Chapitre 1

Recherche de racines

Chapitre 2

Optimisation et Interpolation

2.1 Contexte de notre étude

Il est très fréquent d'avoir à chercher l'extremum d'une fonction, minimum ou maximum. Les ingénieurs doivent en permanence concevoir des produits qui réalisent des tâches de façon efficace. Ils sont contraints par les lois derrière ces produits et aussi doivent-ils en général minimiser les coûts aussi ! Ainsi sont ils en permanence confrontés à des problèmes d'optimisation !

La recherche d'une racine, comme vu au chapitre précédent, et la recherche dans ce chapitre d'un extremum (optimisation) sont des méthodes très liées car toutes deux impliquent la recherche d'un point particulier. La racine d'une racine d'une fonction f correspond à $f(x) = 0$ tandis que la recherche de l'extremum correspond à $f'(x) = 0$. Par ailleurs, le signe de la dérivée seconde $f''(x)$ indique s'il s'agit d'un maximum ($f''(x) < 0$) ou d'un minimum ($f''(x) > 0$). Ainsi, ce lien entre racine et extremum peut nous permettre de trouver une stratégie pour trouver ces derniers.

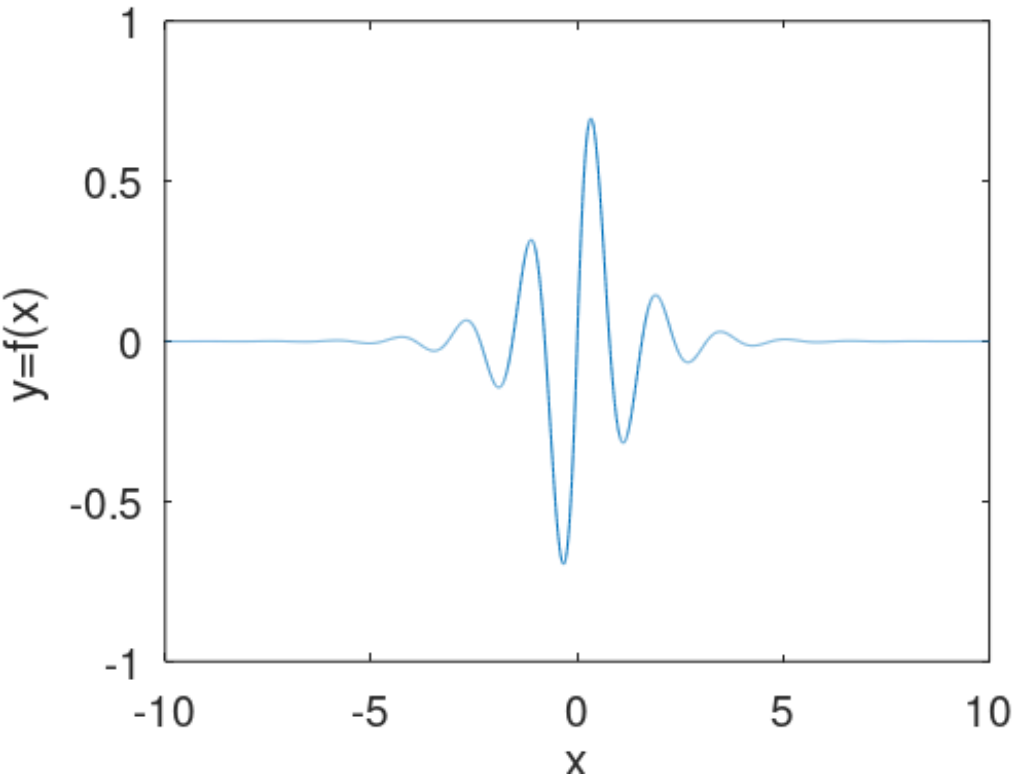
Des méthodes analytiques (calcul explicite de la dérivée, multiplicateurs de Lagrange par exemple aussi pour des problèmes avec contraintes etc...) existent mais nous n'étudieront ici que des problèmes que l'on ne peut résoudre analytiquement, et qui nécessitent des calculateurs électroniques.

Notre étude ne portera pas sur les systèmes contraints,mais sans contrainte. On étudie d'abord le cas d'une fonction à une variable, puis à plusieurs variables.

2.2 Optimisation non contrainte à une dimension

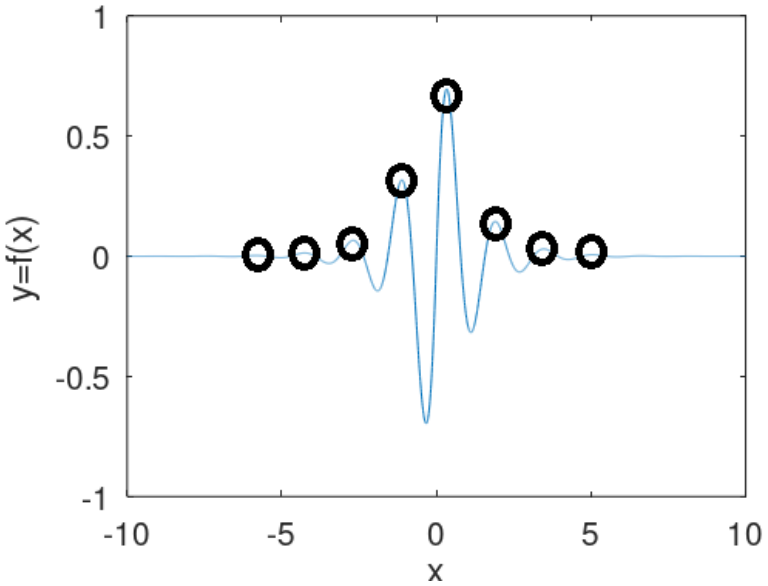
Nous allons décrire ici des méthodes qui permettent de trouver les extremum d'une fonction réelle $f(x)$ de la variable réelle à une dimension x.

Considérons par exemple la fonction f représentée ci après :

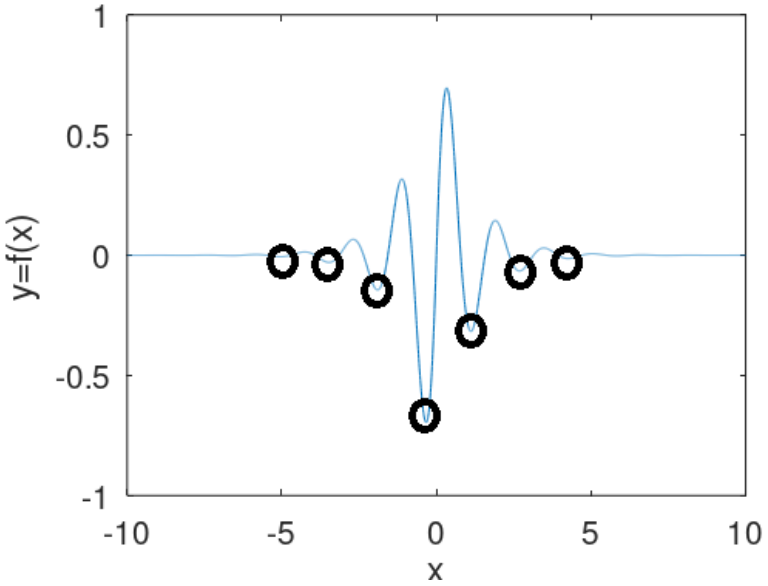


On peut observer que cette fonction possède :

— des maxima locaux (on a représenté les plus visibles) et un maximum global (le plus haut)



— des minima locaux (on a représenté les plus visibles) et un minimum global (le plus bas)



La plupart du temps on recherchera le minimum ou le maximum global donc il faudra faire attention à ne pas se satisfaire que d'un extremum local ! Tout d'abord, pour éviter cette erreur, tracer l'allure de la courbe représentative de la fonction peut être très utile. Ensuite, chaque méthode employée devra être à postériori analysée pour déterminer si l'on a obtenu effectivement un extremum global, ou seulement un extremum local...

2.2.1 Méthode de Newton

a - Justification théorique de la méthode

Rappelons-nous la méthode de Newton-Raphson, qui permet de déterminer la racine d'une fonction $f(x)$:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

On peut évidemment supposer, comme l'extremum vérifie la relation $g(x) = f'(x) = 0$, appliquer cette même méthode de Newton à la fonction $g(x) = f'(x)$. La suite définie par récurrence selon :

$$x_{n+1} = x_n - \frac{g(x_n)}{g'(x_n)}$$

c'est-à-dire :

$$\boxed{x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}}$$

convergera ainsi vers un extremum de la fonction $f(x)$!

Il est bon de noter cependant que :

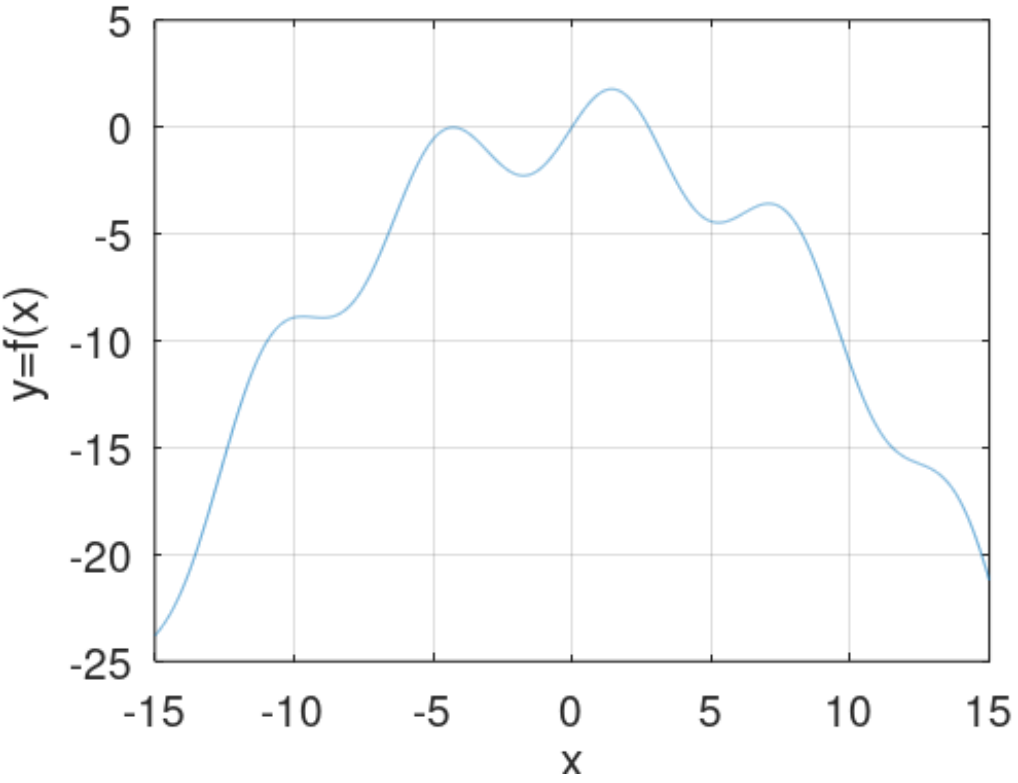
— Cette méthode peut tout de même diverger, si le point de départ x_0 est trop éloigné de l'extremum.
— Il sera bon de calculer le signe de $f''(x)$ afin de vérifier s'il s'agit d'un minimum ($f''(x) > 0$) ou d'un maximum ($f''(x) < 0$) de $f(x)$.

b - Exemple

Soit la fonction $f(x)$ définie ci-dessous ; il s'agit de trouver un extremum de cette fonction en partant de $x_0 = 2,5$:

$$f(x) = 2sin(x) - \frac{x^2}{10}$$

Traçons la fonction $f(x)$:



On observe la présence de plusieurs minima et maxima. En partant de la valeur $x_0 = 2,5$, l'algorithme de Newton devrait converger vers l'extremum le plus proche, qui semble être un maximum de valeur approchée située entre 1 et 2. Calculons les dérivées première et seconde de $f(x)$:

$$f'(x) = 2cos(x) - \frac{x}{5}$$

$$f''(x) = -2sin(x) - \frac{1}{5}$$

La méthode de Newton de recherche d'optimum s'écrit ainsi :

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$

$$x_{n+1} = x_n - \frac{2cos(x_n) - \frac{x_n}{5}}{-2sin(x_n) - \frac{1}{5}}$$

En utilisant la technique de la touche "ANS" ou "REP", on évite alors dans notre cas présent décrire un programme en C++, et cela nous permet d'écrire à la calculatrice, après

avoir entré d'abord et validé la valeur $2,5$ (attention à bien se placer sur la calculatrice en mode RADIAN) :

$$ANS - \frac{2 * cos(ANS) - 0.2 * ANS}{-2 * sin(ANS) - 0.2}$$

On obtient le tableau de valeurs suivantes :

n	0	1	2	3	4	5	6
x_n	2,5	0,995081551	1,46901075	1,42764232	1,42755178	1,42755178	1,42755178

On constate que la méthode de Newton nous a permis d'obtenir un extremum local, qui se trouve être un maximum local car $f''(1,42755178) = -2,17951606 < 0$, ce qui confirme notre intuition initiale. De plus, grâce à la courbe représentative, on peut déclarer que ce maximum est aussi le maximum global de la fonction !

c - Conclusion

- Cette méthode fonctionne correctement mais il est bon d'avoir l'allure de la représentation graphique de la fonction étudiée, et de calculer les dérivées secondes des points obtenus, afin de faire correspondre la valeur obtenue à un minimum ou un maximum.
- Cette méthode n'est pas applicable si l'on ne connait pas les expressions analytiques des dérivées première et seconde de $f(x)$; dans ce dernier cas on peut établir une méthode équivalente à la méthode de la sécante vue lors du chapitre précédent.
- Il est évident que cette méthode peut diverger, selon la nature de la fonction et la qualité de la valeur initiale ; on ne l'applique donc en général que si l'on est "proche" de l'extremum.

d - Généralisations

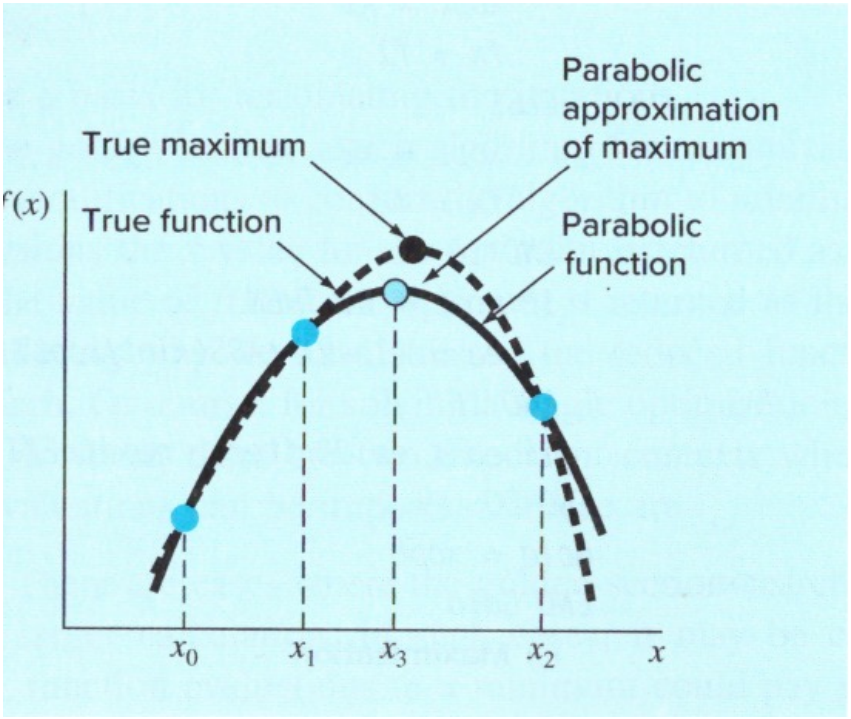
Comme il suffit de résoudre l'équation $f'(x) = 0$ pour obtenir un extremum, il est évident que d'autres méthodes que la méthode de Newton peuvent être employées : méthode de dichotomie, méthode de la sécante, méthode de Halley etc... Nous en étudierons quelques unes lors du TD correspondant à ce chapitre.

2.2.2 Méthode d'interpolation parabolique

a - Justification théorique de la méthode

Cette méthode profite du fait que suffisamment proche d'un extremum, on peut approximer la fonction étudiée $f(x)$ par un polynôme de second degré, qui fournit souvent une très bonne approximation de f.

Tout comme entre deux points d'une courbe représentative d'une fonction, une droite et une seule droite peut passer et approximer cette fonction ; on sait qu'un seul polynôme de second degré passe entre trois points. Si deux point entourent un troisième point qui est proche de l'extremum, on peut donc faire passer une parabole qui approxime la courbe représentative de f autour de l'extremum. Ensuite, il nous reste à dériver le polynôme de second degré correspondant, résoudre l'équation correspondante égale à zéro, et on peut donc approximer la valeur de l'extremum !



Après quelques calculs algébriques, on obtient :

$$x_3 = \frac{f(x_0)(x_1^2 - x_2^2) + f(x_1)(x_2^2 - x_0^2) + f(x_2)(x_0^2 - x_1^2)}{2f(x_0)(x_1 - x_2) + 2f(x_1)(x_2 - x_0) + 2f(x_2)(x_0 - x_1)}$$

avec x_0, x_1 et x_2 sont les trois valeurs essais de départ, et x_3 est la valeur de x qui correspond au maximum de la parabole qui passe par les trois points précédents, c'est donc notre estimation de l'extremum. A l'itération suivante, on peut ensuite alors effectuer la permutation :

$$x_0 \leftarrow x_{1precedent}$$

$$x_1 \leftarrow x_{2precedent}$$

$$x_2 \leftarrow x_{3precedent}$$

et recommencer d'appliquer la formule précédente !

b - Exemple

Soit toujours la même la fonction $f(x)$ définie ci-dessous ; il s'agit de trouver un extremum de cette fonction en partant de $x_0 = 0$, $x_1 = 1$ et $x_3 = 4$ (une vue de la courbe représentative justifie ces choix approximatifs) :

$$f(x) = 2sin(x) - \frac{x^2}{10}$$

Donnons un listing codé sur Arduino DUE :

```
1 double x0(0.0),x1(1.0),x2(4.0),x3(0.0);
2 long i(0),i_max(50);
3 double eps(1.e-15),dx(1.0);
4
5 double f(double);
```

```
6
7  void setup() {
8    // put your setup code here, to run once:
9    Serial.begin(9600);
10   delay(5000);
11   do{
12     i++;
13     x3=(f(x0)*(x1*x1-x2*x2)+f(x1)*(x2*x2-x0*x0)+f(x2)*(x0*x0-x1
         *x1))/(2.*f(x0)*(x1-x2)+2.*f(x1)*(x2-x0)+2.*f(x2)*(x0-x1
         ));
14     dx=x2-x3;
15     x0=x1;
16     x1=x2;
17     x2=x3;
18     Serial.println(x3,19);
19   }while((i<i_max)&&(fabs(dx)>eps));
20 }
21
22 void loop() {
23   // put your main code here, to run repeatedly:
24
25 }
26
27 double f(double x)
28 {
29   return 2*sin(x)-0.1*x*x;
30 }
```

On obtient le résultat suivant (fin du cycle d'affichage) :

```
1.4275517078833798834
1.4275517594928935238
1.4275810313404566187
1.4277099184737906512
1.4275517158296402442
1.4275517619971318517
1.4275256222547582751
1.4277791233864909337
1.4275518299049805293
1.4275517855726935323
1.4275787187839306113
1.4274444847080931531
1.4275517524120584056
1.4275517123558378429
1.4275225395454301136
1.4276318668475447282
1.4275517771889727924
1.4275516741324947744
1.4275675472548945421
1.4275901157481070136
1.4275520536424850526
```

La précision toute relative est due au choix variable qu'il faut faire lors de la permutation précédente entre les différents x_i, cependant la valeur est obtenue avec une précision de 5 décimales juste ce qui est honorable !

c - Conclusion

— La convergence peut être très variable, aléatoire ou très longue.
— Cette méthode peut être améliorée par des tests de convergence plus précis et une meilleure gestion des permutations entre les différents x_i.
— Cette méthode fournit cependant une bonne approximation du résultat recherché et peut permettre d'initialiser une méthode plus précise ensuite.

Nous verrons en TD un autre exemple d'application.

2.3 Optimisation non contrainte à plusieurs dimensions

cf Diaporama

2.4 Interpolation

Il est utile de savoir interpoler, c'est-à-dire trouver une fonction qui représente une courbe expérimentale (des datas) de façon plus ou moins exacte. On suppose connues m données expérimentales $(x_i, f(x_i))$ pour $i = 1, ..., m$. On cherche donc une courbe $y(x)$ qui représente cette courbe expérimentale $f(x)$ de façon approchée. On va utiliser l'approximation des moindres carrés pour trouver $y(x)$ telle que les écarts entre f et y soient minimaux.

On va écrire $y(x)$ sous la forme :

$$y(x) = \sum_{k=0}^{n} c_k \varphi_k(x)$$

avec $\varphi_k(x)$: fonction de base, et les coefficients c_k sont à déterminer. Par exemple, une base possible connue est $(1, x, x^2, ..., x^n)$. Cette dernière base est en fait très peu utilisée car elle n'est pas orthogonale et les vecteurs sont de normes différentes. On préfèrera utiliser des fonction de base orthogonales.

Avant tout, avant de trouver une bonne base et de savoir comment calculer les coefficients c_k afin que $y(x)$ "colle" au mieux à $f(x)$, il faudra pouvoir évaluer simplement la somme vue ci-dessus $\sum_{k=0}^{n} c_k \varphi_k(x)$. Nous allons utiliser l'algorithme de Clenshaw pour cela.

2.4.1 Algorithme de Clenshaw

a - Justification théorique de la méthode

Soit à calculer de la façon efficace et simple la somme :

$$y(x) = \sum_{k=0}^{n} c_k \varphi_k(x)$$

On suppose connue une relation du type :

$$\varphi_{k+1} + a_k \varphi_k + b_k \varphi_{k-1} = 0$$

pour $k = 0, 1, 2, ...$ et $a_k(x)$ et $b_k(x)$ connus.

(Par exemple, on connait la relation sur les polynômes de Chebychev :

$$T_{k+1}(x) + T_{k-1}(x) = 2xT_k(x)$$

avec $x = cos(\theta)$. On a donc ici $a_k(x) = -2x$ et $b_k(x) = 1$.)

On lui associe alors une suite u_k telle que :

$$u_k + a_k u_{k+1} + b_{k+1} u_{k+2} = c_k$$

pour $k = 0, 1, 2, ..., n$ et $u_{n+1} = u_{n+1} = 0$.

On en déduit que :

$$y(x) = \sum_{k=0}^{n} c_k \varphi_k(x) = \sum_{k=0}^{n} (u_k + a_k u_{k+1} + b_{k+1} u_{k+2}) \varphi_k(x)$$

$$y(x) = u_0\varphi_0 + u_1\varphi_1 + \sum_{k=2}^{n} u_k \varphi_k(x)$$

$$+a_0 u_1 \varphi_0(x) + \sum_{k=1}^{n} a_k u_{k+1} \varphi_k(x) + \sum_{k=0}^{n} b_{k+1} u_{k+2} \varphi_k(x)$$

avec, comme $u_{n+1} = 0$:

$$\sum_{k=1}^{n} a_k u_{k+1} \varphi_k(x) = \sum_{k=2}^{n+1} a_{k-1} u_k \varphi_{k-1}(x) = \sum_{k=2}^{n} a_{k-1} u_k \varphi_{k-1}(x)$$

et avec, comme $u_{n+1} = u_{n+2} = 0$:

$$\sum_{k=0}^{n} b_{k+1} u_{k+2} \varphi_k(x) = \sum_{k=2}^{n+2} b_{k-1} u_k \varphi_{k-2}(x) = \sum_{k=2}^{n} b_{k-1} u_k \varphi_{k-2}(x)$$

Au final :

$$y(x) = u_0\varphi_0 + u_1\varphi_1 + a_0 u_1 \varphi_0(x) + \sum_{k=2}^{n} u_k \varphi_k(x) + a_{k-1} u_k \varphi_{k-1}(x) + b_{k-1} u_k \varphi_{k-2}(x)$$

$$y(x) = u_0\varphi_0 + u_1\varphi_1 + a_0 u_1 \varphi_0(x) + \sum_{k=2}^{n} (\varphi_k(x) + a_{k-1}\varphi_{k-1}(x) + b_{k-1}\varphi_{k-2}(x)) u_k$$

$$y(x) = u_0\varphi_0 + u_1\varphi_1 + a_0 u_1 \varphi_0(x)$$
$$y(x) = \varphi_0 u_0 + (\varphi_1 + a_0\varphi_0(x)) u_1$$

C'est une énorme simplification : connaissant uniquement $\varphi_0(x)$, $\varphi_1(x)$ et u_0, u_1, on connaitra donc complètement la somme $y(x)$!

En résumé : Pour évaluer la somme $y(x) = \sum_{k=0}^{n} c_k \varphi_k(x)$, avec la base des $(\varphi_k(x))_k$ vérifiant une relation de récurrence du type : $\varphi_{k+1} + a_k\varphi_k + b_k\varphi_{k-1} = 0$ pour $k = 0, 1, 2, ...$ et $a_k(x)$ et $b_k(x)$ connus, il suffit de créer la suite u_k vérifiant : $u_k + a_k u_{k+1} + b_{k+1} u_{k+2} = c_k$ pour $k = 0, 1, 2, ..., n$ et $u_{n+1} = u_{n+1} = 0$. On a alors simplement : $y(x) = \varphi_0 u_0 + (\varphi_1 + a_0\varphi_0(x)) u_1$.

Remarque : Si de plus on a $b_0 = 0$ ou $\varphi_{-1} = 0$, alors comme pour $k = 0$ on a la relation $\varphi_1 + a_0\varphi_0 + b_0\varphi_{-1} = 0$, on en déduit que $\varphi_1 + a_0\varphi_0 = 0$, ce qui implique que :

$$y(x) = \varphi_0 u_0$$

Si finalement, on a en plus $\varphi_0 = 1$, ce qui est souvent le cas (cela est vérifié pour les polynômes de Chebychev, Legendre ... etc ...), alors :

$$y(x) = u_0$$

b - Algorithme

On en déduit, en se plaçant dans le cas favorable maximum précédent, l'algorithme suivant :

Algorithm 1: Algorithme de Clenshaw

Data: $a_k(x)$ et $b_k(x)$ et c_k
$u_{n+1} = u_{n+2} = 0$ et $u_n = c_n$
for *k=n-1,...,0* **do**
  $u_k = c_k - a_k u_{k+1} - b_{k+1} u_{k+2}$
end
Result: $y = u_0$: valeur de $y(x)$

c - Exemple 1 : calcul d'un polynôme - schéma de Hörner

Soit la base $\varphi_k(x) = x^k$. On a bien $\varphi_0 = 1$. De plus, comme $\varphi_{k+1}(x) = x\varphi_k(x)$, on a la relation voulue :

$$\varphi_{k+1}(x) - x\varphi_k(x) = 0$$

donc $a_k(x) = -x$ et $b_k = 0$. On est donc dans la situation où $b_0 = 0$ (et $\varphi_0 = 1$) donc $y(x) = u_0$! On en déduit l'algorithme, dit de Hörner, de calcul de :

$$y(x) = \sum_{k=0}^{n} c_k x^k$$

Algorithm 2: Algorithme de Hörner

Data: x et c_k
$u_{n+1} = u_{n+2} = 0$ et $u_n = c_n$
:
for *k=n-1,...,0* **do**
  $u_k = c_k + x u_{k+1}$
end
Result: $y = u_0$: valeur de $y(x)$

On peut écrire un code qui calcule un exemple de polynôme : le polynôme de Chebychev de première espèce, d'ordre $n = 9$, par exemple pour 50 valeurs de x :

$$P(x) = T_9(x) = 256x^9 - 576x^7 + 432x^5 - 120x^3 + 9x$$

soit de façon directe (et on calcule un temps d'exécution t_1), soit grâce à l'algorithme de Hörner (temps d'exécution t_2).

```
1  double x(.0),y(0.);
2  unsigned long temps1(0.),temps2(0.);
3
4  double f(double);
5
6  void setup() {
7    Serial.begin(9600);
8
9    delay(5000);
10   temps1=micros();
```

```
11   for(int i(0);i<=50;i++)
12   {
13   x+=0.01;
14   y=f(x);
15   }
16   temps2=micros();
17   Serial.print("duree␣(microsecondes)=");
18   Serial.println(temps2-temps1);
19
20 }
21
22 void loop() {
23 }
24
25 double f(double x)
26 {
27   return 256.*pow(x,9)-576.*pow(x,7)+432.*pow(x,5)-120.*pow(x
       ,3)+9.*x;
28 }
```

dont l'exécution fournit :

```
duree (microsecondes)=35544
```

```
1 double x(0.),y(0.);
2 double c[10]={0.,9.,0.,-120.,0.,432.,0.,-576.,0,256.};
3 long n(9);
4 unsigned long temps1(0.),temps2(0.);
5
6 void setup() {
7   Serial.begin(9600);
8
9   delay(5000);
10   temps1=micros();
11   for(int i(0);i<=50;i++)
12   {
13   x+=0.01;
14   y=0.;
15   for(int k(n);k>=0;k--)
16   {
17     y+=c[k]+x*y;
18     }
19
20   }
21   temps2=micros();
22   Serial.print("duree␣(microsecondes)=");
23   Serial.println(temps2-temps1);
24
```

```
25 }
26
27 void loop() {
28 }
```

dont l'exécution fournit :

```
duree (microsecondes)=2396
```

On observe bien évidemment un temps très inférieur pour la méthode de Hörner par rapport à un calcul direct !

Exemple 2 : cas où la base $\varphi_k(x)$ est la base des polynômes de Chebychev de première espèce $T_k(x)$

Pour ces polynômes, on $T_0 = 1$ et $T_1 = x$ et la relation de récurrence :

$$T_{k+1}(x) - 2xT_k(x) + T_{k-1}(x) = 0$$

On en déduit : $a_k(x) = -2x$ et $b_k(x) = 1$. On pose aussi souvent $x = cos(\theta)$. On rappelle qu'on essaie de calculer :

$$y(x) = \sum_{k=0}^{n} c_k T_k(x)$$

On applique l'algorithme de Clenshaw (cf TD).

2.4.2 Algorithme de Forsythe : méthode des moindres carrés

a - Justification théorique de la méthode

On veut calculer $f(x)$ pour x compris entre x_1 et x_m mais autre que les x_i. On va alors construire :

$$y(x) = \sum_{k=0}^{n} c_k \varphi_k(x) \simeq f(x)$$

Notons le résidu au point x :

$$r(x) = f(x) - y(x)$$

On va minimiser le carrés des résidus :

$$S = \sum_{i=1}^{m} w(x_i) r^2(x_i)$$

S doit être minimum, avec $r(x_i) = f(x_i) - y(x_i)$ est le résidu au point i, et $w_(x_i)$: une fonction de poids. En effet, certains points peuvent être plus importants que d'autres, on leur attribue un poids. En ces points, le résidu doit être plus faible encore, d'où une meilleure

approximation, car si w_i augmente, il faut que r_i diminue afin que S soit minimale. Par exemple, on peut prendre :

$$w_i = \frac{1}{\Delta_i}$$

avec Δ_i : l'incertitude affectée à la mesure en x_i. Plus l'incertitude est faible, plus le point est fiable et donc plus il faut affecter un poids important à ce point... !

On a donc :

$$S = ||r||^2 = \sum_{i=1}^{m} w(x_i) r^2(x_i)$$

$$S = (r, Dr) = r^t.(Dr)$$

(cf. les définitions de produit scalaire...), à condition de prendre :

$$D = \begin{pmatrix} w_1 & 0 & 0 & \cdots & 0 \\ 0 & w_2 & 0 & \cdots & 0 \\ 0 & 0 & w_3 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & w_m \end{pmatrix}$$

et :

$$r = \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_m \end{pmatrix}$$

Dans toute la suite, on notera pour simplifier sans écrire la matrice D, c'est-à-dire que l'on notera par exemple $(r, r) = r^t.(Dr)$.

Si $D = I$ (matrice identité), on retrouve la norme habituelle. On a le produit scalaire discret :

$$(\varphi_k, \varphi_j) = \sum_{i=1}^{m} w(x_i) \varphi_k(x_i) \varphi_j(x_i)$$

On note alors :

$$\gamma_k = (\varphi_k, \varphi_k) = \sum_{i=1}^{m} w(x_i) \varphi_k^2(x_i) = ||\varphi_k||^2 > 0$$

$$||f||^2 = \sum_{i=1}^{m} w(x_i) f^2(x_i)$$

On suppose la base $(\varphi)_k$ orthogonale, c'est-à-dire que :

$$(\varphi_k, \varphi_j) = \delta_{kj} \gamma_k$$

Comme :

$$S = (r, r) = (f - y, f - y) = ||f||^2 - 2(y, f) + ||y||^2$$

$$S = ||f||^2 - 2 \sum_{k=0}^{n} (f, \varphi_k) c_k + (\sum_{k=0}^{n} c_k \varphi_k, \sum_{j=0}^{n} c_j \varphi_j)$$

On pose :

$$b_k = (f, \varphi_k)$$

et on calcule :

$$(\sum_{k=0}^{n} c_k\varphi_k, \sum_{j=0}^{n} c_j\varphi_j) = \sum_{k=0}^{n} c_k(\varphi_k, \sum_{j=0}^{n} c_j\varphi_j) = \sum_{k=0}^{n} c_k[\sum_{j=0}^{n} c_j(\varphi_k, \varphi_j)]$$

$$= \sum_{k=0}^{n} c_k[\sum_{j=0}^{n} c_j\delta_{kj}\gamma_k] = \sum_{k=0}^{n} c_k[c_k\gamma_k] = \sum_{k=0}^{n} c_k^2\gamma_k$$

On en déduit :

$$S = ||f||^2 - 2\sum_{k=0}^{n} b_k c_k + \sum_{k=0}^{n} c_k^2\gamma_k = S(c_0, ..., c_n)$$

N'oublions pas que les c_k sont à chercher, pour rendre S minimum. Il faut donc que :

$$\overrightarrow{grad}S(c_0, ..., c_n) = \vec{0}$$

, ce qui implique que :

$$\frac{\partial S}{\partial c_k} = 0$$

soit encore :

$$-2b_k + 2c_k\gamma_k = 0$$

donc au final, pour $k = 0, ..., n$:

$$c_k = \frac{b_k}{\gamma_k} = \frac{(f, \varphi_k)}{(\varphi_k, \varphi_k)} = \frac{\sum_{i=1}^{m} w_(x_i)f(x_i)\varphi_k(x_i)}{\sum_{i=1}^{m} w_(x_i)\varphi_k^2(x_i)}$$

Le résidu minimum de S vaut :

$$S_m = ||r_m||^2 = ||f||^2 - 2\sum_{k=0}^{n} \gamma_k c_k^2 + \sum_{k=0}^{n} \gamma_k c_k^2 = ||f||^2 - \sum_{k=0}^{n} \gamma_k c_k^2$$

avec bien entendu $||f||^2 > 0$ et fixé, et aussi $\gamma_k > 0$ et bien sûr aussi par définition $S_m > 0$. Donc, S_m diminue au fur et à mesure que l'on ajoute des φ_k. Et :

$$\lim_{n \to +\infty} S_m(n) = 0$$

Le résidu sera donc naturellement nul pour une infinité de termes... ! En pratique, on remarquera que 4 ou 5 termes suffisent amplement pour avoir un faible résidu.

Le problème non résolu jusque là est qu'il faut fabriquer les fonctions orthogonales φ_k !

On peut montrer qu'on aura une relation de récurrence du type :

$$\varphi_{k+1} - x\varphi_k = \lambda_k\varphi_k + \lambda_{k-1}\varphi_{k-1}$$

c'est-à-dire une relation de récurrence à trois termes. On note pour simplifier :

$$\lambda_k = -\alpha_k$$

$$\lambda_{k-1} = -\beta_k$$

d'où la relation :

$$\varphi_{k+1} + (\alpha_k - x)\varphi_k + \beta_k\varphi_{k-1} = 0$$

On montre :

$$\alpha_k = \frac{(x\varphi_k, \varphi_k)}{\gamma_k}$$

pour $k = 0, 1, ..., n$

$$\beta_k = \frac{(x\varphi_{k-1}, \varphi_k)}{\gamma_{k-1}} = \frac{\gamma_k}{\gamma_{k-1}}$$

pour $k = 1, ..., n$ et $\beta_0 = 0$

b - algorithme de Forsythe

Algorithm 3: Algorithme de calcul de la base φ_k

Data: x et α_k et β_k
$\varphi_{-1} = 0$, $\varphi_0 = 1$, $\varphi_1 = x - \alpha_0$
for *k=1,...,n-1* **do**
| $\varphi_{k+1} = (x - \alpha_k)\varphi_k - \beta_k\varphi_{k-1}$
end
Result: $(\varphi_k(x))_k$: base de polynômes orthogonaux recherchée

On verra en TD et TP comment finir cet algorithme de Forsythe, et on fera des applications.