Compte rendu TP analyse numérique

TP1&TP2

NOM & PRÉNOM : **Meddeb Hadil &**

Laajili Hela

FILIÈRE : **FIA2-GL**

GROUPE : **G4**

ANNÉE UNIVERSITAIRE : **2021/2022**

Tp1 : Interpolation et approximation polynômiale

Méthode de Lagrange :

Application 1 :

- Dans cette application il nous demande de créer une **fonction Lagrange** qui retourne en sortie la valeur du polynôme d'interpolation de Lagrange $p_n(c)$.
- Le code suivant présente se ci :

lagrange.m

```
function yint =lagrange(n,x,y,c)
s=0;
for i=1:n
    l=1;
    for j=1:n
        if(i~=j)
            l=l*(c-x(j))/(x(i)-x(j));
        end
    end
    s=s+y(i).*l;
end
    yint=s;
end
```

Calcul de Lj(x)

S est la valeur de polynôme de

Application 2 :

Résolution analytique :

- Chercher analytiquement le polynôme $P_2(x)$: pour n = 0, 1, 2 et y =1, 2,5.

$$P_2(x) = y_0 \times L_0(x) + y_1 \times L_1(x) + y_2 \times L_2(x)$$

$$L_0(x) = \frac{x - x_1}{x_0 - x_1} \times \frac{x - x_2}{x_0 - x_2}$$

$$L_1(x) = \frac{x - x_0}{x_1 - x_0} \times \frac{x - x_2}{x_1 - x_2}$$

$$L_2(x) = \frac{x - x_0}{x_2 - x_0} \times \frac{x - x_1}{x_2 - x_1}$$

$$P_2(x) = y_0 \times \left(\frac{x-x_1}{x_0-x_1} \times \frac{x-x_2}{x_0-x_2}\right) + y_1 \times \left(\frac{x-x_0}{x_1-x_0} \times \frac{x-x_2}{x_1-x_2}\right) + y_2 \times \left(\frac{x-x_0}{x_2-x_0} \times \frac{x-x_1}{x_2-x_1}\right)$$

$$\Longrightarrow P_2(x) = 1 \times \left(\frac{(x-1)(x-2)}{(-1)(-2)}\right) + 2 \times \left(\frac{(x)(x-2)}{(1)(-1)}\right) + 5 \times \left(\frac{(x)(x-1)}{(2)(1)}\right)$$

$$\Longrightarrow P_2(x) = \left(\frac{(x^2-3x+2)}{2}\right) - 2 \times \left(\frac{(x^2-2x)}{1}\right) + 5 \times \left(\frac{(x^2-x)}{2}\right)$$

$$\Longrightarrow P_2(x) = \left(\frac{x^2 - \cancel{3x} + 2 - 4x^2 + \cancel{8x} + 5x^2 - \cancel{5x}}{2}\right)$$

Donc $P_2(x)$=x²+1

- En se basant sur les données fournit dans l'exercice nous allons créer un code qui permet de construire et de tracer les éléments de base de Lagrange L0 (x), L1 (x) et L2 (x).

```
clear all
clc
% t
close all
x=[0 1 2];
n=3;
y=[1 2 5];
interv=1000;
dx=(x(3)-x(1))/interv;
xvar=x(1):dx:x(3);
polyn=0;
col={'+k','+r','+m'};
for j=1:length(xvar)
    polyn(j)=lagrange(n,x,y,xvar(j));
end
figure(1)
hold on
for i=1:n
    plot(x(i),y(i),col{i},'MarkerSize',12,'LineWidth',2);
end
plot(xvar,polyn,'b');
hold off
grid
axis([-0.5 2.5 -0.5 5.5]);
title('Interpolation selon lagrange ')
coeff=polyfit(xvar,polyn,n-1);
```

Explication de code :

*__close all__ : supprimer les variables de espace de travaille actuel.

*__n,x,y,interv ,xvar__ se sont des variables en effet **n** est un polynôme de degré 2 , **interv** est un intervalle égale 1000et **xvar** une valeur qui varie entre(1,1001).

* **col={'+k','+r','+m'};** :chaque point x1,x2,x3 va être colorée par les couleurs choisie.

* **polyn(j)=Lagrange(n,x,y,xvar(j));** :appel de la fonction de lagrange pour calculer la valeur de polyn(j).

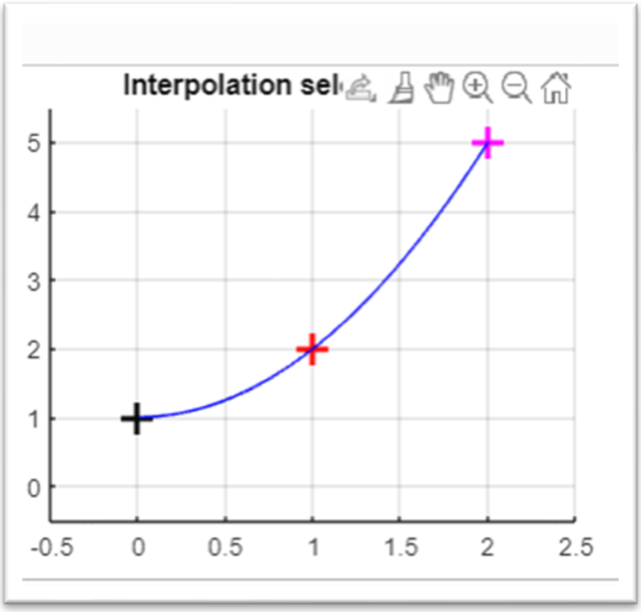
*__hold__ : sert à garder tous les lignes sur le graphique.

***plot** : sert à afficher dans notre cas elle va afficher le polynôme ainsi que les n+1 points.

***grid** : sert à afficher le graphique avec une gride.

***axis** : spécifie les limites des axes.

Exécution de code :



Interprétation :

La figure si dessus est une figure d'Interpolation selon Lagrange qui montre le polynôme de degré n = 2 interpolant relatifs aux points d'interpolations.

- Maintenant nous allons tracer le polynôme $P_2(x)$=**x²+1** et les points d'interpolation.

```
L=zeros(n,length(xvar));
for j=1:length(xvar)
    L(:,j)=lag_base(n,x,xvar(j));
end
figure (2)
hold on
plot(xvar,L(1,:),'r');
plot(xvar,L(2,:),'k');
plot(xvar,L(3,:),'b');
grid
hold off
```

Affichage graphique de L1, L2, L3 respectivement en

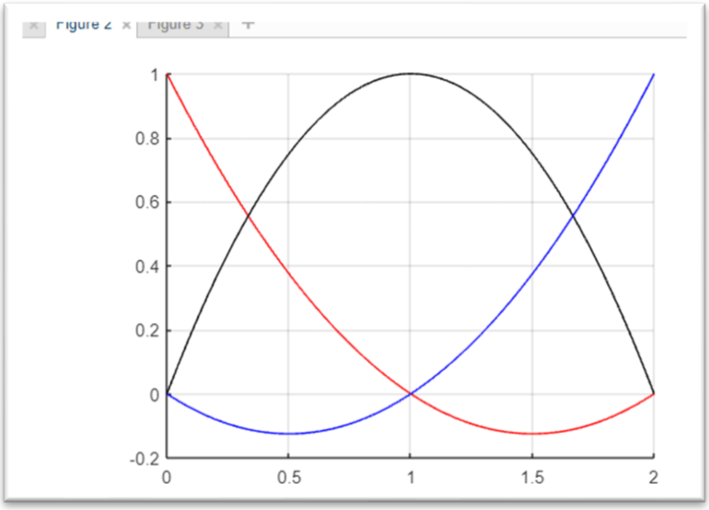
en rouge, noir et bleu

Explication de code :

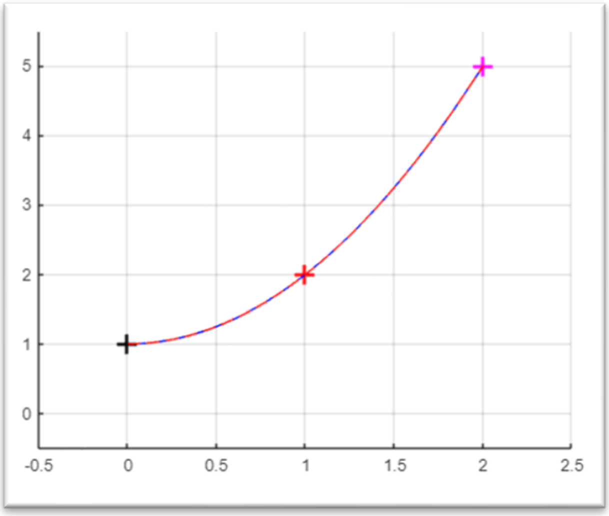
Renvoie une matrice de même taille que xvar comme le montre la première instruction puis l'appelle de la fonction **lag_base** pour calculer à chaque fois L et par suite afficher les courbes en utilisant le mot clé **plot.**

Exécution de code :

Voilà la figure que nous obtenons :



- Et voila sur la même figure on a le polynôme obtenu et les points d'interpolations.



Application 3 :

Résolution analytique :

$p(x) = L_0(x)\ y_0 + L_1(x)\ y_1 + L_2(x)\ y_2 + L_3(x)\ y_3 + L_4(x)\ y_4$

$p(x) = 0.034 \times \frac{(x-4)(x-8)(x-12)(x-18)}{1920} + 0.069 \times \frac{(x-4)(x-8)(x-12)(x-18)}{896} + 0.139 \times \frac{(x-4)(x-8)(x-12)(x-18)}{3360} + 0.020 \times \frac{(x-4)(x-8)(x-12)(x-18)}{6480} + 0.3 \times \frac{(x-4)(x-8)(x-12)(x-18)}{13440}$

- Traçage du graphe de fonction **f (x) =sin(x)** ainsi que de son polynôme **d'interpolation Pn (x).**

```
clear all
clc
close all
n=[2 4 7 11 14];
col_a={'+k','+r','+m','+y','+y','+c','+g','+b','+m','+k','+r','+m','+y','+y','+c','+g','+b','+m'};
col_b=col_a;
col=[col_a col_b];
E=zeros(1,length(n));
for i=1:length(n)
    x = linspace(0,3*pi,n(i));

    f = sin(x);
    interv=1000;
    dx=(x(length(x))-x(1))/interv;
    xvar=x(1):dx:x(length(x));
    f_xvar=sin(xvar);
    polyn=0;
    for j=1:length(xvar)
        polyn(j)=lagrange(n(i),x,f,xvar(j));

    end
figure(i)

hold on
for k=1:n(i)
    plot(x(k),f(k),col{k},'MarkerSize',12,'LineWidth',2);
end
plot(xvar,polyn,'--b');
plot(xvar,f_xvar,'--r');

hold off

grid
axis([-0.5 3*pi+1 -1.5 2.0])
title ('Interpolation selon Lagrange');
```

Déclaration des variables

Linspace :assure l'équdistance entre les nombres déterminé des pts entre 0 et 3*pi

Appelle de la fonction « lagrange »pour calculer l'interpolation de f de chaque élément xvar

Affichage de figure contenant un nombre de n(i) point

Affichage des points

Affiche la courbe de polynôme en bleu

Affichage de la courbe de f-xvar en rouge

```
coeff = polyfit(xvar, polyn, n(i)-1)
E(i)=max(abs(f_xvar-polyn));

end
figure(length(n)+1)
stairs(n,E,'b');
grid
title('Erreur d interpolation de Lagrange E_n pour des points equidistants')
```

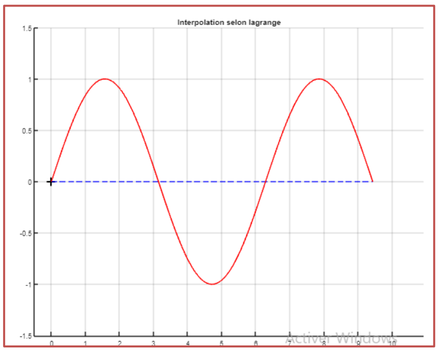
Polyfit : retourne la coefficient de d'interpolation

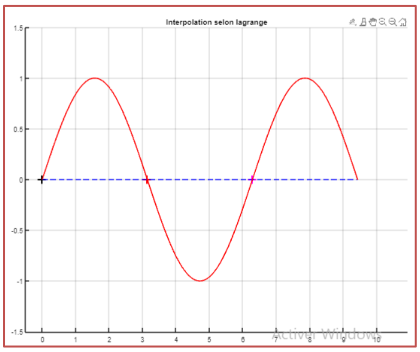
E(i) prend la valeur maximale entre f_xvar et polyn

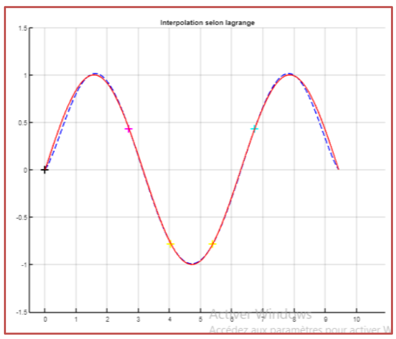
Vecteur d'erreur

➢ Cette partie de code concerne l'évaluation de l'erreur d'interpolation $E_n(x)$.

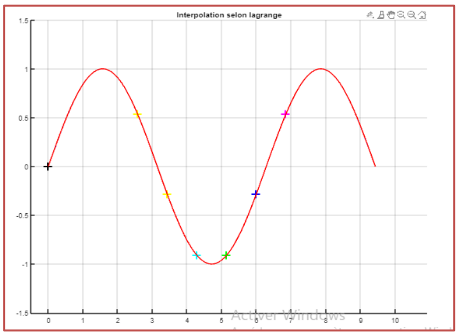
Résultat d'exécution du code :



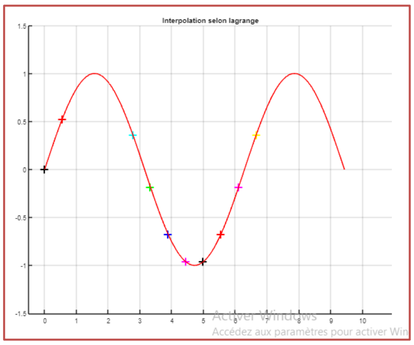




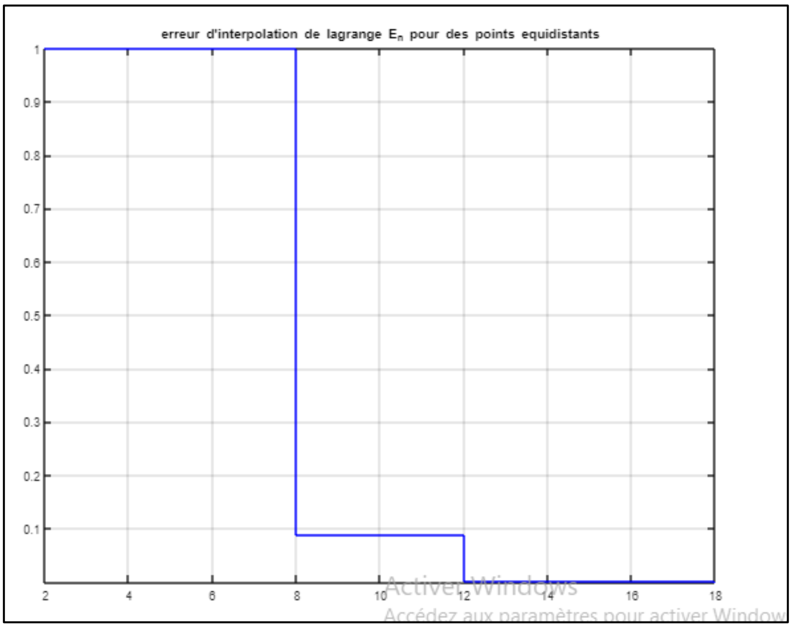
Interpolation selon Lagrange (n=2) Interpolation selon Lagrange (n=4) Interpolation selon Lagrange (n=8)



Interpolation selon Lagrange (n=12)



Interpolation selon Lagrange (n=18)



Erreur d'interpolation de Lagrange

- A partir de ses figures on peut remarquer que l'augmentation des points (n) entraine la diminution de l'erreur.

Application 4 :

- Dans cette application notre objectif est d'étudier le Phénomène de Runge.
- Le même code de l'application 3 va être utilisé dans cette application en remplaçant f(x) par g (x) $=\frac{1}{1+x^2}$ et n = 2, 4, 7, 11, 14.

Résolution analytique :

$$
\begin{aligned}
G_x = y_0 \times \left(\frac{(x-x_1)}{(x_0-x_1)} \times \frac{(x-x_2)}{(x_0-x_2)} \times \frac{(x-x_3)}{(x_0-x_3)} \times \frac{(x-x_4)}{(x_0-x_4)}\right) \\
+ y_1\left(\frac{(x-x_0)}{(x_1-x_0)} \times \frac{(x-x_2)}{(x_1-x_2)} \times \frac{(x-x_3)}{(x_1-x_3)} \times \frac{(x-x_4)}{(x_1-x_4)}\right) \\
+ y_2\left(\frac{(x-x_0)}{(x_2-x_0)} \times \frac{(x-x_1)}{(x_2-x_1)} \times \frac{(x-x_3)}{(x_0-x_3)} \times \frac{(x-x_4)}{(x_0-x_4)}\right) \\
+ y_3\left(\frac{(x-x_0)}{(x_3-x_0)} \times \frac{(x-x_1)}{(x_3-x_1)} \times \frac{(x-x_2)}{(x_3-x_2)} \times \frac{(x-x_4)}{(x_3-x_4)}\right) \\
+ y_4\left(\frac{(x-x_0)}{(x_4-x_0)} \times \frac{(x-x_1)}{(x_4-x_1)} \times \frac{(x-x_2)}{(x_4-x_2)} \times \frac{(x-x_3)}{(x_4-x_3)}\right)
\end{aligned}
$$

$$G_x = y_0 \times \left(\frac{(x-4)}{(2-4)} \times \frac{(x-7)}{(2-7)} \times \frac{(x-11)}{(2-11)} \times \frac{(x-14)}{(2-14)}\right)$$
$$+ y_1 \left(\frac{(x-2)}{(4-2)} \times \frac{(x-7)}{(4-7)} \times \frac{(x-11)}{(4-11)} \times \frac{(x-14)}{(4-14)}\right)$$
$$+ y_2 \left(\frac{(x-2)}{(7-2)} \times \frac{(x-4)}{(7-4)} \times \frac{(x-11)}{(7-11)} \times \frac{(x-14)}{(7-11)}\right)$$
$$+ y_3 \left(\frac{(x-2)}{(11-2)} \times \frac{(x-4)}{(11-4)} \times \frac{(x-7)}{(11-7)} \times \frac{(x-14)}{(11-14)}\right)$$
$$+ y_4 \left(\frac{(x-2)}{(14-2)} \times \frac{(x-4)}{(14-4)} \times \frac{(x-7)}{(14-7)} \times \frac{(x-11)}{(14-11)}\right)$$

$$G_x = y_0 \times L_0(x) + y_1 \times L_1(x) + y_2 \times L_2(x) + y_3 \times L_3(x) + y_4 \times L_4(x)$$

$$L_0(x) = \frac{(x-x_1)}{(x_0-x_1)} \times \frac{(x-x_2)}{(x_0-x_2)} \times \frac{(x-x_3)}{(x_0-x_3)} \times \frac{(x-x_4)}{(x_0-x_4)}$$

$$L_1(x) = \frac{(x-x_0)}{(x_1-x_0)} \times \frac{(x-x_2)}{(x_1-x_2)} \times \frac{(x-x_3)}{(x_1-x_3)} \times \frac{(x-x_4)}{(x_1-x_4)}$$

$$L_2(x) = \frac{(x-x_0)}{(x_2-x_0)} \times \frac{(x-x_1)}{(x_2-x_1)} \times \frac{(x-x_3)}{(x_0-x_3)} \times \frac{(x-x_4)}{(x_0-x_4)}$$

$$L_3(x) = \frac{(x-x_0)}{(x_3-x_0)} \times \frac{(x-x_1)}{(x_3-x_1)} \times \frac{(x-x_2)}{(x_3-x_2)} \times \frac{(x-x_4)}{(x_3-x_4)}$$

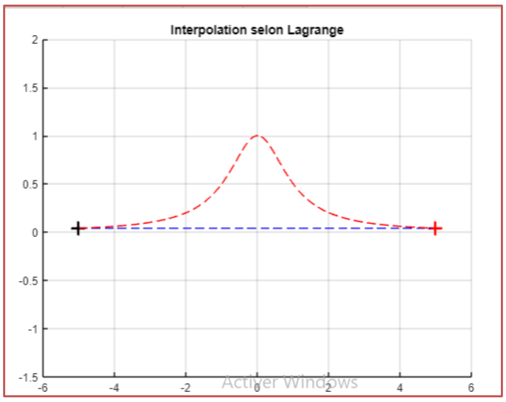
$$L_4(x) = \frac{(x-x_0)}{(x_4-x_0)} \times \frac{(x-x_1)}{(x_4-x_1)} \times \frac{(x-x_2)}{(x_4-x_2)} \times \frac{(x-x_3)}{(x_4-x_3)}$$

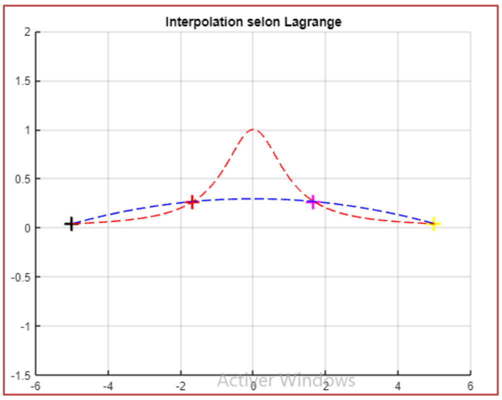
Résultat d'exécution du code :

Les coefficients de polynôme d'interpolation :

```
coeff =

   -0.0000    0.0385


coeff =

    0.0000   -0.0102   -0.0000    0.2930


coeff =

   -0.0008   -0.0000    0.0335    0.0000   -0.3514   -0.0000    1.0000


coeff =

   -0.0000   -0.0000    0.0013    0.0000   -0.0244   -0.0000    0.1974    0.0000   -0.6742   -0.0000    1.0000


coeff =

   -0.0000    0.0000    0.0000   -0.0001   -0.0000    0.0027    0.0000   -0.0325   -0.0000    0.1994    0.0000   -0.6107   -0.0000    0.9572
```

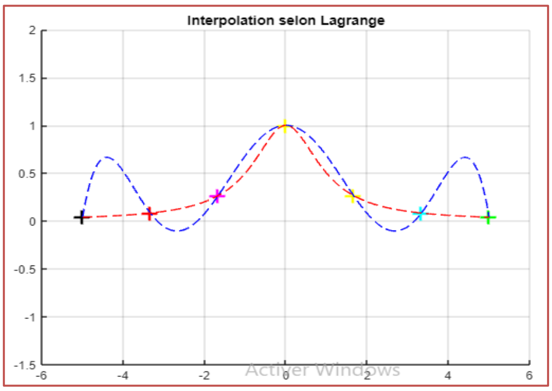
TP analyse numérique

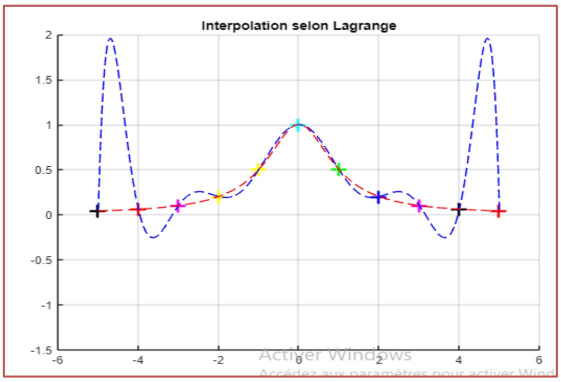




Interpolation selon Lagrange (n=2)

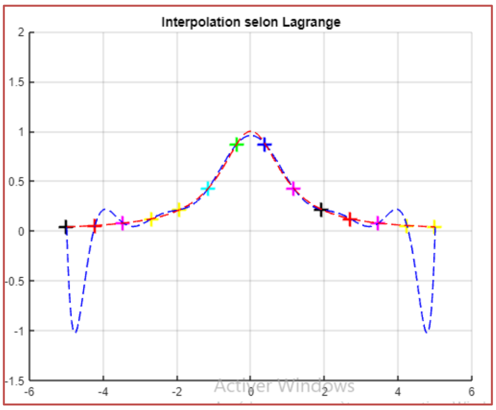
Interpolation selon Lagrange (n=4)



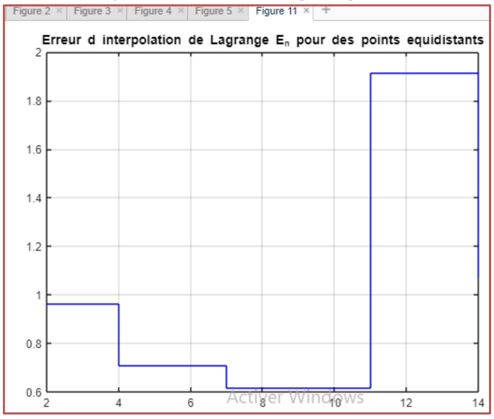


Interpolation selon Lagrange (n=7)

Interpolation selon Lagrange (n=11)



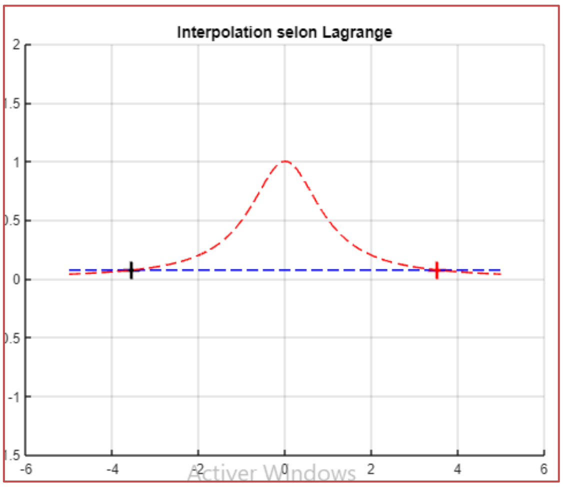
Interpolation selon Lagrange (n=14)

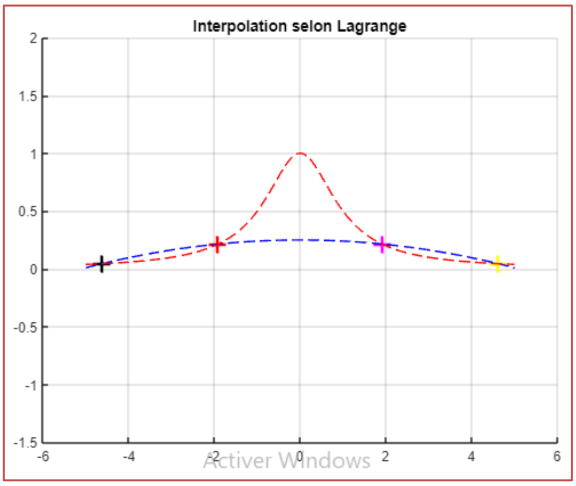


Erreur d'interpolation de Lagrange

- A partir des résultats obtenu on peut affirmer que l'erreur dépend de fonction choisie et non pas de n ce qui rend l'hypothèse que l'erreur dépend de n n'est pas toujours vraie.
    - Refaire le même travail pour les points de **Tchebychev**

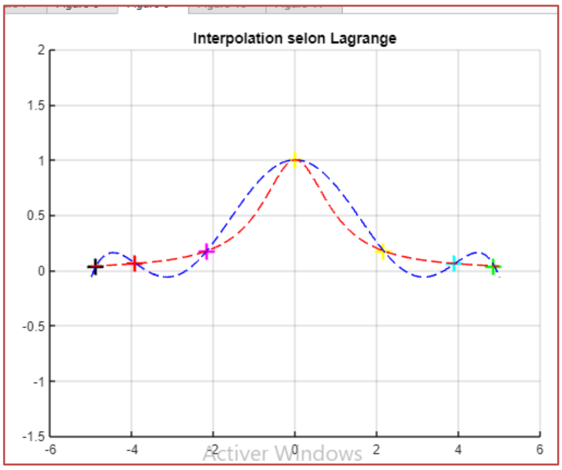
Résultat d'exécution selon Tchebychev:

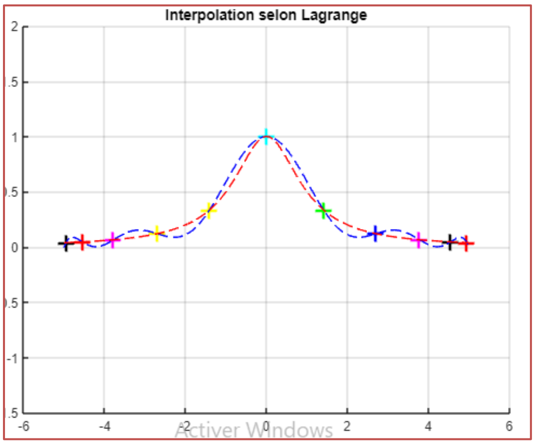




Interpolation selon Lagrange (n=2)

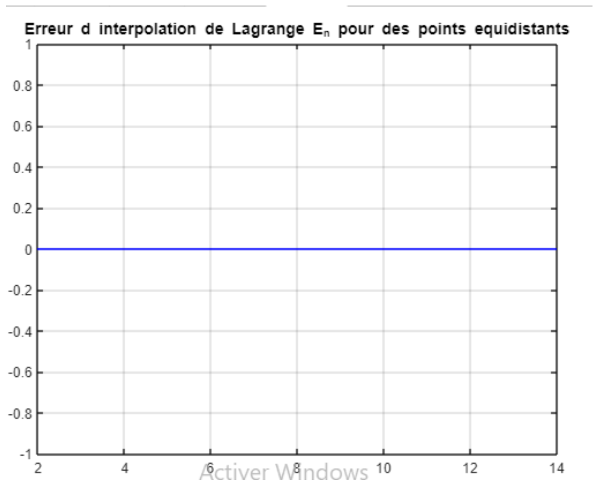
Interpolation selon Lagrange (n=4)





Interpolation selon Lagrange (n=7)

Interpolation selon Lagrange (n=11)



Erreur d’interpolation de Lagrange pour les points de Tchebychev

```
coeff =

  -0.0000    0.0741


coeff =

   0.0000   -0.0096   -0.0000    0.2497


coeff =

  -0.0004   -0.0000    0.0195    0.0000   -0.2572   -0.0000    1.0000


coeff =

  -0.0000   -0.0000    0.0003    0.0000   -0.0085   -0.0000    0.0983    0.0000   -0.4991   -0.0000    1.0000


coeff =

  -0.0000    0.0000    0.0000   -0.0000   -0.0000    0.0005    0.0000   -0.0090   -0.0000    0.0833    0.0000   -0.3929   -0.0000    0.8766
```

Les coefficients de polynôme d'interpolation selon Tchebychev

- Grace aux résultats obtenus en peut conclure que pour avoir une erreur minimale il faut choisir comme point d’interpolation les points de Tchebychev.

<u>Méthode de Newton :</u>

- Nous allons calculer les polynômes en utilisant la méthode de Newton

Application 1 :

- Écrire une fonction Coefficient

```
function d = Coefficient( n,x,y )
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
D=zeros(n,n);
D(:,1)=y';
for j=2:n
    for i=j:n
        D(i,j)=(D(i,j-1)-D(i-1,j-1))/(x(i)-x(i-j+1));
    end
end
d=diag(D);
end
```

- Écrire une fonction Newton

```
function P = Newton(n,x,c,d)
s=d(1);
for i=2:n
    base=1;
    for k=1:i-1
        base=base*(c-x(k));
    end
    s=s+base*d(i);
    P=s;
end
end
```

Application 2 :

Résolution analytique :

x-i	f(x_i)	f[x_i, x_{i+1}]	f[x_i, x_{i+1}, x_{i+2}]	
4	1.59			
6	1.817	$\frac{1.817-1.59}{6-4}=0.1135$		
8	2	$\frac{2-1.817}{8-6}=0.0915$	$\frac{0.0915-0.1135}{8-4}=-0.0055$	
10	2.1544	$\frac{2.1544-2}{10-8}=0.0772$	$\frac{0.0772-0.0915}{10-6}=-0.0035$	$\frac{-0.0035+0.0055}{10-4}=0.0003$

Donc selon Newton le polynôme est le suivant :

$P_3(x)$= 1.59 + 0.1135(x – 4) + 0.0055(x – 4)(x – 6) + 0.0003(x – 4)(x – 6)(x – 8)

- Développer un code qui permettant de construire et de tracer les éléments de base de Newton.

```
function L_base=lag_base(n,x,a)
L_base=[];
for i=1:n
    L=1;
    for j=1:n
        if(i~=j)
            L=L.*(a-x(j))./(x(i)-x(j));
        end
    end
    L_base=[L_baseL];
end
end
```

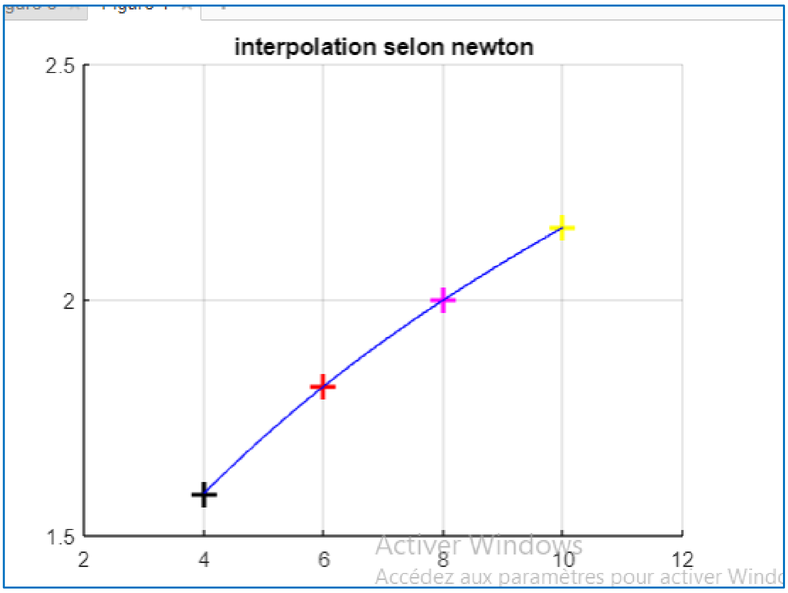
- Code pour tracer le polynôme $P_3(x)$ et les points d'interpolation.

```
clear all
close all
clc

x(1)=4;x(2)=6;x(3)=8;x(4)=10;
n=4 ;
y(1)=1.59; y(2)=1.817; y(3)=2; y(4)=2.1544;
interv=1000;
dx=(x(4)-x(1))/interv;
xvar = x(1):dx:x(4);
polyn=0;
col={'+k','+r','+m','+y'};
d=Coefficient(n,x,y);
for j=1:length(xvar)
    polyn(j)=Newton(n,x,xvar(j),d);
end
figure(1)
hold on
for i=1:n
    plot(x(i),y(i),col{i},'MarkerSize',12,'LineWidth',2);
end

plot(xvar,polyn,'b');
hold off
grid
axis([2 12 1.5 2.5])
title('interpolation selon newton')
coeff=polyfit(xvar,polyn,n-1)
```

Résultat d'exécution du code :

Les coefficients de polynôme d'interpolation :

```
coeff =

    0.0003   -0.0113    0.2019    0.9424
```



Interpolation selon Newton

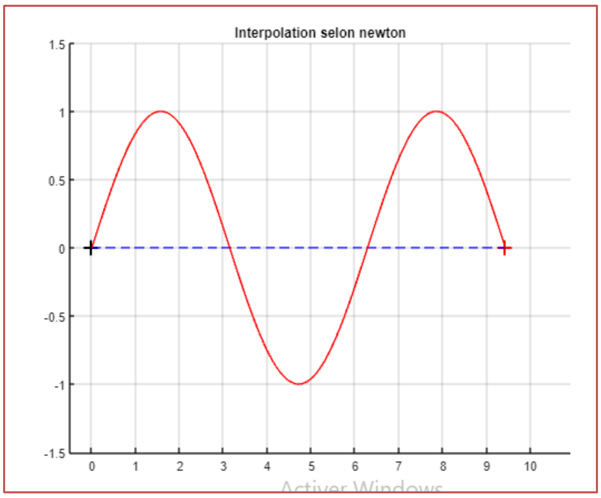
Application 3 :

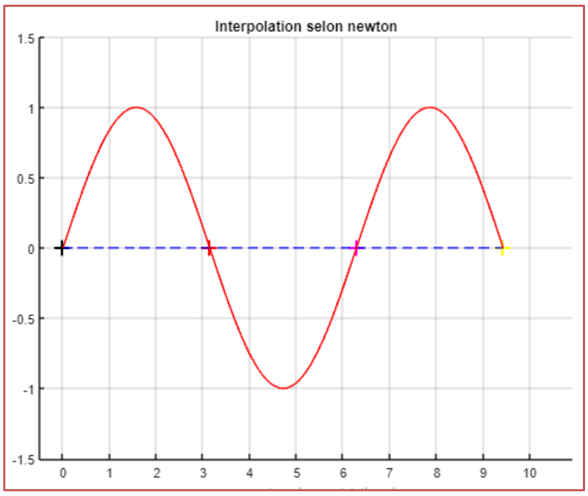
- Traçage du graphe de fonction f (x) ainsi que son polynôme d'interpolation $P_n(x)$ en utilisant les fonctions : tic et toc ; voici le code que fait se ci :

Retourne le temps d'exécution de l'interpolation

```
clear all
clc
tic
close all
n=[ 2 4 8 12 18];
col_a={'+k','+r','+m','+y','+y','+c','+g','+b','+m','+k','+r','+m','+y','+y','+c','+g','+b','+m'};
col_b=col_a;
col=[col_a col_b];
E=zeros(1,length(n));
for i=1:length(n)
    x = linspace(0,3*pi,n(i));
    f=sin(x);
    interv=1000;
    dx=(x(length(x))-x(1))/interv;
    xvar=x(1):dx:x(length(x));
    f_xvar=sin(xvar);
    polyn=0;
    d=Coefficient(n(i),x,f);
    for j=1:length(xvar)
        polyn(j)=Newton(n(i),x,xvar(j),d);
    end
    figure(i)
    hold on
    for k=1:n(i)
        plot(x(k),f(k),col{k},'Markersize',12,'LineWidth',2);
    end
    plot(xvar,polyn,'--b');
    plot(xvar,f_xvar,'r');
    hold off
    grid
    axis([-0.5 3*pi+1.5 -1.5 1.5]);
    title('Interpolation selon lagrange');
    coeff= polyfit(xvar,polyn,n(i)-1) ;
    E(i)= max(abs(f_xvar-polyn));
end
figure(length(n)+1);
stairs(n,E,'b');
grid
title('erreur de l''interpolation de lagrangre');
toc
```

Résultat d'exécution du code :

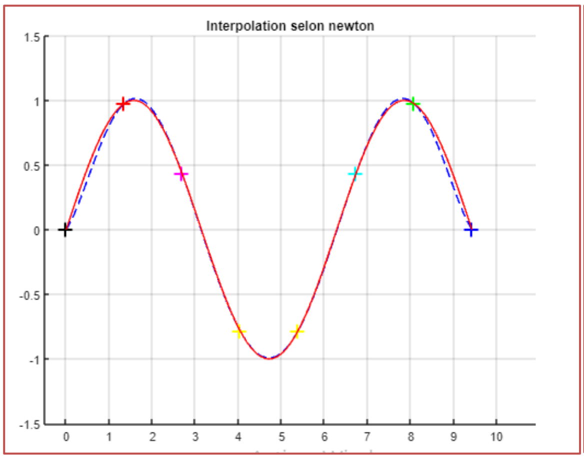




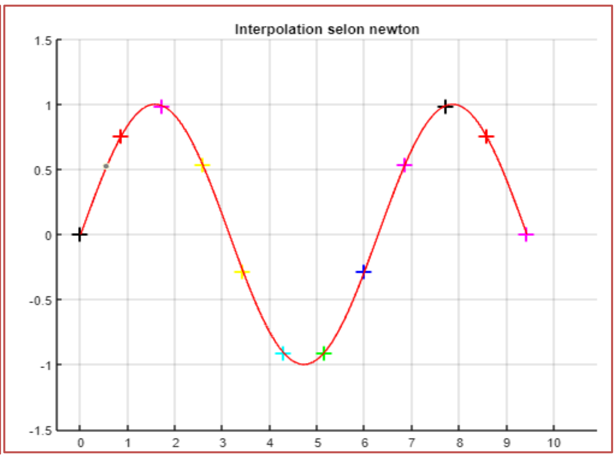
Interpolation selon Newton (n=2)

Interpolation selon Newton (n=4)

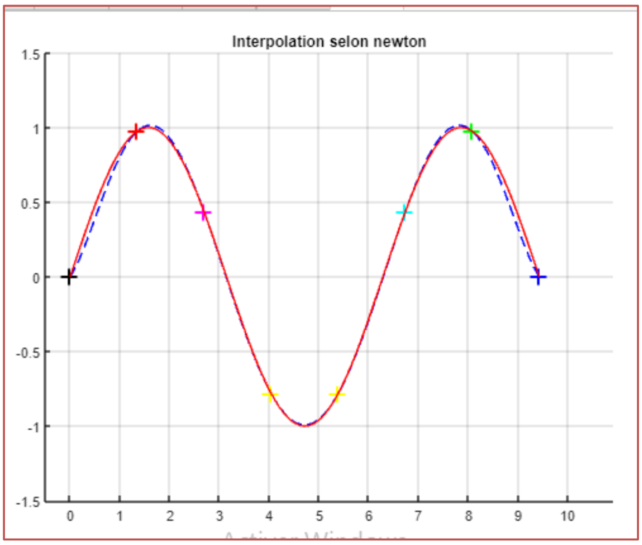
TP analyse numérique



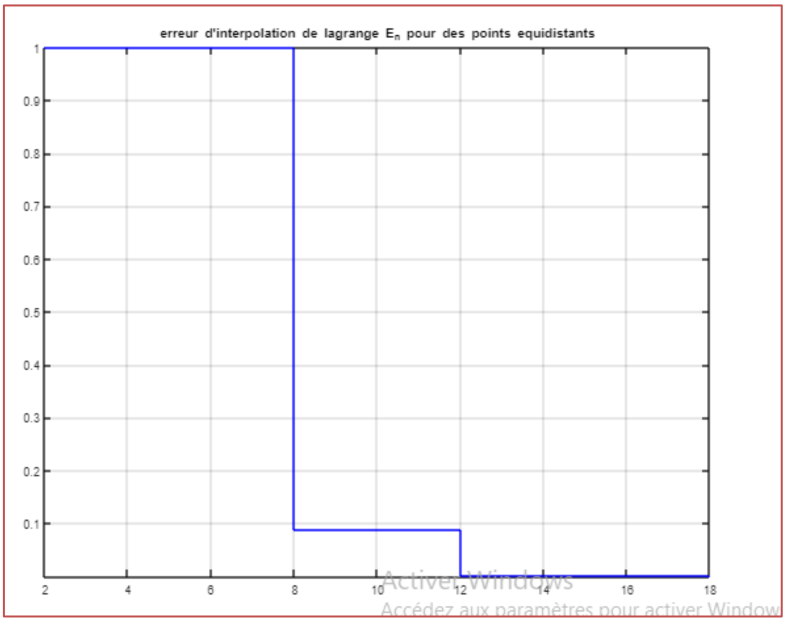
Interpolation selon Newton (n=8)



Interpolation selon Newton (n=12)



Interpolation selon Newton (n=18)



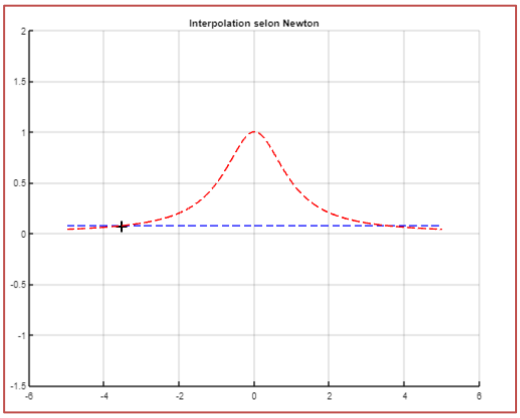
Erreur d'interpolation selon Newton

- On peut conclure que le temps d'exécution d'interpolation de Newton est rapide.
    - Code et exécution des questions de la section 1.2.4 selon Newton.

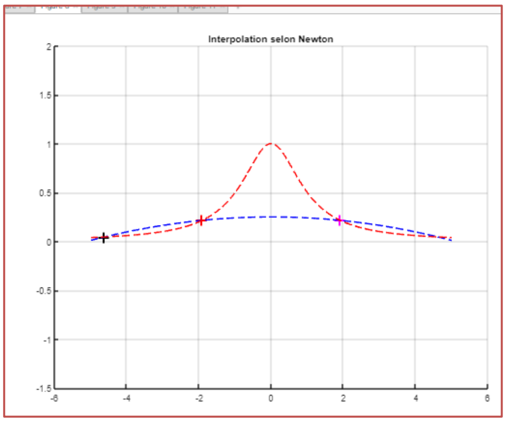
```
close all
clc
n=[2 4 7 11 14];
col_a={'+k','+r','+m','+y','+y','+c','+g','+b','+m','+k','+r','+m','+y','+y','+c','+g','+b','+m'};
col_b=col_a;
col=[col_a col_b];
E=zeros(1,length(n));
for i=1:length(n)
    x=linspace(-5,5,n(i));
    x_c=Cheby_Points(n(i),-5,5);
    f_c=1./(1+x_c.^2);
    interv=1000;
    dx=(x(length(x))-x(1))/interv;
    xvar=x(1):dx:x(length(x));
    f_xvar=1./(1+xvar.^2);
    polyn=0;
    d = Coefficient(n(i),x_c,f_c);
    for j=1:length(xvar)
        polyn_c(j)=Newton(n(i),x_c,xvar(j),d);
    end
    figure (length(n)+1+i)
    hold on
    for k=1:n(i)
        plot(x_c(k),f_c(k),col{k},'MarkerSize',12,'LineWidth',2);
    end
    plot (xvar,polyn_c,'--b');
    plot(xvar,f_xvar,'r');
    hold off
    grid
    axis ([-6.0 6.0 -1.5 2.0]);
    title ('interpolation selon Lagrenge ');
    coeff =polyfit(xvar,polyn_c,n(i)-1);

    E_1(i)=max(abs(f_xvar-polyn_c));
end
figure (2*(length(n)+1))
stairs(n,E_1,'b');
grid
title('erreur d''interpolation de Lagrenge E_n')
```

```
   -0.0007
    0.0003
   -0.0001
    0.0000
   -0.0000


coeff =

   -0.0000   -0.0000    0.0003    0.0000   -0.0085   -0.0000    0.0983    0.0000   -0.4991   -0.0000    1.0000


d =

    0.0389
    0.0162
    0.0056
    0.0020
    0.0009
    0.0005
    0.0001
   -0.0003
    0.0002
   -0.0001
    0.0000
   -0.0000
    0.0000
   -0.0000


coeff =

    0.0000    0.0000   -0.0000   -0.0000    0.0000    0.0005   -0.0000   -0.0090    0.0000    0.0833   -0.0000   -0.3929    0.0000    0.8766

>>
```

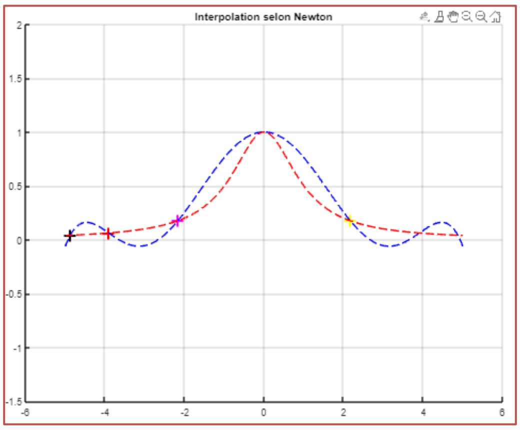
Les coefficients de polynôme d'interpolation



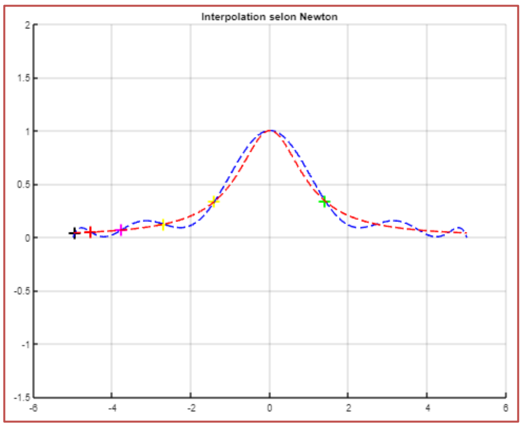
Interpolation selon Lagrange (n=2)



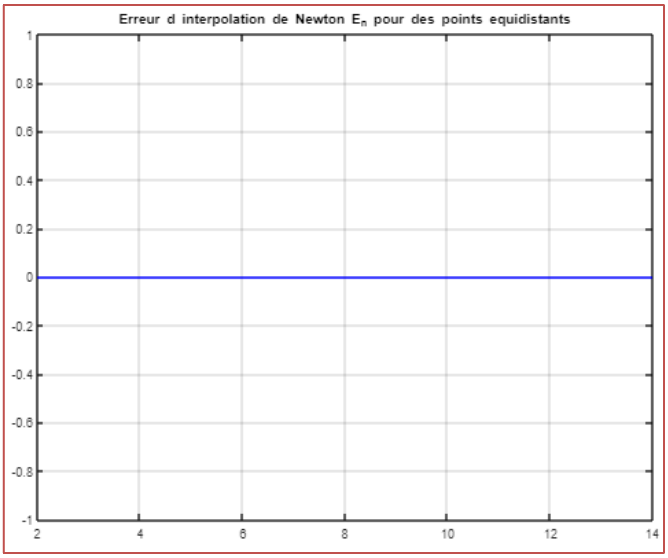
Interpolation selon Lagrange (n=4)



Interpolation selon Lagrange (n=7)



Interpolation selon Lagrange (n=11)



Erreur d’interpolation selon Newton pour les points de Tchebychev

- On comparant l’interpolation de Newton par l’interpolation de Lagrange on peut affirmer que l’interpolation selon Newton est meilleure de celle selon Lagrange au niveau de temps d’exécution.

Méthode de Hermite:

- Implémentation de la méthode de Hermite et détermination de polynôme :

```
function yint = Hermite(n,x,y,yder,c)
s=0;

for i=1:n
lag=1;ci=0;
for j=1:n
if (i~=j)
lag=(c - x(j))./(x(i) - x(j)).*lag;
ci=ci +(1/(x(i) -x(j)));
end
end

Di=1-2.*(c - x(i)).*ci;

s=s + (y(i).*Di + yder(i).*(c -x(i))).*(lag).^2;


yint=s;

end
```

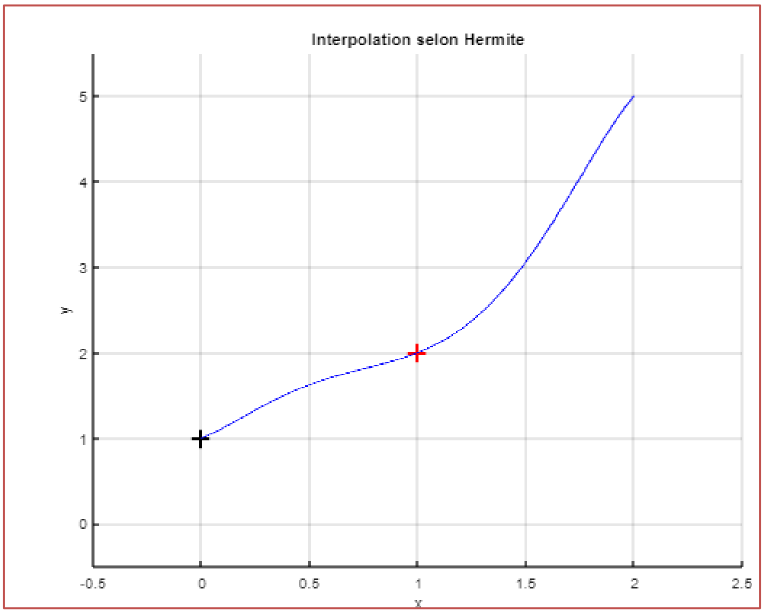
- Traçage le polynôme obtenu et les points d’interpolation :

```
clear all
clc
close all
x=[0, 1, 2]; n=3;
y=[1, 2, 5]; interv=1000; dx=(x(3)-x(1))/interv;

yder=[y(1),(y(2)-y(1))/(x(2)-x(1)),(y(3)-y(2))/(x(3)-x(2))];
xvar=x(1):dx:x(3);polyn=0;
col={'+k','+r','+m'};
for i=1:n
    lag=1;ci=0;
    for j=1:n
        if (i~=j)
            lag=(xvar - x(j))./(x(i) - x(j)).*lag;
            ci=ci+(1/(x(i) - x(j)));
        end
    end
    Di=1-2.*(xvar - x(i)).*ci;
    polyn=polyn + (y(i).*Di + yder(i).*(xvar -x(i))).*(lag).^2;
    figure(1)
    hold on
    plot(x(i),y(i),col{i},'MarkerSize',12,'LineWidth',2);
end
plot(xvar,polyn,'b','LineWidth',1);
hold off
grid
xlabel('x');
ylabel('y');
axis([-0.5 2.5 -0.5 5.5])
title('Interpolation selon Hermite')
coeff = polyfit(xvar,polyn,2*(n-1)+1)
evalp=polyval(coeff,xvar);
```

Polynôme de degré 2(n-1) +1

```
coeff =

  Column 1

   -1.0000

  Column 2

    4.5000

  Column 3

   -6.0000

  Column 4

    2.5000

  Column 5

    1.0000

  Column 6

    1.0000
```

Les coefficients de polynôme d'interpolation



Interprétation :

Après avoir déterminer le polynôme d’Hermite et exécuter le code créé en peut conclure que Hermite est meilleure que Newton en interpolation.

Tp2 : Intégration numérique

- Dans ce TP nous allons étudier les méthodes d'intégrations numériques ; étant la méthode des rectangle à gauche , méthode de rectangle à droite , la méthode des point milieu ,Simpson, et trapèze, on va les comparer tous afin de trouver la meilleur méthode qui nous permet d'approcher la valeur de l'intégrale d'une fonction f(x) sur un intervalle [a,b].
- ***Méthode de rectangle :***

Application1 :

- Création de fonction de méthode rectangle à gauche prenant en paramètre : la fonction les bornes de l'intégrale et le nombre de subdivisions :

```
function I=rectD(fun,lowerBound,upperBound,n)

dx=(upperBound - lowerBound)/n;
x=lowerBound:dx:upperBound;
int=0;

 for i=1:length(x)-1
xbar=x(i+1);

int=int+dx*fun(xbar);

 end
 I=int;
end
```

- Création de fonction de méthode rectangle à droite : prenant en paramètre : la fonction les bornes de l'intégrale et le nombre de subdivisions :

```
function I=rectG(fun,lowerBound,upperBound,n)

dx=(upperBound - lowerBound)/n;
x=lowerBound:dx:upperBound;
int=0;

 for i=1:length(x)-1
xbar=x(i);

int=int+dx*fun(xbar);

 end
 I=int; %somme
end
```

- Création de fonction de méthode rectangle au milieu : prenant en paramètre : la fonction les bornes de l'intégrale et le nombre de subdivisions :

```
function y=point_milieu(fun,lowerBound,upperBound,n)

dx=(upperBound - lowerBound)/n;
x=lowerBound:dx:upperBound;
int=0;

 for i=1:length(x)-1
xbar=(x(i)+ x(i+1))/2;

int=int+dx*fun(xbar);

 end
 y=int;
end
```

- Créer le code pour comparer entre les trois méthodes de rectangle

```
clear all;close all;clc;

%%%%%% M�thode des rectangles: Application

syms x

f=exp(sin(x));
I=int(f,0,5);
f_1=diff(f,x);
f_2=diff(f_1,x);
t=0:0.01:5;
y=exp(sin(t)).*cos(t).^2 - exp(sin(t)).*sin(t);
M=max(y)

N=[50 100 150 200 250 300 350 400 450 500 ];
E_N_G=zeros(1,length(N)); E_N_D=zeros(1,length(N)); E_N_M=zeros(1,length(N));
I_1=zeros(1,length(N)); I_2=zeros(1,length(N)); I_3=zeros(1,length(N));E_N_test=zeros(1,length(N));
for i=1:length(N)
I_1 (i)=rectG(@(t) exp(sin(t)),0,5,N(i));
I_2 (i)=rectD(@(t) exp(sin(t)),0,5,N(i));
I_3 (i)=point_milieu(@(t) exp(sin(t)),0,5,N(i));
E_N_G (i)=abs(I-I_1 (i));E_N_D (i)=abs(I-I_2 (i));E_N_M (i)=abs(I-I_3 (i));
E_N_test (i)=5^3/(24*(N(i))^2)*M;
end
figure
hold on
stairs(N,E_N_G,'b');
stairs(N,E_N_D,'--r');
stairs(N,E_N_M,'--k');
axis([ 50 500  -0.005 0.04])
hold off
grid
title('erreur d''integration')
```

f-1 sert à calculer le dérivé premier de la fonction f.

f-2 sert à calculer le dérivé second de la fonction f.

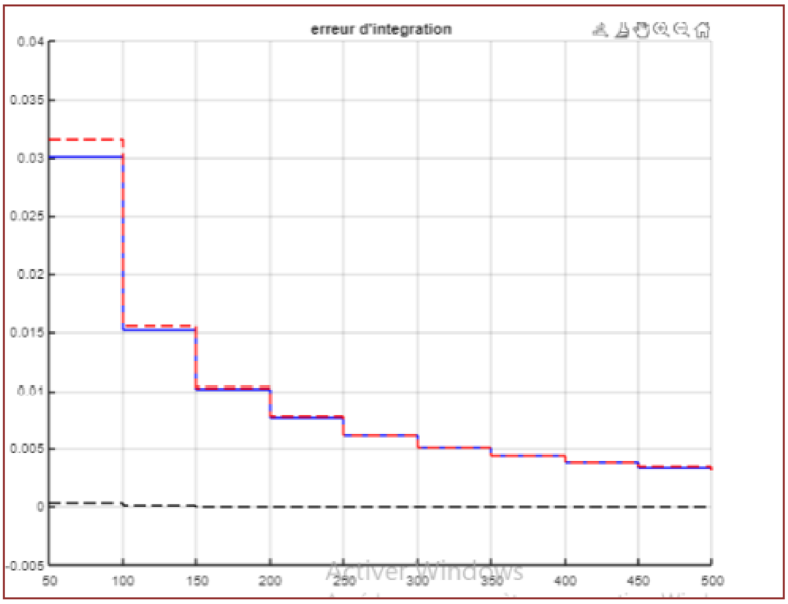
Les Fonctions utilisés :

Figure → ouvrir une nouvelle fenêtre de figure et afficher par la suite la figure courante.
Stairs → traçage de la courbe.
Axis → pour régler et préciser les limites d'axe (de 50 à 500 su l'axe des abscisses et de -0.005 au 0.04 sur l'axe des y) .
Grid → permet d'activer et désactiver les lignes de la grille.
Zéros → permet de générer une matrice des zéros.
Title → préciser le titre de la figure qui est dans notre cas (erreur d'integration)

`F = exp(sin(x))` → définir la fonction f
`I = int(f,0,5)` → calcule l'intégrale définie de f de 0 à 5
`M = max(y)` →retourne l'élément maximale de y.
`Abs(I_I_1(i))` → retourne la valeur absolu de chaque élément du tableau (I_I_1) .

1- →Appel des fonctions de rectangle à droite, à gauche, au milieu pour calculer l'intégrale de f entre a=0 et b=5 par chacun des trois méthodes.
2→ traçage des courbes d'erreur d'intégration pour les trois méthodes :
E_N_G :erreur d'intégration par la méthode de rectangle à gauche elle tracé en bleu('b').
E_N_D :erreur d'intégration par la méthode de rectangle à droite elle sera tracé en rouge ('r').
E_N_M :erreur d'intégration par la méthode de rectangle au point milieu elle sera tracé en noir ('k').
3→ on calcule l'erreur d'intégration de chaque méthode en faisant la différence entre la valeur réel de l'intégrale calculée auparavant I = Int(f,0,5) et la valeur de l'intégration calculé par la méthode

Exécution de code :



Erreur d'intégration E_n(f)

- Vérification de l'erreur de l'intégration

```
%%%test d'inégalité
E_N_test - E_N_M
```

Résultat d'exécution :

```
ans =

    0.0017    0.0004    0.0002    0.0001    0.0001    0.0000    0.0000    0.0000    0.0000    0.0000
```

Interprétation :

A partir du résultat obtenu on observe sur l'exemple que la méthode de rectangle au point milieu donne une meilleure précision que les méthodes de rectangles à droite et à gauche. Puisque sa courbe d'erreur d'intégration est toujours fixé à zéro ce qui signifie que l'approximation de l'intégrale calculé par cette méthode est presque confondu avec la valeur réel de l'intégrale .

Méthode de trapèze :

Application1 :

- Création de la fonction trapèze

```
function int=methode_trapeze(fun,lowerBound,upperBound,n)

dx=(upperBound - lowerBound)/n;
x=lowerBound:dx:upperBound;
int=0;const=(fun(1)+fun(n+1))*(dx/2);
for i=2:length(x)-1

    int=int+dx*fun(x(i));

end
int=const+int;
end
```

Explication :

lowerBound : borne inferieur de l'intégrale

UpperBound : borne supérieur de l'intégrale.

N : nombre de subdivision de l'intervalle .

Fun : notre fonction.

1 : dx reçoit la longueur d'une subdivision de l'intégrale il s'agit de n subdivisions tous égaux.

2 : const reçoit la somme de la dernière + la première valeur de f le tous est multiplié par h = dx = la longueur d'une subdivision

3 : par une boucle for on fait la sommation de tous les f(x(i)) multiplié chacune par la longueur des subdivisions dx, x(i) étant une abscisse obtenu lors de la division de l'intégrale sur n mini intervalles, par la suite on additionne la somme obtenu avec const calculé au début pour obtenir ainsi la valeur de l'intégrale entre lowerBound et upperBound par la méthode des trapèzes.

- Calcul d'erreur d'intégration selon la méthode de trapèze

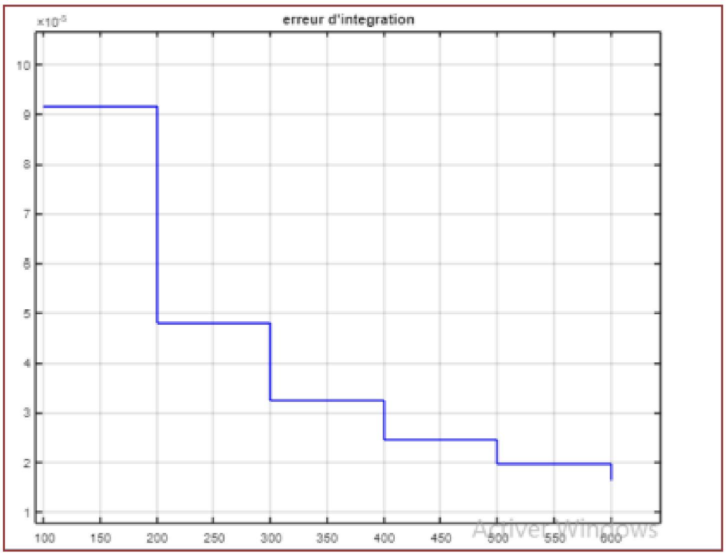
```
clear all;close all;clc;
%%%%%% M�thode des Trap�zes
%%%%%% Application 1
 syms x
f=1/sin(x)*exp(-x^2);
I=int(f,1,2);N=[100  200  300  400  500 600];
E_N_T=zeros(1,length(N));
I_1=zeros(1,length(N));
for i=1:length(N)

I_1(i)=trapeze(@(t) (1/sin(t))*exp(-t^2),1,2,N(i));
E_N_T (i)=abs(I-I_1 (i));
end
figure
stairs(N,E_N_T,'b');
grid
title('erreur d''integration')
```

Explication du code :

On a tout d'abord déclaration de la fonction f = 1/sin(x) *exp(-x²) , par la suite on calcule l'intégrale de f entre 1 et 2 par la méthode Int prédéfinie , par la suite on a par zéros(1,length(N)) on obtient un tableau des zeros de taille = length (N) =6 qui sera stocké dans E_N_T et on fait le même pour I_1 puis par une boucle for de 1 jusqu'à la longueur de N on remplit le tableau des zeros I_1 ; chaque case de I_1 va recevoir la valeur d'intégration de f par la méthode de trapèze mais à chaque fois le nombre de subdivision va varier selon le tableau N par exple pour i=1 on a N =100 alors la première case de I_1 va recevoir la valeur d'intégration de f entre 1 et 2 en divisant l'intervalle sur 100 subdivisions tous égaux .

En parallèle on calcule la valeur d'erreur d'intégrations pour chaque case de I_1 et on la stocke dans une case de même ordre dans la table E_N_T ce calcule d'erreur = la valeur abs de la différence entre I (valeur réel de l'intégration) et la valeur d'intégrations de la case i de la table I_1 ;
On trace notre courbe en bleu (stairs(N,E_N_T, 'b') en se basant sur la table E_N_T

Exécution de code :



Erreur d'intégration E_n(f)

RQ : plus on augmente le nombre des subdivisions plus l'erreur d'intégration diminue

Application 2

- Création de méthode qui permet de comparer entre méthode de point milieu et de trapèzes

```
clear all;close all;clc;
%%%%%% M�thode des Trap�zes
%%%%%% Application 2
 syms x


f=x.*exp(-x).*cos(2.*x);
I=int(f,0,2*pi);

I_1=point_milieu(@(t) t*exp(-t)*cos(2*t),0,2*pi,1000);

I_2=trapeze(@(t) t*exp(-t)*cos(2*t),0,2*pi,1000);

erreur_point_milieu=round(eval(abs(I-I_1)),5)
erreur_methode_trapeze=round(eval(abs(I-I_2)),5)
```

Explication du code :

1- Déclaration de la fonction à tester f= x* exp(-x)*cos(2x)
2- Calcul de la valeur réel de l'intégrale par la méthode int(f,0,2*pi)
3- Calcul de l'intégrale de f entre 0 et 2pi par la methode de point milieu et du trapèze puis on calcule l'erreur de chaque methode

Résultat d'exécution :

```
erreur_point_milieu =

     0


erreur_methode_trapeze =

   5.2000e-04
```

Interprétation :

L'erreur tend vers 0, mais pas à la même vitesse, plus rapidement pour les trapèzes et le point milieu que pour les rectangles.

Méthode de Simpson :

Application1

- Création de fonction Simpson

```
function int=methode_simpson(fun,lowerBound,upperBound,n)
dx=(upperBound - lowerBound)/n;
x=lowerBound:dx:upperBound;
som1=0;som2=0;
for i=1:n/2
som1=som1+fun(x(2*i-1));
end
for i=1:n/2-1
som2=som2+fun(x(2*i));
end
int=(dx/3)*(fun(1)+fun(n)+4*som1+2*som2);
end
```

Explication du code :

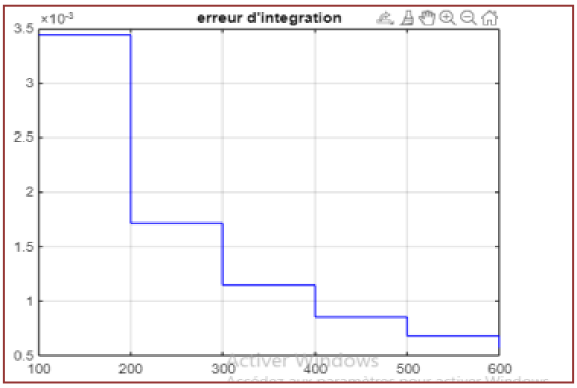
- ➔ Cette fonction prend en paramètre la fonction dont on veut faire l'approximation ainsi que la borne inferieur , la borne supérieur de l'intégrale et le nombre des subdivisions .
- ➔ dx= la longueur d'une subdivision de l'intervalle
- ➔ initialisation de som1 et som2 à 0
- ➔ par une boucle for; on fait la somme des tous les images des x(i) tel que i est impaire (i appartient à (1..n-1)/ des valeurs paires) par f et on le stocke dans som1 et dans som2 on stocke les images de x(i) par f tel que i est paires (on prend seulement les valeurs paires i appartient (2..n))
- ➔ par la suite pour calculer la valeur de l'intégrale par la méthode de Simpson on suit la formule indiqué dans la capture si dessus .

Calcul d'erreur d'intégration :

```
clear all;close all;clc;
%%%%%% M�thode de Simpson
%%%%%% Application 1
 syms x


f=exp(-x^2);
I=int(f,1,2);

N=[100  200  300  400  500 600 ];
E_N_S=zeros(1,length(N));
I_1=zeros(1,length(N));
for i=1:length(N)

I_1(i)=simpson(@(t) exp(-t^2),1,2,N(i));
E_N_S (i)=abs(I-I_1 (i));

end
figure
stairs(N,E_N_S,'b');
grid
title('erreur d''integration')
```

RQ : Même principe que le calcul d'erreur des autres méthodes

Résultat d'exécution :



Erreur d'intégration E_n(f)

RQ : Interprétation : plus N[i] nombre des subdivisions augmente plus la valeur de l'erreur d'intégration diminue

Application2 :

- Comparer entre méthode de trapèze et celle de Simpson

```
clear all;close all;clc;
%%%%%% M�thode de simpson
%%%%%% Application 2
 syms x


f=x.*exp(-x).*cos(2.*x);
I=int(f,0,2*pi);


I_1=trapeze(@(t) t*exp(-t)*cos(2*t),0,2*pi,500);
I_2=simpson(@(t) t*exp(-t)*cos(2*t),0,2*pi,500);


erreur_methode_trapeze=eval(abs(I-I_1))
erreur_methode_simpson=eval(abs(I-I_2))
```

Explication du code : même principe calcul de l'intervalle par la methode int prédéfinie par les bibliothèque du Matlab puis on calcule la valeur d'intégration par la methode de trapèze et de Simpson ainsi que les erreurs d'intégrations pour chaque methode par les fonctions errreur_methode_trapeze et errreur_methode_Simpson et on les affiche par eval ;

Rq : **eval(expression)** : permet d'évaluer une expression retourne la valeur de retour dune expression donnée

si dessous les valeurs trouvées :

Résultat d'exécution :

```
erreur_methode_trapeze =

    0.0010
```

```
erreur_methode_simpson =

   8.6546e-04
```

Interprétation :

On peut conclure que l’erreur d’intégration selon la méthode de Simpson est inferieur à celles des autres méthodes . Simpson est plus précises que les autres.