_i ← allocation(tableau de réels[1:n])

Pour i de 1 à n  faire :
      x_i[i] ← a+((b-a)div(n))*i
finPour

retourner x_i
```

Fin

Jeux de tests :

Pour un intervalle donné : [2, 6] et pour n = 5

On aura le résultat ci-contre :

```
--------------------------------------------------
        Les points de depart: (X0, X1, ..., Xn)
--------------------------------------------------

2, 3, 4, 5, 6
```

Supposons d'autres intervalles : [-2, 2] ; [0, 1]

On aura les résultats suivants :

[-2, 2] , et pour n = 5

```
--------------------------------------------------
        Les points de depart: (X0, X1, ..., Xn)
--------------------------------------------------

-2, -1, 0, 1, 2
```

[0, 1] , et pour n = 5

```
--------------------------------------------------
        Les points de depart: (X0, X1, ..., Xn)
--------------------------------------------------

0, 0.25, 0.5, 0.75, 1
```

On a le premier point c'est bien le point a de l'intervalle, et le dernier point est bien le point b de l'intervalle, et le nombre de point est 5 puisque n = 5, et les différences entre les points sont égaux.

Tâches 3 :

Rôle : Chercher le point max sur l'axe des abscisses.

Fonction *X_i_Max* (pointeur vers une fonction d f, réel d a, réel d b) : réel

// pré : b > a

Variables : `réel max, x, eps    // eps est une constante`
`// c'est une variable globale.`

Début :

```
x ← a
max ← x

Tantque (x <= b)    faire:
       Si (f(x) > f(max))  alors
              max <- x
       finSi
       x ← x + eps
finTanque

retourner max
```

Fin

//---

Rôle : Chercher le point max sur l'axe des ordonnées.

Fonction *Fx_i_Max* (pointeur vers une fonction d f, réel d a, réel d b) : réel

// pré : b > a

Début :

```
retourner f(X_i_Max(f, a, b))
```

Fin

Jeux de tests :

Pour une fonction donnée :

$f(x) = x^2 + x$ et pour un intervalle [2, 6]

On aura le résultat ci-contre :

```
------------------------------------------------
      Le point max sur l'intervalle [a, b]
------------------------------------------------

x = 6
f(x) = 42
```

C'est une fonction polynomiale où le maximum sur c'est intervalle c'est lorsque x=6, et f(x)=42, et donc à partir de ce point-là la fonction sera strictement croissante.

Tâches 4 :

Rôle : construir un tableau de n-1 points d'extremes sur des sous-intervalles [x_i, x_i+1] données.

Fonction *Z_i_Max* (pointeur vers une fonction d f,

pointeur vers une tableau de réels d x_i,

entier d n) : pointeur vers une tableau de réels

// pré : n >= 3

Variables : `pointeur vers une tableau de réels : z_i, entier : i`

Début :

```
z_i ← allocation(tableau de réels[1:n-1] )

Pour i de 1 à n-1    faire :
       z_i[i] ← Fx_i_Max(f, x_i[i], x_i[i+1])
finPour

retourner z_i
```

Fin

Jeux de tests :

Pour une fonction donnée :

$f(x) = x^2 + x$, et pour un tableau x_i [n] = {2, 3, 4, 5, 6}, où n = 5

On aura le résultat ci-contre :

```
--------------------------------------------
       Les points: (Z0, Z1, ..., Zn-1)
--------------------------------------------

12, 20, 30, 42
```

Cela signifie que pour cette fonction sur les intervalles : [2, 3] ; [3, 4] ; [4, 5] ; [5, 6]

Les points maximaux sont 12, 20, 30, 42 dans l'ordre.

Tâches 5 :

Rôle : créer une matrice A de n lignes et m colonnes.

Fonction *creer_Matrice_Vide* (entier d n, entier d m) : pointeur vers une tableau de réels

// pré : n > 0 et m > 0

Variables : `pointeur vers une tableau de réels : M`

Début :

```
M ← allocation(tableau de réels[1:n*m])

retourner M
```

Fin

//--

Rôle : créer une matrice A de (E1), de n lignes et m colonnes.

Fonction *construire_Matrice_A* (entier d n, entier d m,

pointeur vers une tableau de réels d x_i) : pointeur vers une tableau de réels

Variables : `pointeur vers une tableau de réels : A,  entier i, j`

Début :

```
A ← créer_Matrice_Vide(n, m)

Pour i de 1 à n  faire :
      Pour j de 1 à m  faire :

            Si (j mod(m-1) = 0 et j mod m != 0)  alors
                  Si (i mod 2 = 0)    alors
                        A [m*i+j] ← 1
                  Sinon
                        A [m*i+j] ← -1
                  finSi
            Sinon
                  A [m*i+j] <- (x_i[i]^j)
            finSi
      finPour
finPour

retourner A
```

Fin

Jeux de tests :

Pour n = m = 5,   et pour un tableau x_i [n] = {2, 3, 4, 5, 6}

On aura le résultat ci-contre qui représente la matrice A de la forme (E1) :

```
------------------------------------
          La matrice A
------------------------------------

1  2  4  8  1

1  3  9  27  -1

1  4  16  64  1

1  5  25  125  -1

1  6  36  216  1
```

Tâches 6 :

<u>Rôle</u> : créer un vecteur V vide de taille n.

Fonction *creer_Vecteur_Vide* (entier <u>d</u> n) : pointeur vers une tableau de réels

// pré : n > 0

<u>Variables</u> : `pointeur vers une tableau de réels : V`

<u>Début</u> :

```
V ← allocation(tableau de réels[1:n])

retourner V
```

<u>Fin</u>

//--

<u>Rôle</u> : créer un vecteur C à partir d'un tableau de points z_i de taille n.

Fonction *construire_Vecteur_C* (pointeur vers une fonction <u>d</u> f,
pointeur vers un tableau de réels <u>d</u> x_i,
entier <u>d</u> n) : pointeur vers une tableau de réels

// pré : n > 0

<u>Variables</u> : `pointeur vers une tableau de réels : C,  entier i`

<u>Début</u> :

```
C ← créer_Vecteur_Vide(n)

Pour i de 1 à n  faire :
       C[i] ← f(x_i[i])
finPour

retourner C
```

<u>Fin</u>

Jeux de tests :

Pour une fonction donnée : $f(x) = x^2 + x$, et pour un tableau x_i [n] = {2, 3, 4, 5, 6}, où n = 5

On aura le résultat suivant qui représente le vecteur C de la forme (E1) :

Où C [i] ← f (x_i [i])

```
------------------------------------
      Le vecteur C
------------------------------------

6
12
20
30
42
```

Tâches 7 :

<u>Rôle</u> : résoudre un système linéaire par la méthode de pivot de gauss.

Fonction gauss (pointeur vers un tableau de réels <u>d</u> A, pointeur vers un tableau de réels <u>d</u> V

entier <u>d</u> n) : pointeur vers un tableau de réels

// pré : n > 0

<u>Variables</u> : `pointeur vers une tableau de réels : z,`

`entier i, j, k,      réel pivot, p, sol`

<u>Début</u> :

```
// pivotation

Pour i de 1 à n-1 faire :
      pivot ← A[n*i+i]

      Pour j de i+1 à n-1 faire :
            p ← -(A[n*j+i]) div pivot

            Pour k de 1 à n faire :
                  A[n*j+k] ← A[n*j+k] + p* A[n*i+k]
            finPour
            V[j] ← V[j] + p*V[i]
      finPour
finPour


// Résolution
Pour i de n à 1 (i ← i-1) faire :
      sol ← 0

      pour j de i+1 à n-1 faire :
            sol ← sol + A[n*i+j] * z[j]
            z[i] ← (V[i]-sol) / A[n*i+i]
      finPour
finPour

retourner z
```

<u>Fin</u>

Jeux de tests :

Pour une matrice A et un vecteur V données (dans les tâches 5 et 6), et pour n = 5

On aura le résultat suivant :

En utilisant la méthode de gauss on a pu résoudre le système linéaire.

```
---------------------------------------
  La matrice A apres la pivotation
---------------------------------------

1  2  4  8  1

0  1  5  19  -2

0  0  2  18  4

0  0  0  6  -8

0  0  0  0  16
```

```
---------------------------------------
      Les coefficients:
---------------------------------------

a0 = 0
a1 = 1
a2 = 1
a3 = 0
a4 = 0
```