In [ ]:
```python
import csv
import codecs
import urllib.request
from collections import Counter
import glob
import codecs
import re
import pandas as pd
import math
from cmath import exp
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
from sklearn import datasets
import statsmodels.api as sm
from pylab import rcParams
from numpy.linalg import inv
from sympy import *
```

2 Gradient Descent and Step Length Controller: In this part, you are required to optimize the booth function using gradient descent. The Figure below provides a visual representation of the booth function in a 3D plot.

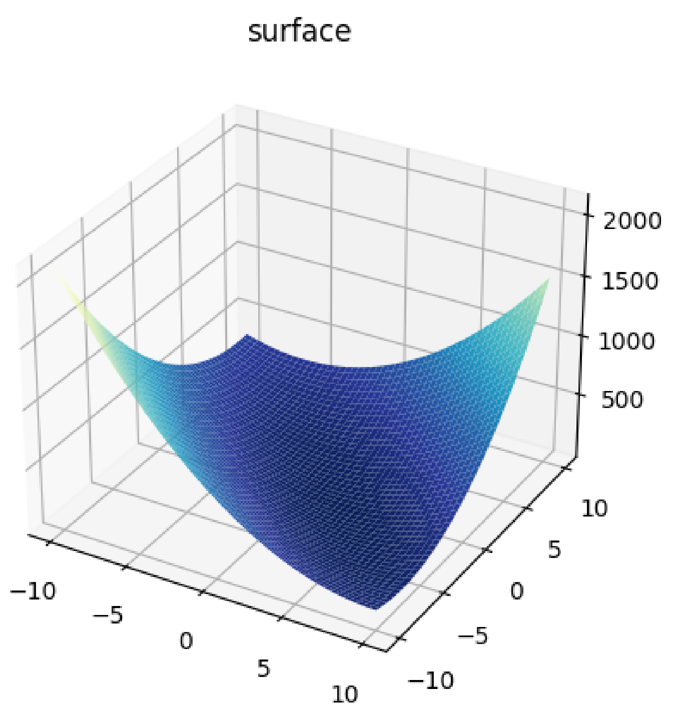
1

In [ ]:
```python
def f(x, y):
    return (x+2*y)**2+(2*x+y-5)**2

x = np.linspace(-10, 10, 50)
y = np.linspace(-10, 10, 50)

X, Y = np.meshgrid(x, y)
Z = f(X, Y)
```

In [ ]:
```python
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
                cmap=plt.cm.YlGnBu_r, edgecolor='none')
ax.set_title('surface');
# ax.view_init(10,20)
```



2

In [ ]:
```
# create a "symbol" called x
x, y= symbols('x y', real=True)

#Define function
f = (x+2*y)**2+(2*x+y-5)**2

#Calculating Derivative
dx_f = f.diff(x)
dy_f = f.diff(y)
```

In [ ]:
```
dx_f
```

Out[ ]: $10x + 8y - 20$

In [ ]:
```
dy_f
```

Out[ ]: $8x + 10y - 10$

3

In [ ]:
```
x = np.arange(-4, 4)
y = np.arange(-4, 4)
```

In [ ]:
```
f = (x+2*y)**2+(2*x+y-5)**2
```

In [ ]:
```
f
```

Out[ ]:
```
array([433, 277, 157,  73,  25,  13,  37,  97])
```

In [ ]:
```
# m denotes the number of examples here, not the number of features
def gradientDescent(x, y, theta, alpha, m, numIterations):
    costList = []
    for i in range(0, numIterations):
        y_pred = np.dot(x, theta)
        error = y_pred - y
        cost = np.sum(error ** 2) / (2 * m)
        costList.append(cost)
        gradient = np.dot(x.T, error) / m
        theta = theta - alpha * gradient
    return theta, costList
```

In [ ]:
```
n = np.max(x.shape)
x = np.vstack([np.ones(n), x, y]).T
m, n = np.shape(x)
numIterations= 1000000
alpha = 0.0001
theta = np.random.rand(n)
```

In [ ]:
```
theta, _ = gradientDescent(x, f, theta, alpha, m, numIterations)
print('Theta is: ', theta)
```

```
Theta is:  [115.         -23.93354681 -24.06645319]
```

5

In [ ]:
```
def backTrackingGradientDescent(x, y, theta, alpha, m, numIterations, beta):
    costList=[]
    t=1
    for i in range(0, numIterations):
        y_pred = np.dot(x, theta)
        error = y_pred - y
        cost = np.sum(error ** 2) / (2 * m)
        costList.append(cost)
        gradient = np.dot(x.T, error) / m
        theta = theta - t*alpha * gradient
```

```python
        t=beta*t
    return theta, t, costList
```

In [ ]:
```python
theta, t, _ = backTrackingGradientDescent(x, f, theta, alpha, m, 1000, beta=0.5)
print('Theta: \n', theta, '\n t is: \n', t)
```

```
Theta: 
 [115.         -23.93354681 -24.06645319]
 t is: 
 9.332636185032189e-302
```

In [ ]: