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 :

$$x_{init} = 4$$

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

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

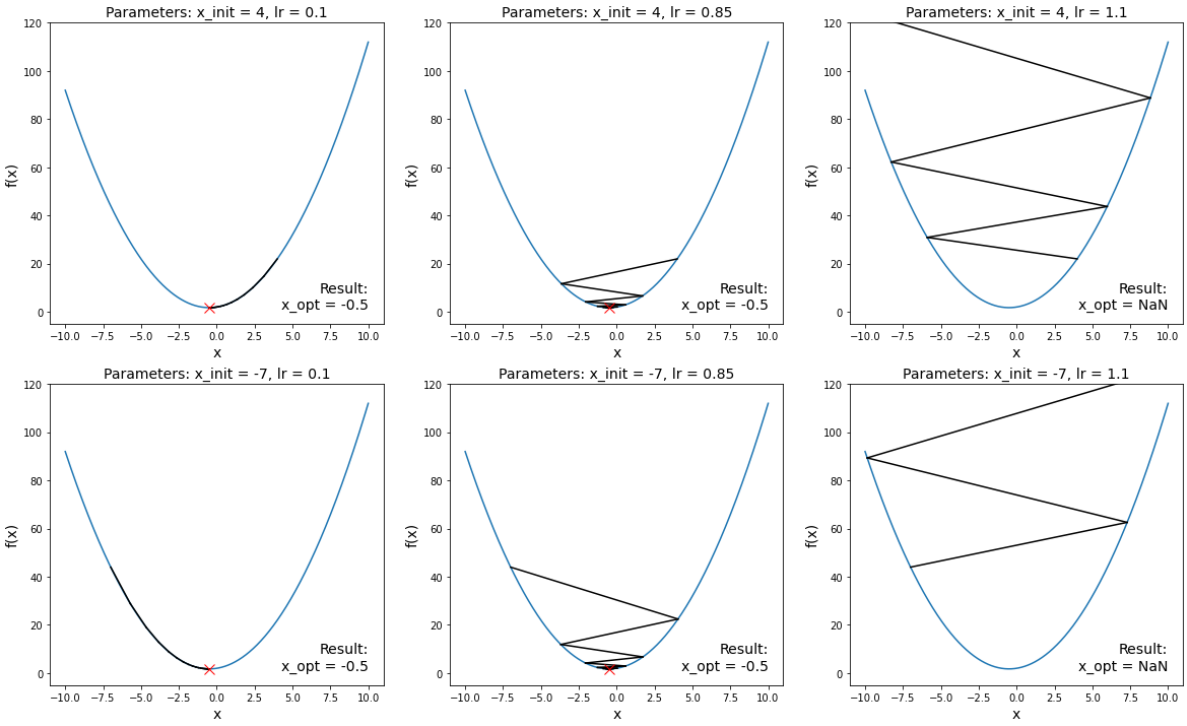
Gradient Descent Method :

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

In [ ]:
```
# Generating x
x = np.linspace(-10, 10, 1000)

# Plotting f(x)
def f(x):
    x = np.array(x)
    return x**2 + x + 2

# Initialising x_opt & learning rate
x_opts = [4, -7]
lrs = [0.1, 0.85, 1.1]

# Gradient descent
def grad_descent(_x_opt, lr, ax):
    found = True
    ax.plot(x, f(x))
    ax.set_xlim(-11, 11)
    ax.set_ylim(-5, 120)
    ax.autoscale(False)
    x_hist = [_x_opt]
    for i in range(1000):
        _x_opt -= lr * (2 * _x_opt + 1)
        if abs(_x_opt) > 15:
            found = False
            x_opt = "NaN"
            break
        x_hist.append(_x_opt)
        ax.plot(x_hist[-2:], f(x_hist[-2:]), color='k')
    if found:
        ax.plot(x_hist[-1], f(x_hist)[-1], color='r', marker='x', markersize
        x_opt = round(_x_opt, 2)
    ax.set_title(f'Parameters: x_init = {x_hist[0]}, lr = {lr}', size=14)
    ax.set_xlabel('x', size=14)
    ax.set_ylabel('f(x)', size=14)
    ax.text(10, 0, f'Result:\nx_opt = {x_opt}', ha='right', va="bottom", siz


fig, axs = plt.subplots(len(x_opts), len(lrs), figsize=(20, 12))
for i, x_opt in enumerate(x_opts):
    for j, lr in enumerate(lrs):
        grad_descent(x_opt, lr, axs[i][j])
```



Example 2 : $f(x) = xsin(x)$

Analytical :

$$f(x) = ssin(x)$$

$$\frac{d}{dx}f(x) = sin(x) + xcos(x) = 0$$

$$\therefore x = \{\pm 7.98, \pm 4.91, \pm 2.03, 0\} \text{ for } x \in [-10, 10]$$

$$\frac{d^2}{dx^2}f(x) = 2cos(x) - xsin(x) > 0 \text{ (Minima)}$$

$$x \in \{\pm 4.91, 0\} \text{ (Analytical Solution)}$$

Gradient Descent Update equation :

$$x_{init} = 4$$

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

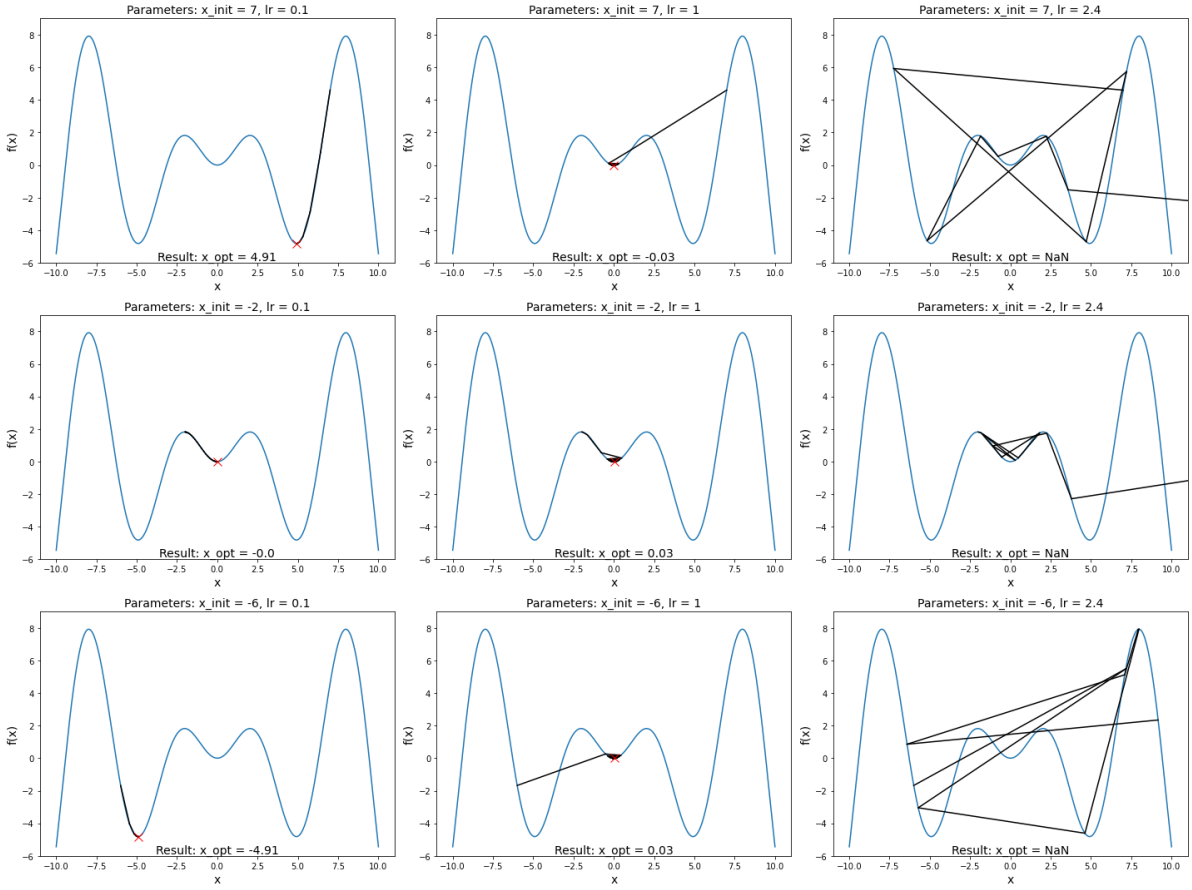
$$x_{updt} = x_{old} - \lambda(sin(x_{old}) + x_{old}cos(x_{old}))$$

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 $f(x)$ = $xsin(x)$
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

In [ ]:
```
# Generating x
x = np.linspace(-10, 10, 1000)

# Plotting f(x)
def f(x):
    x = np.array(x)
    return x * np.sin(x)

# Initialising x_opt & learning rate
x_opts = [7, -2, -6]
lrs = [0.1, 1, 2.4]

# Gradient descent
def grad_descent(_x_opt, lr, axs):
    found = True
    axs.plot(x, f(x))
    axs.set_xlim(-11, 11)
    axs.set_ylim(-6, 9)
    axs.autoscale(False)
    x_hist = [_x_opt]
    for i in range(1000):
        _x_opt -= lr * (np.sin(_x_opt) + _x_opt*np.cos(_x_opt))
        if abs(_x_opt) > 15:
            found = False
            x_opt = "NaN"
            break
        x_hist.append(_x_opt)
        axs.plot(x_hist[-2:], f(x_hist[-2:]), color='k')
    if found:
        axs.plot(x_hist[-1], f(x_hist)[-1], color='r', marker='x', markersi
        x_opt = round(_x_opt, 2)
    axs.set_title(f'Parameters: x_init = {x_hist[0]}, lr = {lr}', size=14)
    axs.set_xlabel('x', size=14)
    axs.set_ylabel('f(x)', size=14)
    axs.text(0, -6, f'Result: x_opt = {x_opt}', ha='center', va="bottom", s


fig, axs = plt.subplots(len(x_opts), len(lrs), figsize=(20, 15))
for i, x_opt in enumerate(x_opts):
    for j, lr in enumerate(lrs):
        grad_descent(x_opt, lr, axs[i][j])

fig.tight_layout()
plt.show()
```



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

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

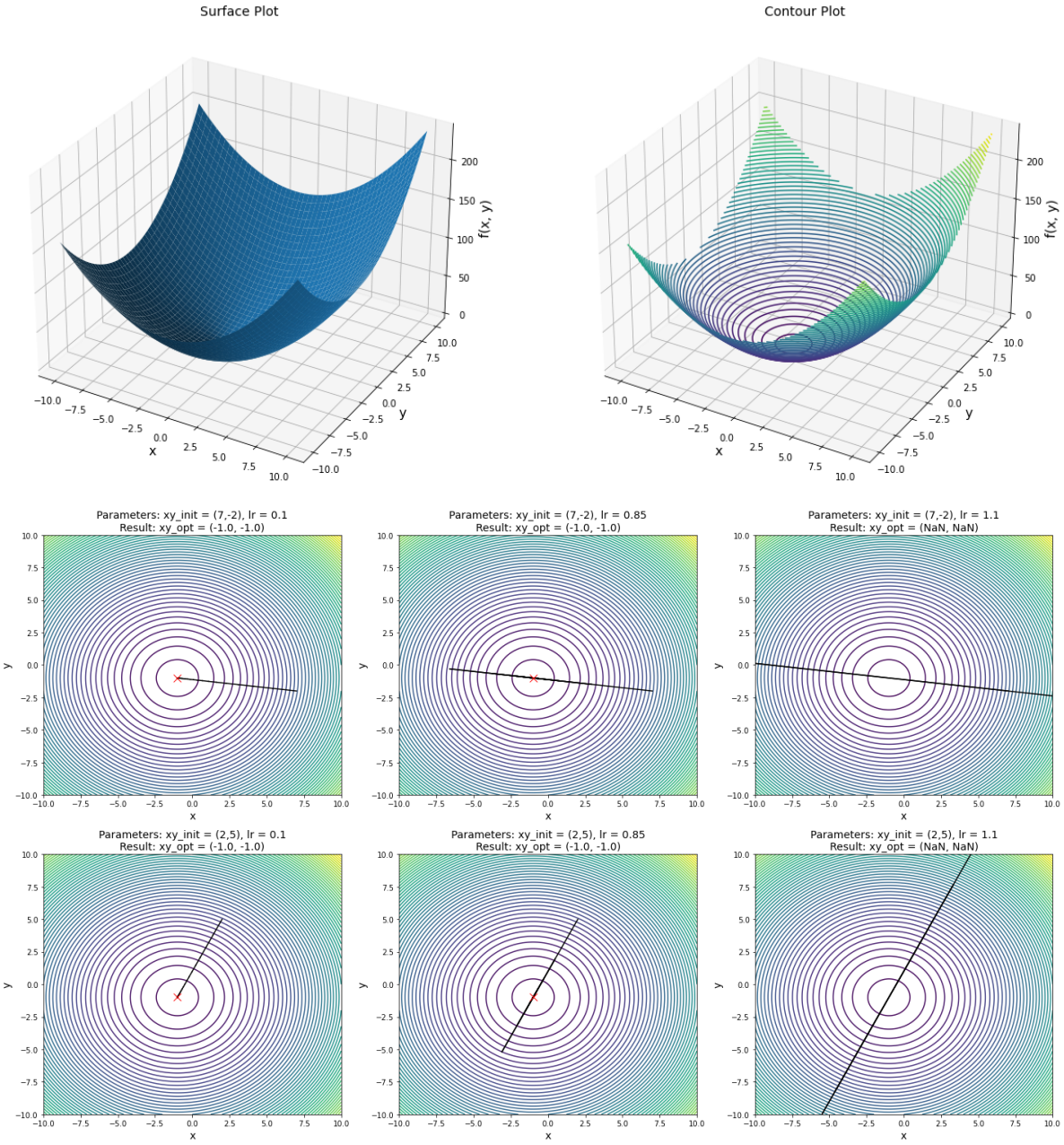
Gradient Descent Method :

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

In [ ]:
```
# Generating x
x = np.linspace(-10, 10, 1000)
y = np.linspace(-10, 10, 1000)
x, y = np.meshgrid(x, y)

def f(x, y):
    x, y = np.array(x), np.array(y)
    return x**2 + y**2 + 2*x + 2*y

def df_dx(x, y):
    x = np.array(x)
    return 2*x + 2
```

```
def df_dy(x, y):
    y = np.array(y)
    return 2*y + 2


# Initialising x_opt & learning rate
x_opts = [7, -2]
y_opts = [2, 5]
lrs = [0.1, 0.85, 1.1]

# Plot the function
z = f(x, y)
fig, ax = plt.subplots(1, 2, subplot_kw={"projection": "3d"}, figsize=(20, 1
ax[0].plot_surface(x, y, z)
ax[0].set_title('Surface Plot', size=14)
ax[0].set_xlabel('x', size=14)
ax[0].set_ylabel('y', size=14)
ax[0].set_zlabel('f(x, y)', size=14)
ax[1].contour(x, y, z, 75)
ax[1].set_title('Contour Plot', size=14)
ax[1].set_xlabel('x', size=14)
ax[1].set_ylabel('y', size=14)
ax[1].set_zlabel('f(x, y)', size=14)
plt.show()

# Gradient descent
def grad_descent(_x_opt, _y_opt, lr, ax):
    found = True
    ax.contour(x, y, f(x, y), 75)
    x_hist = [_x_opt]
    y_hist = [_y_opt]
    ax.autoscale(False)
    for i in range(1000):
        _x_opt -= lr * df_dx(_x_opt, _y_opt)
        _y_opt -= lr * df_dy(_x_opt, _y_opt)
        if abs(_x_opt) > 12 or abs(_y_opt) > 12:
            found = False
            x_opt = "NaN"
            y_opt = "NaN"
            break
        x_hist.append(_x_opt)
        y_hist.append(_y_opt)
        ax.plot(x_hist[-2:], y_hist[-2:], color='k')
    if found:
        ax.plot(x_hist[-1], y_hist[-1], color='r', marker='x', markersize=10
        x_opt = round(_x_opt, 2)
        y_opt = round(_y_opt, 2)
    ax.set_title(f'Parameters: xy_init = ({x_hist[0]},{y_hist[0]}), lr = {lr
    ax.set_xlabel('x', size=14)
    ax.set_ylabel('y', size=14)


fig, axs = plt.subplots(len(x_opts), len(lrs), figsize=(20, 12))
for i, (x_opt, y_opt) in enumerate((x_opts, y_opts)):
    for j, lr in enumerate(lrs):
        grad_descent(x_opt, y_opt, lr, axs[i][j])

fig.tight_layout()
plt.show()
```



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

In [ ]:
```
# Generating x
x = np.linspace(-10, 10, 1000)
y = np.linspace(-10, 10, 1000)
x, y = np.meshgrid(x, y)

def f(x, y):
    x, y = np.array(x), np.array(y)
```

```
    return x * np.sin(x) + y * np.sin(y)

def df_dx(x, y):
    x = np.array(x)
    return np.sin(x) + x * np.cos(x)

def df_dy(x, y):
    y = np.array(y)
    return np.sin(y) + y * np.cos(y)


# Initialising x_opt & learning rate
x_opts = [7, -2]
y_opts = [2, 5]
lrs = [0.1, 1, 3]

# Plot the function
z = f(x, y)
fig, ax = plt.subplots(1, 2, subplot_kw={"projection": "3d"}, figsize=(20, 1
ax[0].plot_surface(x, y, z)
ax[0].set_title('Surface Plot', size=14)
ax[0].set_xlabel('x', size=14)
ax[0].set_ylabel('y', size=14)
ax[0].set_zlabel('f(x, y)', size=14)
ax[1].contour(x, y, z, 75)
ax[1].set_title('Contour Plot', size=14)
ax[1].set_xlabel('x', size=14)
ax[1].set_ylabel('y', size=14)
ax[1].set_zlabel('f(x, y)', size=14)
ax[0].plot_surface(x, y, z)
ax[1].contour(x, y, z, 75)
plt.show()

# Gradient descent
def grad_descent(_x_opt, _y_opt, lr, ax):
    found = True
    ax.contour(x, y, f(x, y), 75)
    x_hist = [_x_opt]
    y_hist = [_y_opt]
    ax.autoscale(False)
    for i in range(5000):
        _x_opt -= lr * df_dx(_x_opt, _y_opt)
        _y_opt -= lr * df_dy(_x_opt, _y_opt)
        if abs(_x_opt) > 25 or abs(_y_opt) > 25:
            found = False
            x_opt = "NaN"
            y_opt = "NaN"
            break
        x_hist.append(_x_opt)
        y_hist.append(_y_opt)
        ax.plot(x_hist[-2:], y_hist[-2:], color='k')
    if found:
        ax.plot(x_hist[-1], y_hist[-1], color='r', marker='x', markersize=10
        x_opt = round(_x_opt, 2)
        y_opt = round(_y_opt, 2)
    ax.set_title(f'Parameters: xy_init = ({x_hist[0]},{y_hist[0]}), lr = {lr
    ax.set_xlabel('x', size=14)
    ax.set_ylabel('y', size=14)


fig, axs = plt.subplots(len(x_opts), len(lrs), figsize=(20, 12))
for i, (x_opt, y_opt) in enumerate((x_opts, y_opts)):
    for j, lr in enumerate(lrs):
        grad_descent(x_opt, y_opt, lr, axs[i][j])
```

```
fig.tight_layout()
plt.show()
```