Md Shariar Imroze Khan #220202354 ¶

First Approach

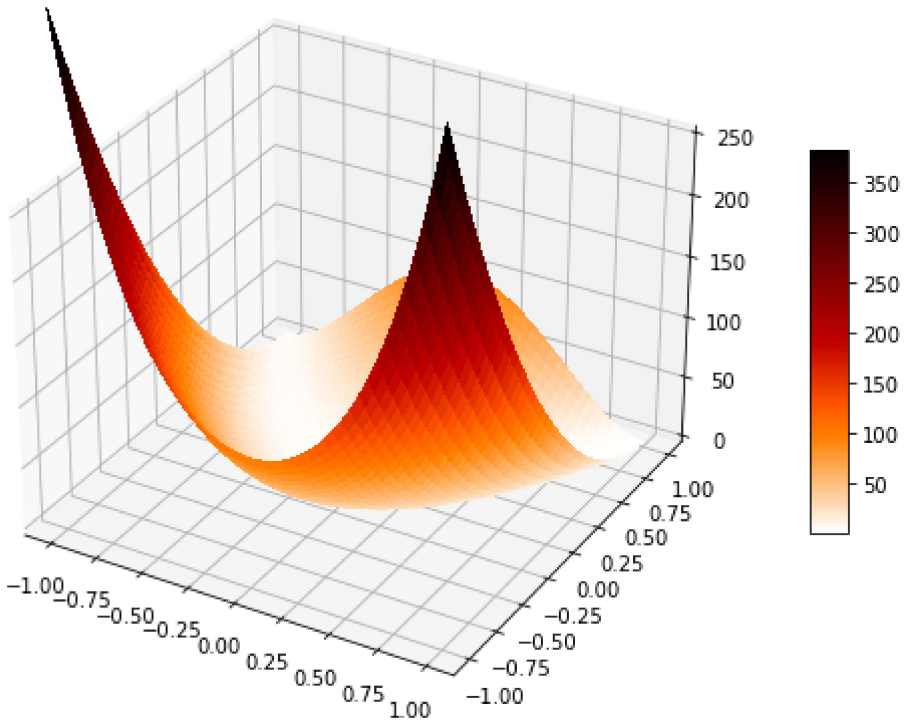
In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

from sympy import symbols,diff,Symbol
from sympy.utilities.lambdify import lambdify
from sympy.solvers import solve
from sympy import symbols, Matrix, Function, simplify, exp, hessian, solve, init_printi

from matplotlib.ticker import LinearLocator, FormatStrFormatter
```

In [2]:

```
#Plotting the rosenbrock function
ros_func = lambda x1,x2: (x1-1)**2 + 100*(x2-x1**2)**2;
# Initialize figure
figRos = plt.figure(figsize=(14, 7))
axRos = figRos.gca(projection='3d')

# Evaluate function
X = np.arange(-1, 1, 0.015)
Y = np.arange(-1, 1, 0.015)
X, Y = np.meshgrid(X, Y)
Z = ros_func(X,Y)

# Plot the surface
surf = axRos.plot_surface(X, Y, Z, cmap=cm.gist_heat_r,linewidth=0, antialiased=False)
axRos.set_zlim(0, 250)
figRos.colorbar(surf, shrink=0.5, aspect=10)
plt.show()
```

```
C:\Users\shari\AppData\Local\Temp/ipykernel_9004/3788292813.py:5: MatplotlibDeprecationWarning: Calling gca() with keyword arguments was deprecated in Matplotlib 3.4. Starting two minor releases later, gca() will take no keyword arguments. The gca() function should only be used to get the current axes, or if no axes exist, create new axes with default keyword arguments. To create a new axes with non-default arguments, use plt.axes() or plt.subplot().
  axRos = figRos.gca(projection='3d')
```



In [4]:

```
x1,x2 = symbols('x1 x2',real=True)
ros_func = (x1-1)**2 + 100*(x2-x1**2)**2  #0.5*x1**2-4*x1+5*x2**2+30*x2
x0 = np.array([-1,1])
```

In [5]:

```
def grad(x0, ros_func):
    dx1 = diff(ros_func,x1)
    dx2 = diff(ros_func,x2)
    #return np.array([lambdify(x1,dx1)(x0[0]), lambdify(x2,dx2)(x0[1])])
    grad_calc=lambdify([x1,x2],[dx1,dx2])
    return grad_calc(x0[0],x0[1])
```

In [6]:

```
diff(ros_func,x1)
```

Out[6]:

$$-400x_1\left(-x_1^2+x_2\right)+2x_1-2$$

In [7]:

```
diff(ros_func,x2)
```

Out[7]:

$$-200x_1^2+200x_2$$

In [8]:

```
grad(x0,ros_func)
```

Out[8]:

```
[-4, 0]
```

In [9]:

```
#local minimum: for a multivariate function ros_func, the necessary condition for a poi
#to be at the local minimum is that the gradient is 0 and the hessian is positive semid
```

In [10]:

```
prog = [float((x0[0]-1)**2 + 100*(x0[1]-x0[0]**2)**2)]
print(prog)

while np.sqrt(float(grad(x0,ros_func)[0]**2) + float(grad(x0,ros_func)[1]**2)) > 0.0000
    s = -1*np.array(grad(x0,ros_func)) #descent in the direction of gradiant
    a = Symbol('a')
    l = np.array(grad(x0+a*s,ros_func)).dot(s)
    a = solve(l,a)
    if len(a) == 0: #required to run python code
        a = np.array([0,0])

    print(l,x0, a,s)

    #x0 = x0 + a * s
    prog.append(float((x0[0]-1)**2 + 100*(x0[1]-x0[0]**2)**2))
    print(float((x0[0]-1)**2 + 100*(x0[1]-x0[0]**2)**2))
print(f'x* = [{float(x0[0]),float(x0[1])}]')
```

```
t(2)/80, 7*sqrt(2)/80 + 1/8] [4 0]
4.0
32*a + 4*(1 - (4*a - 1)**2)*(400 - 1600*a) - 16 [-1  1] [1/2, 1/8 - 7*sqr
t(2)/80, 7*sqrt(2)/80 + 1/8] [4 0]
4.0
32*a + 4*(1 - (4*a - 1)**2)*(400 - 1600*a) - 16 [-1  1] [1/2, 1/8 - 7*sqr
t(2)/80, 7*sqrt(2)/80 + 1/8] [4 0]
4.0
32*a + 4*(1 - (4*a - 1)**2)*(400 - 1600*a) - 16 [-1  1] [1/2, 1/8 - 7*sqr
t(2)/80, 7*sqrt(2)/80 + 1/8] [4 0]
4.0
32*a + 4*(1 - (4*a - 1)**2)*(400 - 1600*a) - 16 [-1  1] [1/2, 1/8 - 7*sqr
t(2)/80, 7*sqrt(2)/80 + 1/8] [4 0]
4.0
32*a + 4*(1 - (4*a - 1)**2)*(400 - 1600*a) - 16 [-1  1] [1/2, 1/8 - 7*sqr
t(2)/80, 7*sqrt(2)/80 + 1/8] [4 0]
4.0
32*a + 4*(1 - (4*a - 1)**2)*(400 - 1600*a) - 16 [-1  1] [1/2, 1/8 - 7*sqr
t(2)/80, 7*sqrt(2)/80 + 1/8] [4 0]
4.0
```

In [11]:

```
fig, ax = plt.subplots(figsize=(12,6)
ax.plot(prog)
ax.set_title('Ros_func Evaluated at x0 for each Iteration')
ax.set_xlabel('Number of Iteration')
ax.set_ylabel('ros_func value')
plt.show()
```

```
  File "C:\Users\shari\AppData\Local\Temp/ipykernel_9004/633785862.py", line
2
    ax.plot(prog)
    ^
SyntaxError: invalid syntax
```

Second Approach

In [12]:

```
1
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from matplotlib import cm
5 from matplotlib.ticker import LinearLocator, FormatStrFormatter
6 from mpl_toolkits.mplot3d import Axes3D
```

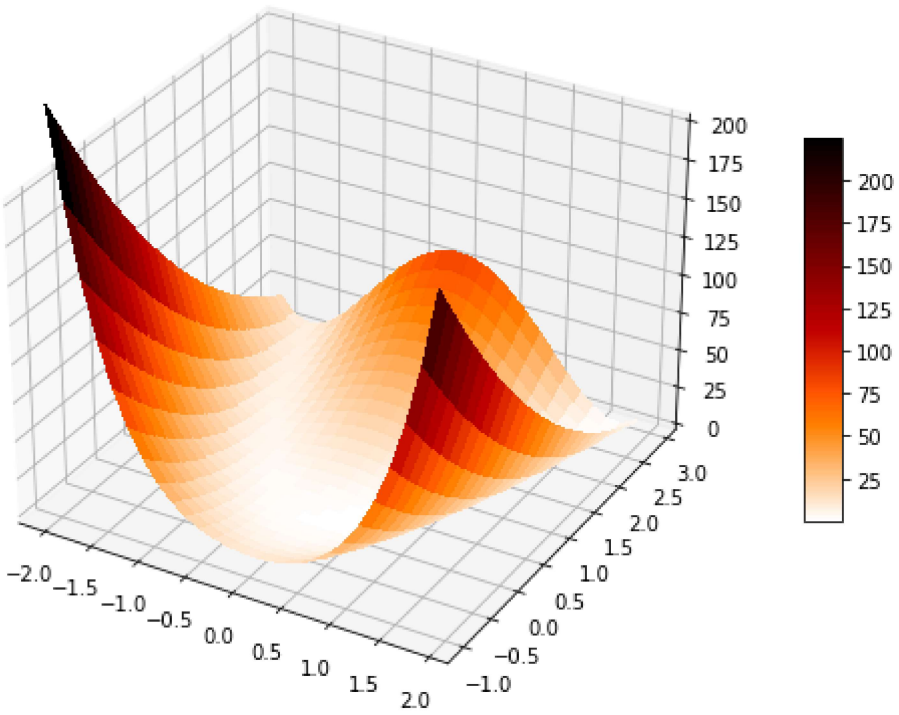
In [13]:

```
1 b=10
2 f = lambda x,y: (x-1)**2 + b*(y-x**2)**2;
```

In [14]:

```
# Initialize figure
figRos = plt.figure(figsize=(14, 7))
axRos = figRos.gca(projection='3d')

# Evaluate function
X = np.arange(-2,2, 0.15)
Y = np.arange(-1, 3, 0.15)
X, Y = np.meshgrid(X, Y)
Z = f(X,Y)

# Plot the surface
surf = axRos.plot_surface(X, Y, Z, cmap=cm.gist_heat_r,linewidth=0, antialiased=False)
axRos.set_zlim(0, 200)
figRos.colorbar(surf, shrink=0.5, aspect=10)
plt.show()
```

```
C:\Users\shari\AppData\Local\Temp/ipykernel_9004/1255043438.py:3: MatplotlibDeprecationWarning: Calling gca() with keyword arguments was deprecated in Matplotlib 3.4. Starting two minor releases later, gca() will take no keyword arguments. The gca() function should only be used to get the current axes, or if no axes exist, create new axes with default keyword arguments. To create a new axes with non-default arguments, use plt.axes() or plt.subplot().
  axRos = figRos.gca(projection='3d')
```



In [15]:

```
df = lambda x,y: np.array([2*(x-1) - 4*b*(y - x**2)*x,
                           2*b*(y-x**2)])
```

In [16]:

```
F = lambda X: f(X[0],X[1])
dF = lambda X: df(X[0],X[1])
```

In [17]:

```
x0 = np.array([-1.5,1])
print(F(x0))
print(dF(x0))
```

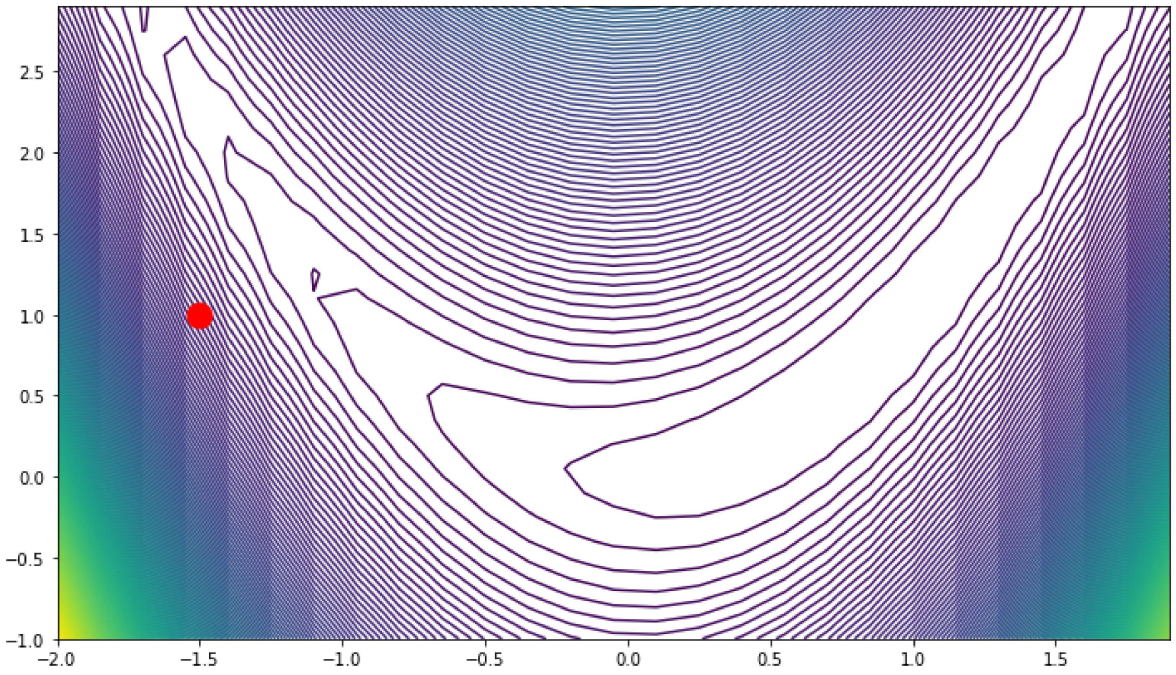
```
21.875
[-80. -25.]
```

In [18]:

```
# Initialize figure
plt.figure(figsize=(12, 7))
plt.contour(X,Y,Z,200)
plt.plot([x0[0]],[x0[1]],marker='o',markersize=15, color ='r')
```

Out[18]:

```
[<matplotlib.lines.Line2D at 0x2a93a4b0640>]
```



In [19]:

```
fx = F(x0);
gx = dF(x0);
s = -gx;
print(s)
```

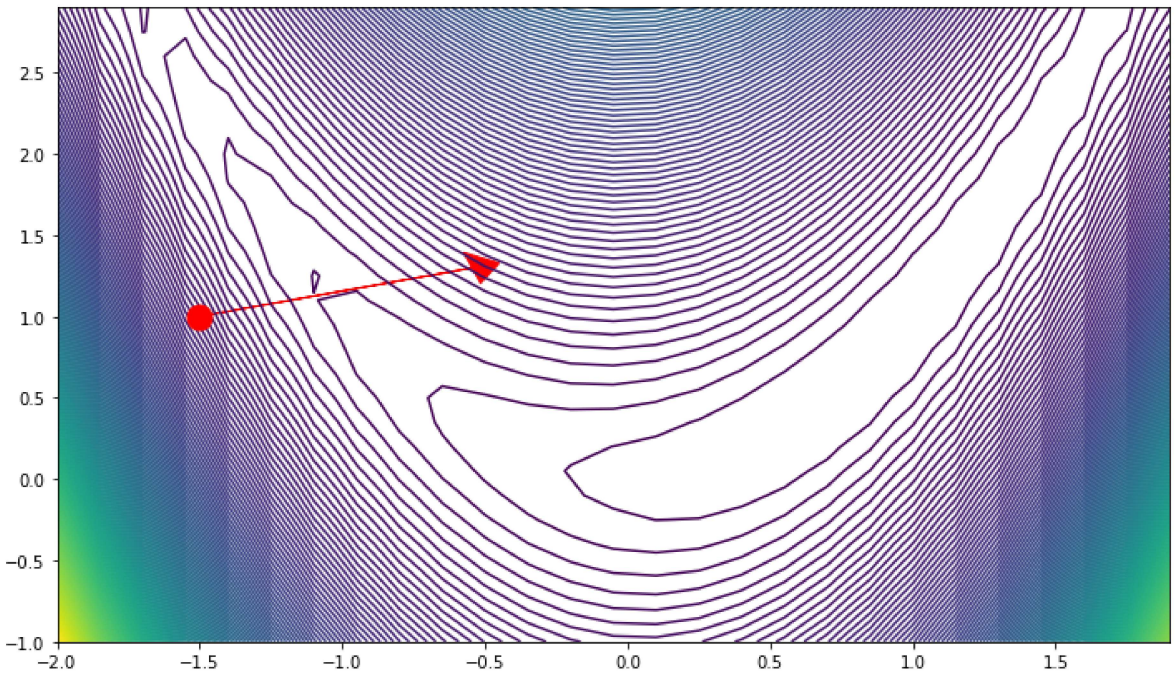
```
[80. 25.]
```

In [20]:

```
# Initialize figure
plt.figure(figsize=(12, 7))
plt.contour(X,Y,Z,200)
ns = np.sqrt(s[0]**2+s[1]**2);
plt.plot([x0[0]],[x0[1]],marker='o',markersize=15, color ='r')
plt.arrow(x0[0],x0[1],s[0]/ns,s[1]/ns, head_width=0.2, head_length=0.1, fc='r', ec='r')
```

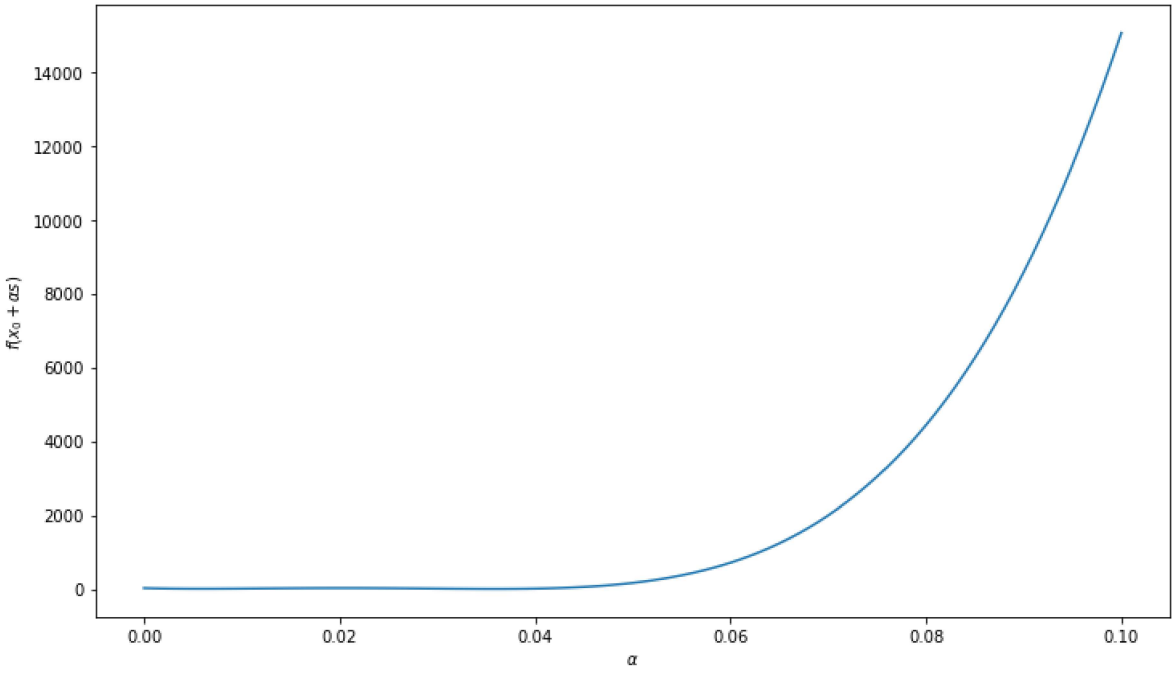
Out[20]:

```
<matplotlib.patches.FancyArrow at 0x2a93a299f40>
```



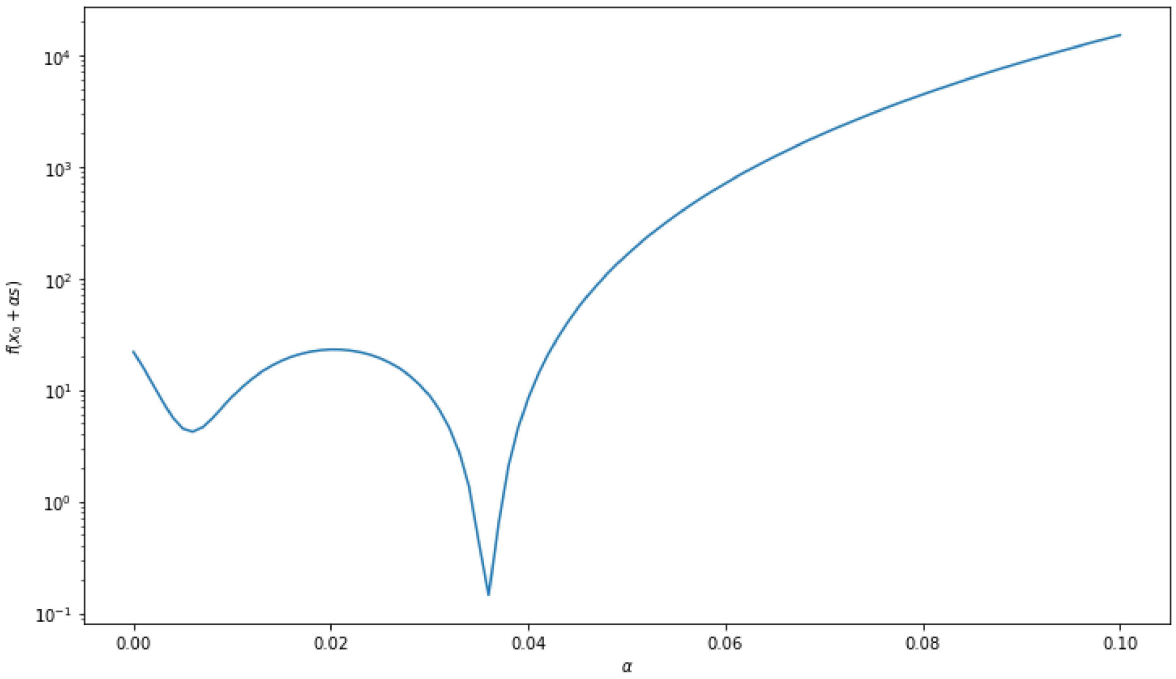
In [21]:

```
al = np.linspace(0,0.1,101)
z = [F(x0+a*s) for a in al]
figLS = plt.figure(figsize=(12, 7))
plt.plot(al,z)
plt.ylabel('$f(x_0+ \\alpha s)$')
plt.xlabel('$\\alpha$')
plt.show()
```



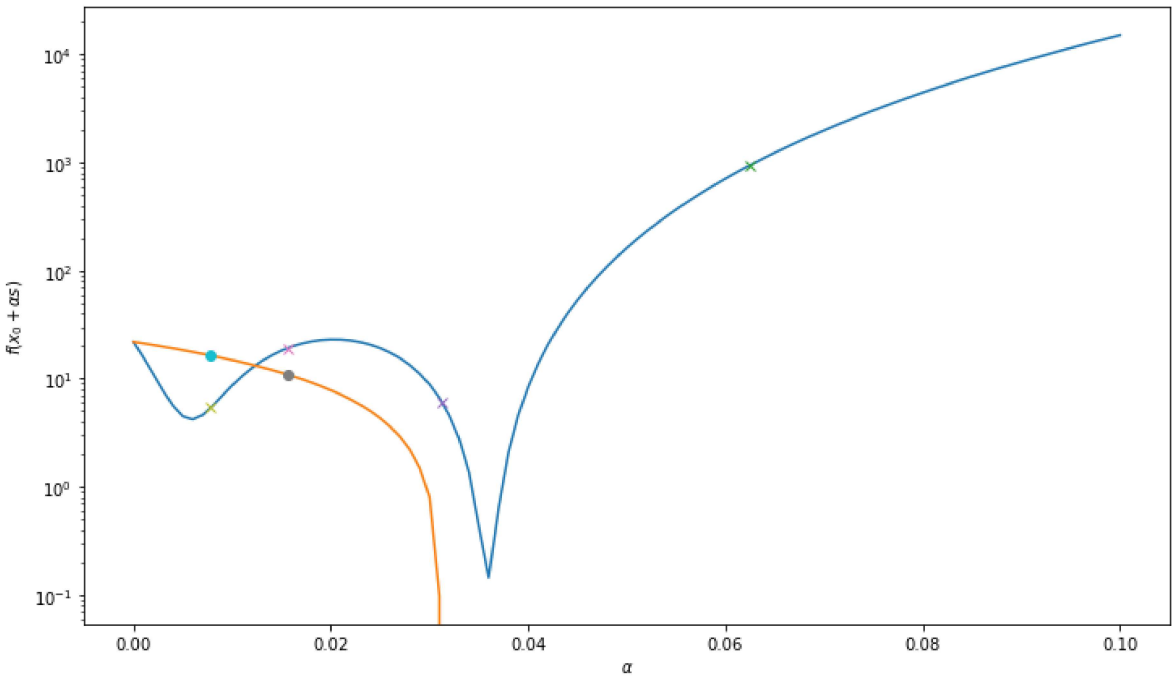
In [22]:

```
figLS = plt.figure(figsize=(12, 7))
plt.plot(al,z)
plt.yscale('log')
plt.ylabel('$f(x_0+ \\alpha s)$')
plt.xlabel('$\\alpha$')
plt.show()
```



In [23]:

```
theta = 0.1
alpha = 1
tol = 1e-10
d = theta*np.dot(gx,s)

figLS1 = plt.figure(figsize=(12, 7))
plt.plot(al,z)
plt.plot(al,[fx+a*d for a in al])

for i in range(10):

    if (alpha<=0.1):
        plt.plot(alpha,F(x0+alpha*s),marker='x');
        plt.plot(alpha,fx + alpha*d,marker='o')

    if F(x0+alpha*s) < (fx + alpha*d):
        break;
    alpha = alpha/2;

plt.yscale('log')
plt.ylabel('$f(x_0+ \\alpha s)$')
plt.xlabel('$\\alpha$')
plt.show()
```



In [24]:

```
[fx,fx+0.01*d]
```

Out[24]:

```
[21.875, 14.85]
```