

## TP 1 - Calcul Scientifique

In [2]:

```
%pylab inline

# %pylab inline est un équivalent de l'importation* de numpy et de matplotlib.pyplot
plot
##matplotlib inline
#from matplotlib.pyplot import *
#from numpy import *
#from scipy import *
```

Populating the interactive namespace from numpy and matplotlib

### Exercice 1 - Interpolation

1) On considère les abscisses  $x = [-2, 0, 1, 2]$  et  $y = [4, 0, 0, 4]$ . Parmi les polynômes suivants, lequel est le polynôme d'interpolation aux points  $x, y$  (justifiez votre réponse) ?

- $p_1(x) = x^4 - \frac{2}{3}x^3 - 3x^2 + \frac{8}{3}x$
- $p_2(x) = \frac{4}{3}x^2 - \frac{4}{3}$
- $p_3(x) = \frac{1}{3}x^3 + x^2 - \frac{4}{3}x$

**Correction** : On rappelle ici que  $P$  est LE polynôme d'interpolation de Lagrange associé aux points  $x, y$  si et seulement si (voir le cours):

$$\begin{cases} \deg(P) \leq 3 \\ \forall i = 0 \dots 3, P(x_i) = y_i \end{cases}$$

On remarque que  $p_1$  est de degré trop grand : le polynome recherché doit être de degré inférieur ou égal à 3 (puisque'il y a 4 points). Le polynome  $p_2$  ne vaut pas 0 en 0. On vérifie que le polynome  $p_3$  convient : il a bien un degré inférieur ou égal à 3 et  $p_3(-2) = 4, p_3(0) = 0, p_3(1) = 0, p_3(2) = 4$ .

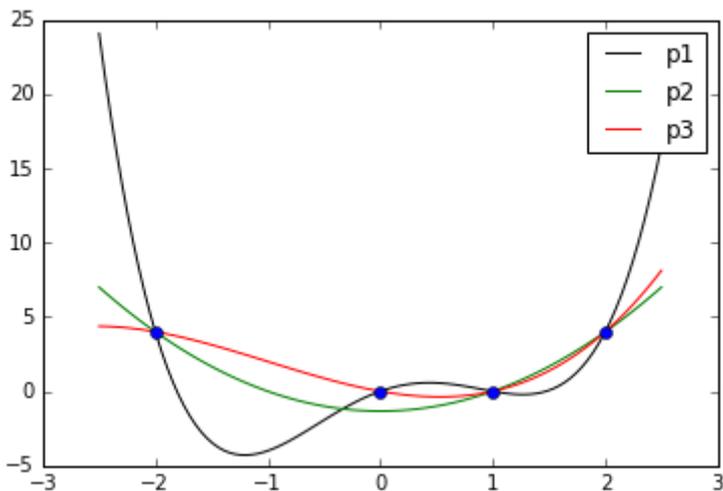
2) Représentez sur une même figure les points d'interpolation, ainsi que les polynômes  $p_1, p_2$  et  $p_3$  respectivement en noir, vert et rouge, sur l'intervalle  $[-2.5, 2.5]$ .

In [3]:

```
#On commence par créer les points d'interpolation :  
Xint=array([-2,0,1,2])  
Yint=array([4,0,0,4])  
#On crée un vecteur x pour tracer les courbes :  
x=linspace(-2.5,2.5,100)  
#On calcule les images correspondantes pour p1, p2 et p3:  
p1=x**4-2.0/3*x**3-3*x**2+8.0/3*x  
p2=4.0/3*x**2-4.0/3  
p3=1.0/3*x**3+x**2-4.0/3*x  
#on trace les graphes de p1 en noir, de p2 en vert et de p3 en rouge :  
plot(x,p1,'black',x,p2,'g',x,p3,'r')  
# je mets une légende : premier graphe j'affiche p1, 2eme j'affiche p2, 3eme j'affiche p3  
legend(['p1','p2','p3'])  
# je mets des ronds sur les points de coordonnées Xint Yint  
plot(Xint,Yint,'o')  
# Résultat : on voit bien que p3 est le seul polynome dont le graphe passe par les 3 points
```

Out[3]:

[<matplotlib.lines.Line2D at 0x10d9dc290>]



## Exercice 2 - Calcul de polynômes d'interpolation

1) Calculez les polynômes d'interpolation aux points suivants :

a)  $x = [-1, 2, 3]$  et  $y = [4, 4, 8]$  **Correction** : *Méthode 1* : par une résolution de système : on cherche  $P$  de degré inférieur ou égal à 2 (car il y a 3 points), on cherche donc  $P$  sous la forme  $P(X) = a + bX + cX^2$ . En écrivant  $P(-1) = 4$ ,  $P(2) = 4$  et  $P(3) = 8$  on obtient le système suivant :

$$\begin{cases} a - b + c = 4 \\ a + 2b + 4c = 4 \\ a + 3b + 9c = 8 \end{cases}$$

ce qui nous donne  $P(X) = 2 - X + X^2$ . *Méthode 2* : par la base de Lagrange : On sait d'après le cours que le polynome  $P$  s'écrit simplement dans la base de Lagrange  $P(X) = y_0 L_0(X) + y_1 L_1(X) + y_2 L_2(X)$  et l'on calcule les polynômes  $L_k(x) = \prod_{j \in \{0, \dots, n\}, j \neq k} \frac{x - x_j}{x_k - x_j}$ . On obtient :

$$\begin{cases} L_0(X) = \frac{1}{12}(X - 2)(X - 3) \\ L_1(X) = \frac{1}{12}(X + 1)(X - 3) \\ L_2(X) = \frac{1}{4}(X + 1)(X - 2) \end{cases}$$

Alors  $P(X) = \frac{1}{3}(X - 2)(X - 3) + \frac{1}{3}(X + 1)(X - 3) + 2(X + 1)(X - 2)$ . Il n'est pas utile de le développer mais si on le faisait, on retrouverait bien sur celui de la méthode 1 (puisque'il est unique d'après le cours).

d)  $x = [-1, 0, 1]$  et  $y = [1, 0, 1]$

**Correction** : Ici c'est plus simple puisqu'on voit à l'oeil que  $Q(X) = X^2$  convient.

e)  $x = [-3, -1, 2, 10]$  et  $y = [-3, -1, 2, 10]$

**Correction** : Ici c'est encore plus simple puisqu'on constate que  $R(X) = X$  convient.

2) Représentez graphiquement les polynômes des points a) et b), accompagnés des points d'interpolation correspondants.

In [4]:

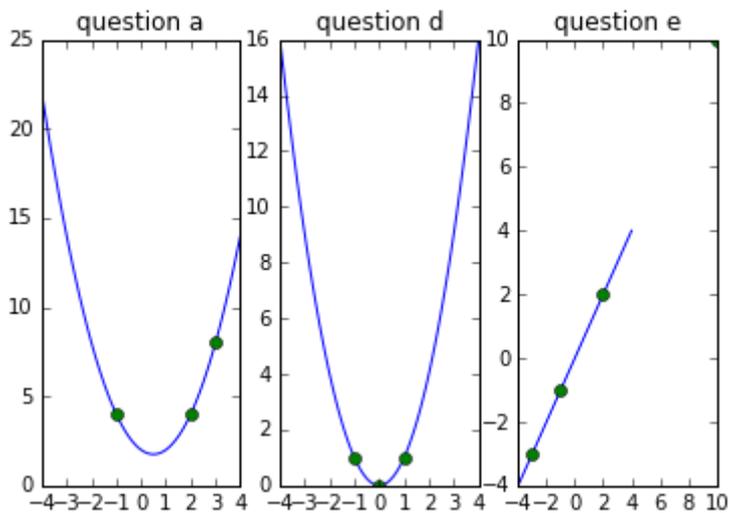
```
Xa=array([-1,2,3])
Ya=array([4,4,8])
Xd=array([-1,0,1])
Yd=array([1,0,1])
Xe=array([-3,-1,2,10])
Ye=Xe
```

```
x=linspace(-4,4,100)
P=2-x+x**2
Q=x**2
R=x
```

```
subplot(1,3,1)
plot(x,P,Xa,Ya,'o')
title('question a')
subplot(1,3,2)
plot(x,Q,Xd,Yd,'o')
title('question d')
subplot(1,3,3)
plot(x,R,Xe,Ye,'o')
title('question e')
```

Out[4]:

<matplotlib.text.Text at 0x10e089c50>



### Exercice 3 - Base de Lagrange

Soit  $x_0, \dots, x_n$  ( $n+1$ ) réels distincts deux à deux. Pour  $k \in \{0, \dots, n\}$ , on note

$$L_k(x) = \prod_{j \in \{0, \dots, n\}, j \neq k} \frac{x - x_j}{x_k - x_j}$$

le  $k$ -ième polynôme de Lagrange.

1) Montrez que  $L_k$  est un polynôme de degré  $n$  vérifiant  $L_k(x_i) = \delta_{ki}$  pour tout  $k, i \in \{0, \dots, n\}$ .

**Correction :** (fait en cours) On voit que  $L_k$  est écrit sous forme factorisé : C'est le produit des  $(X - x_j)$  pour  $j$  différent de  $k$ . Cela montre bien que  $L_k$  est un polynôme de degré  $n + 1 - 1 = n$ , et qu'il admet tous les  $x_j$  pour  $j$  différent de  $k$  comme racines. La valeur en  $x_k$  est obtenue en calculant  $L_k(x_k)$  avec la formule et on obtient un produit de  $n$  1, c'est à dire 1.

2) En déduire que la famille de polynôme  $\{L_k\}_{k \in \{0, \dots, n\}}$  forme une base de  $\mathbb{R}_n[X]$ .

**Correction :** (fait en cours) Je vous renvoie ici au cours mais je rappelle au passage qu'ici l'espace vectoriel dont on parle est l'espace des polynômes de degré inférieur ou égal à  $n$  et qu'il est de dimension  $n + 1$ . On cherche donc une famille de  $n + 1$  polynômes de degré inférieur ou égal à  $n$ . On utilise donc la question précédent pour dire que les  $L_k$  sont bien dans le bon espace. Par ailleurs, il y en a bien  $n + 1$  dans la famille. Il reste à montrer que la famille est libre. En effet, on rappelle qu'une famille est une base si et seulement si elle a le bon nombre d'éléments (égal à la dimension de l'espace) et qu'elle est libre.

Pour montrer qu'elle est libre, il faut supposer que l'on a des réels  $\alpha_0, \dots, \alpha_n$  tels que

$$\alpha_0 L_0 + \alpha_1 L_1 + \dots + \alpha_n L_n = 0$$

et montrer que les  $\alpha_i$  doivent tous être nuls. Ce que l'on veut dire par là est que le polynôme  $\alpha_0 L_0 + \alpha_1 L_1 + \dots + \alpha_n L_n$  est le polynôme nul. On montre que  $\alpha_0 = 0$  en écrivant cette équation en  $X = x_0$ . A gauche, il ne reste que  $\alpha_0 L_0(x_0) = \alpha_0$  puisque tous les  $L_i(x_0)$  sont nuls pour  $i = 1, \dots, n$  d'après la question précédente.

3) Écrire une fonction `PolLagrange(xint, x, k)` qui calcule les valeurs prises par le  $k$ -ième polynôme de la base de Lagrange associé aux abscisses  $X_{int} = (x_0, \dots, x_n)$ , en un point  $x$ .

4) Écrire une fonction `ApproxLagrange(xint, f, x)` qui renvoie la valeur en  $x$  du polynôme d'interpolation de Lagrange de  $f$  associé aux noeuds  $X_{int} = (x_0, \dots, x_n)$ .

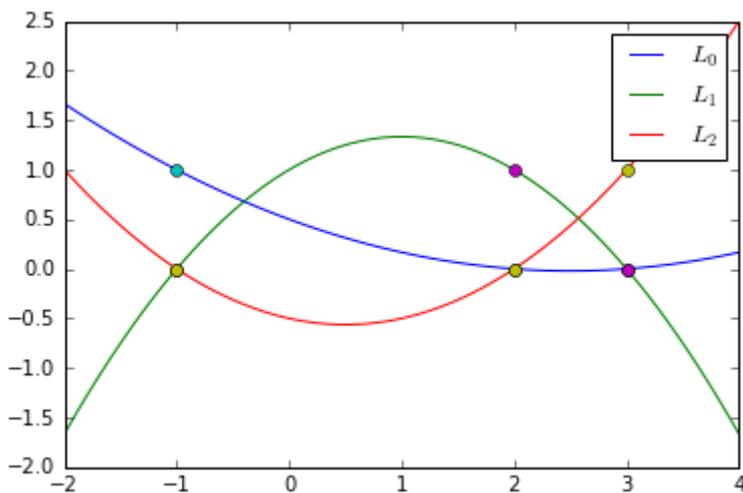
In [5]:

```
# Ici on veut coder un produit, donc on initialise à 1 et on fait une boucle pour
i de 0 à n.
# Il faut ensuite retirer le cas i=k, et on le fait avec un test.
def PolLagrange(Xint, x, k):
    P=1
    for i in range(len(Xint)):
        if i!=k:
            P=P*(x-Xint[i])/(Xint[k]-Xint[i])
    return P

# On trace ci dessous L_0, L_1 et L_2 pour les points d'interpolation x0=-1, x1=
2, x2=3
x=linspace(-2,4,100)
Xint=array([-1, 2, 3])
l0=PolLagrange(Xint,x,0)
l1=PolLagrange(Xint,x,1)
l2=PolLagrange(Xint,x,2)
Y0=array([1,0,0])
Y1=array([0,1,0])
Y2=array([0,0,1])
plot(x,l0,x,l1,x,l2)
legend(['$L_0$', '$L_1$', '$L_2$'])
plot(Xint,Y0,'o',Xint,Y1,'o',Xint,Y2,'o')
```

Out[5]:

```
[<matplotlib.lines.Line2D at 0x10e35bd10>,
<matplotlib.lines.Line2D at 0x10e371f50>,
<matplotlib.lines.Line2D at 0x10e37d4d0>]
```



On observe bien ce que l'on souhaitait :  $L_0$  vaut bien 1 en  $x_0 = -1$  et 0 en  $x_1 = 2$  et  $x_2 = 3$ , et de même pour  $L_1$  qui vaut 0 en  $-1$  et  $3$  et 1 en  $2$ , et  $L_2$  qui vaut 0 en  $-1$  et  $2$  et 1 en  $3$ .

In [6]:

```
# Pour commencer on va se donner des points Xint et Yint,
# et on utilise juste la formule de P dans la base de Lagrange et on va donc coder la somme des y_k L_k(X) pour k de 0 à n.
# Pour cela (comme c'est une somme), on initialise à 0 et on fait une boucle de 0 à n.
# Pour L_k(X) on appelle la fonction précédente

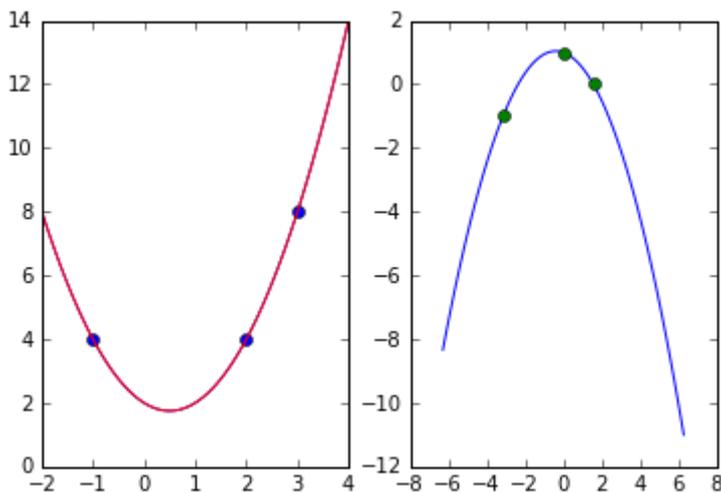
def ApproxLagrange(Xint,Yint,x):
    P=0
    for k in range(0,len(Xint)):
        P=P+Yint[k]*PolLagrange(Xint,x,k)
    return P

# On va tester en traçant le polynome d'interpolation de la quetsion 2-a et celui calculé avec le programme.
Xint=array([-1,2,3])
Yint=array([4,4,8])
x=linspace(-2,4,100)
y=ApproxLagrange(Xint,Yint,x)
z=2-x+x**2
subplot(1,2,1)
plot(x,y,'b',Xint,Yint,'o',x,z,'r') #on voit bien que les deux polynomes sont les memes !

#Si on veut passer une fonction en argument pour calculer le polynome de Lagrange d'une fonction on fait comme avant,
# sauf que les y_k sont les f(x_k).
def ApproxLagrangef(Xint,f,x):
    P=0
    for k in range(0,len(Xint)):
        P=P+f(Xint[k])*PolLagrange(Xint,x,k)
    return P
Xint=array([-pi,0,pi/2])
x=linspace(-2*pi, 2*pi, 100)
y=ApproxLagrangef(Xint,cos,x)
subplot(1,2,2)
plot(x,y,Xint,cos(Xint),'o')
```

Out[6]:

```
[<matplotlib.lines.Line2D at 0x10e609550>,  
<matplotlib.lines.Line2D at 0x10e6ed950>]
```



### Exercice 4 - Méthode de Simpson pour le calcul approché d'intégrales

Soit une fonction continue  $g : [1, 2] \rightarrow \mathbb{R}$ . On cherche à approcher l'intégrale  $I(g) = \int_1^2 g(x)dx$  par la formule d'intégration numérique  $J_2$  suivante:

$$J_2(g) = \lambda_0 g(1) + \lambda_1 g\left(\frac{3}{2}\right) + \lambda_2 g(2)$$

où les  $\lambda_i$  sont des réels.

1) Ecrire le système que doit vérifier  $(\lambda_0, \lambda_1, \lambda_2)$  pour que la méthode soit exacte pour les polynômes de degré inférieur ou égal à 2.

**Correction :** La méthode est exacte pour les polynômes de degré inférieur ou égal à 2 si et seulement si elle est exacte pour les polynômes  $1, X$  et  $X^2$ , soit :

$$\begin{cases} \lambda_0 + \lambda_1 + \lambda_2 = 1 \\ \lambda_0 + \frac{3}{2}\lambda_1 + 2\lambda_2 = \frac{3}{2} \\ \lambda_0 + \frac{9}{4}\lambda_1 + 4\lambda_2 = \frac{7}{3} \end{cases}$$

2) En déduire les valeurs de  $\lambda_0, \lambda_1$  et  $\lambda_2$ .

**Correction :** On obtient après résolution  $\lambda_0 = \lambda_2 = \frac{1}{6}$ ,  $\lambda_1 = \frac{2}{3}$ .

3) Quel est le degré d'exactitude de la méthode ?

**Correction :** On sait déjà qu'elle est au moins de degré d'exactitude 2 puisqu'elle a été construite pour être exacte pour les polynômes de degré inférieur ou égal à 2. Mais elle peut être de degré d'exactitude plus grand, il faut tester sur  $P = X^3$ , puis  $P = X^4$  etc jusqu'à trouver un degré pour lequel  $J_2(P) \neq \int_1^2 P(x)dx$ . Ici, on calcule  $J_2(X^3) = \frac{15}{4} = \int_1^2 x^3 dx$ . Par contre on trouve par le calcul que  $J_2(X^4) \neq \int_1^2 x^4 dx$ .

4) Déduire de cette étude une valeur approchée de  $\ln(2)$ . Que peut-on faire pour avoir une approximation aussi fine de que l'on souhaite ?

**Correction :** On utilise le fait que  $\ln(2) = \int_1^2 \frac{dx}{x}$ . Maintenant,  
 $\int_1^2 \frac{dx}{x} \approx J_2\left(\frac{1}{x}\right) = \frac{1}{6} * 1 + \frac{2}{3} * \frac{2}{3} + \frac{1}{6} * \frac{1}{2} = \frac{25}{36}$ .

## Exercice 5 - Méthode des rectangles en des trapèzes composite

1) Écrire une fonction `RectComp(a, b, f, h)`, qui retourne une approximation numérique de l'intégrale de la fonction  $f$  sur  $[a, b]$ , en utilisant la méthode d'intégration de rectangle à gauche composite sur une subdivision uniforme de pas  $h$ .

2) De même qu'à la question 1), écrire une fonction `TrapComp(a, b, f, h)`, qui retourne une approximation numérique de l'intégrale de la fonction  $f$  sur  $[a, b]$ , en utilisant la méthode d'intégration de trapèze à gauche composite sur une subdivision uniforme de pas  $h$ .

**Correction :** On rappelle la formule vue en cours pour les méthode des rectangles à gauche et des trapèzes composites : si  $a_0 = a, \dots, a_n = b$  désigne une subdivision uniforme de pas  $h$  de  $[a, b]$  (c'est-à-dire que  $a_i = a + i * h$ ), alors

$$J_{comp}^{rg}(f) = h \sum_{k=0}^{n-1} f(a_k), \quad J_{comp}^{trap}(f) = \frac{h}{2} \sum_{k=0}^{n-1} (f(a_k) + f(a_{k+1})).$$

3) Utiliser ces fonctions pour calculer une approximation de l'intégrale de la fonction  $f(x) = \cos(x)$  sur  $[0, 2]$  par les méthodes composites des rectangles et trapèzes, avec  $h = 0.1$ , puis  $h = 0.01$  et enfin  $h = 0.001$ . Quelle erreur obtient-on dans chaque cas ? Cela est-il en accord avec les résultats théorique vus en cours ?

In [10]:

```
# On commence par la méthode "naïve" : on doit coder une somme, on fait une boucle...

def RectComp1(a,b,f,h):
    S=0
    N=int((b-a)/h)
    for k in range(N):
        S=S+h*f(a+k*h)
    return(S)

# On peut sinon utiliser le vecteur des a_k, calculer f(a_k) et ensuite sommer en utilisant
# à fond le caractère vectoriel de numpy.

def RectComp(a,b,f,h):
    return(h*sum(f(arange(a,b,h))))

def TrapComp(a,b,f,h):
    return(h/2*sum(f(arange(a,b,h)))+h/2*sum(f(arange(a+h,b+h,h))))

#affichage des erreurs pour h=0.1, h=0.01, h=0.001. On utilise pour cela la valeur théorique : sin(2)
print(abs(RectComp(0,2,cos,0.1)-sin(2)))
print(abs(RectComp1(0,2,cos,0.1)-sin(2)))
print(abs(RectComp(0,2,cos,0.01)-sin(2)))
print(abs(RectComp1(0,2,cos,0.01)-sin(2)))
print(abs(RectComp(0,2,cos,0.001)-sin(2)))
print(abs(RectComp1(0,2,cos,0.001)-sin(2)))
#Resultats théoriques pour la méthode des rectangles composite : erreur de l'ordre de h
# On remarque que les deux méthodes avec arange et la boucle ne donnent pas exactement les memes valeurs,
# c'est à cause des arrondis dans "arange"

print(abs(TrapComp(0,2,cos,0.1)-sin(2)))
print(abs(TrapComp(0,2,cos,0.01)-sin(2)))
print(abs(TrapComp(0,2,cos,0.001)-sin(2)))
#Resultats théoriques pour la méthode des rectangles composite : erreur de l'ordre de h^2

0.0700494676503
0.546469716227
0.00707315669155
0.803224538627
0.000707997643487
0.918468445751
0.000757874177074
7.57749118596e-06
7.57747867786e-08
```

## Exercice 6 - Une nouvelle méthode d'intégration numérique

Soient  $x_1, x_2 \in [-1, 1]$ ,  $x_1 < x_2$ , et  $\lambda_1, \lambda_2 \in \mathbb{R}$ .

On définit, pour  $f$  une fonction continue sur  $[-1, 1]$ , la méthode d'intégration numérique  $T$  de la façon suivante :

$$T(f) = \lambda_1 f(x_1) + \lambda_2 f(x_2).$$

a) Montrer que  $T$  est exacte pour les polynômes de degré inférieur ou égal à 1 sur  $[-1, 1]$  si et seulement si  $\lambda_1 = \frac{2x_2}{x_2 - x_1}$  et  $\lambda_2 = \frac{2x_1}{x_1 - x_2}$ .

**Correction :** Par définition, la méthode  $T$  est exacte pour les polynômes de degré inférieur ou égal à 1 si et seulement si elle est exacte pour  $P(X) = 1$  et  $P(X) = X$ , c'est-à-dire:

$$\begin{cases} \lambda_1 + \lambda_2 &= \int_{-1}^1 dx = 2 \\ \lambda_1 x_1 + \lambda_2 x_2 &= \int_{-1}^1 x dx = 0 \end{cases}$$

Ce qui donne finalement  $\lambda_1 = \frac{2x_2}{x_2 - x_1}$  et  $\lambda_2 = \frac{2x_1}{x_1 - x_2}$ .

b) Pour quelles valeurs de  $\lambda_1, \lambda_2, x_1$  et  $x_2$ ,  $T$  est-elle exacte pour des polynômes de degré inférieur ou égal à 3 ? Quel est alors le degré d'exactitude de la méthode ?

**Correction :** Pour que  $T$  soit exacte pour les polynômes de degré inférieur ou égal à 3, il faut et il suffit qu'elle soit exacte

- (a) pour les polynômes de degré inférieur ou égal à 1
- (b) pour  $P(X) = X^2$
- (c) pour  $P(X) = X^3$ .

Or, on a vu que (a) est vérifié si et seulement si  $\lambda_1 = \frac{2x_2}{x_2 - x_1}$  et  $\lambda_2 = \frac{2x_1}{x_1 - x_2}$ . De plus,

$$(b) \Leftrightarrow \lambda_1 x_1^2 + \lambda_2 x_2^2 = \int_{-1}^1 x^2 dx = \frac{2}{3} \Leftrightarrow x_1 x_2 = -\frac{1}{3}.$$

Enfin, (c) est vrai si et seulement si  $\lambda_1 x_1^3 + \lambda_2 x_2^3 = 0$  c'est-à-dire  $x_1^2 = x_2^2$  (car par hypothèse  $x_1 \neq x_2$ ). D'après (b),  $x_1$  et  $x_2$  sont de signe différents, ce qui donne au final  $x_1^2 = \frac{1}{3}$  :  $T$  est exacte pour les polynômes de degré inférieur ou égal à 3 si et seulement si

$$x_1 = -\frac{1}{\sqrt{3}}, \quad x_2 = \frac{1}{\sqrt{3}}, \quad \lambda_1 = \lambda_2 = 1.$$

c) Dédurre des questions précédentes une méthode d'intégration de degré d'exactitude 3 sur un segment  $[a, b]$  quelconque.

**Correction :** On utilise le changement de variable expliqué en cours pour passer de notre méthode sur  $[-1, 1]$  à une méthode sur  $[a, b]$  quelconque. On obtient

$$J(f) = \frac{(b-a)}{2} (f(t_1) + f(t_2))$$

$$\text{avec } t_1 = -\frac{b-a}{2} \frac{\sqrt{3}}{3} + \frac{b+a}{2} \text{ et } t_2 = \frac{b-a}{2} \frac{\sqrt{3}}{3} + \frac{b+a}{2}$$

c) Programmer la méthode d'intégration obtenue, puis la méthode d'intégration composite correspondante. Vérifiez l'ordre de la méthode en testant sur la fonction cos sur  $[0, 2]$ .

In [11]:

```
def MethIntExo6(a,b,f):
    #On se contente de recopier la formule précédente :
    t1 = -((b-a)/2)*(sqrt(3)/3) + ((b+a)/2)
    t2 = ((b-a)/2)*(sqrt(3)/3) + ((b+a)/2)
    Jf = ((b-a)/2)*(f(t1) + f(t2))
    return Jf

print("Méthode de l'exercice 6 pour cos entre 0 et 2")
print(MethIntExo6(0,2,cos),sin(2))

#La méthode composite peut s'obtenir avec en s'inspirant de l'exercice précédent
:

def MethCompExo6(a,b,f,h):
    A=arange(a,b+h,h) #subdivision uniforme de [a,b] de pas h : a0...an
    n=len(A)-1
    J=0
    for k in range(0,n):
        J=J+MethIntExo6(A[k],A[k+1],f)
    return J

print("Méthode composite de l'exercice 6 pour cos entre 0 et 2, pas 0.1")
print(MethCompExo6(0,2,cos,0.1),sin(2))
print("Méthode composite de l'exercice 6 pour cos entre 0 et 2, pas 0.01")
print(MethCompExo6(0,2,cos,0.01),sin(2))
```

```
Méthode de l'exercice 6 pour cos entre 0 et 2
(0.90545138523558433, 0.90929742682568171)
Méthode composite de l'exercice 6 pour cos entre 0 et 2, pas 0.1
(0.90929740577044627, 0.90929742682568171)
Méthode composite de l'exercice 6 pour cos entre 0 et 2, pas 0.01
(0.90929742682357673, 0.90929742682568171)
```

In [ ]: