# 第二讲 线性回归

在第一讲中，我们整体介绍了机器学习的概念和四大要素，并通过一个简单的模型初步感知机器学习的基本思想。在第二讲中，我们将深入探讨这一模型：线性回归。这是机器学习中最基础、直观的模型之一，通过它，我们可以更清晰地理解机器学习的工作原理和优化过程，同时还会分析其在实际应用中的过拟合问题及其解决方法。

### 2.1 计算机科学中的回归问题

回归问题来源于人们对数据的测量和估计。人们在测量事物的时候因为客观条件所限，求得的都是测量值，而不是事物真实的值，为了能够得到真实值，无限次的进行测量，最后通过这些测量数据计算**回归到真实值**，这就是回归的由来。

回归模型的主要作用是作数据拟合，比如有两组数据：投入(包括广告、人员、设备)和营收利润，分别记作 x 和 y, 为了描述利润和投入之间的关系，我们可以拟合一个 y 关于 x 的线性方程，根据 x 估计 y 的目标值,如图 2.1 所示。

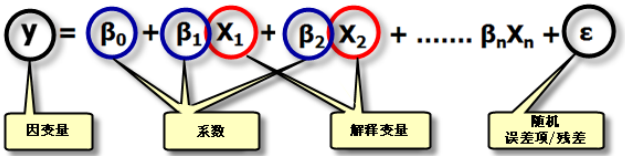



图 2.1 回归模型

这里的 $x_1$，…，$x_n$ 称为自变量或者解释变量，y 称为因变量，$\beta_0,\ldots\beta_n$ 为回归系数。

> 定义：回归是用来描述一个因变量和若干自变量之间关系的一种数学方法。

在定义回归模型的数学形式后，我们可以通过大量观测数据寻找数据属性之间的关系，刻画数据的潜在规律，这不仅能够帮助我们对已知数据进行分析，还可以对未知数据进行预测。

回归模型的实际应用包括：

- **预测**：通过建立回归模型，预测未来变量的趋势，例如房价预测、市场销售额预估等。
- **因子分析**：研究变量间的联系，解释现象背后的机制，例如分析火灾发生的潜在原因。
- **控制**：在机器人和自动驾驶等领域，基于当前状态预测最优的控制变量值。

例如，在房价预测中，自变量可能包括房屋面积、地段、周边设施等，而因变量则是房屋的价格。通过建立回归模型，我们可以通过输入这些因素估计房价。这类回归模型不仅能帮助我们发现数据间的规律，还为后续的决策提供支持。

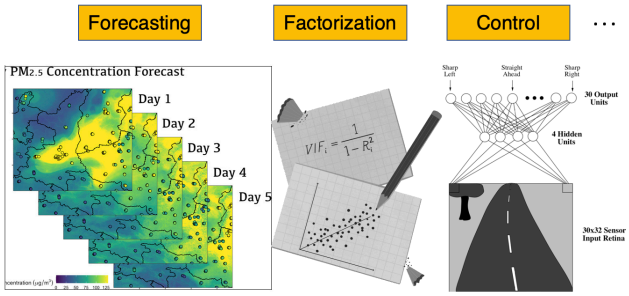



图 2.2 回归模型的应用

## 2.2 线性回归模型

所谓线性回归，就是用**线性函数**构造的回归模型。对于给定的数据集 $D=\{x^{(i)},y^{(i)}\}$,线性回归模型拟合以下线性函数：

$$y=f(x)=\mathbf{w}^{T}x+w_0$$

其中，w 是模型参数矩阵，也称为回归系数。$w_0$ 是一个常数偏置项。

在几何上，线性回归表示一条**线性超平面**。对于 d 维空间的数据，线性回归拟合一个 d+1 维的线性超平面。

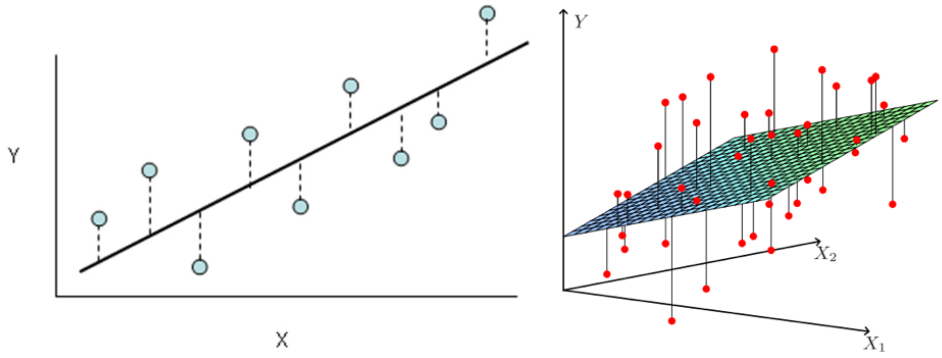



图 2.3 二维和三维线性回归模型

### 1. 模型结构

图 2.4 显示了机器学习视角下的线性回归模型结构。每个输入样本是一个 d 维向量 $x=(x_1,\ldots,x_d)$,通过一个线性函数 $f(x)=\mathbf{w}^{T}x+w_0$,输出对应的预测值 y。对于每个输入，同时会提供一个目标值 r,算法计算误差作为损失函数，供模型优化修正。

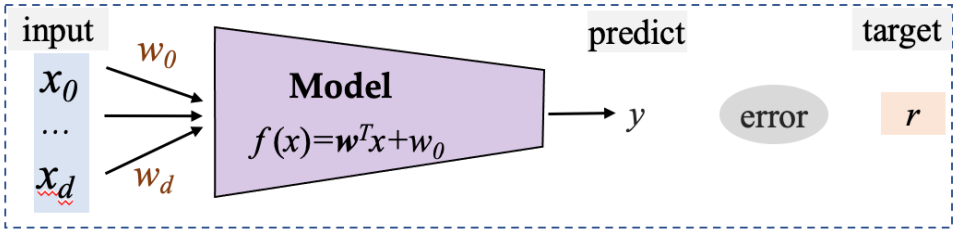



图 2.4 线性回归模型结构

线性回归模型的简单结构使其成为机器学习中的经典算法，同时也为理解更复杂的模型奠定了基础。

**2. 损失函数**

在线性回归中，我们常使用**均方误差（Mean Squared Error, MSE）**作为损失函数，它衡量模型预测值与实际目标值之间的偏差。对于数据集 $D = \{(x^{(1)}, r^{(1)}), \ldots, (x^{(N)}, r^{(N)})\}$，损失函数定义如下：

$$L(\boldsymbol{w}, w_0 \mid D) = \frac{1}{2N}\sum_{l=1}^{N}(r^{(l)}-y^{(l)})^2$$

该公式的数学意义是计算所有样本点预测值与真实值的平方误差的平均值，以平滑误差的影响。其中$(r-y)^2$代表每个样本点的误差。

**3. 优化算法**

损失函数是一个关于系数 w,$w_0$ 的打分函数。我们的目标是最小化上述损失函数：

$$\min_{w,\, w0} L(\mathbf{w}, w_0)$$

这是一个标准的无约束优化问题。根据高等数学的原理，可以通过对 L 求偏导数，并令导数为零来找到极小值点：

$$\frac{\partial L}{\partial \boldsymbol{w}} = 0, \quad \frac{\partial L}{\partial w_0} = 0$$

在简单场景下，这种解析解方法适用，但对于更复杂的损失函数或大规模数据集，这种方法可能计算成本过高，甚至无解。因此，我们通常采用**数值优化方法**，如梯度下降法。

梯度下降是一种迭代优化算法，它的核心思想是将参数超损失函数下降最快的方向迭代。在损失函数中，梯度是参数对损失函数的偏导数，表示当前点在各参数方向上的变化率。如图 2.5 所示，通过沿着损失函数梯度的反方向（即减小损失的方向）调整模型参数，从而逐步逼近最优解。

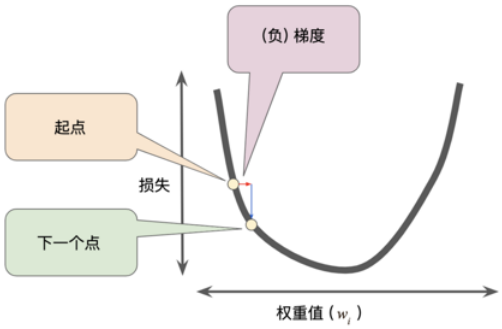



图 2.5 梯度下降法示例图

从直观上理解，我们可以将损失函数想象成一座山的地形，优化过程类似于从山顶沿着坡度最陡的方向下山，直至到达谷底（最小值点）。在二维和三维空间中，梯度下降过程可以直观地用等高线图或曲面图表示。

（补充：插入 **梯度下降的图示**，如损失函数曲线和等高线图。）

对于线性回归，梯度下降的迭代更新公式如下：

$$\mathbf{w}^{(t+1)}=\mathbf{w}^{(t)}-\eta\frac{\partial L}{\partial \boldsymbol{w}}, \qquad w_0^{(t+1)}=w_0^{(t)}-\eta\frac{\partial L}{\partial w_0}$$

其中，$\eta$ 是学习率，控制每次更新的步长大小。学习率的选择在学习优化中很

关键，学习率过小会导致优化收敛缓慢，而学习率过大会导致跳跃过大甚至发散。因此，学习率的选择需要结合实验经验和特定问题的需求。

综合以上分析，我们可以总结通过梯度下降法训练线性回归模型的完整算法。

**Input:** $D = \{(x^{(l)}, r^{(l)})\}$ (l =1:N)
**for** j = 0,…,d
    $\mathbf{w}_j \leftarrow$ rand (-0.01, 0.01)
**repeat**
    **for** j = 0,…,d
        $\Delta \mathbf{w}_j \leftarrow 0$
    **for** l = 1,…,N
        $y \leftarrow 0$
        **for** j = 0, …, d
            $y \leftarrow y + \mathbf{w}_j x_j^{(l)}$
        $\Delta \mathbf{w}_j \leftarrow \Delta \mathbf{w}_j + (r^{(l)} - y) x_j^{(l)}$
    $\Delta \mathbf{w}_j = \Delta \mathbf{w}_j / N$
    **for** j = 0,…,d
        $w_j \leftarrow w_j + \eta \Delta w_j$
**until** convergence

在实际应用中，单纯的梯度下降可能面临一些问题，比如：对大数据集计算量大。容易陷入局部最优解。为了解决这些问题，我们可以采用一些改进算法，更高效、更稳定地训练线性回归模型，例如：

- **随机梯度下降（SGD）**：每次迭代只使用一个样本点计算梯度，从而加快计算速度。
- **小批量梯度下降（Mini-batch GD）**：在每次迭代中使用一小部分样本点，兼顾了效率和稳定性。
- **动量法（Momentum）**：通过引入历史梯度加速收敛。
- **自适应学习率算法**：如 AdaGrad、RMSprop 和 Adam，根据梯度变化动态调整学习率。

**知识加油站**

推导线性回归模型的优化算法

$L(\boldsymbol{w}, w_0 \mid D) = -\frac{1}{2N} \sum_{l=1}^{N} (r^{(l)} - y^{(l)})^2$

For each $\boldsymbol{w}_j$ ($j$ =1,…,$d$ ):

$$\frac{\partial L}{\partial w_j} = -\frac{1}{N} \sum_l \left(r^{(l)} - y^{(l)}\right) \frac{\partial y^{(l)}}{\partial w_j} = -\frac{1}{N} \sum_l \left(r^{(l)} - y^{(l)}\right) x^{(l)}$$

$\boldsymbol{w}_{\text{new}} = \boldsymbol{w}_{\text{old}} + \frac{1}{N} \sum_{l=1}^{N} (r^{(l)} - y^{(l)})\ x^{(l)}$

## 2.3 线性回归模型的矩阵形式

在线性回归中，为了更高效地表示和计算模型，我们通常采用矩阵形式进行描述。这种方式不仅简洁明了，而且在实现高维数据计算时具有更高的可扩展性。

假设我们有一个包含 N 个样本的数据集，其中每个样本是一个 d 维向量 $\mathbf{x}^{(i)}$。将所有样本的输入变量合并成矩阵，得到

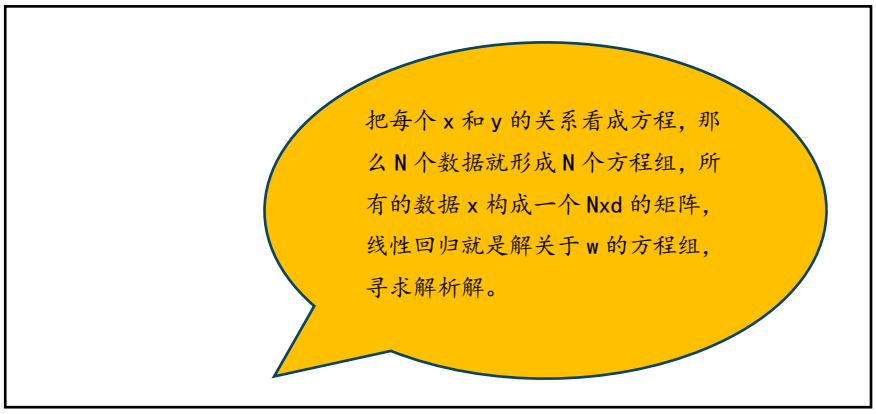
$$\mathbf{X}=\begin{bmatrix}\mathbf{x}(1)\\ \mathbf{x}(2)\\ \vdots\\ \mathbf{x}(\mathrm{N})\end{bmatrix}=\begin{bmatrix}x_0^{(1)} & x_1^{(1)} & x_2^{(1)} & \cdots & x_d^{(1)}\\ x_0^{(2)} & x_1^{(2)} & x_2^{(2)} & \cdots & x_d^{(2)}\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ x_0^{(N)} & x_1^{(N)} & x_2^{(N)} & \cdots & x_d^{(N)}\end{bmatrix},$$

同样，可以将每个样本的目标值合并，形成目标值向量 $\boldsymbol{r}=\begin{bmatrix}r^{(1)}\\ r^{(2)}\\ \vdots\\ r^{(N)}\end{bmatrix}$。

这样，线性回归模型表示成以下的矩阵形式：

$$\mathbf{y}=\mathbf{X}\mathbf{w}=\begin{bmatrix}\mathbf{x}^{(1)}\mathbf{w}\\ \mathbf{x}^{(2)}\mathbf{w}\\ \vdots\\ \mathbf{x}^{(\mathrm{N})}\mathbf{w}\end{bmatrix}$$

其中 $\mathbf{w}=\begin{bmatrix}\mathrm{W}1\\ \mathrm{W}2\\ \vdots\\ \mathrm{W}d\end{bmatrix}$ 是模型的参数矩阵（回归系数），$\mathrm{y}\in\mathrm{R}^{\mathrm{N}}$ 是模型的预测值向量。




采用矩阵形式的 MSE 损失函数:

$$\mathbf{L}(\mathbf{w})=\frac{1}{2}(\mathbf{r}\text{-}\mathbf{y})^{\mathrm{T}}(\mathbf{r}\text{-}\mathbf{y})=\frac{1}{2}(\mathbf{r}\text{-}\mathbf{X}\mathbf{w})^{\mathrm{T}}(\mathbf{r}\text{-}\mathbf{X}\mathbf{w})$$

损失函数的梯度为：

$$\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}}=\text{-}\boldsymbol{X}^{T}(\boldsymbol{r}\text{-}\boldsymbol{X}\boldsymbol{w})$$

令梯度等于 0，可以得到 $\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}}=0\ \Rightarrow \boldsymbol{X}^{T}(\boldsymbol{r}-\boldsymbol{X}\boldsymbol{w})=0$

整理后，得到 $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{r}=\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{w}$

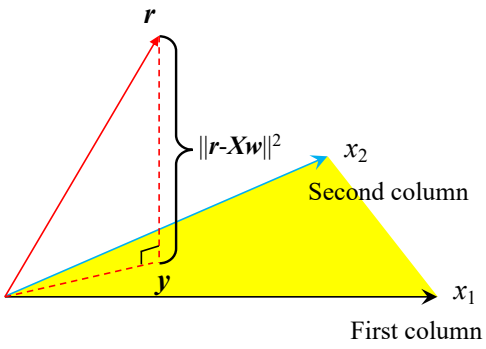
最终，得到解析解为 $\boldsymbol{w}^{*}=(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{r}$

将解析解代入原始回归表达式，得到相应的预测值为 $\boldsymbol{y}=\boldsymbol{X}(\boldsymbol{X}^{T}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{r}$

**几何解释**

在上述矩阵形式的预测表达式中，我们令 $H=\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T$，得到 $\boldsymbol{y}=\boldsymbol{H}\boldsymbol{r}$，

可以看到预测值 $\boldsymbol{y}$ 是一个关于 $\boldsymbol{r}$ 的线性函数，H 是 r 的一个线性变换。它将 r 映射到哪里呢？映射到一个预测的 y。根据线性回归定义，y 是关于 x 的线性变换，也就是由 $[\boldsymbol{x}_1,\boldsymbol{x}_2,\ldots,\boldsymbol{x}_d]$张成的空间上的一个向量，我们希望这个向量离 r 最近，换句话说，就是在$[\boldsymbol{x}_1,\ \boldsymbol{x}_2,\cdots,\ \boldsymbol{x}_d]$张成的空间上寻找一个点离 r 最近，这个点必然是 r 在该空间上的投影。简言之，H 对 r 进行线性变换，得到其在 X 空间上的投影 y。




## 2.4 过拟合与正则化修正

观察到线性回归解析解的表达式中有一项$(\boldsymbol{X}^T\boldsymbol{X})^{-1}$，这要求该矩阵可逆。然而，当数据特征之间存在高度相关性（例如 $x_2=3x_1$），矩阵 $\boldsymbol{X}^T\boldsymbol{X}$ 可能变得奇异（不可逆），从而导致解析解 $w^\star=(X^TX)^{-1}X^Tr$ 无法直接计算。为了解决这一问题，我们可以引入正则化（Regularization），通过增加对模型参数的约束来避免过拟合并提高模型的泛化能力。

正则化的核心思想是在损失函数中加入对模型参数的惩罚项，从而限制模型的复杂度。常用的正则化方法包括 L2 正则化，其损失函数形式如下：

$$L(w)=\frac{1}{2}(r-y)^T(r-y)=\frac{1}{2}(r-Xw)^T(r-Xw)+\frac{\lambda}{2}\|w\|_2^2$$

其中：

- $\frac{1}{2}(r-Xw)^T(r-Xw)$ 是原始的均方误差损失。
- $\lambda/2\|w\|_2^2$ 是 L2 正则化项，$\lambda$ 是正则化系数，控制正则化的强度。
- $\|w\|_2^2=w^Tw$ 是权重向量 **w** 的二范数平方。

加入 L2 正则化后，损失函数的梯度变为：

$$\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}}=-\boldsymbol{X}^T(\boldsymbol{r}-\boldsymbol{X}\boldsymbol{w})+\lambda\boldsymbol{w}$$

令梯度为零，求解最优权重：$\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}}=0\ \Rightarrow -\boldsymbol{X}^T(\boldsymbol{r}-\boldsymbol{X}\boldsymbol{w})+\lambda\boldsymbol{w}=0$

整理后得到：$\boldsymbol{X}^T\boldsymbol{r}=(\boldsymbol{X}^T\boldsymbol{X}+\lambda\mathbf{I})\boldsymbol{w}$

最终，正则化后的解析解为：$\boldsymbol{w}^\star=(\boldsymbol{X}^T\boldsymbol{X}+\lambda\mathbf{I})^{-1}\boldsymbol{X}^T\boldsymbol{r}$

其中：I 是单位矩阵。$\lambda$I 的加入确保了 $X^TX+\lambda I$ 始终可逆，即使 $X^TX$ 是奇异的。

除了解决矩阵奇异的问题外，我们还可以从以下几点来分析正则化的作用：从模型复杂度来看，正则化通过最小化权重向量的二范数$\|w\|_2^2$，限制权重的幅度，从而降低模型的复杂度。这类似于将高次曲线函数简化为线性函数，避免模

型过度拟合训练数据。从贝叶斯视角来看，最小化二范数相当于假设权重 w 的先验分布服从高斯分布（零均值，方差由 $\lambda$ 控制），从而引入对模型参数的先验约束。最后，也是最精妙的一点，引入正则化等价于对数据作了增强。正则化项 $\lambda\mathbf{I}$ 的加入可以看作是在原始数据 $\mathbf{X}^\top\mathbf{X}$ 的基础上增加了随机数据（方差为 $\lambda$）。这种“虚拟数据”的引入在数据不充分时提高了模型的鲁棒性，限制了模型的复杂度，从而提升了泛化能力。换句话说，正则化通过对权重的惩罚，间接地对数据分布进行了约束，实现了数据增强的效果。在机器学习中，数据规模和模型复杂度常常是天平的两端，当数据较多时，模型复杂度可以相应增加以捕捉数据中的复杂模式；而过高的模型复杂度又会导致过拟合，要求提供更多数据。正则化通过限制模型复杂度，在数据不足时提供了一种平衡机制。降低模型复杂度相当于对数据进行了增强，使得模型在有限数据下仍能保持合理的性能。

在实际应用中，正则化系数 $\lambda$ 的选择至关重要：当 $\lambda$ 过大时，模型会过于简单，可能导致欠拟合。当 $\lambda$ 过小时，正则化的效果不明显，模型可能仍然过拟合。

**正则化的思想在现代机器学习中得到了广泛应用，是防止过拟合的重要工具之一。**通过线性回归的例子，我们窥见了正则化的精妙之处。在机器学习的其他模型中，如逻辑回归、支持向量机、神经网络等，正则化的思想同样被广泛应用。希望读者能够举一反三，深入理解正则化在不同场景中的作用，并将其灵活应用于实际问题中。


## 2.4 敲一敲代码

以下是一段完整的线性回归代码示例，展示如何使用矩阵形式实现线性回归。代码包括数据生成、解析解的求解以及梯度下降的实现。

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

# 1. 数据生成
np.random.seed(42)
N = 100  # 样本数
d = 1    # 特征维度
X = 2 * np.random.rand(N, d)  # 生成随机输入特征
true_w = np.array([3])          # 真正的权重
true_b = 2                      # 真正的偏置
noise = np.random.randn(N) * 0.5  # 噪声
y = X @ true_w + true_b + noise   # 生成目标值
```

```
# 添加一列 1 到 X 中，形成扩展矩阵
X_extended = np.hstack([X, np.ones((N, 1))])

# 2. 解析解
# w = (X^T X)^-1 X^T y
w_analytic = np.linalg.inv(X_extended.T @ X_extended) @ X_extended.T @ y
print(f"解析解的权重和偏置: {w_analytic}")

# 3. 梯度下降法实现
def gradient_descent(X, y, lr=0.01, iterations=100):
    N, d = X.shape
    w = np.zeros(d)  # 初始化参数
    loss_history = []
    for i in range(iterations):
        y_pred = X @ w
        gradient = -2 / N * X.T @ (y - y_pred)  # 计算梯度
        w -= lr * gradient                      # 更新参数
        loss = np.mean((y - y_pred) ** 2)       # 计算损失
        loss_history.append(loss)
    return w, loss_history

# 使用梯度下降法
w_gd, loss_history = gradient_descent(X_extended, y, lr=0.1, iterations=500)
print(f"梯度下降的权重和偏置: {w_gd}")

# 4. 可视化
plt.figure(figsize=(12, 6))

# 数据与拟合线
plt.subplot(1, 2, 1)
plt.scatter(X, y, label="Data", color="blue")
plt.plot(X, X @ w_analytic[:-1] + w_analytic[-1], label="Analytic Solution", color="green
")
plt.plot(X, X @ w_gd[:-1] + w_gd[-1], label="Gradient Descent", color="red", linestyle="
dashed")
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.title("线性回归拟合")

# 损失随迭代变化
plt.subplot(1, 2, 2)
plt.plot(range(len(loss_history)), loss_history, label="Loss")
plt.xlabel("Iterations")
plt.ylabel("Loss")
plt.title("梯度下降损失曲线")
plt.grid()

plt.tight_layout()
plt.show()
```

**输出说明**

1. **解析解：**

    - 利用矩阵操作直接求得最优权重和偏置。
2. **梯度下降：**
    - 使用循环迭代更新权重，逐步逼近最优解，同时记录损失随迭代的变化。
3. **可视化：**
    - 图 1 展示了数据点、解析解拟合的线和梯度下降拟合的线。
    - 图 2 展示了梯度下降中损失函数的收敛趋势。

**示例运行结果**

- **解析解的权重和偏置：**接近真实值 $w=3, b=2$。
- **梯度下降结果：**与解析解结果一致。
- **可视化结果：**绿色线（解析解）和红色虚线（梯度下降）重合，表明梯度下降成功收敛。

这个代码清晰地展示了如何通过矩阵形式实现线性回归，同时对比了解析解与梯度下降法的性能。