TP2 Matlab
Interpolation, approximation

Filali Ayoub —LAY PHOAN Lida — Skalli Khaled
CPES 2

16 mai 2012

1 Inrerpolation polynomiale de Lagrange

a)

Dans matlab, un polynome est représenté par le vecteur de ses coefficients, commençant par le degré le plus élevé.
Exemple :
Soit le polynome : $P(x) = 8x^4 + 5x^2 + x - 7$, il sera représenté par le vecteur :

$$\mathrm{P} = \begin{bmatrix} 8 \\ 0 \\ 5 \\ 1 \\ -7 \end{bmatrix}$$

– On considère deux points du plan $A(x_0, y_0)$, $B(x_1, y_1)$ avec $x_0 < x_1$. La formule donnant le polunôme de degré 1 qui passe par ses deux points consiste à résoudre le système :

$$\left\{ \begin{array}{ll} ax_0 + b & = y_0 \\ ax_1 + b & = y_1 \end{array} \right. \Leftrightarrow \begin{bmatrix} x_0 & 1 \\ x_1 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \end{bmatrix} \quad (1)$$

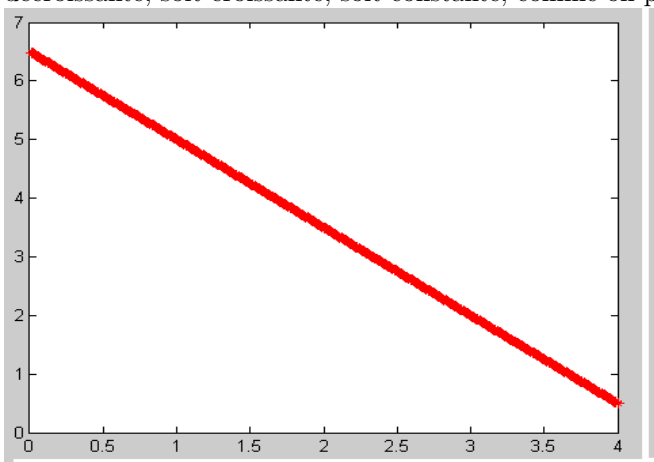
Pour commencer, on choisit un polynôme de degré 1 dont on connait déjà les coéfficients a et b,par des formules calculées a la main. On verra alors de suite que les formules sont fausses dès que matlab nous renvoie une mauvaise solution à nos scripts.Sinon, comme dans notre cas simple, les formules étant bonnes, les courbes obtenues sont satisfaisantes.
On commence par calculer les coéfficients de p_1 à la main. Puis on fait le script en utilisant la fonction polyval qui va évaluer le polynôme $p1$ (la fonction polynômiale) en des points donnés.

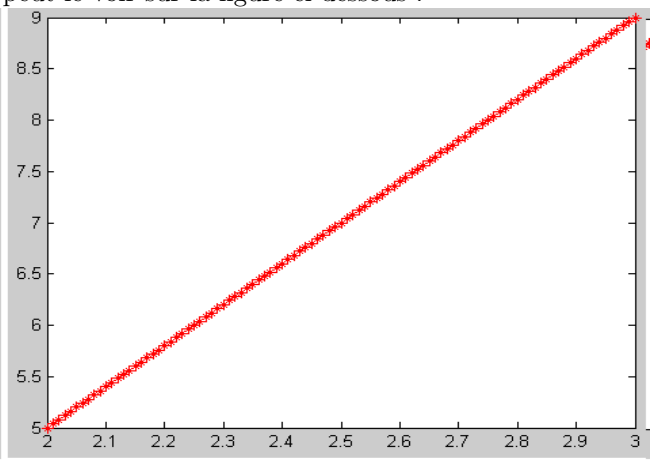
interpol1.m

```
% 2 points sur l'axe (Ox)
x1=0
x2=2
% 2 points sur l'axe (Ox)
y1=1
y2=5
% calcul a et b à la main
a=(y2-y1)/(x2-x1)
b=(y1+y2-(y2-y1)*(x1+x2)/(x2-x1))/2

% matrice obtenu
[p1]=[a b];

% tracé du polynôme
h=1e-2
xx=x1-1:h:x2+1;
yy=polyval(p1,xx);
plot(xx, yy,'*r');
```

La syntaxe est $polyval(p1, xx)$ où x est soit une valeur numérique, soit un vecteur. Dans notre exercice, on obtient un vecteur contenant les valeurs de la fonction polynômiale aux différents points spécifiés dans le vecteur xx.

– Lorsque l'on fait varier les 2 points de coordonnées (x_0, y_0) et (x_1, y_1), on change alors la pente de la courbe, ainsi que son ordonnée a l'origine. En effet, la courbe du polynôme obtenue peut etre soit décroissante, soit croissante, soit constante, comme on peut le voir sur la figure ci-dessous :





b)

Résoudre un système d'équation à 2 inconnues est facile, mais au delà de 3 inconnues, cela devient laborieux. C'est pourquoi pour évaluer un polynôme p_n de degré n avec $n \geq 2$, on fera appel à des fonctions déjà existantes sous matlab. Pour ce faire, on va utiliser deux méthodes :

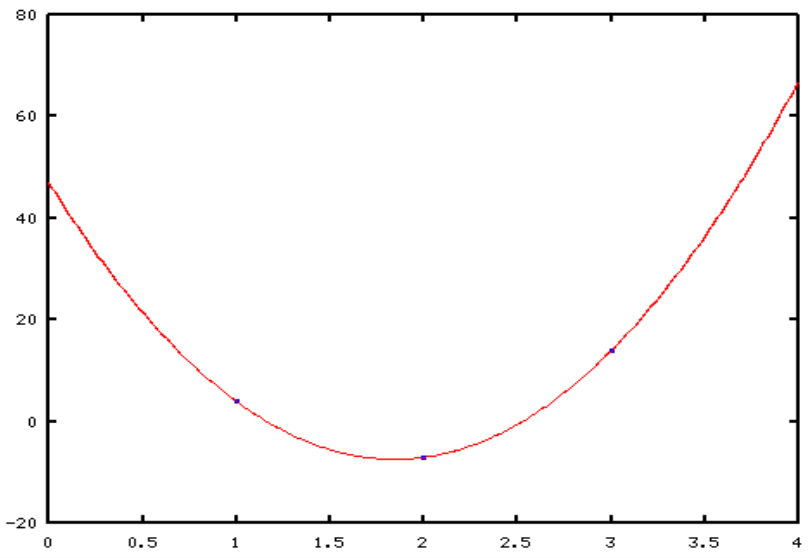
– La boucle for :
Dans le script interpol2.m, ci-dessous, on voit qu'à la ligne 7, la matrice V se construit au fur et à mesure en suivant l'incrémentation ayant lieu à chaque itération de la boucle for (figure interpol2.m, l. 5-14).

– La matrice de Vandermonde :
On peut également calculer la matrice d'inverse d'un polynôme en utilisant la matrice de Vandermonde à l'aide de la commande vander. La méthode de Vandermonde peut être utilisée pour trouver un polynôme d'interpolation. La substitution des points dans le polynôme désiré conduit à la résolution du système d'équations linéaires. Les coefficients du polynôme sont déterminés par l'élimination de Gauss.

En utilisant la même méthode que la question 1), on cherche un polynôme de degré 2 passant par trois points (x_0, y_0), (x_1, y_1), (x_2, y_2).

interpol2.m

```
% recherche polynôme de degré 2 passant par 3 points
x=[ 1 2 3];
y=[4 -7 14];

%possibilité 1
%n=degré du polynome
% V=zeros(n+1);
% for i=1:n+1
%     for j=1:n+1
%         V(i,j)= x(i)^(n+1-j);
%         %pause
%     end
% end
%possibilité2

% vander : matrice de Vandermonde permettant de calculer la matrice inverse
V=vander(x)
p2=V\y'

h=1e-2
x1=min(x)
x2=max(x)
xx=x1-1:h:x2+1;
yy=polyval(p2,xx);
plot(x,y,'*',xx, yy,'r');
```



Ici, on voit bien que l'on trouve le polynôme d'interpolation en seulement deux lignes de commande grâce à la fonction vander (figure interpol2.m, l. 17-18). Par rapport à la question 1.a), on a amélioré nos script : Le script est moins lourd et plus lisible et on gagne du temps car c'est matlab qui fait les calculs à notre place.

c)

– Le problème général d'interpolation de $n+1$ points (x_i, y_i), $i = 0...n$ par un polynôme de degré n peut s'écrire comme un système linéaire $Vp' = y'$ où p' est le vecteur colonne des coefficients du polynôme, y' le vecteur colonne des ordonnées et V la matrice dépendant des abscisses x car p' dépend linéairement des coefficients inconnus (cf, figure interpol2.m où V est une matrice donnée par la commande vander). On utilise cette méthode en particulier pour faire de l'approximation polynômiale.

On fait un script général du calcul d'interpolation polynômiale :

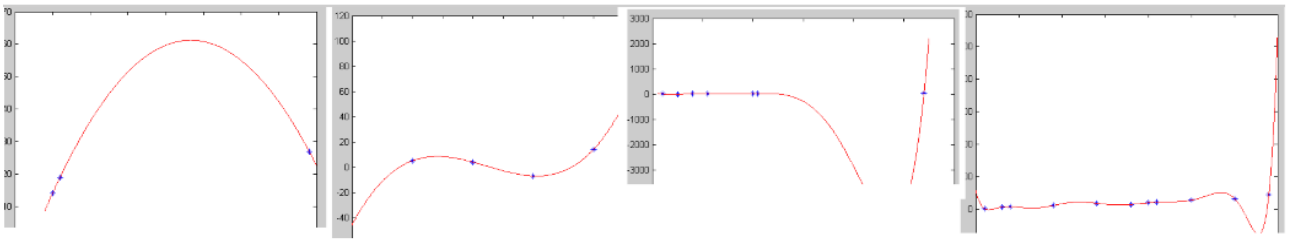
interpol.m

```
function [p]=interpol(x,y)
V=vander(x);
p=V\y';
```

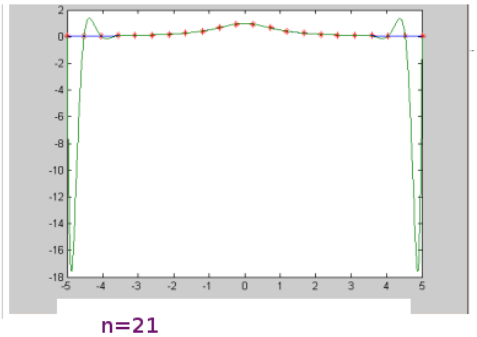
script.m

```
x=[0 6 2 9]
y=[5 4 0 14]
p=interpol(x,y)
h=1e-2
x1=min(x)
x2=max(x)
xx=x1-1:h:x2+1;
yy=polyval(p,xx);
plot(x,y,'*',xx, yy,'r');
```

Grâce à ce script, on peut calculer facilement le polynôme d'interpolation passant par le nombre de points quelconque. On a juste à modifier x et y. On trace quelques courbes passant par le nombre de points différents :



3 points 4 points 7 points 11 points



n=21

– Comparaison avec le polynôme de Lagrange

Dans le domaine mathématique de l'analyse numérique, le phénomène de Runge se manifeste dans le contexte de l'interpolation polynomiale, en particulier l'interpolation de Lagrange. Avec certaines fonctions (même infiniment dérivables), l'augmentation du nombre n de points d'interpolation ne constitue pas nécessairement une bonne stratégie d'approximation.

Interpolation lagrangienne *On se donne $n+1$ points $(x_0, y_0), \ldots, (x_n, y_n)$ (avec les x_i distincts deux à deux). On construit un polynôme de degré minimal qui aux abscisses x_i prend les valeurs y_i,*

$$L(X) = \sum_{j=0}^{n} y_j \left(\prod_{i=0, i\neq j}^{n} \frac{X - x_i}{x_j - x_i} \right) \tag{2}$$

La fonction permettant de calculer le polynôme d'interpolation lagrangienne :

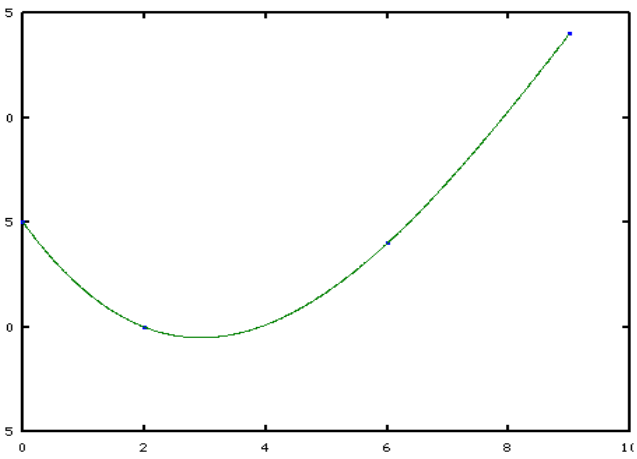
lagrange.m

```
% polynôme d'interpolation lagrangienne
function s=lagrange(x,y,X)
        % x est le vecteur des xi
        % y est le vecteur des yi
        % On évalue polynôme p passant par les points xi, yi X€R
        % s = p(Y)
s=0;
n = length (x)-1;
for i=0:n
    %ici calcul du bloc i
    p=y(i+1);
    for j=0:n
        if j ~= i
            p=p*(X-x(j+1))/(x(i+1)-x(j+1));
        end
    end
    s=s+p;
end
```

– Comparaison du résultat avec la fonction polyfit :
Dans les documents ci-dessous, on a tracé la courbe de la fonction polynômiale avec deux méthodes différentes, l'une utilisant polyfit et l'autre utilisant interpol. On voit bien qu'il existe une petite différence entre les deux méthodes au niveau de la précision et de l'approximation polynômiale. En effet, pour calculer le polynôme p_n de degré faible, on a une très faible différence entre les deux méthodes (figure 1) mais lorsque l'on augmente le degré du polynôme recherché, on voit que polyfit reste la solution la plus proche du résultat cherché. On voit que la courbe du deuxième script ne passe pas par tous les points contrairement à la courbe de la fonction polyfit (figure2) .

Figure 1

```
xe=[0 6 2 9];
ye=[5 4 0 14];
B5 = polyfit (xe, ye, 3)
t = -0:0.01:9;
plot(xe, ye, 'o', t, polyval(B5,t))
```



```
x=[0 6 2 9];
y=[5 4 0 14];
p=interpol(x,y);
h=1e-2;
x1=min(x);
x2=max(x);
xx=x1-1:h:x2+1;
yy=polyval(p,xx);
plot(x,y,'*',xx, yy,'r')
```

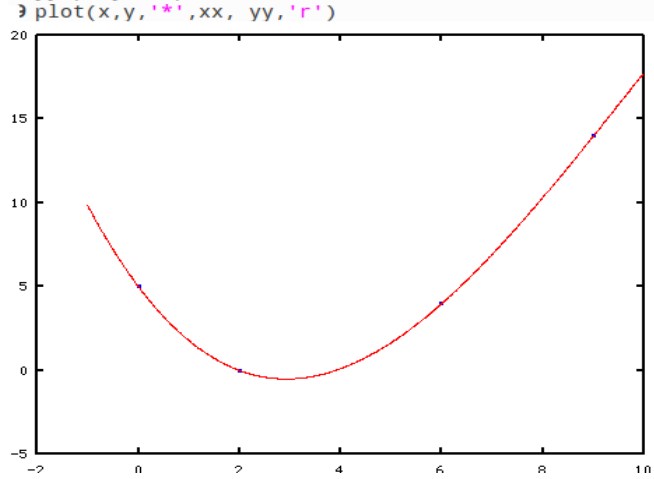
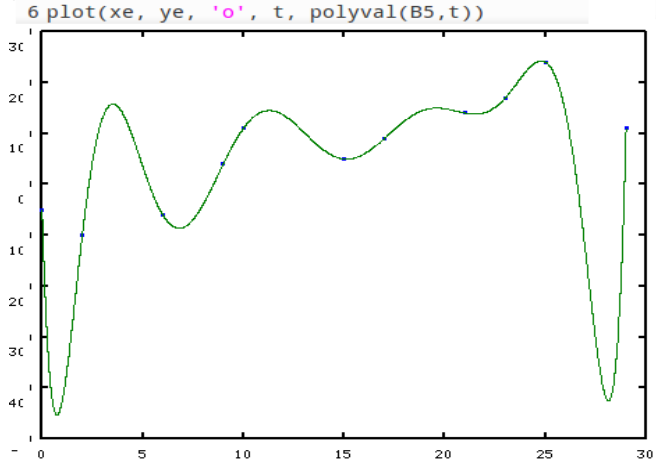
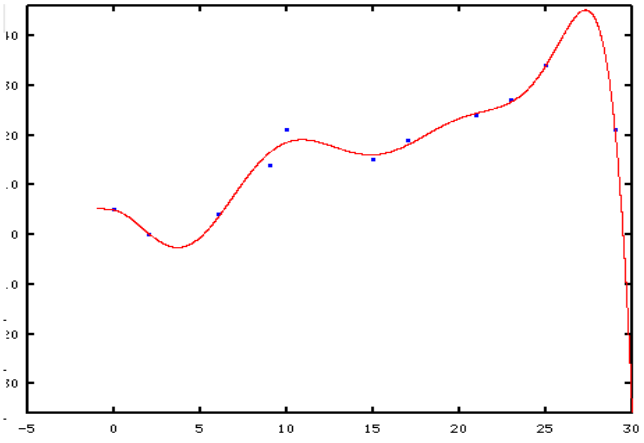


Figure 2

```
1 xe=[0 6 2 9 10 15 17 21 23 25 29];
2 ye=[5 4 0 14 21 15 19 24 27 34 21];
3
4 B5 = polyfit (xe, ye, 10)
5 t = -0:0.01:29;
6 plot(xe, ye, 'o', t, polyval(B5,t))
```




```
1 x=[0 6 2 9 10 15 17 21 23 25 29];
2 y=[5 4 0 14 21 15 19 24 27 34 21];
3 p=interpol(x,y);
4 h=1e-2;
5 x1=min(x);
6 x2=max(x);
7 xx=x1-1:h:x2+1;
8 yy=polyval(p,xx);
9 plot(x,y,'*',xx, yy,'r')
```



– Conclusion :
Dans cet exercice, on peut utiliser directement la fonction polyfit, qui, en plus des coefficients du polynôme, renvoie des informations sur la qualité de l'approximation réalisée. Ainsi, polyfit attend en entrée simplement les x_i,les y_i et l'ordre du polynôme recherché et la fonction polyval peut être utilisée pour recalculer le polynôme d'approximation en tout point. De plus, la fonction polyfit permet de calculer le polynome d'approximation au sens des moindres carrés que l'on verra dans l'exercice 3.

2 Approximation d'une fonction par un polyôme, phénomène de Runge, points de Tchebechev

a) Calcul du polynôme d'interpolation p_n passant par les points x et y

On choisit $x = x_0, ..., x_n$ subdivision équidistante de l'intervalle $I = [-5, 5]$ et $y = f(x)$ où $f(x) = \frac{1}{1+x^2}$.
On fait le script permettant de calculer le polynôme d'interpolation p_n passant par les points x et y sur l'intervalle $I = [-5, 5]$ et on trace la fonction f et le polynôme d'interpolation :

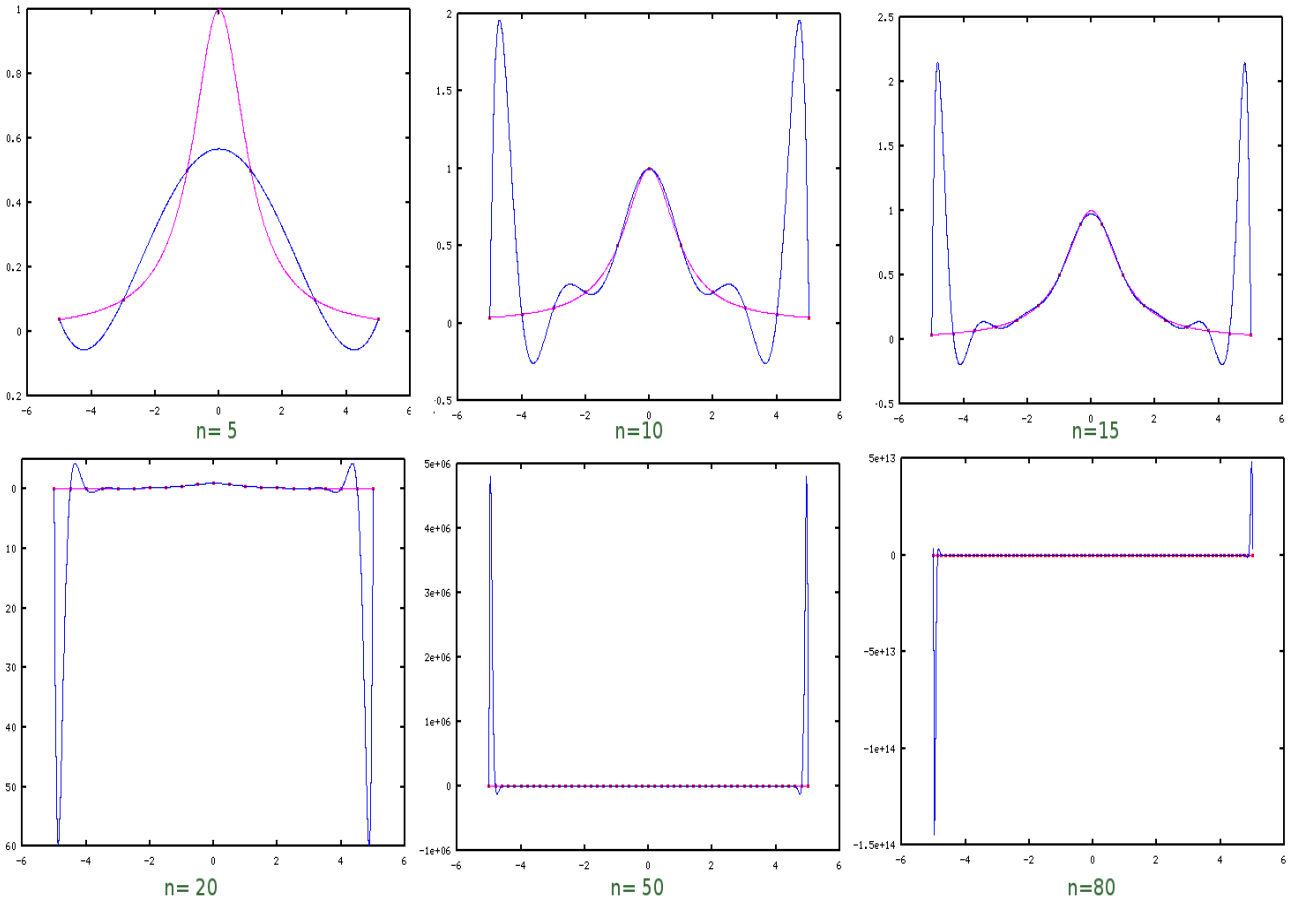
Interpol2.m

```
1 function [x,xx,fx,fxx,pxx] = Interpol2(a,b,h,n )%définition de la fonction f.
2  xx = a : h : b;
3 fxx = 1./(1+xx.^2); %interpolation pour n+1 points.
4 x = linspace(a,b,n+1);% subdivision equidistante
5 fx = 1./(1+x.^2);
6 p = polyfit(x,fx,n); %polynome interpolant les points (x,fx)
7 pxx = polyval(p, xx);
```

s2.m

```
1 % script : calcul polynôme d'interpolation
2 a=-5
3 b=5
4 h=1e-2
5 n=80
6 [x,xx,fx,fxx,pxx] = Interpol2(a,b, h,n );
7 plot(x,fx,'*r',xx,fxx,'m',xx,pxx)
```

En faisant varier n, on obtient les courbes suivantes :



Si n augmente on s'attend naturellement à voir le polynôme se rapprocher uniformément de la fonction f. Mais on remarque que, dans le cas des points équidistants, l'erreur d'interpolation diverge aux bornes de l'intervalle (l'erreur augmente avec n). Par contre cette divergence, appelé phénomène de Runge prouve que l'interpolation polynomiale n'est pas apte d'approximer correctement toutes les fonctions réelles.
Dans la figure ci-dessous, on voit bien que le polynôme p_n (courbe en bleu) donne l'impression de converger vers la fonction f (courbe en magenta). Pour $n \geq 50$, on voit que sur cet intervalle $I = [-5, 5]$, on obtient quasiment la même courbe pour les deux fonctions.

b)

– Point Tchebychev avec Lagrange
Sur l'intervalle $I = [a, b]$, et pour n fixé, on définit les points de Tchebychev par :

$$x_i = \frac{a+b}{2} - \frac{b-a}{2}\cos(\frac{(2i+1)\pi}{(n+1)2}), i = 0...n \tag{3}$$

Le phénomène de Runge est un effet de bord. On espère l'atténuer en changeant les points d'interpollation, et en les mettant plus près du bord de l'intervalle. Un résultat théorique assez technique montre que le choix des abscisse de Tchebychev est essentiellement optimal. En reprenant le script de la question précédente et utilisant les points x et $y = f(x)$

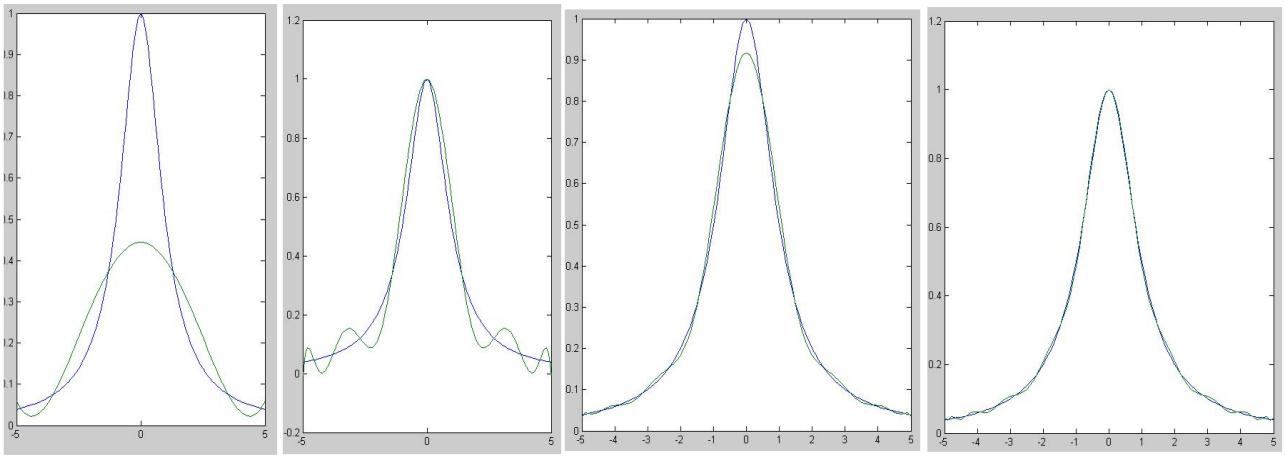
lagrange.m

```
% polynôme d'interpolation lagrangienne
function s=lagrange(x,y,X)
        % x est le vecteur des xi
        % y est le vecteur des yi
        % On évalue polynôme p passant par les points xi, yi X€R
        % s = p(Y)
s=0;
n = length (x)-1;
for i=0:n
    %ici calcul du bloc i
    p=y(i+1);
    for j=0:n
        if j ~= i
            p=p*(X-x(j+1))/(x(i+1)-x(j+1));
        end
    end
    s=s+p;
end
```

s2b.m

```
a=-5;
b=5;
n=5
h=1e-1;
;
%créer x les abscisses  de Chebyshev
j = 0 : n;
x = (a+b)./2 - (b-a)./2 *cos((2*j+1)./(n+1)*pi/2);
%créer les ordonnées y=f(x)
fx=f(x);
xx=a:h:b;
fxx=f(xx);
l=length(xx);
pxx=zeros(1,l);
for i=1:l
    X=xx(i);
    pxx(i) =lagrange(x,fx,X);
end
plot(xx,fxx,xx,pxx)%on trace la courbe d'erreur entre les 2 fonctions
```

Dans la figure ci-dessous, on voit que lorsqu'on augmente n, on obtient une meilleure précision du résultat cherché entre les deux courbes (en bleu : fonction f et en vert : polynôme d'interpolation cherché à l'aide de la méthode de Lagrange.



n = 5 n = 10 n = 15 n = 20

On remarque que les points de Tchebychev sont plus concentrés sur les bords de l'intervalle et il ne semble pas y avoir d'effet de Runge. Les polynômes de Tchebychev constituent un outil important dans le domaine de l'interpolation : en effet, on démontre que pour minimiser l'erreur d'interpolation

de Lagrange, les racines des polynômes de Tchebychev sont les candidats pour être les points d'interpolations.

– Points Tchebychev avec polyfit

points_Chebyshev3.m

```
function[x,xx,fx,fxx,pxx] = points_Chebyshev3( a,b,h,n )
%définition de la fonction f(x)
xx = a :h: b;
yy = 1./(1+xx.^2);

% interpolation par noeuds de Tchebychev pour n+1 points
i = 0 : n;
%x = a : 0.001 : b;
x = (a+b)./2 - (b-a)./2 *cos((2*i+1)./(n+1)*pi/2);
fx= 1./(1+x.^2);

% Polyfit renvoie les coefficients d'interpolation du polynôme p de degré n
p = polyfit(x,fx,n);

% Polyval renvoie la valeur du polynôme évaluée à x.
pxx = polyval(p,xx);

```

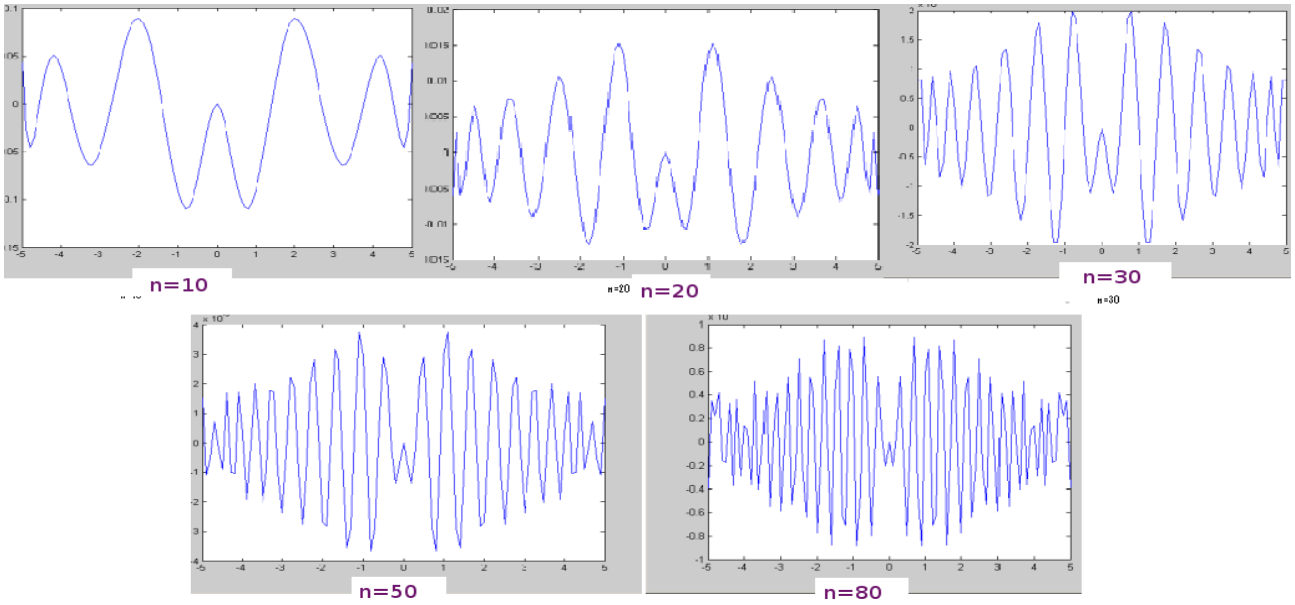
On peut également utiliser la fonction polyfit ci-dessus pour calculer notre polynôme. On va juste modifier quelques lignes sur notre script s2b.m et on obtiendra plus ou moins les même résultats pour n petit.

– comparaison ave polyfit erreur :

s2b_erreur.m

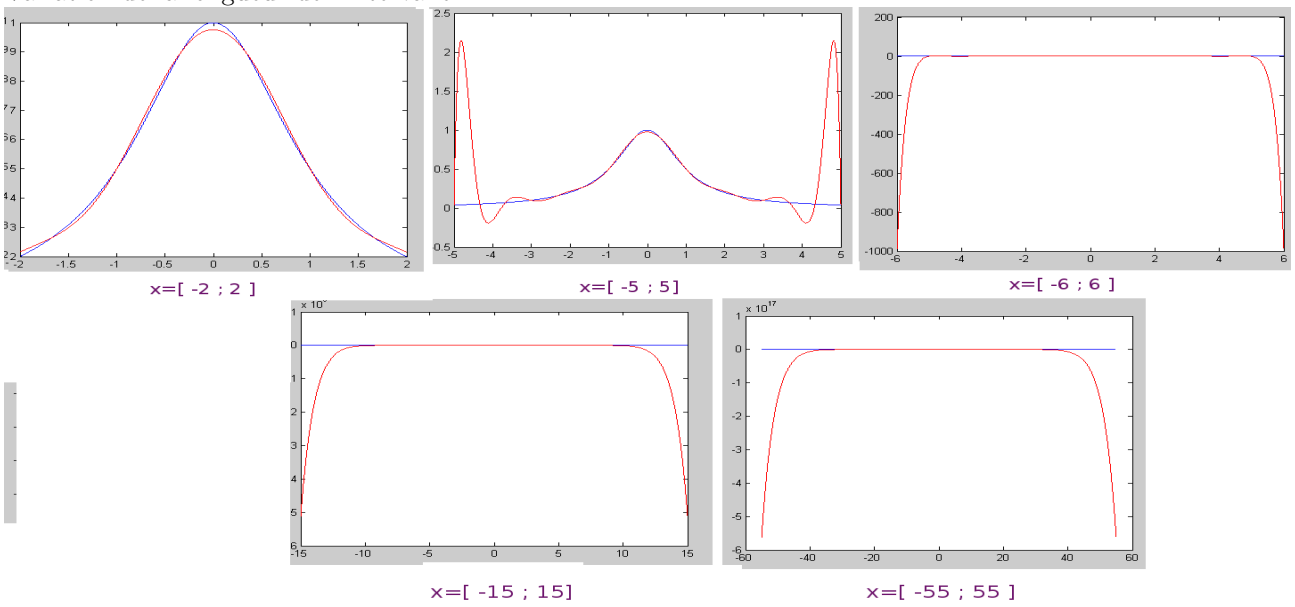
```
a=-5;
b=5;
n=80
h=1e-1;
%créer x les abscisses  de Chebyshev
j = 0 : n;
x = (a+b)./2-(b-a)./2 *cos((2*j+1)./(n+1)*pi/2);
%créer les ordonnées y=f(x)
fx=f(x);
xx=a:h:b;
fxx=f(xx);
l=length(xx);
pxx=zeros(1,l);
        for i=1:l
            X=xx(i);
            pxx(i) =lagrange(x,fx,X);
        end
plot(xx,(fxx-pxx))
```

Dans la figure ci-dessous, on voit que le polynôme donne l'impression de converger vers la fonction f lorsque n tend vers l'infini. En effet plus on augmente n et plus le nombre d'oscillations augmente.De plus, on remarque que lorsque l'on augmente n de 10, la courbe d'erreur diminue d'une puissance de 10(par exemple,si n passe de 10 à20, l'erreur passe de 10^{-1} à 10^{-2} En fait, lorsqu'on augmente le nombre de points, on constate que le polynôme se met à osciller fortement entre les points x_i , comme l'illustre la figure.



Le phénomène de Runge a montré que l'erreur d'interpolation entre P_n et f tend vers zéro lorsque n augmente. Autrement dit, plus on fixe de points où le polynôme a la même valeur que f, meilleure est l'approche de la fonction.

– Variation de la longueur de l'intervalle



Sur la figure ci-dessus, on a fait varier la longueur des x et on voit qu'à partir de $x = [-6, 6]$, on ne verra plus grande chose.

3 Approximation polynomiale au sens des moindres carrés

a)

On se donne $n+1$ points (x_i, y_i), $i = 0...n$ et on veut déterminer le polynôme p_1 de degré 1 qui minimise l'erreur quadratique :

$$S = \sum_{i=0}^{n} [y_i - p(x_i)]^2 \tag{4}$$

Afin de montrer que ce problème possède une unique solution que l'on déterminera ensuite. On commence par poser $p(x_i) = ax_i + b$. L'équation (4) devient alors :

$$S(a,b) = \sum_{i=0}^{n} [y_i - ax_i - b]^2 \tag{5}$$

En développant ceci (5), on obtient une forme quadratique en a et b, où a et b sont des constantes. On fait la dérivée partielle par rapport à a et b puis on cherche le minimum de la dérivée partielle de S :

$$\begin{cases} \frac{d}{da}S(a,b) & = \sum_{i=0}^{n}[2y_i x_i - 2ax_i^2 - 2x_i b] & = 0 \\ \frac{d}{db}S(a,b) & = \sum_{i=0}^{n}[2y_i - 2ax_i - 2b] & = 0 \end{cases} \tag{6}$$

On obtient ceci : $\begin{bmatrix} \sum_{i=0}^{n} x_i^2 & \sum_{i=0}^{n} x_i \\ \sum_{i=0}^{n} x_i & n+1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \sum_{i=0}^{n} y_i x_i \\ \sum_{i=0}^{n} y_i \end{bmatrix}$

Sous matlab, on crée une fonction permettant de calculer la matrice. $p1 = \begin{bmatrix} a \\ b \end{bmatrix}$

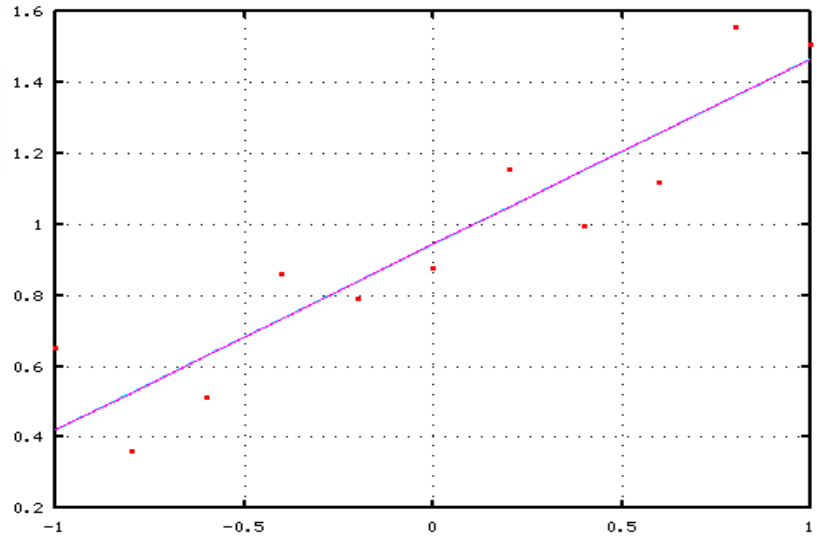
Une fois la matrice $p1$ obtenue, on crée un script permettant de calculer l'erreur quadratique de tous les polynômes de degré 1 :

mc1.m

```
% polynôme de degré 1
function p1=mc1(x,y)
n=length(x)-1;

% matrice A
sx2=sum(x.^2);
sx=sum(x);
A=[sx2 sx;sx n+1]

% matrice Y
sxy=sum(x.*y);
sy=sum(y);
Y=[sxy;sy]

%polynôme p1 obtenu par calcul de la matrice inverse de Y.
p1=A\Y
```

s1.m

```
% polynôme de degré 1 qui minimise l'erreur quadratique
n=10;

% randn génère des nombres alÃ©atoires suivant une loi gaussienne
pas=2/n;
x=-1:pas:1
y=0.5*x + 1+ randn(1,n+1)*0.2;
% Lorsque l'on baisse la valeur du coefficient qui est de 0,2 au départ, les
% Les points se rapprochent de plus en plus
% de la droite d'équation grâce à la fonction randn.

% les coéfficients cherchés
p1=mc1(x,y);
a=p1(1);
b=p1(2);

% polynôme de degré 1
h=1e-2;
xx=min(x):h:max(x);
yy=a*xx+b;
pf=polyfit(x,y,1);
% On trace les 2 courbes pour en déduire l'erreur quadratique de façon
% graphique.
plot(xx,yy,'c',x,y,'*r', x, polyval(pf,x), 'm')
grid
```



Comme on le voit dans le script ci-dessus, on a préféré utiliser la commande randn pour définir la fonction y car randn permet de générer des nombres aléatoires suivant une loi gaussienne, ce qui fait qu'on a une meilleure approximation de l'erreur quadratique autour de la courbe yy.

On voit que les deux courbes en cyan (approximation polynômiale au sens des moindres carrés) et en magenta (approximation par la fonction polyfit) sont confondues et représentées par une droite est tracée par rapport à la moyenne des points rouges de. On n'a donc pas une bonne approximation pour un polynôme d'ordre 1.

b)

– En reprenant la même démarche que dans la question a), on peut déterminer un polynôme de degré 2 avec des points (x_i, y_i), $i = 0..n$. On obtient cette équation :

$$\begin{bmatrix} \sum_{i=0}^{n} x_i^4 & \sum_{i=0}^{n} x_i^3 & \sum_{i=0}^{n} x_i^2 \\ \sum_{i=0}^{n} x_i^3 & \sum_{i=0}^{n} x_i^2 & \sum_{i=0}^{n} x_i \\ \sum_{i=0}^{n} x_i^2 & \sum_{i=0}^{n} x_i & n+1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} \sum_{i=0}^{n} y_i x_i^2 \\ \sum_{i=0}^{n} y_i x_i \\ \sum_{i=0}^{n} y_i \end{bmatrix}$$

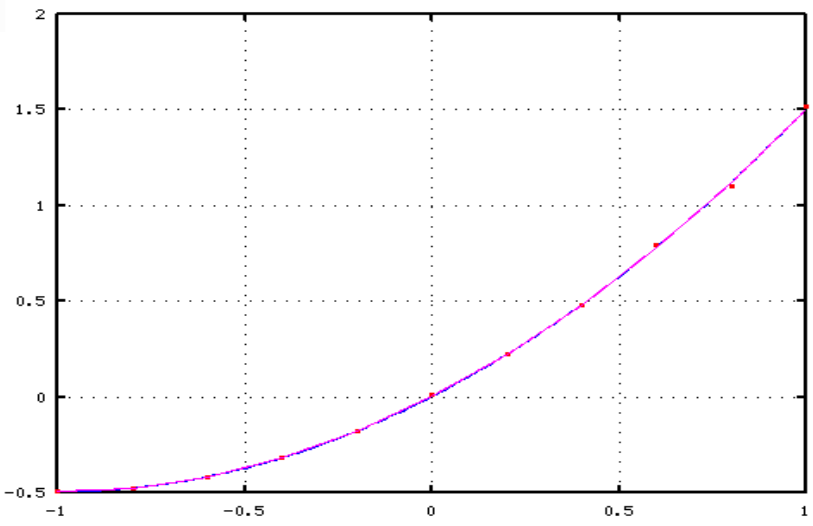
On fait le script et la fonction correspondant :

mc2.m

```
% polynôme de degré 2
function p2=mc2(x,y)
n=length(x)-1;

% matrice B
sx4=sum(x.^4);
sx3=sum(x.^3);
sx2=sum(x.^2);
sx=sum(x);
B=[sx4 sx3 sx2;sx3 sx2 sx;sx2 sx n+1]

% matrice Y
syx2=sum(y.*x.^2);
sxy=sum(y.*x);
sy=sum(y);
Y=[syx2;sxy;sy]

% matrice p2 obtenu par calcul de la matrice inverse.
p2=B\Y
%Le principe reste le même à l'ordre 2.
```

s2f.m

```
% polynôme de degré 1 qui minimise l'erreur quadratique
n=10

pas=2/n;
x=-1:pas:1;
y=0.5*x.^2+x + randn(1,n+1)*0.01;
% Il faut être plus précis car on veut approximer ces points par un polynôme
% du 2nd ordre.On choisit alors par exple ici des valeurs autour de 1e-2.

%les coeficients de p2
p2=mc2(x,y);
a=p2(1);
b=p2(2);
c=p2(3);

% polynôme de degré 2 minimisant l'erreur quadratique.
h=1e-2
xx=min(x):h:max(x);
yy=polyval(p2,xx);
pf=polyfit(x,y,2)
% On trace les 2 courbes et on teste différentes valeurs de coefficients
% devant "randn(1,n+1)" afin de minimiser l'erreur de façon graphique.
plot(xx,yy,x,y,'*r', x,polyval(pf,x), 'm')
grid
```



En comparaison avec la question précédente, ici, on a un meilleur résultat pour un polynôme de degré 2. En effet la courbe du polynôme passe pratiquement par tous les points. De plus, en comparant avec l'instruction polyfit, on obtient un très bon résultat puisque les deux courbes obtenues sont confondues.

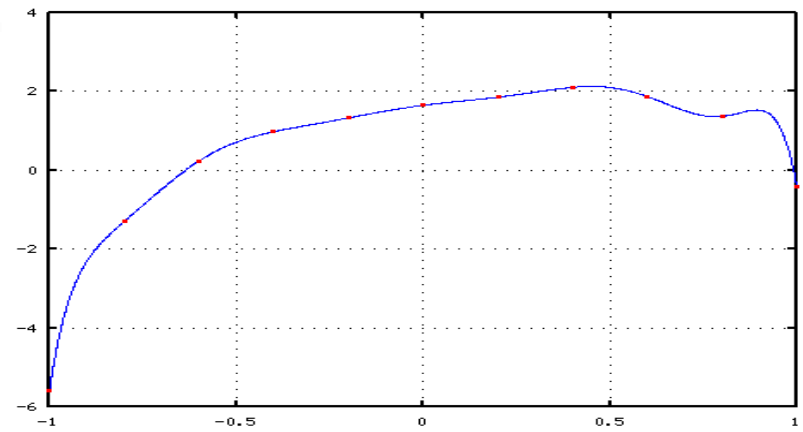
– On reprend le même travail, mais cette fois ci, on va généraliser avec un polynôme de degré d, $d \leq n$.

mcd1.m

```
function [pd,condA]=mcd1(x,y,d)
n=length(x)-1;
% d<=n degré du polynome et (d+1) est la taille de la matrice.

% sxd=sum(x.^d);

A=zeros(d+1);
Y=zeros(d+1,1);
 for i=1:d+1
     for j=1:d+1
        %formule des coéfficient de la matrice A en fction de i et j
        A(i,j)= sum(x.^(2*d+2-i-j));

     end
     % formule des coéfficient de Y (matrice colonne) en fonction de i .
     Y(i) = sum(y.*(x.^(d-i+1)));

 end
% calcul du conditionnement de A
condA=cond(A);
pd=A\Y;
```

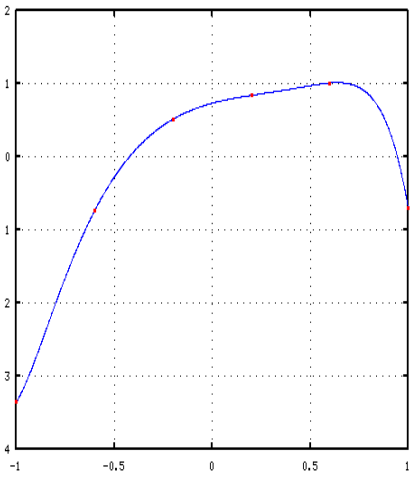
sd1.m

```
n=10
d=10 % On règle le degré du polynôme .
pas=2/n;
x=-1:pas:1;
% on crée maintenant un polynôme de degré d, alétoire.
p = randn(1,d+1);
px = polyval(p,x); % On l'évalue au point x.
bruit = randn(1,n+1)*0.05;
y = px +bruit;

% condA=norm(A)*norm(A.^-1),norm=norme euclidienne.
[pd,condA]=mcd1(x,y,d)

h=1e-3;
xx=min(x):h:max(x);
% On trace la modélisation par un polynôme de degré (d) des points obtenus
yy=polyval(pd,xx);
% Moins le coef devant randn du bruit est élevé et plus les points sont
% rapprochés de "pd"

plot(xx,yy, x,y,'*r')
grid
% on obtient vraiment le même résultat en utilisant l'instruction polyfit.
pf=polyfit(x,y,d);
% Erreur entre les 2 méthodes.
erreur=norm(pd-pf')
```

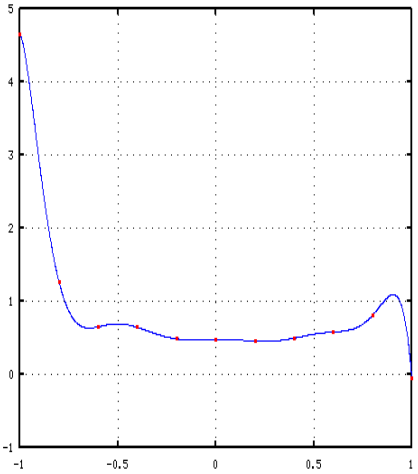


On peut jouer en faisant varier les différentes valeurs de n et d. Pour d fixé, en faisant varier n, vis versa,on obtient les tableaux suivants :

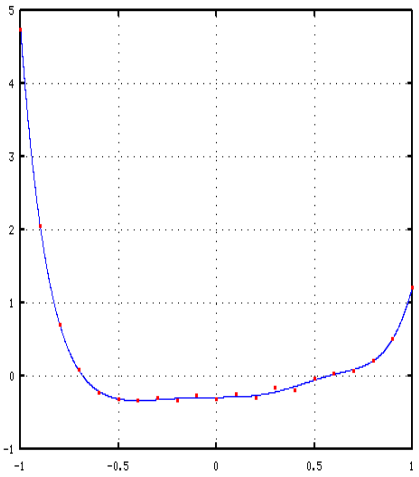
n	10	20	30	50	80
condA	2.10^8	$1,1.10^7$	$8,7.10^6$	$8,3.10^6$	$8,46.10^6$
erreur	$3,54.10^{-7}$	9.10^{-10}	$1,2.10^{-9}$	$3,8.10^{-11}$	$5,14.10^{-10}$



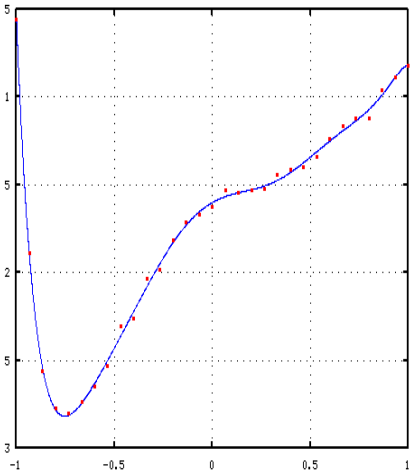
n=5



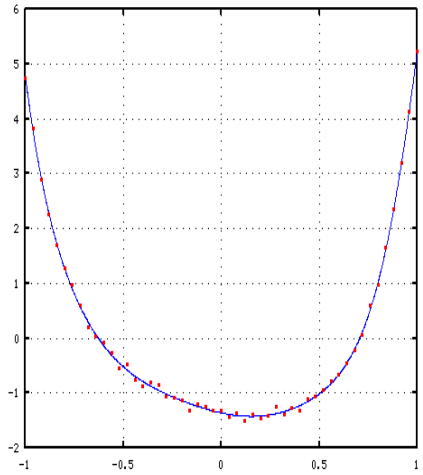
n=10



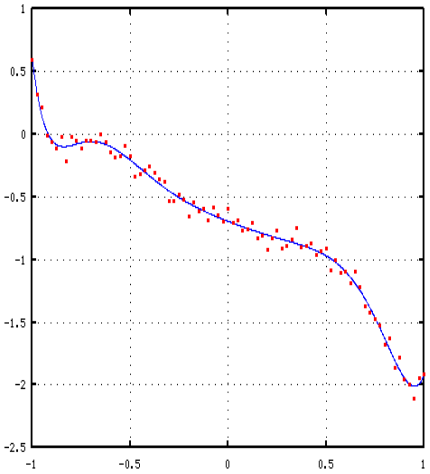
n=20



n=30

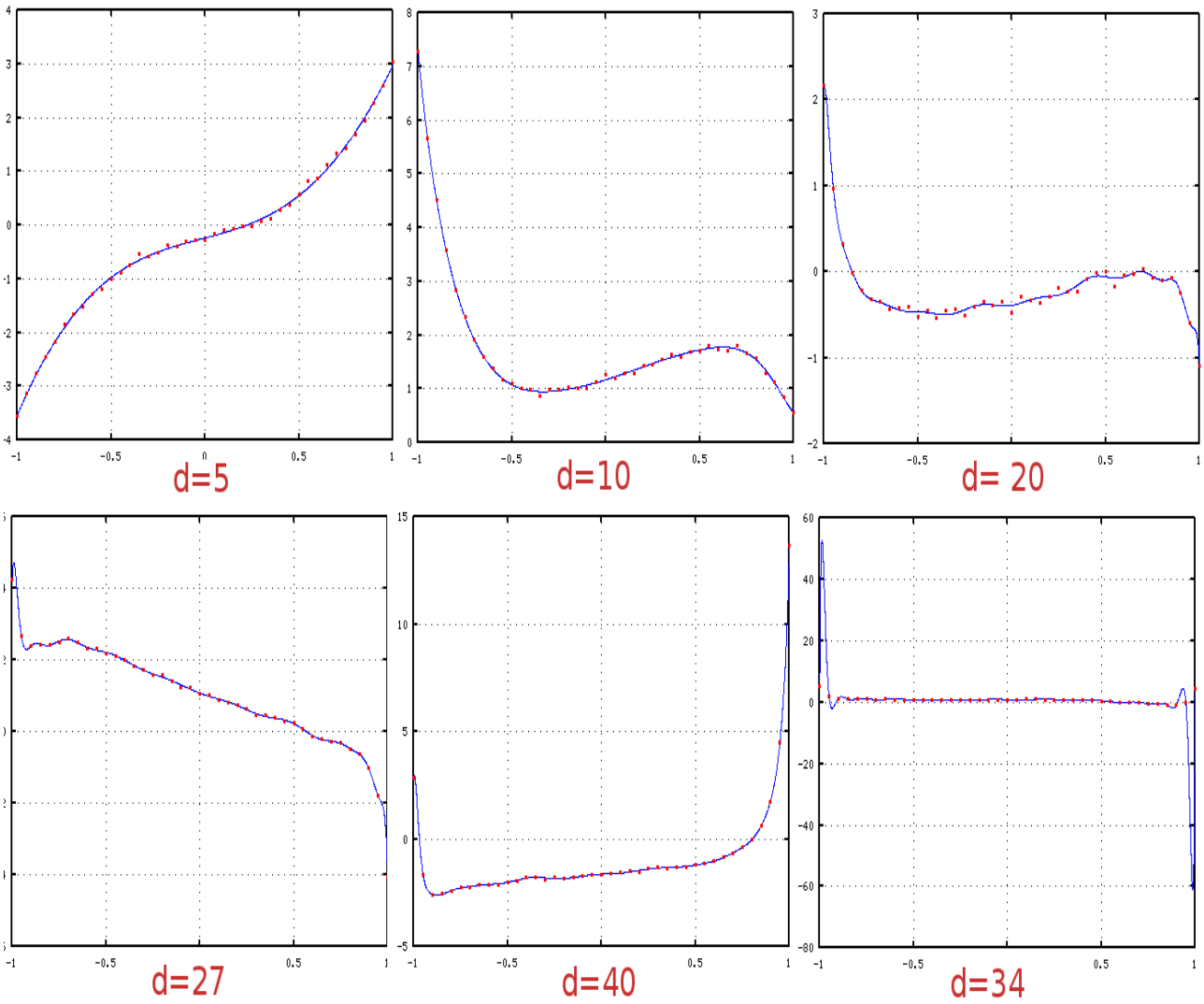


n=50

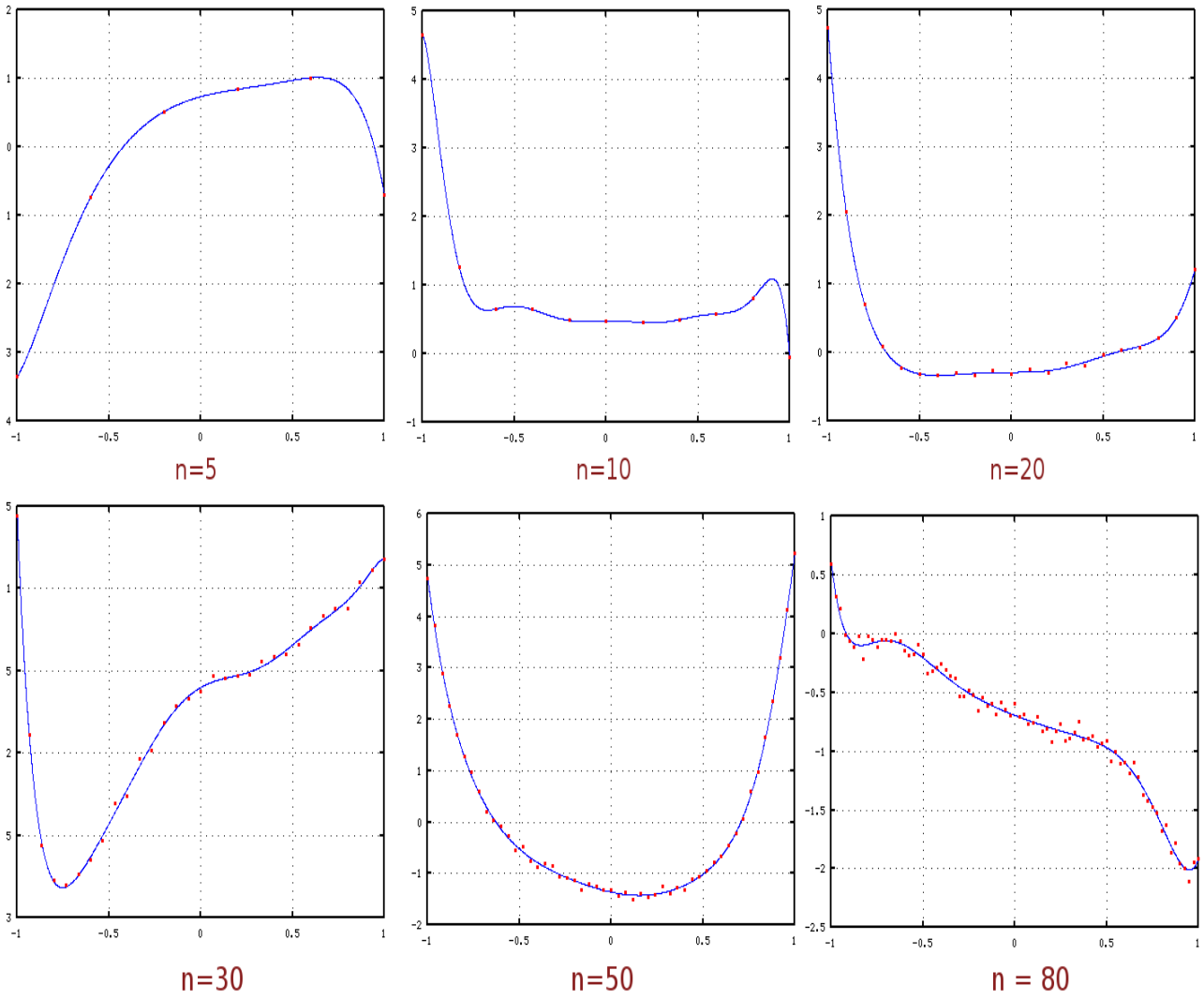


n = 80

On remarque dans le premier tableau que l'erreur ne varie pas dans un sens, c'est à dire qu'elle diminue, puis réaugmente ...
Cependant, dans le tableau ci-dessous, on remarque qu'à n fixé, valant 10, on a l'erreur qui augmente lorsque l'on augmente d allant de 1 à 10.Les valeurs supèrieures à 10 ne nous intéressent pas, en fait puisque d ne doit pas dépasser n.



d	5	10	20	27	34	40
condA	1633	$8,3.10^{6}$	$6.9.10^{14}$	$3,2.10^{17}$	$2,7.10^{18}$	$1,4.10^{18}$
erreur	$4,3.10^{-14}$	$1,3.10^{-9}$	884,6	$9,5.10^{7}$	$1,3.10^{12}$	$3,910^{15}$



– comparaison avec Polyfit : On calcule l'erreur $\|(pd - pf')\|$, on utilise le calcul de conditionnement, cond A définit par $condA = \|A\|\|A^{-1}\|$
Pour d=n,on remarque que la courbe et les points sont confondus,ce qui était d'avance prévisible.

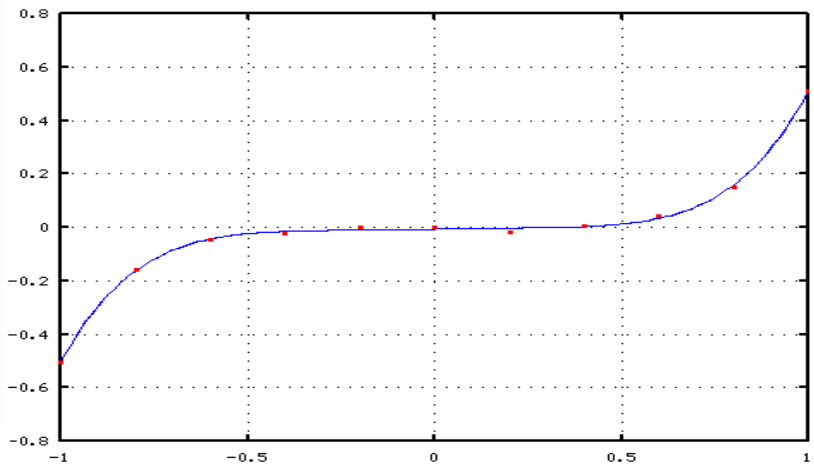
– Comparaison avec le travail d'Augustin et Murat
Leur script est beaucoup plus court que le nôtre :

mcd murat.m

```
function [p] = moindrescarresn(x, y, d)
% d degré du polynôme
lx = length(x);
ly = length(y);

if lx == ly
    % d+1 est la longueur d'une ligne/colonne de notre matrice

    for i = 1:d+1
        for j = 1:d+1
            A(i, j) = sum(x.^(2*d+2-i-j));
        end

        Y(i) = sum(y.*(x.^(d-i+1)));
    end
    A
    Y
    [p] = A\Y';
else
    error('Les vect x et y n ont pas la meme longueur');
end
```

*sd.m

```
n=10;
d=5 % on choisit le degré du polynôme
pas=2/n;
x=-1:pas:1;
y=0.5*x.^d + randn(1,n+1)*0.01;

pd=mcd(x,y,d);


h=1e-2
xx=min(x):h:max(x);
yy=polyval(pd,xx)

plot(xx,yy,x,y,'*r')
grid
```



On voit que leur groupe obtient presque la même chose que nous. Mais nous ne pourrons pas comparer nos travaux du fait qu'on a chacun utilisé la commande randn qui fait qu'à chaque exécution d'un même script sans modification des variables, on obtient des résultats différents sur l'approximation polynômiale au sens des moindres carrés.Cependant, nos méthodes paraissent tout de même similaires dans le script.