In [1]:

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

Θεωρούμε τη συνάρτηση $f(x) = \beta \sum_{i=1}^{n} x_i^2,\ \beta > 0$

Υλοποιουμε τη μέθοδο απότομης καθόδου (με σταθερό μέγεθος βήματος) και τη μέθοδο Newton για μέγιστο αριθμό επαναλήψεων 10^6 και κριτήριο τερματισμου $\|\nabla f(x)\| \leq 10^{-12}$

In [2]:

```
def f_c(x):
    # convex
    s=beta*np.linalg.norm(x)**2
    return s
```

In [3]:

```
def grad_f_c(x):

    #convex
    s=beta*2*x

    return s
```

In [4]:

```
def hessian_f_c(x):
    B=np.eye(len(x))
    B=2*beta*B
    return B
```

In [5]:

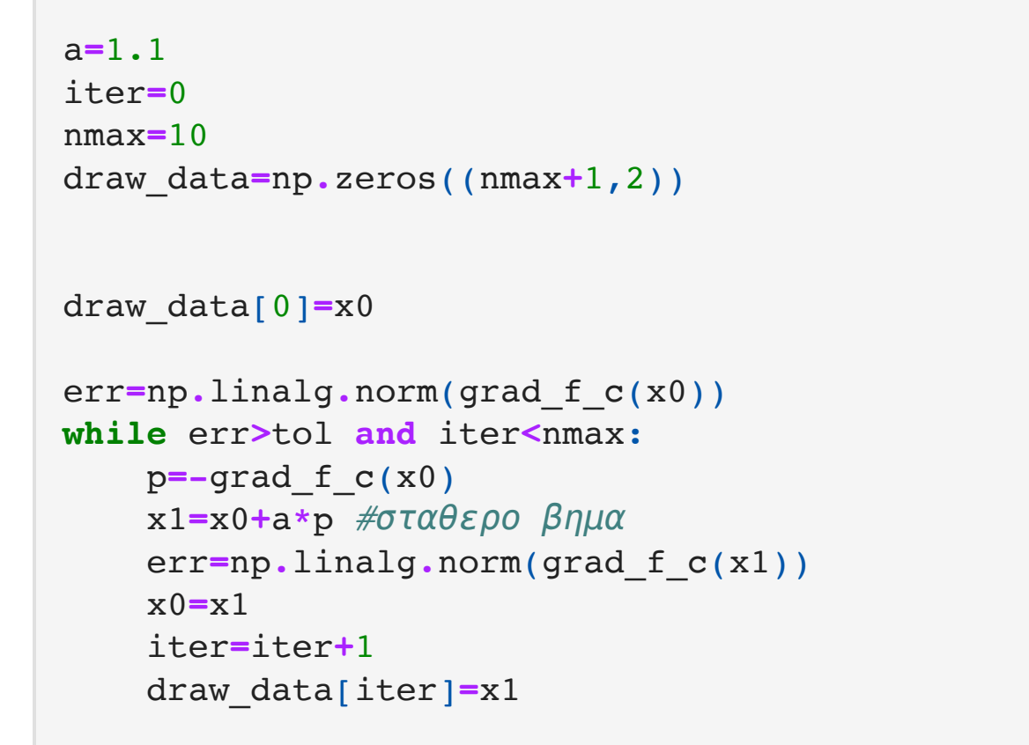
```
#Μεθοδος Νευτωνα
beta=1/200
x0=np.array([10,10])
tol=1.e-12
nmax=10**6

iter=0
err=np.linalg.norm(grad_f_c(x0))

while (err>tol) and iter<nmax:
    A=hessian_f_c(x0)
    b=-grad_f_c(x0)
###solve Ap=b
    p=np.linalg.solve(A,b)
    x1=x0+p
    print(x1)
    err=np.linalg.norm(grad_f_c(x1))
    print('grad=',err)
    x0=x1
    iter=iter+1
    print('-------')

print(iter)
print(x1)
```

```
[0. 0.]
grad= 0.0
-------
1
[0. 0.]
```

Δοκιμάζουμε με $x_0 = (10, 10)$ και $x_0 = (-5, 10)$ και συντελεστη βηματος $a = 0.1$

In [6]:

```
# Steep_Descent without line search
n_dim=2 # dimension
beta=1/2
#x0=10*np.ones(n_dim)
x0=np.array([-1,1])
tol=1.e-12

a=1.1
iter=0
nmax=10
draw_data=np.zeros((nmax+1,2))


draw_data[0]=x0

err=np.linalg.norm(grad_f_c(x0))
while err>tol and iter<nmax:
    p=-grad_f_c(x0)
    x1=x0+a*p #σταθερο βημα
    err=np.linalg.norm(grad_f_c(x1))
    x0=x1
    iter=iter+1
    draw_data[iter]=x1

plt.plot(draw_data[:,0],draw_data[:,1])
plt.show()

print('Προσεγγιση:',x1,'Σφάλμα:',err,'Επαναλήψεις:',iter)
```



```
Προσεγγιση: [-1.e-10  1.e-10] Σφάλμα: 1.4142135623731077e-10 Επαναλήψεις: 10
```

Δοκιμάζουμε τωρα για διάσταση $n = 10$ για $\beta = 1/2, 1/20, 1/200$, $x_0 = (10, \ldots, 10)$, $a = 0.1, 0.01$. Πόσες επαναλήψεις χρειάζονται κάθα φορά;

In [7]:

```
# Steep_Descent without line search
n_dim=10 # dimension


beta_data=[1/2,1/20,1/200] # παράμετρος συνάρτησης
tol=1.e-12
nmax=10**6

a_data=[0.1,0.01] # σταθερό βήμα στην καθοδο


for beta in beta_data:
    for a in a_data:
        x0=10*np.ones(n_dim)
        iter=0
        err=np.linalg.norm(grad_f_c(x0))
        while err>tol and iter<nmax:
            p=-grad_f_c(x0)
            x1=x0+a*p
            err=np.linalg.norm(grad_f_c(x1))
            x0=x1
            iter=iter+1
        print('beta:', beta,'a:',a)
        print('Προσεγγιση:',x1,'Σφάλμα:',err,'Επαναλήψεις:',iter)
        print('---------------')
```

```
beta: 0.5 a: 0.1
Προσεγγιση: [2.85616172e-13 2.85616172e-13 2.85616172e-13 2.85616172e-13
 2.85616172e-13 2.85616172e-13 2.85616172e-13 2.85616172e-13
 2.85616172e-13 2.85616172e-13] Σφάλμα: 9.031976398057448e-13 Επαναλήψεις: 296
---------------
beta: 0.5 a: 0.01
Προσεγγιση: [3.15978032e-13 3.15978032e-13 3.15978032e-13 3.15978032e-13
 3.15978032e-13 3.15978032e-13 3.15978032e-13 3.15978032e-13
 3.15978032e-13 3.15978032e-13] Σφάλμα: 9.992102725386636e-13 Επαναλήψεις: 3093
---------------
beta: 0.05 a: 0.1
Προσεγγιση: [3.15643847e-12 3.15643847e-12 3.15643847e-12 3.15643847e-12
 3.15643847e-12 3.15643847e-12 3.15643847e-12 3.15643847e-12
 3.15643847e-12 3.15643847e-12] Σφάλμα: 9.981534849079812e-13 Επαναλήψεις: 2864
---------------
beta: 0.05 a: 0.01
Προσεγγιση: [3.1620249e-12 3.1620249e-12 3.1620249e-12 3.1620249e-12 3.1620249e-12
 3.1620249e-12 3.1620249e-12 3.1620249e-12 3.1620249e-12 3.1620249e-12] Σφάλμα: 9.99920069089116e-13 Επαναλήψεις: 28768
---------------
beta: 0.005 a: 0.1
Προσεγγιση: [3.16065343e-11 3.16065343e-11 3.16065343e-11 3.16065343e-11
 3.16065343e-11 3.16065343e-11 3.16065343e-11 3.16065343e-11
 3.16065343e-11 3.16065343e-11] Σφάλμα: 9.994863724279642e-13 Επαναλήψεις: 26467
---------------
beta: 0.005 a: 0.01
Προσεγγιση: [3.16197585e-11 3.16197585e-11 3.16197585e-11 3.16197585e-11
 3.16197585e-11 3.16197585e-11 3.16197585e-11 3.16197585e-11
 3.16197585e-11 3.16197585e-11] Σφάλμα: 9.999045607178614e-13 Επαναλήψεις: 264785
---------------
```

Θεωρούμε τωρα το κριτήριο καθόδου Armijo

$$f(x_k + a_k p_k) < f(x_k) + \mu a_k p_k^T \nabla(f(x_k)), \quad \mu = 1/2, a_k \in \{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots, \}$$

Βρείτε τον αριθμό των επαναλήψεων για διάσταση $n = 10$ για $\beta = 1/2, 1/20, 1/200$, $x_0 = (10, \ldots, 10)$

In [8]:

```
# Line Search using armijo line search
def line_search(f,grad_f,x,p,c1):
# parameter for armijo condition - c1

    a=1
    l=f(x)+a*c1*np.dot(p,grad_f(x))
    cond=f(x+a*p)>l
    while cond:
        a=a/2
        l=f(x)+a*c1*np.dot(p,grad_f(x))
        cond=f(x+a*p)>l
    return a
```

In [9]:

```
# Steep_Descent with line_search
n_dim=10 # dimension


beta_data=[1/2,1/20,1/200] # παράμετρος συνάρτησης
c1_data=[0.5,0.1,0.01]
tol=1.e-12
nmax=10**6


for beta in beta_data:
        for c1 in c1_data:
            x0=10*np.ones(n_dim)
            iter=0
            err=np.linalg.norm(grad_f_c(x0))
            while err>tol and iter<nmax:
                p=-grad_f_c(x0)
                a=line_search(f_c,grad_f_c,x0,p,c1)
                x1=x0+a*p
                err=np.linalg.norm(grad_f_c(x1))
                x0=x1
                iter=iter+1
            print('beta:', beta, 'c1:',c1)
            print('Προσεγγιση:',x1,'Σφάλμα:',err,'Επαναλήψεις:',iter)
            print('---------------')
```

```
beta: 0.5 c1: 0.5
Προσεγγιση: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.] Σφάλμα: 0.0 Επαναλήψεις: 1
---------------
beta: 0.5 c1: 0.1
Προσεγγιση: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.] Σφάλμα: 0.0 Επαναλήψεις: 1
---------------
beta: 0.5 c1: 0.01
Προσεγγιση: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.] Σφάλμα: 0.0 Επαναλήψεις: 1
---------------
beta: 0.05 c1: 0.5
Προσεγγιση: [2.90033115e-12 2.90033115e-12 2.90033115e-12 2.90033115e-12
 2.90033115e-12 2.90033115e-12 2.90033115e-12 2.90033115e-12
 2.90033115e-12 2.90033115e-12] Σφάλμα: 9.171652389434054e-13 Επαναλήψεις: 274
---------------
beta: 0.05 c1: 0.1
Προσεγγιση: [2.90033115e-12 2.90033115e-12 2.90033115e-12 2.90033115e-12
 2.90033115e-12 2.90033115e-12 2.90033115e-12 2.90033115e-12
 2.90033115e-12 2.90033115e-12] Σφάλμα: 9.171652389434054e-13 Επαναλήψεις: 274
---------------
beta: 0.05 c1: 0.01
Προσεγγιση: [2.90033115e-12 2.90033115e-12 2.90033115e-12 2.90033115e-12
 2.90033115e-12 2.90033115e-12 2.90033115e-12 2.90033115e-12
 2.90033115e-12 2.90033115e-12] Σφάλμα: 9.171652389434054e-13 Επαναλήψεις: 274
---------------
beta: 0.005 c1: 0.5
Προσεγγιση: [3.15310015e-11 3.15310015e-11 3.15310015e-11 3.15310015e-11
 3.15310015e-11 3.15310015e-11 3.15310015e-11 3.15310015e-11
 3.15310015e-11 3.15310015e-11] Σφάλμα: 9.970978149600642e-13 Επαναλήψεις: 2635
---------------
beta: 0.005 c1: 0.1
Προσεγγιση: [3.15310015e-11 3.15310015e-11 3.15310015e-11 3.15310015e-11
 3.15310015e-11 3.15310015e-11 3.15310015e-11 3.15310015e-11
 3.15310015e-11 3.15310015e-11] Σφάλμα: 9.970978149600642e-13 Επαναλήψεις: 2635
---------------
beta: 0.005 c1: 0.01
Προσεγγιση: [3.15310015e-11 3.15310015e-11 3.15310015e-11 3.15310015e-11
 3.15310015e-11 3.15310015e-11 3.15310015e-11 3.15310015e-11
 3.15310015e-11 3.15310015e-11] Σφάλμα: 9.970978149600642e-13 Επαναλήψεις: 2635
---------------
```

Από τη βιβλιοθηκη scipy.optimize χρησιμοποιήστε τη μεθοδο line_search για να βρείτε το a χρησιμοποιωντας το κριτηριο Wolfe. Η παράμετρος c1 δηλώνει τη μεταβλητη του κριτηριου καθοδου Armijo και η c2 δηλώνει τη μεταβλητη του κριτηριου καμπυλότητας Wolfe.

Η μέθοδος επιστρεφει μια λιστα, με πρωτη συνιστωσα το ζητουμενο βημα a

In [10]:

```
# Steep_Descent with line_search (wolfe)
import scipy.optimize as opt
#x0=np.array([-5,10])

tol=1.e-12
iter=0
nmax=10**6

beta_data=[1/2,1/20,1/200] # παράμετρος συνάρτησης



for beta in beta_data:
    x0=10*np.ones(10)
    err=np.linalg.norm(grad_f_c(x0))
    while err>tol and iter<nmax:
        p=-grad_f_c(x0)
    ###### Wolfe c2
    ###### Armijo c1
    ###### default values c1,c2
        a=opt.line_search(f_c,grad_f_c,x0,p)
    #######

        x1=x0+a[0]*p
        err=np.linalg.norm(grad_f_c(x1))
        x0=x1
        iter=iter+1
    print('beta:', beta)
    print('Προσεγγιση:',x1,'Σφάλμα:',err,'Επαναλήψεις:',iter)
    print('---------------')



for beta in beta_data:
    x0=10*np.ones(10)
    err=np.linalg.norm(grad_f_c(x0))
    while err>tol and iter<nmax:
        p=-grad_f_c(x0)
    ###### Wolfe c2
    ###### Armijo c1
        a=opt.line_search(f_c,grad_f_c,x0,p,c1=0.01,c2=0.5)
    #######

        x1=x0+a[0]*p
        err=np.linalg.norm(grad_f_c(x1))
        x0=x1
        iter=iter+1
    print('beta:', beta,'c1:',0.01,'c2:',0.5)
    print('Προσεγγιση:',x1,'Σφάλμα:',err,'Επαναλήψεις:',iter)
    print('---------------')
```

```
beta: 0.5
Προσεγγιση: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.] Σφάλμα: 0.0 Επαναλήψεις: 1
---------------
beta: 0.05
Προσεγγιση: [3.13915912e-12 3.13915912e-12 3.13915912e-12 3.13915912e-12
 3.13915912e-12 3.13915912e-12 3.13915912e-12 3.13915912e-12
 3.13915912e-12 3.13915912e-12] Σφάλμα: 9.926892761668862e-13 Επαναλήψεις: 215
---------------
beta: 0.005
Προσεγγιση: [3.09350969e-11 3.09350969e-11 3.09350969e-11 3.09350969e-11
 3.09350969e-11 3.09350969e-11 3.09350969e-11 3.09350969e-11
 3.09350969e-11 3.09350969e-11] Σφάλμα: 9.78253659950125e-13 Επαναλήψεις: 367
---------------
beta: 0.5 c1: 0.01 c2: 0.5
Προσεγγιση: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.] Σφάλμα: 0.0 Επαναλήψεις: 368
---------------
beta: 0.05 c1: 0.01 c2: 0.5
Προσεγγιση: [2.62144e-12 2.62144e-12 2.62144e-12 2.62144e-12 2.62144e-12 2.62144e-12
 2.62144e-12 2.62144e-12 2.62144e-12 2.62144e-12] Σφάλμα: 8.289721149471757e-13 Επαναλήψεις: 386
---------------
beta: 0.005 c1: 0.01 c2: 0.5
Προσεγγιση: [2.9098126e-11 2.9098126e-11 2.9098126e-11 2.9098126e-11 2.9098126e-11
 2.9098126e-11 2.9098126e-11 2.9098126e-11 2.9098126e-11 2.9098126e-11] Σφάλμα: 9.20163537669305e-13 Επαναλήψεις: 412
---------------
```

Δοκιμάζουμε παρομοια για τη συνάρτηση

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

In [11]:

```
def f_r(x):
    #rosenbrock
    s=100*(x[1]-x[0]**2)**2+(1-x[0])**2
    return s
```

In [12]:

```
def grad_f_r(x):
     #gradient rosenbrock
    s1=-2*(1-x[0])-400*(x[1]-x[0]**2)*x[0]
    s2=200*(x[1]-x[0]**2)
    s=np.array([s1,s2])
    return s
```

In [13]:

```
# Steep_Descent with line_search
x0=np.array([-1,2])
tol=1.e-12
iter=0
nmax=10**6
c1=0.5
##########
err=np.linalg.norm(grad_f_r(x0))
while err>tol and iter<nmax:
    p=-grad_f_r(x0)
    a=line_search(f_r,grad_f_r,x0,p,c1)
    #print(a)
    x1=x0+a*p
    err=np.linalg.norm(grad_f_r(x1))
    x0=x1
    iter=iter+1
print('beta:', beta)
print('Προσεγγιση:',x1,'Σφάλμα:',err,'Επαναλήψεις:',iter)
print('---------------')
```

```
beta: 0.005
Προσεγγιση: [1. 1.] Σφάλμα: 9.811037539333113e-13 Επαναλήψεις: 1515
---------------
```

In [14]:

```
# Steep_Descent with line_search (wolfe)
import scipy.optimize as opt
x0=np.array([1,2])
#x0=np.array([-1,2])
#x0=np.array([-5,-2])
tol=1.e-12
iter=0
nmax=10**6

err=np.linalg.norm(grad_f_r(x0))
while err>tol and iter<nmax:
    p=-grad_f_r(x0)
    ###### Wolfe c2
    ###### Armijo c1
    ####### default values for c1,c2
    a=opt.line_search(f_r,grad_f_r,x0,p,c1=0.01,c2=0.5)
    #######
    #print(a)
    x1=x0+a[0]*p
    err=np.linalg.norm(grad_f_r(x1))
    x0=x1
    iter=iter+1
print('Προσεγγιση:',x1,'Σφάλμα:',err,'Επαναλήψεις:',iter)
print('---------------')
```

```
Προσεγγιση: [1. 1.] Σφάλμα: 9.321287586935148e-13 Επαναλήψεις: 16368
---------------
```

In [ ]: