21K-4827

Convex function

Definition:

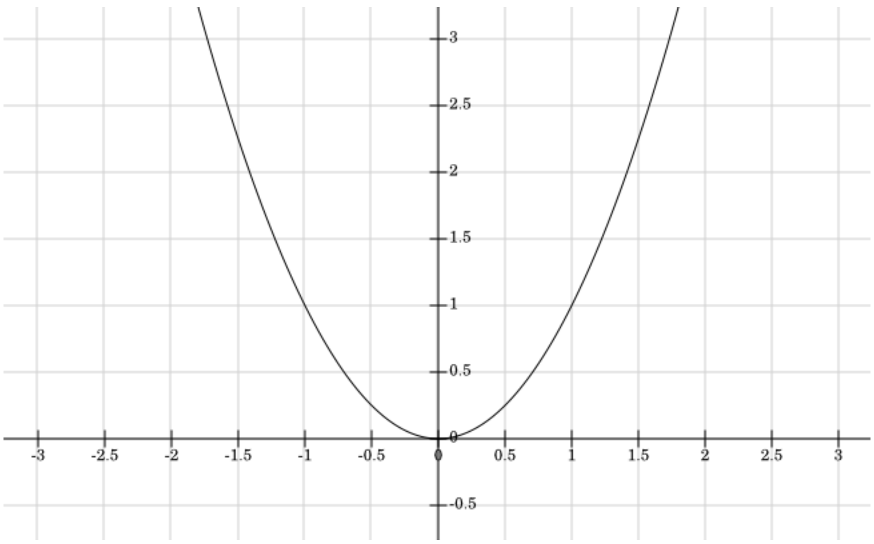
A function f(x) is said to be convex if, for any two points (x1, f(x1)) and (x2, f(x2)) on the graph of the function, the line segment connecting these two points lies above or on the graph. Mathematically, it can be expressed as:

$f(\lambda x1 + (1 - \lambda)x2) \leq \lambda f(x1) + (1 - \lambda)f(x2)$

where x1, x2 are points in the function's domain, and λ is a scalar value between 0 and 1.

Example Equation:

Let's consider the function f(x) = x^2. This is a simple example of a convex function. For any two points on the graph, the line segment connecting them lies entirely above or on the curve.

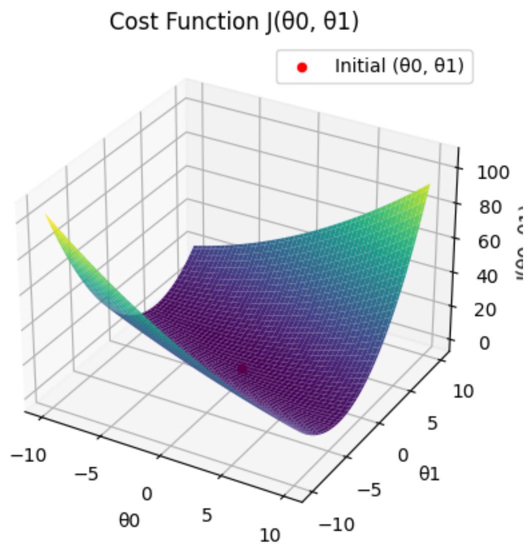



```
import numpy as np

def gradient_descent(x,y):
    m_curr = b_curr = 0
    iterations = 2000
    n = len(x)
    learning_rate = 0.01

    for i in range(iterations):
        y_predicted = m_curr * x + b_curr
        cost = (1/n) * sum([val**2 for val in (y-y_predicted)])
        md = -(2/n)*sum(x*(y-y_predicted))
        bd = -(2/n)*sum(y-y_predicted)
        m_curr = m_curr - learning_rate * md
        b_curr = b_curr - learning_rate * bd
        print ("m {}, b {}, J {} iteration {}".format(m_curr,b_curr,cost, i))

for i in range(10):
  x = np.random.rand(10, 1)
  y = 2 * x + np.random.randn(10, 1)

gradient_descent(x,y)
```

```
m [3.56405334], b [-0.30523142], J [0.43835395] iteration 1948
m [3.56426613], b [-0.30534477], J [0.43834812] iteration 1949
m [3.56447859], b [-0.30545795], J [0.43834231] iteration 1950
m [3.5646907], b [-0.30557094], J [0.43833652] iteration 1951
m [3.56490248], b [-0.30568375], J [0.43833075] iteration 1952
m [3.56511391], b [-0.30579638], J [0.438325] iteration 1953
m [3.56532501], b [-0.30590882], J [0.43831926] iteration 1954
m [3.56553576], b [-0.30602109], J [0.43831355] iteration 1955
m [3.56574618], b [-0.30613318], J [0.43830785] iteration 1956
m [3.56595626], b [-0.30624508], J [0.43830217] iteration 1957
m [3.566166], b [-0.30635681], J [0.43829651] iteration 1958
m [3.56637541], b [-0.30646836], J [0.43829086] iteration 1959
m [3.56658448], b [-0.30657973], J [0.43828524] iteration 1960
m [3.56679322], b [-0.30669092], J [0.43827963] iteration 1961
m [3.56700161], b [-0.30680193], J [0.43827404] iteration 1962
m [3.56720968], b [-0.30691276], J [0.43826847] iteration 1963
m [3.56741741], b [-0.30702342], J [0.43826292] iteration 1964
m [3.56762481], b [-0.30713389], J [0.43825738] iteration 1965
m [3.56783187], b [-0.30724419], J [0.43825187] iteration 1966
m [3.5680386], b [-0.30735432], J [0.43824637] iteration 1967
m [3.568245], b [-0.30746426], J [0.43824089] iteration 1968
m [3.56845107], b [-0.30757403], J [0.43823542] iteration 1969
m [3.56865681], b [-0.30768363], J [0.43822997] iteration 1970
m [3.56886222], b [-0.30779305], J [0.43822454] iteration 1971
m [3.5690673], b [-0.30790229], J [0.43821913] iteration 1972
m [3.56927205], b [-0.30801135], J [0.43821374] iteration 1973
m [3.56947647], b [-0.30812025], J [0.43820836] iteration 1974
m [3.56968056], b [-0.30822896], J [0.438203] iteration 1975
m [3.56988432], b [-0.30833751], J [0.43819766] iteration 1976
m [3.57008776], b [-0.30844587], J [0.43819233] iteration 1977
m [3.57029087], b [-0.30855407], J [0.43818702] iteration 1978
m [3.57049366], b [-0.30866209], J [0.43818173] iteration 1979
m [3.57069612], b [-0.30876994], J [0.43817646] iteration 1980
m [3.57089825], b [-0.30887761], J [0.4381712] iteration 1981
m [3.57110006], b [-0.30898511], J [0.43816596] iteration 1982
m [3.57130155], b [-0.30909244], J [0.43816073] iteration 1983
m [3.57150271], b [-0.3091996], J [0.43815552] iteration 1984
m [3.57170355], b [-0.30930658], J [0.43815033] iteration 1985
m [3.57190407], b [-0.30941339], J [0.43814516] iteration 1986
m [3.57210426], b [-0.30952004], J [0.43814] iteration 1987
m [3.57230414], b [-0.30962651], J [0.43813486] iteration 1988
m [3.57250369], b [-0.30973281], J [0.43812974] iteration 1989
m [3.57270292], b [-0.30983893], J [0.43812463] iteration 1990
m [3.57290184], b [-0.30994489], J [0.43811954] iteration 1991
m [3.57310043], b [-0.31005068], J [0.43811446] iteration 1992
m [3.5732987], b [-0.3101563], J [0.4381094] iteration 1993
m [3.57349666], b [-0.31026175], J [0.43810436] iteration 1994
m [3.5736943], b [-0.31036703], J [0.43809933] iteration 1995
m [3.57389162], b [-0.31047214], J [0.43809432] iteration 1996
m [3.57408863], b [-0.31057708], J [0.43808933] iteration 1997
m [3.57428532], b [-0.31068185], J [0.43808435] iteration 1998
m [3.57448169], b [-0.31078646], J [0.43807939] iteration 1999
```

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

for i in range(10):
  x = np.random.rand(10, 1)
  y = 2 * x + np.random.randn(10, 1)

def cost_function(theta0, theta1, x, y):
    m = len(x)
    h = theta0 + theta1 * x
    J = (1 / (2 * m)) * np.sum((h - y) ** 2)
    return J

# Step 6a: Plot of J(θ0, θ1) with initial values
theta0_initial = 0
theta1_initial = 0
J_initial = cost_function(theta0_initial, theta1_initial, x, y)

# Plotting J(θ0, θ1)
theta0_vals = np.linspace(-10, 10, 100)
theta1_vals = np.linspace(-10, 10, 100)
J_vals = np.zeros((len(theta0_vals), len(theta1_vals)))

for i, theta0 in enumerate(theta0_vals):
    for j, theta1 in enumerate(theta1_vals):
        J_vals[i, j] = cost_function(theta0, theta1, x, y)

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

```
ax = fig.add_subplot(111, projection='3d')
theta0_grid, theta1_grid = np.meshgrid(theta0_vals, theta1_vals)
ax.plot_surface(theta0_grid, theta1_grid, J_vals, cmap='viridis')
ax.set_xlabel('θ0')
ax.set_ylabel('θ1')
ax.set_zlabel('J(θ0, θ1)')
ax.scatter(theta0_initial, theta1_initial, J_initial, color='red', label='Initial (θ0, θ1)')
ax.legend()
plt.title('Cost Function J(θ0, θ1)')
plt.show()
```




```
# Gradient Descent Algorithm
alpha = 0.01  # learning rate
iterations = 2000  # number of iterations

theta0 = 0  # initial value for θ0
theta1 = 0  # initial value for θ1
m = len(x)  # number of data points

# Lists for storing the values at each iteration
cost_history = []
theta0_history = []
theta1_history = []

for i in range(iterations):
    h = theta0 + theta1 * x
    theta0 -= alpha * (1 / m) * np.sum(h - y)
    theta1 -= alpha * (1 / m) * np.sum((h - y) * x)

    # Calculate and store the cost for each iteration
    cost = cost_function(theta0, theta1, x, y)
    cost_history.append(cost)
    theta0_history.append(theta0)
    theta1_history.append(theta1)

    # Print the working of each iteration
    print(f"Iteration: {i+1}")
    print(f"hypothesis = {h}")
    print(f"theta0 = {theta0}, theta1 = {theta1}")
    print(f"Cost = {cost}")
    print()
# Step 6b: Table with Columns J, θ0, and θ1
print("Iteration\tJ\t\tθ0\t\tθ1")
for i in range(iterations):
    print(f"{i+1}\t\t{cost_history[i]}\t{theta0_history[i]}\t{theta1_history[i]}")

# Step 6b: Plot of J(θ0, θ1) with value of J at each iteration
plt.plot(range(iterations), cost_history)
plt.xlabel('Iteration')
plt.ylabel('J(θ0, θ1)')
plt.title('Cost Function vs. Iteration')
plt.show()
```

```
# Step 7: Scatter Plot of x, y and line y = θ0 + θ1 * x
plt.scatter(x, y, label='Data Points')
plt.plot(x,theta0 + theta1 * x, color='red', label='Best Fit Line')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Scatter Plot of Data Points and Best Fit Line')
plt.legend()
plt.show()
```

1996	0.24759703119505028	0.6226086806353952	-0.5413015905090417
1997	0.2475918535353789	0.6226930957980356	-0.5415129249642514
1998	0.24758668050360977	0.6227774732262052	-0.5417241649507357
1999	0.24758151209560633	0.622861812936772	-0.5419353105107232
2000	0.2475763483072357	0.622946114946596	-0.5421463616864234