—————————————————**TP0:**———————————————

La structure Lambda:

```
z= lambda x,y: x**2-y**2
z(2,3)
```

—————————————Représentation graphique:
———————————————

```
import matplotlib.pyplot as plt
plt.figure(figsize=(20,10))
x=np.arange(-np.pi,np.pi,0.01)
y=np.sin(x)
z=np.cos(x)
plt.plot(x,y,'ro-',x,z,'b^-')
plt.xlabel('x')
plt.legend(('sin(x)','cos(x)'),loc = 0)
plt.grid(True)
plt.savefig('firstplot.png',format='png')
plt.show()
```

————————————————TP1:———————————————————

—————————————CONCATENER—————————————

```
import numpy as np
def concatener(A,b):
    if(A.shape[0]==len(b)):
        n = len(b)
        Ab = np.zeros((n,n+1))
        Ab[0:n,0:n] = A
        Ab[0:n,n] = b[:,0]
        return Ab
    else:
        print("La matrice et le vecteur n'ont pas le meme nombre de lignes")
        return 0
```

——————————————————PERMUTATION
————————————————

```
def permuter(A,i,j):
    P=A.copy()
    L=P[i,:].copy()
    P[i,:]=P[j,:]
    P[j,:]=L
    return P
```

//TEST//print(permuter(Ab,0,1))

—————————————COMBINAISON—————————————

```
def ajouter(A,i,j,alpha):
    n=A.shape[0]
    C = A.copy()
    C[i,0:n] += alpha*C[j,0:n]
    return C
```

—————————————CHOIX PIVOT—————————————

```
def choix_pivot(A,j):
    n=A.shape[0]
    if A[j,j]!=0:
        Ind_max=j
    else:
        Ind=np.where(A[j+1:n,j]!=0)
        Ind_max=Ind[0][0]+j+1
    return Ind_max
```

—————————————Recherche du pivot partiel—————————————

```
def choix_pivot_partiel(A,j):
    n=A.shape[0]
    return np.argmax(np.abs(A[j:n,j]))+j
```

—————————————TRIANGULARISER—————————————

```
def triangulariser(A):
    n=A.shape[0]
    T=A.copy()
    for j in np.arange(0,n):
        ind_pivot=choix_pivot(T,j)
        if ind_pivot>j:
            T=permuter(T,j,ind_pivot)
        for i in np.arange(j+1,n):
            T=ajouter(T,i,j,-T[i,j]/T[j,j])
    return T
```

—————————————REMONTER—————————————

```
def remonter(A):
    n=A.shape[0]
    x=np.zeros((n,1))
    x[n-1]=A[n-1,n]/A[n-1,n-1]
```

```
    for i in np.arange(n-1,-1,-1):
        S=A[i,i+1:n].dot(x[i+1:n])
        x[i]=(A[i,n]-S)/A[i,i]
    return x
```

———————————————GAUSS———————————————

```
def PivotDeGauss(A,b):
    if np.linalg.det(A)==0:
        print('A n est pas inversible')
    else :
        Ab=concatener(A,b)
        T=triangulariser(Ab)
        X=remonter(T)
    return X
```

———————————————COMPARAISON TEMPS DE CALCUL
———————————————

```
%timeit PivotDeGauss(A,b)
%timeit np.linalg.solve(A,b)
# 1microsecond=10-^6 seconds
```

———————————————METHODES ITERATIVES———————————————

—————————————VERFICATION DIAGONALE DOMINANTE
—————————————

```
def Matrice_diag_dominante(A):
    for i in np.arange(0,A.shape[0]):
        if np.abs(A[i,i])<=np.sum(np.abs(A[i,:]))-np.abs(A[i,i]):
            #print('A n\'est pas à diagonale strictement dominante')
            etat=0
            break
    else:
        #print('A est à diagonale strictement dominante')
        etat=1
    return etat
```

———————————————METHODE 1 JACOBI———————————————

```
# Méthode 1
def jacobi(A, b, X0, epsilon):
    etat=int(Matrice_diag_dominante(A))
    if  etat==0:
        print('A n\'est pas à diagonale strictement dominante')
```

```
        return
    else:
        D = np.diagflat(np.diag(A))
        N = D-A
        X1=np.linalg.inv(D).dot(N.dot(X0)+b)
        k=1
        while np.linalg.norm(X1-X0,1)>epsilon:
            X0=X1
            X1=np.linalg.inv(D).dot(N.dot(X0)+b)
            k+=1
        return X0,k

//TEST//
X0=np.ones((4,1))
epsilon=10**(-6)
jacobi(A, b, X0, epsilon)
```

——————————————METHODE GAUSS-SEIDEL
——————————————

```
# Méthode 2
def Gauss_Seidel(A, b, X0, epsilon):
    etat=int(Matrice_diag_dominante(A))
    if  etat==0:
        print('A n\'est pas à diagonale strictement dominante')
    else:
        D = np.tril(A)
        N = D-A
        X1=np.linalg.inv(D).dot(N.dot(X0)+b)
        k=1
        while np.linalg.norm(X1-X0,1)>epsilon:
            X0=X1
            X1=np.linalg.inv(D).dot(N.dot(X0)+b)
            k+=1
        return X0,k

//TEST//
X0=np.ones((4,1))
epsilon=10**(-6)
Gauss_Seidel(A, b, X0, epsilon)
```

—————————————COMPARAISON TEMPS DE CALCUL2
—————————————

```
%timeit jacobi(A,b,X0,epsilon)
%timeit Gauss_Seidel(A,b,X0,epsilon)
%timeit np.linalg.solve(A,b)
```

```
%timeit PivotDeGauss(A,b)
```

————————————————-POLYNOME DE LAGRANGE
———————————————

```
import numpy as np
import matplotlib.pyplot as plt

def Lagrange(t,i,x):
    n=len(x)
    L=1
    for j in np.arange(0,n):
        if j!=i:
            L*=(t-x[j])/(x[i]-x[j])
    return L
```

//AFFICHAGE DES 3 POLYNOMES DE LAGRANGE//

```
x=np.arange(-1,2,1)
t=np.linspace(-1,1,21) # permet d'obtenir un tableau 1D allant de -1 à 1 contenant
21 éléments.
plt.figure(figsize=(20,10))
plt.plot(t,Lagrange(t,0,x),'ro--',t,Lagrange(t,1,x),'b^--',t, Lagrange(t,
2,x),'g*--',linewidth=3,markersize=12)
plt.xlabel('t',fontsize=30)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.legend(('L0','L1','L2'),fontsize=20, loc = 0)
plt.grid(True)
plt.text(-1,-0.1,"(-1,0)",ha="center",va="top",fontsize=30)
plt.text(0,-0.1,"(0,0)",ha="center",va="top",fontsize=30)
plt.text(1,-0.1,"(1,0)",ha="center",va="top",fontsize=30)
plt.text(-1,1.05,"(-1,1)",ha="center",va="bottom",fontsize=30)
plt.text(0,1.05,"(0,1)",ha="center",va="bottom",fontsize=30)
plt.text(1,1.05,"(1,1)",ha="center",va="bottom",fontsize=30)
```

————————————————-INTERPOLATION DE LAGRANGE
———————————————

```
def Interpolation_Lagrange(t,x,y):
    n=len(x)
    P=np.zeros((len(t)))
    for i in np.arange(0,n):
        P+=y[i]*Lagrange(t,i,x)
    return P
```

//AFFICHAGE DU POLYNOME D'INTERPOLATION

```
y=[8,3,6]
P=Interpolation_Lagrange(t,x,y)
plt.figure(figsize=(20,10))
plt.plot(t,P,'mo--',linewidth=3,markersize=12)
plt.xlabel('t',fontsize=30)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.grid(True)
plt.legend(('Polynome d\'interpolation de Lagrange',),fontsize=20, loc = 0)
plt.text(-1,8.5,"(-1,8)",ha="center",va="top",fontsize=30)
plt.text(0,3.5,"(0,3)",ha="center",va="top",fontsize=30)
plt.text(1,6.5,"(1,6)",ha="center",va="top",fontsize=30)
```

————————————EXEMPLE INTERPOLATION COSINUS ——————————————

```
x=np.linspace(-np.pi,np.pi,5)
t=np.linspace(-1.5*np.pi,1.5*np.pi,100)
P=Interpolation_Lagrange(t,x,np.cos(x))
plt.figure(figsize=(20,10))
plt.plot(t,P,'r-',t,np.cos(t), 'b--',x,np.cos(x),'mo',linewidth=3,markersize=12)
plt.xlabel('t',fontsize=30)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.grid(True)
plt.legend(('Polynome d\'interpolation de Lagrange', 'cosinus', 'points'),fontsize=20,
loc = 0)
```

——————————————————NEWTON ——————————————————————

```
//PREMIEREMENT POLYNOME NEWTON

def Newton(t,i,x):
    n=len(x)
    if i==0:
        return np.ones((len(t)))
    else:
        W=1
        for j in np.arange(0,i):
            W*=(t-x[j])
    return W

//AFFICHAGE NEWTON

x=np.arange(-1,2,1)
```

```
t=np.linspace(-1,1,21)
plt.figure(figsize=(20,10))
plt.plot(t,Newton(t,0,x),'ro--',t,Newton(t,1,x),'b^--',t, Newton(t,
2,x),'g*--',linewidth=3,markersize=12)
plt.xlabel('t',fontsize=30)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.legend(('W0','W1','W2'),fontsize=20, loc = 0)
plt.grid(True)
plt.text(-1,-0.1,"(-1,0)",ha="center",va="top",fontsize=30)
plt.text(0,-0.1,"(0,0)",ha="center",va="top",fontsize=30)
plt.text(1,-0.1,"(1,0)",ha="center",va="top",fontsize=30)
plt.text(-1,1.05,"(-1,1)",ha="center",va="bottom",fontsize=30)
plt.text(0,1.05,"(0,1)",ha="center",va="bottom",fontsize=30)
plt.text(1,1.05,"(1,1)",ha="center",va="bottom",fontsize=30)


//DIFFERENCE DIVISEE D'ABORD !!!!!!

def diff_div(x,y):
   n=len(y)
   beta=y.copy()
   for i in np.arange(1,n):
      for j in np.arange(n-1,i-1,-1):
         beta[j]=(beta[j]-beta[j-1])/(x[j]-x[j-i])
   return beta

//PUIS INTERPOLATION DE NEWTON: !!!!!!!!!

def Interpolation_Newton(t,x,y):
   n=len(x)
   P=np.zeros((len(t)))
   beta=diff_div(x,y)
   for i in np.arange(0,n):
      P+=beta[i]*Newton(t,i,x)
   return P

//AFFICHAGE INTERPOLATION

P=Interpolation_Newton(t,x,y)
plt.figure(figsize=(20,10))
plt.plot(t,P,'mo--',linewidth=3,markersize=12)
plt.xlabel('t',fontsize=30)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.grid(True)
plt.legend(('Polynome d\'interpolation de Newton',),fontsize=20, loc = 0)
```

```
plt.text(-1,8.5,"(-1,8)",ha="center",va="top",fontsize=30)
plt.text(0,3.5,"(0,3)",ha="center",va="top",fontsize=30)
plt.text(1,6.5,"(1,6)",ha="center",va="top",fontsize=30)
```

——————————————NEWTON OPT————————————————-

```
//NEWTON OPTIMISEE

def Newton_opt(t,i,x):
    n=len(x)
    if i==0:
        return np.ones((len(t)))
    elif i==1:
        return t-x[0]
    else:
        return (t-x[i-1])*Newton_opt(t,i-1,x)

//INTERPOLATION DE NEWTON OPTIMISEE

def Interpolation_Newton_opt(t,x,y,beta=diff_div(x,y)):
    n=len(y)
    if len(x)==2:
        return beta[0]+beta[1]*Newton_opt(t,1,x[0:1])
    else:
        return Interpolation_Newton_opt(t,x[0:n-1],y[0:n-1])
+beta[len(x)-1]*Newton_opt(t,len(x)-1,x)

//AFFICHAGE INTERPOLATION DE NEWTON OPTIMISEE

P=Interpolation_Newton_opt(t,x,y)
plt.figure(figsize=(20,10))
plt.plot(t,P,'mo--',linewidth=3,markersize=12)
plt.xlabel('t',fontsize=30)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.grid(True)
plt.legend(('Polynome d\'interpolation de Newton',),fontsize=20, loc = 0)
plt.text(-1,8.5,"(-1,8)",ha="center",va="top",fontsize=30)
plt.text(0,3.5,"(0,3)",ha="center",va="top",fontsize=30)
plt.text(1,6.5,"(1,6)",ha="center",va="top",fontsize=30)
```

——————————————ERREUR D’INTERPOLATION ————————————————-

```
//ERREUR D'INTERPOLATION EXEMPLE AVEC f(x)=1/(1+8*x**2)

f=lambda x: 1/(1+8*x**2)
N=np.arange(5,31,5)
t= np.linspace(-1,1,1000)
Erreur=np.zeros((len(N),len(t)))
for i in np.arange(0,len(N)):
    x=np.linspace(-1,1,N[i])
    Erreur[i,:]=np.abs(f(t)-Interpolation_Lagrange(t,x,f(x)))

//AFFICHAGE ERREUR D'INTERPOLATION

plt.figure(figsize=(20,10))
plt.subplot(3,2,1)
plt.plot(t,Erreur[0,:],linewidth=3)
plt.title('Erreur pour n=5',fontsize=20)
plt.grid(True)
plt.subplot(3,2,2)
plt.plot(t,Erreur[1,:],linewidth=3)
plt.title('Erreur pour n=10',fontsize=20)
plt.grid(True)
plt.subplot(3,2,3)
plt.plot(t,Erreur[2,:],linewidth=3)
plt.title('Erreur pour n=15',fontsize=20)
plt.grid(True)
plt.subplot(3,2,4)
plt.plot(t,Erreur[3,:],linewidth=3)
plt.title('Erreur pour n=20',fontsize=20)
plt.grid(True)
plt.subplot(3,2,5)
plt.plot(t,Erreur[4,:],linewidth=3)
plt.title('Erreur pour n=25',fontsize=20)
plt.grid(True)
plt.xlabel('t',fontsize=30)
plt.subplot(3,2,6)
plt.plot(t,Erreur[5,:],linewidth=3)
plt.title('Erreur pour n=30',fontsize=20)
plt.grid(True)
plt.xlabel('t',fontsize=30)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
```

————————Least Squared Approximation des Moindres carrés ——————————

```
import numpy as np
import matplotlib.pyplot as plt
```

```
def LSA(X,Y,p):
    n=len(X)
    A=np.ones((n,p+1))
    for i in np.arange(1,p+1):
        A[:,i]=X[:,0]**i
    Lambda=np.linalg.inv(A.transpose().dot(A)).dot(A.transpose().dot(Y))
    return Lambda

//évalue un polynôme de coefficients coeff en t.
def Poly(t,coeff):
    t.reshape(len(t),1)
    n=len(coeff)
    P=coeff[0]*np.ones((len(t),1))
    for i in np.arange(1,n):
        P+=coeff[i]*t**i
    return P

//Exercice exemple

X=np.array([[1,2,3,4,5]]).transpose()    //les Xi
Y=np.array([[0.9,1.5,3.5,4.2,4.9]]).transpose()     //les Yi
Lambda=LSA(X,Y,1)

//affichage

t=np.arange(-1,6,.01)
t=t.reshape(len(t),1)
plt.figure(figsize=(20,10))
plt.plot(X,Y,'ro',t,Poly(t,Lambda),'b--',linewidth=3,markersize=12)
plt.xlabel('t',fontsize=30)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.legend(('Mesures','Approximation'),fontsize=30, loc = 0)
plt.grid(True)
```

———————————————INTEGRALE NUMERIQUE
———————————————————

```
import numpy as np
import matplotlib.pyplot as plt
```

———————————————RECTANGLE GAUCHE
———————————————————

```
def rectangle_gauche_composite(f,a,b,n):
    h = (b-a)/n
    s = 0
```

```
    for k in np.arange(0,n):
        s += f(a+k*h)
    return h*s
```

———————————————RECTANGLE DROITE
——————————————————

```
def rectangle_droite_composite(f,a,b,n):
    h= (b-a)/n
    s = 0
    for k in np.arange(1,n+1):
        s+= f(a+k*h)
    return h*s
```

———————————————COMPOSITE TRAPEZE
——————————————————

```
def trapeze_composite(f,a,b,n):
    h= (b-a)/n
    s = (f(a)+f(b))*0.5
    for k in np.arange(1,n):
        s+= f(a+k*h)
    return h*s
```

———————————————COMPOSITE SIMPSON
——————————————————

```
def simpson_composite(f,a,b,n) :
    h=(b-a)/n
    z0=f(a)+f(b)
    z_paire=0
    z_impaire=0
    p=n//2
    for i in np.arange(1,p) :
        z_paire+=f(a+2*i*h)
    for i in np.arange(0,p) :
        z_impaire+=f(a+(2*i+1)*h)
    z=(z0+2*z_paire+4*z_impaire)/3
    return h*z
```

———————————————NOMBRE D’ITERATIONS
——————————————————

```
def integrale_precise(f,a,b,I,epsilon, methode=''):
    n=2
    val =methode(f,a,b,n)
    while (abs(val-I) >epsilon):
```

```
        n+=1
        val=methode(f,a,b,n)
    n_necessaire=n
    return n_necessaire
```

————————————————RESOLUTION DICHOTOMIE
————————————————

```
import numpy as np
import matplotlib.pyplot as plt
import sympy as sp

def dichotomie(f,a,b,epsilon,Nmax):
    k = 1
    if f(a)*f(b) > 0:
        print ('f(a) et f(b) sont de meme signe')
    else:
        while (b-a > epsilon) and (k<=Nmax):
            c = (a+b)/2
            if f(a)*f(c) < 0:
                b = c
            elif f(c)==0:
                a=c
                b=c
            else:
                a = c
            k += 1
    return ((a+b)/2, k-1)
```

————————————————RESOLUTION NEWTON
——————————————————

```
def Newton(x,f,df,epsilon,Nmax):
    E=[abs(f(x)/df(x)).evalf()]
    k=1
    while (E[-1]>epsilon) and (k<Nmax):
        k=k+1
        x=x-(f(x)/df(x))
        E.append(abs(f(x)/df(x)).evalf())
    return k,x.evalf(),E
```

———————————COMPARAISON DICHOTOMIE NEWTON
—————————————

```
Nmax=10**3
a=0
b=np.pi/4
```

```
epsilon=1/10**np.arange(2,9)
x0=np.pi/4
x=sp.symbols('x')
f=sp.Lambda([x],sp.cos(2*x)-x**2)
df=sp.Lambda([x],sp.diff(f(x),x))
It_D=[]
It_N=[]
for eps in epsilon:
    It_D.append(dichotomie(f,a,b,eps,Nmax)[1])
    It_N.append(Newton(x0,f,df,eps,Nmax)[0])

//affichage

plt.figure(figsize=(20,10))
plt.plot(epsilon,It_N,'bo-',epsilon,It_D,'r*-',markersize=12,linewidth=2)
plt.xscale('log')#axis scaling
plt.grid(True)
plt.ylabel('Nombre d\'itération',fontsize=30)
plt.xlabel('Précision',fontsize=20)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.title("Nombre d\'itération versus précision", fontsize=30, color='blue')
plt.legend(('Newton','Dichotomie'), loc=0, fontsize=30)
plt.show()
```