Lab 1 Solution: Linear Regression

```
def predict(x, w):
"""
Compute the prediction of a linear model.
Inputs:
x: np.ndarray input data of shape [num_samples, num_feat + 1]
w: np.ndarray weights of shape [num_feat + 1, 1]
Outputs:
h: np.ndarray predictions of shape [num_samples, 1]
"""
      h = np.dot(x,w)
      return h



def compute_cost(x, y, w):
"""
Inputs:
x: np.ndarray input data of shape [num_samples, num_feat + 1]
y: np.ndarray targets data of shape [num_samples, 1]
w: np.ndarray weights of shape [num_feat + 1, 1]
Outputs:
mse: scalar.
"""
      m = x.shape[0]
      mse = np.sum((y-predict(x,w))**2) / (2*m)
      return mse




def gradient_descent(x, y, w, learning_rate, num_iters):
"""
Inputs:
x: np.ndarray input data of shape [num_samples, num_feat + 1]
y: np.ndarray targets data of shape [num_samples, 1]
w: np.ndarray weights of shape [num_feat + 1, 1]
learning_rate: scalar, the learning rate.
num_iters: int, the number of iterations.
Outputs:
j_hist: list of loss values of shape [num_iters]
w_opt: [num_feat + 1, 1]
w_hist: [num_feat + 1, num_iters + 1]
"""
      n_sample, n_feat = len(x), len(w) - 1
      j_hist = np.zeros([num_iters])
      w_hist = w
```

```python
    for i in range(num_iters):
        h = np.dot(x,w)
        dw = np.dot((h-y).T,x).T / n_sample
        w = w - learning_rate*dw
        w_hist = np.append(w_hist,w,1)
        j_hist[i] = compute_cost(x,y,w)
    w_opt = w_hist[:,(np.argmin(j_hist)+1)].reshape(-1,1)
    return j_hist, w_opt, w_hist
```

```python
def compute_cost_multivariate(x, y, w):
"""
Inputs:
x: np.ndarray input data of shape [num_samples, num_feat + 1]
y: np.ndarray targets data of shape [num_samples, 1]
w: np.ndarray weights of shape [num_feat + 1, 1]
Outputs:
mse: scalar.
"""
    m = x.shape[0]
    res = predict(x,w) - y
    mse = np.dot(res.T, res)[0,0] / (2*m)
    return mse
```