1 Présentation de la méthode de Remez (1896)

1.1 Problématique et Théorèmes généraux

Dans le cas de l'approximation polynomiale usuelle où X est un intervalle compact $[a, b]$ et où l'on cherche des polynômes dans la base des monômes $1, x, \ldots, x^n$, il existe un algorithme dû à Rémez (1896) pour calculer le polynôme de meilleure approximation. On distingue en fait parfois deux algorithmes appelés premier algorithme de Rémez et second algorithme de Rémez ; c'est le second qui est intéressant en pratique et le premier est souvent ignoré de sorte qu'on parle souvent de l'algorithme de Rémez pour désigner le second algorithme. Veidinger a prouvé que cet algorithme a une convergence quadratique sous des conditions raisonnables (qui sont rarement mentionnées).

Puisque la meilleure approximation est unique, on peut définir l'opérateur qui associe à chaque fonction continue son meilleur polynôme approximant $p^\star$ de degré fixé. Il est bien connu que cet opérateur, bien que continu, est non linéaire et des méthodes itératives sont nécessaires pour calculer $p^\star$. L'algorithme de Remez est donc une de ces méthodes. On rappelle deux théorèmes essentiels qui lui sont relatifs. Le premier a été démontré par Borel en 1905 :

Théorème 1.1 *Théorème d'alternance (ou Propriété d'équioscillation).*
Soit g une fonction continue sur $[a, b]$. Un polynôme $p = \sum_{j=0}^{n} a_j x^j$ est de meilleure approximation pour g (parmi les polynômes de degré inférieur ou égal à n) si et seulement si il existe $n + 2$ points $x_0 < x_1 < \ldots < x_{n+1}$ dans $[a, b]$ et $\mu \in \mathbb{R}$ tels que :

$$\left\{ \begin{array}{c} \forall i = 0, \ldots, n+1,\ p(x_i) - g(x_i) = (-1)^i \mu \\ |\mu| = \|p - g\|_\infty \end{array} \right. \tag{1}$$

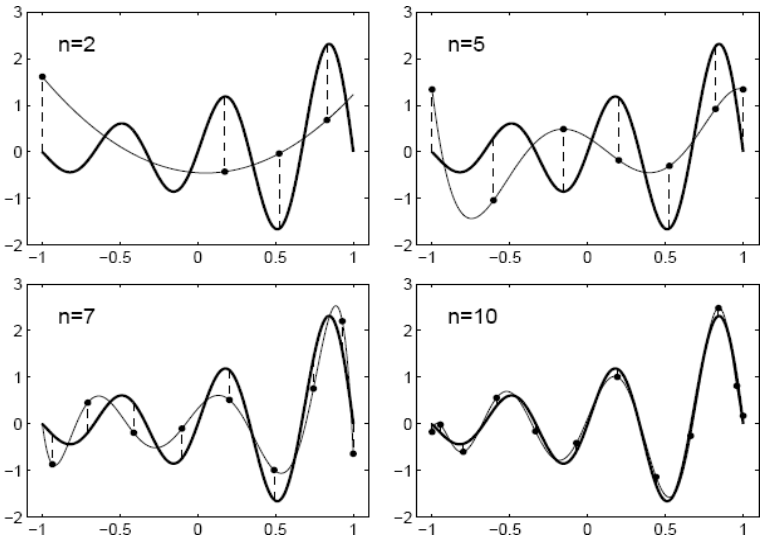

FIGURE 1 – Meilleures approximations polynomiales (traits fins) de degrés 2, 5, 7 et 10 pour $f(x) = \sin(3\pi x)\exp(x)$ (traits gras).

Un ensemble de points $A^\star := \{x_i\}_{i=0}^{n+1}$ qui satisfait (1) est appelé une référence. La figure FIG.1 montre le polynôme $p^\star$ de meilleure approximation pour f définie par $f(x) = \sin(3\pi x)\exp(x)$(traits gras) sur $[-1, 1]$ et les références de 4, 7, 9 et 12 points respectivement où $f - p^\star$ équioscille.

Le Théorème d'alternance affirme qu'un polynôme est optimal si et seulement si son erreur oscille parfaitement entre ses valeurs extrêmes au moins $n + 2$ fois. Le Théorème de La Vallée Poussin, établi en 1910, que nous allons

voir à présent, raffine le Théorème 1.1. Il affirme la chose suivante : soit un polynôme p dont l'erreur oscille au moins $n+2$ fois, mais imparfaitement. Alors, la qualité d'approximation de p est reliée à la qualité des oscillations de sa fonction d'erreur.

Théorème 1.2 *(La Vallée Poussin).*
On se place sous les mêmes hypothèses que pour le théorème précédent. On suppose donné un polynôme p et on suppose qu'il existe $n+2$ points $x_0 < x_1 < \ldots < x_{n+1}$ dans $[a,b]$ et $\epsilon \in \mathbb{R}$ tels que :

$$\forall i = 0, \ldots, n+1,\ p(x_i) - g(x_i) = (-1)^i \varepsilon.$$

On note μ l'erreur optimale. Alors

$$|\varepsilon| \leq \mu \leq \|p - g\|_\infty. \tag{2}$$

Le polynôme p étant donné, on a tout intérêt à choisir les points x_i de telle sorte que ε soit maximal. Ceci fournit une formulation équivalente du Théorème de La Vallée Poussin.

Théorème 1.3 *(La Vallée Poussin, autre formulation).*
Toujours sous les mêmes hypothèses, on suppose qu'il existe $n+2$ points $x_0 < x_1 < \ldots < x_{n+1}$ où l'erreur entre p et g a un extrémum local et tels que le signe de l'erreur alterne entre deux x_i successifs. On suppose en outre que l'extrémum global est atteint en l'un des x_i. Alors l'erreur optimale μ vérifie

$$\min_i |p(x_i) - g(x_i)| \leq \mu \leq \max_i |p(x_i) - g(x_i)|. \tag{3}$$

Outre le fait qu'il énonce un résultat profond (l'amplitude des oscillations est reliée à la qualité d'approximation), ce théorème fournit un moyen effectif de mesurer la qualité d'approximation d'un polynôme. En effet, p étant donné, on peut calculer ε et $\|p-g\|_\infty$. Si ces deux valeurs sont suffisamment proches l'une de l'autre, p peut être considéré comme pratiquement optimal.
En d'autres termes, nous pouvons voir $\|p-g\|_\infty$ comme une approximation de l'erreur optimale μ. L'erreur relative entre les deux valeurs peut etre majorée en fonction de ε :

$$\left| \frac{\|p-g\|_\infty - \mu}{\mu} \right| \leq \frac{\|p-g\|_\infty - |\varepsilon|}{|\varepsilon|} = \delta. \tag{4}$$

La valeur δ ainsi définie mesure la qualité d'approximation de p. Si par exemple $\delta \leq 10^{-5}$, cela signifie que $\|p-g\|_\infty$ (ou $|\varepsilon|$) est une estimation de l'erreur optimale avec approximativement cinq décimales correctes.

1.2 L'algorithme de Remez

Décrivons tout d'abord l'algorithme de Remez, qui est l'algorithme classique utilisé pour calculer le polynôme de meilleure approximation de degré inférieur ou égal à n d'une fonction sur un intervalle compact $[a,b]$.
L'algorithme de Remez prend pour entrées une fonction continue g, un intervalle $[a,b]$, un entier n et un paramètre réel strictement positif *qualité*. Il calcule un polynôme p de degré inférieur ou égal à n qui approche le polynôme $p^\star$ de meilleure approximation de g sur $[a,b]$. Plus précisément, il utilise le Théorème de La Vallée Poussin (cf. lŠéquation (2)) pour garantir que

$$\|p^\star - g\|_\infty \leq \|p - g\|_\infty \leq \|p^\star - g\|_\infty (1+\delta),$$

avec $\delta <$*qualité*. L'algorithme est résumé ci-dessous :

Input : $g, [a,b], n,$*qualité*
Output : un polynôme p **tel que** $\|p^\star - g\|_\infty \leq \|p-g\|_\infty \leq \|p^\star - g\|_\infty (1+$ *qualité*$)$

1. **Choisir** $n+2$ **points** $x_0 < x_1 < \ldots < x_{n+1}$ **dans** $[a,b]$ **;**
2. $\delta \leftarrow +\infty$ **;**
3. **while** (*qualité*) **do**
4. Calculer la meilleure approximation p de g sur $\{x_0, \ldots, x_{n+1}\}$;
5. $\varepsilon \leftarrow \max_i |p(x_i) - g(x_i)|$;
6. $M \leftarrow \|p-g\|_\infty$;
7. Choisir $n+2$ nouveaux points $x_0 < x_1 < \ldots < x_{n+1}$ tels que
$$\begin{cases} \exists i_0 = 0, \ldots, n+1,\ |p(x_{i_0}) - g(x_{i_0})| = M \\ \forall i = 0, \ldots, n+1,\ |p(x_i) - g(x_i)| \geq \varepsilon \\ \text{le signe de } p-g \text{ alterne entre deux } x_i successifs \end{cases}$$
8. $\varepsilon \leftarrow \min_i |p(x_i) - g(x_i)|$;
9. $\delta \leftarrow (M - \varepsilon')/\varepsilon'$;
10. **end ;**
11. **return** p ;

L'algorithme est présenté sous un formalisme assez général qui permettra de l'étendre facilement au cas de l'approximation sur un ensemble X quelconque. Cependant, si l'on veut implémenter spécifiquement l'algorithme pour l'approximation polynomiale sur un intervalle compact $[a, b]$, on peut effectuer l'étape 4 efficacement à l'aide d'un algorithme rapide d'interpolation. Pour faire cette interpolation rapidement et de façon numériquement stable, on a même intérêt à représenter les polynômes dans une base bien adaptée au calcul.

L'étape 5 est séparée de l'étape 4 dans l'algorithme mais, en pratique, p et ε sont calculés simultanément (ε est l'erreur optimale du problème d'approximation restreint à $\{x_0, \ldots, x_{n+1}\}$).

L'étape 7 laisse une certaine liberté pour le choix des nouveaux points. Là encore, la formulation est délibérément assez générale mais, en pratique, certains choix de points sont plus pertinents que d'autres. Si l'erreur $p - g$ a exactement $n + 2$ extréma locaux alternés dans l'intervalle $[a, b]$, on a intérêt à choisir les extréma locaux comme nouveau choix de points. Sous cette hypothèse (et en supposant de plus que g est deux fois différentiable sur $[a, b]$), Veidinger a montré que la convergence de l'algorithme est quadratique, c'est-à-dire que δ est au moins élevé au carré entre deux étapes. S'il y a plus que $n+2$ extréma locaux, le choix des nouveaux points est plus délicat.

À l'étape 9, on utilise le Théorème de La Vallée Poussin pour calculer un majorant δ de la qualité d'approximation de p. L'algorithme boucle jusqu'à ce que cette qualité soit suffisante.

Pour choisir le jeu de points initial $x_0 < \ldots < x_{n+1}$, il est souvent judicieux de considérer les points correspondant aux extréma du polynôme de Tchebychev de degré $n + 1$ sur $[a, b]$:

$$x_i = \frac{a+b}{2} + \frac{b-a}{2} \cos\left(\frac{(n+1-i)\pi}{n+1}\right), \; i = 0, \ldots, n+1. \tag{5}$$

2 Exercices

Exercice 1

On considère le programme suivant utilisable sous *SCILAB*.

```
nc=21 ;
ngrid=nc*250 ;
fgrid=.5*(0 :(ngrid-1))/(ngrid-1) ;
mag(1 :ngrid/2)=ones(1 :ngrid/2) ;
mag(ngrid/2+1 :ngrid)=0*ones(1 :ngrid/2) ;
weight=ones(fgrid) ;
guess=round(1 :ngrid/nc :ngrid) ;
guess(nc+1)=ngrid ;
guess(nc+2)=ngrid ;
an=remez(guess,mag,fgrid,weight) ;
h = sum[an(i)*cos(weight)], i=1 :n
```

1. Décrypter chacune des commandes de ce programme.
2. Analyser les paramètres de la fonction « remez ».

Exercice 2

On présente à la page suivante la méthode de Remez programmée par Sherif A. Tawfik (Faculty of Engineering, Cairo University) et utilisable sous MATLAB.

1. Interpréter chacune des commandes du programme.
2. Implémenter cette version de l'algorithme sous *SCILAB*.
3. Implémenter une variante de votre programme précédent sous *SCILAB*, utilisant les abscisses de Tchebycheff plutôt que les abscisses générées par la commande :

   y=linspace(interval(1),interval(2),order+2).

   Comparer les performances des deux programmes sur un même exemple.

```
function A=remez(fun, fun_der,interval,order)

powers=ones(order+2,1)*([0 :order]);
coeff_E=(-1).^[1 :order+2];
coeff_E=coeff_E( :);
t=1 :order;
t=t( :);

y=linspace(interval(1),interval(2),order+2);

for i=1 :10
y=y( :);   h=(y-interval(1))*ones(1,order+1);
coeff_h=h.^powers;
M=[coeff_h coeff_E];
N=feval(fun,y);
A=M\N;
A1=A(1 :end-1);
A_der=A(2 :end-1).*t;
z(1)=interval(1);
z(order+3)=interval(2);
for k=1 : order+1
z(k+1)=findzero(@err,y(k),y(k+1),fun,A1,interval(1));
end
for k=1 :order+2
if sign(err(z(k),fun_der,A_der,interval(1) ))~=sign(err(z(k+1),fun_der,A_der,interval(1)))
y1(k)=findzero(@err,z(k),z(k+1),fun_der,A_der,interval(1));
v(k)=abs(err(y1(k),fun,A1,interval(1)));
else
v1=abs(err(z(k),fun,A1,interval(1)));
v2=abs(err(z(k+1),fun,A1,interval(1)));
if v1>v2
y1(k)=z(k);
v(k)=v1;
else
y1(k)=z(k+1);
v(k)=v2;
end
end
end
[mx ind]=max(v);
if abs(y(ind)-y1(ind)) < 2^-30
break;
end
if ind<length(y) & abs(y(ind+1)-y1(ind)) < 2^-30
```

```
function e= err(x,fun, A, first)

A=A( :);
x=x( :);
order=length(A)-1;
powers=ones(length(x),1)*[0 :order];
temp=((x-first)*ones(1,order+1)).^powers;
temp=temp*A;
e=feval(fun,x)-temp;
```

```
function y=findzero(fun,x0,x1,varargin)

f0=feval(fun,x0,varargin :) ;

f1=feval(fun,x1,varargin :) ;

if sign(f0)==sign(f1)
error('the function at the two endpoints must be of opposite signs') ;
end
f=feval(fun,x,varargin{:}) ;
while abs(f)> 2^-52
if sign(f)==sign(f0)
x0=x ;
f0=f ;
else
x1=x ;
f1=f ;
end
x=x0 - f0 * ((x1-x0)/(f1-f0)) ;
f=feval(fun,x,varargin :) ;
end
y=x ;
```

```
fun=inline('exp(x)') ;
fun_der= inline('exp(x)') ;
interval=[0, 2^(-10)] ;
order =2 ;
A= remez(fun, fun_der, interval, order) ;
A1=A(1 :end-1) ;
E=A(end) ;
x=0 : 2^-15 :2^-10 ;
e=err(x,fun, A1, interval(1)) ;
plot(x,e)
xlabel('x')
ylabel('e(x)=f(x)-p(x)')
title('Error function for when approximating exp(x)')
fun=inline('sin(x)') ;
fun_der= inline('cos(x)') ;
interval=[0, 2^(-10)] ;
order =2 ;
A= remez(fun, fun_der, interval, order) ;
A1=A(1 :end-1) ;
E=A(end) ;
x=0 : 2^-15 :2^-10 ;
e=err(x,fun, A1, interval(1)) ;
figure ;
plot(x,e)
xlabel('x')
ylabel('e(x)=f(x)-p(x)')
title('Error function for when approximating sin(x)')
```

Références

[1] Aurélien ESNARD. *Algorithmique Numérique*

[2] Ricardo PACHÓN et Lloyd N. TREFETHEN. *Barycentric-Remez Algorithms For Best Polynomial Approximation In The Chebfun System*
http ://people.maths.ox.ac.uk/trefethen/remez_new2.pdf

[3] Sylvain CHEVILLARD. *Évaluation Efficace de Fonctions numériques. Outils et Exemples*
http ://www-sop.inria.fr/members/Sylvain.Chevillard/download/papers/TheseSylvainChevillard.pdf