In [ ]:

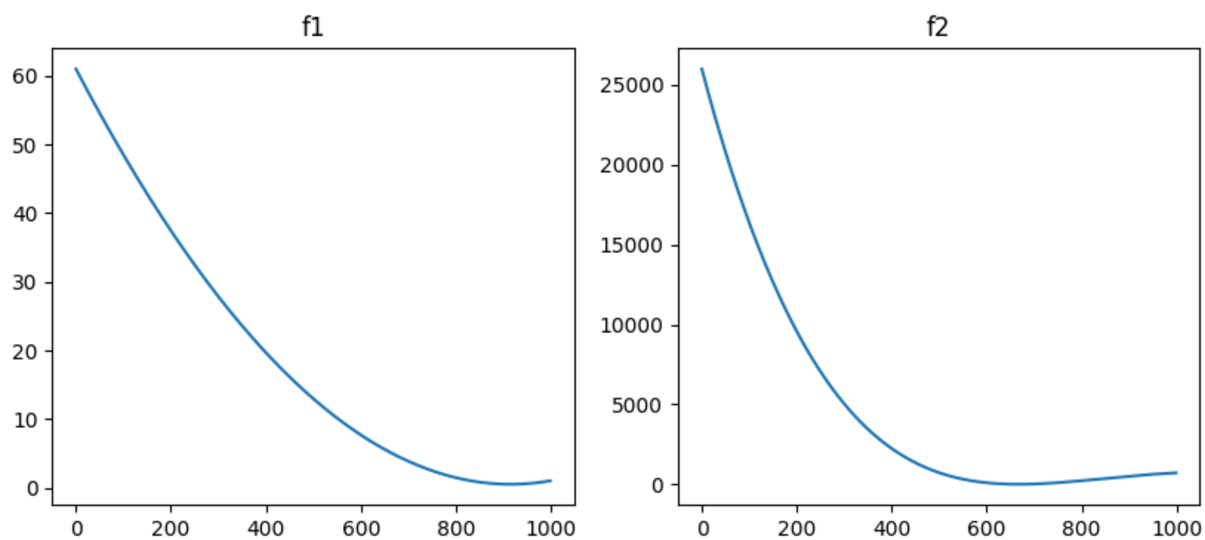
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sb
```

In [ ]:

```
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(10,4))
x = np.linspace(-3,3,1000)
y = np.linspace(-3,3,1000)

def fx1(x, y):
    return (x-2)**2+(y-3)**2
def fx2(x, y):
    return (1-(y-3))**2+20*((x+3)-(y-3)**2)**2
ax1.plot(fx1(x,y))
ax1.set_title('f1')
ax2.plot(fx2(x,y))
ax2.set_title('f2')

plt.show()
```



In [ ]:

```
# specify initial values
x1_c = 0
x2_c = 0
y1_c = 0
y2_c = 0
learning_rate = 0.5
T = 100
MIN_VAL = 99999

def dfx1(x,y):
    return 2*x-4
def dfy1(x,y):
    return 2*y-6
def dfx2(x,y):
    return 40*(x-(y-3)**2+3)
def dfy2(x,y):
```

```python
    return (-80)*(-1*(y-3)**2+x+3)*(y-3)-2*(4-y)

# this gives you the result of the derivative (the gradient)
def calc_gradient(fx, x=None, y=None):
    if x is not None and y is not None:
        return fx(x, y)
    elif x is not None:
        return fx(x)
    elif y is not None:
        return fx(y)
    else:
        return None

# with gradient calculated, use
def gradient_descent(val, lr, grad):
    # return x+1
    return val - (lr * grad)

# run T amount of iterations
def run_iter(fx, dfx, dfy, xi, yi, T):
    x_c,y_c = xi, yi
    MIN_VAL = 99999.9
    for iter in range(T):
        gradx = calc_gradient(dfx, x_c, y_c)
        grady = calc_gradient(dfy, x_c, y_c)
        x_c = gradient_descent(x_c, learning_rate, gradx)
        y_c = gradient_descent(y_c, learning_rate, grady)
        print(x_c,y_c)
        if min(MIN_VAL,fx(x_c,y_c)) is fx(x_c,y_c):
            MIN_VAL = fx(x_c,y_c)
    return (x_c, y_c, MIN_VAL)
```

In [ ]:

```python
# Test different inputs
x1, y1, min1 = run_iter(fx1, dfx1, dfy1, x1_c, y1_c,T)
x2, y2, min2 = run_iter(fx2, dfx2, dfy2, x2_c, y2_c, T)

print("Results for f1: x: " + x1 + "; y: " + y1 + "; optimal minimum: " + min1)
print("Results for f2: x: " + x2 + "; y: " + y2 + "; optimal minimum: " + min2)
```

```
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
```

```
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
120.0 724.0
10394480.0 -14988667116.0
4.493202840083638e+21 1.3469424333631776e+32
3.6285078375886363e+65 -9.774782352477998e+97
```

```
---------------------------------------------------------------------------
OverflowError                             Traceback (most recent call last)
Cell In[57], line 3
      1 # Test different inputs
      2 x1, y1, min1 = run_iter(fx1, dfx1, dfy1, x1_c, y1_c,T)
----> 3 x2, y2, min2 = run_iter(fx2, dfx2, dfy2, x2_c, y2_c, T)
      5 print("Results for f1: x: " + x1 + "; y: " + y1 + "; optimal minimum: " + mi
n1)
      6 print("Results for f2: x: " + x2 + "; y: " + y2 + "; optimal minimum: " + mi
n2)

Cell In[56], line 45, in run_iter(fx, dfx, dfy, xi, yi, T)
     43     y_c = gradient_descent(y_c, learning_rate, grady)
     44     print(x_c,y_c)
---> 45     if min(MIN_VAL,fx(x_c,y_c)) is fx(x_c,y_c):
     46         MIN_VAL = fx(x_c,y_c)
     47 return (x_c, y_c, MIN_VAL)

Cell In[51], line 8, in fx2(x, y)
      7 def fx2(x, y):
----> 8     return (1-(y-3))**2+20*((x+3)-(y-3)**2)**2

OverflowError: (34, 'Result too large')
```

In [ ]: