Séance 16 :

Lignes de niveau

1. Préparation des fonctions élémentaires

On pose $f(x, y) = x^2 + 2y^2 + 2xy$

(a) ***Enregistrer la fonction*** `f(u)` ***qui prend pour seule entrée la liste ou tableau*** `u = [x, y]` ***et retourne*** $f(x, y)$. J'importe d'abord les modules :

```
> main.py
import numpy as np
import matplotlib.pylab as plt
```

Puis je code f :

```
> main.py
def f(u):
    x, y = u[0], u[1]
    return x**2 + 2*y**2 + 2*x*y
```

(b) ***Enregistrer*** `Nabla(u)` ***qui prend pour seule entrée la liste ou tableau*** `u = [x, y]` ***et retourne le vecteur*** $\nabla f(x, y)$ ***sous forme de tableau de type*** `array` ***de*** `numpy`.

```
> main.py
def nabla(u):
    x, y = u[0], u[1]
    a = 2*x + 2*y
    b = 4*y + 2*x
    return np.array([a, b])
```

(c) ***Enregistrer*** `Vunit(u)` ***qui prend pour seule entrée la liste ou tableau*** `u = [x, y]` ***eet retourne le vecteur unitaire*** $\vec{\omega} = \frac{\vec{u}}{||\vec{u}||}$ ***sous forme de tableau de type*** `array` ***de*** `numpy`.

```
> main.py
def Vunit(u):
    x, y = u[0], u[1]
    norme = np.sqrt(x**2+y**2)
    return np.array([x/norme, y/norme])
```

2. Calculer une ligne

Construire `points(n, pas, x, y)` ***qui renvoie une liste*** $A = [A_i]_{i \in [\![o, n-1]\!]}$ ***contenant*** n ***points*** A_i ***sur la ligne de niveau*** C_k ***où*** $k = f(A_0)$ ***en commençant par*** $A_0 = [x, y]$ ***et où tous les points suivants*** A_i ***seront construits par récurrence de la manière suivante :***

(a) ***En*** A_i ***sur la ligne de niveau*** C_0 ***on calcule vecteur tangent unitaire*** $\vec{T}$ ***à partir de*** $\nabla f(A_i)$

(b) ***On calcule le point intermédiaire*** B_i ***tel que*** $\overrightarrow{A_iB_i} =$ `pas` $\times \vec{T}$

(c) ***On calcule*** A_{i+1} ***tel que*** $\overrightarrow{OA_{i+1}} = \overrightarrow{OB_i} + \lambda \nabla f(B_i)$ ***où*** λ ***est tel que, d'après la formule de Taylor-Young à l'ordre 1, le point*** A_{i+1} ***soit sur la ligne de niveau*** C_k ***où*** $k = f(A_0)$***. La valeur de*** λ ***est supposée être faible devant celle du*** `pas`.

Par exemple, pour 10 points avec un pas de 0.5 à partir de (1, 0) on trouve `A = points(10, 0.5, 1, 0)` $\Rightarrow$ A = [[1.0, 0.0], [0.62, 0.32], [0.21, 0.6], [-0.24, 0.82], [-0.71, 0.97], [-1.19, 0.99], [-1.43, 0.85], [-1.44, 0.52], [-1.16, 0.15], [-0.8, -0.19]]

❏ Je crée le fichier `test/exceptedOutput.py` dans lequel je mets les résultats attendus pour tester ma fonction à la fin :

> test/exceptedOutput.py

```python
pointsOuput = [
    [1.0, 0.0],
    [0.62, 0.32],
    [0.21, 0.6],
    [-0.24, 0.82],
    [-0.71, 0.97],
    [-1.19, 0.99],
    [-1.43, 0.85],
    [-1.44, 0.52],
    [-1.16, 0.15],
    [-0.8, -0.19]
]
```

- ❏ J'importe les résultats dans le fichier `main.py`

> main.py

```python
from test.exceptedOutput import pointsOuput
```

- ❏ Je détermine λ : On a : $f(A_{i+1}) = k$, donc d'après la formule de Taylor à l'ordre 1 :

$$k = f(A_{i+1}) = f(B_i) + \nabla f(B_i) \cdot \overrightarrow{B_iA_{i+1}}$$

Or : $\overrightarrow{B_iA_{i+1}} = \lambda \nabla f(B_i)$
Donc : $\nabla f(B_i) \cdot \overrightarrow{B_iA_{i+1}} = \nabla f(B_i) \cdot \lambda \nabla f(B_i)$
D'où :

$$\boxed{\lambda = \frac{k - f(B_i)}{||\nabla f(B_i)||^2}}$$

- ❏ Je code la fonction `points`

> main.py

```python
def points(n, pas, x, y):
    A = [np.array([x, y])]
    k = f(A[0])
    for i in range(n-1):
        Tx, Ty = Vunit(nabla(A[i]))
        T = np.array([-Ty, Tx])
        Bi = A[i] + pas * T
        Vn = nabla(Bi)
        Lambda = (k-f(Bi))/(Vn[0]**2+Vn[1]**2)
        Ai = Bi + Lambda * Vn
        A.append(list(Ai))
    A[0] = list(A[0])
    return A
```

- ❏ Je code un fonction `around` pour vérifier mes résultats :

> helpers/help.py

```python
def around(A):
    return list(np.around(np.array(A), 2))
```

- ❏ Je teste tout ça :

> main.py

```python
# Imports ---------------------------------------------------------------
from helpers.help import around
from test.exceptedOutput import pointsOuput

# Calculs des points ----------------------------------------------------
A = points(10, 0.5, 1, 0)
print(around(A) == pointsOuput)
```

Et le console affiche `True` !

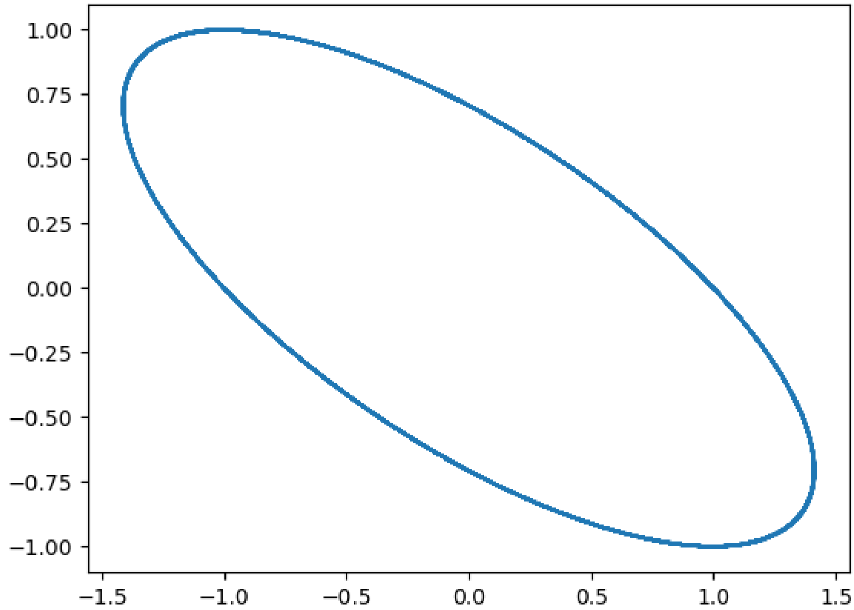
3. Dessin de ligne de niveau

A partir de `A = points(n, pas, 1, 0)` ***avec*** `n = 3 000` ***et*** `pas = 0.01`***, Tacer la ligne de niveau*** C_1 ***qui passe par le point*** $A_0 = [1, 0]$***.***

```
> main.py
```

```python
def traceLigne(A):
    A = np.array(A)
    X, Y = A[:, 0], A[:, 1]
    plt.plot(X, Y)
    plt.show()


A = points(3000, 0.1, 1, 0)
traceLigne(A)
```



4. Dichotomie

Donner la méthode de dichotomie `dicho(g,a,b)` ***qui retourne à la précision*** 10^{-6} ***près une solution de*** $g(x) = 0$ ***sur un intervalle*** $[a, b]$ ***si*** $g(a)g(b) \leq 0$ ***sinon elle ne retourne rien.***

```
> main.py
```

```python
def dicho(g, a, b):
    if (g(a)*g(b) <= 0):
        return None

    PRES = 10**(-6)

    # On remet dans l’ordre au cas ou
    if (a > b):
        a, b = b, a

    while(b-a > PRES):
        m = (a+b)/2
        if(g(a)*g(m) <= 0):
            b = m
        else:
            a = m
    return m
```

5. Test de la dichotomie

Tester votre fonction en donnant après reconnaissance graphique les 3 solutions de l'équation $e^x = 3x^2$ ***à*** 10^{-6} ***près.***
Vérifier que le produit des solutions vaut : `-1.559156`.

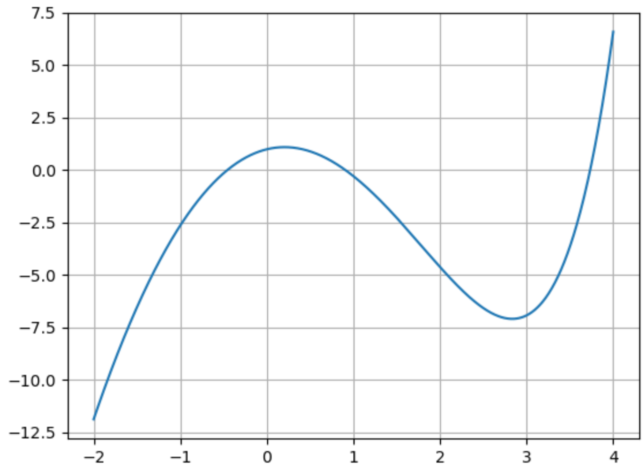
- Je trace la fonction :

> main.py

```python
def TraceFunction(f, borneMax, borneMin, nbPoints):
    X = np.linspace(borneMin, borneMax, nbPoints)
    Y = [f(x) for x in X]
    plt.plot(X, Y)
    plt.grid()
    plt.show()

def h(x): return np.exp(x)-3*x**2

TraceFunction(h, -2, 4, 1000)
```



- J'en déduit les 3 intervalles :

$$\begin{cases} [-1, 0] \\ [0, 1] \\ [3, 4] \end{cases}$$

- Je résous grâce à `dicho` :

> main.py

```python
def resoudre():
    a = dicho(h, -1, 0)
    b = dicho(h, 0, 1)
    c = dicho(h, 3, 4)
    return a, b, c

a, b, c = resoudre()
A = int(a*b*c*10**6)/10**6
print(A == -1.559156)
```

Ce qui affiche `True`.

6. Dichotomie améliorée

Améliorer la fonction `dicho` ***pour qu'elle retourne une solution si elle existe de*** $g(x) = 0$ ***à*** 10^{-6} ***près en cherchant d'abord un intervalle*** $[c, d] \subset [a, b]$ ***tel que*** $d - c \leq \frac{b-a}{100}$ ***et*** $g(c)g(d) \leq 0$***. Si aucun***

changement de signe $g(c)g(d) \leq 0$ de ce type n'est repéré, la fonction renvoie à nouveau l'ensemble vide.

```
> main.py
def racine(f, a, b):
    if (a > b):
        a, b = b, a
    r, i = None, 0
    while (r == None):
        c = a + (b-a)*i/100
        d = a + (b-a)*(i+1)/100
        r = dicho(f, c, d)
        i += 1
    if (r != None):
        return r
```

7. Tracer un exemple

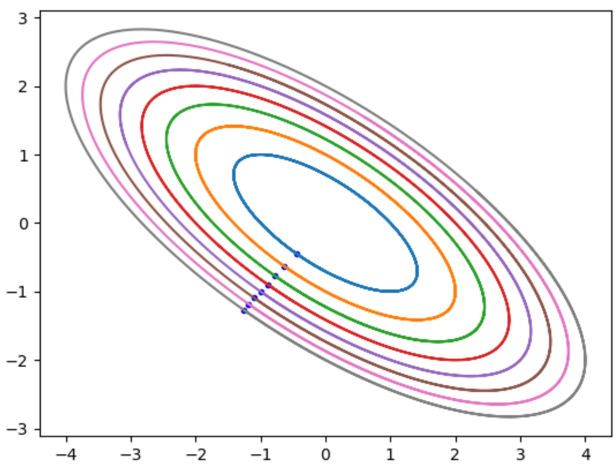
Ecrire une procédure `lignes(n, k0, k1)` qui dessine les lignes de niveau de $f(x, y) = x^2+2y^2+2xy = k$ pour k allant de `k0` à `k1` en recherchant les départs de la ligne de niveau sur la droite $y = x$ du plan.

En supprimant la ligne `plt.show()` de la fonction `traceLinge()` :

```
> main.py
def lignes(n, k0, k1):
    pas = 0.01
    k = k0
    while(k <= k1):
        def g(x):
            return f([x, x]) - k
        c = racine(g, -10, 10)
        if (c != None):
            plt.plot(c, c, 'b.')
            traceLigne(points(n, pas, c, c))
        k += 1

lignes(3000, 1, 8)
plt.show()
```

Ce qui donne :



8. Un autre exemple

Dessiner les lignes de niveau de $h(x,y) = x^4 + y^4 - 4xy$

❏ On redéfinit `h` :

> main.py

```
def h(u):
    x, y = u[0], u[1]
    return x**4 + y**4 - 4*x*y
```

❏ On redéfinit `nabla` qui doit maintenant retourner $\nabla h(x,y)$

> main.py

```
def nablah(u):
    x, y = u[0], u[1]
    a = 4*x**3 - 4*y
    b = 4*y**3 - 4*x
    return np.array([a, b])
```

❏ On pense à changer `f` en `h` dans `points` et `ligne`. Puis on affiche :

> main.py

```
lignes(3000, -1.5, 2.5)
plt.show()
```

Ce qui donne :