```python
from tabulate import tabulate
import numpy as np

def gradient_descent(X, y, alpha, num_iters):
    m = y.size  # number of training examples
    theta = np.zeros(2)  # initialize parameters to zero
    data = []  # keep track of cost function values
    for i in range(num_iters):
        # Calculate hypothesis and cost function
        h = X.dot(theta)
        J = np.sum((h - y) ** 2) / (2 * m)
        partial_derivatives = X.T.dot(h - y) / m
        data.append([i+1, np.copy(theta), J, partial_derivatives])
        #update parameters
        theta -= alpha / m * (X.T.dot(h - y))

        # Print table of parameter values and partial derivatives of J
    print(tabulate(data, headers=['Iteration', 'theta', 'J', 'Partial derivatives'], tablefmt='github'))

    return

# Generate some example data points
x = np.array([4, 5, 6])
y = np.array([7, 8, 9])

# Add a column of ones for the intercept term
X = np.vstack((np.ones(x.size), x)).T

# Set the learning rate and number of iterations
alpha = 0.05
gradient_descent(X, y, alpha, num_iters=4)
```

```
|   Iteration | theta                     |         J | Partial derivatives          |
|-------------|---------------------------|-----------|------------------------------|
|           1 | [0. 0.]                   | 32.3333   | [ -8.         -40.66666667] |
|           2 | [0.4        2.03333333]   |  3.64981  | [ 2.56666667 13.52222222]   |
|           3 | [0.27166667 1.35722222]   |  0.486427 | [-0.94222222 -4.47296296]   |
|           4 | [0.31877778 1.58087037]   |  0.137364 | [0.22312963 1.50289506]     |
```