MEMBRES DU GROUPE

Noms et prenoms	Matricule
KAPTCHOUANG YVAN DERICK	20U2869
NGEUKEU MEKENGUIM ROBINSON RADLEY	20U2628
ABEL DJIDA	19M2203
NDJOMNANG NJOMOU Daniel junior	19H2963

INTRODUCTION

Dans l'optique de la manipulation des algorithmes concourants à la minimisation des fonctions sans contrainte , Nous avons appliqué de façon pratique ces différents algorithmes sur une fonction à deux variables dans le langage de programmation python tout en relevant une comparaison sommaire entre ces algorithmes.

$$\mathbf{f(x,y)} = 0.5x \text{ ** } 2 + 3.5y \text{ ** } 2$$

I- Algorithme de descente du gradient à pas fixe

Ici , on quitte d'un point initial ou de départ et on se fixe un pas selon notre choix et on choisit la meilleur descente (gradient de la fonction) pour atteindre un point critique tout en vérifiant à chaque fois que $f(Xk + 1) < f(Xk)$ et on calcule

$$x_{k+1} = x_k - s\nabla f(x_k).$$

IMPLÉMENTATION DE L'ALGORITHME SOUS PYTHON

➔ on importe d'abord les bibliothèques à utilisées

```
import numpy as np
import matplotlib.pyplot as plt
import math
import pandas as pd
```

➔ on définit les fonctions intermédiaires que nou aurons à utiliser

```
#declaration de la fonction de depart
f_x_y = lambda x , y : 0.5*(x**2)+ (3.5)*(y**2)

#calcul du gradiant de la fonction
def gradient_f_x_y(x , y ):
    return  x , 7*y ;

def point_suivant(x , y , s):
    gradient_x ,gradient_y = gradient_f_x_y(x , y)
    return x - s*gradient_x , y - s*gradient_y

def direction_newton( x ,y):
    return -x , -y
```

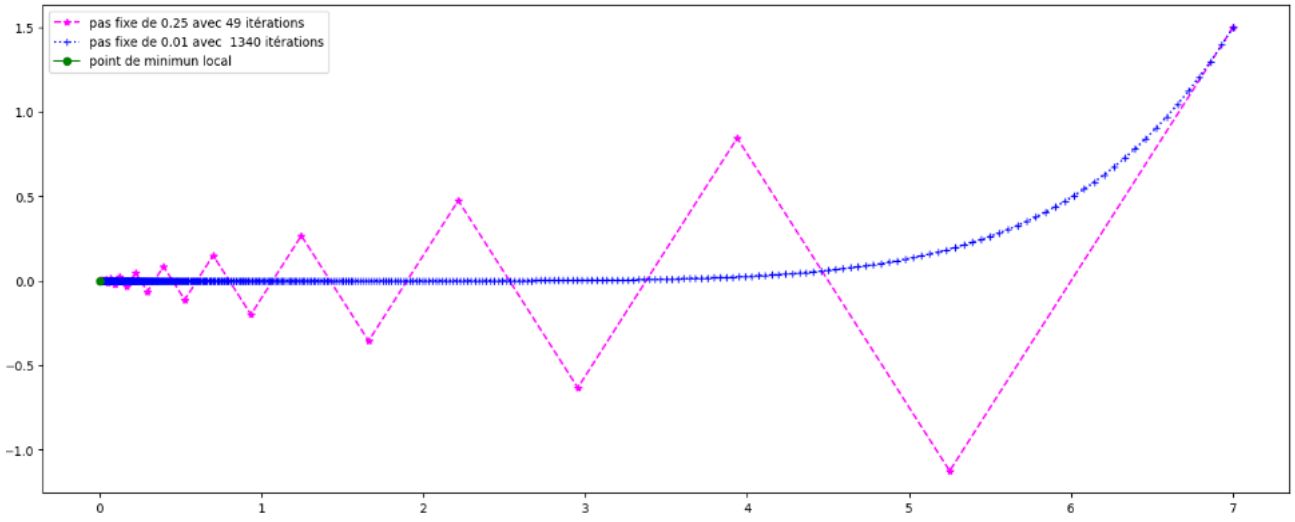
➔ Algorithme de descente proprement dit

```
def pas_fixe(s) :

    #initialisation des paramètre
    tolerence = 10**(-5)
    i = 1
    x0 = 7
    y0 = 1.5
    abscis = [x0]
    ordone = [y0]
    x,y = point_suivant(x0 , y0 , s)
    pas = [s]
    # calcul du gradient suivant X et Y en un point précis
    gradient_x ,gradient_y = gradient_f_x_y(x, y)
    # calcul de la norme du gradient suivant X et Y en ce point précis
    norm_gradient = math.sqrt(gradient_x**2 + gradient_y**2)
    while norm_gradient > tolerence and
         abs((f_x_y(x,y) - f_x_y(abscis[i-1] , ordone[i-1])))/ abs(f_x_y(abscis[i-1] , ordone[i-1])) > tolerence :
        abscis.append(x)
        ordone.append(y)
        x  , y = point_suivant(x ,y , s)
        gradient_x ,gradient_y = gradient_f_x_y(x, y)
        norm_gradient = math.sqrt(gradient_x**2 + gradient_y**2) # norme du gradient
        i += 1

    return abscis ,ordone ,i
```

Cet algorithme prend en paramètre un pas aléatoire et retourne une liste de vecteurs correspondants aux différents vecteurs visités.

Comme condition d'arrêt , nous avons utilisé la stagnation de la valeurs courante donnée par : $|f(x_{k+1}) - f(x_k)| < \varepsilon |f(x_k)|$

Le tracé pour un pas = 0.25 puis pas = 0.01 nous donne la visualisation suivante :



LES TABLEAUX LISTANT LES DIFFÉRENTES VALEURS EST :

Tableau récapitulatif de l`algorithme de descente pour un pas fixe de 0.25

	F(x,y)	||Vf(x,y)||	Sk	Xk	Yk
0	3.237500e+01	12.619429	0.25	7.000000	1.500000
1	1.821094e+01	9.464572	0.25	5.250000	-1.125000
2	1.024365e+01	7.098429	0.25	3.937500	0.843750
3	5.762054e+00	5.323822	0.25	2.953125	-0.632812
4	3.241156e+00	3.992866	0.25	2.214844	0.474609
5	1.823150e+00	2.994650	0.25	1.661133	-0.355957
6	1.025522e+00	2.245987	0.25	1.245850	0.266968
7	5.768561e-01	1.684490	0.25	0.934387	-0.200226
8	3.244815e-01	1.263368	0.25	0.700790	0.150169
9	1.825209e-01	0.947526	0.25	0.525593	-0.112627
10	1.026680e-01	0.710644	0.25	0.394195	0.084470
11	5.775074e-02	0.532983	0.25	0.295646	-0.063353
12	3.248479e-02	0.399737	0.25	0.221734	0.047515
13	1.827270e-02	0.299803	0.25	0.166301	-0.035636
14	1.027839e-02	0.224852	0.25	0.124726	0.026727
15	5.781595e-03	0.168639	0.25	0.093544	-0.020045
16	3.252147e-03	0.126479	0.25	0.070158	0.015034
17	1.829333e-03	0.094860	0.25	0.052619	-0.011275
18	1.029000e-03	0.071145	0.25	0.039464	0.008457
19	5.788123e-04	0.053359	0.25	0.029598	-0.006342
20	3.255819e-04	0.040019	0.25	0.022198	0.004757
21	1.831398e-04	0.030014	0.25	0.016649	-0.003568
22	1.030162e-04	0.022511	0.25	0.012487	0.002676
23	5.794659e-05	0.016883	0.25	0.009365	-0.002007
24	3.259496e-05	0.012662	0.25	0.007024	0.001505
25	1.833466e-05	0.009497	0.25	0.005268	-0.001129
26	1.031325e-05	0.007123	0.25	0.003951	0.000847
27	5.801202e-06	0.005342	0.25	0.002963	-0.000635
28	3.263176e-06	0.004006	0.25	0.002222	0.000476
29	1.835537e-06	0.003005	0.25	0.001667	-0.000357
30	1.032489e-06	0.002254	0.25	0.001250	0.000268
31	5.807753e-07	0.001690	0.25	0.000938	-0.000201
32	3.266861e-07	0.001268	0.25	0.000703	0.000151
33	1.837609e-07	0.000951	0.25	0.000527	-0.000113
34	1.033655e-07	0.000713	0.25	0.000396	0.000085
35	5.814310e-08	0.000535	0.25	0.000297	-0.000064
36	3.270550e-08	0.000401	0.25	0.000222	0.000048
37	1.839684e-08	0.000301	0.25	0.000167	-0.000036
38	1.034822e-08	0.000226	0.25	0.000125	0.000027
39	5.820876e-09	0.000169	0.25	0.000094	-0.000020
40	3.274243e-09	0.000127	0.25	0.000070	0.000015
41	1.841761e-09	0.000095	0.25	0.000053	-0.000011
42	1.035991e-09	0.000071	0.25	0.000040	0.000008
43	5.827448e-10	0.000054	0.25	0.000030	-0.000006
44	3.277940e-10	0.000040	0.25	0.000022	0.000005
45	1.843841e-10	0.000030	0.25	0.000017	-0.000004
46	1.037161e-10	0.000023	0.25	0.000013	0.000003
47	5.834028e-11	0.000017	0.25	0.000009	-0.000002
48	3.281641e-11	0.000013	0.25	0.000007	0.000002

Tableau récapitulatif de l`algorithme de descente pour un pas fixe de 0.01

	F(x,y)	||Vf(x,y)||	Sk	Xk	Yk
0	3.237500e+01	12.619429	0.01	7.000000	1.500000e+00
1	3.082354e+01	11.974144	0.01	6.930000	1.395000e+00
2	2.942551e+01	11.381649	0.01	6.860700	1.297350e+00
3	2.816131e+01	10.838044	0.01	6.792093	1.206535e+00
4	2.701395e+01	10.339651	0.01	6.724172	1.122078e+00
...	...	...	...	...	...
1335	5.434201e-11	0.000010	0.01	0.000010	1.261327e-42
1336	5.326060e-11	0.000010	0.01	0.000010	1.173034e-42
1337	5.220071e-11	0.000010	0.01	0.000010	1.090921e-42
1338	5.116192e-11	0.000010	0.01	0.000010	1.014557e-42
1339	5.014380e-11	0.000010	0.01	0.000010	9.435380e-43

[1340 rows x 5 columns]

II- Algorithme de descente à pas optimal

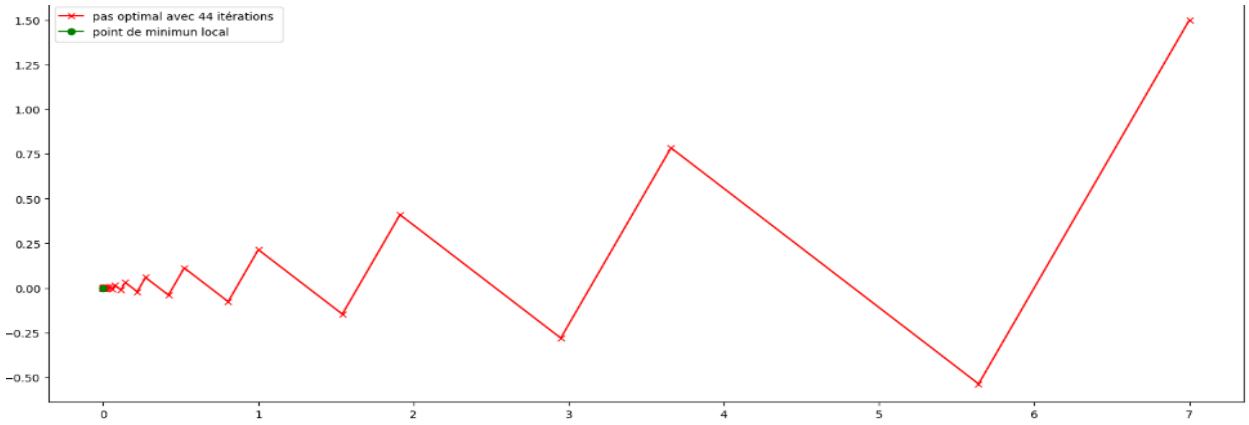
Le principe est le même que celui du pas fixe à la seule différence que nous aurons chaque fois à calculer le pas optimal qui pour notre fonction est donnée par :

: $s_k = (x_k^2 + 7^2 y_k^2)/(x_k^2 + 7^3 y_k^2)$. et $x_{k+1} = x_k + \frac{x_k^2 + 7^2 y_k^2}{x_k^2 + 7^3 y_k^2} \begin{pmatrix} -x_k \\ -7y_k \end{pmatrix}$

➔ L'algorithme sous python est donnée par :

```
def pas_optimal() :

    #initialisation des paramètre
    tolerence = 10**(-5)
    i = 1
    x0 = 7
    y0 = 1.5
    abscis = [x0]
    ordone = [y0]
    s = (x0**2 + 49*(y0**2))/( x0**2 + 343*(y0**2))
    pas   = [0]
    x,y = point_suivant(x0 , y0 , s)
    # calcul du gradient suivant X et Y en un point précis
    gradient_x ,gradient_y = gradient_f_x_y(x0 , y0)
    # calcul de la norme du gradient suivant X et Y en ce point précis
    norm_gradient = math.sqrt(gradient_x**2 + gradient_y**2)
    while norm_gradient > tolerence and
    abs(f_x_y(x,y) - f_x_y(abscis[i-1] , ordone[i-1]))/abs((f_x_y(abscis[i-1] , ordone[i-1]))) > tolerence :
        abscis.append(x)
        ordone.append(y)
        pas.append(s)
         # calcul du gradient suivant X et Y en un point précis
        gradient_x ,gradient_y = gradient_f_x_y(x , y)
        norm_gradient = math.sqrt(gradient_x**2 + gradient_y**2) # norme du gradient
        s = (x**2 + 49*(y**2))/( x**2 + 343*(y**2))
        x  , y = point_suivant(x ,y , s)
        i += 1
    return abscis ,ordone ,i ,pas
```

➔ Le tracé de la courbe nous donne la visualisation suivante :



LE TABLEAU LISTANT LES DIFFÉRENTES VALEURS EST :

Tableau récapitulatif de l`algorithme de descente à pas optimal

	F(x,y)	||Vf(x,y)||	Sk	Xk	Yk
0	3.237500e+01	12.619429	0.000000	7.000000	1.500000e+00
1	1.692537e+01	6.780589	0.194030	5.641791	-5.373134e-01
2	8.848440e+00	6.597330	0.351351	3.659540	7.841872e-01
3	4.625889e+00	3.544834	0.194030	2.949480	-2.809029e-01
4	2.418375e+00	3.449028	0.351351	1.913176	4.099663e-01
5	1.264306e+00	1.853209	0.194030	1.541963	-1.468536e-01
6	6.609683e-01	1.803122	0.351351	1.000192	2.143269e-01
7	3.455486e-01	0.968842	0.194030	0.806125	-7.677382e-02
8	1.806498e-01	0.942657	0.351351	0.522892	1.120483e-01
9	9.444219e-02	0.506502	0.194030	0.421435	-4.013669e-02
10	4.937357e-02	0.492813	0.351351	0.273363	5.857788e-02
11	2.581208e-02	0.264795	0.194030	0.220323	-2.098312e-02
12	1.349433e-02	0.257638	0.351351	0.142912	3.062401e-02
13	7.054723e-03	0.138433	0.194030	0.115183	-1.096980e-02
14	3.688149e-03	0.134691	0.351351	0.074713	1.600997e-02
15	1.928133e-03	0.072371	0.194030	0.060217	-5.734915e-03
16	1.008011e-03	0.070415	0.351351	0.039059	8.369877e-03
17	5.269797e-04	0.037835	0.194030	0.031481	-2.998165e-03
18	2.755005e-04	0.036813	0.351351	0.020420	4.375700e-03
19	1.440293e-04	0.019780	0.194030	0.016458	-1.567415e-03
20	7.529728e-05	0.019245	0.351351	0.010675	2.287579e-03
21	3.936477e-05	0.010341	0.194030	0.008604	-8.194311e-04
22	2.057957e-05	0.010061	0.351351	0.005581	1.195926e-03
23	1.075882e-05	0.005406	0.194030	0.004498	-4.283916e-04
24	5.624620e-06	0.005260	0.351351	0.002918	6.252201e-04
25	2.940503e-06	0.002826	0.194030	0.002352	-2.239595e-04
26	1.537270e-06	0.002750	0.351351	0.001525	3.268597e-04
27	8.036716e-07	0.001478	0.194030	0.001229	-1.170841e-04
28	4.201526e-07	0.001438	0.351351	0.000797	1.708795e-04
29	2.196522e-07	0.000772	0.194030	0.000643	-6.121056e-05
30	1.148323e-07	0.000752	0.351351	0.000417	8.933433e-05
31	6.003334e-08	0.000404	0.194030	0.000336	-3.200036e-05
32	3.138492e-08	0.000393	0.351351	0.000218	4.670322e-05
33	1.640777e-08	0.000211	0.194030	0.000176	-1.672951e-05
34	8.577840e-09	0.000205	0.351351	0.000114	2.441605e-05
35	4.484421e-09	0.000110	0.194030	0.000092	-8.746046e-06
36	2.344417e-09	0.000107	0.351351	0.000060	1.276450e-05
37	1.225641e-09	0.000058	0.194030	0.000048	-4.572358e-06
38	6.407547e-10	0.000056	0.351351	0.000031	6.673171e-06
39	3.349811e-10	0.000030	0.194030	0.000025	-2.390390e-06
40	1.751252e-10	0.000029	0.351351	0.000016	3.488677e-06
41	9.155398e-11	0.000016	0.194030	0.000013	-1.249675e-06
42	4.786364e-11	0.000015	0.351351	0.000009	1.823850e-06
43	2.502270e-11	0.000008	0.194030	0.000007	-6.533196e-07

III- Méthode de Newton locale

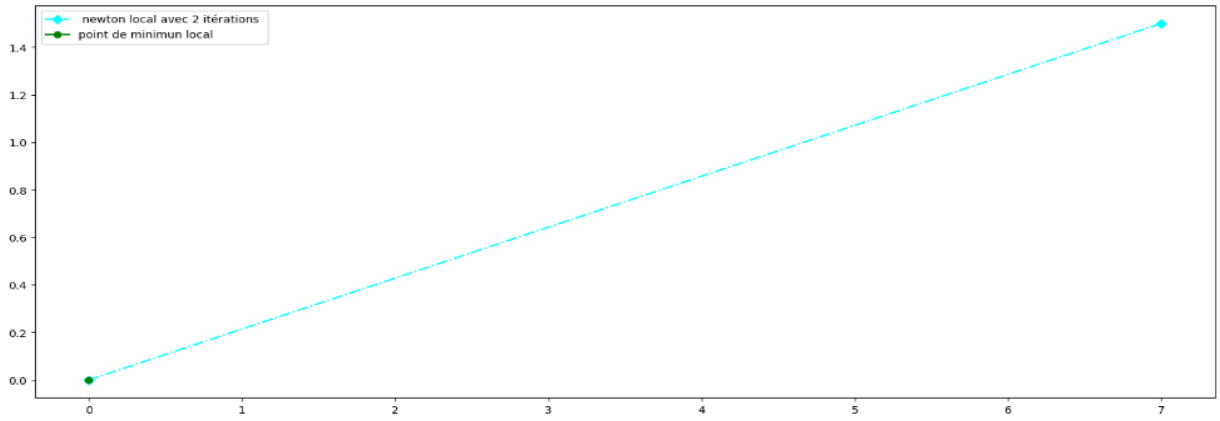
Ici , la direction dk est calculé a chaque fois et est solution de l'équation linéaire

$$H[f](x_k)d_k = -\nabla f(x_k)$$.

➔ L'Algorithme sous python est donnée par :

```python
def newton_local() :

    #initialisation des paramètre
    tolerence = 10**(-5)
    i = 1
    x0 = 7
    y0 = 1.5
    abscis = [7]
    ordone = [1.5]
    dx ,dy = direction_newton(abscis[i-1] , ordone[i-1])
    x,y  = x0+dx , y0+dy
    # calcul du gradient suivant X et Y en un point précis
    gradient_x ,gradient_y = gradient_f_x_y(abscis[i-1] , ordone[i-1])
    # calcul de la norme du gradient suivant X et Y en ce point précis
    norm_gradient = math.sqrt(gradient_x**2 + gradient_y**2)
    while norm_gradient > tolerence:
        abscis.append(x)
        ordone.append(y)
         # calcul du gradient suivant X et Y en un point précis
        gradient_x ,gradient_y = gradient_f_x_y(x , y)
        norm_gradient = math.sqrt(gradient_x**2 + gradient_y**2) # norme du gradient
        dx ,dy = direction_newton(x,y)
        x,y  = x0+dx , y0+dy
        i += 1
    return abscis ,ordone ,i
```

➔ Le tracé de la courbe nous donne la visualisation suivante :

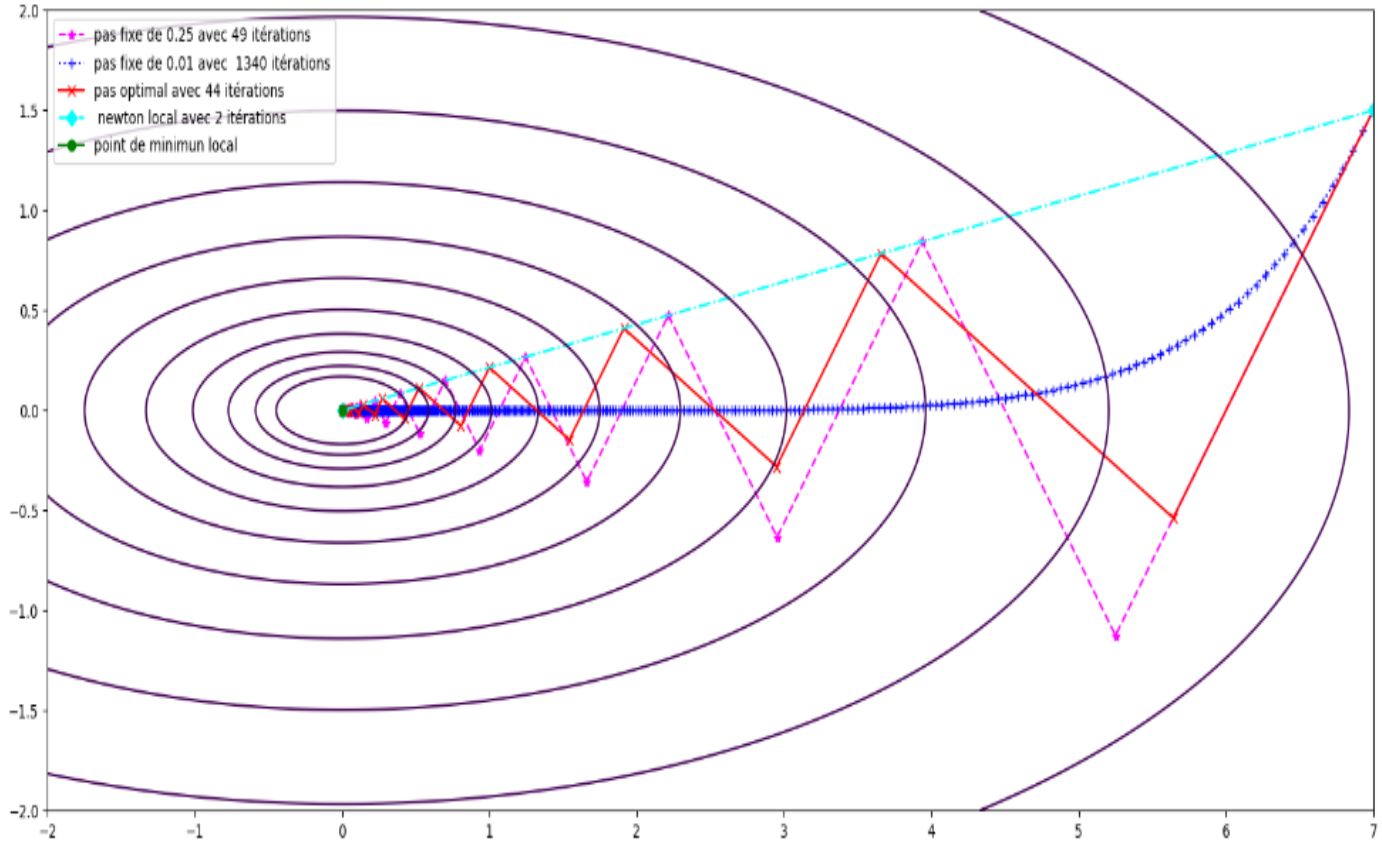


LE TABLEAU LISTANT LES DIFFÉRENTES VALEURS EST :

Tableau récapitulatif de l`algorithme par la méthode de Newton locale

	F(x,y)	||Vf(x,y)||	Sk	Xk	Yk
0	32.375	12.619429	1	7	1.5
1	0.000	0.000000	1	0	0.0

IV- Fonction finale



V- CODE PRINCIPAL

```python
import numpy as np
import matplotlib.pyplot as plt
import math
import pandas as pd
#declaration de la fonction de depart
f_x_y = lambda x , y : 0.5*(x**2)+ (3.5)*(y**2)

#calcul du gradiant de la fonction
def gradient_f_x_y(x , y ):
    return  x , 7*y ;

def point_suivant(x , y , s):
    gradient_x ,gradient_y = gradient_f_x_y(x , y)
    return x - s*gradient_x , y - s*gradient_y

def direction_newton( x ,y):
    return -x , -y
def pas_fixe(s) :

    #initialisation des paramètre
    tolerence = 10**(-5)
    i = 1
    x0 = 7
    y0 = 1.5
    abscis = [x0]
    ordone = [y0]
    x,y = point_suivant(x0 , y0 , s)
    pas = [s]
    # calcul du gradient suivant X et Y en un point précis
    gradient_x ,gradient_y = gradient_f_x_y(x, y)
    # calcul de la norme du gradient suivant X et Y en ce point précis
    norm_gradient = math.sqrt(gradient_x**2 + gradient_y**2)
    while norm_gradient > tolerence and abs((f_x_y(x,y) - f_x_y(abscis[i-1] , ordone[i-1])))/ abs(f_x_y(abscis[i-1] , ordor
        abscis.append(x)
```

```python
        ordone.append(y)
        x  , y = point_suivant(x ,y , s)
        gradient_x ,gradient_y = gradient_f_x_y(x, y)
        norm_gradient = math.sqrt(gradient_x**2 + gradient_y**2) # norme du gradient
        i += 1

    return abscis ,ordone ,i

def pas_optimal() :

    #initialisation des paramètre
    tolerence = 10**(-5)
    i = 1
    x0 = 7
    y0 = 1.5
    abscis = [x0]
    ordone = [y0]
    s = (x0**2 + 49*(y0**2))/( x0**2 + 343*(y0**2))
    pas   = [0]
    x,y = point_suivant(x0 , y0 , s)
    # calcul du gradient suivant X et Y en un point précis
    gradient_x ,gradient_y = gradient_f_x_y(x0 , y0)
    # calcul de la norme du gradient suivant X et Y en ce point précis
    norm_gradient = math.sqrt(gradient_x**2 + gradient_y**2)
    while norm_gradient > tolerence and abs(f_x_y(x,y) - f_x_y(abscis[i-1] , ordone[i-1]))/abs((f_x_y(abscis[i-1] , ordone[
        abscis.append(x)
        ordone.append(y)
        pas.append(s)
         # calcul du gradient suivant X et Y en un point précis
        gradient_x ,gradient_y = gradient_f_x_y(x , y)
        norm_gradient = math.sqrt(gradient_x**2 + gradient_y**2) # norme du gradient
        s = (x**2 + 49*(y**2))/( x**2 + 343*(y**2))
        x  , y = point_suivant(x ,y , s)
        i += 1
```

```
    return abscis ,ordone ,i ,pas


def newton_local() :

    #initialisation des paramètre
    tolerence = 10**(-5)
    i = 1
    x0 = 7
    y0 = 1.5
    abscis = [7]
    ordone = [1.5]
    dx ,dy = direction_newton(abscis[i-1] , ordone[i-1])
    x,y  = x0+dx , y0+dy
    # calcul du gradient suivant X et Y en un point précis
    gradient_x ,gradient_y = gradient_f_x_y(abscis[i-1] , ordone[i-1])
    # calcul de la norme du gradient suivant X et Y en ce point précis
    norm_gradient = math.sqrt(gradient_x**2 + gradient_y**2)
    while norm_gradient > tolerence:
        abscis.append(x)
        ordone.append(y)
         # calcul du gradient suivant X et Y en un point précis
        gradient_x ,gradient_y = gradient_f_x_y(x , y)
        norm_gradient = math.sqrt(gradient_x**2 + gradient_y**2) # norme du gradient
        dx ,dy = direction_newton(x,y)
        x,y  = x0+dx , y0+dy
        i += 1
    return abscis ,ordone ,i

if __name__ == "__main__":

    pas_1 = 0.25
    pas_2 = 0.01
    points_abscisse_1 , points_ordonnée_1  , i1 = pas_fixe(pas_1)
    points_abscisse_2 , points_ordonnée_2  , i2 = pas_fixe(pas_2)
    points_abscisse_4 , points_ordonnée_4  , i4 = newton_local()

    #on trace la courbe avec le second pas
    plt.figure(figsize=(19,8))
    plt.plot(points_abscisse_1 ,points_ordonnée_1 ,'*--', color='magenta',label= f'pas fixe de {pas_1} avec {i1} itérations
    #on trace la courbe avec le premier pas
    plt.plot( points_abscisse_2 ,points_ordonnée_2 ,'+:',color = 'blue' ,label= f'pas fixe de {pas_2} avec  {i2} itérations

     #on trace la courbe grace au pas optimal calculé à chaque fois
    plt.plot( points_abscisse_3 ,points_ordonnée_3 ,'x-' , color = 'red', label= f'pas optimal avec {i3} itérations ')
    plt.plot( points_abscisse_4 ,points_ordonnée_4 ,'D-.' , color = 'cyan', label= f' newton local avec {i4} itérations ')

     #on place le point critique sur la courbe
    plt.plot(points_abscisse_1[-1] , points_ordonnée_1[-1] ,color = 'green', marker = "o" , label ='point de minimun local'

    X , Y = np.meshgrid( np.linspace(-2 , 7 ,1000) ,np.linspace(-2 , 2 ,1000))
    Z = f_x_y( X, Y)
    cs = plt.contour(X,Y,Z,np.logspace(-1,3.5,20,base=10))
    plt.legend()

    #tracée des tableaux
    i = 0
    j = 0
    k = 0
    n = 0
    liste_1_f = []
    liste_1_norm_f = []
    liste_2_f = []
    liste_2_norm_f = []
    liste_3_f = []
    liste_3_norm_f = []
    liste_4_f = []
    liste_4_norm_f = []
```

```
#Tableau  récapitulatif de l`algorithme de descente  pour un pas fixe de 0.25
while i < len(points_abscisse_1):
    f = 0.5*(points_abscisse_1[i]**2) + 3.5*(points_ordonnée_1[i]**2)
    gradient_x ,gradient_y = gradient_f_x_y(points_abscisse_1[i] , points_ordonnée_1[i])
    norm_f = math.sqrt(gradient_x**2 + gradient_y**2)
    liste_1_f.append(f)
    liste_1_norm_f.append(norm_f)
    i += 1

#Tableau  récapitulatif de l`algorithme de descente  pour un pas fixe de 0.01
while k < len(points_abscisse_2):
    f = 0.5*(points_abscisse_2[k]**2) + 3.5*(points_ordonnée_2[k]**2)
    gradient_x ,gradient_y = gradient_f_x_y(points_abscisse_2[k] , points_ordonnée_2[k])
    norm_f = math.sqrt(gradient_x**2 + gradient_y**2)
    liste_2_f.append(f)
    liste_2_norm_f.append(norm_f)
    k += 1

#Tableau  récapitulatif de l`algorithme de descente  à  pas  optimal
while j < len(points_abscisse_3):
    f = 0.5*(points_abscisse_3[j]**2) + 3.5*(points_ordonnée_3[j]**2)
    gradient_x ,gradient_y = gradient_f_x_y(points_abscisse_3[j] , points_ordonnée_3[j])
    norm_f = math.sqrt(gradient_x**2 + gradient_y**2)
    liste_3_f.append(f)
    liste_3_norm_f.append(norm_f)
    j += 1

#Tableau  récapitulatif de l`algorithme  par la méthode de Newton local
while n < len(points_abscisse_4):
    f = 0.5*(points_abscisse_4[n]**2) + 3.5*(points_ordonnée_4[n]**2)
    gradient_x ,gradient_y = gradient_f_x_y(points_abscisse_4[n] , points_ordonnée_4[n])
    norm_f = math.sqrt(gradient_x**2 + gradient_y**2)
    liste_4_f.append(f)
    liste_4_norm_f.append(norm_f)
```

```
#Tableau  récapitulatif de l`algorithme de descente  pour un pas fixe de 0.25
tableau_1 = pd.DataFrame({ 'F(x,y)': liste_1_f,'||Vf(x,y)||': liste_1_norm_f,'Sk': pas_1,'Xk': points_abscisse_1, 'Yk':
print('Tableau  récapitulatif de l`algorithme de descente  pour un pas fixe de 0.25 \n')
print(tableau_1)
#Tableau  récapitulatif de l`algorithme de descente  pour un pas fixe de 0.01
tableau_2 = pd.DataFrame({ 'F(x,y)': liste_2_f,'||Vf(x,y)||': liste_2_norm_f,'Sk': pas_2,'Xk': points_abscisse_2, 'Yk':
print('\nTableau  récapitulatif de l`algorithme de descente  pour un pas fixe de 0.01 \n')
print(tableau_2)
#Tableau  récapitulatif de l`algorithme  par la méthode de Newton local
tableau_4 = pd.DataFrame({ 'F(x,y)': liste_4_f,'||Vf(x,y)||': liste_4_norm_f,'Sk': 1,'Xk': points_abscisse_4,'Yk': poir
print('\nTableau  récapitulatif de l`algorithme  par la méthode de Newton locale\n')
print(tableau_4)
```