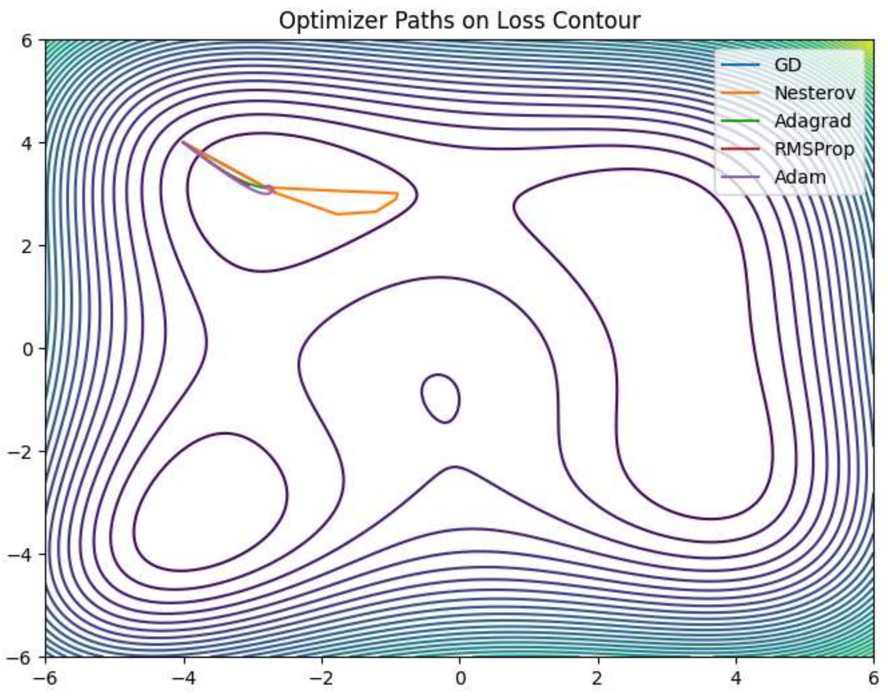
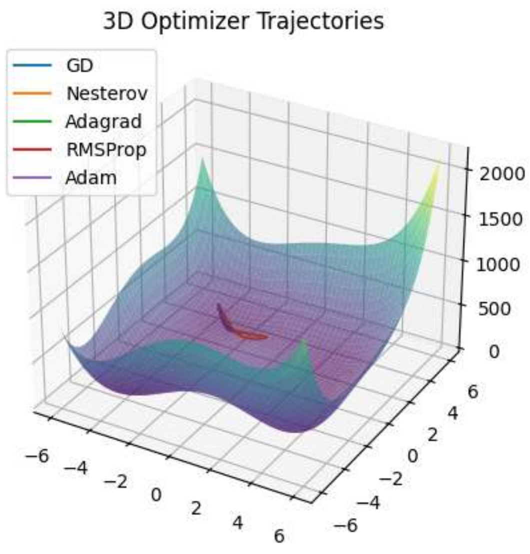
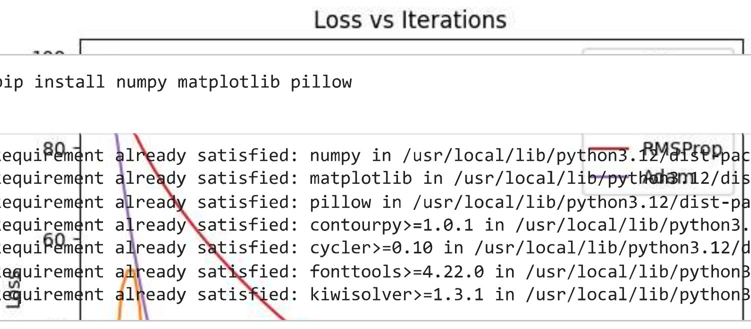
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
import tensorflow as tf
from sklearn.preprocessing import StandardScaler
def J(w):
    w1, w2 = w
    return (w1**2 + w2 - 11)**2 + (w1 + w2**2 - 7)**2
def grad_J(w):
    w1, w2 = w
    dw1 = 4*w1*(w1**2 + w2 - 11) + 2*(w1 + w2**2 - 7)
    dw2 = 2*(w1**2 + w2 - 11) + 4*w2*(w1 + w2**2 - 7)
    return np.array([dw1, dw2])
def GD(init, lr=0.01, steps=100):
    w = init.copy()
    path=[w.copy()]
    loss=[]
    for _ in range(steps):
        w -= lr*grad_J(w)
        path.append(w.copy())
        loss.append(J(w))
    return np.array(path), loss
def Nesterov(init, lr=0.01, gamma=0.9, steps=100):
    w = init.copy()
    v = np.zeros_like(w)
    path=[w.copy()]
    loss=[]
    for _ in range(steps):
        lookahead = w - gamma*v
        g = grad_J(lookahead)
        v = gamma*v + lr*g
        w -= v
        path.append(w.copy())
        loss.append(J(w))
    return np.array(path), loss
def Adagrad(init, lr=0.4, steps=100):
    w = init.copy()
    G = np.zeros_like(w)
    eps=1e-8
    path=[w.copy()]
    loss=[]
    for _ in range(steps):
        g=grad_J(w)
        G += g**2
        w -= lr*g/(np.sqrt(G)+eps)
        path.append(w.copy())
        loss.append(J(w))
    return np.array(path), loss
def RMSProp(init, lr=0.01, beta=0.9, steps=100):
    w=init.copy()
    Eg=np.zeros_like(w)
    eps=1e-8
    path=[w.copy()]
    loss=[]
    for _ in range(steps):
        g=grad_J(w)
        Eg=beta*Eg+(1-beta)*(g**2)
        w -= lr*g/(np.sqrt(Eg)+eps)
        path.append(w.copy())
        loss.append(J(w))
    return np.array(path), loss
def Adam(init, lr=0.05, steps=100):
    w=init.copy()
    m=np.zeros_like(w)
    v=np.zeros_like(w)
    b1,b2=0.9,0.999
    eps=1e-8
    path=[w.copy()]
    loss=[]
    for t in range(1,steps+1):
        g=grad_J(w)
        m=b1*m+(1-b1)*g
        v=b2*v+(1-b2)*(g**2)
        mhat=m/(1-b1**t)
        vhat=v/(1-b2**t)
        w -= lr*mhat/(np.sqrt(vhat)+eps)
        path.append(w.copy())
        loss.append(J(w))
    return np.array(path), loss
init = np.array([-4.0, 4.0])
```

```
optimizers = {
    "GD": GD,
    "Nesterov": Nesterov,
    "Adagrad": Adagrad,
    "RMSProp": RMSProp,
    "Adam": Adam
}
paths={}
losses={}
for name,opt in optimizers.items():
    paths[name], losses[name] = opt(init)
x=np.linspace(-6,6,200)
y=np.linspace(-6,6,200)
X,Y=np.meshgrid(x,y)
Z=J([X,Y])
plt.figure(figsize=(8,6))
plt.contour(X,Y,Z,40)
for name,p in paths.items():
    plt.plot(p[:,0],p[:,1],label=name)
plt.legend()
plt.title("Optimizer Paths on Loss Contour")
plt.show()
#3D SURFACE
fig=plt.figure()
ax=fig.add_subplot(111,projection='3d')
ax.plot_surface(X,Y,Z,cmap=cm.viridis,alpha=0.6)

for name,p in paths.items():
    z=[J(w) for w in p]
    ax.plot(p[:,0],p[:,1],z,label=name)

ax.set_title("3D Optimizer Trajectories")
plt.legend()
plt.show()
#LOSS CURVES
plt.figure()
for name,l in losses.items():
    plt.plot(l,label=name)
plt.legend()
plt.xlabel("Iterations")
plt.ylabel("Loss")
plt.title("Loss vs Iterations")
plt.show()
#Load MNIST
(x_train, y_train), _ = tf.keras.datasets.mnist.load_data()
mask=(y_train==0)|(y_train==1)
x=x_train[mask]
y=y_train[mask]
x=x.reshape(len(x),-1)
f1=x[:,:392].mean(axis=1)
f2=x[:,392:].mean(axis=1)
X=np.vstack((f1,f2)).T
X=StandardScaler().fit_transform(X)
y=y.astype(np.float32)
def sigmoid(z):
    return 1/(1+np.exp(-z))
def loss_lr(w):
    p=sigmoid(X.dot(w))
    return -np.mean(y*np.log(p+1e-8)+(1-y)*np.log(1-p+1e-8))
def grad_lr(w):
    p=sigmoid(X.dot(w))
    return X.T.dot(p-y)/len(y)
def train_adam_lr(steps=200, lr=0.05):
    w=np.zeros(2)
    m=np.zeros(2)
    v=np.zeros(2)
    b1,b2=0.9,0.999
    eps=1e-8
    path=[w.copy()]
    losses=[]
    for t in range(1,steps+1):
        g=grad_lr(w)
        m=b1*m+(1-b1)*g
        v=b2*v+(1-b2)*(g**2)
        mhat=m/(1-b1**t)
        vhat=v/(1-b2**t)
        w -= lr*mhat/(np.sqrt(vhat)+eps)
        path.append(w.copy())
        losses.append(loss_lr(w))
    return np.array(path),losses
path_lr,loss_lr_vals=train_adam_lr()
#loss curve
```

```
plt.plot(loss_lr_vals)
plt.title("Logistic Regression Loss")
plt.xlabel("Iterations")
plt.ylabel("Loss")
plt.show()
w1=np.linspace(-5,5,100)
w2=np.linspace(-5,5,100)
W1,W2=np.meshgrid(w1,w2)
Z=np.zeros_like(W1)

for i in range(W1.shape[0]):
    for j in range(W1.shape[1]):
        Z[i,j]=loss_lr(np.array([W1[i,j],W2[i,j]]))

plt.contour(W1,W2,Z,40)
plt.plot(path_lr[:,0],path_lr[:,1],'r')
plt.title("Optimizer Path on Logistic Loss Surface")
plt.show()
```

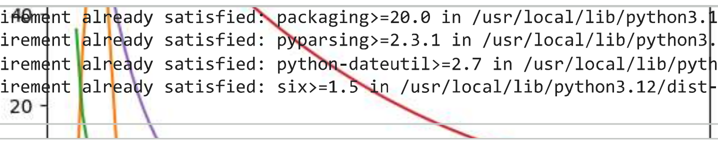






```
pip install numpy matplotlib pillow
```

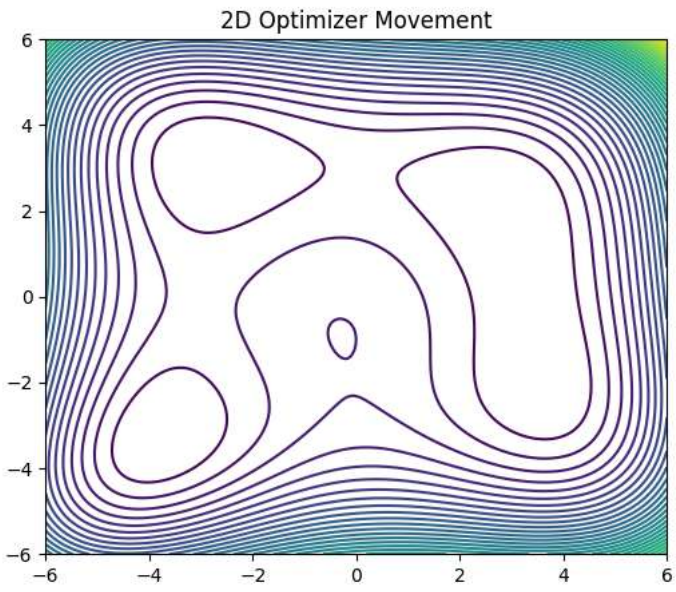
```
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (2.0.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages (3.10.0)
Requirement already satisfied: pillow in /usr/local/lib/python3.12/dist-packages (11.3.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (4.61.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.4.9)
```



```
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (26.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (3.3.2)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (2.9.0.post
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.7->matplotlib) (
```

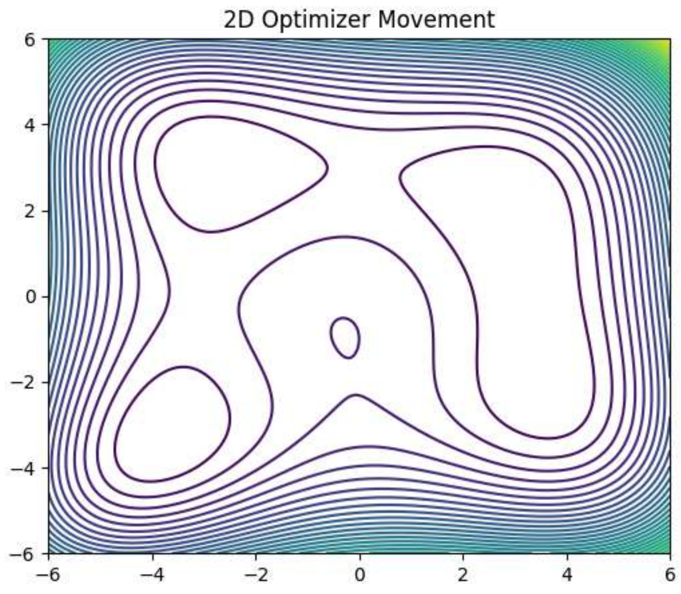
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation, cm

# Himmelblau Function
def J(w):
    w1, w2 = w
    return (w1**2 + w2 - 11)**2 + (w1 + w2**2 - 7)**2

# Gradient
def grad_J(w):
    w1, w2 = w
    dw1 = 4*w1*(w1**2 + w2 - 11) + 2*(w1 + w2**2 - 7)
    dw2 = 2*(w1**2 + w2 - 11) + 4*w2*(w1 + w2**2 - 7)
    return np.array([dw1, dw2])

# Gradient Descent optimizer
def gradient_descent(start, lr=0.01, steps=80):
    w = start.copy()
    path = [w.copy()]
    for _ in range(steps):
        w -= lr * grad_J(w)
        path.append(w.copy())
    return np.array(path)

# Starting point
init = np.array([-4.0, 4.0])
path = gradient_descent(init)

# Create grid for contour & surface
x = np.linspace(-6, 6, 200)
y = np.linspace(-6, 6, 200)
X, Y = np.meshgrid(x, y)
Z = J([X, Y])

############################################################
# 🎬 2D ANIMATED CONTOUR
############################################################

fig2d, ax2d = plt.subplots(figsize=(6,5))
ax2d.contour(X, Y, Z, levels=40)
line, = ax2d.plot([], [], 'r-', lw=2)
point, = ax2d.plot([], [], 'ro')

ax2d.set_title("2D Optimizer Movement")

def animate2d(i):
    line.set_data(path[:i,0], path[:i,1])
    point.set_data(path[i-1,0], path[i-1,1])
    return line, point

ani2d = animation.FuncAnimation(
    fig2d, animate2d, frames=len(path),
    interval=120, blit=True)

ani2d.save("optimizer_2D.gif", writer="pillow")
plt.close()

############################################################
# 🎬 3D ANIMATED SURFACE
############################################################

fig3d = plt.figure(figsize=(7,6))
ax3d = fig3d.add_subplot(111, projection='3d')

def animate3d(i):
    ax3d.clear()
    ax3d.plot_surface(X, Y, Z, cmap=cm.viridis, alpha=0.6)

    px = path[:i,0]
    py = path[:i,1]
    pz = [J(p) for p in path[:i]]

    ax3d.plot(px, py, pz, 'r-', linewidth=2)
    ax3d.scatter(px[-1], py[-1], pz[-1], color='red', s=40)
    ax3d.set_title("3D Optimizer Movement")
    ax3d.set_xlabel("w1")
    ax3d.set_ylabel("w2")
    ax3d.set_zlabel("Loss")

ani3d = animation.FuncAnimation(
    fig3d, animate3d, frames=len(path),
    interval=120)

ani3d.save("optimizer_3D.gif", writer="pillow")
plt.close()

print("Animations saved:")
print("optimizer_2D.gif")
print("optimizer_3D.gif")
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
/tmp/ipython-input-2470978122.py in <cell line: 0>()
     50     return line, point
     51 
---> 52 ani2d = animation.FuncAnimation(
     53     fig2d, animate2d, frames=len(path),
     54     interval=120, blit=True)

                                  19 frames
/usr/local/lib/python3.12/dist-packages/matplotlib/lines.py in set_xdata(self, x)
   1288         """
   1289         if not np.iterable(x):
-> 1290             raise RuntimeError('x must be a sequence')
   1291         self._xorig = copy.copy(x)
   1292         self._invalidx = True

RuntimeError: x must be a sequence
```



```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation, cm

# Himmelblau Function
def J(w):
    w1, w2 = w
    return (w1**2 + w2 - 11)**2 + (w1 + w2**2 - 7)**2

# Gradient
def grad_J(w):
    w1, w2 = w
    dw1 = 4*w1*(w1**2 + w2 - 11) + 2*(w1 + w2**2 - 7)
    dw2 = 2*(w1**2 + w2 - 11) + 4*w2*(w1 + w2**2 - 7)
    return np.array([dw1, dw2])

# Gradient Descent optimizer
def gradient_descent(start, lr=0.01, steps=80):
    w = start.copy()
    path = [w.copy()]
    for _ in range(steps):
        w -= lr * grad_J(w)
        path.append(w.copy())
    return np.array(path)

# Starting point
init = np.array([-4.0, 4.0])
path = gradient_descent(init)

# Create grid for contour & surface
x = np.linspace(-6, 6, 200)
y = np.linspace(-6, 6, 200)
X, Y = np.meshgrid(x, y)
Z = J([X, Y])

############################################################
# 🎬 2D ANIMATED CONTOUR
############################################################

fig2d, ax2d = plt.subplots(figsize=(6,5))
ax2d.contour(X, Y, Z, levels=40)
line, = ax2d.plot([], [], 'r-', lw=2)
point, = ax2d.plot([], [], 'ro')

ax2d.set_title("2D Optimizer Movement")

def animate2d(i):
    line.set_data(path[:i,0], path[:i,1])
    point.set_data(path[i-1,0], path[i-1,1])
    return line, point

ani2d = animation.FuncAnimation(
    fig2d, animate2d, frames=len(path),
    interval=120, blit=True)

ani2d.save("optimizer_2D.gif", writer="pillow")
plt.close()

############################################################
# 🎬 3D ANIMATED SURFACE
############################################################

fig3d = plt.figure(figsize=(7,6))
ax3d = fig3d.add_subplot(111, projection='3d')

def animate3d(i):
    ax3d.clear()
    ax3d.plot_surface(X, Y, Z, cmap=cm.viridis, alpha=0.6)

    px = path[:i,0]
    py = path[:i,1]
    pz = [J(p) for p in path[:i]]

    ax3d.plot(px, py, pz, 'r-', linewidth=2)
    ax3d.scatter(px[-1], py[-1], pz[-1], color='red', s=40)

    ax3d.set_title("3D Optimizer Movement")
    ax3d.set_xlabel("w1")
    ax3d.set_ylabel("w2")
    ax3d.set_zlabel("Loss")

ani3d = animation.FuncAnimation(
    fig3d, animate3d, frames=len(path),
    interval=120)

ani3d.save("optimizer_3D.gif", writer="pillow")
plt.close()

print("Animations saved:")
print("optimizer_2D.gif")
print("optimizer_3D.gif")
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
/tmp/ipython-input-1076376227.py in <cell line: 0>()
     50     return line, point
     51 
---> 52 ani2d = animation.FuncAnimation(
     53     fig2d, animate2d, frames=len(path),
     54     interval=120, blit=True)

19 frames
/usr/local/lib/python3.12/dist-packages/matplotlib/lines.py in set_xdata(self, x)
   1288         """
   1289         if not np.iterable(x):
-> 1290             raise RuntimeError('x must be a sequence')
   1291         self._xorig = copy.copy(x)
   1292         self._invalidx = True

RuntimeError: x must be a sequence
```



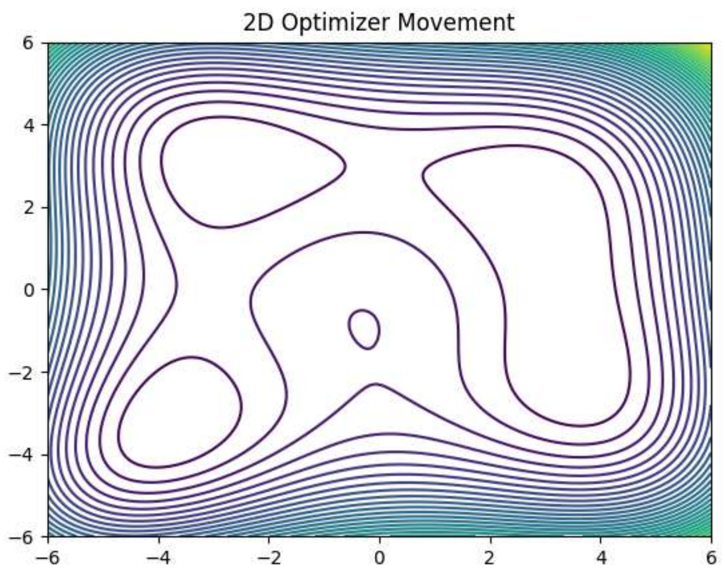
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation, cm

# Himmelblau function
def J(w):
    w1, w2 = w
    return (w1**2 + w2 - 11)**2 + (w1 + w2**2 - 7)**2

# gradient
def grad_J(w):
    w1, w2 = w
    dw1 = 4*w1*(w1**2 + w2 - 11) + 2*(w1 + w2**2 - 7)
    dw2 = 2*(w1**2 + w2 - 11) + 4*w2*(w1 + w2**2 - 7)
    return np.array([dw1, dw2])

# gradient descent path
def gradient_descent(start, lr=0.01, steps=80):
    w = start.copy()
    path = [w.copy()]
    for _ in range(steps):
        w -= lr * grad_J(w)
        path.append(w.copy())
    return np.array(path)

init = np.array([-4.0, 4.0])
path = gradient_descent(init)

# grid
x = np.linspace(-6,6,200)
y = np.linspace(-6,6,200)
X,Y = np.meshgrid(x,y)
Z = J([X,Y])

#################################################
# 🎬 2D ANIMATION (FIXED)
#################################################

fig2d, ax2d = plt.subplots()
```

```
ax2d.contour(X,Y,Z,40)
line, = ax2d.plot([], [], 'r-', lw=2)
point, = ax2d.plot([], [], 'ro')

def animate2d(i):
    if i == 0:
        return line, point
    line.set_data(path[:i,0], path[:i,1])
    point.set_data([path[i-1,0]], [path[i-1,1]])
    return line, point

ani2d = animation.FuncAnimation(
    fig2d, animate2d,
    frames=len(path),
    interval=120,
    blit=True
)

ani2d.save("optimizer_2D.gif", writer="pillow")
plt.close()

##################################################
# 🎬 3D ANIMATION (FIXED)
##################################################

fig3d = plt.figure()
ax3d = fig3d.add_subplot(111, projection='3d')

def animate3d(i):
    ax3d.clear()
    ax3d.plot_surface(X,Y,Z,cmap=cm.viridis,alpha=0.6)

    if i == 0:
        return

    px = path[:i,0]
    py = path[:i,1]
    pz = [J(p) for p in path[:i]]

    ax3d.plot(px,py,pz,'r-',linewidth=2)
    ax3d.scatter(px[-1],py[-1],pz[-1],color='red')

ani3d = animation.FuncAnimation(
    fig3d, animate3d,
    frames=len(path),
    interval=120
)

ani3d.save("optimizer_3D.gif", writer="pillow")
plt.close()

print("Animations saved successfully!")
```

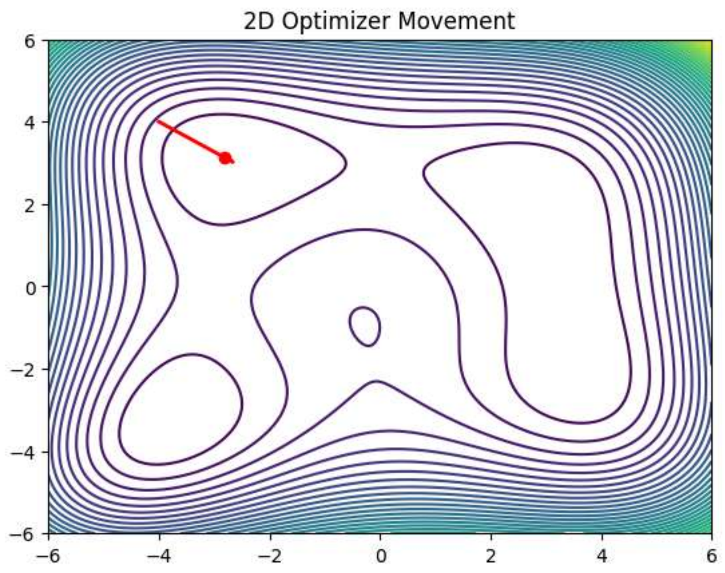
```
Animations saved successfully!
```

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation, cm
from IPython.display import HTML
def J(w):
    w1, w2 = w
    return (w1**2 + w2 - 11)**2 + (w1 + w2**2 - 7)**2
def grad_J(w):
    w1, w2 = w
    dw1 = 4*w1*(w1**2 + w2 - 11) + 2*(w1 + w2**2 - 7)
    dw2 = 2*(w1**2 + w2 - 11) + 4*w2*(w1 + w2**2 - 7)
    return np.array([dw1, dw2])
def gradient_descent(start, lr=0.01, steps=80):
    w = start.copy()
    path = [w.copy()]
    for _ in range(steps):
        w -= lr * grad_J(w)
        path.append(w.copy())
    return np.array(path)
init = np.array([-4.0, 4.0])
path = gradient_descent(init)
x = np.linspace(-6,6,200)
y = np.linspace(-6,6,200)
X,Y = np.meshgrid(x,y)
Z = J([X,Y])
fig2d, ax2d = plt.subplots()
```

```
ax2d.contour(X,Y,Z,40)
line, = ax2d.plot([], [], 'r-', lw=2)
point, = ax2d.plot([], [], 'ro')
ax2d.set_title("2D Optimizer Movement")
def animate2d(i):
    if i == 0:
        return line, point
    line.set_data(path[:i,0], path[:i,1])
    point.set_data([path[i-1,0]], [path[i-1,1]])
    return line, point

ani2d = animation.FuncAnimation(
    fig2d, animate2d,
    frames=len(path),
    interval=120,
    blit=True
)
HTML(ani2d.to_jshtml())
```

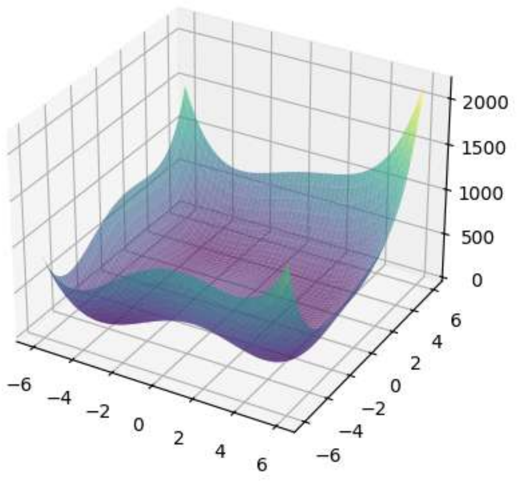


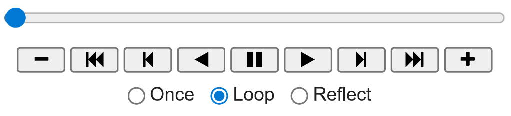
○ Once [✓ Loop] ○ Reflect

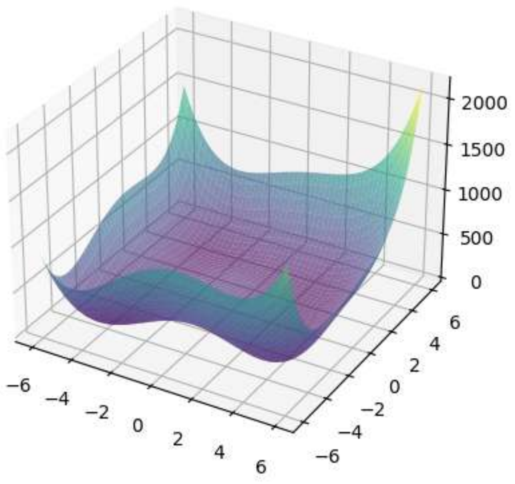


```
fig3d = plt.figure()
ax3d = fig3d.add_subplot(111, projection='3d')

def animate3d(i):
    ax3d.clear()
    ax3d.plot_surface(X,Y,Z,cmap=cm.viridis,alpha=0.6)
```

```
        if i == 0:
            return

        px = path[:i,0]
        py = path[:i,1]
        pz = [J(p) for p in path[:i]]

        ax3d.plot(px,py,pz,'r-',linewidth=2)
        ax3d.scatter(px[-1],py[-1],pz[-1],color='red')

        ax3d.set_title("3D Optimizer Movement")

    ani3d = animation.FuncAnimation(
        fig3d, animate3d,
        frames=len(path),
        interval=150
    )

    HTML(ani3d.to_jshtml())
```



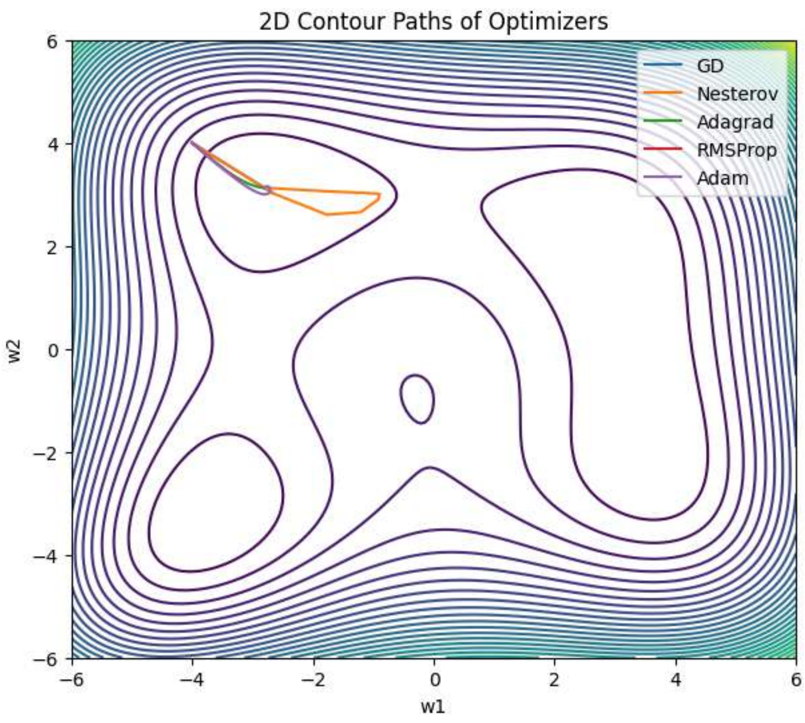


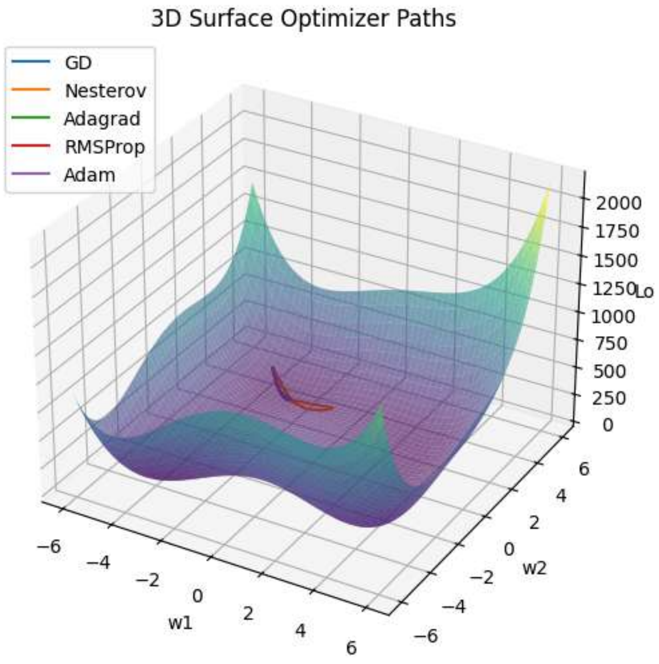


```
    import numpy as np
    import matplotlib.pyplot as plt
    from matplotlib import cm

    # Himmelblau function
    def J(w):
        w1, w2 = w
        return (w1**2 + w2 - 11)**2 + (w1 + w2**2 - 7)**2
```

```
# gradient
def grad_J(w):
    w1, w2 = w
    dw1 = 4*w1*(w1**2 + w2 - 11) + 2*(w1 + w2**2 - 7)
    dw2 = 2*(w1**2 + w2 - 11) + 4*w2*(w1 + w2**2 - 7)
    return np.array([dw1, dw2])

# ---------- OPTIMIZERS ----------

def GD(w, lr): return w - lr*grad_J(w)

def Nesterov(w, v, lr, gamma=0.9):
    look = w - gamma*v
    g = grad_J(look)
    v = gamma*v + lr*g
    return w - v, v

def Adagrad(w, G, lr):
    g = grad_J(w)
    G += g**2
    w -= lr*g/(np.sqrt(G)+1e-8)
    return w, G

def RMSProp(w, Eg, lr, beta=0.9):
    g = grad_J(w)
    Eg = beta*Eg + (1-beta)*(g**2)
    w -= lr*g/(np.sqrt(Eg)+1e-8)
    return w, Eg

def Adam(w, m, v, t, lr):
    g = grad_J(w)
    m = 0.9*m + 0.1*g
    v = 0.999*v + 0.001*(g**2)
    mhat = m/(1-0.9**t)
    vhat = v/(1-0.999**t)
    w -= lr*mhat/(np.sqrt(vhat)+1e-8)
    return w, m, v

# run optimizers
init = np.array([-4.0,4.0])
steps = 80
lr = 0.01

paths = {}

# Gradient Descent
w = init.copy()
path=[w.copy()]
for _ in range(steps):
    w = GD(w, lr)
    path.append(w.copy())
paths["GD"]=np.array(path)

# Nesterov
w=init.copy(); v=np.zeros(2)
path=[w.copy()]
for _ in range(steps):
    w,v=Nesterov(w,v,lr)
    path.append(w.copy())
paths["Nesterov"]=np.array(path)

# Adagrad
w=init.copy(); G=np.zeros(2)
path=[w.copy()]
for _ in range(steps):
    w,G=Adagrad(w,G,0.4)
    path.append(w.copy())
paths["Adagrad"]=np.array(path)

# RMSProp
w=init.copy(); Eg=np.zeros(2)
path=[w.copy()]
for _ in range(steps):
    w,Eg=RMSProp(w,Eg,lr)
    path.append(w.copy())
paths["RMSProp"]=np.array(path)

# Adam
w=init.copy(); m=v=np.zeros(2)
path=[w.copy()]
for t in range(1,steps+1):
    w,m,v=Adam(w,m,v,t,0.05)
```

```
        path.append(w.copy())
    paths["Adam"]=np.array(path)

    # grid for surface
    x = np.linspace(-6,6,200)
    y = np.linspace(-6,6,200)
    X,Y = np.meshgrid(x,y)
    Z = J([X,Y])


    plt.figure(figsize=(7,6))
    plt.contour(X,Y,Z,40)

    for name, p in paths.items():
        plt.plot(p[:,0], p[:,1], label=name)

    plt.legend()
    plt.title("2D Contour Paths of Optimizers")
    plt.xlabel("w1")
    plt.ylabel("w2")
    plt.show()


    fig = plt.figure(figsize=(8,6))
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_surface(X,Y,Z,cmap=cm.viridis,alpha=0.6)

    for name, p in paths.items():
        z = [J(w) for w in p]
        ax.plot(p[:,0], p[:,1], z, label=name)

    ax.set_title("3D Surface Optimizer Paths")
    ax.set_xlabel("w1")
    ax.set_ylabel("w2")
    ax.set_zlabel("Loss")
    ax.legend()

    plt.show()
```

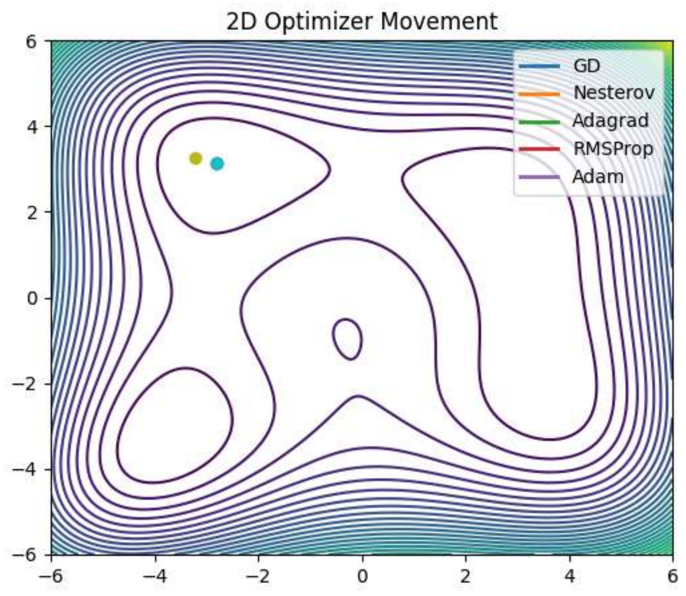


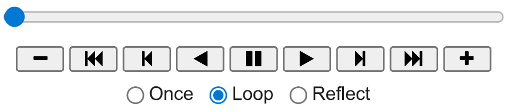


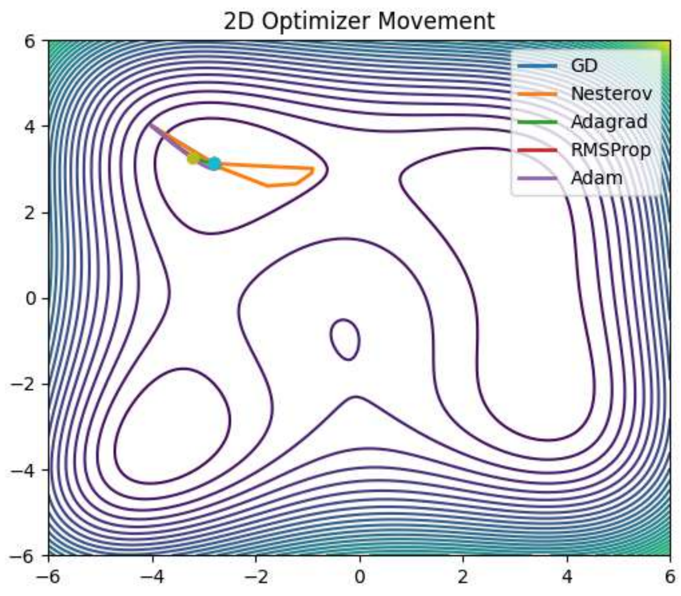
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation, cm
from IPython.display import HTML

# Himmelblau function
def J(w):
    w1, w2 = w
    return (w1**2 + w2 - 11)**2 + (w1 + w2**2 - 7)**2

# gradient
def grad_J(w):
    w1, w2 = w
    dw1 = 4*w1*(w1**2 + w2 - 11) + 2*(w1 + w2**2 - 7)
    dw2 = 2*(w1**2 + w2 - 11) + 4*w2*(w1 + w2**2 - 7)
    return np.array([dw1, dw2])


def GD(w, lr): return w - lr*grad_J(w)

def Nesterov(w, v, lr, gamma=0.9):
    look = w - gamma*v
    g = grad_J(look)
    v = gamma*v + lr*g
    return w - v, v
```

```
def Adagrad(w, G, lr):
    g = grad_J(w)
    G += g**2
    w -= lr*g/(np.sqrt(G)+1e-8)
    return w, G

def RMSProp(w, Eg, lr, beta=0.9):
    g = grad_J(w)
    Eg = beta*Eg + (1-beta)*(g**2)
    w -= lr*g/(np.sqrt(Eg)+1e-8)
    return w, Eg

def Adam(w, m, v, t, lr):
    g = grad_J(w)
    m = 0.9*m + 0.1*g
    v = 0.999*v + 0.001*(g**2)
    mhat = m/(1-0.9**t)
    vhat = v/(1-0.999**t)
    w -= lr*mhat/(np.sqrt(vhat)+1e-8)
    return w, m, v

# run optimizers
init = np.array([-4.0,4.0])
steps = 80
lr = 0.01

paths = {}

# GD
w=init.copy(); path=[w.copy()]
for _ in range(steps):
    w=GD(w,lr); path.append(w.copy())
paths["GD"]=np.array(path)

# Nesterov
w=init.copy(); v=np.zeros(2); path=[w.copy()]
for _ in range(steps):
    w,v=Nesterov(w,v,lr); path.append(w.copy())
paths["Nesterov"]=np.array(path)

# Adagrad
w=init.copy(); G=np.zeros(2); path=[w.copy()]
for _ in range(steps):
    w,G=Adagrad(w,G,0.4); path.append(w.copy())
paths["Adagrad"]=np.array(path)

# RMSProp
w=init.copy(); Eg=np.zeros(2); path=[w.copy()]
for _ in range(steps):
    w,Eg=RMSProp(w,Eg,lr); path.append(w.copy())
paths["RMSProp"]=np.array(path)

# Adam
w=init.copy(); m=v=np.zeros(2); path=[w.copy()]
for t in range(1,steps+1):
    w,m,v=Adam(w,m,v,t,0.05); path.append(w.copy())
paths["Adam"]=np.array(path)

# grid
x=np.linspace(-6,6,200)
y=np.linspace(-6,6,200)
X,Y=np.meshgrid(x,y)
Z=J([X,Y])


fig2d, ax2d = plt.subplots(figsize=(6,5))
ax2d.contour(X,Y,Z,40)

lines = {n: ax2d.plot([],[],lw=2,label=n)[0] for n in paths}
points = {n: ax2d.plot([],[],'o')[0] for n in paths}

ax2d.legend()
ax2d.set_title("2D Optimizer Movement")

def animate2d(i):
    for name in paths:
        p = paths[name]
        if i < len(p):
            lines[name].set_data(p[:i,0], p[:i,1])
            points[name].set_data([p[i-1,0]], [p[i-1,1]])
    return list(lines.values()) + list(points.values())
```

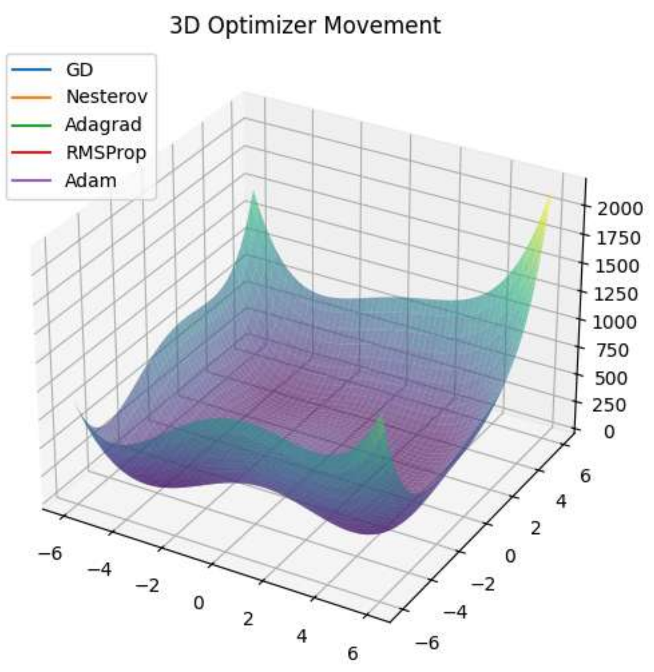
```
ani2d = animation.FuncAnimation(
    fig2d, animate2d,
    frames=steps,
    interval=120,
    blit=True
)

HTML(ani2d.to_jshtml())
```







```
fig3d = plt.figure(figsize=(7,6))
ax3d = fig3d.add_subplot(111, projection='3d')

def animate3d(i):
    ax3d.clear()
    ax3d.plot_surface(X,Y,Z,cmap=cm.viridis,alpha=0.6)

    for name,p in paths.items():
        if i < len(p):
            px=p[:i,0]
            py=p[:i,1]
            pz=[J(pt) for pt in p[:i]]
            ax3d.plot(px,py,pz,label=name)
        ax3d.set_title("3D Optimizer Movement")
        ax3d.legend()

    ani3d = animation.FuncAnimation(
        fig3d, animate3d,
        frames=steps,
        interval=120
    )

    HTML(ani3d.to_jshtml())
```


3D Optimizer Movement
GD
Nesterov
Adagrad
RMSProp
Adam