Lab 3 : Convex Optimisation

Gradient Descent

Write the code following the instructions to obtain the desired results

Import all the required libraries

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Find the value of x at which $f(x)$ is minimum :

1. Find x analytically
2. Write the update equation of gradient descent
3. Find x using gradient descent method

Example 1 : $f(x) = x^2 + x + 2$

Analytical :

$$\frac{d}{dx}f(x) = 2x + 1 = 0$$

$$\frac{d^2}{dx^2}f(x) = 2 \ (Minima)$$

$$x = -\frac{1}{2} \ (analytical \ solution)$$

Gradient Descent Update equation :

$$x_{init} = 4$$

$$x_{updt} = x_{old} - \lambda(\frac{d}{dx}f(x)|x = x_{old})$$

$$x_{updt} = x_{old} - \lambda(2x_{old} + 1)$$

Gradient Descent Method :

Follow the below steps and write your code in the block below

1. Generate x, 1000 data points from -10 to 10
2. Generate and Plot the function $f(x) = x^2 + x + 2$
3. Initialize the starting point (x_{init}) and learning rate (λ)
4. Use Gradient descent algorithm to compute value of x at which the function $f(x)$ is minimum
5. Also vary the learning rate and initialisation point and plot your observations

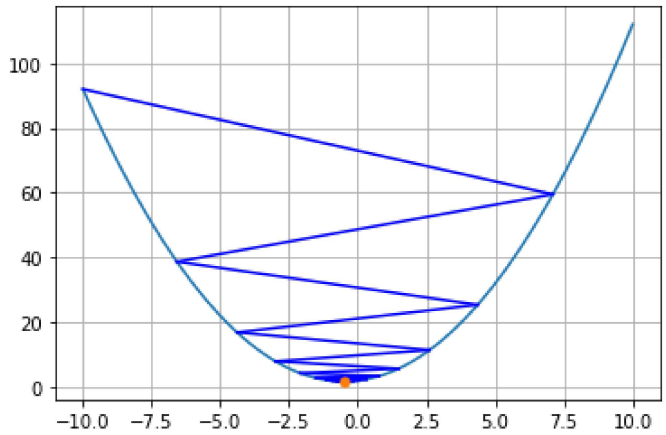
Derivative of f w.r.t. x : (Analytical)

$$\frac{d}{dx}f(x) = 2x + 1$$

Note that we won't make use of $\frac{d^2}{dx^2}f(x)$ in the gradient descent process

In [ ]:
```
## Write your code here

# generate 1000 points
x = np.linspace(-10, 10, 1000)
# generation of function
def f(x):
    y = x**2 + x + 2
    return y
# plotting of function
fig = plt.figure()
ax = plt.axes()
plt.grid()
plt.plot(x, f(x))

# initializatin of x_init, learning rate lambda (lam), num of iterations (n_i), min cost change t
x_init = -10
lam = 0.9
n_i = 10000
min_dc = 0.0000001

# computation of slope + gradient descent
def slope(x):
    return 2*x + 1

def gradient_descent(x1, lam):
    x2 = x1 - (lam*slope(x1))
    return x2

# initial point setting; then iterations of gradient descent (GD)
x_curr = x_init
x_prev = x_init
for i in range(n_i):
    x_curr = gradient_descent(x_curr, lam) # do GD step

    x_vals = [x_prev, x_curr]              # plotting of line
    y_vals = [f(x_prev), f(x_curr)]
    plt.plot(x_vals, y_vals, color = 'blue')

    x_diff = np.abs(x_curr - x_prev)       # record difference in values after GD
    if(x_diff < min_dc):
        break

    x_prev = x_curr                        # update x_prev for next iteration

# print value of x at which f(x) is minimum
print("Value of x for which value of f(x) is minimum is: x = ", x_curr)
plt.plot(x_curr, f(x_curr), marker="o", markersize=5)
```

Out[ ]:
```
Value of x for which value of f(x) is minimum is: x =  -0.500000044000994
[<matplotlib.lines.Line2D at 0x149ca5f95e0>]
```



Example 2 : $f(x) = xsinx$

Analytical : Find solution analytically

Gradient Descent Update equation : Write Gradient descent update equations

Gradient Descent Method :

Follow the below steps and write your code in the block below

1. Generate x, 1000 data points from -10 to 10
2. Generate and Plot the function $f(x) = xsinx$
3. Initialize the starting point (x_{init}) and learning rate (λ)
4. Use Gradient descent algorithm to compute value of x at which the function $f(x)$ is minimum
5. Also vary the learning rate and initialisation point and plot your observations

Derivative of f w.r.t. x : (Analytical)

$$\frac{d}{dx}f(x) = xcosx + sinx$$

Note that we won't make use of $\frac{d^2}{dx^2}f(x)$ in the gradient descent process

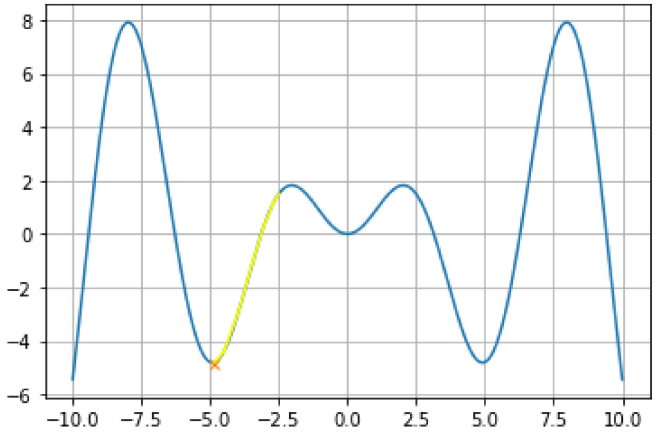
In [ ]:
```
## Write your code here

# generate 1000 points
x = np.linspace(-10, 10, 1000)
# generation of function
def f(x):
    y = x*np.sin(x)
    return y
# plotting of function
fig = plt.figure()
ax = plt.axes()
plt.grid()
plt.plot(x, f(x))

# initializatin of x_init, learning rate lambda (lam), num of iterations (n_i), min cost change t
x_init = -2.5
lam = 0.15
n_i = 10000
min_dc = 0.0000001

# computation of slope + gradient descent
def slope(x):
    return (x*np.cos(x) + np.sin(x))
```

```
def gradient_descent(x1, lam):
    x2 = x1 - (lam*slope(x1))
    return x2

# initial point setting; then iterations of gradient descent (GD)
x_curr = x_init
x_prev = x_init
for i in range(n_i):
    x_curr = gradient_descent(x_curr, lam) # do GD step

    x_vals = [x_prev, x_curr]              # plotting of line
    y_vals = [f(x_prev), f(x_curr)]
    plt.plot(x_vals, y_vals, color = 'yellow')

    x_diff = np.abs(x_curr - x_prev)       # record difference in values after GD
    if(x_diff < min_dc):
        break

    x_prev = x_curr                        # update x_prev for next iteration

# print value of x at which f(x) is minimum
print("Value of x for which value of f(x) is minimum is: x = ", x_curr)
plt.plot(x_curr, f(x_curr), marker="x", markersize=5)
```

```
Value of x for which value of f(x) is minimum is: x =  -4.913180431944305
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x149cbd80730>]
```



Find the value of x and y at which $f(x, y)$ is minimum :

Example 1 : $f(x, y) = x^2 + y^2 + 2x + 2y$

Gradient Descent Method :

Follow the below steps and write your code in the block below

1. Generate x and y, 1000 data points from -10 to 10
2. Generate and Plot the function $f(x, y) = x^2 + y^2 + 2x + 2y$
3. Initialize the starting point (x_{init},y_{init}) and learning rate (λ)
4. Use Gradient descent algorithm to compute value of x and y at which the function $f(x, y)$ is minimum
5. Also vary the learning rate and initialisation point and plot your observations

Derivatives of f w.r.t. x, y : (Analytical)

$$\frac{\partial}{\partial x} f(x,y) = 2x + 2$$

$$\frac{\partial}{\partial y} f(x,y) = 2y + 2$$

Note that we won't make use of $\frac{\partial^2}{\partial x^2} f(x,y)$, $\frac{\partial^2}{\partial y^2} f(x,y)$, $\frac{\partial^2}{\partial xy} f(x,y)$ in the gradient descent process

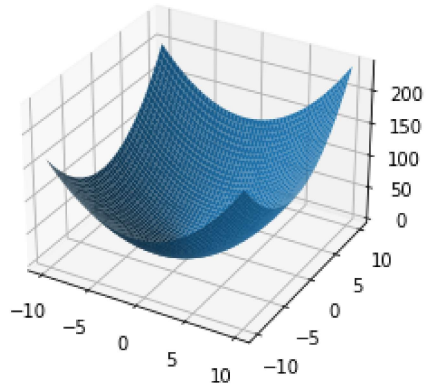
In [ ]:
```
## Write your code here (Ignore the warning)
## Write your code here

# generate 1000 points
x = np.linspace(-10, 10, 1000)
y = np.linspace(-10, 10, 1000)
x, y = np.meshgrid(x, y)
# generation of function
def f(x, y):
    z = x**2 + y**2 + 2*x + 2*y
    return z
# plotting of function
fig = plt.figure()
ax = plt.axes(projection='3d')
# ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(x, y, f(x,y))

# plotting of contours
plt.figure()
ax2 = plt.axes(projection='3d')
ax2.contour3D(x, y, f(x, y), 50)
ax2.set_xlabel('x')
ax2.set_ylabel('y')
ax2.set_zlabel('z')

plt.figure()
ax3 = plt.axes()
ax3.contour(x, y, f(x,y), 100)

# initializatin of x_init, y_init, learning rate lambda (lam), num of iterations (n_i), min cost
x_init = -7.5
y_init = -7.5
lam = 0.9
n_i = 10000
min_dc = 0.0000001

# computation of slope + gradient descent
def slope_x(x):
    return (2*x + 2)
def slope_y(y):
    return (2*y + 2)

def gradient_descent(x1, y1, lam):
    x2 = x1 - (lam*slope_x(x1))
    y2 = y1 - (lam*slope_y(y1))
    return x2, y2

# initial point setting; then iterations of gradient descent (GD)
x_curr = x_init
x_prev = x_init
y_curr = y_init
y_prev = y_init
for i in range(n_i):
    x_curr, y_curr = gradient_descent(x_curr, y_curr, lam) # do GD step

    x_vals = [x_prev, x_curr]              # plotting of line
    y_vals = [y_prev, y_curr]
```

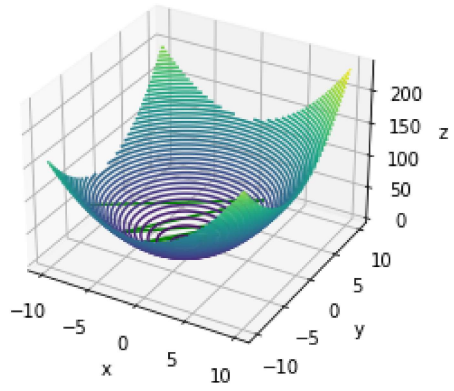
```
    z_vals = [f(x_prev, y_prev), f(x_curr, y_curr)]
    ax2.plot(x_vals, y_vals, z_vals, color = "green")
    ax3.plot(x_vals, y_vals)                # color changes show the movement during GD


    x_diff = np.abs(x_curr - x_prev)        # record difference in values after GD
    y_diff = np.abs(y_curr - y_prev)
    xy_diff = x_diff**2 + y_diff**2
    if(xy_diff < min_dc):
        break

    x_prev = x_curr                         # update x_prev for next iteration
    y_prev = y_curr

# print value of x at which f(x) is minimum
print("Value of x, y for which value of f(x) is minimum is:")
print("x = ", x_curr)
print("y = ", y_curr)
ax3.plot(x_curr, y_curr, marker="+", markersize=10)
```

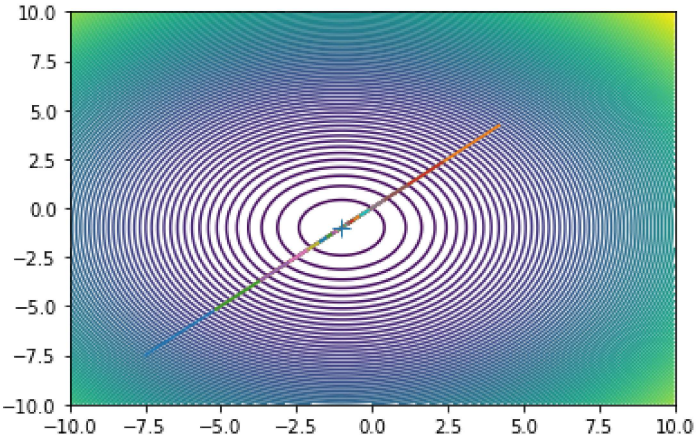
```
Value of x, y for which value of f(x) is minimum is:
x =  -1.0000927711000258
y =  -1.0000927711000258
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x149d65b4460>]
```







Example 2 : $f(x,y) = xsin(x) + ysin(y)$

Gradient Descent Method :

Follow the below steps and write your code in the block below

1. Generate x and y, 1000 data points from -10 to 10
2. Generate and Plot the function $f(x,y) = xsin(x) + ysin(y)$
3. Initialize the starting point (x_{init},y_{init}) and learning rate (λ)
4. Use Gradient descent algorithm to compute value of x and y at which the function $f(x,y)$ is minimum
5. Also vary the learning rate and initialisation point and plot your observations

Derivatives of f w.r.t. x, y : (Analytical)

$$\frac{\partial}{\partial x} f(x,y) = xcosx + sinx$$

$$\frac{\partial}{\partial y} f(x,y) = ycosy + siny$$

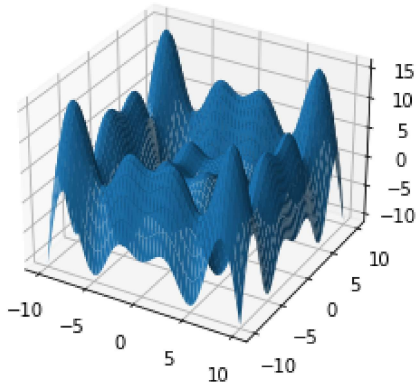
Note that we won't make use of $\frac{\partial^2}{\partial x^2} f(x,y)$, $\frac{\partial^2}{\partial y^2} f(x,y)$, $\frac{\partial^2}{\partial xy} f(x,y)$ in the gradient descent process

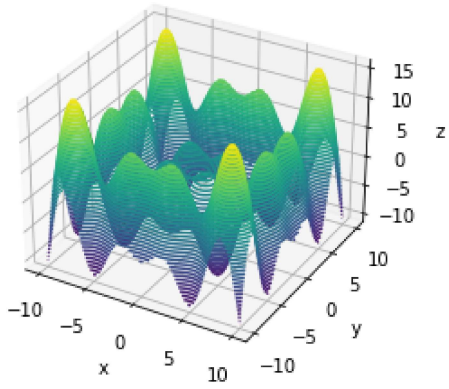
In [ ]:
```
## Write your code here (Ignore the warning)
## Write your code here (Ignore the warning)
## Write your code here

# generate 1000 points
x = np.linspace(-10, 10, 1000)
y = np.linspace(-10, 10, 1000)
x, y = np.meshgrid(x, y)
# generation of function
def f(x, y):
    z = x*np.sin(x) + y*np.sin(y)
    return z
# plotting of function
fig = plt.figure()
ax = plt.axes(projection='3d')
# ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(x, y, f(x,y))

# plotting of contours
plt.figure()
ax2 = plt.axes(projection='3d')
ax2.contour3D(x, y, f(x, y), 50)
ax2.set_xlabel('x')
```

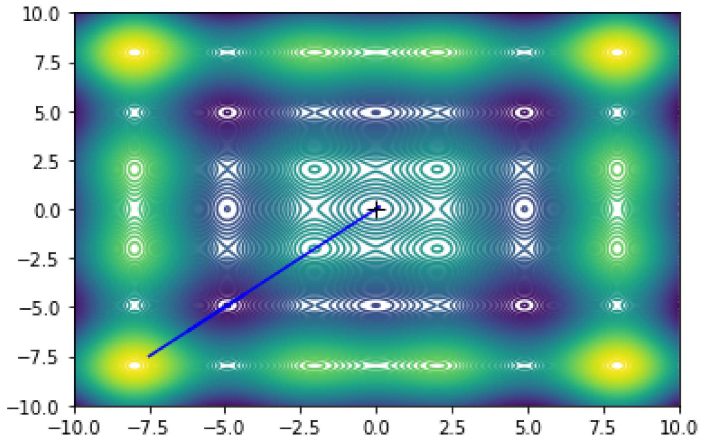
```
ax2.set_ylabel('y')
ax2.set_zlabel('z')

plt.figure()
ax3 = plt.axes()
ax3.contour(x, y, f(x,y), 100)

# initializatin of x_init, y_init, learning rate lambda (lam), num of iterations (n_i), min cost
x_init = -7.5
y_init = -7.5
lam = 0.75
n_i = 10000
min_dc = 0.0000001

# computation of slope + gradient descent
def slope_x(x):
    return (x*np.cos(x) + np.sin(x))
def slope_y(y):
    return (y*np.cos(y) + np.sin(y))

def gradient_descent(x1, y1, lam):
    x2 = x1 - (lam*slope_x(x1))
    y2 = y1 - (lam*slope_y(y1))
    return x2, y2

# initial point setting; then iterations of gradient descent (GD)
x_curr = x_init
x_prev = x_init
y_curr = y_init
y_prev = y_init
for i in range(n_i):
    x_curr, y_curr = gradient_descent(x_curr, y_curr, lam) # do GD step

    x_vals = [x_prev, x_curr]              # plotting of line
    y_vals = [y_prev, y_curr]
    z_vals = [f(x_prev, y_prev), f(x_curr, y_curr)]
    ax2.plot(x_vals, y_vals, z_vals, color = "green")
    ax3.plot(x_vals, y_vals, color="blue")


    x_diff = np.abs(x_curr - x_prev)       # record difference in values after GD
    y_diff = np.abs(y_curr - y_prev)
    xy_diff = x_diff**2 + y_diff**2
    if(xy_diff < min_dc):
        break

    x_prev = x_curr                        # update x_prev for next iteration
    y_prev = y_curr

# print value of x at which f(x) is minimum
print("Value of x, y for which value of f(x) is minimum is:")
print("x = ", x_curr)
print("y = ", y_curr)
ax3.plot(x_curr, y_curr, marker="+", markersize=10, color="black")
```

```
Value of x, y for which value of f(x) is minimum is:
x =  -5.0776181807926396e-05
y =  -5.0776181807926396e-05
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x149e1bcb3a0>]
```


15
10
5
0
−5
−10
−10
−5
0
5
10
10
5
0
−5
−10


15
10
5
0
−5
−10
z
−10
−5
0
5
10
x
10
5
0
−5
−10
y


10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

In [ ]: