In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

np.random.seed(42)

# Create 60 data points
x = np.linspace(1, 10, 60)
y_true = 5*x + 3

# Manual standardization
x_mean = np.mean(x)
x_std = np.std(x)
x_stdized = (x - x_mean) / x_std
```

In [3]:
```
# Normal noise
noise_normal = np.random.normal(0, 2, len(x))
y_normal = y_true + noise_normal

# t-distribution noise
noise_t = np.random.standard_t(df=3, size=len(x))
y_t = y_true + noise_t
```

In [4]:
```
def stats(data):
    return np.mean(data), np.var(data), np.std(data)

print("Normal Noise:", stats(noise_normal))
print("t Noise:", stats(noise_t))
```

```
Normal Noise: (-0.30930936614097987, 3.2466151327875243, 1.8018365999134118)
t Noise: (-0.02722509804638934, 1.0729480646977867, 1.035832063945593)
```

In [5]:
```
def error_bucket(errors):
    buckets = []
    for e in errors:
        if abs(e) < 1:
            buckets.append(0)
        elif abs(e) < 3:
            buckets.append(1)
        else:
```

```
            buckets.append(2)
    return buckets

def pmf(buckets):
    total = len(buckets)
    return {i: buckets.count(i)/total for i in [0,1,2]}
```

In [6]:
```
X = np.column_stack((np.ones(len(x_stdized)), x_stdized))
theta = np.linalg.inv(X.T @ X) @ X.T @ y_normal

c = theta[0]
m = theta[1]

y_pred = X @ theta
residuals = y_normal - y_pred
```

In [7]:
```
# Degree 2
X2 = np.column_stack((np.ones(len(x_stdized)),
                      x_stdized,
                      x_stdized**2))

theta2 = np.linalg.inv(X2.T @ X2) @ X2.T @ y_normal
y_pred2 = X2 @ theta2

# Degree 3
X3 = np.column_stack((np.ones(len(x_stdized)),
                      x_stdized,
                      x_stdized**2,
                      x_stdized**3))

theta3 = np.linalg.inv(X3.T @ X3) @ X3.T @ y_normal
y_pred3 = X3 @ theta3
```

In [8]:
```
split = int(0.8 * len(x))
X_train, X_test = X[:split], X[split:]
y_train, y_test = y_normal[:split], y_normal[split:]
```

In [9]:
```
def MAE(y, yhat):
    return np.mean(np.abs(y - yhat))

def MSE(y, yhat):
```

```
    return np.mean((y - yhat)**2)

def RMSE(y, yhat):
    return np.sqrt(MSE(y, yhat))
```

In [10]:
```
m_values = np.linspace(-10, 10, 50)
c_values = np.linspace(-10, 10, 50)

cost_surface = []

for m in m_values:
    for c in c_values:
        y_hat = m*x_stdized + c
        cost = np.mean((y_normal - y_hat)**2)
        cost_surface.append(cost)
```

In [11]:
```
m, c = 0, 0
lr = 0.01
n = len(x_stdized)

for i in range(100):
    y_hat = m*x_stdized + c
    dm = (-2/n)*np.sum(x_stdized*(y_normal - y_hat))
    dc = (-2/n)*np.sum(y_normal - y_hat)

    m -= lr*dm
    c -= lr*dc

    cost = np.mean((y_normal - y_hat)**2)
    print(i, cost)
```

```
0 1088.5411602279762
1 1045.5634720023072
2 1004.2877002303748
3 964.6464490206106
4 926.5749913587533
5 890.0111634203056
6 854.8952630682201
7 821.1699523700776
8 788.7801639755814
9 757.6730112015072
10 727.7977016772863
11 699.1054544102248
12 671.5494201349386
13 645.0846048169539
14 619.6677961855615
15 595.257493175972
16 571.8138381655623
17 549.2985518935649
18 527.6748709579385
19 506.907487787363
20 486.9624929903422
21 467.8073199872835
22 449.41069183514594
23 431.742570157833
24 414.7741060989417
25 398.4775932167824
26 382.8264222447566
27 367.79503764322305
28 353.3588958719103
29 339.4944253147415
30 326.17898779163653
31 313.3908415944466
32 301.10910598666527
33 289.31372710895215
34 277.9854452347965
35 267.1057633228574
36 256.656916814631
37 246.62184462813056
38 236.9841613002154
39 227.7281302320857
40 218.838637994254
41 210.3011696490404
```

```
42 202.10178505029725
43 194.22709608166434
44 186.6642447961893
45 179.4008824216191
46 172.4251491970818
47 165.72565500823617
48 159.29146078926888
49 153.11206066137265
50 147.17736477854118
51 141.47768285266983
52 136.00370833106294
53 130.74650320051168
54 125.69748339313024
55 120.84840477012112
56 116.19134966058317
57 111.71871393338287
58 107.42319458097973
59 103.29777779493178
60 99.33572751361127
61 95.53057442343109
62 91.87610539562209
63 88.36635334131428
64 84.99558746835706
65 81.75830392396895
66 78.64921680793863
67 75.6632495417031
68 72.79552657921047
69 70.04136544603259
70 67.39626909372852
71 64.85591855697572
72 62.41616590147833
73 60.07302745113859
74 57.822677283432334
75 55.66144098236724
76 53.58578963882432
77 51.59233408848571
78 49.677819377940516
79 47.8391194499329
80 46.073232039074384
81 44.37727376968587
82 42.74847544776513
83 41.18417753939245
```

```
84 39.68182582819137
85 38.23896724475382
86 36.85324586122043
87 35.52239904447495
88 34.24425376167256
89 33.016723032069166
90 31.837802519358046
91 30.70556725895032
92 29.618168514854727
93 28.5738307610253
94 27.57084878224754
95 26.607584889829358
96 25.68246624755093
97 24.793982303506777
98 23.94068232364674
99 23.121173022989165
```

In [12]:

```
for epoch in range(20):
    for i in range(n):
        y_hat = m*x_stdized[i] + c
        dm = -2*x_stdized[i]*(y_normal[i] - y_hat)
        dc = -2*(y_normal[i] - y_hat)

        m -= lr*dm
        c -= lr*dc
```

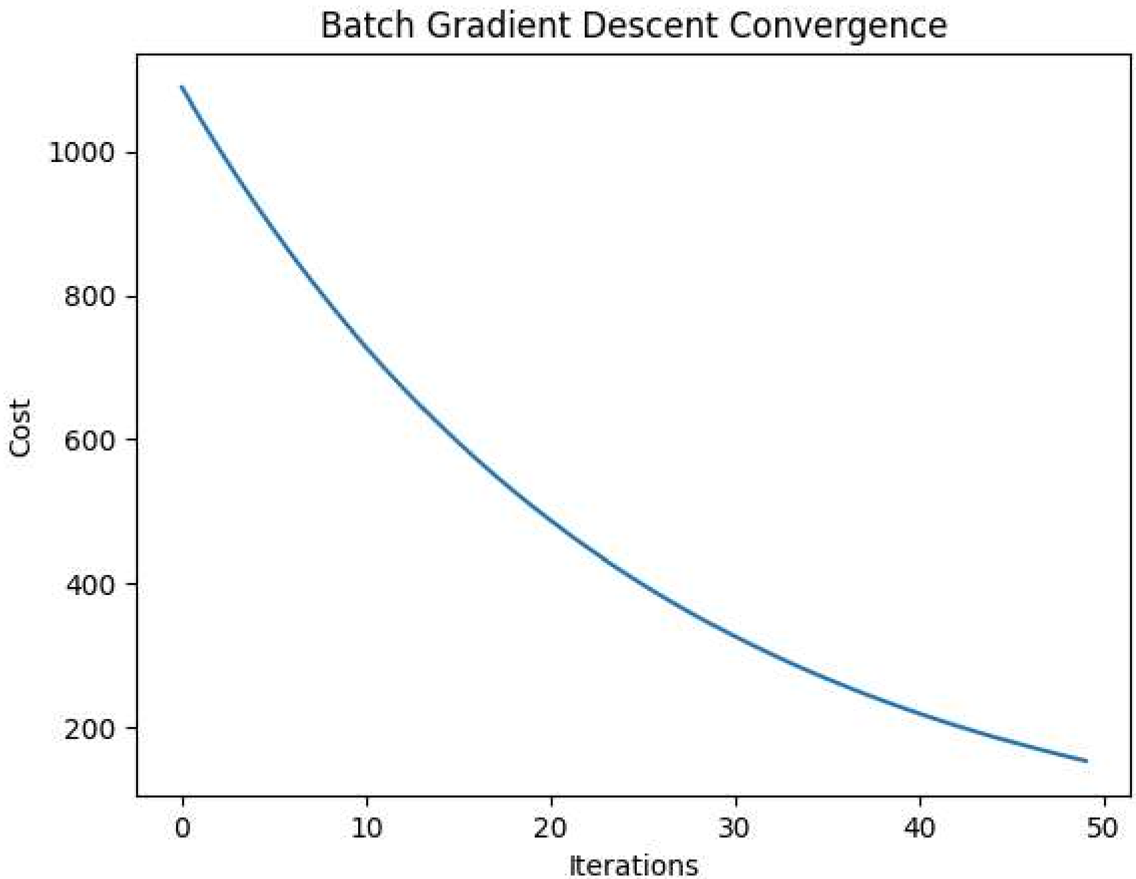
In [13]:

```
batch_size = 4

for epoch in range(50):
    for i in range(0, n, batch_size):
        xb = x_stdized[i:i+batch_size]
        yb = y_normal[i:i+batch_size]

        y_hat = m*xb + c

        dm = (-2/len(xb))*np.sum(xb*(yb - y_hat))
        dc = (-2/len(xb))*np.sum(yb - y_hat)

        m -= lr*dm
        c -= lr*dc
```

In [14]:
```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)

# Generate dataset
x = np.linspace(1, 10, 60)
y_true = 5*x + 3

# Standardize x manually
x_mean = np.mean(x)
x_std = np.std(x)
x_stdized = (x - x_mean) / x_std

# Add normal noise
noise = np.random.normal(0, 2, len(x))
y = y_true + noise

n = len(x_stdized)
```

In [15]:
```
def batch_gd(x, y, lr=0.01, epochs=50):
    m, c = 0, 0
    cost_history = []

    for epoch in range(epochs):
        y_hat = m*x + c

        dm = (-2/n)*np.sum(x*(y - y_hat))
        dc = (-2/n)*np.sum(y - y_hat)

        m -= lr*dm
        c -= lr*dc

        cost = np.mean((y - y_hat)**2)
        cost_history.append(cost)

    return m, c, cost_history

m_b, c_b, cost_b = batch_gd(x_stdized, y)
```

In [16]:
```
plt.figure()
plt.plot(cost_b)
plt.title("Batch Gradient Descent Convergence")
plt.xlabel("Iterations")
plt.ylabel("Cost")
plt.show()
```

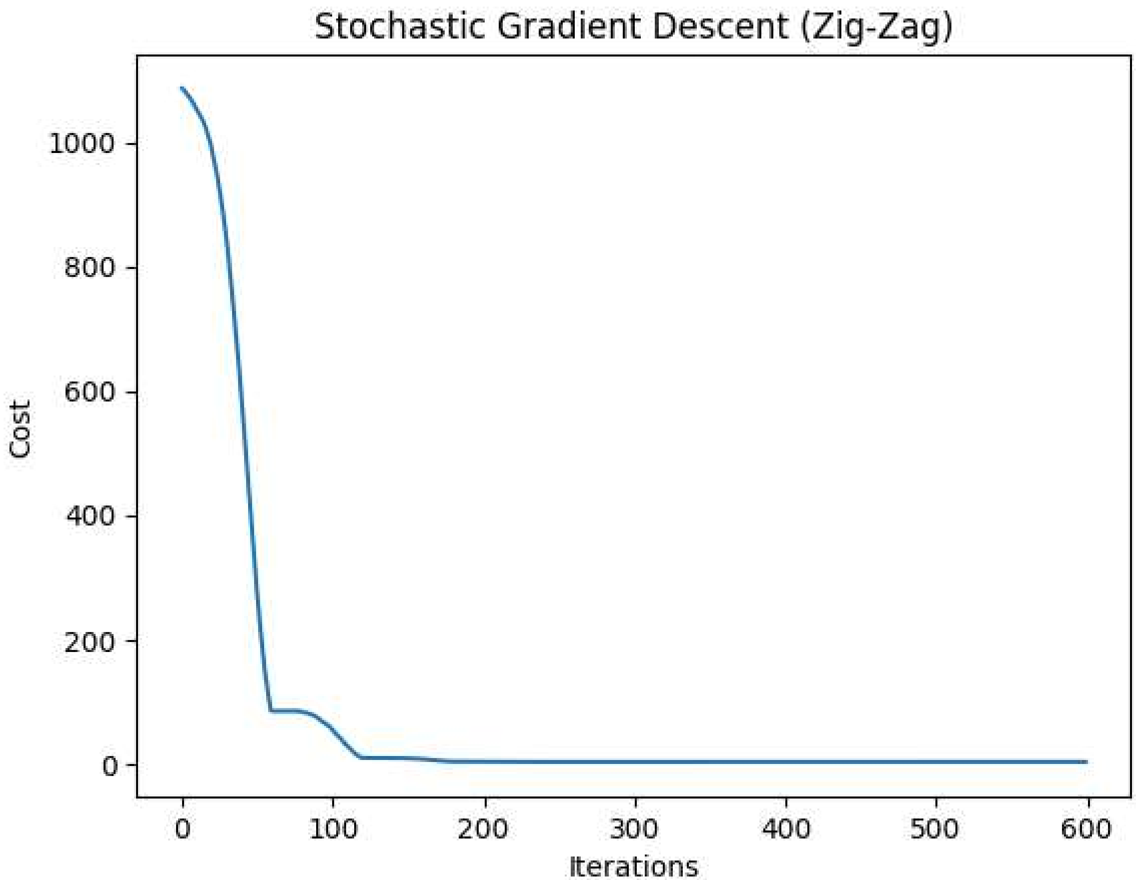


In [17]:
```
def sgd(x, y, lr=0.01, epochs=10):
    m, c = 0, 0
    cost_history = []

    for epoch in range(epochs):
        for i in range(n):
            y_hat = m*x[i] + c
```

```
            dm = -2*x[i]*(y[i] - y_hat)
            dc = -2*(y[i] - y_hat)

            m -= lr*dm
            c -= lr*dc

            cost = np.mean((y - (m*x + c))**2)
            cost_history.append(cost)

    return m, c, cost_history

m_s, c_s, cost_s = sgd(x_stdized, y)
```

In [18]:

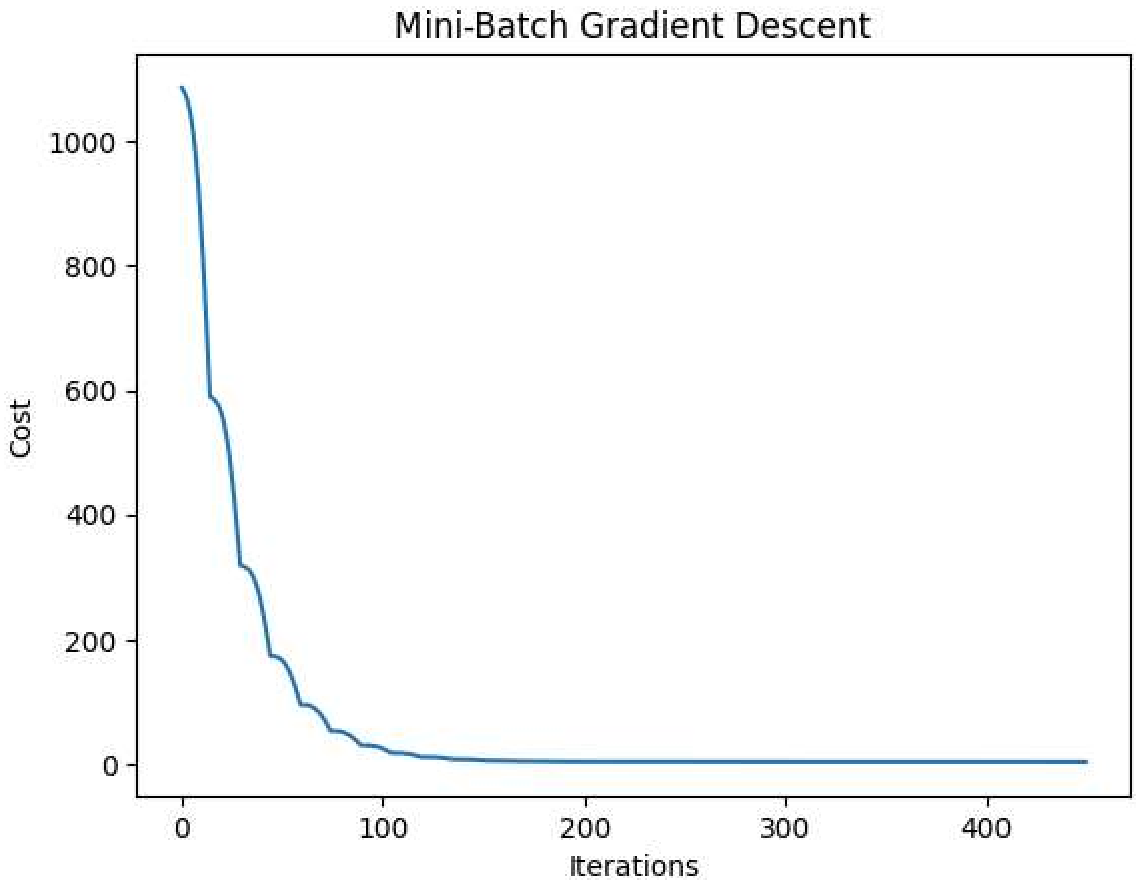
```
plt.figure()
plt.plot(cost_s)
plt.title("Stochastic Gradient Descent (Zig-Zag)")
plt.xlabel("Iterations")
plt.ylabel("Cost")
plt.show()
```



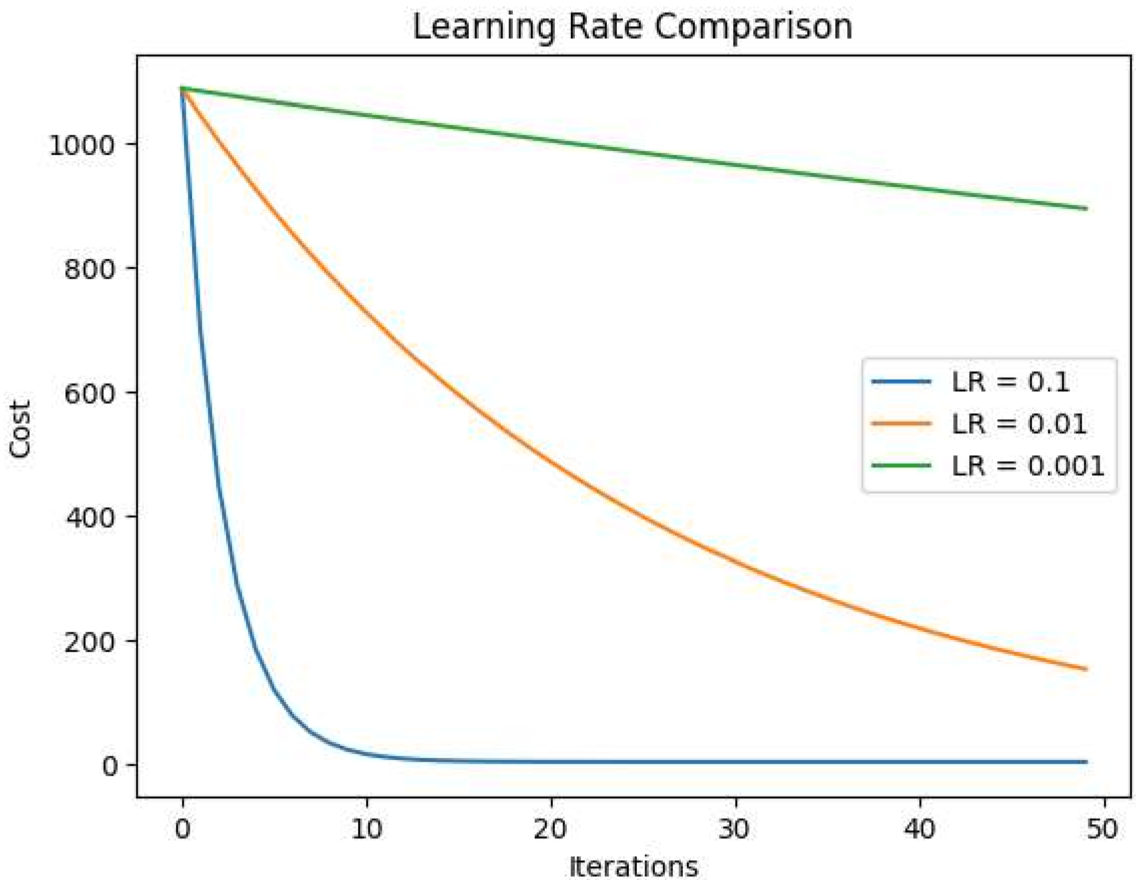
In [19]:
```
def mini_batch_gd(x, y, lr=0.01, epochs=30, batch_size=4):
    m, c = 0, 0
    cost_history = []

    for epoch in range(epochs):
        for i in range(0, n, batch_size):
            xb = x[i:i+batch_size]
            yb = y[i:i+batch_size]

            y_hat = m*xb + c

            dm = (-2/len(xb))*np.sum(xb*(yb - y_hat))
            dc = (-2/len(xb))*np.sum(yb - y_hat)
```

```
            m -= lr*dm
            c -= lr*dc

            cost = np.mean((y - (m*x + c))**2)
            cost_history.append(cost)

    return m, c, cost_history

m_mb, c_mb, cost_mb = mini_batch_gd(x_stdized, y)
```

In [20]:
```
plt.figure()
plt.plot(cost_mb)
plt.title("Mini-Batch Gradient Descent")
plt.xlabel("Iterations")
plt.ylabel("Cost")
plt.show()
```



In [21]:
```
learning_rates = [0.1, 0.01, 0.001]

plt.figure()

for lr in learning_rates:
    _, _, cost_lr = batch_gd(x_stdized, y, lr=lr, epochs=50)
    plt.plot(cost_lr, label=f"LR = {lr}")

plt.title("Learning Rate Comparison")
plt.xlabel("Iterations")
plt.ylabel("Cost")
plt.legend()
plt.show()
```

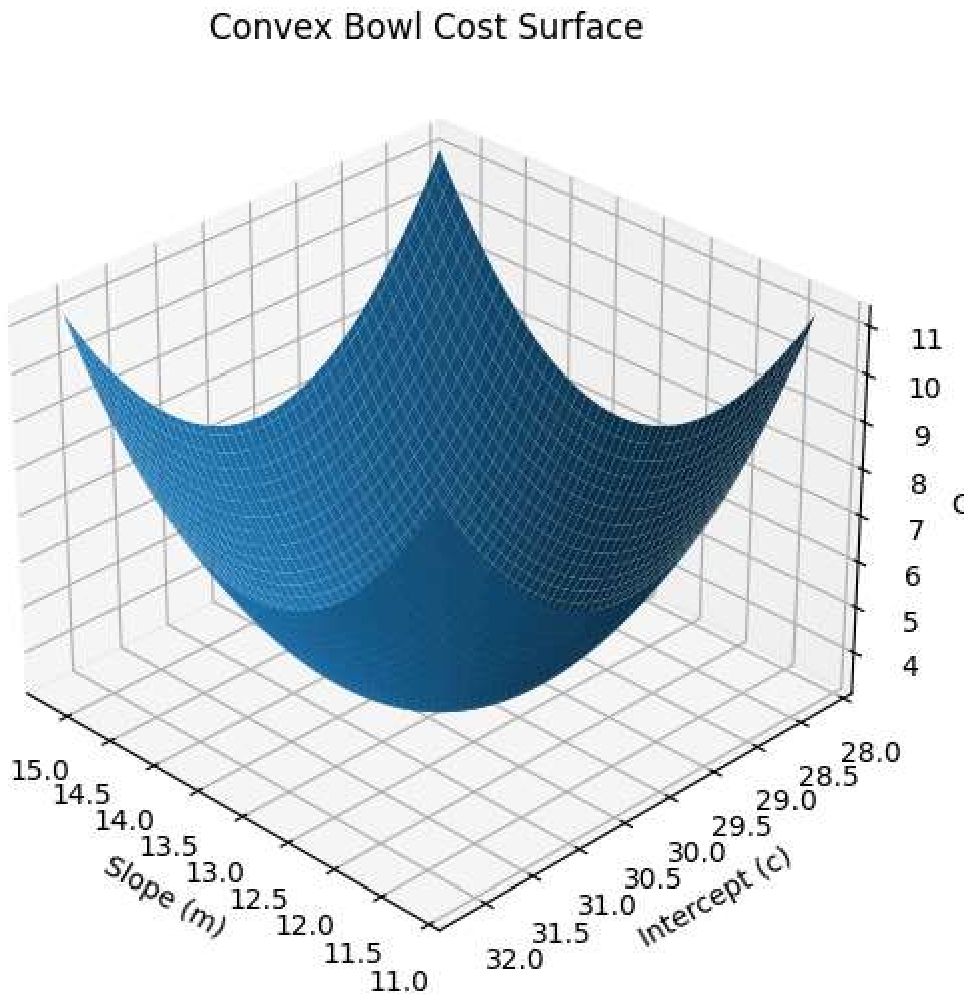


In [24]:

```
# Compute optimal m and c using Normal Equation
X = np.column_stack((np.ones(len(x_stdized)), x_stdized))
theta = np.linalg.inv(X.T @ X) @ X.T @ y

c_opt = theta[0]
m_opt = theta[1]

print("Optimal m:", m_opt)
print("Optimal c:", c_opt)
```

```
Optimal m: 13.183981047638127
Optimal c: 30.190690633859028
```

In [25]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Create small range around true minimum
m_values = np.linspace(m_opt - 2, m_opt + 2, 100)
c_values = np.linspace(c_opt - 2, c_opt + 2, 100)

M, C = np.meshgrid(m_values, c_values)

Cost = np.zeros_like(M)

# Compute cost surface
for i in range(M.shape[0]):
    for j in range(M.shape[1]):
        y_hat = M[i,j] * x_stdized + C[i,j]
        Cost[i,j] = np.mean((y - y_hat)**2)

# 3D Plot
fig = plt.figure(figsize=(8,6))
ax = fig.add_subplot(111, projection='3d')

ax.plot_surface(M, C, Cost)

ax.set_xlabel("Slope (m)")
ax.set_ylabel("Intercept (c)")
ax.set_zlabel("Cost")
ax.set_title("Convex Bowl Cost Surface")

# Better viewing angle
ax.view_init(elev=30, azim=135)

plt.show()
```



Convex Bowl Cost Surface
11
10
9
8
7
6
5
4
15.0
14.5
14.0
13.5
13.0
12.5
12.0
11.5
11.0
Slope (m)
28.0
28.5
29.0
29.5
30.0
30.5
31.0
31.5
32.0
Intercept (c)

In [ ]: