

# Univariate Linear Regression with Gradient Descent

From first principles to a clean NumPy implementation, plus learning-rate tuning and convergence checks

2018-12-01

## Table of contents

<b>1 Hypothesis and problem setup</b>	<b>1</b>
<b>2 Cost function</b>	<b>2</b>
<b>3 Gradient descent updates</b>	<b>2</b>
<b>4 Geometry</b>	<b>2</b>
<b>5 Reference implementation</b>	<b>2</b>
<b>6 Learning rate and convergence</b>	<b>3</b>
<b>7 Closed-form comparison</b>	<b>3</b>
<b>8 Common pitfalls</b>	<b>4</b>
<b>9 Checklist for reuse</b>	<b>4</b>
<b>10 Where to go next</b>	<b>4</b>

This walkthrough takes univariate linear regression from scratch to a working implementation. Each section pairs the core idea with the supporting math and code.

## 1 Hypothesis and problem setup

The model assumes the relationship between  $x$  and  $y$  is well described by a straight line whose slope and intercept are learned from data.

**Mathematics.**

$$\hat{y} = h_{\theta}(x) = \theta_0 + \theta_1 x,$$

where  $\theta_0$  is the intercept and  $\theta_1$  the slope.

## 2 Cost function

The mean squared error objective averages the squared residuals between predictions and observed targets, amplifying large mistakes and keeping the optimisation convex.

**Mathematics.**

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (\theta_0 + \theta_1 x^{(i)} - y^{(i)})^2.$$

The  $\frac{1}{2}$  factor simplifies derivatives.

---

## 3 Gradient descent updates

The gradient components measure how changes to the intercept or slope affect the total error, allowing gradient descent to update both parameters in a coordinated way.

**Mathematics.**

$$\frac{\partial J}{\partial \theta_0} = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}), \quad \frac{\partial J}{\partial \theta_1} = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x^{(i)}.$$

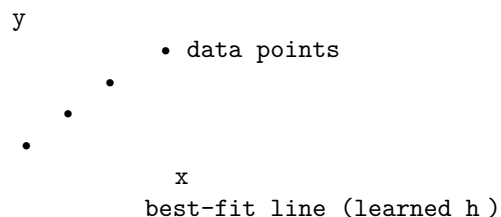
**Update rule.**

$$\theta_j \leftarrow \theta_j - \alpha \cdot \frac{\partial J}{\partial \theta_j}, \quad j \in \{0, 1\}.$$

---

## 4 Geometry

- $h_{\theta}(x)$  is a straight line in the  $(x, y)$  plane.
- The cost surface  $J(\theta_0, \theta_1)$  is convex with a unique minimum.
- The learning rate  $\alpha$  determines how quickly gradient descent approaches that minimum.



## 5 Reference implementation

```

import numpy as np

# Synthetic data: y = 1.5 + 2.0*x + noise
rng = np.random.default_rng(7)
m = 200
x = rng.uniform(-3, 3, size=m)
y = 1.5 + 2.0 * x + rng.normal(0, 0.6, size=m)

X = np.column_stack([np.ones_like(x), x])

theta = np.zeros(2)
alpha = 0.05
epochs = 2000

def cost(X, y, th):
    r = X @ th - y
    return 0.5 / len(y) * (r @ r)

history = []
for it in range(epochs):
    r = X @ theta - y
    grad = (X.T @ r) / m
    theta -= alpha * grad
    if it % 50 == 0:
        history.append(cost(X, y, theta))

print("theta:", theta)
print("final cost:", cost(X, y, theta))

```

---

## 6 Learning rate and convergence

Select  $\alpha$  so that the cost decreases steadily without divergence or oscillation.

**Checks.** - Plot cost vs. iterations; it should decline smoothly and flatten. - Inspect residuals occasionally; they should shrink and centre around zero.

---

## 7 Closed-form comparison

Comparing gradient-descent parameters with the closed-form solution verifies that the implementation and optimisation are consistent.

**Mathematics.**

$$\theta_1^* = \frac{\sum_i (x^{(i)} - \bar{x})(y^{(i)} - \bar{y})}{\sum_i (x^{(i)} - \bar{x})^2}, \quad \theta_0^* = \bar{y} - \theta_1^* \bar{x}.$$


---

## 8 Common pitfalls

- Forgetting the intercept term.
  - Using a learning rate that is too high or too low.
  - Scaling or centring with statistics computed on the full dataset (data leakage).
  - Stopping before the cost stabilises.
- 

## 9 Checklist for reuse

- ☐ Define  $h_{\theta}(x) = \theta_0 + \theta_1 x$ .
  - ☐ Use  $J(\theta) = \frac{1}{2m} \sum (h_{\theta}(x^{(i)}) - y^{(i)})^2$ .
  - ☐ Implement batch gradient descent with the derived gradients.
  - ☐ Tune  $\alpha$  on a log scale; monitor cost.
  - ☐ Validate with a holdout set or cross-validation.
  - ☐ (Optional) Cross-check with the normal equation.
- 

## 10 Where to go next

- Extend to multivariate regression and reuse the vectorised gradient descent from the companion walk-through.
- Add  $L_2$  regularisation and observe how coefficients shrink as you increase the penalty.
- Introduce polynomial features to capture non-linear trends, then compare training and validation error to watch for overfitting.