Machine Learning Reading Seminar Code Computation.

Note that : For stochastic optimization, each computation will be different due to randomly select.

Generate the Datas

In [ ]:

```
"""
# Generating synthetic data
n = 500
x = np.random.normal(0, 1, n)
epsilon = np.random.normal(0, 1, n)
beta_true = np.array([2, 3])
Y = beta_true[0] + beta_true[1] * x + epsilon

# Save synthetic data
np.save("synthetic_x.npy", x)
np.save("synthetic_Y.npy", Y)
"""
```

Out[ ]:

```
'\n# Generating synthetic data\nn = 500\nx = np.random.normal(0, 1, n)\neps
ilon = np.random.normal(0, 1, n)\nbeta_true = np.array([2, 3])\nY = beta_tr
ue[0] + beta_true[1] * x + epsilon\n\n# Save synthetic data\nnp.save("synth
etic_x.npy", x)\nnp.save("synthetic_Y.npy", Y)\n'
```

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
```

```
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel
(R) SSE4.2) enabled only processors has been deprecated. Intel oneAPI Math
Kernel Library 2025.0 will require Intel(R) Advanced Vector Extensions (Int
el(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel
(R) SSE4.2) enabled only processors has been deprecated. Intel oneAPI Math
Kernel Library 2025.0 will require Intel(R) Advanced Vector Extensions (Int
el(R) AVX) instructions.
```

In [ ]:

```
# beta_0 = 2 and beta_1 = 3
beta_true = np.array([2, 3])
# this data we select for good output with Gradient Descent with Momentum
x = np.load("synthetic_x.npy")
Y = np.load("synthetic_Y.npy")
```

Loss function :

$$\text{Loss} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - (\beta_0 + \beta_1 x_i))^2$$

In [ ]:

```
def loss(Y, x, beta):
    return np.mean((Y - (beta[0] + beta[1] * x))**2)
```

Gradient respect to β_0 and β_1 we

$$\frac{\partial}{\partial \beta_0}\text{Loss} = -\frac{2}{n}\sum_{i=1}^{n} Y_i - (\beta_0 + \beta_1 x_i)$$

$$\frac{\partial}{\partial \beta_1}\text{Loss} = -\frac{2}{n}\sum_{i=1}^{n} Y_i - (\beta_0 + \beta_1 x_i) x_i$$

In [ ]:

```
def compute_gradient(Y, x, beta):
    residuals = Y - (beta[0] + beta[1] * x)
    d_beta0 = -2 * np.mean(residuals)
    d_beta1 = -2 * np.mean(residuals * x)
    return np.array([d_beta0, d_beta1])
```

Gradient Descent Function

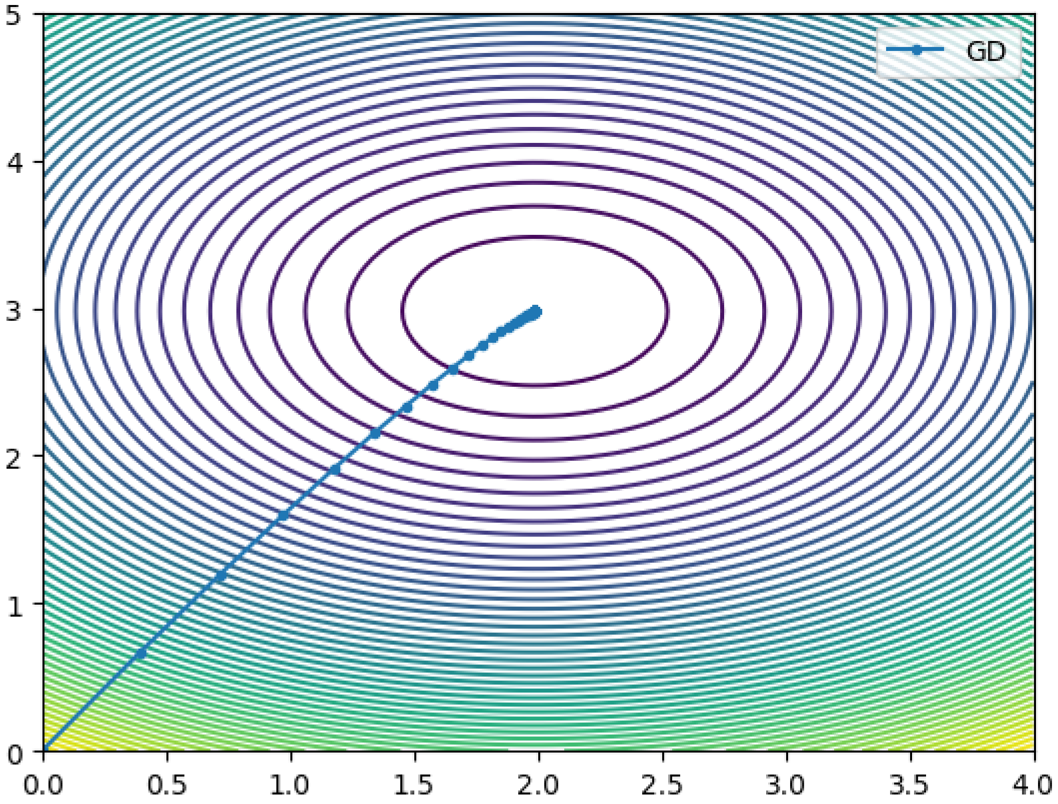
In [ ]:

```
def GD(x, Y, num_iterations, alpha):
    beta = np.array([0, 0])
    beta_values = [beta]
    loss_values = []
    for _ in range(num_iterations):
        gradient = compute_gradient(Y, x, beta)
        beta = beta - alpha * gradient
        beta_values.append(beta)
        loss_values.append(loss(Y, x, beta))
    return beta_values,loss_values
```

In [ ]:

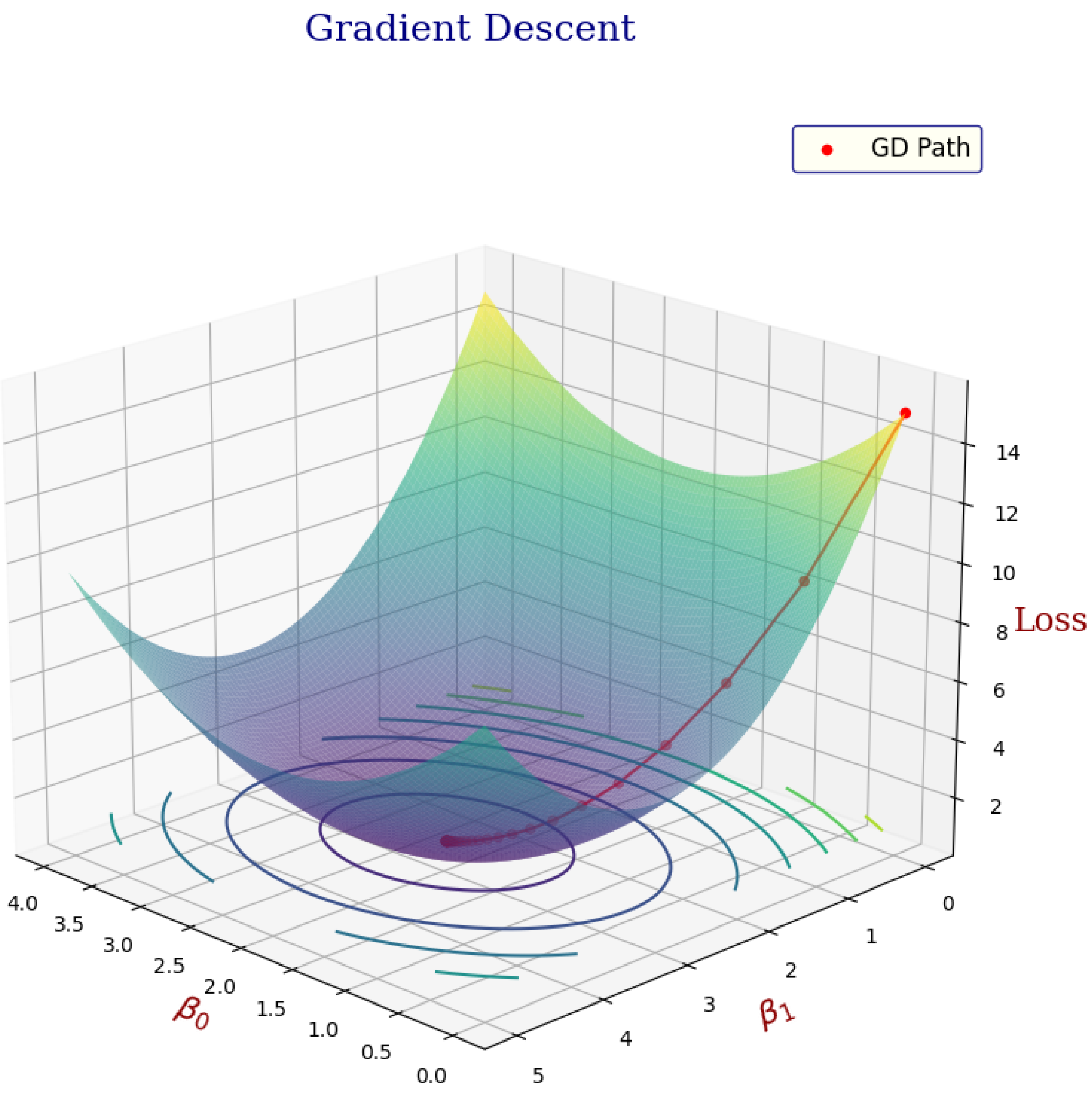
```
gd_iterations = 50
learning_rate = 0.1

gd_path,gd_loss  = GD(x, Y, gd_iterations, learning_rate)
gd_path = np.array(gd_path)

B0_range = [min(gd_path[:, 0].min(), beta_true[0]-2), max(gd_path[:, 0].max(
B1_range = [min(gd_path[:, 1].min(), beta_true[1]-2), max(gd_path[:, 1].max(
B0, B1 = np.meshgrid(np.linspace(B0_range[0], B0_range[1], 100),
                     np.linspace(B1_range[0], B1_range[1], 100))
Loss = np.zeros_like(B0)

for i in range(B0.shape[0]):
    for j in range(B0.shape[1]):
        Loss[i, j] = loss(Y, x, [B0[i, j], B1[i, j]])

levels = np.linspace(np.min(Loss), np.max(Loss), 50)
plt.contour(B0, B1, Loss, levels=levels)
plt.plot(gd_path[:, 0], gd_path[:, 1], label='GD', marker='.')
```

```
plt.legend()
plt.show()
```



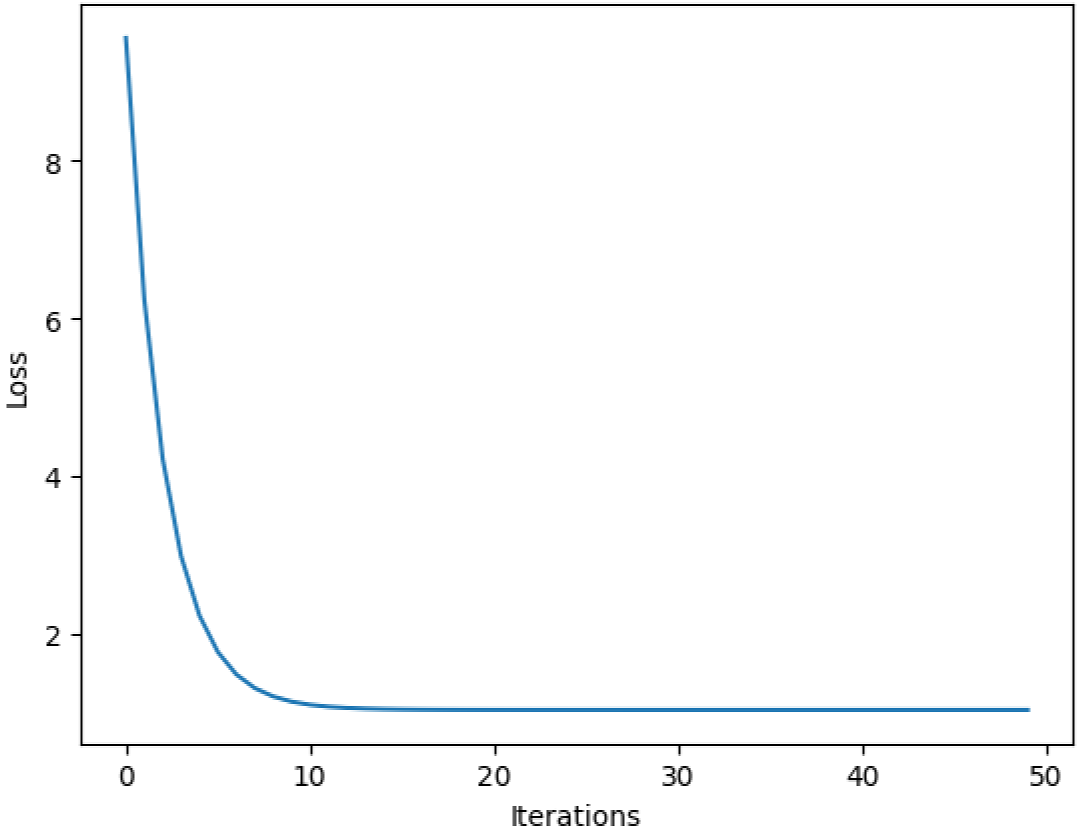
In [ ]:

```
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

surf = ax.plot_surface(B0, B1, Loss, cmap='viridis', rstride=1, cstride=1, a
contour = ax.contour(B0, B1, Loss, zdir='z', offset=np.min(Loss), cmap='viri

ax.scatter(gd_path[:, 0], gd_path[:, 1], np.array([loss(Y, x, beta) for beta
ax.plot(gd_path[:, 0], gd_path[:, 1], np.array([loss(Y, x, beta) for beta in


font_dict = {'family': 'serif', 'color':  'darkred', 'weight': 'normal', 'si
ax.set_xlabel(r'$\beta_0$', fontdict=font_dict)
ax.set_ylabel(r'$\beta_1$', fontdict=font_dict)
ax.set_zlabel('Loss', fontdict=font_dict)
ax.set_title('Gradient Descent', pad=35, fontsize=18, fontdict={'family': 's

ax.legend(loc='upper right', fontsize=12, frameon=True, facecolor='ivory', e

ax.view_init(elev=20, azim=135)

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



In [ ]:

```
iterations = range(len(gd_loss))
plt.plot(iterations, gd_loss, '-')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.show()
```



Stochastic Gradient Descent

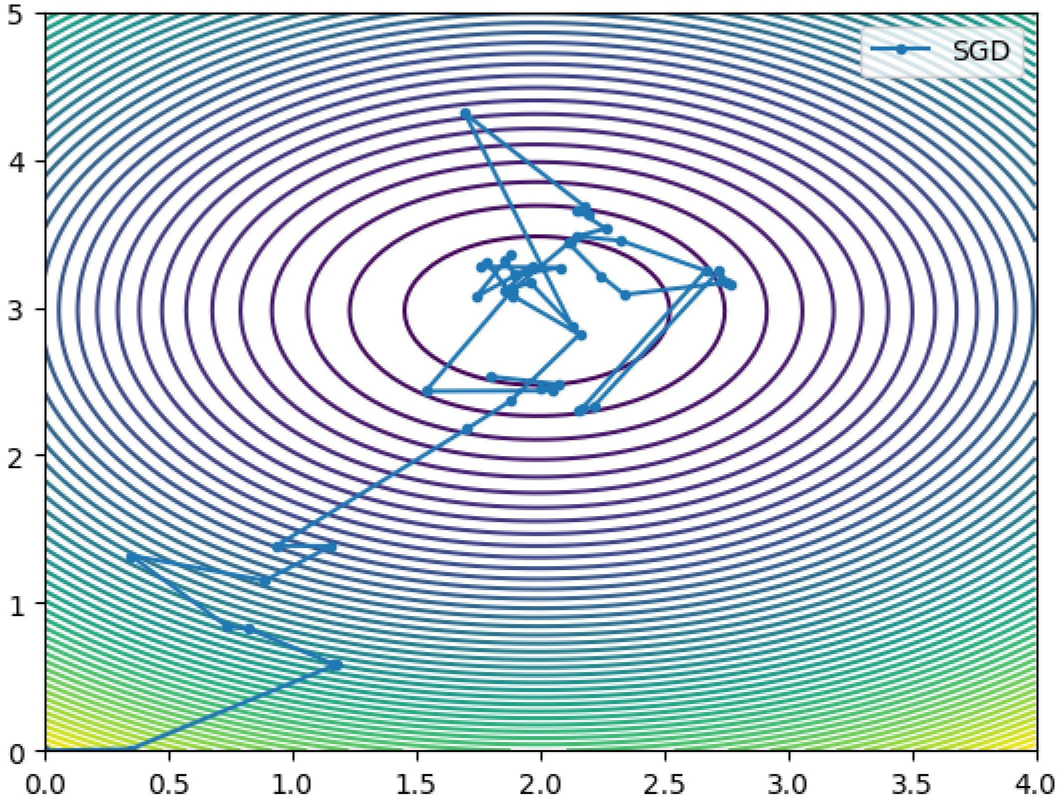
In [ ]:

```
def compute_stochastic_gradient(Y, x, beta, index):
    residual = Y[index] - (beta[0] + beta[1] * x[index])
    d_beta0 = -2 * residual
    d_beta1 = -2 * residual * x[index]
    return np.array([d_beta0, d_beta1])


def SGD(x, Y, num_iterations, alpha):
    beta = np.array([0, 0]) # Initial guess
    beta_values = [beta]
    loss_values = []

    for _ in range(num_iterations):
        index = np.random.randint(0, len(Y))  # Randomly select one data poi
        gradient = compute_stochastic_gradient(Y, x, beta, index)
        beta = beta - alpha * gradient
        beta_values.append(beta)
        loss_values.append(loss(Y, x, beta))

    return beta_values, loss_values
```

In [ ]:

```
num_iterations = 50
alpha = 0.1

sgd_path, sgd_loss = SGD(x, Y, num_iterations, alpha)
```

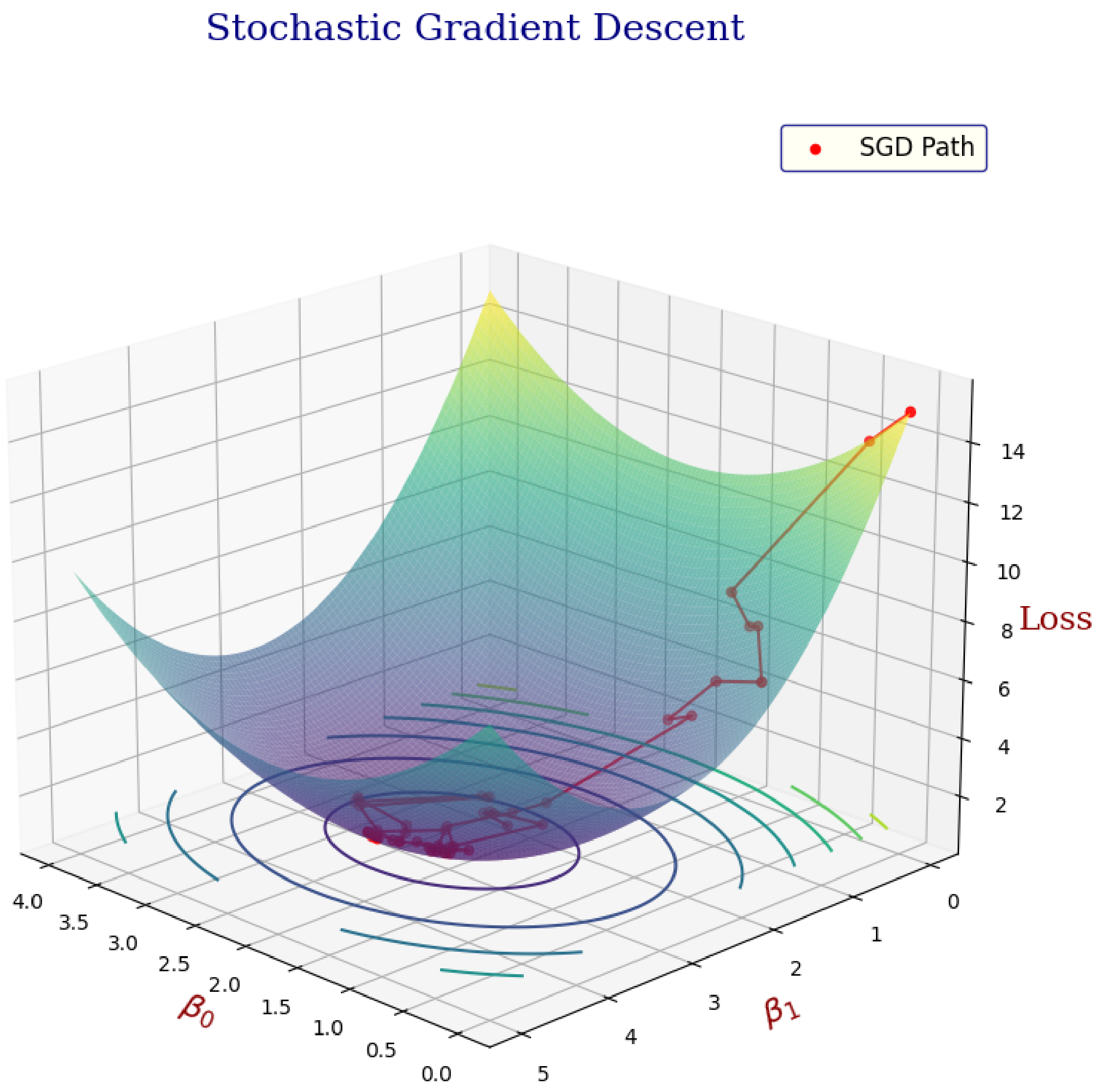
```
sgd_path = np.array(sgd_path)

B0_range = [min(sgd_path[:, 0].min(), beta_true[0]-2), max(sgd_path[:, 0].ma
B1_range = [min(sgd_path[:, 1].min(), beta_true[1]-2), max(sgd_path[:, 1].ma
B0, B1 = np.meshgrid(np.linspace(B0_range[0], B0_range[1], 100),
                     np.linspace(B1_range[0], B1_range[1], 100))
Loss = np.zeros_like(B0)

for i in range(B0.shape[0]):
    for j in range(B0.shape[1]):
        Loss[i, j] = loss(Y, x, [B0[i, j], B1[i, j]])

levels = np.linspace(np.min(Loss), np.max(Loss), 50)
plt.contour(B0, B1, Loss, levels=levels)
plt.plot(sgd_path[:, 0], sgd_path[:, 1], label='SGD', marker='.')
plt.legend()
plt.show()
```



In [ ]:

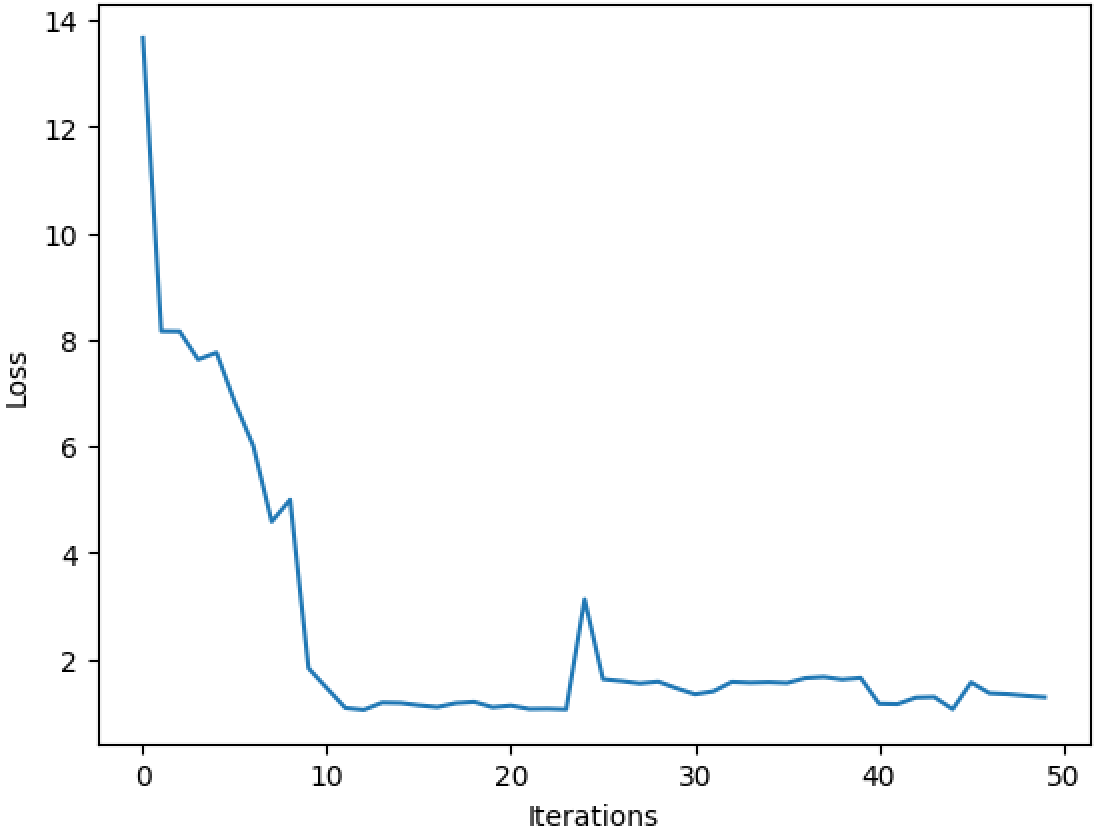
```
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

surf = ax.plot_surface(B0, B1, Loss, cmap='viridis', rstride=1, cstride=1, a
contour = ax.contour(B0, B1, Loss, zdir='z', offset=np.min(Loss), cmap='viri

ax.scatter(sgd_path[:, 0], sgd_path[:, 1], np.array([loss(Y, x, beta) for be
ax.plot(sgd_path[:, 0], sgd_path[:, 1], np.array([loss(Y, x, beta) for beta 


font_dict = {'family': 'serif', 'color':  'darkred', 'weight': 'normal', 'si
ax.set_xlabel(r'$\beta_0$', fontdict=font_dict)
ax.set_ylabel(r'$\beta_1$', fontdict=font_dict)
```

```
ax.set_zlabel('Loss', fontdict=font_dict)
ax.set_title('Stochastic Gradient Descent', pad=35, fontsize=18, fontdict={'

ax.legend(loc='upper right', fontsize=12, frameon=True, facecolor='ivory', e

ax.view_init(elev=20, azim=135)

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



In [ ]:

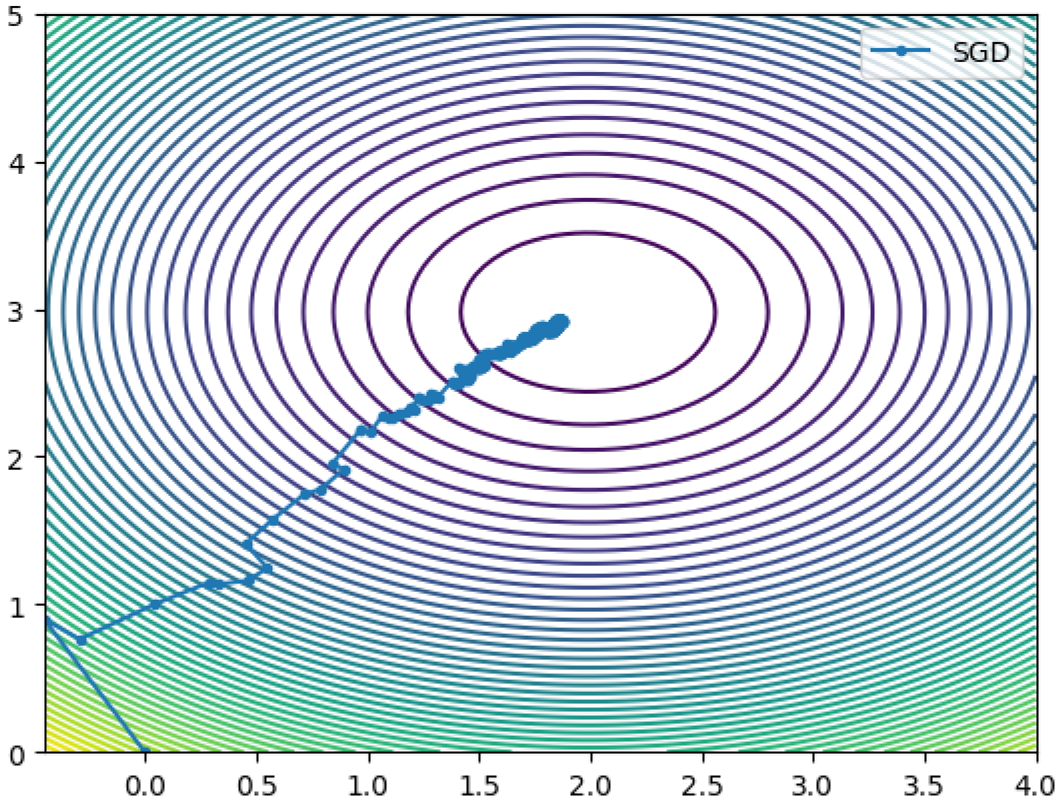
```
iterations = range(len(sgd_loss))
plt.plot(iterations, sgd_loss, '-')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.show()
```



In [ ]:

```
def SGD(x, Y, num_iterations, alpha):
    beta = np.array([0, 0])
    beta_values = [beta]
    loss_values = []
    count = 1
    for _ in range(num_iterations):
        index = np.random.randint(0, len(Y))
        gradient = compute_stochastic_gradient(Y, x, beta, index)
        beta = beta - alpha/count**(3/4) * gradient
        beta_values.append(beta)
        loss_values.append(loss(Y, x, beta))

        count = count + 1
    return beta_values,loss_values

sgd2_iterations = 700
learning_rate_sgd2 = 0.1

sgd_path2,sgd_loss2 = SGD(x, Y, sgd2_iterations, learning_rate_sgd2)
sgd_path2 = np.array(sgd_path2)

B0_range = [min(sgd_path2[:, 0].min(), beta_true[0]-2), max(sgd_path2[:, 0].
B1_range = [min(sgd_path2[:, 1].min(), beta_true[1]-2), max(sgd_path2[:, 1].
B0, B1 = np.meshgrid(np.linspace(B0_range[0], B0_range[1], 100),
                     np.linspace(B1_range[0], B1_range[1], 100))
Loss = np.zeros_like(B0)

for i in range(B0.shape[0]):
    for j in range(B0.shape[1]):
```

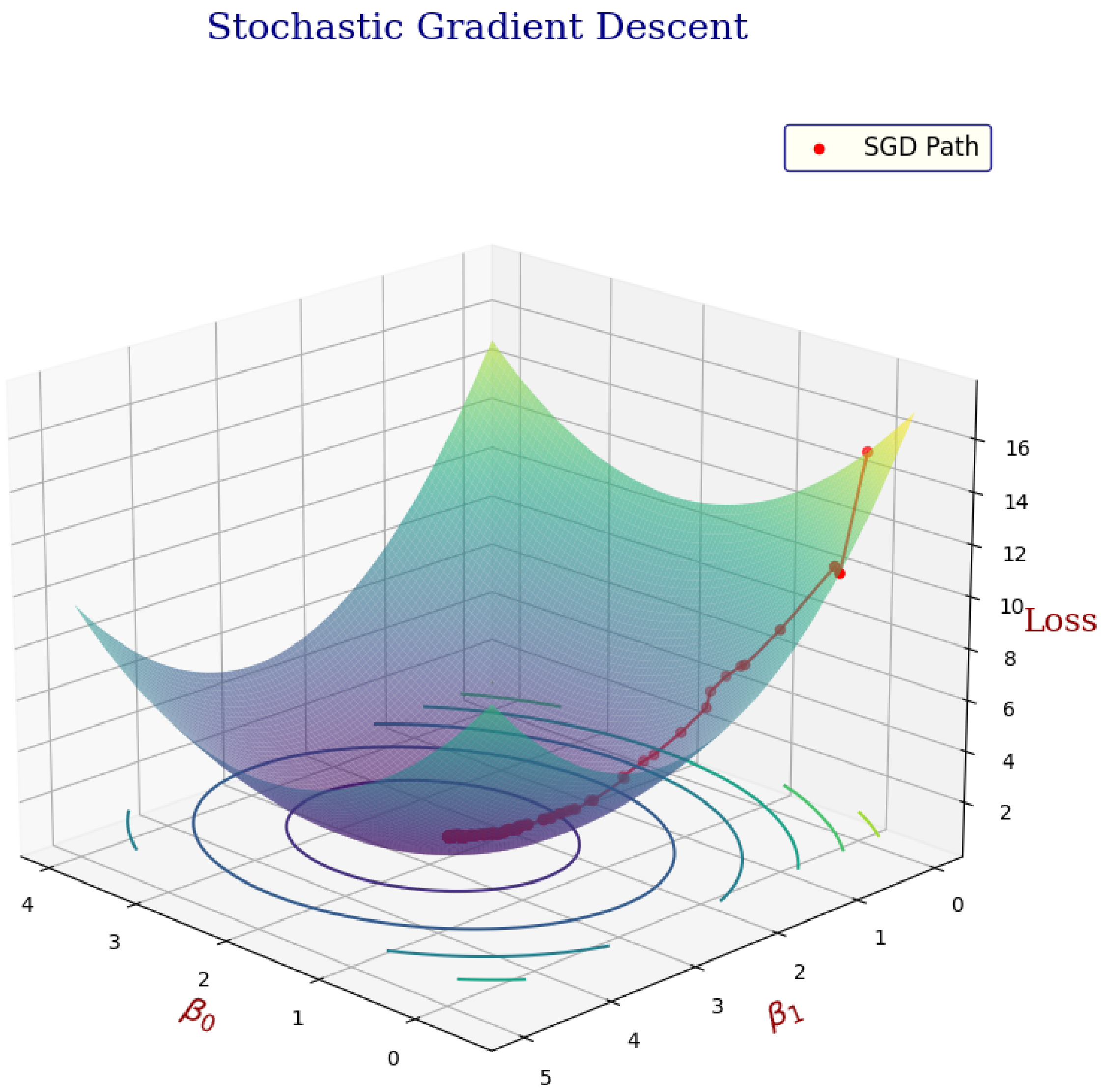
```
        Loss[i, j] = loss(Y, x, [B0[i, j], B1[i, j]])

levels = np.linspace(np.min(Loss), np.max(Loss), 50)
plt.contour(B0, B1, Loss, levels=levels)
plt.plot(sgd_path2[:, 0], sgd_path2[:, 1], label='SGD', marker='.')

plt.legend()
plt.show()
```



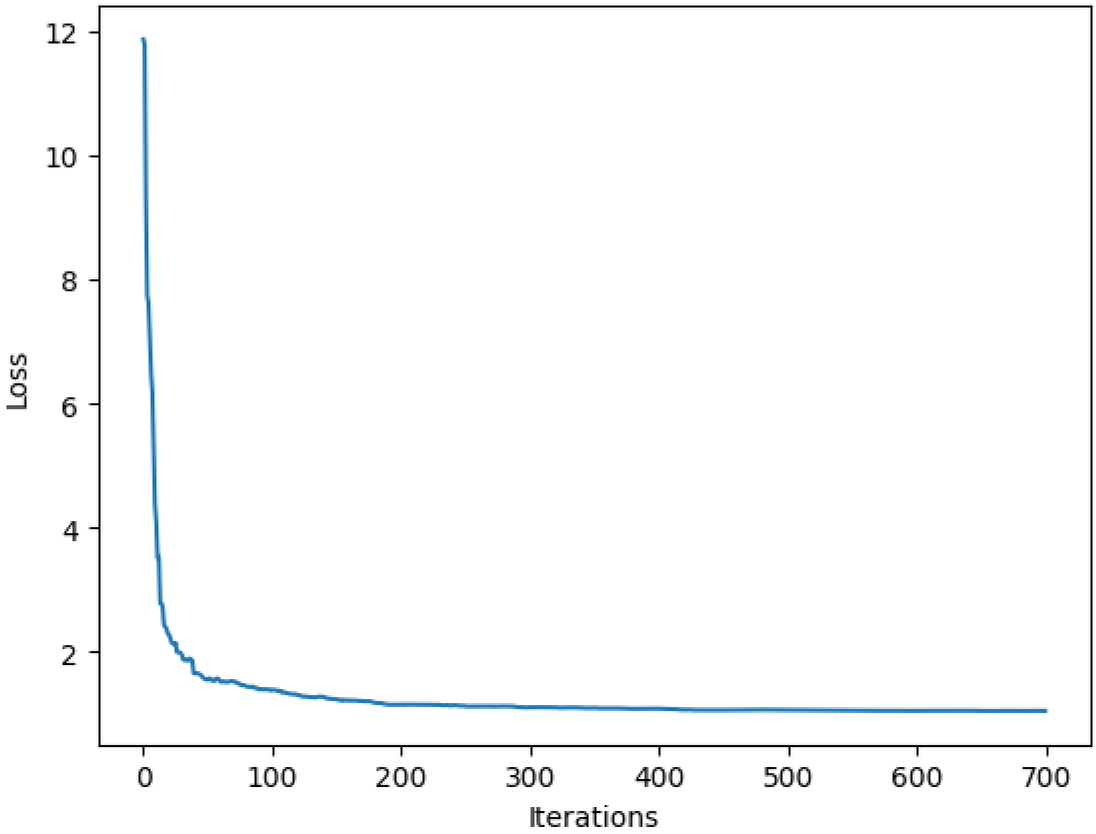
In [ ]:

```
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

surf = ax.plot_surface(B0, B1, Loss, cmap='viridis', rstride=1, cstride=1, a
contour = ax.contour(B0, B1, Loss, zdir='z', offset=np.min(Loss), cmap='viri

ax.scatter(sgd_path2[:, 0], sgd_path2[:, 1], np.array([loss(Y, x, beta) for
ax.plot(sgd_path2[:, 0], sgd_path2[:, 1], np.array([loss(Y, x, beta) for bet


font_dict = {'family': 'serif', 'color':  'darkred', 'weight': 'normal', 'si
ax.set_xlabel(r'$\beta_0$', fontdict=font_dict)
ax.set_ylabel(r'$\beta_1$', fontdict=font_dict)
ax.set_zlabel('Loss', fontdict=font_dict)
ax.set_title('Stochastic Gradient Descent', pad=35, fontsize=18, fontdict={'

ax.legend(loc='upper right', fontsize=12, frameon=True, facecolor='ivory', e

ax.view_init(elev=20, azim=135)
```

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



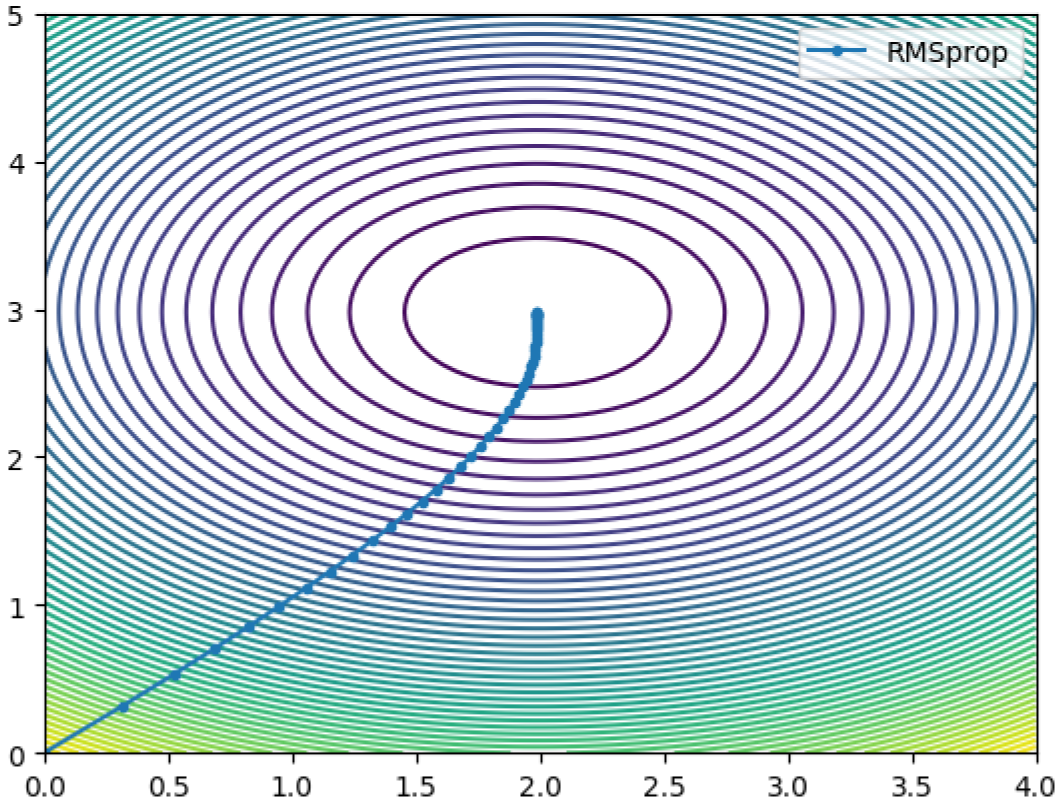
```
In [ ]: iterations = range(len(sgd_loss2))
        plt.plot(iterations, sgd_loss2, '-')
        plt.xlabel('Iterations')
        plt.ylabel('Loss')
        plt.show()
```



In [ ]:

```
def RMSprop(x, Y, num_iterations, alpha, decay_factor=0.9, epsilon=1e-8):
    params = np.array([0, 0])  # Initial guess
    params_values = [params]
    loss_values = []
    s = np.zeros_like(params)  # Initialize running average of squared gradi

    for _ in range(num_iterations):
        gradient = compute_gradient(Y, x, params)
        s = decay_factor * s + (1 - decay_factor) * gradient**2
        params = params - alpha * gradient / (np.sqrt(s+epsilon))
        params_values.append(params)
        loss_values.append(loss(Y, x, params))


    return params_values,loss_values

Rmsprop_iteration = 50
alpha = 0.1

rmsprop_path,rmsprop_loss = RMSprop(x, Y, Rmsprop_iteration, alpha)
rmsprop_path = np.array(rmsprop_path)

B0_range = [min(rmsprop_path[:, 0].min(), beta_true[0]-2), max(rmsprop_path[
B1_range = [min(rmsprop_path[:, 1].min(), beta_true[1]-2), max(rmsprop_path[
B0, B1 = np.meshgrid(np.linspace(B0_range[0], B0_range[1], 100),
                     np.linspace(B1_range[0], B1_range[1], 100))
Loss = np.zeros_like(B0)

for i in range(B0.shape[0]):
```

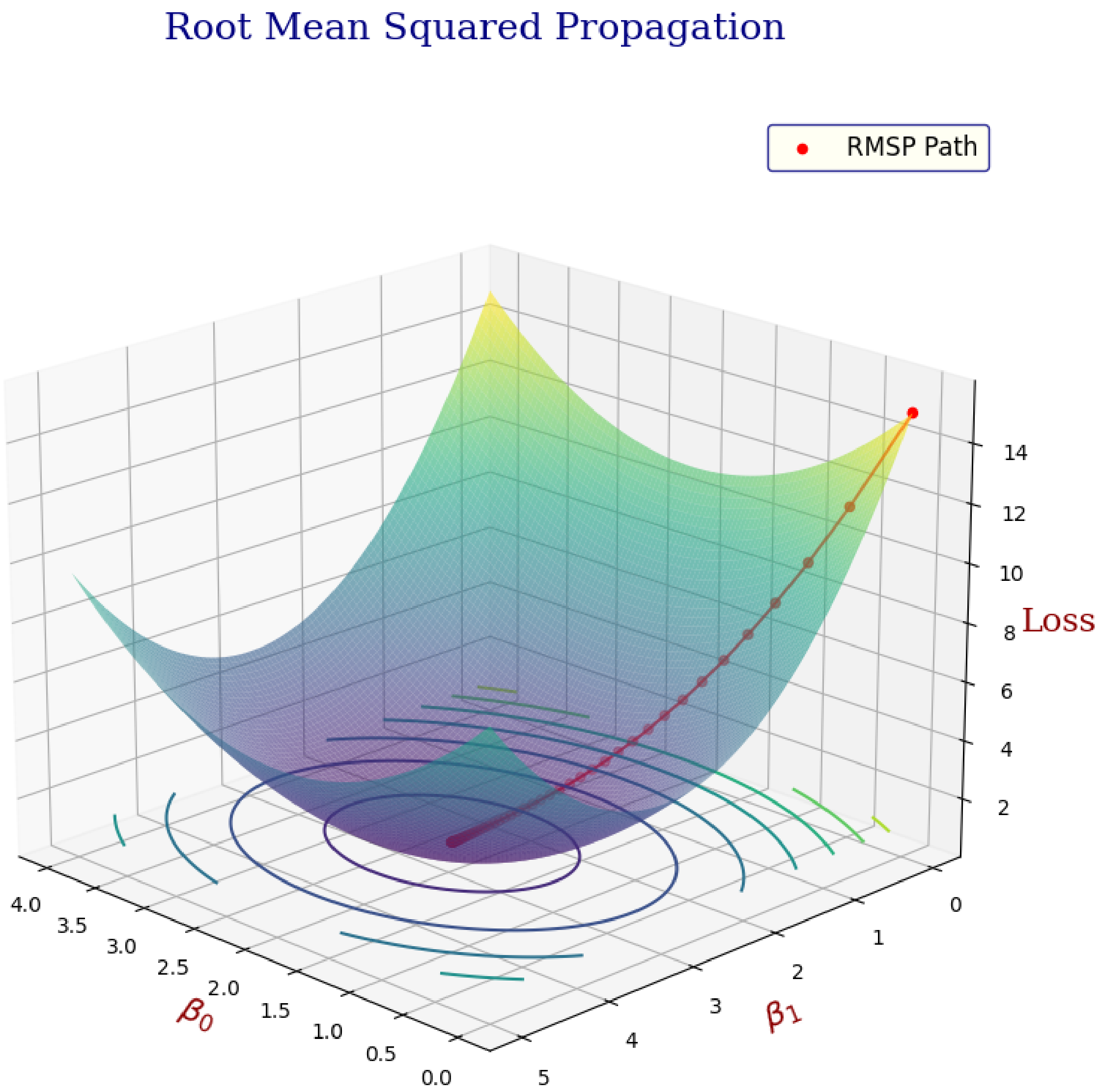
```
    for j in range(B0.shape[1]):
        Loss[i, j] = loss(Y, x, [B0[i, j], B1[i, j]])


levels = np.linspace(np.min(Loss), np.max(Loss), 50)
plt.contour(B0, B1, Loss, levels=levels)
plt.plot(rmsprop_path[:, 0], rmsprop_path[:, 1], label='RMSprop', marker='.'
plt.legend()
plt.show()
```



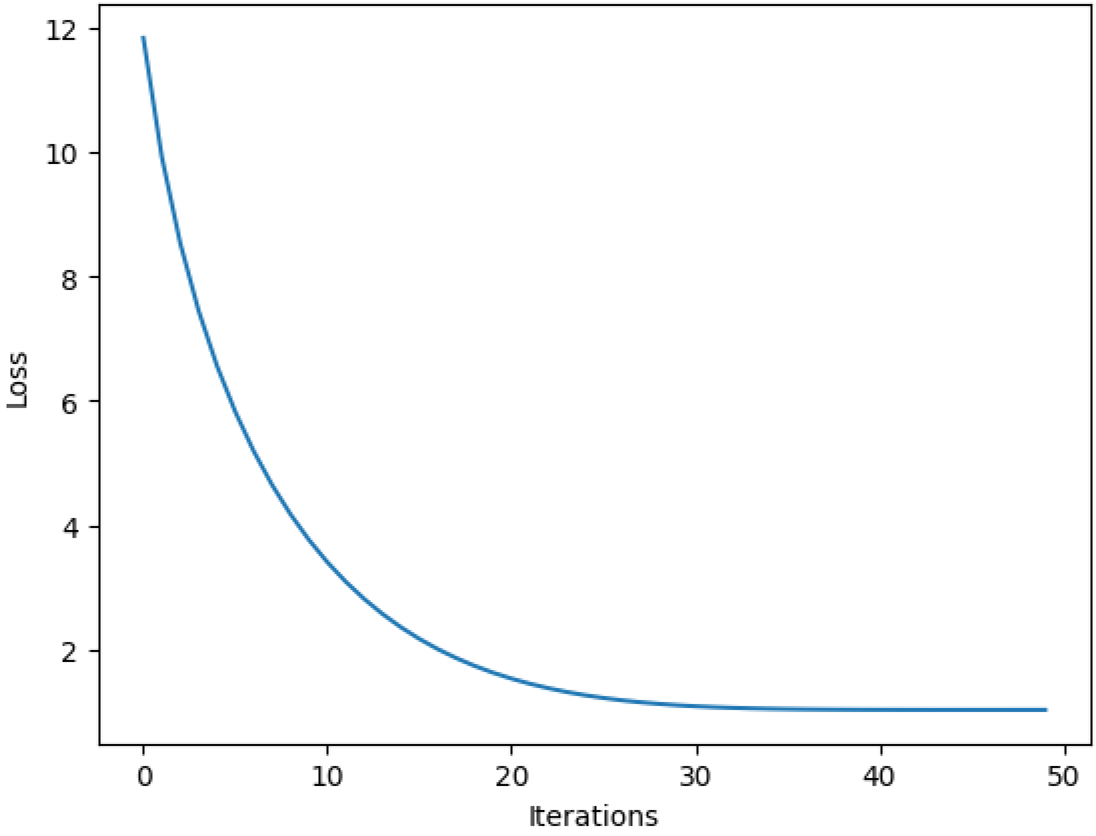
In [ ]:

```
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

surf = ax.plot_surface(B0, B1, Loss, cmap='viridis', rstride=1, cstride=1, a
contour = ax.contour(B0, B1, Loss, zdir='z', offset=np.min(Loss), cmap='viri

ax.scatter(rmsprop_path[:, 0], rmsprop_path[:, 1], np.array([loss(Y, x, beta
ax.plot(rmsprop_path[:, 0], rmsprop_path[:, 1], np.array([loss(Y, x, beta) f


font_dict = {'family': 'serif', 'color':  'darkred', 'weight': 'normal', 'si
ax.set_xlabel(r'$\beta_0$', fontdict=font_dict)
ax.set_ylabel(r'$\beta_1$', fontdict=font_dict)
ax.set_zlabel('Loss', fontdict=font_dict)
ax.set_title('Root Mean Squared Propagation', pad=35, fontsize=18, fontdict=

ax.legend(loc='upper right', fontsize=12, frameon=True, facecolor='ivory', e

ax.view_init(elev=20, azim=135)
```

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



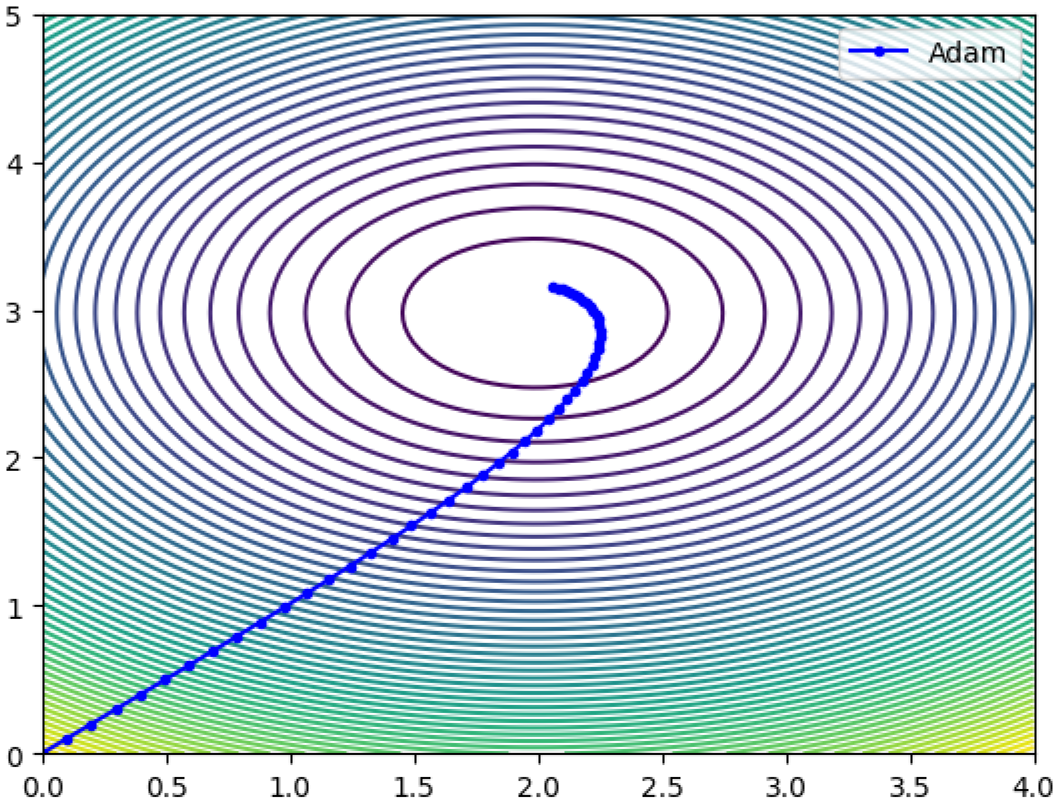
```
In [ ]: iterations = range(len(rmsprop_loss))
        plt.plot(iterations, rmsprop_loss, '-')
        plt.xlabel('Iterations')
        plt.ylabel('Loss')
        plt.show()
```



In [ ]:

```
def Adam(x, Y, num_iterations, alpha, beta1=0.9, beta2=0.999, epsilon=1e-8):
    beta = np.array([0, 0])
    beta_values = [beta]
    loss_values = []
    m = np.array([0, 0])
    v = np.array([0, 0])

    for t in range(1, num_iterations + 1):
        gradient = compute_gradient(Y, x, beta)
        m = beta1 * m + (1 - beta1) * gradient
        v = beta2 * v + (1 - beta2) * gradient**2

        m_corrected = m / (1 - beta1**t)
        v_corrected = v / (1 - beta2**t)

        beta = beta - alpha * m_corrected / (np.sqrt(v_corrected+epsilon))
        beta_values.append(beta)
        loss_values.append(loss(Y, x, beta))

    return beta_values,loss_values

adam_path,adam_loss = Adam(x, Y, 50, 0.1)
adam_path = np.array(adam_path)

B0_range = [min(adam_path[:, 0].min(), beta_true[0]-2), max(adam_path[:, 0].
B1_range = [min(adam_path[:, 1].min(), beta_true[1]-2), max(adam_path[:, 1].
B0, B1 = np.meshgrid(np.linspace(B0_range[0], B0_range[1], 100),
                     np.linspace(B1_range[0], B1_range[1], 100))
Loss = np.zeros_like(B0)
```

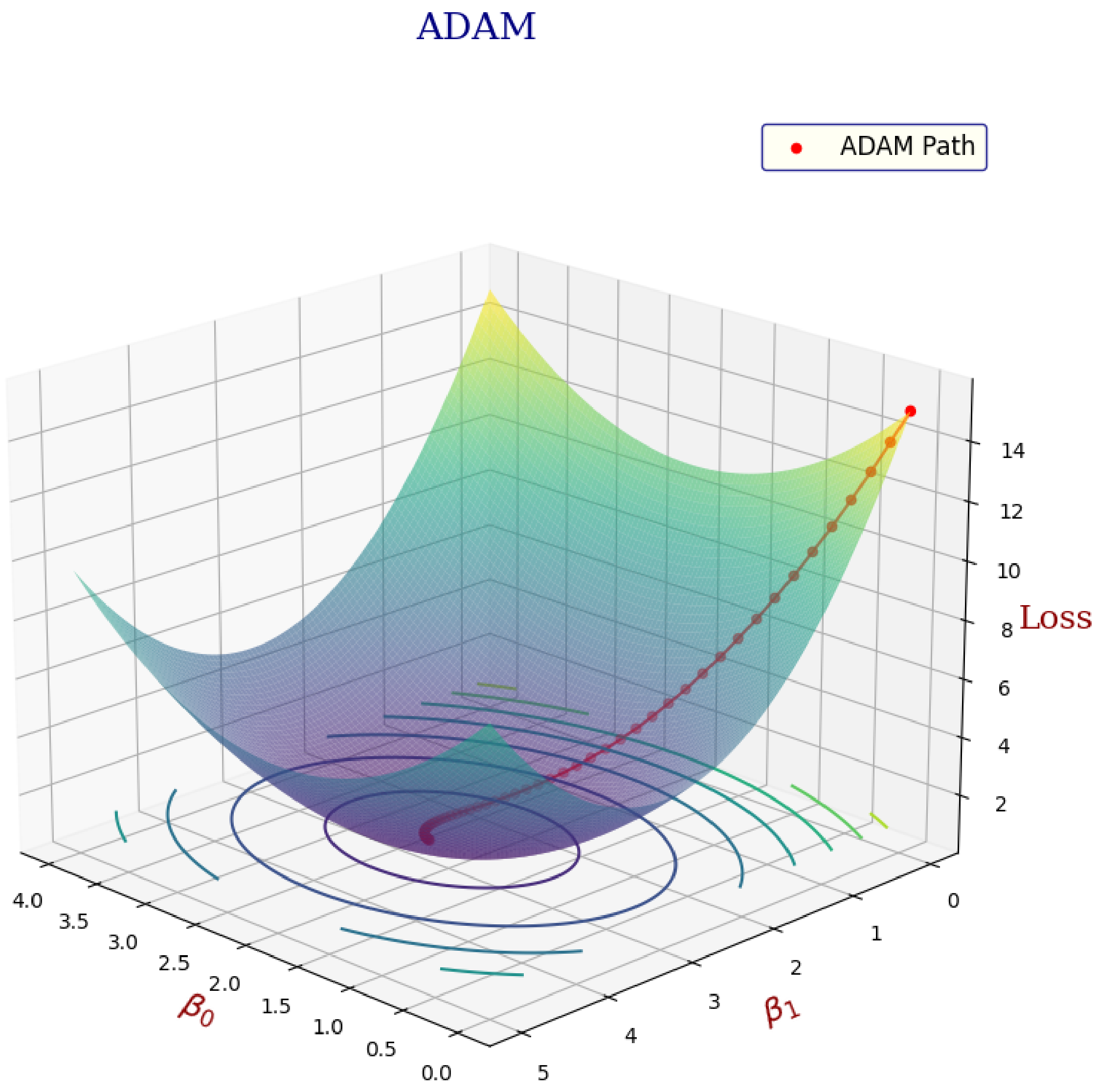
```
for i in range(B0.shape[0]):
    for j in range(B0.shape[1]):
        Loss[i, j] = loss(Y, x, [B0[i, j], B1[i, j]])


levels = np.linspace(np.min(Loss), np.max(Loss), 50)
plt.contour(B0, B1, Loss, levels=levels)
plt.plot(adam_path[:, 0], adam_path[:, 1], label='Adam', marker='.', color='
plt.legend()
plt.show()
```



In [ ]:

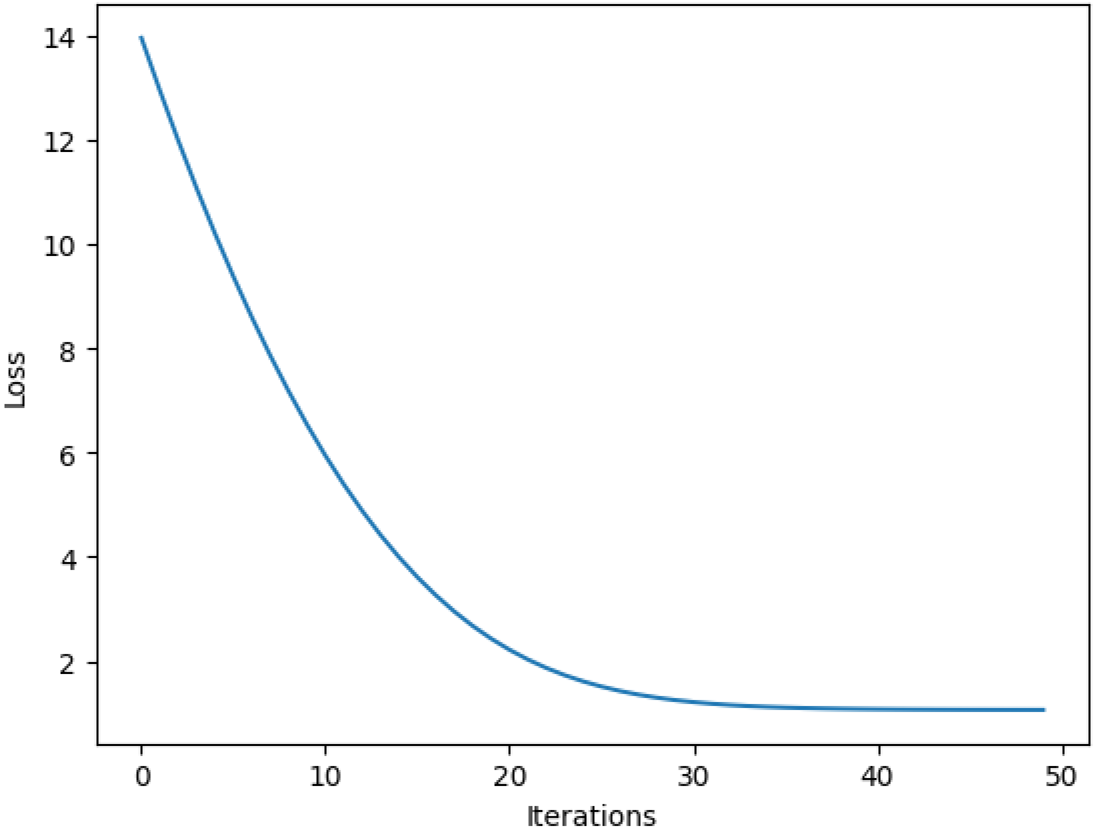
```
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

surf = ax.plot_surface(B0, B1, Loss, cmap='viridis', rstride=1, cstride=1, a
contour = ax.contour(B0, B1, Loss, zdir='z', offset=np.min(Loss), cmap='viri

ax.scatter(adam_path[:, 0], adam_path[:, 1], np.array([loss(Y, x, beta) for 
ax.plot(adam_path[:, 0], adam_path[:, 1], np.array([loss(Y, x, beta) for bet


font_dict = {'family': 'serif', 'color':  'darkred', 'weight': 'normal', 'si
ax.set_xlabel(r'$\beta_0$', fontdict=font_dict)
ax.set_ylabel(r'$\beta_1$', fontdict=font_dict)
ax.set_zlabel('Loss', fontdict=font_dict)
ax.set_title('ADAM', pad=35, fontsize=18, fontdict={'family': 'serif', 'colo

ax.legend(loc='upper right', fontsize=12, frameon=True, facecolor='ivory', e

ax.view_init(elev=20, azim=135)
```

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



In [ ]:

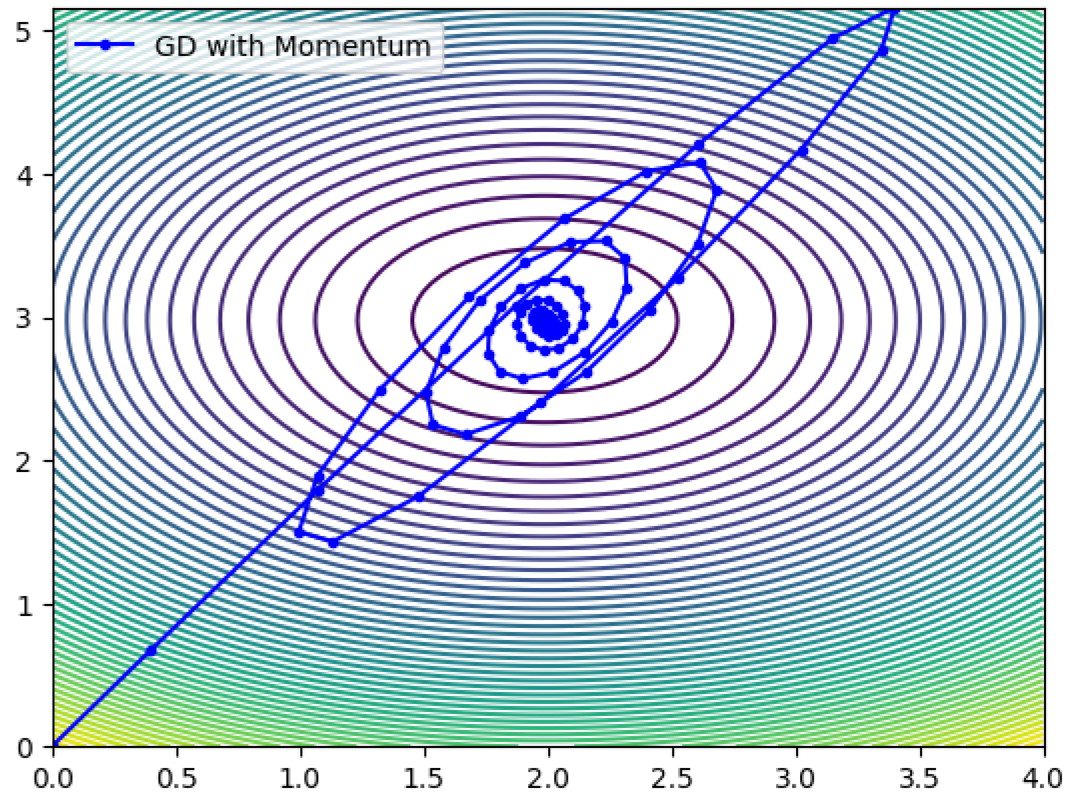
```
iterations = range(len(adam_loss))
plt.plot(iterations, adam_loss, '-')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.show()
```



In [ ]:

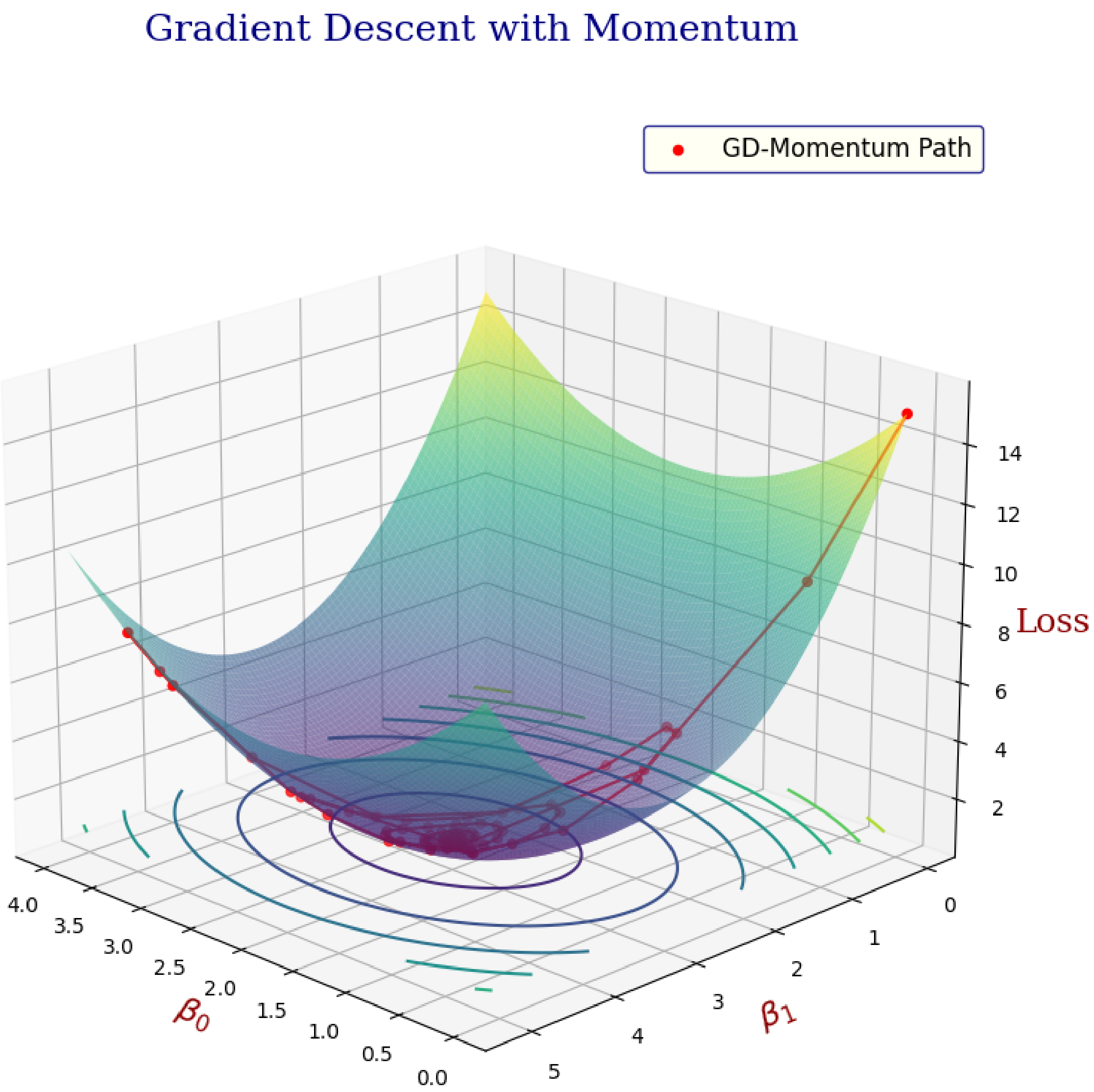
```
def GD_Momentum(x, Y, num_iterations, alpha, gamma=0.9):
    beta = np.array([0, 0])
    beta_values = [beta]
    loss_values = []
    v = np.array([0, 0])

    for _ in range(num_iterations):
        gradient = compute_gradient(Y, x, beta)
        v = gamma * v + alpha * gradient
        beta = beta - v
        beta_values.append(beta)
        loss_values.append(loss(Y, x, beta))
    return beta_values,loss_values

momentum_path,momentum_loss = GD_Momentum(x, Y, 100, 0.1)
momentum_path = np.array(momentum_path)

B0_range = [min(momentum_path[:, 0].min(), beta_true[0]-2), max(momentum_pat
B1_range = [min(momentum_path[:, 1].min(), beta_true[1]-2), max(momentum_pat
B0, B1 = np.meshgrid(np.linspace(B0_range[0], B0_range[1], 100),
                     np.linspace(B1_range[0], B1_range[1], 100))
Loss = np.zeros_like(B0)

for i in range(B0.shape[0]):
    for j in range(B0.shape[1]):
        Loss[i, j] = loss(Y, x, [B0[i, j], B1[i, j]])

levels = np.linspace(np.min(Loss), np.max(Loss), 50)
plt.contour(B0, B1, Loss, levels=levels)
```

```
plt.plot(momentum_path[:, 0], momentum_path[:, 1], label='GD with Momentum',
plt.legend()
plt.show()
```



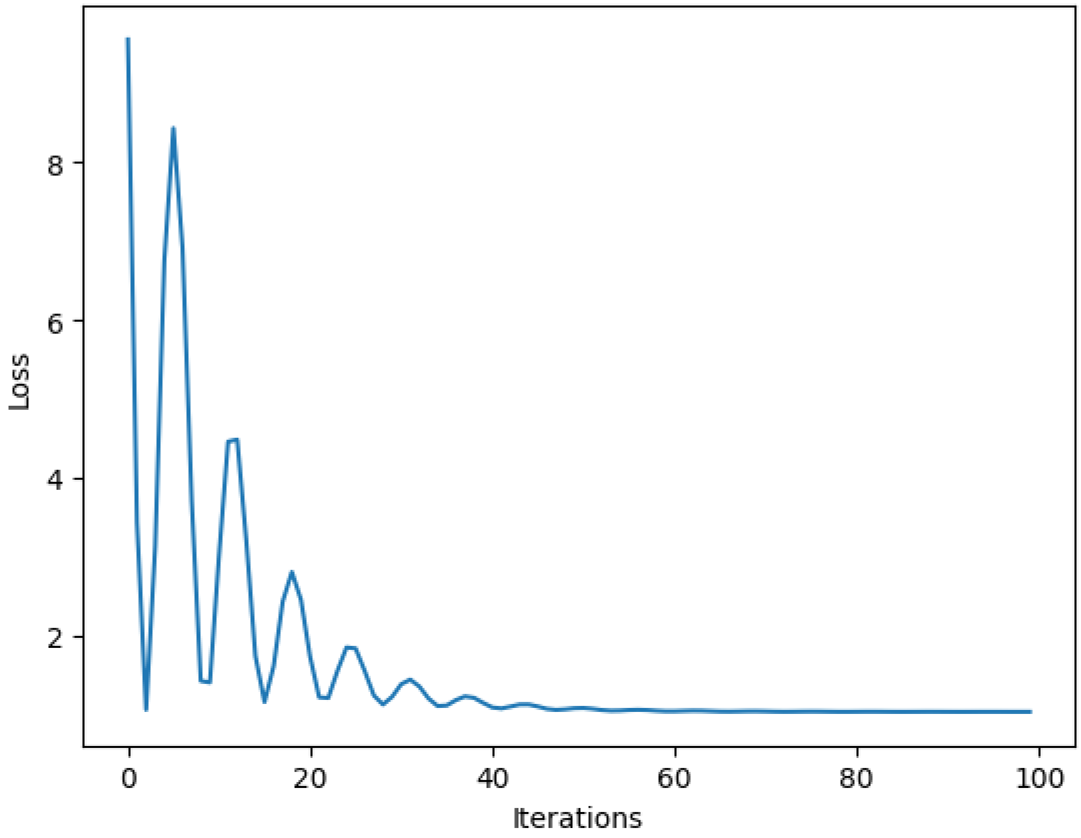
In [ ]:

```
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

surf = ax.plot_surface(B0, B1, Loss, cmap='viridis', rstride=1, cstride=1, a
contour = ax.contour(B0, B1, Loss, zdir='z', offset=np.min(Loss), cmap='viri

ax.scatter(momentum_path[:, 0], momentum_path[:, 1], np.array([loss(Y, x, be
ax.plot(momentum_path[:, 0], momentum_path[:, 1], np.array([loss(Y, x, beta)


font_dict = {'family': 'serif', 'color':  'darkred', 'weight': 'normal', 'si
ax.set_xlabel(r'$\beta_0$', fontdict=font_dict)
ax.set_ylabel(r'$\beta_1$', fontdict=font_dict)
ax.set_zlabel('Loss', fontdict=font_dict)
ax.set_title('Gradient Descent with Momentum', pad=35, fontsize=18, fontdict

ax.legend(loc='upper right', fontsize=12, frameon=True, facecolor='ivory', e

ax.view_init(elev=20, azim=135)

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



In [ ]:

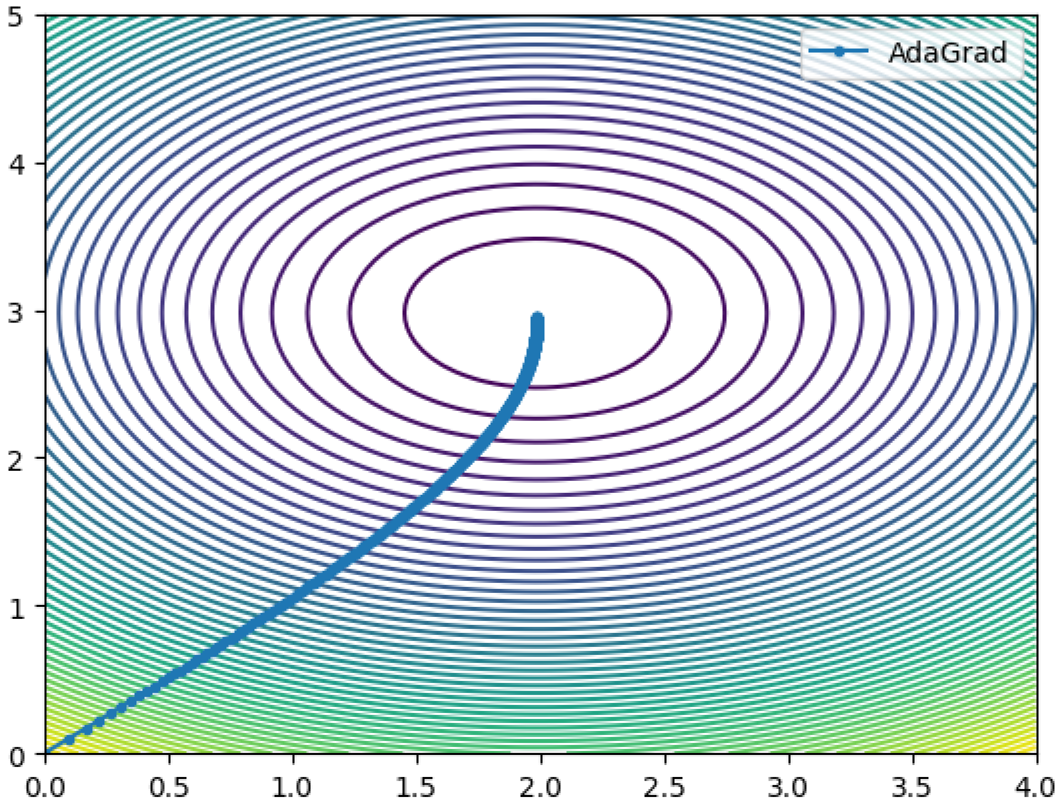
```
iterations = range(len(momentum_loss))
plt.plot(iterations, momentum_loss, '-')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.show()
```



In [ ]:

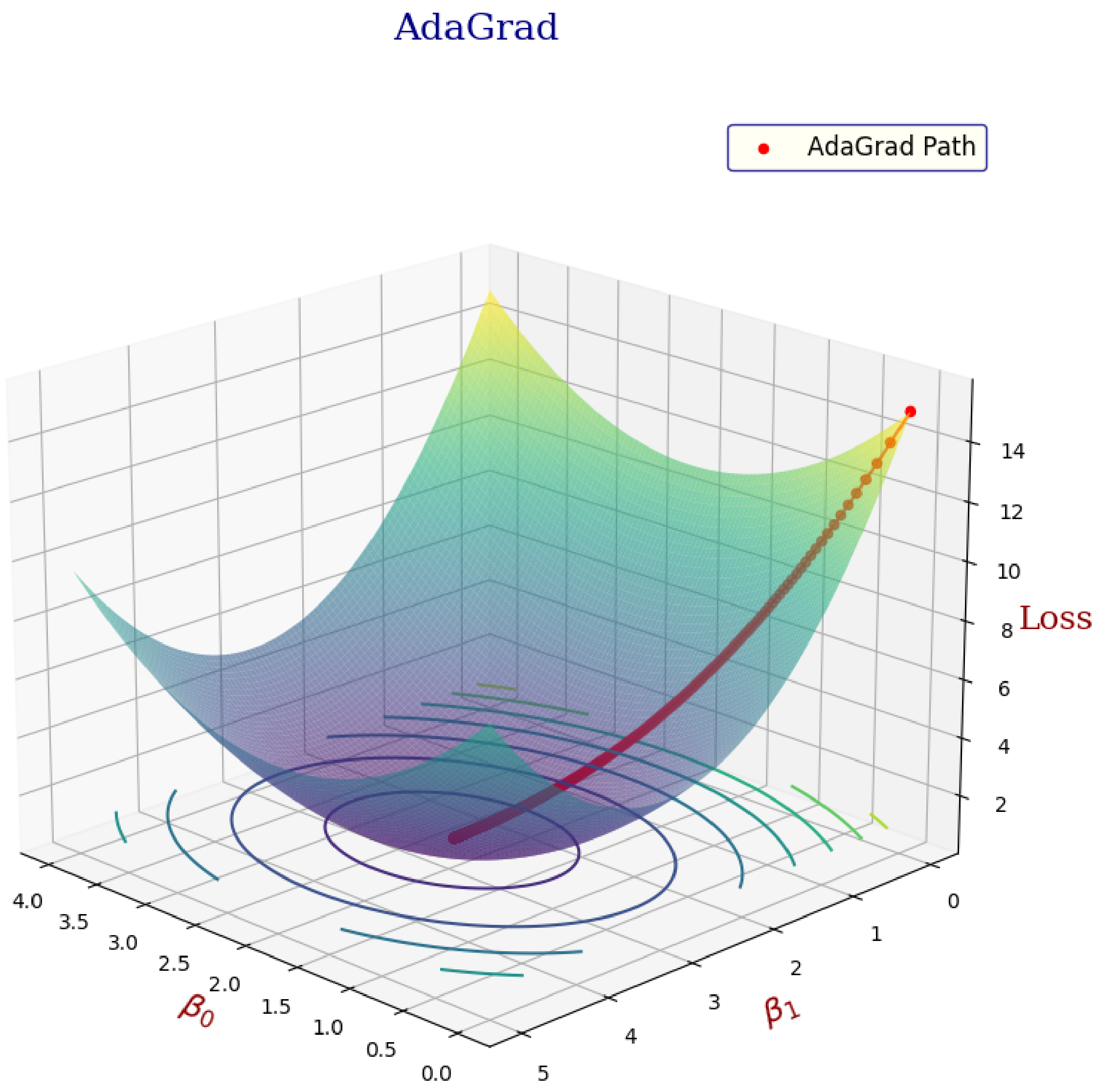
```
def AdaGrad(x, Y, num_iterations, alpha, epsilon=1e-8):
    beta = np.array([0.0, 0.0])
    beta_values = [beta]
    loss_values = []
    G = np.zeros_like(beta)

    for _ in range(num_iterations):
        gradient = compute_gradient(Y, x, beta)

        G += gradient ** 2

        adjusted_gradient = gradient / (np.sqrt(G+ epsilon))

        beta = beta - alpha * adjusted_gradient
        beta_values.append(beta)
        loss_values.append(loss(Y, x, beta))

    return beta_values,loss_values

x = np.load("synthetic_x.npy")
Y = np.load("synthetic_Y.npy")

beta_true = np.array([2, 3])

adagrad_path, adagrad_loss = AdaGrad(x, Y, 1000, 0.1)
adagrad_path = np.array(adagrad_path)

B0_range = [min(adagrad_path[:, 0].min(), beta_true[0]-2), max(adagrad_path[
B1_range = [min(adagrad_path[:, 1].min(), beta_true[1]-2), max(adagrad_path[
```

```
B0, B1 = np.meshgrid(np.linspace(B0_range[0], B0_range[1], 100),
                     np.linspace(B1_range[0], B1_range[1], 100))
Loss = np.zeros_like(B0)

for i in range(B0.shape[0]):
    for j in range(B0.shape[1]):
        Loss[i, j] = loss(Y, x, [B0[i, j], B1[i, j]])

levels = np.linspace(np.min(Loss), np.max(Loss), 50)
plt.contour(B0, B1, Loss, levels=levels)
plt.plot(adagrad_path[:, 0], adagrad_path[:, 1], label='AdaGrad', marker='.'
plt.legend()
plt.show()
```



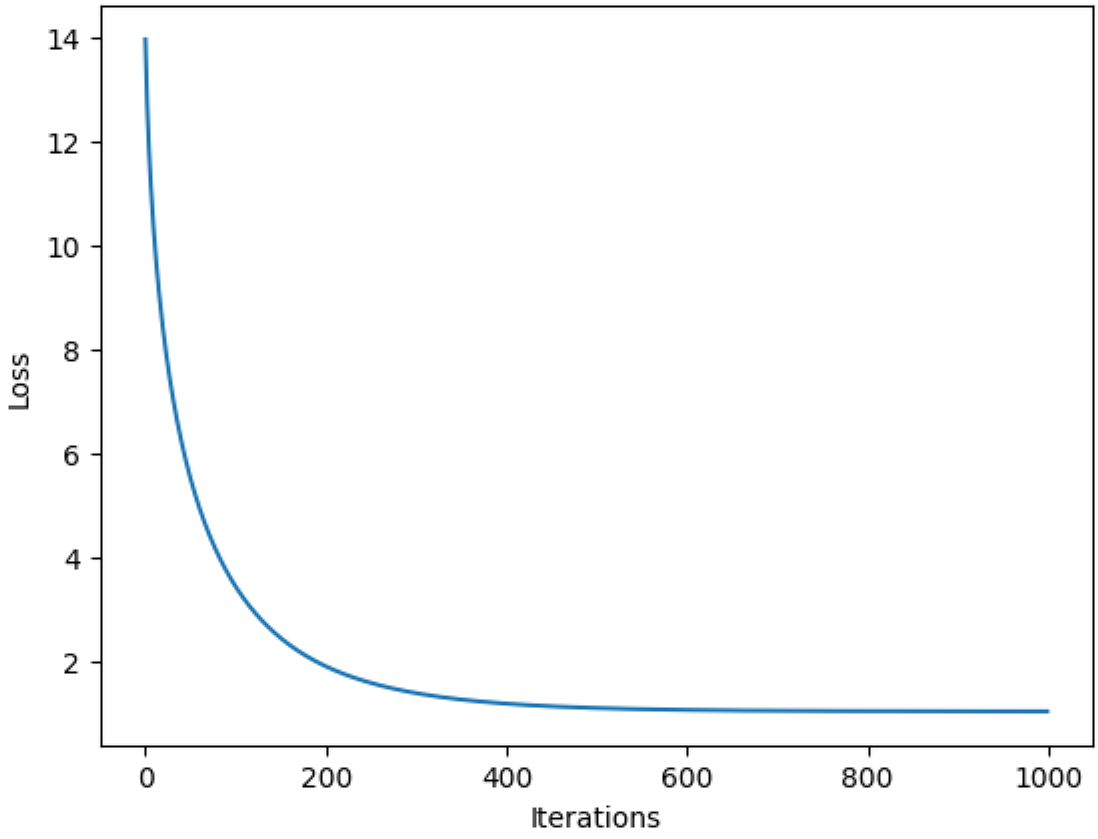
In [ ]:
```
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

surf = ax.plot_surface(B0, B1, Loss, cmap='viridis', rstride=1, cstride=1, a
contour = ax.contour(B0, B1, Loss, zdir='z', offset=np.min(Loss), cmap='viri

ax.scatter(adagrad_path[:, 0], adagrad_path[:, 1], np.array([loss(Y, x, beta
ax.plot(adagrad_path[:, 0], adagrad_path[:, 1], np.array([loss(Y, x, beta) f


font_dict = {'family': 'serif', 'color':  'darkred', 'weight': 'normal', 'si
ax.set_xlabel(r'$\beta_0$', fontdict=font_dict)
ax.set_ylabel(r'$\beta_1$', fontdict=font_dict)
ax.set_zlabel('Loss', fontdict=font_dict)
ax.set_title('AdaGrad', pad=35, fontsize=18, fontdict={'family': 'serif', 'c

ax.legend(loc='upper right', fontsize=12, frameon=True, facecolor='ivory', e
```

```
ax.view_init(elev=20, azim=135)

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



In [ ]:

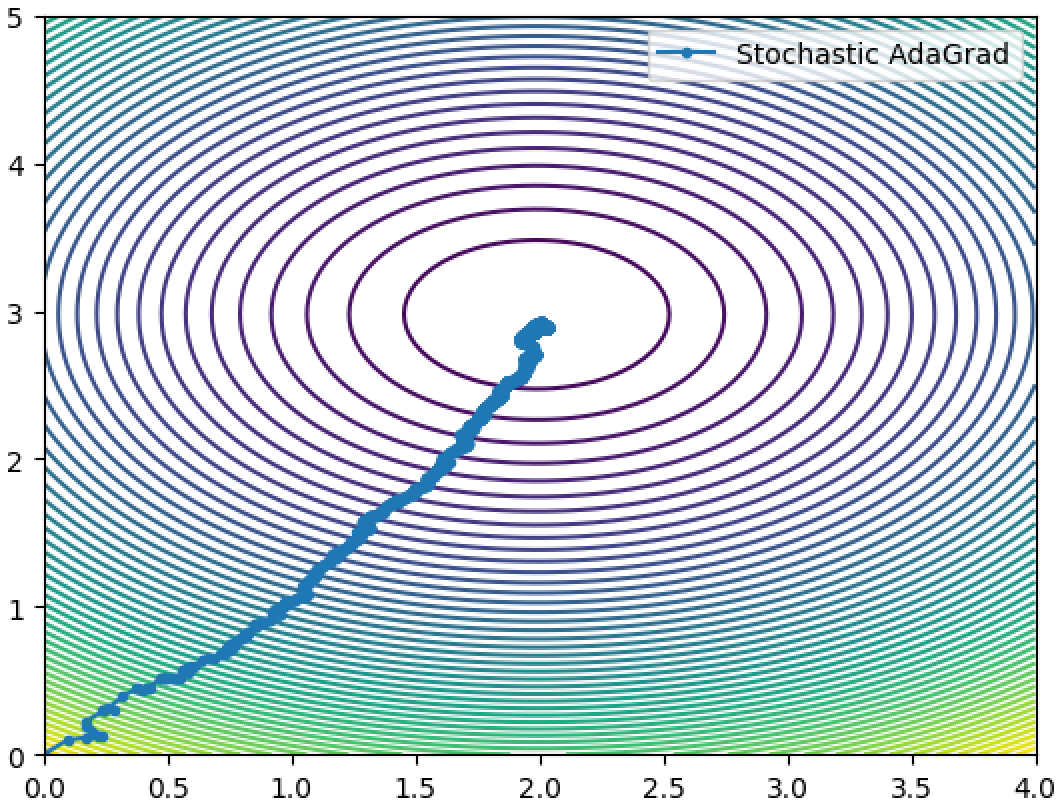
```
iterations = range(len(adagrad_loss))
plt.plot(iterations, adagrad_loss, '-')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.show()
```



In [ ]:

```python
def compute_stochastic_gradient(Y, x, beta):
    i = np.random.randint(0, len(Y))
    residuals = Y[i] - (beta[0] + beta[1] * x[i])
    d_beta0 = -2 * residuals
    d_beta1 = -2 * residuals * x[i]
    return np.array([d_beta0, d_beta1])

def S_AdaGrad(x, Y, num_iterations, alpha, epsilon=1e-8):
    beta = np.array([0.0, 0.0])
    beta_values = [beta]
    loss_values = []

    G = np.zeros_like(beta)

    for _ in range(num_iterations):
        gradient = compute_stochastic_gradient(Y, x, beta)

        G += gradient ** 2

        adjusted_gradient = gradient / (np.sqrt(G+ epsilon))

        beta = beta - alpha * adjusted_gradient
        beta_values.append(beta)
        loss_values.append(loss(Y, x, beta))

    return beta_values,loss_values

x = np.load("synthetic_x.npy")
Y = np.load("synthetic_Y.npy")
```

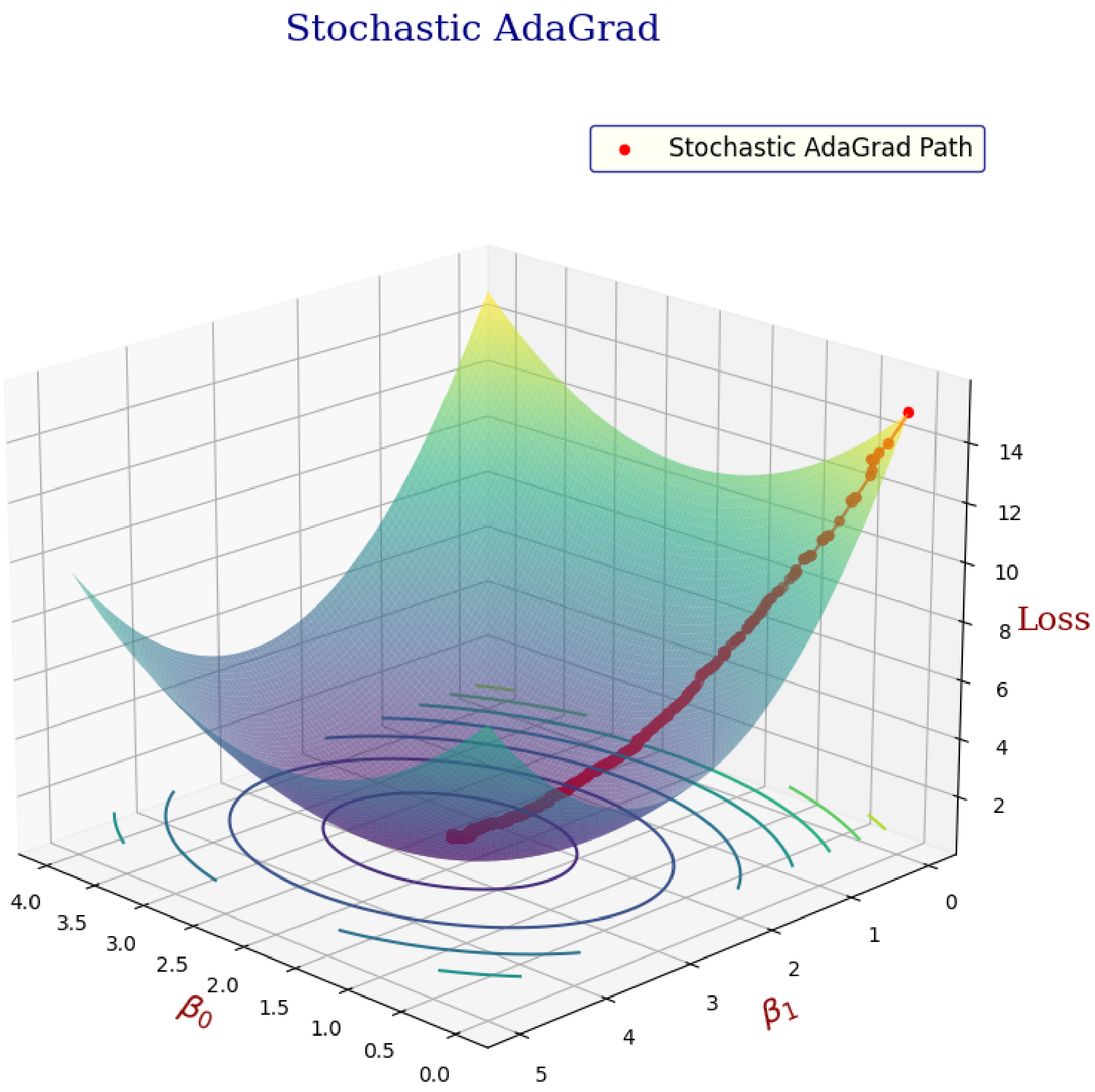
```
beta_true = np.array([2, 3])

s_adagrad_path, s_adagrad_loss = S_AdaGrad(x, Y, 3000, 0.1)
s_adagrad_path = np.array(s_adagrad_path)

B0_range = [min(s_adagrad_path[:, 0].min(), beta_true[0]-2), max(s_adagrad_p
B1_range = [min(s_adagrad_path[:, 1].min(), beta_true[1]-2), max(s_adagrad_p
B0, B1 = np.meshgrid(np.linspace(B0_range[0], B0_range[1], 100),
                     np.linspace(B1_range[0], B1_range[1], 100))
Loss = np.zeros_like(B0)

for i in range(B0.shape[0]):
    for j in range(B0.shape[1]):
        Loss[i, j] = loss(Y, x, [B0[i, j], B1[i, j]])

levels = np.linspace(np.min(Loss), np.max(Loss), 50)
plt.contour(B0, B1, Loss, levels=levels)
plt.plot(s_adagrad_path[:, 0], s_adagrad_path[:, 1], label='Stochastic AdaGr
plt.legend()
plt.show()
```



In [ ]:

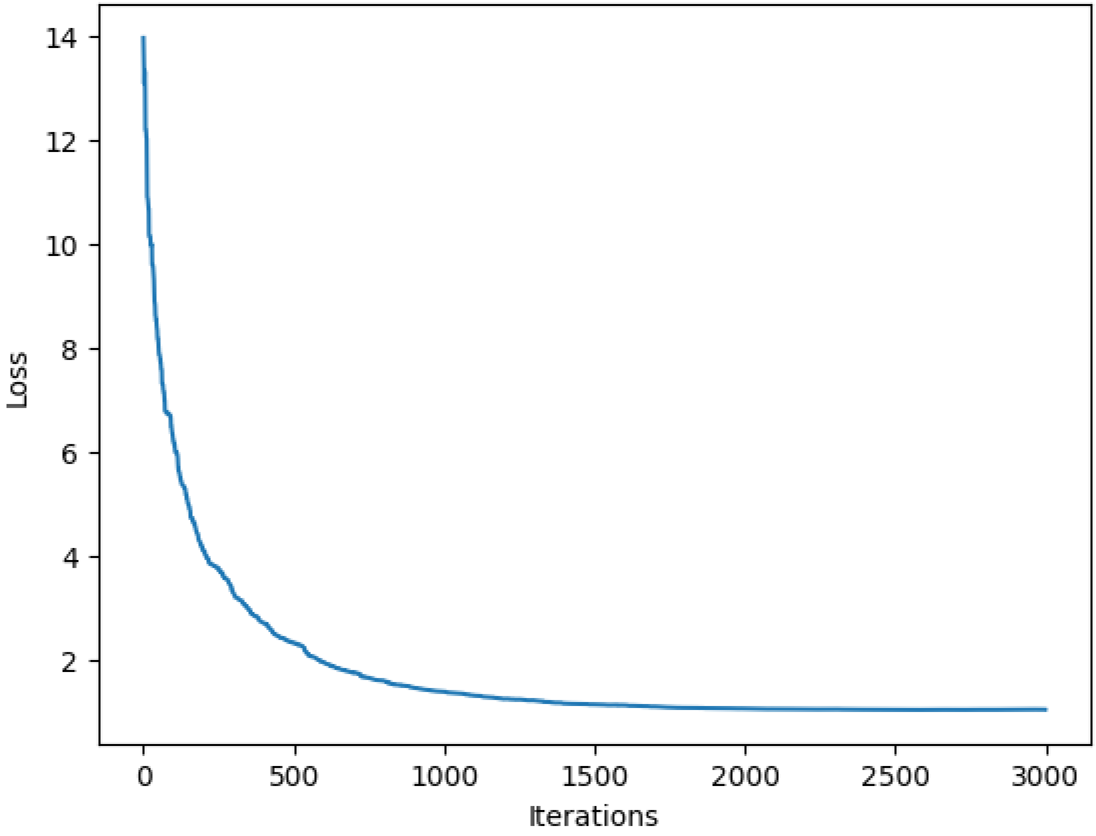
```
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

surf = ax.plot_surface(B0, B1, Loss, cmap='viridis', rstride=1, cstride=1, a
contour = ax.contour(B0, B1, Loss, zdir='z', offset=np.min(Loss), cmap='viri

ax.scatter(s_adagrad_path[:, 0], s_adagrad_path[:, 1], np.array([loss(Y, x,
ax.plot(s_adagrad_path[:, 0], s_adagrad_path[:, 1], np.array([loss(Y, x, bet
```

```
font_dict = {'family': 'serif', 'color':  'darkred', 'weight': 'normal', 'si
ax.set_xlabel(r'$\beta_0$', fontdict=font_dict)
ax.set_ylabel(r'$\beta_1$', fontdict=font_dict)
ax.set_zlabel('Loss', fontdict=font_dict)
ax.set_title('Stochastic AdaGrad', pad=35, fontsize=18, fontdict={'family':

ax.legend(loc='upper right', fontsize=12, frameon=True, facecolor='ivory', e

ax.view_init(elev=20, azim=135)

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



In [ ]:

```
iterations = range(len(s_adagrad_loss))
plt.plot(iterations, s_adagrad_loss, '-')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.show()
```



Saddle Point and Local minimum problems

Given function

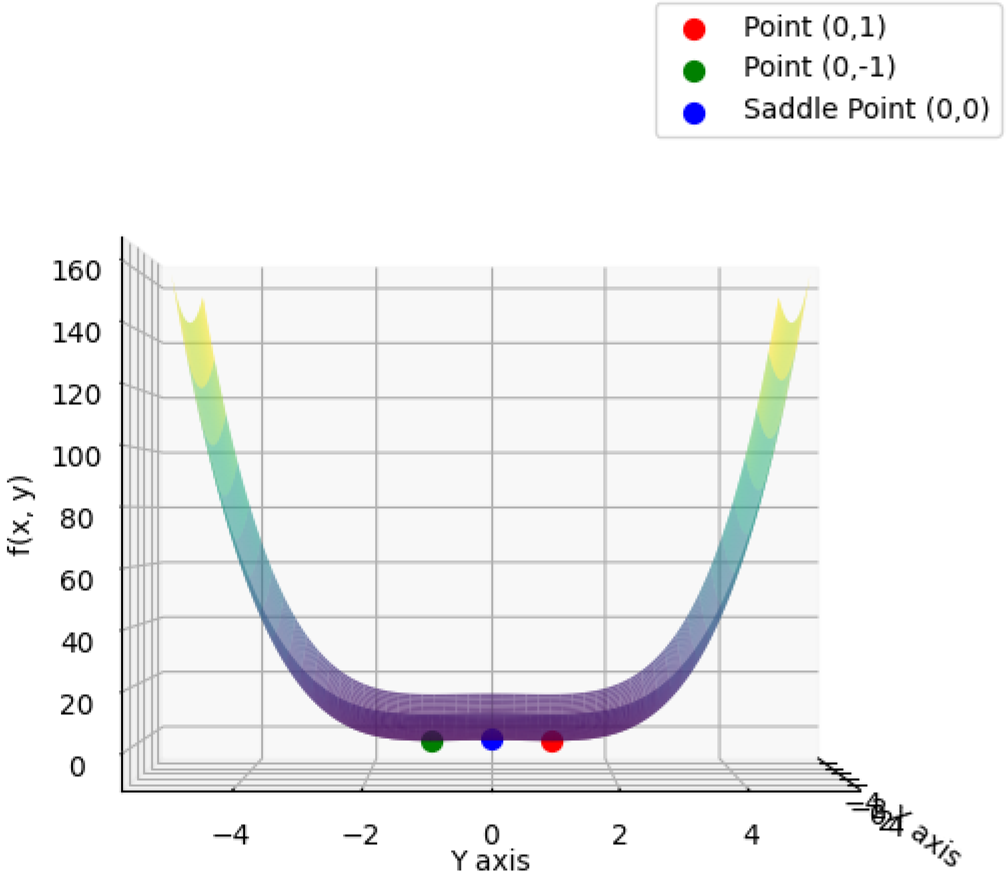
$$f(x,y) = \frac{1}{2}x^2 + \frac{1}{4}y^4 - \frac{1}{2}y^2$$

and

$$\nabla f(x,y) = \begin{bmatrix} x \\ y^3 - y \end{bmatrix}$$

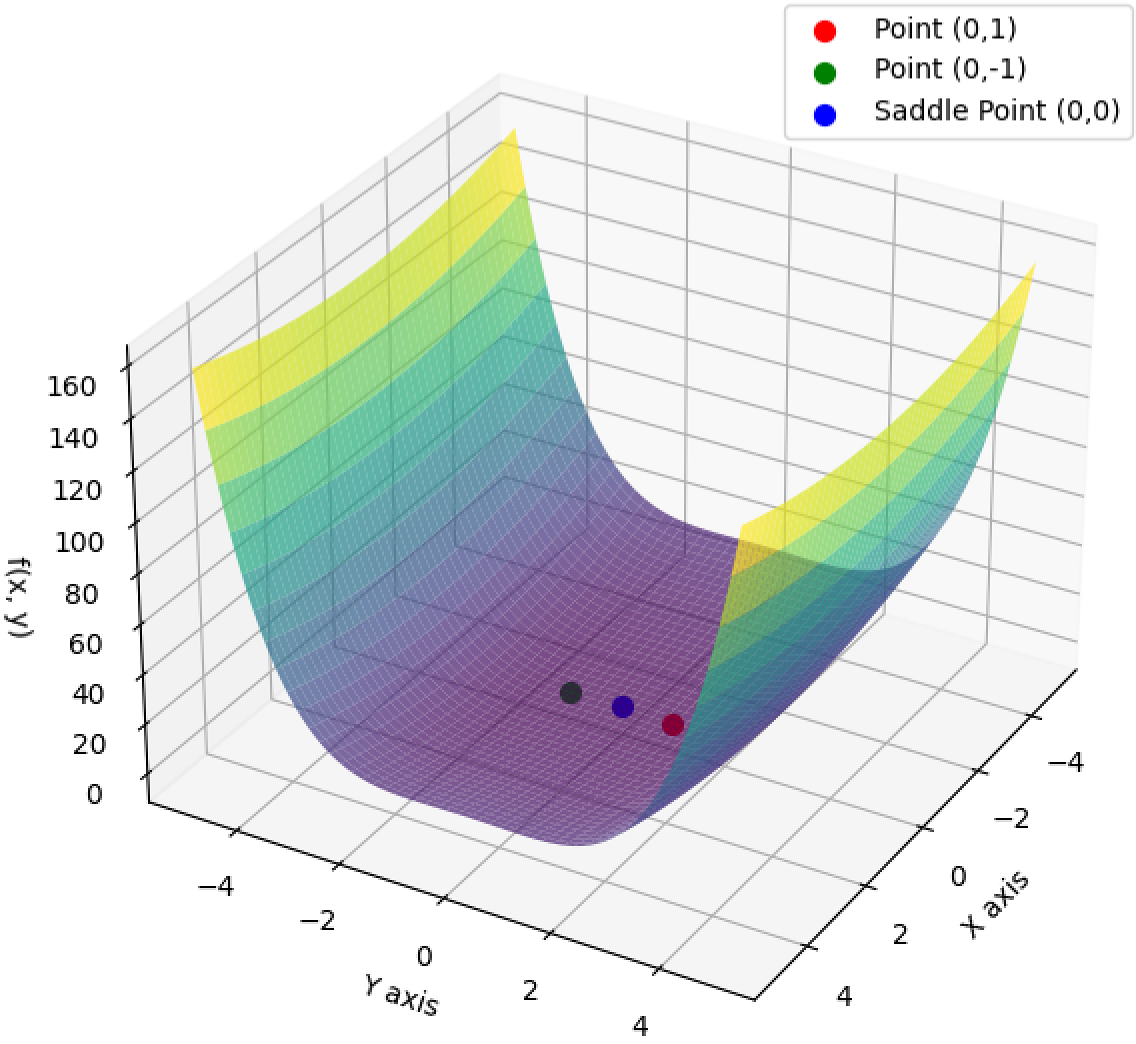
In [ ]:
```
def f(x, y):
    return 0.5 * x**2 + 0.25 * y**4 - 0.5 * y**2

x = np.linspace(-5, 5, 400)
y = np.linspace(-5, 5, 400)
x, y = np.meshgrid(x, y)

z = f(x, y)

# Coordinates for the given points
point_1 = np.array([0, 1])  # (0, 1)
point_2 = np.array([0, -1]) # (0, -1)
saddle_point = np.array([0, 0]) # (0, 0)
```

```
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')

ax.plot_surface(x, y, z, cmap='viridis', alpha=0.7)

ax.scatter(*point_1, f(*point_1), color='red', s=50, label='Point (0,1)')
ax.scatter(*point_2, f(*point_2), color='green', s=50, label='Point (0,-1)')
ax.scatter(*saddle_point, f(*saddle_point), color='blue', s=50, label='Saddl

ax.set_xlabel('X axis')
ax.set_ylabel('Y axis')
ax.set_zlabel('f(x, y)')

ax.view_init(elev=0, azim=0)

# Add legend
plt.legend()

# Show the plot
plt.show()
```



In [ ]:

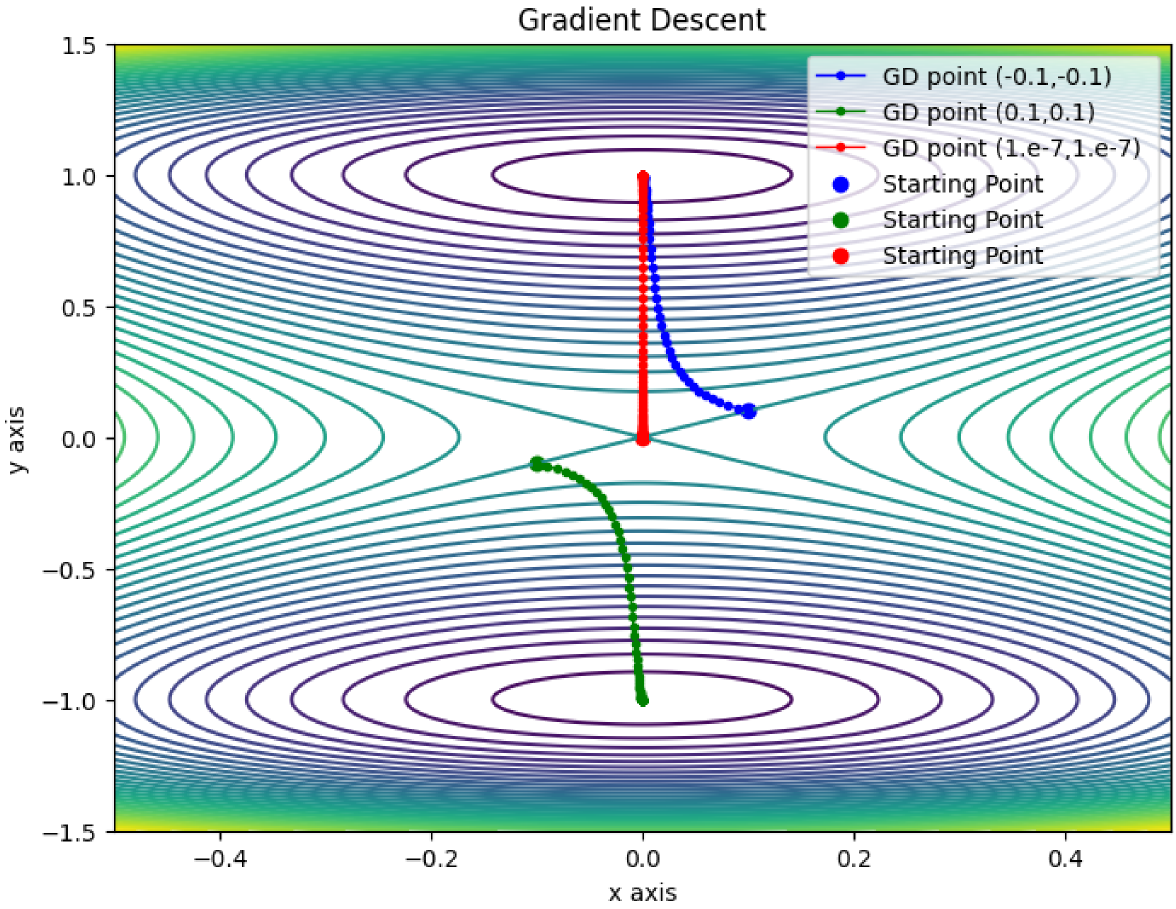
```
x = np.linspace(-5, 5, 400)
y = np.linspace(-5, 5, 400)
x, y = np.meshgrid(x, y)

z = f(x, y)

point_1 = np.array([0, 1])  # (0, 1)
point_2 = np.array([0, -1]) # (0, -1)
saddle_point = np.array([0, 0]) # (0, 0)


fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')

ax.plot_surface(x, y, z, cmap='viridis', alpha=0.7)

ax.scatter(*point_1, f(*point_1), color='red', s=50, label='Point (0,1)')
ax.scatter(*point_2, f(*point_2), color='green', s=50, label='Point (0,-1)')
ax.scatter(*saddle_point, f(*saddle_point), color='blue', s=50, label='Saddl

ax.set_xlabel('X axis')
ax.set_ylabel('Y axis')
ax.set_zlabel('f(x, y)')

ax.view_init(elev=30, azim=30)

plt.legend()

plt.show()
```



In [ ]:

```
def gradient_f(x, y):
    """Gradient of the function f."""
    df_dx = x
    df_dy = y**3 - y
    return np.array([df_dx, df_dy])

def gradient_descent(starting_point, learning_rate, iterations):
    point = np.array(starting_point)
    trajectory = [point.copy()]
    for _ in range(iterations):
        grad = gradient_f(*point)
        point -= learning_rate * grad
        trajectory.append(point.copy())
    return np.array(trajectory)


learning_rate = 0.1
iterations = 1000

point_1 = (0.1,0.1)
point_2 = (-0.1,-0.1)
point_3 = (1.e-7,1.e-7)
```

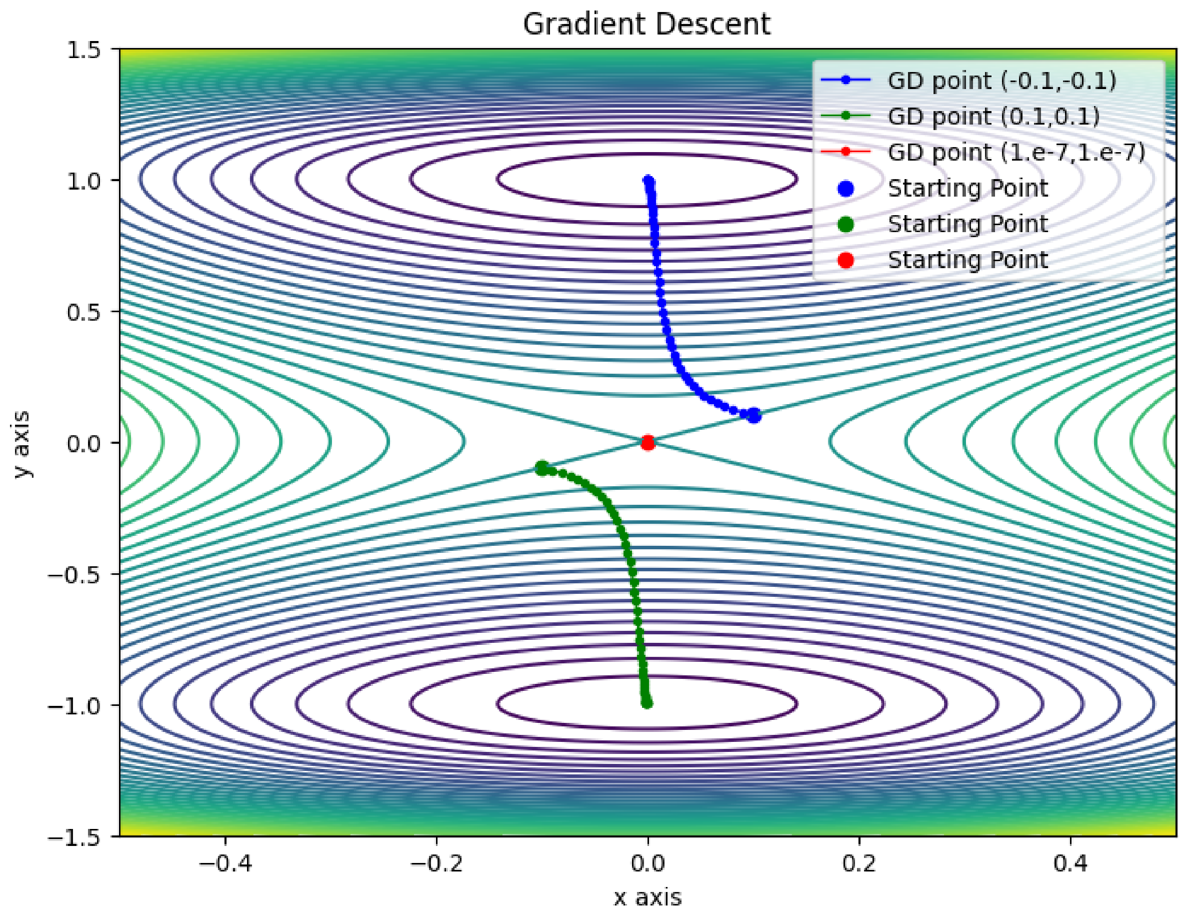
```
# Create a contour plot of the function f
x = np.linspace(-0.5, 0.5, 400)
y = np.linspace(-1.5, 1.5, 400)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)


First = gradient_descent(point_1, learning_rate, iterations)
Second = gradient_descent(point_2, learning_rate, iterations)
Third = gradient_descent(point_3, learning_rate, iterations)

plt.figure(figsize=(8, 6))
plt.contour(X, Y, Z, levels=50, cmap='viridis')

plt.plot(First[:, 0], First[:, 1], marker='o', color='blue', markersize=3, l
plt.plot(Second[:, 0], Second[:, 1], marker='o', color='green', markersize=3
plt.plot(Third[:, 0], Third[:, 1], marker='o', color='red', markersize=3, li


plt.scatter(*point_1, color='blue', label='Starting Point')
plt.scatter(*point_2, color='green', label='Starting Point')
plt.scatter(*point_3, color='red', label='Starting Point')


plt.xlabel('x axis')
plt.ylabel('y axis')
plt.title('Gradient Descent')
plt.legend()
plt.show()
```



In [ ]:

```
learning_rate = 0.1
iterations = 50

point_1 = (0.1,0.1)
point_2 = (-0.1,-0.1)
point_3 = (1.e-7,1.e-7)

# Create a contour plot of the function f
x = np.linspace(-0.5, 0.5, 400)
y = np.linspace(-1.5, 1.5, 400)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)


First = gradient_descent(point_1, learning_rate, iterations)
Second = gradient_descent(point_2, learning_rate, iterations)
Third = gradient_descent(point_3, learning_rate, iterations)

plt.figure(figsize=(8, 6))
plt.contour(X, Y, Z, levels=50, cmap='viridis')

plt.plot(First[:, 0], First[:, 1], marker='o', color='blue', markersize=3, l
plt.plot(Second[:, 0], Second[:, 1], marker='o', color='green', markersize=3
plt.plot(Third[:, 0], Third[:, 1], marker='o', color='red', markersize=3, li


plt.scatter(*point_1, color='blue', label='Starting Point')
```

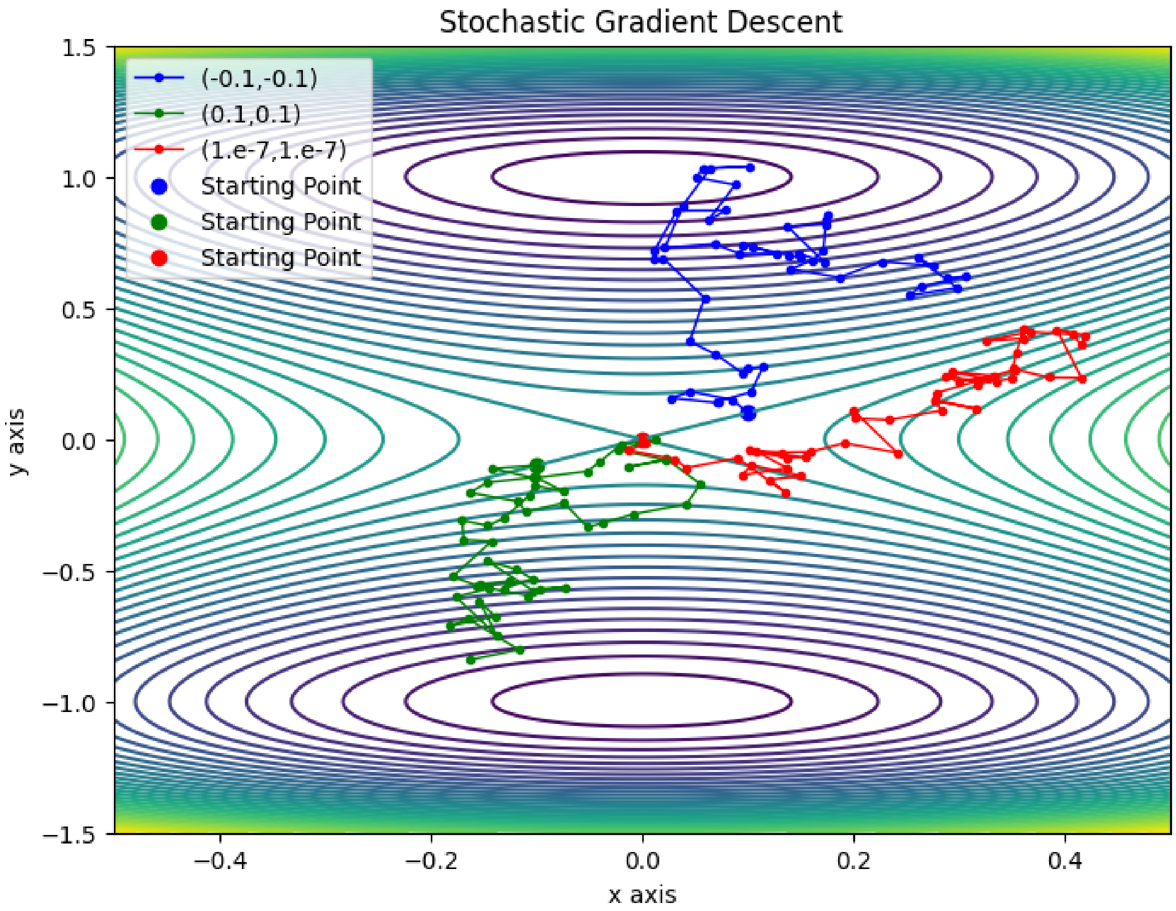
```
plt.scatter(*point_2, color='green', label='Starting Point')
plt.scatter(*point_3, color='red', label='Starting Point')


plt.xlabel('x axis')
plt.ylabel('y axis')
plt.title('Gradient Descent')
plt.legend()
plt.show()
```



In [ ]:

```
def stochastic_gradient_descent(starting_point, learning_rate, iterations, c
    point = np.array(starting_point)
    trajectory = [point.copy()]

    for _ in range(iterations):
        # Randomly select one data point (here simulated by choosing a rando
        random_index = np.random.randint(0, len(data))
        selected_data = data[random_index]

        # Compute the gradient at the selected data point
        grad = gradient_f(*selected_data)

        # Update the point
        point -= learning_rate * grad
        trajectory.append(point.copy())

    return np.array(trajectory)
```

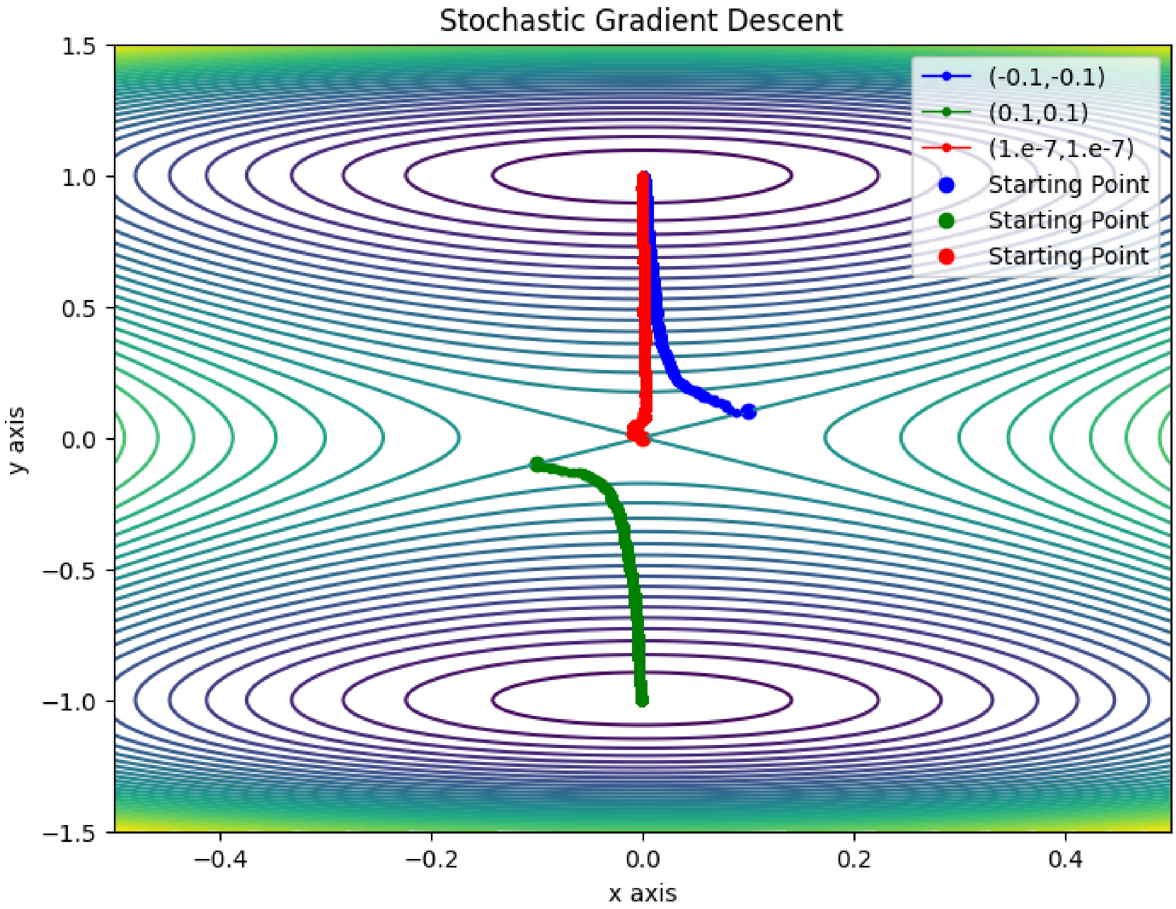
```
# Create a contour plot of the function f
x = np.linspace(-0.5, 0.5, 400)
y = np.linspace(-1.5, 1.5, 400)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)

data = np.c_[X.ravel(), Y.ravel()]

learning_rate = 0.1
iterations = 50

First = stochastic_gradient_descent(point_1, learning_rate, iterations,data)
Second = stochastic_gradient_descent(point_2, learning_rate, iterations,data
Third = stochastic_gradient_descent(point_3, learning_rate, iterations,data)

plt.figure(figsize=(8, 6))
plt.contour(X, Y, Z, levels=50, cmap='viridis')

plt.plot(First[:, 0], First[:, 1], marker='o', color='blue', markersize=3, l
plt.plot(Second[:, 0], Second[:, 1], marker='o', color='green', markersize=3
plt.plot(Third[:, 0], Third[:, 1], marker='o', color='red', markersize=3, li


plt.scatter(*point_1, color='blue', label='Starting Point')
plt.scatter(*point_2, color='green', label='Starting Point')
plt.scatter(*point_3, color='red', label='Starting Point')


plt.xlabel('x axis')
plt.ylabel('y axis')
plt.title('Stochastic Gradient Descent')
plt.legend()
plt.show()
```



In [ ]:

```
def s_gradient_f(x, y):
    i = np.random.normal(0, 0.1, 2)
    df_dx = x + i[0]
    df_dy = y**3 - y + i[1]
    return np.array([df_dx, df_dy])


def stochastic_gradient_descent(starting_point, learning_rate, iterations):
    point = np.array(starting_point)
    trajectory = [point.copy()]
    count = 1
    for _ in range(iterations):
        grad = s_gradient_f(*point)
        point -= learning_rate/count**(3/4) * (grad)
        trajectory.append(point.copy())
        count = count + 1
    return np.array(trajectory)

learning_rate = 0.1
iterations = 100000

First = stochastic_gradient_descent(point_1, learning_rate, iterations)
Second = stochastic_gradient_descent(point_2, learning_rate, iterations)
Third = stochastic_gradient_descent(point_3, learning_rate, iterations)

plt.figure(figsize=(8, 6))
plt.contour(X, Y, Z, levels=50, cmap='viridis')
```

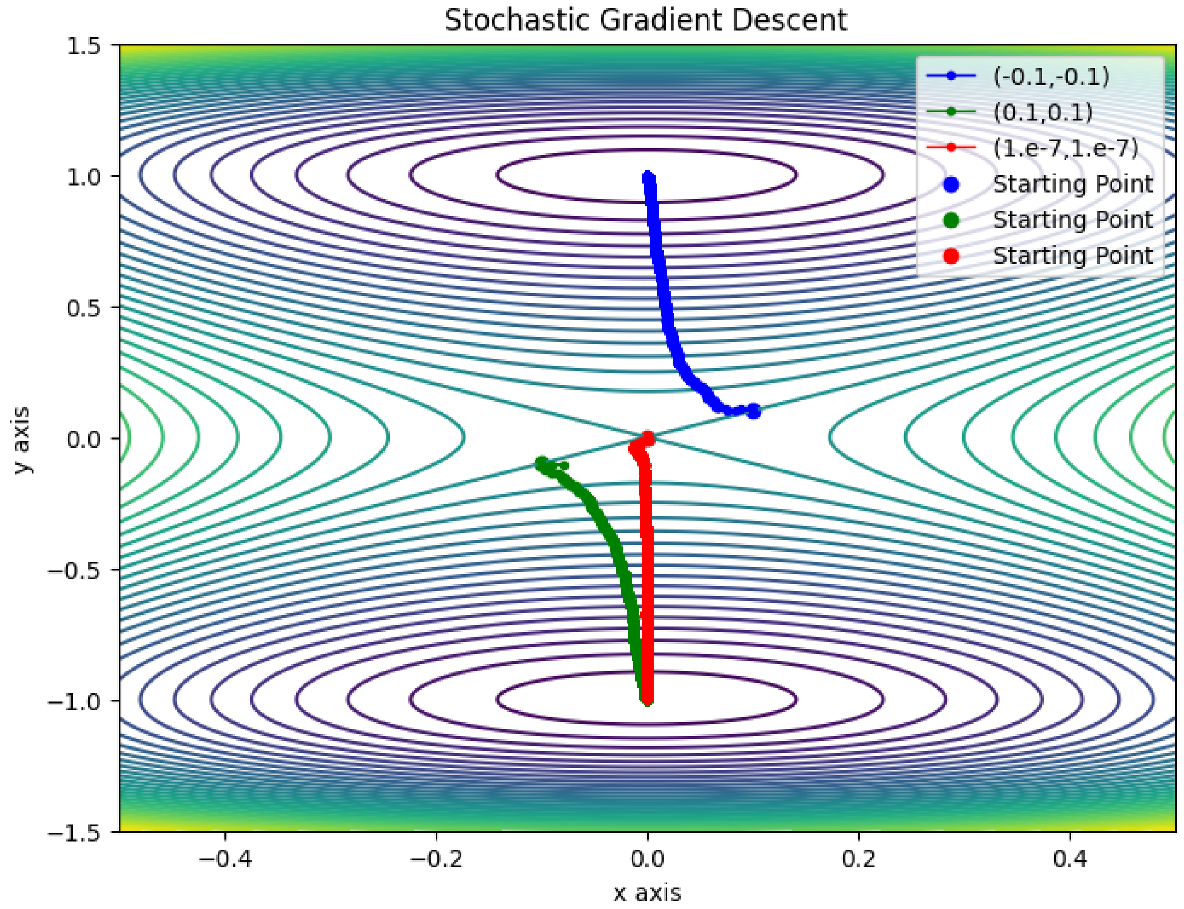
```
plt.plot(First[:, 0], First[:, 1], marker='o', color='blue', markersize=3, l
plt.plot(Second[:, 0], Second[:, 1], marker='o', color='green', markersize=3
plt.plot(Third[:, 0], Third[:, 1], marker='o', color='red', markersize=3, li


plt.scatter(*point_1, color='blue', label='Starting Point')
plt.scatter(*point_2, color='green', label='Starting Point')
plt.scatter(*point_3, color='red', label='Starting Point')


plt.xlabel('x axis')
plt.ylabel('y axis')
plt.title('Stochastic Gradient Descent')
plt.legend()
plt.show()
```



In [ ]:

```
def s_gradient_f(x, y):
    i = np.random.normal(0, 0.1, 2)
    df_dx = x + i[0]
    df_dy = y**3 - y + i[1]
    return np.array([df_dx, df_dy])


def stochastic_gradient_descent(starting_point, learning_rate, iterations):
    point = np.array(starting_point)
    trajectory = [point.copy()]
    count = 1
```

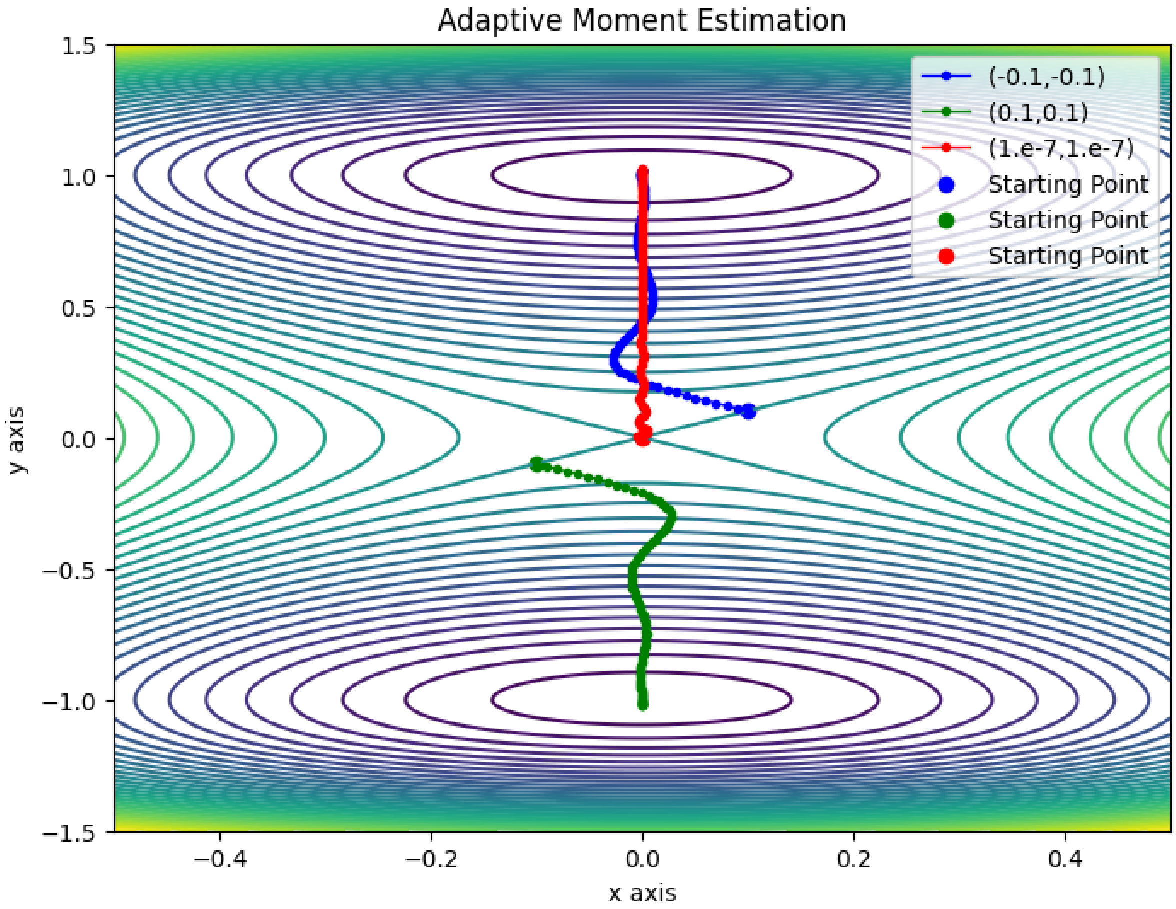
```
    for _ in range(iterations):
        grad = s_gradient_f(*point)
        point -= learning_rate/count**(3/4) * (grad)
        trajectory.append(point.copy())
        count = count + 1
    return np.array(trajectory)

learning_rate = 0.1
iterations = 100000

First = stochastic_gradient_descent(point_1, learning_rate, iterations)
Second = stochastic_gradient_descent(point_2, learning_rate, iterations)
Third = stochastic_gradient_descent(point_3, learning_rate, iterations)

plt.figure(figsize=(8, 6))
plt.contour(X, Y, Z, levels=50, cmap='viridis')

plt.plot(First[:, 0], First[:, 1], marker='o', color='blue', markersize=3, l
plt.plot(Second[:, 0], Second[:, 1], marker='o', color='green', markersize=3
plt.plot(Third[:, 0], Third[:, 1], marker='o', color='red', markersize=3, li


plt.scatter(*point_1, color='blue', label='Starting Point')
plt.scatter(*point_2, color='green', label='Starting Point')
plt.scatter(*point_3, color='red', label='Starting Point')


plt.xlabel('x axis')
plt.ylabel('y axis')
plt.title('Stochastic Gradient Descent')
plt.legend()
plt.show()
```



In [ ]:

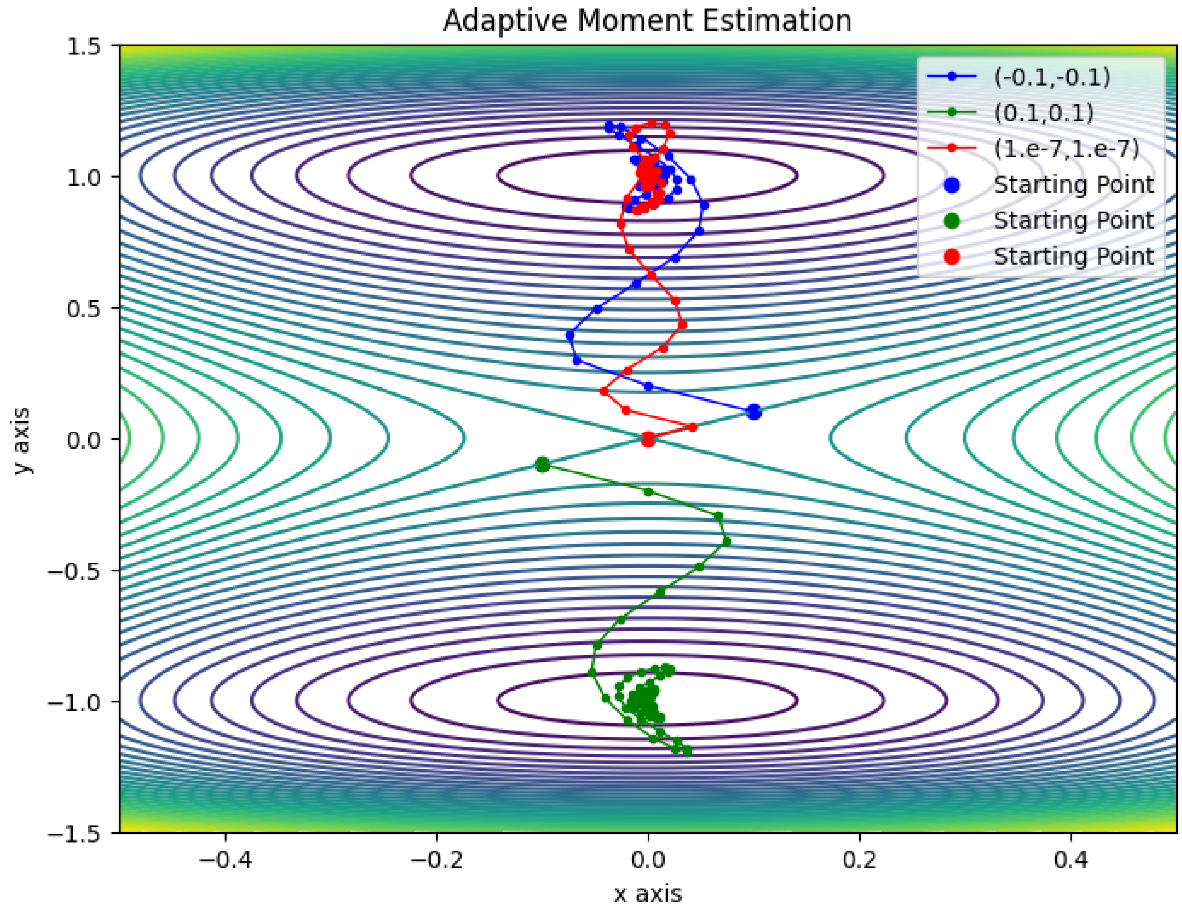
```
def adam(starting_point, learning_rate, iterations, beta1=0.9, beta2=0.999,
    point = np.array(starting_point)
    trajectory = [point.copy()]
    m, v = np.zeros_like(point), np.zeros_like(point)

    for t in range(1, iterations + 1):
        grad = gradient_f(*point)
        m = beta1 * m + (1 - beta1) * grad
        v = beta2 * v + (1 - beta2) * np.square(grad)

        m_hat = m / (1 - beta1 ** t)
        v_hat = v / (1 - beta2 ** t)

        point -= learning_rate * m_hat / (np.sqrt(v_hat+epsilon))
        trajectory.append(point.copy())

    return np.array(trajectory)


learning_rate_adam = 0.01
iterations_adam = 100

First = adam(point_1, learning_rate_adam, iterations_adam)
Second = adam(point_2, learning_rate_adam, iterations_adam)
Third = adam(point_3, learning_rate_adam, iterations_adam)

plt.figure(figsize=(8, 6))
```

```
plt.contour(X, Y, Z, levels=50, cmap='viridis')

plt.plot(First[:, 0], First[:, 1], marker='o', color='blue', markersize=3, l
plt.plot(Second[:, 0], Second[:, 1], marker='o', color='green', markersize=3
plt.plot(Third[:, 0], Third[:, 1], marker='o', color='red', markersize=3, li


plt.scatter(*point_1, color='blue', label='Starting Point')
plt.scatter(*point_2, color='green', label='Starting Point')
plt.scatter(*point_3, color='red', label='Starting Point')


plt.xlabel('x axis')
plt.ylabel('y axis')
plt.title('Adaptive Moment Estimation')
plt.legend()
plt.show()
```



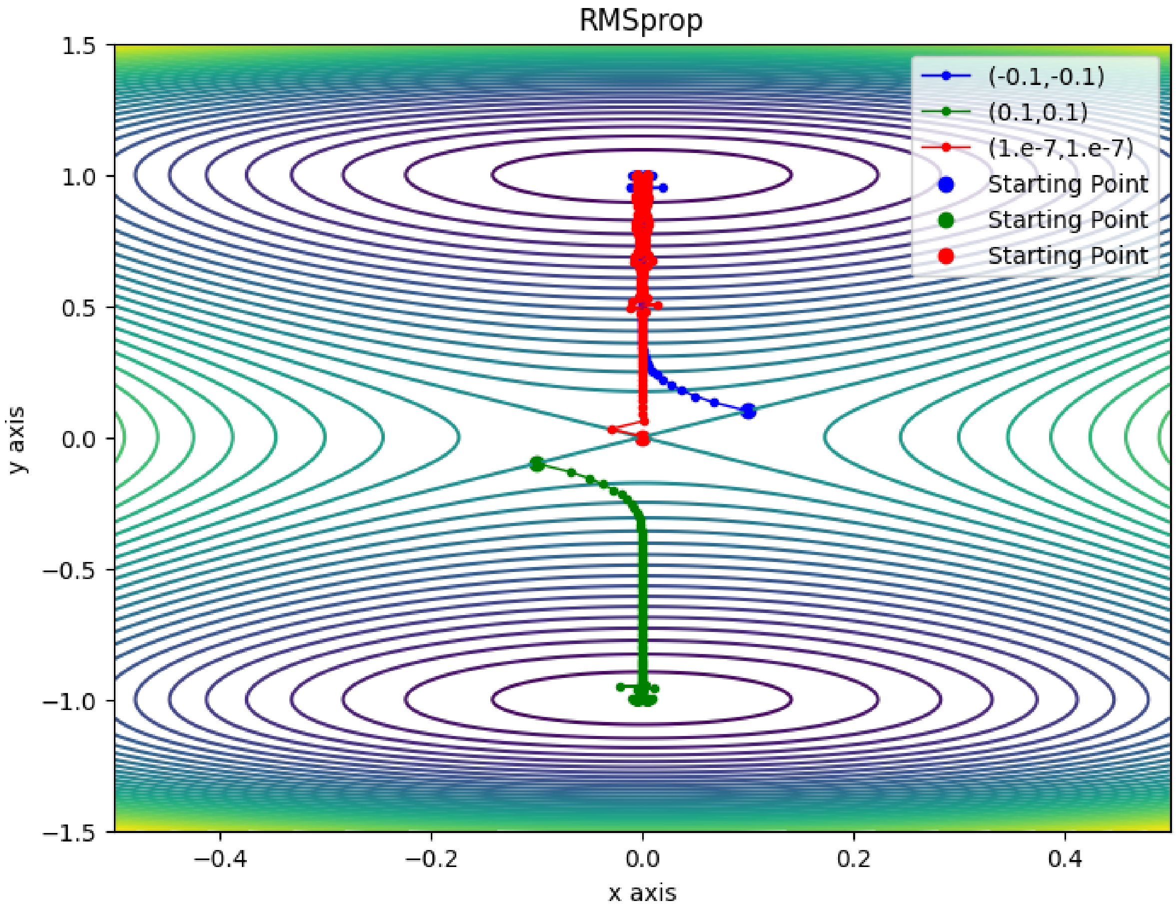
In [ ]:
```
learning_rate_adam = 0.1
iterations_adam = 100

First = adam(point_1, learning_rate_adam, iterations_adam)
Second = adam(point_2, learning_rate_adam, iterations_adam)
Third = adam(point_3, learning_rate_adam, iterations_adam)

plt.figure(figsize=(8, 6))
plt.contour(X, Y, Z, levels=50, cmap='viridis')
```

```
plt.plot(First[:, 0], First[:, 1], marker='o', color='blue', markersize=3, l
plt.plot(Second[:, 0], Second[:, 1], marker='o', color='green', markersize=3
plt.plot(Third[:, 0], Third[:, 1], marker='o', color='red', markersize=3, li


plt.scatter(*point_1, color='blue', label='Starting Point')
plt.scatter(*point_2, color='green', label='Starting Point')
plt.scatter(*point_3, color='red', label='Starting Point')


plt.xlabel('x axis')
plt.ylabel('y axis')
plt.title('Adaptive Moment Estimation')
plt.legend()
plt.show()
```



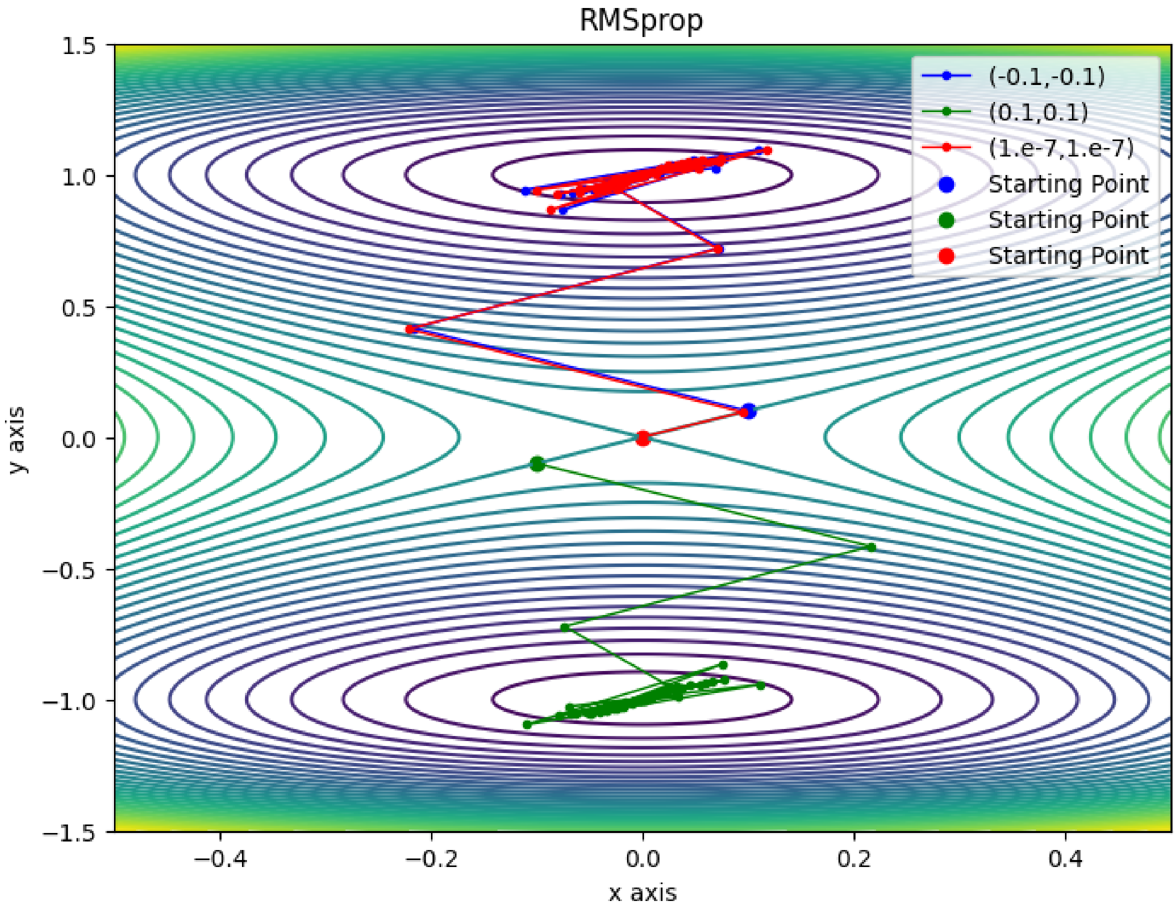
In [ ]:
```
def rmsprop(starting_point, learning_rate, iterations, beta=0.9, epsilon=1e-
    point = np.array(starting_point)
    trajectory = [point.copy()]
    v = np.zeros_like(point)

    for t in range(iterations):
        grad = gradient_f(*point)
        v = beta * v + (1 - beta) * np.square(grad)
        point -= learning_rate * grad / (np.sqrt(v+epsilon))
        trajectory.append(point.copy())

    return np.array(trajectory)
```

```
learning_rate_rmsprop = 0.01
iterations_rmsprop = 500

First = rmsprop(point_1, learning_rate_rmsprop, iterations_rmsprop)
Second = rmsprop(point_2, learning_rate_rmsprop, iterations_rmsprop)
Third = rmsprop(point_3, learning_rate_rmsprop, iterations_rmsprop)

plt.figure(figsize=(8, 6))
plt.contour(X, Y, Z, levels=50, cmap='viridis')

plt.plot(First[:, 0], First[:, 1], marker='o', color='blue', markersize=3, l
plt.plot(Second[:, 0], Second[:, 1], marker='o', color='green', markersize=3
plt.plot(Third[:, 0], Third[:, 1], marker='o', color='red', markersize=3, li


plt.scatter(*point_1, color='blue', label='Starting Point')
plt.scatter(*point_2, color='green', label='Starting Point')
plt.scatter(*point_3, color='red', label='Starting Point')


plt.xlabel('x axis')
plt.ylabel('y axis')
plt.title('RMSprop')
plt.legend()
plt.show()
```



In [ ]:

```
learning_rate_rmsprop = 0.1
iterations_rmsprop = 50

First = rmsprop(point_1, learning_rate_rmsprop, iterations_rmsprop)
Second = rmsprop(point_2, learning_rate_rmsprop, iterations_rmsprop)
Third = rmsprop(point_3, learning_rate_rmsprop, iterations_rmsprop)

plt.figure(figsize=(8, 6))
plt.contour(X, Y, Z, levels=50, cmap='viridis')

plt.plot(First[:, 0], First[:, 1], marker='o', color='blue', markersize=3, l
plt.plot(Second[:, 0], Second[:, 1], marker='o', color='green', markersize=3
plt.plot(Third[:, 0], Third[:, 1], marker='o', color='red', markersize=3, li


plt.scatter(*point_1, color='blue', label='Starting Point')
plt.scatter(*point_2, color='green', label='Starting Point')
plt.scatter(*point_3, color='red', label='Starting Point')


plt.xlabel('x axis')
plt.ylabel('y axis')
plt.title('RMSprop')
plt.legend()
plt.show()
```

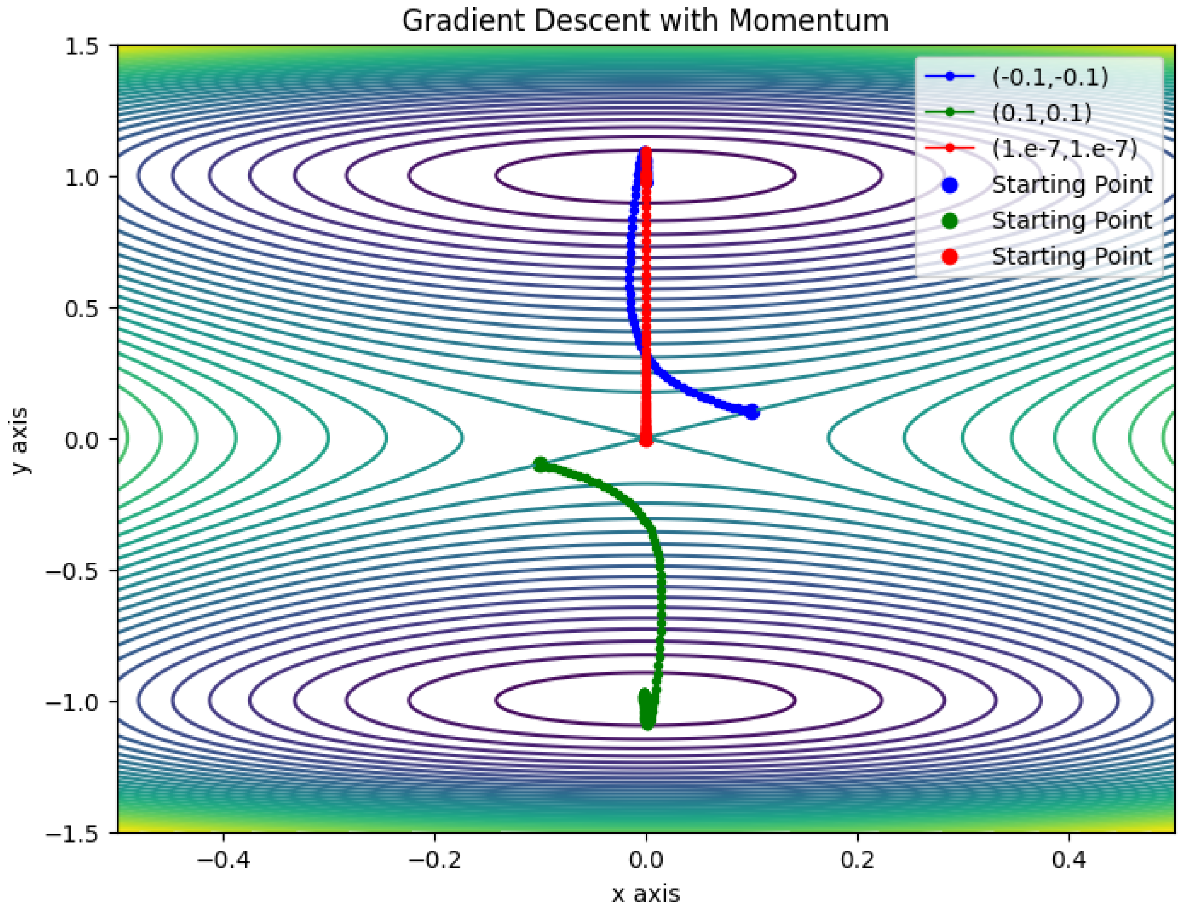


Note that for the Blue line it actually return to the local minimum

In [ ]:

```
# Defining the gradient descent with momentum function
def gradient_descent_with_momentum(starting_point, learning_rate, iterations
    point = np.array(starting_point)
    trajectory = [point.copy()]
    v = np.zeros_like(point)

    for _ in range(iterations):
        grad = gradient_f(*point)
        v = momentum * v - learning_rate * grad
        point += v
        trajectory.append(point.copy())

    return np.array(trajectory)


learning_rate_momentum = 0.01
iterations_momentum = 500

First = gradient_descent_with_momentum(point_1, learning_rate_momentum, iter
Second = gradient_descent_with_momentum(point_2, learning_rate_momentum, ite
Third = gradient_descent_with_momentum(point_3, learning_rate_momentum, iter

plt.figure(figsize=(8, 6))
plt.contour(X, Y, Z, levels=50, cmap='viridis')
```

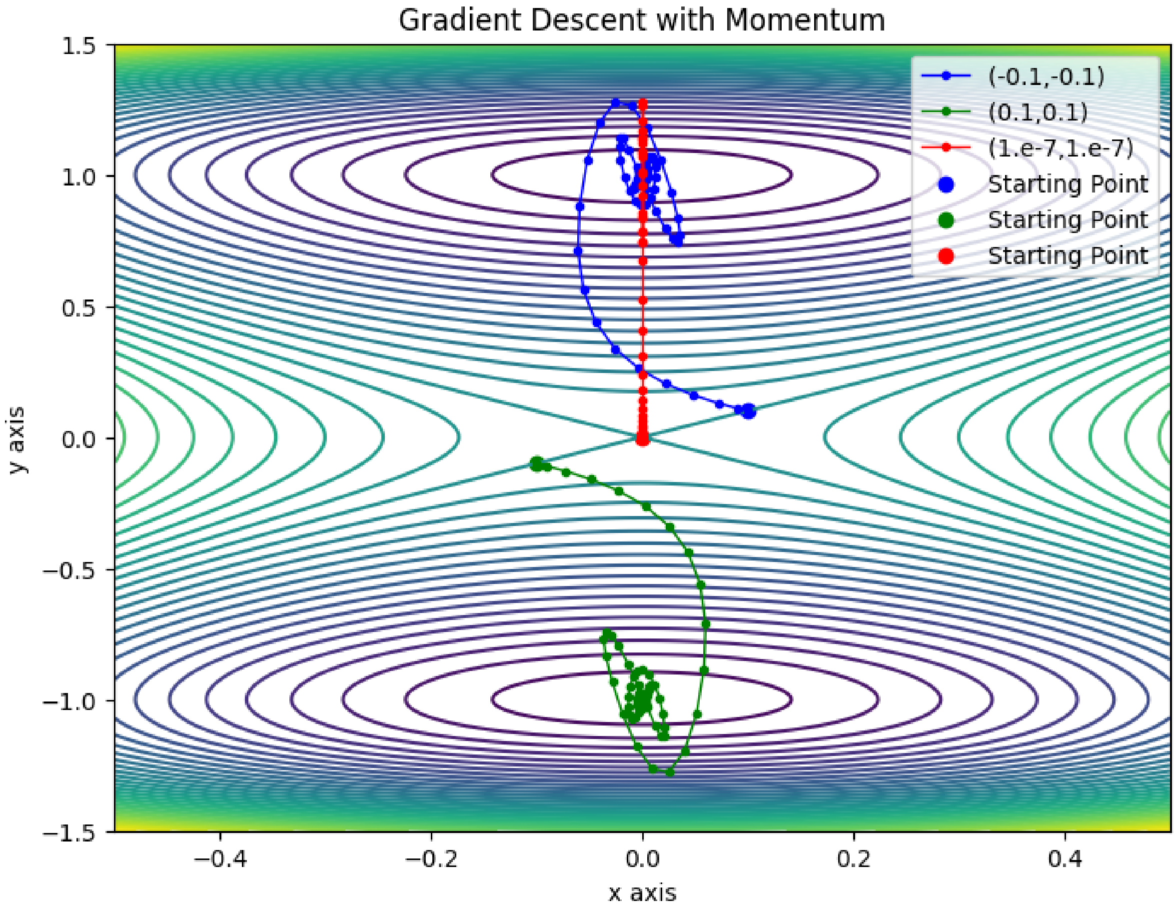
```
plt.plot(First[:, 0], First[:, 1], marker='o', color='blue', markersize=3, l
plt.plot(Second[:, 0], Second[:, 1], marker='o', color='green', markersize=3
plt.plot(Third[:, 0], Third[:, 1], marker='o', color='red', markersize=3, li


plt.scatter(*point_1, color='blue', label='Starting Point')
plt.scatter(*point_2, color='green', label='Starting Point')
plt.scatter(*point_3, color='red', label='Starting Point')


plt.xlabel('x axis')
plt.ylabel('y axis')
plt.title('Gradient Descent with Momentum')
plt.legend()
plt.show()
```



In [ ]:

```
learning_rate_momentum = 0.1
iterations_momentum = 80

First = gradient_descent_with_momentum(point_1, learning_rate_momentum, iter
Second = gradient_descent_with_momentum(point_2, learning_rate_momentum, ite
Third = gradient_descent_with_momentum(point_3, learning_rate_momentum, iter

plt.figure(figsize=(8, 6))
plt.contour(X, Y, Z, levels=50, cmap='viridis')

plt.plot(First[:, 0], First[:, 1], marker='o', color='blue', markersize=3, l
plt.plot(Second[:, 0], Second[:, 1], marker='o', color='green', markersize=3
```

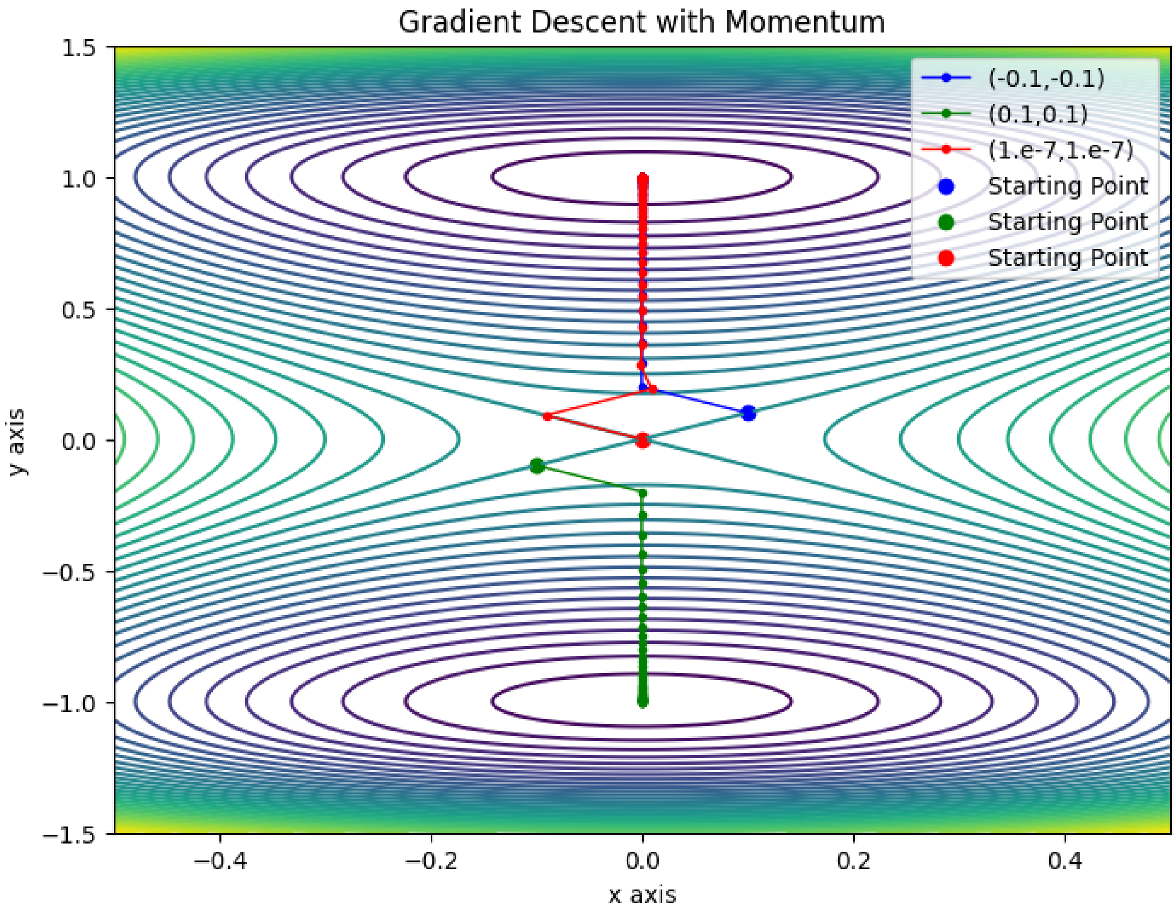
```
plt.plot(Third[:, 0], Third[:, 1], marker='o', color='red', markersize=3, li


plt.scatter(*point_1, color='blue', label='Starting Point')
plt.scatter(*point_2, color='green', label='Starting Point')
plt.scatter(*point_3, color='red', label='Starting Point')


plt.xlabel('x axis')
plt.ylabel('y axis')
plt.title('Gradient Descent with Momentum')
plt.legend()
plt.show()
```



In [ ]:

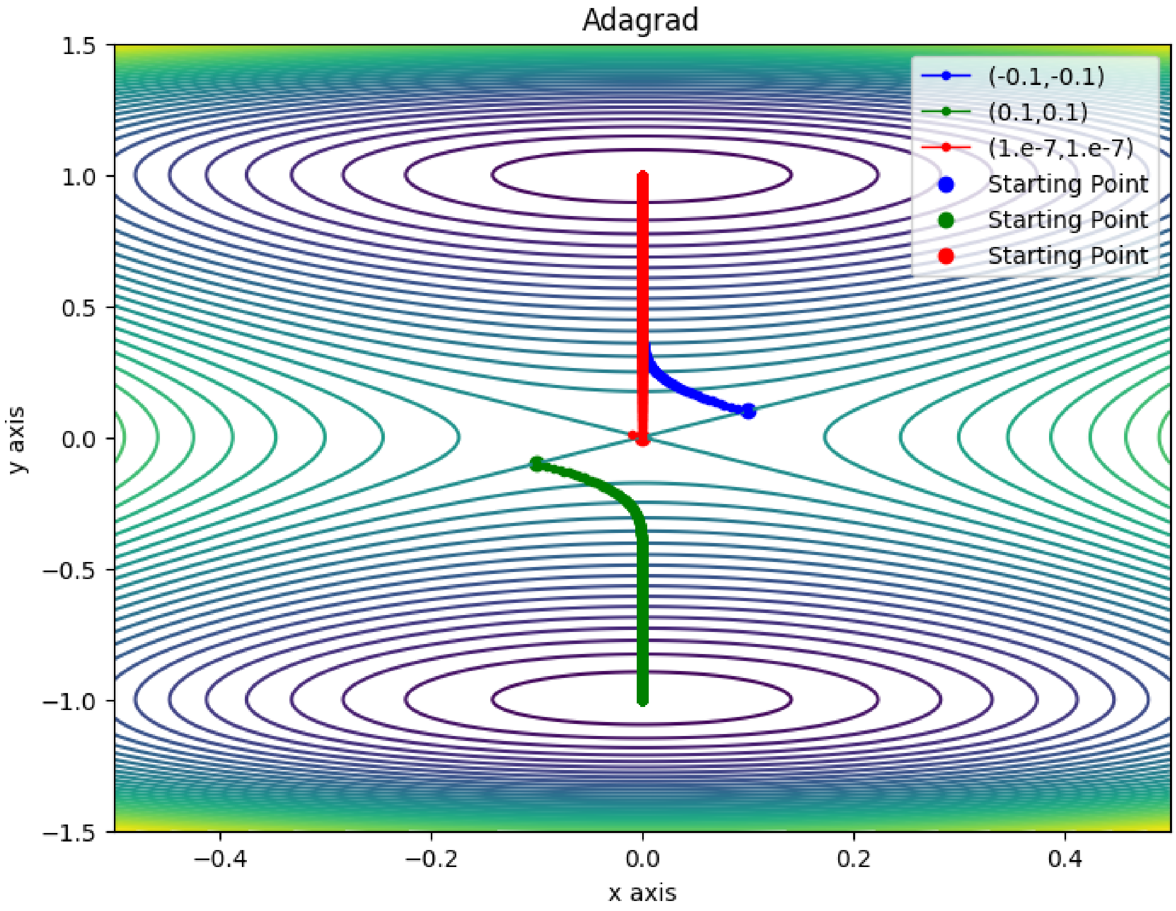
```
def adagrad(starting_point, learning_rate, iterations, epsilon=1e-8):
    point = np.array(starting_point)
    trajectory = [point.copy()]
    v = np.zeros_like(point)

    for _ in range(iterations):
        grad = gradient_f(*point)
        v += np.square(grad)
        point -= learning_rate * grad / (np.sqrt(v) + epsilon)
        trajectory.append(point.copy())

    return np.array(trajectory)
```

```
learning_rate_adagrad = 0.1
iterations_adagrad = 50


First = adagrad(point_1, learning_rate_adagrad, iterations_adagrad)
Second = adagrad(point_2, learning_rate_adagrad, iterations_adagrad)
Third = adagrad(point_3, learning_rate_adagrad, iterations_adagrad)

plt.figure(figsize=(8, 6))
plt.contour(X, Y, Z, levels=50, cmap='viridis')

plt.plot(First[:, 0], First[:, 1], marker='o', color='blue', markersize=3, l
plt.plot(Second[:, 0], Second[:, 1], marker='o', color='green', markersize=3
plt.plot(Third[:, 0], Third[:, 1], marker='o', color='red', markersize=3, li


plt.scatter(*point_1, color='blue', label='Starting Point')
plt.scatter(*point_2, color='green', label='Starting Point')
plt.scatter(*point_3, color='red', label='Starting Point')


plt.xlabel('x axis')
plt.ylabel('y axis')
plt.title('Gradient Descent with Momentum')
plt.legend()
plt.show()
```



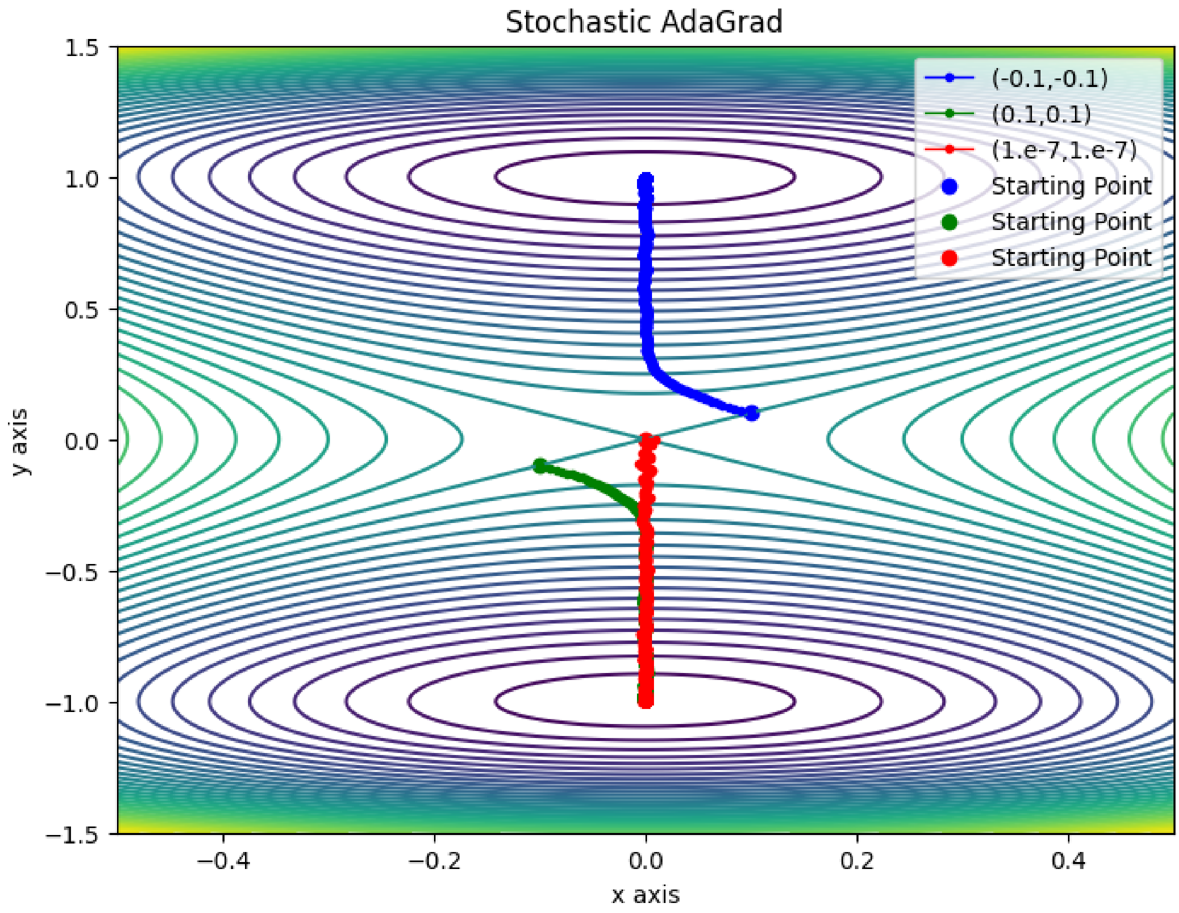
In [ ]:

```
learning_rate_adagrad = 0.01
iterations_adagrad = 5000


First = adagrad(point_1, learning_rate_adagrad, iterations_adagrad)
Second = adagrad(point_2, learning_rate_adagrad, iterations_adagrad)
Third = adagrad(point_3, learning_rate_adagrad, iterations_adagrad)

plt.figure(figsize=(8, 6))
plt.contour(X, Y, Z, levels=50, cmap='viridis')

plt.plot(First[:, 0], First[:, 1], marker='o', color='blue', markersize=3, l
plt.plot(Second[:, 0], Second[:, 1], marker='o', color='green', markersize=3
plt.plot(Third[:, 0], Third[:, 1], marker='o', color='red', markersize=3, li


plt.scatter(*point_1, color='blue', label='Starting Point')
plt.scatter(*point_2, color='green', label='Starting Point')
plt.scatter(*point_3, color='red', label='Starting Point')


plt.xlabel('x axis')
plt.ylabel('y axis')
plt.title('Adagrad')
plt.legend()
plt.show()
```



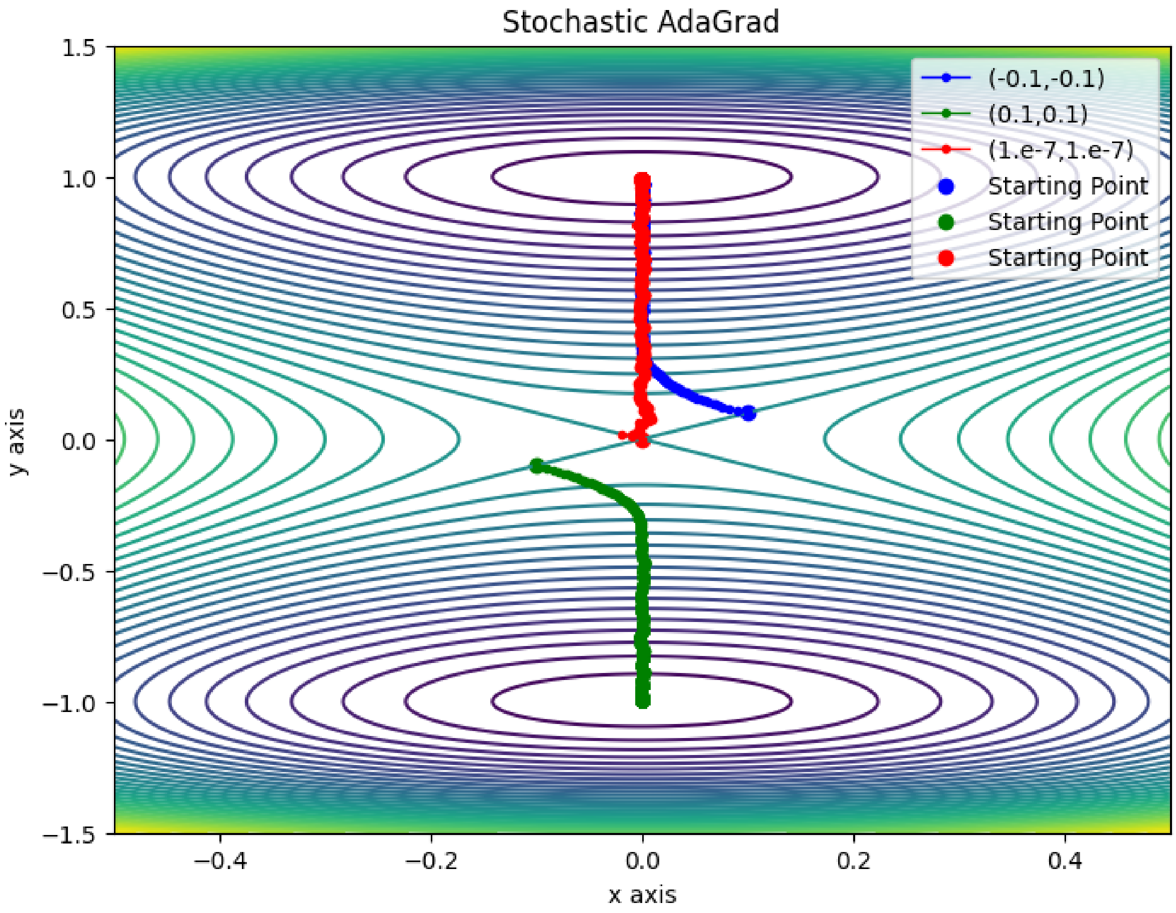
In [ ]:

```
def stochastic_adagrad(starting_point, learning_rate, iterations, epsilon=1e
    point = np.array(starting_point)
    trajectory = [point.copy()]
    v = np.zeros_like(point)

    for _ in range(iterations):
        stochastic_point = point + np.random.normal(scale=0.01, size=point.s
        grad = gradient_f(*stochastic_point)
        v += np.square(grad)
        point -= learning_rate * grad / (np.sqrt(v+epsilon))
        trajectory.append(point.copy())

    return np.array(trajectory)

learning_rate_sadagrad = 0.01
iterations_sadagrad = 5000


First = stochastic_adagrad(point_1, learning_rate_sadagrad, iterations_sadag
Second = stochastic_adagrad(point_2, learning_rate_sadagrad, iterations_sada
Third = stochastic_adagrad(point_3, learning_rate_sadagrad, iterations_sadag

plt.figure(figsize=(8, 6))
plt.contour(X, Y, Z, levels=50, cmap='viridis')

plt.plot(First[:, 0], First[:, 1], marker='o', color='blue', markersize=3, l
plt.plot(Second[:, 0], Second[:, 1], marker='o', color='green', markersize=3
plt.plot(Third[:, 0], Third[:, 1], marker='o', color='red', markersize=3, li


plt.scatter(*point_1, color='blue', label='Starting Point')
plt.scatter(*point_2, color='green', label='Starting Point')
plt.scatter(*point_3, color='red', label='Starting Point')

plt.xlabel('x axis')
plt.ylabel('y axis')
plt.title('Stochastic AdaGrad')
plt.legend()
plt.show()
```



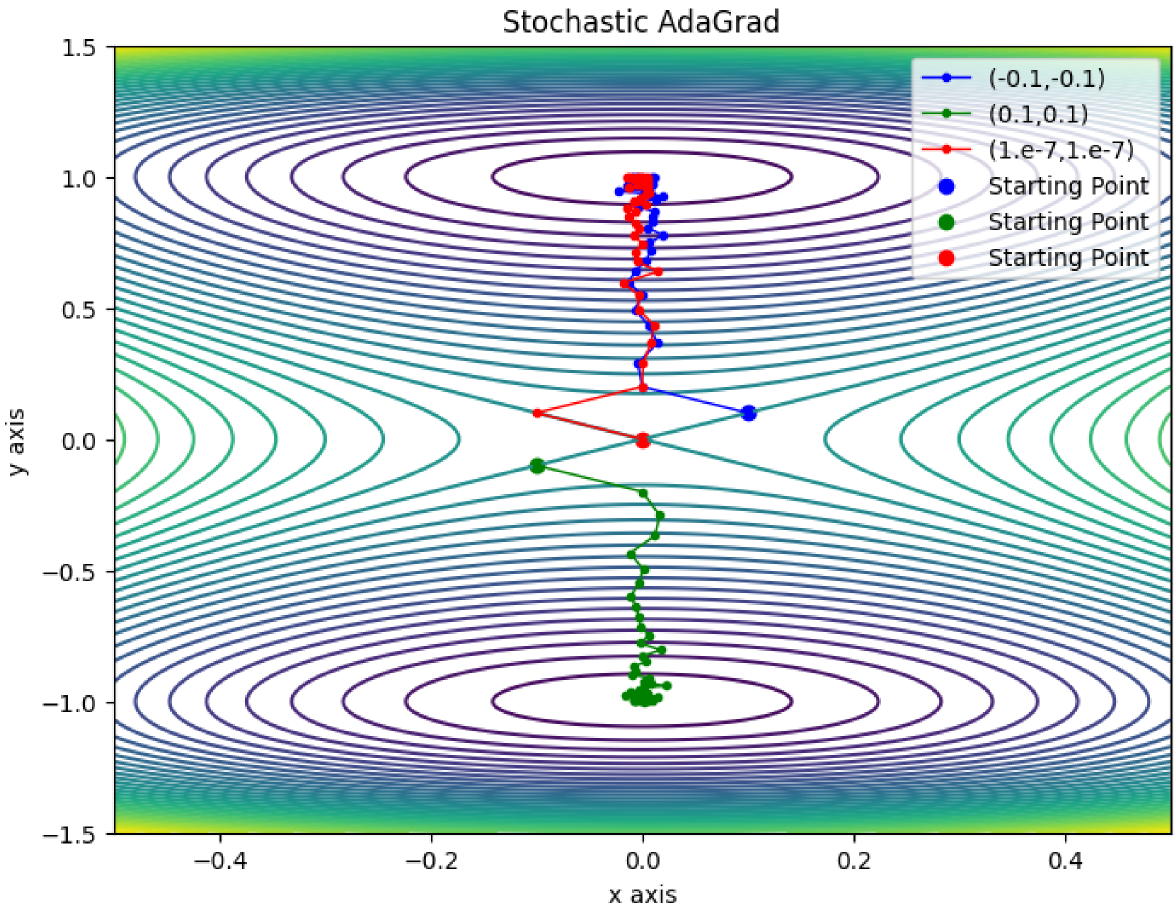
In [ ]:

```
learning_rate_sadagrad = 0.01
iterations_sadagrad = 5000


First = stochastic_adagrad(point_1, learning_rate_sadagrad, iterations_sadag
Second = stochastic_adagrad(point_2, learning_rate_sadagrad, iterations_sada
Third = stochastic_adagrad(point_3, learning_rate_sadagrad, iterations_sadag

plt.figure(figsize=(8, 6))
plt.contour(X, Y, Z, levels=50, cmap='viridis')

plt.plot(First[:, 0], First[:, 1], marker='o', color='blue', markersize=3, l
plt.plot(Second[:, 0], Second[:, 1], marker='o', color='green', markersize=3
plt.plot(Third[:, 0], Third[:, 1], marker='o', color='red', markersize=3, li


plt.scatter(*point_1, color='blue', label='Starting Point')
plt.scatter(*point_2, color='green', label='Starting Point')
plt.scatter(*point_3, color='red', label='Starting Point')

plt.xlabel('x axis')
plt.ylabel('y axis')
plt.title('Stochastic AdaGrad')
plt.legend()
plt.show()
```



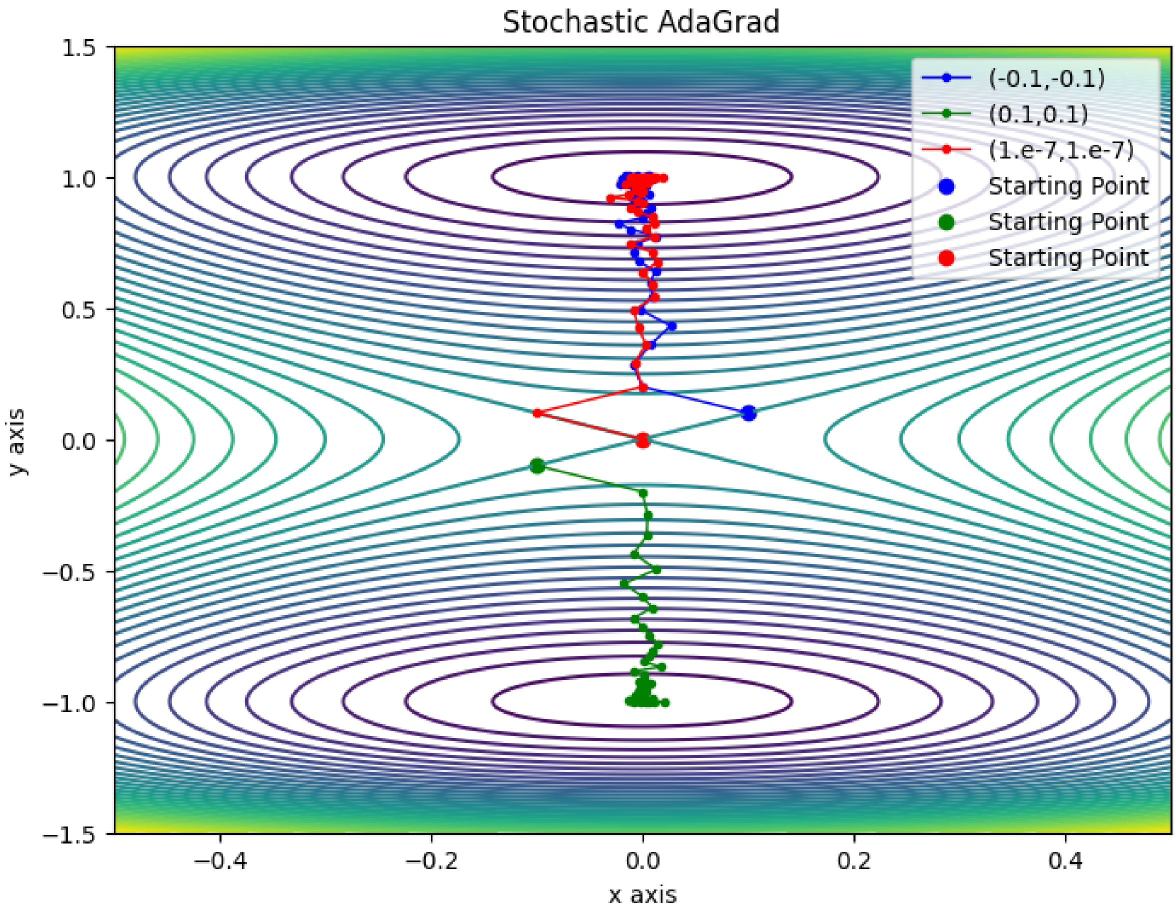
In [ ]:

```
learning_rate_sadagrad = 0.1
iterations_sadagrad = 50


First = stochastic_adagrad(point_1, learning_rate_sadagrad, iterations_sadag
Second = stochastic_adagrad(point_2, learning_rate_sadagrad, iterations_sada
Third = stochastic_adagrad(point_3, learning_rate_sadagrad, iterations_sadag

plt.figure(figsize=(8, 6))
plt.contour(X, Y, Z, levels=50, cmap='viridis')

plt.plot(First[:, 0], First[:, 1], marker='o', color='blue', markersize=3, l
plt.plot(Second[:, 0], Second[:, 1], marker='o', color='green', markersize=3
plt.plot(Third[:, 0], Third[:, 1], marker='o', color='red', markersize=3, li


plt.scatter(*point_1, color='blue', label='Starting Point')
plt.scatter(*point_2, color='green', label='Starting Point')
plt.scatter(*point_3, color='red', label='Starting Point')

plt.xlabel('x axis')
plt.ylabel('y axis')
plt.title('Stochastic AdaGrad')
plt.legend()
plt.show()
```



In [ ]:

```
learning_rate_sadagrad = 0.1
iterations_sadagrad = 50


First = stochastic_adagrad(point_1, learning_rate_sadagrad, iterations_sadag
Second = stochastic_adagrad(point_2, learning_rate_sadagrad, iterations_sada
Third = stochastic_adagrad(point_3, learning_rate_sadagrad, iterations_sadag

plt.figure(figsize=(8, 6))
plt.contour(X, Y, Z, levels=50, cmap='viridis')

plt.plot(First[:, 0], First[:, 1], marker='o', color='blue', markersize=3, l
plt.plot(Second[:, 0], Second[:, 1], marker='o', color='green', markersize=3
plt.plot(Third[:, 0], Third[:, 1], marker='o', color='red', markersize=3, li


plt.scatter(*point_1, color='blue', label='Starting Point')
plt.scatter(*point_2, color='green', label='Starting Point')
plt.scatter(*point_3, color='red', label='Starting Point')

plt.xlabel('x axis')
plt.ylabel('y axis')
plt.title('Stochastic AdaGrad')
plt.legend()
plt.show()
```



MP4 Generator

If you will like to see a different algorithm in MP4, make sure change the xxxx_path to the one you want see.

In [ ]:

```
from matplotlib.animation import FuncAnimation
import matplotlib.animation as animation

fig, ax = plt.subplots(figsize=(8, 6))

contour = ax.contour(B0, B1, Loss, levels=levels)
path, = ax.plot([], [], label='RMSP', marker='.')
ax.legend()

def update(i):
    path.set_data(rmsprop_path[:i, 0], rmsprop_path[:i, 1])
    return path,

ani = FuncAnimation(fig, update, frames=np.arange(len(rmsprop_path)), blit=T

# Save animation as an mp4 file
Writer = animation.writers['ffmpeg']
writer = Writer(fps=7, metadata=dict(artist='Me'), bitrate=1800)
ani.save('RMSP_path.mp4', writer=writer)
```

```
plt.show()


fig = plt.figure(figsize=(12, 12))
ax = fig.add_subplot(111, projection='3d')

surf = ax.plot_surface(B0, B1, Loss, cmap='viridis', rstride=1, cstride=1, a
contour = ax.contour(B0, B1, Loss, zdir='z', offset=np.min(Loss), cmap='viri

scatter, = ax.plot([], [], [], 'r', marker='o', markersize=5, label='RMSP Pa
line, = ax.plot([], [], [], 'r', linestyle='solid', linewidth=2)

def init():
    scatter.set_data([], [])
    scatter.set_3d_properties([])
    line.set_data([], [])
    line.set_3d_properties([])
    return scatter, line,

def update(i):
    current_beta = rmsprop_path[i]
    loss_value = loss(Y, x, current_beta)
    scatter.set_data(rmsprop_path[:i+1, 0], rmsprop_path[:i+1, 1])
    scatter.set_3d_properties(loss_values[:i+1])
    line.set_data(rmsprop_path[:i+1, 0], rmsprop_path[:i+1, 1])
    line.set_3d_properties(loss_values[:i+1])
    return scatter, line,

loss_values = [loss(Y, x, beta) for beta in rmsprop_path]
ax.view_init(elev=20, azim=135)

font_dict = {'family': 'serif', 'color':  'darkred', 'weight': 'normal', 'si
ax.set_xlabel(r'$\beta_0$', fontdict=font_dict)
ax.set_ylabel(r'$\beta_1$', fontdict=font_dict)
ax.set_zlabel('Loss', fontdict=font_dict)
ax.set_title('Root Mean Squre Propagation Path Animation', pad=35, fontsize=

ax.legend(loc='upper right', fontsize=12, frameon=True, facecolor='ivory', e

ani = FuncAnimation(fig, update, frames=np.arange(len(rmsprop_path)), init_f

# Save animation as an mp4 file
Writer = animation.writers['ffmpeg']
writer = Writer(fps=7, metadata=dict(artist='Me'), bitrate=1800)
ani.save('RMSP_path_3d.mp4', writer=writer)

plt.tight_layout()
plt.show()





iterations = range(len(rmsprop_loss))

fig, ax = plt.subplots()
line, = ax.plot([], [], lw=2)
```

```
ax.set_xlim(0, len(rmsprop_loss) - 1)
ax.set_ylim(np.min(rmsprop_loss), np.max(rmsprop_loss))
ax.set_xlabel('Iterations')
ax.set_ylabel('Loss')

def init():
    line.set_data([], [])
    return line,

def animate(i):
    x = iterations[:i+1]
    y = rmsprop_loss[:i+1]
    line.set_data(x, y)
    return line,

ani = FuncAnimation(fig, animate, init_func=init, frames=len(rmsprop_loss),

Writer = animation.writers['ffmpeg']
writer = Writer(fps=7, metadata=dict(artist='Me'), bitrate=1800)
ani.save('loss_over_iterations.mp4', writer=writer)
```