Lab1 Solution: Linear Regression

```python
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.
    """
    h = predict(x, w) # Shape [num_samples, 1]
    mse = ((y - h) ** 2).sum() / (2 * x.shape[0])
    return mse
```

```python
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 + 1]
        w_opt: [num_feat + 1, 1]
        w_hist: [num_feat + 1, num_iters + 1]
    """
    num_samples, num_feat = len(x), len(w) - 1
    j_hist = np.zeros([num_iters])
    w_hist = np.zeros([num_feat + 1, num_iters + 1])
    w_hist[:, 0] = w.T

    for i in range(num_iters):
        h = np.dot(x, w) # Shape [num_samples, 1]

        # Compute gradient
        dw = 1 / num_samples * np.dot((h - y).T, x).T # Shape [num_feat + 1, 1]
        w = w - learning_rate * dw
        w_hist[:, i + 1] = w.T
```

```
        j_hist[i] = compute_cost(x, y, w)
    return j_hist, w, w_hist

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.
    """
    num_samples = len(x)
    h = predict(x, w)
    mse = np.dot((h - y).T, h - y)[0, 0] / (2 * num_samples)
    return mse
```