第 3 章

CHAPTER 3

数学基础知识

MATLAB 是一种广泛使用的数学软件，因其优秀的数学能力赢得了广泛的认可和使用。本章将针对 MATLAB 中的数学知识点进行整理，以帮助读者更好地掌握和运用 MATLAB 进行数学计算。

3.1 矩阵的微分

在应用中，矩阵函数与函数矩阵的微分、积分常常是同时出现的，因此在学习了矩阵函数的计算后，还需学习函数矩阵的微分、积分。

以变量 t 的函数为元素的矩阵 $\boldsymbol{A}(t)=(a_{ij}(t))_{m\times n}$ 称为函数矩阵。如果 $t\in[a,b]$，则称 $\boldsymbol{A}(t)$ 是定义在 $[a,b]$ 上的；如果每个 $a_{ij}(t)$ 在 $[a,b]$ 上连续（可微、可积），则称 $\boldsymbol{A}(t)$ 在 $[a,b]$ 上连续（可微、可积）。当 $\boldsymbol{A}(t)$ 在 $[a,b]$ 可微时，规定其导数为

$$\boldsymbol{A}'(t)=(a_{ij}'(t))_{m\times n} \text{ 或 } \frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{A}(t)=\left(\frac{\mathrm{d}}{\mathrm{d}t}a_{ij}(t)\right)$$

当 $\boldsymbol{A}(t)$ 在 $[a,b]$ 上可积时，规定 $\boldsymbol{A}(t)$ 在 $[a,b]$ 上的积分为

$$\int_a^b \boldsymbol{A}(t)\mathrm{d}t=\left(\int_a^b a_{ij}(t)\mathrm{d}t\right)_{m\times n}$$

3.1.1 标量与矩阵求导通用的法则

向量可以看成行或列为 1 的矩阵，下面对标量与标量、向量与标量、标量与矩阵的结合方式进行介绍。

1. 标量与标量

标量与标量就是正常的对函数求导。

2. 标量与向量

标量与向量分两种情况，分别为向量对标量求导、标量对向量求导。

1）向量对标量求导

向量对标量求导即向量的每个分量对标量求导为

$$\boldsymbol{y}=[y_1 \quad y_2 \quad \cdots \quad y_m]^{\mathrm{T}}$$

$$\frac{\partial \boldsymbol{y}}{\partial x}=\begin{bmatrix}\dfrac{\partial y_1}{\partial x}\\ \dfrac{\partial y_2}{\partial x}\\ \vdots\\ \dfrac{\partial y_m}{\partial x}\end{bmatrix}$$

2）标量对向量求导

标量对向量求导结果为一个与向量同阶的向量，每个元素为标量对应位置向量元素的倒数：

$$\boldsymbol{x}=[x_1 \quad x_2 \quad \cdots \quad x_n]^{\mathrm{T}}$$

因为是对向量求导，此处采用分子布局（即分母不变，分子转置）。用分子和分母布局求出来的结果互为转置：

$$\frac{\partial y}{\partial \boldsymbol{x}}=\begin{bmatrix}\dfrac{\partial y}{\partial x_1} & \dfrac{\partial y}{\partial x_2} & \cdots & \dfrac{\partial y}{\partial x_n}\end{bmatrix}$$

3. 标量与矩阵

标量的结合比较特殊，很简单。

1）矩阵对标量求导

对矩阵中的每个元素分别对标量求导即可。

$$\frac{\partial \boldsymbol{Y}}{\partial x}=\begin{bmatrix}\dfrac{\partial y_{11}}{\partial x} & \dfrac{\partial y_{12}}{\partial x} & \cdots & \dfrac{\partial y_{1n}}{\partial x}\\ \dfrac{\partial y_{21}}{\partial x} & \dfrac{\partial y_{22}}{\partial x} & \cdots & \dfrac{\partial y_{2n}}{\partial x}\\ \vdots & \vdots & \ddots & \vdots\\ \dfrac{\partial y_{m1}}{\partial x} & \dfrac{\partial y_{m2}}{\partial x} & \cdots & \dfrac{\partial y_{mn}}{\partial x}\end{bmatrix}$$

2）标量对矩阵求导

标量对矩阵的求导，即求导结果为一个与矩阵同阶的矩阵，其中元素为标量对应位置元素的倒数，如：

$$\frac{\partial y}{\partial \boldsymbol{X}}=\begin{bmatrix}\dfrac{\partial y}{\partial x_{11}} & \dfrac{\partial y}{\partial x_{12}} & \cdots & \dfrac{\partial y}{\partial x_{1q}}\\ \dfrac{\partial y}{\partial x_{21}} & \dfrac{\partial y}{\partial x_{22}} & \cdots & \dfrac{\partial y}{\partial x_{2q}}\\ \vdots & \vdots & \ddots & \vdots\\ \dfrac{\partial y}{\partial x_{p1}} & \dfrac{\partial y}{\partial x_{p2}} & \cdots & \dfrac{\partial y}{\partial x_{pq}}\end{bmatrix}$$

3.1.2 矩阵和向量求导的通用法则

向量可以看成行或列为 1 的特殊矩阵，矩阵可以分解成行或列向量组成的列矩阵或行矩阵。

对于 $m\times n$ 阶矩阵 $\boldsymbol{Y}$：

$$\boldsymbol{Y}=\begin{bmatrix} y_{11} & \cdots & y_{1n} \\ \vdots & \ddots & \vdots \\ y_{m1} & \cdots & y_{mn} \end{bmatrix}$$

对于 $p\times q$ 阶矩阵 $\boldsymbol{X}$：

$$\boldsymbol{X}=\begin{bmatrix} x_{11} & \cdots & x_{1q} \\ \vdots & \ddots & \vdots \\ x_{p1} & \cdots & x_{pq} \end{bmatrix}$$

将 $\boldsymbol{Y}$ 分解为 m 个 $1\times n$ 的行向量组成的列向量：

$$\boldsymbol{Y}=\begin{bmatrix} y_{11} & \cdots & y_{1n} \\ \vdots & \ddots & \vdots \\ y_{m1} & \cdots & y_{mn} \end{bmatrix}=\begin{bmatrix} \boldsymbol{y}_1^{\mathrm{T}} \\ \boldsymbol{y}_2^{\mathrm{T}} \\ \vdots \\ \boldsymbol{y}_m^{\mathrm{T}} \end{bmatrix}$$

将 $\boldsymbol{X}$ 分解为 q 个 $p\times 1$ 的行向量组成的行向量：

$$\boldsymbol{X}=\begin{bmatrix} x_{11} & \cdots & x_{1q} \\ \vdots & \ddots & \vdots \\ x_{p1} & \cdots & x_{pq} \end{bmatrix}=[\boldsymbol{x}_1 \quad \boldsymbol{x}_2 \quad \cdots \quad \boldsymbol{x}_q]$$

（1）以上问题可以先变为列向量对行向量求导。根据求导法则，结果为

$$\frac{\partial \boldsymbol{Y}}{\partial \boldsymbol{X}}=\begin{bmatrix} \dfrac{\partial y_1^{\mathrm{T}}}{\partial x_1} & \cdots & \dfrac{\partial y_1^{\mathrm{T}}}{\partial x_q} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial y_m^{\mathrm{T}}}{\partial x_1} & \cdots & \dfrac{\partial y_m^{\mathrm{T}}}{\partial x_q} \end{bmatrix}$$

此处是一个 $m\times q$ 的大矩阵，每一个元素是 $1\times n$ 的行向量对 $p\times 1$ 的列向量求导的结果，为 $p\times n$ 阶矩阵：

$$\frac{\partial \boldsymbol{y}_i^{\mathrm{T}}}{\partial \boldsymbol{x}_j}$$

（2）接下来问题就变为行向量对列向量求导。对于每一个小块矩阵，根据上述分解结果有

$$\boldsymbol{y}^{\mathrm{T}}=[y_1 \quad y_2 \quad \cdots \quad y_n]$$

$$\boldsymbol{x}=\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix}$$

根据求导法则：

$$\frac{\partial \boldsymbol{y}^{\mathrm{T}}}{\partial \boldsymbol{x}}=\begin{bmatrix}\dfrac{\partial y_1}{\partial x_1} & \cdots & \dfrac{\partial y_n}{\partial x_1}\\ \vdots & \ddots & \vdots\\ \dfrac{\partial y_1}{\partial x_p} & \cdots & \dfrac{\partial y_n}{\partial x_n}\end{bmatrix}$$

（3）最后就是标量对标量的求导。

结果为 $m\times p$ 行、$n\times q$ 列的矩阵，至此结束。

3.1.3　MATLAB 的实现

在 MATLAB 中，提供了 diff 函数用于实现求导数，jacobian 函数用于求雅可比矩阵。

（1）diff 函数。

diff 函数用于求矩阵的差分和近似导数。函数的语法格式为：

Y = diff(X)：计算沿大小不等于 1 的第一个数组维度的 X 相邻元素之间的差分。

① 如果 X 是长度为 m 的向量，则 Y = diff(X) 返回长度为 m−1 的向量。Y 的元素是 X 相邻元素之间的差分。

Y = [X(2)−X(1),X(3)−X(2),...,X(m)−X(m−1)]

② 如果 X 是不为空的非向量 p × m 矩阵，则 Y = diff(X) 返回大小为 (p−1) × m 的矩阵，其元素是 X 的行之间的差分。

Y = [X(2,:)−X(1,:); X(3,:)−X(2,:); ... X(p,:)−X(p−1,:)]

③ 如果 X 是 0 × 0 的空矩阵，则 Y = diff(X) 返回 0 × 0 的空矩阵。

④ 如果 X 是一个 p × m 表或时间表，则 Y = diff(X) 返回一个大小为 (p−1) × m 的表或时间表，其元素是 X 的行之间的差分。如果 X 是一个 1 × m 表或时间表，则 Y 的大小是 0 × m。

Y = diff(X,n)：通过递归应用 diff(X) 运算符 n 次来计算第 n 个差分。在实际操作中，这表示 diff(X,2) 与 diff(diff(X)) 相同。

Y = diff(X,n,dim)：沿 dim 指定的维计算的第 n 个差分。dim 输入是一个正整数标量。

（2）jacobian 函数。

jacobian 函数用于计算雅可比矩阵，函数的语法格式为：

jacobian(f,v)：计算 f 关于 v 的雅可比矩阵，其第 (i,j) 个元素为

$$\frac{\partial \mathrm{f(i)}}{\partial \mathrm{v(j)}}$$

其中，f 为标量或者向量函数，可为符号表达式、符号函数、符号向量等。如果 f 是一个标量，f 的雅可比矩阵是 f 的梯度的转置。

v 为要计算雅可比的变量向量、符号变量、符号向量。

① 如果 v 是一个标量，则结果等价于 diff(f,v) 的转置。

② 如果 v 是空符号对象，如 sym([ ])，则结果返回空符号对象。

【例 3-1】演示标量、向量、矩阵的相互求导。

```
>> clear all;
syms x y z v;
f1 = [x*y, x+y; y*z, y+z; x*y*z, x+y+z];            % 矩阵
```

```
f2 = [x*y;y*z;x*y*z];                                    % 列向量
f3 = x*y*z;                                              % 标量

v1 = x;                                                  % 标量
v2 = [x;y];                                              % 列向量
v3 = [x*y, x+y; y*z, y+z;x*y*z, x+y+z];                  % 矩阵

% 对标量求导用 diff
f3v1=diff(f3,v1)                                         % 标量对标量求导为标量
f2v1=diff(f2,v1)                                         % 列向量对标量求导为列向量
f1v1=diff(f1,v1)                                         % 矩阵对标量求导为矩阵

% 对向量求导用 jacobian
% 标量对向量求导为向量，以下两种情况结果相同
f3v2=jacobian(f3,v2)
f3v2=jacobian(f3,v2.')
% 数学上定义 1×n 行向量对 m×1 列向量求导后构成 m×n 矩阵
%jacobian 函数通过向量对向量求导构成矩阵，以下 4 种情况结果相同
% 注：syms 型求转置需用 .'
f2v2=jacobian(f2.',v2)
f2v2=jacobian(f2,v2.')
f2v2=jacobian(f2,v2)
f2v2=jacobian(f2.',v2.')
```

运行程序，输出如下：

```
f3v1 =
y*z
f2v1 =
  y
  0
y*z
f1v1 =
[  y, 1]
[  0, 0]
[y*z, 1]
f3v2 =
[y*z, x*z]
f3v2 =
[y*z, x*z]
f2v2 =
[  y,   x]
[  0,   z]
[y*z, x*z]
f2v2 =
[  y,   x]
[  0,   z]
[y*z, x*z]
f2v2 =
[  y,   x]
[  0,   z]
```

```
[y*z, x*z]
f2v2 =
[  y,   x]
[  0,   z]
[y*z, x*z]
```

3.2　向量和矩阵积分

在机器学习中，经常遇到一系列变量分析问题，向量和矩阵微积分是单个变量微积分的延伸。

3.2.1　向量梯度

令 $g(\boldsymbol{w})$ 为一个包含 m 个变量的可微数值函数，其中

$$\boldsymbol{w}=[w_1,w_2,\cdots,w_m]^{\mathrm{T}}$$

由此可得到 $g(\boldsymbol{w})$ 的梯度，采用 g 的偏微分形式：

$$\Delta g=\frac{\partial g}{\partial \boldsymbol{w}}=\begin{pmatrix}\dfrac{\partial g}{\partial w_1}\\ \vdots\\ \dfrac{\partial g}{\partial w_m}\end{pmatrix}$$

相似地，可定义二阶梯度矩阵或 Hessian 矩阵：

$$\frac{\partial^2 g}{\partial \boldsymbol{w}^2}=\begin{pmatrix}\dfrac{\partial^2 g}{\partial w_1^2} & \cdots & \dfrac{\partial^2 g}{\partial w_1 w_m}\\ \vdots & \ddots & \vdots\\ \dfrac{\partial^2 g}{\partial w_m w_1} & \cdots & \dfrac{\partial^2 g}{\partial w_m^2}\end{pmatrix}$$

将向量值函数进行推广，得到

$$g(\boldsymbol{w})=[g_1(\boldsymbol{w}),g_2(\boldsymbol{w}),\cdots,g_n(\boldsymbol{w})]^{\mathrm{T}}$$

从而得到 Jacobian 矩阵的定义：

$$\frac{\partial \mathrm{g}}{\partial \boldsymbol{w}}=\begin{pmatrix}\dfrac{\partial g}{\partial w_1} & \cdots & \dfrac{\partial g}{\partial w_1}\\ \vdots & \ddots & \vdots\\ \dfrac{\partial g_1}{\partial w_m} & \cdots & \dfrac{\partial g_n}{\partial w_m}\end{pmatrix}$$

在向量转化中，Jacobian 矩阵的列向量是对应的分量函数 $g_i(\boldsymbol{w})$ 的梯度。

3.2.2　微分公式

常用的微分公式为

$$\frac{\partial f(\boldsymbol{w})g(\boldsymbol{w})}{\partial \boldsymbol{w}}=\frac{\partial f(\boldsymbol{w})}{\partial \boldsymbol{w}}g(\boldsymbol{w})+f(\boldsymbol{w})\frac{\partial g(\boldsymbol{w})}{\partial \boldsymbol{w}}$$

$$\frac{\partial f(\boldsymbol{w})/g(\boldsymbol{w})}{\partial \boldsymbol{w}}=\frac{\dfrac{\partial f(\boldsymbol{w})}{\partial \boldsymbol{w}}g(\boldsymbol{w})-f(\boldsymbol{w})\dfrac{\partial g(\boldsymbol{w})}{\partial \boldsymbol{w}}}{g^2(\boldsymbol{w})}$$

$$\frac{\partial a^{\mathrm{T}}\boldsymbol{w}}{\partial \boldsymbol{w}}=a$$

3.2.3 优化方法

梯度下降是对给定代价函数 $J(\boldsymbol{w})$ 进行最小化的方法：

$$\nabla J(\boldsymbol{w})=-\alpha(t)\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}}$$

优化步骤为：

（1）初始值是 $\boldsymbol{w}(0)$；

（2）计算 $\boldsymbol{w}(0)$ 处的梯度 $\nabla J\boldsymbol{w}$；

（3）向负梯度或最陡下降方向移动一段距离；

（4）重复上述步骤，直到连续点足够接近。

3.2.4 拉格朗日乘子法

通常，约束优化问题可表述为

$$\min J(\boldsymbol{w})$$
$$\text{s.t.}\quad H_i(w)=0, i=1,2,\cdots,k$$

其中，J 为代价函数，H_i 为限制条件。

拉格朗日乘子法广泛应用于求解带约束的优化问题，使用时需要构建拉格朗日方程：

$$L(\boldsymbol{w},\lambda_1,\lambda_2,\cdots,\lambda_k)=\lambda(\boldsymbol{w})+\sum_{i=1}^{k}\lambda_i H_i(\boldsymbol{w})$$

其中，λ_i 是拉格朗日乘子。

令 L 的梯度为 0，可以求解最值问题，即有

$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}}+\sum_{i=1}^{k}\frac{\partial H_i(\boldsymbol{w})}{\partial \boldsymbol{w}}=0$$

3.2.5 向量矩阵积分实现

在 MATLAB 中，提供了 trapz 函数为梯形函数积分，可用于实现向量矩阵积分。函数的语法格式如下。

Q = trapz(Y)：通过梯形法计算 Y 的近似积分（采用单位间距）。根据 Y 的大小确定求积分所沿用的维度。

① 如果 Y 为向量，则 trapz(Y) 是 Y 的近似积分。

② 如果 Y 为矩阵，则 trapz(Y) 对每列求积分并返回积分值的行向量。

③ 如果 Y 是多维数组，则 trapz(Y) 对大小不等于 1 的第一个维度求积分。该维度的大小变为 1，而其他维度的大小保持不变。

Q = trapz(X,Y)：根据 X 指定的坐标或标量间距对 Y 进行积分。

① 如果 X 是坐标向量，则 length(X) 必须等于 Y 的大小。

② 如果 X 是标量间距，则 trapz(X,Y) 等于 X*trapz(Y)。

Q = trapz(___,dim)：使用上述任意语法沿维度 dim 求积分。必须指定 Y，也可以指定 X。如果指定 X，则它可以是长度等于 size(Y,dim) 的标量或向量。例如，如果 Y 为矩阵，则 trapz(X,Y,2) 对 Y 的每行求积分。

【例 3-2】利用 trapz 函数对向量及矩阵求积分。

```
>>% 计算数据点之间的间距为 1 的向量的积分
 Y = [1 4 9 16 25];
>> % 使用 trapz 按单位间距对数据求积分
>> Q = trapz(Y)
Q =
    42
```

近似积分生成值为 42。在这种情况下，确切答案比近似积分稍小，为 $41\frac{1}{3}$。trapz 函数高估积分值，因为 $f(x)$ 是向上凸的。

```
% 对具有非均匀数据间距的矩阵的行求积分
>> X = [1 2.5 7 10];
Y = [5.2   7.7   9.6   13.2;
     4.8   7.0  10.5   14.5;
     4.9   6.5  10.2   13.8];
% 使用 trapz 分别对每一行进行积分，然后求出每次试验中经过的总距离。由于数据不是按固定间隔计
% 算的，因此指定 X 来表示数据点之间的间距。由于数据位于 Y 的行中，因此指定 dim = 2
>> Q1 = trapz(X,Y,2)
Q1 =
   82.8000
   85.7250
   82.1250
```

结果为积分值的列向量，Y 中的每行对应一个列向量。

3.3　特征值分解和奇异值分解

特征值分解只适用于方阵，然而在实际应用中，大部分矩阵不是方阵。而奇异值分解适用于任意矩阵，本节将对特征值分解与奇异值分解进行介绍。

3.3.1　特征值分解

特征值分解是将一个方阵 $\boldsymbol{A}$ 分解为如下形式：

$$\boldsymbol{A}=\boldsymbol{Q}\boldsymbol{\Sigma}\boldsymbol{Q}^{-1}$$

其中，$\boldsymbol{Q}$ 是方阵 $\boldsymbol{A}$ 的特征向量组成的矩阵，Σ 是一个对角矩阵，对角线元素是特征值。

通过特征值分解得到的前 N 个特征向量，表示矩阵 $\boldsymbol{A}$ 最主要的 N 个变化方向。利用这前 N 个变化方向，即可以近似这个矩阵（变换）。

【例 3-3】矩阵的特征值分解演示。

```
>> A =[ 0    -6    -1; 6     2   -16;  -5    20   -10];
```

此方程的解用矩阵指数 $x(t)=e^{tA}x(0)$ 表示：

```
>> lambda = eig(A)
lambda =
  -3.0710 + 0.0000i
  -2.4645 +17.6008i
  -2.4645 -17.6008i
```

每个特征值的实部都为负数，因此随着 t 的增加，$e^{\lambda t}$ 将会接近零。两个特征值（$+w$ 与 $-w$）的非零虚部为微分方程的解提供了振动分量 $\sin(wt)$。使用这两个输出参数，eig 可以计算特征向量并将特征值存储在对角矩阵中。

```
>> [V,D] = eig(A)
V =
  -0.8326 + 0.0000i   0.2003 - 0.1394i   0.2003 + 0.1394i
  -0.3553 + 0.0000i  -0.2110 - 0.6447i  -0.2110 + 0.6447i
  -0.4248 + 0.0000i  -0.6930 + 0.0000i  -0.6930 + 0.0000i
D =
  -3.0710 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  -2.4645 +17.6008i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i  -2.4645 -17.6008i
```

第一个特征向量为实数，另外两个向量互为共轭复数，三个向量都归一化为等于 1 的欧几里得长度 norm(v,2)。

矩阵 V × D × inv(V)（可更简洁地写为 V × D/V）位于 A 的舍入误差界限内，inv(V) × A × V 或 V\A × V 都在 D 的舍入误差界限内。

（1）多重特征值。

某些矩阵没有特征向量分解，这些矩阵是不可对角化的，例如：

```
>> [V,D] = eig(A)
V =
    1.0000    1.0000   -0.5571
         0    0.0000    0.7428
         0         0    0.3714
D =
     1     0     0
     0     1     0
     0     0     3
```

λ=1 时有一个双精度特征值，V 的第一列和第二列相同，对于此矩阵，并不存在一组完整的线性无关特征向量。

（2）Schur 分解。

许多高级矩阵计算不需要进行特征分解，而是使用 Schur 分解，公式为

$$A=USU'$$

其中，U 是正交矩阵，S 是对角线上 1×1 和 2×2 块的块上三角矩阵。特征值是通过 S 的对角元素和块显示的，而 U 的列提供了正交基，它的数值属性要远远优于一组特征向量。

【例 3-4】比较下面的亏损矩阵的特征值和 Schur 分解。

```
>> A = [ 6    12    19;-9   -20   -33; 4     9    15 ];
[V,D] = eig(A)
V =
  -0.4741 + 0.0000i  -0.4082 - 0.0000i  -0.4082 + 0.0000i
   0.8127 + 0.0000i   0.8165 + 0.0000i   0.8165 + 0.0000i
  -0.3386 + 0.0000i  -0.4082 + 0.0000i  -0.4082 - 0.0000i
D =
  -1.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   1.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   1.0000 - 0.0000i
>> [U,S] = schur(A)                                    %Schur 分解
U =
   -0.4741    0.6648    0.5774
    0.8127    0.0782    0.5774
   -0.3386   -0.7430    0.5774
S =
   -1.0000   20.7846  -44.6948
         0    1.0000   -0.6096
         0    0.0000    1.0000
```

矩阵 A 为亏损矩阵，因为它不具备一组完整的线性无关特征向量（V 的第二列和第三列相同）。由于 V 的列并不全部是线性无关的，因此它有一个很大的条件数，约为 1×10^8。但 schur 可以计算 U 中的 3 个不同基向量。由于 U 是正交矩阵，因此 cond(U) = 1。

矩阵 S 的实数特征值作为对角线上的第一个条目，并通过右下方的 2×2 块表示重复的特征值，2×2 块的特征值也是 A 的特征值：

```
>> eig(S(2:3,2:3))
ans =
   1.0000 + 0.0000i
   1.0000 - 0.0000i
```

3.3.2　奇异值分解

奇异值分解（Singular Value Decomposition，SVD）是在机器学习领域广泛应用的算法，它不仅可以用于降维算法中的特征分解，还可以用于推荐系统及自然语言处理等领域。它能适用于任意的矩阵的分解。

SVD 是将 $m\times n$ 的矩阵 A 分解为如下形式：

$$A=USV^{\mathrm{T}}$$

其中，U 和 V 是正交矩阵，即 $U^{\mathrm{T}}U=I$，$V^{\mathrm{T}}V=I$，U 是左奇异矩阵，$U\in\mathbf{R}^{m\times m}$，$S$ 是 $m\times n$ 的对角阵（对角线上的元素是奇异值，非对角线元素都是 0），V^{T} 为右奇异向量，$V\in\mathbf{R}^{n\times n}$。

特征值用来描述方阵，可看作从一个空间到自身的映射。奇异值可以描述长方阵或奇异

矩阵，可看作从一个空间到另一个空间的映射。奇异值和特征值是有关系的，奇异值就是矩阵 $\boldsymbol{A}\times\boldsymbol{A}^{\mathrm{T}}$ 的特征值的平方根。

奇异值分解输入为样本数据，输出为左奇异矩阵，奇异值矩阵，右奇异矩阵，其步骤如下。

（1）计算特征值：特征值分解 $\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}$，其中 $\boldsymbol{A}\in\mathbf{R}^{m\times n}$ 为原始样本数据。

$$\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}=\boldsymbol{U}\sum\sum^{\mathrm{T}}\boldsymbol{U}^{\mathrm{T}}$$

得到左奇异矩阵 $\boldsymbol{U}\in\mathbf{R}^{m\times m}$ 和奇异值矩阵 $\sum'\in\mathbf{R}^{m\times n}$。

（2）间接求部分右奇异矩阵：求 $\boldsymbol{V}\in\mathbf{R}^{m\times n}$。

利用 $\boldsymbol{A}=\boldsymbol{U}\sum'\boldsymbol{V}'$ 可得

$$\boldsymbol{V}'=(\boldsymbol{U}\sum')^{-1}\boldsymbol{A}=(\sum)^{-1}\boldsymbol{U}^{\mathrm{T}}\boldsymbol{A}$$

（3）返回 $\boldsymbol{U}$、Σ'、$\boldsymbol{V}'$，分别为左奇异矩阵、奇异值矩阵、右奇异矩阵。

在 MATLAB 中，提供了 svd 函数实现奇异值分解。函数的语法格式如下。

S = svd(A)：降序返回矩阵 A 的奇异值。

[U,S,V] = svd(A)：执行矩阵 A 的奇异值分解，因此 A = U × S × V'。

[___] = svd(A,"econ")：使用上述任一输出参数组合生成 A 的精简分解。如果 A 是 m × n 矩阵，则：

① 如果 m > n，只计算 U 的前 n 列，S 是一个 n × n 矩阵；

② 如果 m = n，svd(A,"econ")，等效于 svd(A)；

③ 如果 m < n，只计算 V 的前 m 列，S 是一个 m × m 矩阵。

精简分解从奇异值的对角矩阵 S 中删除额外的零值行或列，以及 U 或 V 中与表达式 A= U × S × V' 中的那些零值相乘的列。删除这些零值和列可以缩短执行时间，并减少存储要求，而且不会影响分解的准确性。

[___] = svd(A,0) 为 m × n 矩阵 A 生成另一种精简分解：

① 如果 m > n，svd(A,0) 等效于 svd(A,"econ")；

② 如果 m <= n，svd(A,0) 等效于 svd(A)。

[___] = svd(___,outputForm)：还可以指定奇异值的输出格式，可以将此选项与上述任一输入或输出参数组合一起使用。指定 vector 以列向量形式返回奇异值，或指定 matrix 以对角矩阵形式返回奇异值。

【例 3-5】求矩形矩阵 $\boldsymbol{A}$ 的奇异值分解。

```
>> A = [1 2; 3 4; 5 6; 7 8];
>> [U,S,V] = svd(A)                                    % 奇异值分解
U =
   -0.1525   -0.8226   -0.3945   -0.3800
   -0.3499   -0.4214    0.2428    0.8007
   -0.5474   -0.0201    0.6979   -0.4614
   -0.7448    0.3812   -0.5462    0.0407
S =
   14.2691         0
         0    0.6268
         0         0
         0         0
V =
   -0.6414    0.7672
```

```
    -0.7672   -0.6414
>> % 在计算机精度范围内确认关系 A = U*S*V'
>> U*S*V'
ans =
    1.0000    2.0000
    3.0000    4.0000
    5.0000    6.0000
    7.0000    8.0000
```

3.4　最优化方法

在生活和工作中，人们对于同一个问题往往会提出多个解决方案，并通过各方面的论证从中提取最佳方案。最优化方法就是专门研究如何从多个方案中科学合理地提取出最佳方案的科学。由于优化问题无所不在，目前最优化方法的应用和研究已经深入生产和科研的各个领域。

3.4.1　无约束优化方法

本节介绍几种常用的无约束方法。

1. 梯度下降法

梯度下降法也叫作最速下降法，因为负梯度是函数局部下降最快的方向。

1）梯度下降

梯度下降法的迭代格式为

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k)$$

梯度下降法非常简单，只需要知道如何计算目标函数的梯度就可以写出迭代格式。因此，尽管在不少情况下，梯度下降法的收敛速度都很慢，但依然不影响它在工业界的广泛应用。

2）随机梯度下降

在机器学习中，经常有 $f(x)=\sum_{i=1}^{m}\ell_i(x)$，其中 $\ell_i(x)$ 是第 i 个训练样本的损失函数。这时可以使用随机梯度下降法，其迭代格式为

$$x_{k+1} = x_k - \alpha_k \nabla \ell_r(x_k)$$

其中，$r \in 1,2,\cdots,m$ 为随机数。这种做法可以理解为随机选择一个训练样本，进行一次梯度下降的训练。

2. 共轭梯度法

由于每次都是沿着当前的负梯度方向逼近极小值，梯度下降法往往不能很快地收敛到极小值点。改进的方法是，在上一次梯度方向的共轭方向上进行迭代。其迭代的公式为

$$x_{k+1} = x_k + \alpha_k d_k$$

其中，d_k 为迭代方向，它由如下公式确定。

$$d_k = -f(x_k) + \beta_k d_{k-1}$$

使用系数 β_k 借助上一次的迭代方向 d_{k-1}，对迭代方向 d_k 进行一个修正。β_k 的表达式不止一种，常用的表达式如下：

$$\beta_k = \frac{(\nabla f(x_k))^{\mathrm{T}}(\nabla f(x_k) - \nabla f(x_{k-1}))}{(\nabla f(x_{k-1})^{\mathrm{T}} d_{k-1})}$$

3. 牛顿法

牛顿法和梯度法最大的区别是前者考虑了 Hessian 矩阵（记 x_k 处的 Hessian 矩阵为 $\nabla^2 f(x_k)$）。牛顿法的迭代格式为

$$x_{k+1} = x_k - \alpha_k (\nabla^2 f(x_k))^{-1} \nabla f(x_k)$$

此处的步长 α_k 有多种取法，此处步长取 1。值得注意的是，虽然写作 $d_k = -(\nabla^2 f(x_k))^{-1}\nabla f(x_k)$，但在计算 d_k 时，并不真正求逆，而是去求解线性方程组 $(\nabla^2 f(x_k))d_k = -\nabla f(x_k)$。

假设 $f(x)$ 是一元函数，上式将变为

$$x_{k+1} = x_k - \alpha \frac{f(x_k)}{f'(x_k)}$$

4. 牛顿法的局限

牛顿法在计算上有一定局限性，主要表现在以下 3 方面。

（1）计算矩阵 $\nabla^2 f(x_k)$ 可能要花费很长时间。

（2）可能没有足够的内存去存储矩阵 $\nabla^2 f(x_k)$。

（3）$\nabla^2 f(x_k)$ 不一定可逆，也就是 $(\nabla^2 f(x_k))^{-1}$ 不一定存在。

因此一般只有当问题规模较小且 $f(x_k)$ 是严格凸函数时，才会考虑牛顿法。在其他情形下使用牛顿法，都需要设法进行一些修正。

5. 拟牛顿法

牛顿法的局限性基本源于 $\nabla^2 f(x_k)$。在拟牛顿法中，不直接使用 $\nabla^2 f(x_k)$，而是使用 $\boldsymbol{H}_k$ 近似代替。

在第一次迭代时，没有任何有效信息可以用于选取 $\boldsymbol{H}_0$，因此一般直接取 $\boldsymbol{H}_0 = \boldsymbol{I}$。对于 $\boldsymbol{H}_{k+1}$ 的确定方法，分别用 BFGS（Brotden-Fletcher Goldfard Shanno）和 DFP（Davidon Fletcher Powell）公式给出。

（1）BFGS 公式为

$$\boldsymbol{H}_{k+1} = \left(\boldsymbol{I} - \frac{\boldsymbol{s}_k \boldsymbol{y}_k^{\mathrm{T}}}{\boldsymbol{y}_k^{\mathrm{T}} \boldsymbol{s}_k}\right) \boldsymbol{H}_k \left(\boldsymbol{I} - \frac{\boldsymbol{y}_k \boldsymbol{s}_k^{\mathrm{T}}}{\boldsymbol{y}_k^{\mathrm{T}} \boldsymbol{s}_k}\right) + \frac{\boldsymbol{s}_k \boldsymbol{y}_k^{\mathrm{T}}}{\boldsymbol{y}_k^{\mathrm{T}} \boldsymbol{s}_k}$$

（2）DFP 公式为

$$\boldsymbol{H}_{k+1} = \boldsymbol{H}_k + \frac{\boldsymbol{s}_k \boldsymbol{s}_k^{\mathrm{T}}}{\boldsymbol{s}_k^{\mathrm{T}} \boldsymbol{y}_k} - \frac{\boldsymbol{H}_k \boldsymbol{y}_k \boldsymbol{y}_k^{\mathrm{T}} \boldsymbol{H}_k^{\mathrm{T}}}{\boldsymbol{y}_k^{\mathrm{T}} \boldsymbol{H}_k^{\mathrm{T}} \boldsymbol{y}_k}$$

于是拟牛顿法的迭代公式变为

$$x_{k+1} = x_k + \alpha_k \boldsymbol{H}_k \nabla f(x_k)$$

步长 α_k 可以使用某种线搜索方法进行确定。拟牛顿法很好地解决了 Hessian 矩阵的计算问题，但是仍然没有解决存储问题。一个很好的解决办法是有限内存 BFGS 方法。这里不做进一步的介绍。

6. 无约束优化实现

在 MATLAB 中，提供了相关函数用于实现无约束优化算法，下面对相关函数进行介绍。

（1）fminunc 函数。

在 MATLAB 中，提供了 fminunc 函数用于求无约束多变量函数的最小值，即求以下问题的最小值：

$$\min_{x\in R^n} f(\boldsymbol{x})$$

其中，$f(\boldsymbol{x})$ 为返回标量的函数，$\boldsymbol{x}$ 为向量或矩阵。函数的语法格式为：

x = fminunc(fun,x0)：在点 x0 处开始并尝试求 fun 中描述的函数的局部最小值 x。点 x0 可以是标量、向量或矩阵。

x = fminunc(fun,x0,options)：使用 options 中指定的优化选项最小化 fun。使用 optimoptions 可设置这些选项。

x = fminunc(problem)：求 problem 的最小值，它是 problem 中所述的一个结构体。

[x,fval] = fminunc(___)：对上述任何语法，返回目标函数 fun 在解 x 处的值。

[x,fval,exitflag,output] = fminunc(___)：返回描述 fminunc 的退出条件的值 exitflag，以及提供优化过程信息的结构体 output。

[x,fval,exitflag,output,grad,hessian] = fminunc(___)：返回 grad - fun 在解 x 处的梯度。

【例 3-6】求非线性函数 $f(x)=x_1\mathrm{e}^{-\|x\|_2^2}+\dfrac{\|x\|_2^2}{20}$ 的最小值，并检测求解过程。

实现的 MATLAB 代码为：

```
>> clear all;
% 设置选项以获取迭代输出并使用 'quasi-newton' 算法
options = optimoptions(@fminunc,'Display','iter','Algorithm',
'quasi-newton');
fun = @(x)x(1)*exp(-(x(1)^2 + x(2)^2)) + (x(1)^2 + x(2)^2)/20;
% 定义目标函数为
fun = @(x)x(1)*exp(-(x(1)^2 + x(2)^2)) + (x(1)^2 + x(2)^2)/20;
% 在 x0=[1,2] 处开始最小化，并获取检查求解质量和过程的输出
x0 = [1,2];
[x,fval,exitflag,output] = fminunc(fun,x0,options)
```

运行程序，输出如下：

```
                                                  First-order
 Iteration  Func-count       f(x)        Step-size      optimality
     0           3         0.256738                       0.173
     1           6         0.222149             1         0.131
     2           9          0.15717             1         0.158
     3          18        -0.227902        0.438133         0.386
     4          21        -0.299271             1          0.46
     5          30        -0.404028        0.102071        0.0458
     6          33        -0.404868             1        0.0296
     7          36        -0.405236             1       0.00119
     8          39        -0.405237             1      0.000252
     9          42        -0.405237             1      7.97e-07
找到局部最小值。
```

```
优化已完成，因为梯度大小小于
最优性容差的值。
< 停止条件详细信息 >
x =
   -0.6691    0.0000
fval =
   -0.4052
exitflag =
     1
output =
包含以下字段的 struct:
       iterations: 9
        funcCount: 42
         stepsize: 2.9343e-04
     lssteplength: 1
    firstorderopt: 7.9721e-07
        algorithm: 'quasi-newton'
          message: ' 找到局部最小值。
```

【例 3-7】（产销量的最佳安排）某工厂生产一种产品，有甲、乙两个牌号，讨论在产销平衡的情况下怎样确定各自的产量，使总利润最大。所谓产销平衡是指工厂的产量等于市场上的销量。其中，$f(x_1,x_2)$ 为总利润；p_1、q_1、x_1 分别表示甲的价格、成本、销量；p_2、q_2、x_2 分别表示乙的价格、成本、销量；a_{ij}、b_i、λ_i、$c_i(i,j=1,2)$ 为待定系数。根据大量的统计数据，求出系数：

$$b_1 = 100,\ a_{11} = 1,\ a_{12} = 0.1;\ b_2 = 280,\ a_{21} = 0.2,\ a_{22} = 2$$
$$r_1 = 30,\ \lambda_1 = 0.015,\ c_1 = 20;\ r_2 = 100,\ \lambda_2 = 0.02,\ c_2 = 30$$

将问题转换为无约束优化问题，求甲、乙两个牌号的产量 x_1、x_2，使总利润 f 最大。

解：将问题转换为求以下函数的极大值：

$$f_1 = (b_1 - a_{11}x_1)x_1 + (b_2 - a_{22}x_2)x_2$$

显然，其解为 $x_1 = \dfrac{b_1}{2a_{11}} = 50$，$x_2 = \dfrac{b_2}{2a_{22}} = 70$，把它作为原问题的初始值。

根据需要，建立描述目标函数的 M 文件，代码为：

```
function f=objfun(x)
f1=((110-x(1)-0.1*x(2))-(30*exp(-0.015*x(1))+20))*x(1);
f2=((280-0.2*x(1)-2*x(2))-(100*exp(-0.025*x(2))+30))*x(2);
f=-f1-f2;
```

调用 fminunc 求解该最优化问题，设定初始点为 [50,70]，代码为：

```
>> clear all;
x0=[50,70];
[x,fval,exitflag]=fminunc(@objfun,x0)
```

运行程序，输出如下：

```
x =
   30.3310   63.2125
```

```
fval =
  -7.1672e+03
exitflag =
     1
```

（2）fminsearch 函数。

MATLAB 中求解无约束优化问题还可以调用 fminsearch 函数，该函数与 fminunc 函数不同，因为 fminsearch 进行寻优的算法基于不使用梯度的单纯形法，其应用范围是无约束的多维非线性规划问题。函数 fminsearch 的调用格式如下。

x = fminsearch(fun,x0)：在点 x0 处开始并尝试求 fun 中描述的函数的局部最小值 x。

x = fminsearch(fun,x0,options)：使用结构体 options 中指定的优化选项求最小值，使用 optimset 可设置这些选项。

x = fminsearch(problem)：求 problem 的最小值，其中 problem 是一个结构体。

[x,fval] = fminsearch(___)：对任何上述输入语法，在 fval 中返回目标函数 fun 在解 x 处的值。

[x,fval,exitflag] = fminsearch(___)：返回描述退出条件的值 exitflag。

[x,fval,exitflag,output] = fminsearch(___)：返回结构体 output 以及有关优化过程的信息。

【例 3-8】假设在以下 Rosenbrock 类型函数中有一个参数 a：

$$f(x,a)=100(x_2-x_1^2)^2+(a-x_1)^2$$

此函数在 $x_1=a$、$x_2=a^2$ 处具有最小值 0。假设 $a=3$，可以通过创建匿名函数将该参数包含在目标函数中。

```
>> % 创建目标函数并将其额外形参作为额外实参
f = @(x,a)100*(x(2) - x(1)^2)^2 + (a-x(1))^2;
% 将参数放在 MATLAB 工作区中
a = 3;
% 单独创建包含参数的工作区值的 x 的匿名函数
fun = @(x)f(x,a);
% 在 x0 = [-1,1.9] 处开始解算该问题
x0 = [-1,1.9];
x = fminsearch(fun,x0)
```

运行程序，输出如下：

```
x =
    3.0000    9.0000
```

（3）fgoalattain 函数。

在运筹学中，多目标规划问题属于比较复杂的一类优化问题，通常没有固定的求解算法，而且对于求解结果，不容易评价其优化性能。

多目标优化问题的处理方法包括以下 4 种。

① 约束法：确定目标函数的取值范围后，将其转换成约束条件。

② 权重法：为每个目标函数分配一定的权重进行加和，将其转换为单目标优化问题。权重法包括固定权重法、适应性权重法与随机权重法。

③ 目标规划法：通过引入目标函数极值和目标的正偏差与负偏差，将求目标函数的极值

问题转换为所有目标函数与对应的目标偏差的最小值问题，进行目标规划求解。

④ 现代化人工智能算法：该算法包括遗传算法、粒子群算法等，利用这些算法直接进行多目标优化。

多目标规划问题的数学模型为

$$\min_{x,\gamma} F(\boldsymbol{x}) - \mathbf{weight}.\gamma \leqslant \mathbf{goal}$$

$$\text{s.t.}\begin{cases} c(\boldsymbol{x}) \leqslant 0 \\ \text{ceq}(\boldsymbol{x}) = 0 \\ \mathbf{A}\boldsymbol{x} \leqslant \boldsymbol{b} \\ \mathbf{Aeq}.\boldsymbol{x} = \mathbf{beq} \\ \mathbf{lb} \leqslant \boldsymbol{x} \leqslant \mathbf{ub} \end{cases}$$

其中，**weight**、**goal**、$\boldsymbol{b}$ 和 **beq** 是向量，$\boldsymbol{A}$ 和 **Aeq** 是矩阵，$F(\boldsymbol{x})$、$c(\boldsymbol{x})$ 和 ceq($\boldsymbol{x}$) 是返回向量的函数。$F(\boldsymbol{x})$、$c(\boldsymbol{x})$ 和 ceq($\boldsymbol{x}$) 可以是非线性函数，$\boldsymbol{x}$、**lb** 和 **ub** 可以作为向量或矩阵传递。

MATLAB 优化工具箱提供了 fgoalattain 函数用于求解多目标规划问题。函数的语法格式如下。

x = fgoalattain(fun,x0,goal,weight)：尝试从 x0 开始，用 weight 指定的权重更改 x，使 fun 提供的目标函数达到 goal 指定的目标。

x = fgoalattain(fun,x0,goal,weight,A,b)：求解满足不等式 Ax ≤ b 的目标达到问题。

x = fgoalattain(fun,x0,goal,weight,A,b,Aeq,beq)：求解满足等式 Aeqx = beq 的目标达到问题。如果不存在不等式，则设置 A = [] 和 b = []。

x = fgoalattain(fun,x0,goal,weight,A,b,Aeq,beq,lb,ub)：求解满足边界 lb ≤ x ≤ ub 的目标达到问题。如果不存在等式，需设置 Aeq = [] 和 beq = []。如果 x(i) 无下界，则设置 lb(i) = –Inf；如果 x(i) 无上界，则设置 ub(i) = Inf。

x = fgoalattain(fun,x0,goal,weight,A,b,Aeq,beq,lb,ub,nonlcon)：求解满足 nonlcon 所定义的非线性不等式 c(x) 或等式 ceq(x) 的目标达到问题。fgoalattain 进行优化，以满足 c(x) ≤ 0 和 ceq(x) = 0。如果不存在边界，则设置 lb = [] 和 ub = []。

x = fgoalattain(fun,x0,goal,weight,A,b,Aeq,beq,lb,ub,nonlcon,options)：使用 options 所指定的优化选项求解目标达到问题。使用 optimoptions 可设置这些选项。

x = fgoalattain(problem)：求解 problem 所指定的目标达到问题，它是 problem 中所述的一个结构体。

[x,fval] = fgoalattain(___)：对上述任何语法，返回目标函数 fun 在解 x 处计算的值。

[x,fval,attainfactor,exitflag,output] = fgoalattain(___)：返回在解 x 处的达到因子、描述 fgoalattain 退出条件的值 exitflag，以及包含优化过程信息的结构体 output。

[x,fval,attainfactor,exitflag,output,lambda] = fgoalattain(___)：返回结构体 lambda，其字段包含在解 x 处的拉格朗日乘子。

【例 3-9】利用 fgoalattain 函数求目标函数 $F(x)=\begin{bmatrix} 2+\|x-p_1\|^2 \\ 5+\dfrac{\|x-p_2\|^2}{4} \end{bmatrix}$ 的最优值。

此外，p_1 =[2,3]，且 p_2 =[4,1]，目标是 [3,6]，权重是 [1,1]，边界是$0 \leqslant x_1 \leqslant 3$、$2 \leqslant x_2 \leqslant 5$。

```
>> % 创建目标函数、目标和权重
p_1 = [2,3];
p_2 = [4,1];
fun = @(x)[2 + norm(x-p_1)^2;5 + norm(x-p_2)^2/4];
goal = [3,6];
weight = [1,1];
% 创建边界
lb = [0,2];
ub = [3,5];
% 将初始点设置为 [1,4]，并求解目标达到问题
x0 = [1,4];
A = []; % 无线性约束
b = [];
Aeq = [];
beq = [];
x = fgoalattain(fun,x0,goal,weight,A,b,Aeq,beq,lb,ub)
x =
    2.6667    2.3333
% 计算 F(x) 在解处的值
>> fun(x)
ans =
    2.8889
    5.8889
```

从结果可看出，fgoalattain 超出满足目标。由于权重相等，求解器结果溢出每个目标的量是相同的。

3.4.2　约束优化与 KKT 条件

约束条件分为等式约束与不等式约束，对于等式约束的优化问题，可以直接应用拉格朗日乘子法去求取最优值；对于含有不等式约束的优化问题，可以转换为在满足 KKT 约束条件下应用拉格朗日乘子法求解。

1. 等式约束优化

首先考虑一个不带任何约束的优化问题，对于变量 $x \in R^N$ 的函数 $f(x)$，无约束优化问题为

$$\min_x f(x)$$

当目标函数加上约束条件后，问题就变成：

$$\min_x f(x)$$
$$\text{s.t.} \quad h_i(x) = 0, \ \ i = 1,2,\cdots,m$$

约束条件会将解的范围限定在一个可行域，此时不一定能找到使 $\nabla_x f(x)$ 为 0 的点，只需找到在可行域内使 $f(x)$ 最小的值即可，常用的方法即为拉格朗日乘子法，构建为

$$L(x,\alpha) = f(x) + \sum_{i=1}^{m} \alpha_i h_i(x)$$

求解方法为：先用拉格朗日乘子法求 α 与 x：

$$\begin{cases}\nabla_x L(x,\alpha)=0\\ \nabla_\alpha L(x,\alpha)=0\end{cases}$$

令导数为 0，求得 x 、α 的值后，将 x 代入 $f(x)$ 即为在约束条件 $h_i(x)$ 下的可行解。

2. 不等式约束优化

当约束加上不等式后，情况变得更加复杂，先来看一个简单情况，给定如下不等式约束问题：

$$\begin{aligned}&\min_x f(x)\\ &\text{s.t.}\quad g(x)\leqslant 0\end{aligned}$$

对应的拉格朗日乘子为

$$L(x,\lambda)=f(x)+\lambda g(x)$$

这时的可行解必须在约束区域 $g(x)$ 内。对于不等式约束，只要满足一定的条件，依然可以使用拉格朗日乘子法解决，这里的条件便是 KKT 条件。

给出形式化的 KKT 条件不等式约束优化问题：

$$\begin{aligned}&\min_x f(x)\\ &\text{s.t.}\quad \begin{aligned}&h_i(x)=0,\ i=1,2,\cdots,m\\ &g_j(x)\leqslant 0,\ j=1,2,\cdots,n\end{aligned}\end{aligned}$$

列出拉格朗日乘子得到无约束优化问题：

$$L(x,\alpha,\beta)=f(x)+\sum_{i=1}^{m}\alpha_i h_i(x)+\sum_{j=1}^{n}\beta_i g_i(x)$$

加上不等式约束后可行解 x 需要满足的就是以下的 KKT 条件：

$$\begin{aligned}&\nabla_x L(x,\alpha,\beta)=0\\ &\beta_j g_j(x)=0,\ j=1,2,\cdots,n\\ &h_i(x)=0,\ i=1,2,\cdots,m\\ &g_j(x)\leqslant 0,\ j=1,2,\cdots,n\\ &\beta_j\geqslant 0,\ j=1,2,\cdots,n\end{aligned}$$

满足 KKT 条件后极小化拉格朗日即可得到在不等式约束条件下的可行解。

3. 约束优化的实现

在 MATLAB 中，提供了相关函数用于求解约束优化问题，下面对相应函数进行介绍。

（1）linprog 函数。

linprog 函数用于求解以下非线性规划问题

$$\begin{aligned}&\min_{\boldsymbol{x}} \boldsymbol{f}^{\mathrm{T}}\boldsymbol{x}\\ &\text{s.t.}\quad \begin{cases}\boldsymbol{A}\cdot\boldsymbol{x}\leqslant\boldsymbol{b}\\ \mathbf{Aeq}\cdot\boldsymbol{x}=\mathbf{beq}\\ \mathbf{lb}\leqslant\boldsymbol{x}\leqslant\mathbf{ub}\end{cases}\end{aligned}$$

其中，$\boldsymbol{f}$、$\boldsymbol{x}$、$\boldsymbol{b}$、**beq**、**lb** 和 **ub** 是向量，**A** 和 **Aeq** 是矩阵。

linprog 函数的语法格式如下。

x = linprog(f,A,b)：求解 min f'x，满足 Ax ≤ b。

x = linprog(f,A,b,Aeq,beq)：包括等式约束 Aeq·x = beq。如果不存在不等式，需设置 A= [] 和 b = []。

x = linprog(f,A,b,Aeq,beq,lb,ub)：定义设计变量 x 的一组下界和上界，使解 x 始终在 [lb, ub] 上。如果不存在等式，需设置 Aeq = [] 和 beq = []。

x = linprog(f,A,b,Aeq,beq,lb,ub,options)：使用 options 所指定的优化选项执行最小化。

x = linprog(problem)：求 problem 的最小值，它是 problem 中所述的一个结构体。

可以使用 mpsread 从 MPS 文件中导入 problem 结构体。还可以使用 prob2struct 从 OptimizationProblem 对象创建 problem 结构体。

对于任何输入参数，[x,fval] = linprog(___) 返回目标函数 fun 在解 x 处的值：fval = f'x。

[x,fval,exitflag,output] = linprog(___)：返回说明退出条件的值 exitflag，以及包含优化过程信息的结构体 output。

[x,fval,exitflag,output,lambda] = linprog(___)：返回结构体 lambda，其字段包含在解 x 处的拉格朗日乘子。

【例 3-10】求解具有线性不等式、线性等式和边界的简单线性规划。线性不等式约束为

$$
\begin{aligned}
&x(1)+x(2)\leqslant 2\\
&x(1)+x(2)/4\leqslant 1\\
&x(1)-x(2)\leqslant 2\\
&-x(1)/4-x(2)\leqslant 1\\
&-x(1)-x(2)\leqslant -1\\
&-x(1)+x(2)\leqslant 2
\end{aligned}
$$

利用 linprog 函数求解的代码为：

```
>> clear all;
A = [1 1; 1 1/4; 1 -1; -1/4 -1; -1 -1; -1 1];
b = [2 1 2 1 -1 2];
% 使用线性等式约束 x(1)+x(2)/4=1/2
Aeq = [1 1/4];
beq = 1/2;
% 设置以下边界：-1 ≤ x(1) ≤ 1.5、-0.5 ≤ x(2) ≤ 1.25
lb = [-1,-0.5];
ub = [1.5,1.25];
% 使用目标函数 -x(1)-x(2)/3
f = [-1 -1/3];
% 求解线性规划
x = linprog(f,A,b,Aeq,beq,lb,ub)
```

运行程序，输出如下：

```
找到最优解。
x =
    0.1875
    1.2500
```

【例 3-11】（连续投资问题）某机构现在拥有资本 205 万元，为了获取更大的收益，该机构决定将这 205 万元进行投资，有 4 个方案可供选择，投资的方式为每年年初将机构持有的所有资本都用于投资。

方案 1：从第 1 年到第 4 年的每年年初都需要投资，次年年末收回本利 1.15 万元。
方案 2：第 3 年年初投资，到第 5 年年末收回本利 1.25 万元，最大投资额为 82 万元。
方案 3：第 2 年年初投资，到第 5 年年末收到本利 1.45 万元，最大投资额为 62 万元。
方案 4：每年年初投资，每年年末收回本利 1.06 万元。
则应该采用哪种投资组合策略，使得该机构 5 年年末的总资本最大?
根据题意，得到该线性规划问题的数学描述为

$$\max f = 1.25x_{23} + 1.45x_{32} + 1.15x_{41} + 1.06x_{54}$$

$$\text{s.t.}\begin{cases} x_{11} + x_{14} = 205 \\ 1.06x_{14} - x_{21} - x_{23} - x_{24} = 0 \\ 1.15x_{11} + 1.06x_{24} - x_{31} - x_{32} - x_{34} = 0 \\ 1.15x_{21} + 1.06x_{34} - x_{41} - x_4 = 0 \\ 1.15x_{31} + 1.06x_{44} - x_{54} = 0 \\ x_{23} \leqslant 62, x_{32} \leqslant 82 \\ x_{ij \geqslant 0}(i = 1,2,\cdots,5; j = 1,2,3,4) \end{cases}$$

将目标函数转换为极小值，取目标函数中设计变量的相反数。

$$\min f = -1.25x_{23} - 1.45x_{32} - 1.15x_{41} - 1.06x_{54}$$

其实现的 MATLAB 代码为：

```
>> clear all;
c=[0 0 0 -1.45 0 0 -1.25 0 -1.15 0 -1.06]';                    % 目标函数
% 线性等式约束
Aeq=[1 1 0 0 0 0 0 0 0 0 0;0 1.06 -1 -1 -1 0 0 0 0 0 0;...
    1.15 0 0 0 1.06 -1 -1 -1 0 0 0;0 0 1.15 0 0 0 0 1.06 -1 -1 0;...
    0 0 0 0 0 1.15 0 0 0 1.06 -1];
beq=[205 0 0 0 0]';
% 设计变量的边界约束
lb=[0 0 0 0 0 0 0 0 0 0 0]';
ub=[inf inf inf 62 inf inf 82 inf inf inf inf]';
% 求最优解 x 和目标函数值 fval，由于无不等式约束，因此设置 A=[],b=[]
[x,fval,exitflag]=linprog(c,[],[],Aeq,beq,lb,ub)
```

运行程序，输出如下：

```
Optimization terminated.
x =
  125.8922
   79.1078
   21.8542
   62.0000
    0.0000
   35.0855
   82.0000
   27.6906
   54.4844
    0.0000
   40.3483
```

```
fval =
 -297.8262
exitflag =
     1
```

（2）fminbnd 函数。

fminbnd 函数用于查找单变量函数在指定区间上的最小值，如求以下条件指定的问题的最小值：

$$\min_x f(x)$$
$$\text{s.t.}\quad x_1 < x < x_2$$

其中，x、x_1 和 x_2 是有限标量，$f(x)$ 为目标函数。

fminbnd 函数的语法格式如下。

x = fminbnd(fun,x1,x2)：返回一个值 x，该值是 fun 中描述的标量值函数在区间 (x_1, x_2) 中的局部最小值。

x = fminbnd(fun,x1,x2,options)：使用 options 中指定的优化选项执行最小化计算。使用 optimset 可设置这些选项。

x = fminbnd(problem)：求 problem 的最小值，其中 problem 是一个结构体。

对于任何输入参数，[x,fval] = fminbnd(___) 返回目标函数在 fun 的解 x 处计算出的值。

[x,fval,exitflag] = fminbnd(___)：返回描述退出条件的值 exitflag。

[x,fval,exitflag,output] = fminbnd(___)：返回一个包含有关优化的信息的结构体 output。

【例 3-12】求 sin(x) 函数在 (0, 2π) 内的最小值的点。

```
>> fun = @sin;
x1 = 0;
x2 = 2*pi;
x = fminbnd(fun,x1,x2)
x =
    4.7124
>> % 为了显示精度，此值与正确值 x=3π/2 相同
3*pi/2
ans =
    4.7124
```

3.4.3　二次规划

二次规划是非线性规划中的一类特殊数学规划问题，在很多方面有应用，如投资组合、约束最小二乘问题的求解、序列二次规划、非线性优化问题等。

二次规划的标准形式为

$$\min f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x} + \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}$$
$$\text{s.t.}\quad \boldsymbol{A}\boldsymbol{x} \geqslant b$$

其中，$\boldsymbol{H}$ 是 Hessian 矩阵，$\boldsymbol{c}$，$\boldsymbol{x}$ 和 $\boldsymbol{A}$ 都是 **R** 中的向量。如果 Hessian 矩阵是半正定的，则称该规划是一个凸二次规划，在这种情况下，该问题的困难程度类似于线性规划。如果至少有一个向量满足约束并且在可行域有下界，则凸二次规划问题就有一个全局最小值。如

果是正定的，则这类二次规划为严格的凸二次规划，那么全局最小值就是唯一的。如果是一个不定矩阵，则为非凸二次规划，这类二次规划更有挑战性，因为它们有多个平稳点和局部极小值点。

在 MATLAB 中，提供了 quadprog 函数求由下式指定的问题的最小值

$$\min_x \frac{1}{2}\boldsymbol{Hx}+\boldsymbol{f}^{\mathrm{T}}\boldsymbol{x}$$

$$\text{s.t.}\quad \begin{cases} \boldsymbol{A}\cdot\boldsymbol{x}\leqslant\boldsymbol{b} \\ \mathbf{Aeq}\cdot\boldsymbol{x}=\mathbf{beq} \\ \mathbf{lb}\leqslant\boldsymbol{x}\leqslant\mathbf{ub} \end{cases}$$

H、***A*** 和 **Aeq** 是矩阵，***f***、***b***、**beq**、**lb**、**ub** 和 ***x*** 是向量。可以将 ***f***、**lb** 和 **ub** 作为向量或矩阵进行传递。

quadprog 函数的语法格式如下。

x = quadprog(H,f)：返回使 1/2x′Hx + f′x 最小的向量 x。要使问题具有有限最小值，输入数据 H 必须为正定矩阵，如果 H 是正定矩阵，则解 x = H\(−f)。

x = quadprog(H,f,A,b)：在 Ax ≤ b 的条件下求 1/2x′Hx + f′x 的最小值。输入数据 A 是由双精度值组成的矩阵，b 是由双精度值组成的向量。

x = quadprog(H,f,A,b,Aeq,beq)：在满足 Aeq · x = beq 的限制条件下求解上述问题。Aeq 是由双精度值组成的矩阵，beq 是由双精度值组成的向量。如果不存在不等式，则设置 A =[] 和 b = []。

x = quadprog(H,f,A,b,Aeq,beq,lb,ub)：在满足 lb ≤ x ≤ ub 的限制条件下求解上述问题。输入数据 lb 和 ub 是由双精度值组成的向量，这些限制适用于每个 x 分量。如果不存在等式，需设置 Aeq = [] 和 beq = []。

x = quadprog(H,f,A,b,Aeq,beq,lb,ub,x0)：从向量 x0 开始求解上述问题。如果不存在边界，需设置 lb = [] 和 ub =[]。

x = quadprog(H,f,A,b,Aeq,beq,lb,ub,x0,options)：使用 options 中指定的优化选项求解上述问题。

x = quadprog(problem)：返回 problem 的最小值，它是 problem 中所述的一个结构体。使用圆点表示法或 struct 函数创建 problem 结构体。或者，使用 prob2struct 从 OptimizationProblem 对象创建 problem 结构体。

对于任何输入变量，[x,fval] = quadprog(___)：还会返回 fval，即在 x 处的目标函数值 fval = 0.5x′Hx + f′x。

[x,fval,exitflag,output] = quadprog(___)：返回 exitflag（描述 quadprog 退出条件的整数）及 output（包含有关优化信息的结构体）。

[x,fval,exitflag,output,lambda] = quadprog(___)：返回 lambda 结构体，其字段包含在解 x 处的拉格朗日乘子。

[wsout,fval,exitflag,output,lambda] = quadprog(H,f,A,b,Aeq,beq,lb,ub,ws)：使用 ws 中的选项，从热启动对象 ws 中的数据启动 quadprog。返回的参数 wsout 包含 wsout.X 中的节点。在后续求解器调用中将 wsout 作为初始热启动对象，使用 quadprog 函数可以提高运行速度。

【例 3-13】求解如下二次规划问题。

$$f(x)=\frac{1}{2}x_1^2+x_2^2-x_1x_2-2x_1-6x_2$$

$$\text{s.t.}\begin{cases}x_1+x_2\leqslant 2\\-x_1+2x_2\leqslant 2\\2x_1+x_2\leqslant 3\\x_1,x_2\geqslant 0\end{cases}$$

将目标函数转换为标准形式：

$$f(x)=\frac{1}{2}(x_1,x_2)\begin{pmatrix}1&-1\\-1&2\end{pmatrix}\begin{pmatrix}x_1\\x_2\end{pmatrix}+(-2,6)\begin{pmatrix}x_1\\x_2\end{pmatrix}$$

其实现的 MATLAB 代码如下：

```
>> clear all;
H = [1 -1; -1 2];
f = [-2; -6];
A = [1 1; -1 2; 2 1];
b = [2; 2; 3];
lb = zeros(2,1);
[x,fval,exitflag,output,lambda] =quadprog(H,f,A,b,[],[],lb)
```

运行程序，输出如下：

```
Optimization terminated.
x =
    0.6667
    1.3333
fval =   -8.2222
exitflag =     1
output =
         iterations: 3
    constrviolation: 1.1102e-016
          algorithm: 'medium-scale: active-set'
      firstorderopt: []
cgiterations: []
            message: 'Optimization terminated.'
lambda =
      lower: [2x1 double]
      upper: [2x1 double]
      eqlin: [0x1 double]
    ineqlin: [3x1 double]
```