第 3 章 二次型与数值优化

3.1 二次型及其矩阵表示

含有 n 个变量 $x_1,x_2,\cdots,x_n$ 的二次齐次函数

$$f(x_1,x_2,\cdots,x_n)=a_{11}x_1^2+a_{22}x_2^2+\cdots+a_{nn}x_n^2+2a_{12}x_1x_2+\cdots+2a_{1n}x_1x_n+\cdots+2a_{n-1,n}x_{n-1}x_n$$

称为 n 元二次型，简称为二次型。利用矩阵乘法，二次型 f 可以表示为 $f=\boldsymbol{X}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{X}$，其中 $\boldsymbol{A}=\begin{bmatrix}a_{11}&a_{12}&\cdots&a_{1n}\\a_{21}&a_{22}&\cdots&a_{2n}\\\vdots&\vdots&\ddots&\vdots\\a_{n1}&a_{n2}&\cdots&a_{nn}\end{bmatrix}$ 是一个实对称矩阵，$\boldsymbol{X}=\begin{bmatrix}x_1\\x_2\\\vdots\\x_n\end{bmatrix}$ 是变量向量。

例 3-1 求二次型 $f(x_1,x_2,x_3)=x_1^2+4x_2^2+4x_3^2-4x_1x_2+4x_1x_3-8x_2x_3$ 的矩阵 $\boldsymbol{A}$，并计算该二次型的秩。

```
import sympy as sp
x1, x2, x3 =sp.symbols('x1 x2 x3')                    #定义符号变量 x1,x2,x3
#定义二次型多项式
f =x1**2 +4 * x2**2 +4 * x3**2 -4 * x1 * x2 +4 * x1 * x3 -8 * x2 * x3
#使用 sp.hessian()函数直接生成二次型矩阵
A =sp.hessian(f, (x1, x2, x3)) / 2
print(A)
```

上述代码的输出结果：

```
Matrix([[ 1, -2,  2],
        [-2,  4, -4],
        [ 2, -4,  4]])
print(A.rank())
```

上述代码的输出结果：

```
1                                                     #该二次型的秩为 1
```

例 3-2 求实对称矩阵 $\boldsymbol{A}=\begin{bmatrix}2&1\\1&3\end{bmatrix}$ 所对应的二次型 f。

```
import sympy as sp

def build_quadratic_form(A, symbols):
      X = sp.Matrix(symbols)                                  #列向量
      return X.T @A @X                                        #返回二次型

x1, x2 = sp.symbols('x1 x2')                                  #定义符号变量 x1,x2
A = sp.Matrix([[2, 1], [1, 3]])                               #定义实对称矩阵 A
variables = [x1, x2]
quad_form = build_quadratic_form(A, variables)
#展开二次型
simplified_form = sp.expand(quad_form[0, 0])
print(simplified_form)
```

上述代码的输出结果：

```
2 * x1**2 + 2 * x1 * x2 + 3 * x2**2
```

根据上述代码的输出结果可知，实对称矩阵 $\boldsymbol{A}$ 对应的二次型为 $f(x_1,x_2)=2x_1^2+2x_1x_2+3x_2^2$。

3.2 二次型的标准形与惯性定律

如果二次型

$$f=\boldsymbol{X}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{X} \tag{3-1}$$

通过可逆线性变换 $\boldsymbol{X}=\boldsymbol{C}\boldsymbol{Y}$ 转换为关于 $\boldsymbol{Y}$ 的二次型

$$f=k_1y_1^2+k_2y_2^2+\cdots+k_ny_n^2 \tag{3-2}$$

其中，$\boldsymbol{C}$ 是一个可逆矩阵，$k_1,k_2,\cdots,k_n$ 是实数。这种形式的二次型(3-2)称为二次型(3-1)的一个标准形。在这种标准形中，二次型仅包含变量的平方项，没有交叉项，从而简化了分析和处理。

在二次型的标准形中：

- 系数为正的平方项的个数称为二次型的正惯性指数；
- 类似地，系数为负的平方项的个数称为二次型的负惯性指数；
- 正惯性指数与负惯性指数的和称为惯性指数。

设有二次型 $f=\boldsymbol{X}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{X}$，若对于任何非零向量 $\boldsymbol{X}\neq0$，恒有 $f=\boldsymbol{X}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{X}>0$，则称 f 为正定二次型，并称矩阵 $\boldsymbol{A}$ 为正定矩阵。

例 3-3 已知二次型

$$f(x_1,x_2,x_3)=x_1^2-2x_2^2+x_3^2+2x_1x_2-4x_1x_3+2x_2x_3$$

(1) 计算标准形；(2) 计算正惯性指数；(3) 判断二次型是否正定。

```
>>>import sympy as sp
>>>x1, x2, x3 = sp.symbols('x1 x2 x3')                       #定义符号变量
#定义二次型多项式
>>>f = x1**2-2 * x2**2+x3**2+2 * x1 * x2-4 * x1 * x3+2 * x2 * x3
```

```
#使用 sp.hessian()函数直接生成二次型矩阵
>>>A =sp.hessian(f, (x1, x2, x3)) / 2
>>>A
Matrix([
[ 1, 1, -2],
[ 1, -2, 1],
[-2, 1, 1]])
>>>eigenvals =A.eigenvals()                                    #计算特征值
>>>eigenvals                                                   #3 个特征值都不重复
{-3: 1, 3: 1, 0: 1}                                           #3 个特征值的出现次数都为 1
#提取并降序排列特征值
>>>eigenvalues =sorted(eigenvals.keys(), reverse=True)
>>>eigenvalues
[3, 0, -3]
>>>eigenvecs =A.eigenvects()                                   #计算特征向量
#存储格式:特征值,特征值重复出现的次数,对应的特征向量
>>>eigenvecs[0]                                               #查看第一个特征向量
(-3, 1, [Matrix([ [ 1],
                  [-2],
                  [ 1]])])
#提取并排序特征向量(依据对应特征值从大到小的顺序)
>>>eigenvectors =[vec[2][0] for vec in sorted(eigenvecs, key=lambda x: -x[0])]
>>>for vec in eigenvectors:
    print(vec)
Matrix([[-1], [0], [1]])
Matrix([[1], [1], [1]])
Matrix([[1], [-2], [1]])
>>>P =sp.Matrix.hstack(*eigenvectors)                          #构造正交矩阵①
>>>P
Matrix([[-1, 1,  1],
        [ 0, 1, -2],
        [ 1, 1,  1]])
>>>D =P.inv() @A @P                                           #计算对角矩阵
>>>D
Matrix([ [3, 0,  0],
        [0, 0,  0],
        [0, 0, -3]])
#计算标准形
>>>y1, y2, y3 =sp.symbols('y1 y2 y3')
>>>Y =sp.Matrix([y1, y2, y3])
>>>standard_form =sp.simplify(Y.T @D @Y)[0, 0]
>>>standard_form
3*y1**2 -3*y3**2                                              #标准形:3y1^2 - 3y3^2
```

① hstack()函数用于水平堆叠(horizontal stack)数组。

```
#计算正惯性指数
>>>positive_inertia =sum(1 for val in eigenvalues if val >0)
>>>positive_inertia
1
#判断二次型是否正定
>>>is_positive_definite =all(val >0 for val in eigenvalues)
>>>is_positive_definite
False
```

知识点补充：

all()函数用于检查一个可迭代对象(如列表、元组、集合等)中的所有元素是否都等价于 True。如果所有元素都为 True(或者在布尔上下文中被视为 True),则 all()函数返回 True;否则,只要有一个元素为 False(或者在布尔上下文中被视为 False),则返回 False。

```
>>>all([1, 'hello'])
True
```

下列代码展示了如何使用 NumPy 库中的 hstack()和 vstack()函数对数组进行水平和垂直堆叠操作。

```
>>>a =np.array([1, 2, 3])
>>>b =np.array([4, 5, 6])
>>>np.hstack((a, b))                                    #水平堆叠
array([1, 2, 3, 4, 5, 6])
>>>np.vstack((a, b))                                    #垂直堆叠(vertical stack)
array([[1, 2, 3],
       [4, 5, 6]])
```

使用单星(*)操作符从数据容器中解包(unpacking)数据。

```
>>>data =[1, 2, 3]
>>>print(data)
[1, 2, 3]
>>>print(*data)                                         #观察解包与不解包时不同的输出结果
1 2 3
```

3.3 正交变换二次型为标准形

用正交变换将二次型化为标准形的步骤如下：

(1) 写出二次型对应的对称矩阵 $\boldsymbol{A}$；

(2) 计算特征值与特征向量：求出矩阵 $\boldsymbol{A}$ 的所有特征值 $\lambda_1,\lambda_2,\cdots,\lambda_n$ 及其对应的特征向量；

(3) 正交归一化特征向量：若特征值不同,则对应特征向量自然正交;若存在重根,则需使用施密特正交化方法对其进行正交化,再进行单位化,最终得到一组标准正交基；

(4) 构造正交矩阵：将上述单位化特征向量作为列向量,构造正交矩阵 $\boldsymbol{P}$,满足 $\boldsymbol{P}^{\mathrm{T}}\boldsymbol{AP}=\boldsymbol{D}$；

(5) 变量替换：令 $\boldsymbol{Y}=\boldsymbol{P}^{\mathrm{T}}\boldsymbol{X}$,则原二次型 $f(\boldsymbol{X})=\boldsymbol{X}^{\mathrm{T}}\boldsymbol{AX}$ 转换为 $f(\boldsymbol{Y})=\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{DY}$,其中 $\boldsymbol{D}$ 是对角

矩阵，其主对角线元素是 $\boldsymbol{A}$ 的特征值；

(6) 写出标准形：新的二次型表示为 $f(\boldsymbol{Y})=\lambda_1 y_1^2+\lambda_2 y_2^2+\cdots+\lambda_n y_n^2$。

例 3-4　已知二次型 $f(x_1,x_2,x_3,x_4)=x_1^2+x_2^2+x_3^2+x_4^2+2x_1x_2-2x_1x_4+2x_2x_3-2x_3x_4$，用正交变换将其化为标准形。

```
>>> from sympy import GramSchmidt
>>> x1, x2, x3, x4 = sp.symbols("x1 x2 x3 x4")        #定义符号变量
#定义二次型多项式
>>> f = x1**2 + x2**2 + x3**2 + x4**2 + 2 * x1 * x2 - 2 * x1 * x4 + 2 * x2 * x3 - 2 * x3 * x4
#使用 sp.hessian()函数直接生成二次型矩阵
>>> A = sp.hessian(f, (x1, x2, x3, x4)) / 2
>>> A
Matrix([
[ 1, 1,  0, -1],
[ 1, 1,  1, 0],
[ 0, 1,  1, -1],
[-1, 0, -1,  1]])
>>> eigenvals = A.eigenvals()                          #计算特征值
>>> eigenvals
{3: 1, -1: 1, 1: 2}                                    #特征值 1 重复出现了 2 次
>>> eigenvalues = list(eigenvals.keys())               #提取特征值
>>> eigenvalues
[3, -1, 1]
>>> eigenvecs = A.eigenvects()                         #计算特征向量
#输出特征值、它们的重数(即重复次数)以及相应的特征向量
>>> for val, multiple, vecs in eigenvecs:
    print(f"特征值: {val},重复次数: {multiple},对应的特征向量: {[v.T for v in vecs]}")
```

上述代码的输出结果：

```
特征值: -1,重复次数: 1,对应的特征向量: [Matrix([[1, -1, 1, 1]])]
特征值: 1,重复次数: 2,对应的特征向量: [Matrix([[-1, 0, 1, 0]]), Matrix([[0, 1, 0, 1]])]
特征值: 3,重复次数: 1,对应的特征向量: [Matrix([[-1, -1, -1, 1]])]
#排序特征值和特征向量
>>> eigenvecs_sorted = sorted(eigenvecs, key=lambda x: -x[0])
#提取并归一化特征向量
>>> eigenvectors = []
>>> for val, mult, vecs in eigenvecs_sorted:
        if mult == 1:
            #单个特征向量归一化,注意使用 append(),而不是 extend()方法
            eigenvectors.append(vecs[0] / vecs[0].norm())
        else:
            ortho_vecs = GramSchmidt(vecs, orthonormal=True)
            eigenvectors.extend(ortho_vecs)
#构造正交矩阵 P
```

```
>>> P = sp.Matrix.hstack( * eigenvectors)
>>> assert (P.T * P - sp.eye(4)).norm() < 1e-10, "P 不是正交矩阵"
>>> P
Matrix([
[-1/2, -sqrt(2)/2,          0,  1/2],
[-1/2,           0, sqrt(2)/2, -1/2],
[-1/2,  sqrt(2)/2,          0,  1/2],
[ 1/2,           0,  sqrt(2)/2,  1/2]])
>>> D = P.T * A * P                                    #计算对角矩阵 D
>>> D
Matrix([
[3, 0, 0,  0],
[0, 1, 0,  0],
[0, 0, 1,  0],
[0, 0, 0, -1]])
#构造标准形
>>> y = sp.Matrix(sp.symbols("y1 y2 y3 y4"))
>>> y
Matrix([
[y1],
[y2],
[y3],
[y4]])
>>> y.shape
(4, 1)
>>> standard_form = y.T * D * y
>>> standard_form
Matrix([[3 * y1**2 + y2**2 + y3**2 - y4**2]])
>>> standard_form.shape
(1, 1)
>>> standard_form[0, 0]
3 * y1**2 + y2**2 + y3**2 - y4**2
```

由上述代码可知，正交变换为 $\boldsymbol{y}=\boldsymbol{P}^{\mathrm{T}}\boldsymbol{x}$，二次型的标准形为 $f=3y_1^2+y_2^2+y_3^2-y_4^2$。

知识点补充：

```
>>> vector = sp.Matrix([3, 4])                         #定义一个向量
>>> vector.norm()                                      #计算 L2 范数(欧几里得范数)
5
>>> vector / vector.norm()                             #向量标准化(单位向量)
Matrix([
[3/5],
[4/5]])
```

3.4　数值优化算法

数值优化算法是用于寻找函数极值(最小值或最大值)的方法。下面介绍3种常见的数值优化算法,它们分别是牛顿法(Newton methods)、拟牛顿法(quasi-Newton methods)和共轭梯度法(conjugate gradient methods)。

1. 牛顿法

牛顿法是一种基于目标函数二阶泰勒展开的迭代优化算法,它利用函数的一阶导数和二阶导数信息来寻找极值点,具有较快的收敛速度。对于单变量目标函数 $f(x)$,牛顿法的迭代更新公式为

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

其中,$f'(x_k)$ 表示函数 $f(x)$ 在点 x_k 处的一阶导数,$f''(x_k)$ 表示在该点的二阶导数。

对于具有两个变量的目标函数 $f(x_1,x_2)$,牛顿法的迭代更新公式为 $x_{k+1}=x_k-\boldsymbol{H}^{-1}(x_k)\nabla f(x_k)$。其中,$x_k=[x_{1,k}x_{2,k}]^{\mathrm{T}}$;梯度向量 $\nabla f(x_k)=\left[\frac{\partial f}{\partial x_{1,k}}\ \frac{\partial f}{\partial x_{2,k}}\right]^{\mathrm{T}}$;Hessian 矩阵 $\boldsymbol{H}(x_k)=\begin{bmatrix}\frac{\partial^2 f}{\partial x_{1,k}^2} & \frac{\partial^2 f}{\partial x_{1,k}\partial x_{2,k}}\\ \frac{\partial^2 f}{\partial x_{2,k}\partial x_{1,k}} & \frac{\partial^2 f}{\partial x_{2,k}^2}\end{bmatrix}$。$\boldsymbol{H}^{-1}(x_k)$ 表示该 Hessian 矩阵的逆矩阵。

对于一般的 n 维目标函数 $f(x)$,牛顿法的迭代更新公式为 $x_{k+1}=x_k-\boldsymbol{H}^{-1}(x_k)\nabla f(x_k)$。其中,$\nabla f(x_k)\in\mathbb{R}^n$ 是梯度向量,$\boldsymbol{H}(x_k)\in\mathbb{R}^{n\times n}$ 是 Hessian 矩阵。

```
def newton_method_2d(f, grad_f, hessian_f, x0, tol=1e-6, max_iter=100):
    """
    使用牛顿法在二维空间中寻找目标函数 f 的极小值点
    参数:
        f:目标函数,输入为 x(数组或元组),输出是一个标量
        grad_f:f 的梯度函数,返回梯度向量
        hessian_f:f 的 Hessian 矩阵函数,返回一个 2×2 的矩阵
        x0:初始猜测值,一维数组,形如[x0, y0]
        tol:收敛精度阈值,当更新步长小于该值时停止迭代
        max_iter:最大迭代次数
    返回值:近似极小值点的坐标 x
    """
    x =x0                                          #初始化当前点为初始猜测值
    for i in range(max_iter):
        gradient =grad_f(x)                        #计算当前点的梯度
        hessian =hessian_f(x)                      #计算当前点的 Hessian 矩阵
        try:
            #求解线性方程组 H * delta_x =gradient,得到 delta_x
            delta_x =np.linalg.solve(hessian, gradient)
        except np.linalg.LinAlgError:
```

```
            #如果 Hessian 矩阵不可逆,抛出异常并终止迭代
            print("Hessian 矩阵是不可逆的!")
            break                                    #终止迭代
        #判断是否收敛:如果步长足够小,说明已经接近极小值点
        if np.linalg.norm(delta_x) <tol:
            break
        x -=delta_x                                  #更新 x 值
    return x                                         #返回找到的最优解

def f(x):                                            #定义目标函数(x-1)^2+(y-2)^2
    return (x[0] -1)**2 +(x[1] -2)**2

def grad_f(x):                                       #定义目标函数的梯度(一阶导数)
    return np.array([2 * (x[0] -1), 2 * (x[1] -2)])

#定义目标函数的 Hessian 矩阵(二阶导数)
def hessian_f(x):
    return np.array([[2, 0], [0, 2]])

x0 =np.array([0.0, 0.0])                             #初始猜测点

optimal_x =newton_method_2d(f, grad_f, hessian_f, x0)    #运行牛顿法
print(f'最优解: { optimal_x }')
```

上述代码的输出结果:

```
最优解: [1. 2.]
```

2. 拟牛顿法

拟牛顿法旨在避免直接计算目标函数的 Hessian 矩阵,从而降低计算复杂度,同时保持较快的收敛速度。该类方法在每次迭代中通过更新一个近似的 Hessian 矩阵或其逆矩阵,逐步逼近真实的二阶导数信息。其中,DFP(Davidon-Fletcher-Powell)和 BFGS(Broyden-Fletcher-Goldfarb-Shanno)算法是最具代表性的两种拟牛顿法。

```
from scipy.optimize import minimize①
import numpy as np

def f(x):                                            #定义目标函数
    return (x[0] -1)**2 +(x[1] -2)**2

def grad_f(x):                                       #定义目标函数的梯度
    return np.array([2 * (x[0]-1), 2 * (x[1]-2)])

x0 =np.array([0.0, 0.0])                             #初始猜测点
```

① SciPy 中的 scipy.optimize.minimize 已经内置了牛顿法的多种变体,支持自动计算梯度和 Hessian 矩阵,适用于大多数非线性优化问题。

```
result =minimize(f, x0, method="BFGS",          #使用 BFGS 拟牛顿法
                        jac=grad_f,             #提供梯度函数
                        options={
                              'gtol':1e-6,      #梯度容差(gradient tolerance)
                              'disp':True       #显示收敛信息 Discription
                        })
print(f"最优解: {result.x}")
```

上述代码的输出结果：

```
Optimization terminated successfully.
    Current function value: 0.000000
    Iterations: 2
    Function evaluations: 3
    Gradient evaluations: 3
最优解: [1. 2.]
```

3. 共轭梯度法

共轭梯度法是一种用于求解大规模无约束优化问题的高效迭代算法，尤其适用于具有对称正定 Hessian 矩阵的二次目标函数。该方法无须直接计算或存储完整的 Hessian 矩阵，而是通过构造一组关于 Hessian 矩阵共轭的方向，从而在较少的迭代次数内快速逼近最优解。

在每次迭代中，共轭梯度法不仅利用当前点的梯度信息，还结合前一步的搜索方向，以确保新方向与之前所有方向关于 Hessian 矩阵是共轭的，即满足某种广义的正交性。

具体来说，对于一个给定的正定二次目标函数 $f(x)=\frac{1}{2}x^{\mathrm{T}}\boldsymbol{A}x-b^{\mathrm{T}}x+c$，其中 $\boldsymbol{A}$ 是对称正定矩阵，共轭梯度法通过以下步骤逐步逼近最小值点。

- 初始化：选择初始点 x_0，并令初始搜索方向 $d_0=-\nabla f(x_0)$。
- 迭代更新：在每一步迭代中，首先计算最优步长 α_k，然后更新当前位置为 $x_{k+1}=x_k+\alpha_k d_k$；接着根据新的梯度信息计算下一个搜索方向 d_{k+1}，确保其与前一方向满足共轭条件。

共轭梯度法的主要优点在于，它不需要显式地计算或存储 Hessian 矩阵，同时对于严格凸的二次函数，能够在有限步数内精确收敛到最优解。此外，由于仅需存储少量向量，如梯度和搜索方向，该方法特别适合求解大规模稀疏系统。

```
from scipy.optimize import minimize
import numpy as np

def f(x):                                       #定义目标函数
     return (x[0] -1)**2 +(x[1] -2)**2

def grad_f(x):                                  #定义目标函数的梯度
     return np.array([2 * (x[0]-1), 2 * (x[1]-2)])

#定义目标函数的 Hessian 矩阵(二阶导数)
```

```
def hessian_f(x):
    return np.array([[2, 0], [0, 2]])

x0 =np.array([0.0, 0.0])                          #初始猜测点
result =minimize(f, x0, method="Newton-CG"①, jac=grad_f, hess=hessian_f)
print(f"最优解: {result.x}")                       #最优解: [1. 2.]
```

3.5 小结

本章介绍了二次型的基本概念，包括如何从二次型多项式中提取系数并求出系数矩阵的秩。此外，还详细讲解了通过正交变换将二次型化为标准形的方法，计算其惯性指数，并判断该二次型是否为正定二次型。

数值优化算法通过迭代逼近函数极值点。牛顿法借助二阶导数实现快速收敛但计算负担较重，拟牛顿法则通过近似二阶信息在效率与精度间取得平衡，而共轭梯度法凭借独特的共轭方向设计，尤其适合处理高维稀疏问题。这些方法的共同特点是以梯度信息为基础，根据问题规模与计算条件灵活选用。

3.6 习题 3

1. 矩阵的秩(rank)是一个重要的概念，查阅资料简要列举其主要意义。

2. 查阅资料并回答 Python 的可迭代对象有哪些？（至少写出 3 种）

3. 已知 $\boldsymbol{x}=\begin{bmatrix}x_1\\x_2\end{bmatrix}$，$\boldsymbol{A}=\begin{bmatrix}3&-2\\-2&6\end{bmatrix}$，计算$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}$。

4. 求下列二次型的系数矩阵，并计算其秩。

(1) $f=3x_1x_2+6x_1x_3-x_2x_3-3x_3^2$；

(2) $f=5x_1^2+2x_3^2+x_1x_3-5x_2x_4-10x_3x_4$。

5. 写出实对称矩阵 $\boldsymbol{A}=\begin{bmatrix}2&1&-2\\1&0&4\\-2&4&6\end{bmatrix}$ 的二次型 f。

6. 求解正交变换 $\boldsymbol{X}=\boldsymbol{P}\boldsymbol{Y}$，把下列二次型化为标准形。

$$f=x_1^2+x_2^2+10x_1x_2$$

7. 设二次型

$$f=x_1^2+x_2^2+x_3^2+x_4^2+2x_1x_2+2x_3x_4$$

求二次型 f 的秩及其正、负惯性指数。

8. 判断下列二次型的正定性。

$$f=x_1^2-5x_2^2+x_3^2+6x_1x_2+6x_1x_3+6x_2x_3$$

9. 已知二次型

① Newton-CG 表示牛顿-共轭梯度法。

$$f = 2x_1^2 + 3x_2^2 + 3x_3^2 + 2mx_2x_3$$

可用正交变换为 $f = y_1^2 + 2y_2^2 + 5y_3^2$，求 m 和所用的正交变换。

10. 求矩阵 $\boldsymbol{A} = \begin{bmatrix} 1 & 1 \\ -2 & 2 \\ 2 & -2 \end{bmatrix}$ 的一个奇异值分解。

11. 对于二次型 $f(x) = x_1^2 + 3x_2^2 - 2x_1x_2$：(1)手动计算梯度 ∇f 和 Hessian $\boldsymbol{H}$；(2)用牛顿法实现求解(初始点$x_0 = [5,5]$)，验证是否一步收敛。

12. 对于二次型 $f(x) = \frac{1}{2}x^{\mathrm{T}}\begin{bmatrix} 4 & 1 \\ 1 & 3 \end{bmatrix}x - [1 \quad 2]\,x$，手动推导前两步共轭梯度法的搜索方向 d_0, d_1 和迭代点 x_1, x_2。