TP 13- Proposition de solutions

Solution 1 1. Compléter l'algorithme de la division euclidienne des polynômes A par B :

Entrée : A,B deux polynômes

$Q \leftarrow 0$
$R \leftarrow A$
Tant que $\deg(R) \geq \deg(B)$ faire
  $T \leftarrow$ le quotient des termes dominants de R et B
  $Q \leftarrow Q + T$
  $R \leftarrow R - T \times B$ (ou $A - Q \times B$)
FinTantque

Sortie : Q,R tel que $A = QB + R$ avec $\deg(R) < \deg(B)$

2. La division euclidienne :

```
def DE(A,B):
    Q=np.poly1d([0])
    R=A
    while R.order>=B.order:
        Q=Q+R[R.order]/B[B.order]*np.poly1d([1,0])**
        R=A-Q*B
    return(Q,R)
```

Cela donne

```
>>> A=np.poly1d([1,0,1,1,-1])
>>> B=np.poly1d([1,0,-1])
>>> Q,R=DE(A,B)
>>> print(Q,R)
   2
1 x + 2
1 x + 1
```

Solution 2 Euclide étendu

1. L'algorithme :

Entrée : A,B deux polynômes

$U_1 \leftarrow 1$, $V_1 \leftarrow 0$, $U_2 \leftarrow 0$ et $V_2 \leftarrow 1$
Tant que $B \neq 0$ faire
  Q et R le quotient et le reste de DE de A et B
  $U_1,V_1,U_2,V_2 \leftarrow U_2,V_2,U_1 - Q \times U_2, V_1 - Q \times V_2$
  $A,B \leftarrow B,R$
Diviser U_1,V_1 et A par le coefficient dominant de A FinTantque

Sortie : U_1,V_1,A tel que $A = QB + R$ avec $\deg(R) < \deg(B)$

2. La fonction :

```
def Bezout(A,B):
    U,V=np.poly1d([1]),np.poly1d([0])
    UU,VV=V,U
    while B.order>0 or abs(B[0])>1e-12:
        Q,R=A/B
        U,V,UU,VV,A,B=UU,VV,U-Q*UU,V-Q*VV,B,R
    U=U/A.coeffs[0]
    V=V/A.coeffs[0]
    A=A/A.coeffs[0]
    return U,V,A
```

Cela donne :

```
>>> A=np.poly1d([0,0,0,1,1,2,3],True)
>>> B=np.poly1d([1,3,4,5],True)
>>> U,V,D=Bezout(A,B)
>>> print(D)
   2
1 x - 4 x + 3
>>> print(D.roots)
[ 3.  1.]
```

Solution 3 Algorithme de Sturm

- La fonction `S(L)` :

```
def S(L):
    while len(L)>0 and abs(L[0])<1e-14:
        L.pop(0)
    a,s,i=L[0],0,0
    while i<len(L)-1:
        i=i+1
        if L[i]*a<-1e-14:
            s,a=s+1,L[i]
    return(s)
```

- Il suffit de considérer le polynôme :

$$\frac{P}{P \wedge P'}$$

Mettre en place l'algorithme d'Euclide entre P et P', puis considérer le quotient entre P et le dernier reste non nul :

```
def simple(P):
    A,B=P,P.deriv()
    while len(B)>0:
        Q,R=np.polydiv(A,B)
        A,B=B,R
    Q,R=np.polydiv(P,A)
    return(Q)
```

- La fonction `sturm(P)` :

```
def sturm(P):
    L=[P,P.deriv()]
    while len(L[-1])>0:
        Q,R=np.polydiv(L[-2],L[-1])
        L.append(-R)
    return(L)
```

- La fonction `nbzeros(P,a,b)` :

```
def nbzeros(L,a,b):
    La=[e(a) for e in L]
    Lb=[e(b) for e in L]
    return(S(La)-S(Lb))
```

- La fonction `racines(P,a,b,h,L=[])` :

```
def racines(P,a,b,h=1e-3):
    assert a<b,'ENC'
    P=simple(P)
    r=[]
    L=sturm(P)
    if abs(P(a))<1e-14:
        r.append(a)
        a=a+h
    if abs(P(b))<1e-14:
        r.append(b)
        b=b-h
    n=nbzeros(L,a,b)
    if n==0:
        return(r)
    R=[[a,b]]
    while len(R)>0:
        c,d=R.pop()
        m=(c+d)/2
        v=False
        if abs(P(m))<1e-14:
            r.append(m)
            v=True
        if v:
            m=m-h
        n=nbzeros(L,c,m)
        if n>0:
            if m-c>h:
                R.append([c,m])
            else:
                r.append((c+m)/2)
        if v:
            m=m+2*h
        n=nbzeros(L,m,d)
        if n>0:
            if d-m>h:
                R.append([m,d])
            else:
                r.append((d+m)/2)
    return(r)
```

⇛ Applications :

```
import numpy.random as rd
L=rd.randint(-4,5,size=10)
#P=np.poly1d(L)
P=np.poly1d(L,True)
P=simple(P)
n=len(P)
b=1+max([abs(P[k]/P[n]) for k in range(n)])
a=-b
print(racines(P,a,b,1e-3))
```

Vérification avec :

```
print([e for e in list(np.roots(P))
       if abs(e.imag)<1e-14])
```

Solution 4 Interpolation de Lagrange

```
def lagrange(X,Y,t):
    S=0
    for i,y in enumerate(Y):
        L=np.prod([(t-x)/(X[i]-x) for j,x in enumerate(
        S=S+y*L
    return S
```

```
def interpolation(f,a,b,n,xm,xM,ym,yM):
    X=[a+i*(b-a)/(n-1) for i in range(n)]
    Y=[f(e) for e in X]
    x=np.arange(xm,xM,0.01)
    y1=[f(e) for e in x]
    y2=[lagrange(X,Y,e) for e in x]
    plt.plot([xm,xM],[0,0],'k-')
    plt.plot([0,0],[ym,yM],'k-')
    plt.plot(x,y1,'r-',lw=3)
    plt.plot(x,y2,'b-',lw=2)
    plt.plot(X,Y,'go',lw=2)
    plt.xlim(xm,xM)
    plt.ylim(ym,yM)
```

Solution 5 Subdivision de Chebychev
Essentiellement, ce sont uniquement deux lignes à changer.

```
def interpolation_T(f,a,b,n,xm,xM,ym,yM):
    r=[np.cos((2*(n-k)-1)*np.pi/(2*n))
                                for k in range(n)]
    X=[(e+1)*(b-a)/2+a for e in r]
    ...
```