2022 年秋季学期信息学院专业硕士研究生课程

《工程数学》课程报告

姓名：_____

学号：_____

专业：_____

年　　月　　日

要求：

（1）熟悉和掌握本课程中讲解的各种方法，按照题目要求完成各实验题，除特殊要求外，字数和页码没有限制；

（2）格式要求：行距为 1.5，中文 5 号宋体，符号、变量、英文字符等用 Times New Roman；

（3）双面打印一份，2023 年春季学期开学第一周交（见群内通知）。

第一部分：简答题（请简要回答以下问题，每小题字数不少于 200 字）

1. 为什么要讨论误差？在实际问题研究中如何发现误差和处理误差？

在一定的条件下讨论误差得到更接进于真实值的最佳测量结果；确定结果的不确定程度；据预先所需结果，选择合理的实验仪器、实验条件和方法或者模型，以降低成本和缩短实验时间。在实际问题研究中，通常有模型误差(由实际问题抽象得到的数学模型，一般都带有误差，这种误差称为模型误差)，观测误差(数学模型中包含的一些由观测得到的物理量（如温度、电阻、长度）或由物理量估算出的模型参数，常常与实际数据之间存在误差，这种误差称为观测误差)，截断误差(求解数学模型所用的数值方法通常是一种近似方法，这种因计算方法产生的误差称为截断误差)，舍入误差(由于计算机只能对有限位数进行运算，计算时按四舍五入进行舍入，或因计算机字长有限，导致数据在存储时进行舍入而产生的误差,这种误差称为舍入误差)等不同种类的误差. 观测误差和模型误差可以通过换用更好的实验设备和更高精度的数学模型来减小。截断误差也可以通过使用更高精度计算方法来减小。舍入误差可以通过使用更长字长的计算机来实现。

2. 在数据处理过程中发现部分数据缺失时，有下述四种方法可以进行处理：（a）利用散点插值求缺失数据；（b）构造插值函数求缺失数据；（c）采用数据拟合求缺失数据；（d）将缺失数据直接删除。你会选择哪一种？请说明理由。

答：使用哪种方法取决于数据的特点和处理数据的目的。如果数据的某个特征缺失比较多比如 80%的数据缺失，我会选择直接删除这个特征。在已知点列的情况下，如果要从整体上靠近数据，则使用数据拟合的方式。如果已知点列并且完全经过点列，则选择插值的方式。比如工资和工龄大致呈线性关系，则可以用线性拟合的方式求缺失数据。又比如在医学图像分割中常用的上采样就是在原图像基础上进行插值得到的。

3. 在 Newton-Cutes 求积公式 $\int_a^b f(x)dx = (b-a)\sum_{i=0}^{n} C_i^{(n)} f(x_i)$ 中，当系数 $C_i^{(n)}$ 是负值时，公式的稳定性不能得到保证，请解释原因并给出一种解决方案。

答：设 $C_i^{(n)}$, i=0, 1,..., n 无误差，但 f(x_i)有误差，但$\tilde{f}(x_i)$有误差，$\tilde{f}(x_i) \approx \mathrm{f}(\mathrm{x_i})$, 反映在数值积分中

$$\sum_{i=0}^{n} C_i^{(n)} \tilde{f}(x_i) \approx \sum_{i=0}^{n} C_i^{(n)} f(x_i)$$

令$\varepsilon = \max_{0\le i\le n}\left|\tilde{f}(x_i) - f(x_i)\right|$, 则有

$$\left|\mathrm{I_n}(\tilde{f}) - \mathrm{I_n}(\mathrm{f})\right| = \left|\sum_{i=0}^{n} C_i^{(n)} \tilde{f}(x_i) - \sum_{i=0}^{n} C_i^{(n)} f(x_i)\right|$$

$$= \left|\sum_{i=0}^{n} C_i^{(n)} [\tilde{f}(x_i) - f(x_i)]\right| \le \sum_{\mathrm{i}=0}^{\mathrm{n}} \left|\mathrm{C_i^{(n)}}\right| \left|\tilde{f}(x_i) - f(x_i)\right| \le \varepsilon \sum_{\mathrm{i}=0}^{\mathrm{n}} \left|\mathrm{C_i^{(n)}}\right|$$

当$C_i^{(n)} > 0$ 时， 得$\left|\sum_{i=0}^{n} C_i^{(n)} \tilde{f}(x_i) - \sum_{i=0}^{n} C_i^{(n)} f(x_i)\right| \le \varepsilon$，又因为$\sum_{i=0}^{n} C_i^{(n)} = 1$，则当$C_i^{(n)}$出现负数时，

$$\left|\sum_{i=0}^{n} C_i^{(n)}\right| > \sum_{i=0}^{n} C_i^{(n)} = 1$$

出现数值不稳定的现象。因为

$$C_i^{(n)} = \frac{(-1)^{n-i}}{i!\,(n-i)!\,n\int_0^n \prod_{k=0,k\neq ii}^{n}[t-k]dt}$$

当 $n \geq 8$ 时，$C_i^{(n)}$出现负数，所以当 $n \geq 8$ 时不采用牛顿柯蒂斯公式。可以将一个积分区间分为多个子区间用低阶的牛顿柯蒂斯公式进行计算后相加。

4. 求解方程组 $Ax = b$ 的迭代格式 $x^{(k+1)} = Mx^{(k)} + f$ 收敛的充要条件是什么？为什么需要这样的条件？

答：M 的谱半径<1 为迭代格式收敛的充要条件。和普通级数收敛类似，不过是多维的， 把绝对值换为模谱半径小于 1，就是说存在 r， 0<r<1，使得|Ax|<=r|x|，对任意的向量 x，这样迭代 n 次后，|A^n x|<=r|A^(n-1) x|<=r^2|A^(n-2) x|<=....<=r^n |x|，对任意的 x,这样不管初始向量 x 取什么值， 总是固定的常数， 而 r^n 是收敛到 0 的

5. 你学过的约束或无约束优化方法有哪些？请对其中一种方法进行原理描述，并举例说明其在机器学习或实际问题求解中是如何应用的。

最速下降法，又称梯度法，基本思路是从某一点出发，选择一个目标函数值下降最快的方向，尽快达到最小点。

考虑无约束问题

$$\min f(x), x \in R^n$$

最速下降方向是目标函数的负梯度方向：

$$d = -\nabla f(x)$$

最速下降的迭代公式为

$$x^{(k+1)} = x^{(k)} + \lambda_k d^{(k)}$$

λ_k为一维搜索的步长，满足：

$$f(x^{(k)} + \lambda_k d^{(k)}) = \min_{\lambda \geq 0} f(x^{(k)} + \lambda d^{(k)})$$

在神经网络的训练中，通常使用 SGD(最速下降法的一种特殊形式)来优化参数，使得损失函数最小化。

6. 在本课程所学的方法中，哪些方法将对你的研究有帮助？哪些方法没有帮助？请举例说明。

答：

1. 我研究的是用深度学习来实现 3D 中的点云配准和点云超分的方法。本课程中的优化方法(比如优化梯度法)是神经网络训练中比较关键的内容。

2. 学习数值计算方法中的误差理论有助于理解算法部署过程中产生的误差。比如将深度学习模型参数由 FP32 转为 INT8 后模型的精度下降的问题，又比如实现硬件 FFT 时，如何确定数据位宽，如何设计乘法器等等。

3. 通过本课程的学习我熟悉了 Python 编程和 Latex 排版，这对之后深度学习代码的编写和论文的排版有帮助。

第二部分：基础题（请按要求完成以下问题）

1. 在四位有效数字的精度下计算积分 $y_n = \int_0^1 \frac{x^n}{x+5}dx, \quad (n = 0,1,2,\cdots,100)$，有以下两个算法：

算法 1：$\begin{cases} y_0 = \int_0^1 \frac{1}{x+5} dx = \ln 6 - \ln 5 \approx 0.1823 \\ y_n = \frac{1}{n} - 5y_{n-1}, \quad n = 1,2,\cdots,100 \end{cases}$

算法 2：$\begin{cases} y_{100} = 0.1815 \times 10^{-2} \\ y_{n-1} = \frac{1}{5n} - \frac{1}{5} y_n, \quad n = 100,99,\cdots,1 \end{cases}$

请对这两个算法进行稳定性分析，说明哪一个算法更好。

答：使用 Python 编程，分别迭代 100 次观察数值的大小。

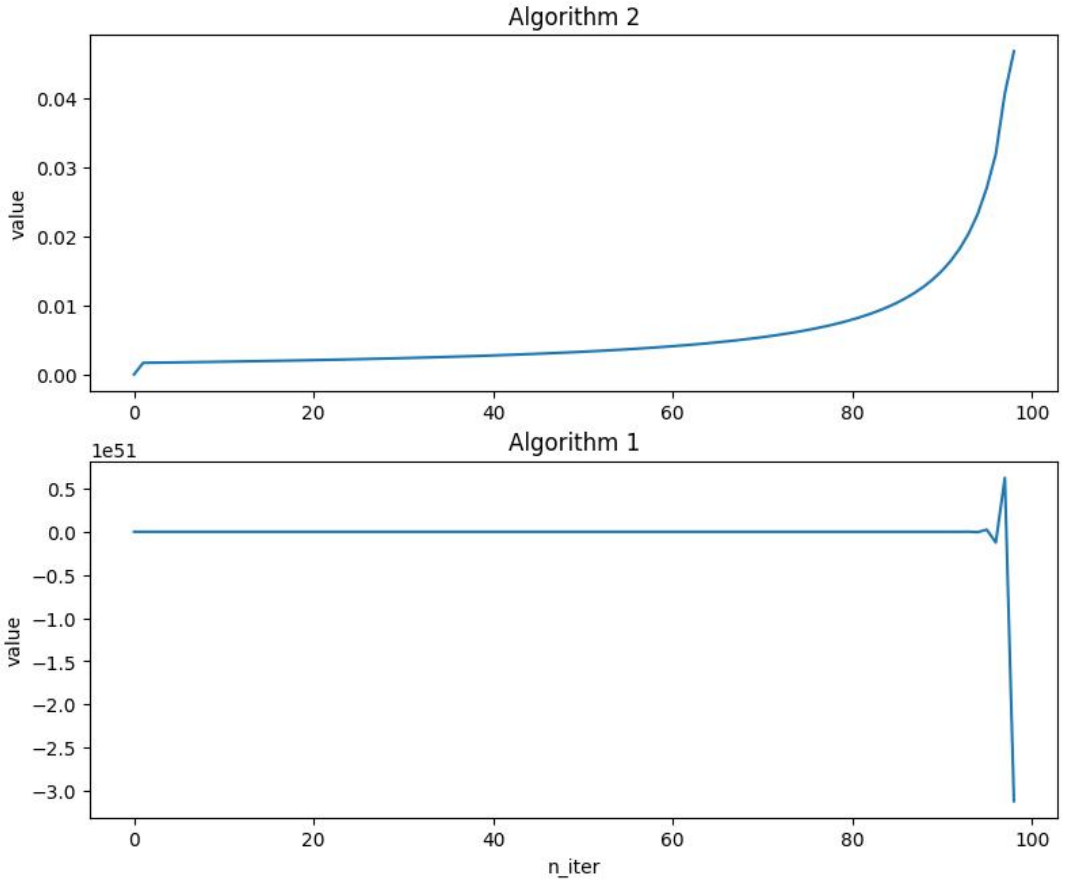


图 1 两种迭代方式的对比

算法 1 在多次迭代后数值过大，而算法 2 多次迭代后收敛到了 0.045。左右显然算法 1 的稳定性不如算法 2。

代码：

```
1.  import numpy as np
2.  import matplotlib.pyplot as plt
3.
4.
5.  def function1(n_iter=100): #方法 1
```

```
    y = np.zeros([n_iter, 1])
    y[0] = np.log(6) - np.log(5)
    for i in range(1, n_iter):
        y[i] = 1 / i - 5 * y[i - 1]
    return y


def function2(n_iter=99): #方法 2
    y = np.zeros([n_iter, 1])
    y[n_iter-1] = 0.1815 * 10 ** (-2)
    for i in range(2, n_iter):
        y[n_iter - i] = 1 / (5 * i) - (1 / 5) * y[n_iter - i + 1]
    return y


n = 100
x = np.linspace(0, n, n)
#迭代
y1 = function1(n)
y2 = function2(n)

#绘图
plt.subplot(2, 1, 1)
plt.plot(x, y2)
plt.title('Algorithm 2')
plt.ylabel('value')
plt.subplot(2, 1, 2)
plt.plot(x, y1)
plt.xlabel('n_iter')
plt.ylabel('value')
plt.title('Algorithm 1')
plt.show()
```

2. 已知函数 $f(x)$ 的数表如下，请用两种插值方法计算 $f(0.596)$ ，给出方法描述和计算过程说明。

x_i	0.40	0.55	0.65	0.80	0.90	1.05
$f(x_i)$	0.41075	0.57815	0.69675	0.88811	1.02652	1.25382

答：使用分段线性插值和三次样条插值的方式进行计算。

分段线性插值：将相邻两个点用直线连起来，得到的图形就是插值函数的图形。

三次样条插值：三次样条插值就是将原始长序列分割成若干段构造多个三次函数（每段一个），使得分段的衔接处具有二阶导数连续的性质。

使用 Python 的 Scipy 包进行计算，得到如下结果

线性插值	三次样条插值

0.632706	0.6319166104035825

```
D:\Python3.10\python.exe D:\PycharmProject\EngineeringMath\2.2.py
线性插值函数在0.596的值为0.632706
三次样条插值函数在0.596的值为0.6319166104035825

Process finished with exit code 0
```

图 2 插值结果

代码：(省略了 import)

```
x = [0.40, 0.55, 0.65, 0.80, 0.90, 1.05]
y = [0.41075, 0.57815, 0.69675, 0.88811, 1.02652, 1.25382]

#线性插值
f_linear = interp.interp1d(x, y, 'linear')

#三次样条
f_cubic = interp.interp1d(x, y, 'cubic')

xx = np.linspace(0.40, 1.05, 20)
yy1 = f_linear(xx)
yy2 = f_cubic(xx)

plt.plot(x, y, 'ro-')
plt.plot(xx, yy1, 'bo-')
plt.plot(xx, yy2, 'yo-')
plt.xlabel('x')
plt.ylabel('y')
plt.legend(['original', 'linear', 'cubic'])
plt.show()

#求值
print(f'线性插值函数在 0.596 的值为{f_linear(0.596)}')
print(f'三次样条插值函数在 0.596 的值为{f_cubic(0.596)}')
```

3. 对某种现象的演变过程进行多次观测，得到的数据如下：

第 x 次	观测值 y	第 x 次	观测值 y	第 x 次	观测值 y
1	5.4167	11	6.5859	21	9.2420
2	5.5196	12	6.7297	22	9.3717
3	5.7428	13	6.9172	23	9.4974

4	5.8796	14	7.2538	25	9.7542
5	6.1465	15	7.4542	26	9.8705
6	6.2828	16	7.7368	28	10.1541
7	6.4653	17	7.8534	30	10.2495
8	6.5994	18	8.2992	31	10.3475
9	6.7209	19	8.7177	32	
10	6.6207	20	9.0859		

分别按下述两种方案进行拟合，求解拟合函数，给出第 27 次观测值并预测第 32 次观测值，需要写出拟合求解过程、主要代码及运行结果，并对这两种方案进行对比分析.

方案 1：拟合函数为三次多项式 $\boldsymbol{f(x)=a_0+a_1x+a_2x^2+a_3x^3}$；

方案 2：拟合函数为非线性函数 $f(x)=a+b\sin\dfrac{\pi x}{10}$。

答：使用 MSE 函数作为评价的指标，

$$\mathrm{MSE}(y,y')=\frac{\sum_{i=1}^{n}(y_i-y_i')^2}{n}$$

使用 scipy.optimize 包进行拟合，得到的拟合函数分别为

$$f(x)=5.688912-0.02188x+0.014675x^2+0.000292x^3$$

$$f(x)=7.74611386\quad -0.09558488\quad \sin\frac{\pi x}{10}$$

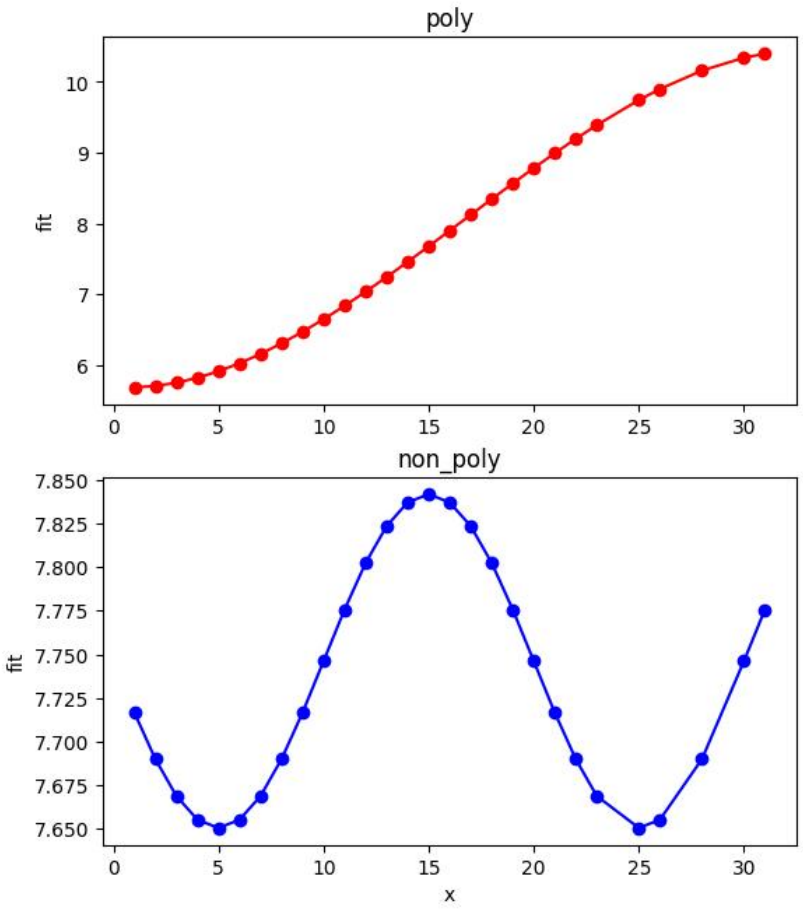


图 3 拟合对比

```
D:\Python3.10\python.exe D:\PycharmProject\EngineeringMath\2.3.py
线性拟合函数的系数为[-2.92558609e-04  1.46755443e-02 -2.18873596e-02  5.68891128e+00]
线性拟合函数在27和32的值分别为10.037993255521364和10.429712624321311
非线性拟合函数的系数为[ 7.74611386 -0.09558488]
非线性拟合函数在27和32的值分别为7.668784063643017和7.8022972397474515
多项式拟合的MSE为[0.04151962]
非多项式拟合的MSE为[2.48237676]

Process finished with exit code 0
```

图 4 拟合结果和误差

多项式拟合的 MSE 为 0.04，非多项式拟合的 MSE 为 2.4823.显然多项式拟合的误差更小，拟合效果更好。

代码：

```
def mse(y, y_fit):
    y = y.reshape([len(y), 1])
    t = (np.power(y - y_fit, 2))
    mse = 1 / len(y) * sum(t)
    return mse
```

```
6.
7.
8.  def func1(x, a, b):
9.      return a + b * np.sin(np.pi * x / 10)
10.
11.
12. def getPolyFitValue(p, x):
13.     degree = len(p)
14.     value = 0
15.     for i in range(degree):
16.         value = value + p[i] * np.power(x, degree - 1 - i)
17.     return value
18. def plot(p, popt, x, y):
19.     poly = np.zeros([len(x), 1])
20.     nonpoly = np.zeros([len(x), 1])
21.     for i in range(len(x)):
22.         poly[i] = getPolyFitValue(p, x[i])
23.         nonpoly[i] = func1(x[i], popt[0], popt[1])
24.     mse_poly = mse(y, poly)
25.     mse_nonpoly = mse(y, nonpoly)
26.     print(f"多项式拟合的 MSE 为{mse_poly}")
27.     print(f"非多项式拟合的 MSE 为{mse_nonpoly}")
28.     plt.subplot(2, 1, 1)
29.     plt.plot(x, poly, 'ro-')
30.     plt.ylabel('fit')
31.     plt.title('poly')
32.     plt.subplot(2, 1, 2)
33.     plt.plot(x, nonpoly, 'bo-')
34.     plt.xlabel('x')
35.     plt.ylabel('fit')
36.     plt.title('non_poly')
37.     plt.show()
38. y0 = [5.4167, 5.5196, 5.7428, 5.8796, 6.1465, 6.2828, 6.4653, 6.5994, 6.7209, 6.6207]
39. y1 = [6.5859, 6.7297, 6.9172, 7.2538, 7.4542, 7.7368, 7.8534, 8.2992, 8.7177, 9.0859]
40. y2 = [9.2420, 9.3717, 9.4974, 9.7542, 9.8705, 10.1541, 10.2495, 10.3475]
41. y0 = np.concatenate([y0, y1])
42. y = np.concatenate([y0, y2])
43. x0 = np.linspace(1, 23, 23)
44. x1 = [25, 26, 28, 30, 31]
45. x = np.concatenate([x0, x1])
46. p = np.polyfit(x, y, 3)
47. print(f'线性拟合函数的系数为{p}')
48. print(f'线性拟合函数在 27 和 32 的值分别为 {getPolyFitValue(p, 27)} 和
    {getPolyFitValue(p, 32)}')
```

```
popt, pcov = opt.curve_fit(func1, x, y)
print(f'非线性拟合函数的系数为{popt}')
print(f'非线性拟合函数在 27 和 32 的值分别为{func1(27, popt[0], popt[1])}和{func1(32, popt[0], popt[1])}')
plot(p, popt, x, y)
```

4. 矩阵三角分解的主要思想是什么？有哪几种常用的三角分解法？请选择你熟悉的一种方法求解下列方程组，对分解过程进行描述，给出相应的分解矩阵和计算结果。

$$\begin{cases} 5x_1 + x_2 - x_3 - 2x_4 = -2 \\ 2x_1 + 8x_2 + x_3 + 3x_4 = -6 \\ x_1 - 2x_2 - 4x_3 - x_4 = 6 \\ -x_1 + 3x_2 + 2x_3 + 7x_4 = 12 \end{cases}$$

一个矩阵如果其 n-1 阶顺序主子式都不为 0，则该矩阵可以进行分解。具体来说，三角分解就是将一个矩阵分解为一个下三角矩阵和一个上三角矩阵之积。常用的三角分解法有 LU 和 QR 分解。

使用杜利特尔分解法

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ l_{21} & 1 & 0 & 0 \\ l_{31} & l_{32} & 1 & 0 \\ l_{41} & l_{42} & l_{43} & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{pmatrix}$$

$$= \begin{pmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ l_{21}u_{11} & l_{21}u_{12} + u_{22} & l_{21}u_{13} + u_{23} & l_{21}u_{14} + u_{24} \\ l_{31}u_{11} & l_{31}u_{12} + l_{32}u_{22} & l_{31}u_{13} + l_{32}u_{23} + u_{33} & l_{31}u_{14} + l_{32}u_{24} + u_{34} \\ l_{41}u_{11} & l_{41}u_{12} + l_{42}u_{22} & l_{41}u_{13} + l_{42}u_{23} + l_{43}u_{33} & l_{41}u_{14} + l_{42}u_{24} + l_{43}u_{34} + u_{44} \end{pmatrix}$$

首先求出 u11=5, u12=1, u13=-1, u14=-2, 然后带入第二行，先求出$l_{21} = a_{21}/u_{11} = 0.4$,然后再从左到右求出$u_{22}, u_{23}, u_{24}$，以此类推求出所有未知数。

得到 LU 分解后，

$$x = U^{-1}(L^{-1}b)$$

通过 Python 计算，得到

```
D:\Python3.10\python.exe D:\PycharmProject\EngineeringMath\2.4.py
L为[[ 1.          0.          0.          0.        ]
 [ 0.4         1.          0.          0.        ]
 [ 0.2        -0.28947368  1.          0.        ]
 [-0.2         0.42105263 -0.35658915  1.        ]]
U为[[ 5.          1.         -1.         -2.        ]
 [ 0.          7.6         1.4         3.8       ]
 [ 0.          0.         -3.39473684  0.5       ]
 [ 0.          0.          0.          5.17829457]]
解得方程为[ 1. -2. -1.  3.]
```

图 5 LU 分解和解方程

代码：

```
import numpy as np
from scipy.linalg import lu
from scipy.linalg import inv
```

```
A = np.mat([[5, 1, -1, -2], [2, 8, 1, 3], [1, -2, -4, -1], [-1, 3, 2, 7]])
b = np.array([-2, -6, 6, 12])

p, L, U = lu(A)
print(f'L 为{L}')
print(f'U 为{U}')

x = np.matmul(inv(U), np.matmul(inv(L), b))
print(f'解得方程为{x}')
```

5. 构造求解方程 $e^x+10x-2=0$ 的根的迭代格式 $x_{n+1}=\varphi(x_n),n=0,1,2,\cdots$，讨论其收敛性，并将根求出来，满足 $|x_{n+1}-x_n|<10^{-4}$。给出计算过程和计算结果。

答：首先构造迭代式 x = ln(2-10x)，其中含有 ln，可能会对迭代产生影响。令 x0=0， 则 x1 = ln2 越等于 0.6, x2 = -inf.该迭代式不收敛。

再构造 $x=\frac{2-e^x}{10}$，用 Python 迭代 20 次观察其收敛性，得到

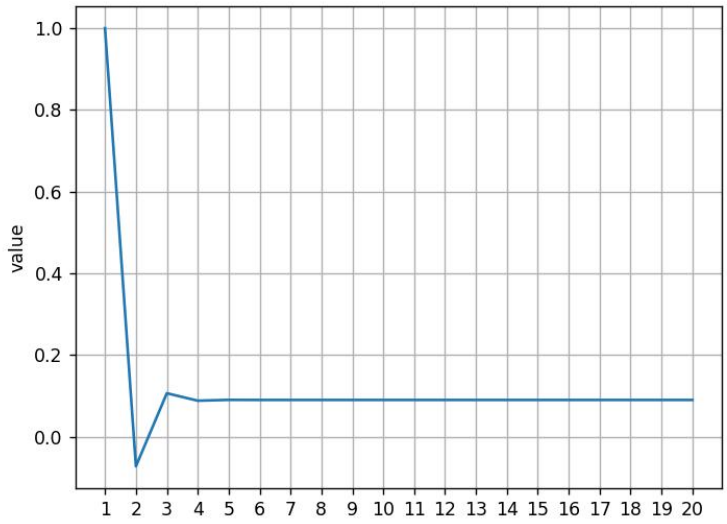


图 6 迭代稳定性

可以观察到 20 次迭代后数据基本收敛，以$x_{n+1}-x_n<10^{-4}$作为迭代终止条件，得到

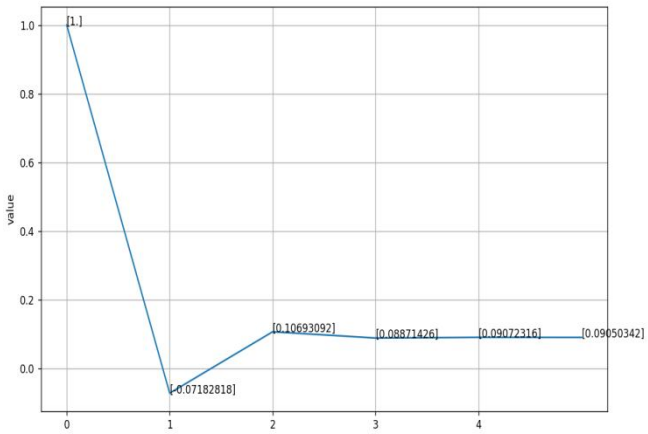


图 7 五次迭代

```
迭代结果
[[ 1.        ]
 [-0.07182818]
 [ 0.10693092]
 [ 0.08871426]
 [ 0.09072316]
 [ 0.09050342]]
```

代码：

```
def iteration(x_pre):
    x_next = (2 - np.exp(x_pre))/10
    return x_next
x = np.ones([20, 1])
xx = np.linspace(1, 20, 20)
for i in range(1, 20):
    x[i] = iteration(x[i - 1])
    if np.abs(x[i] - x[i - 1]) < 10**(-4):
        break
x = x[0:i]
plt.plot(range(0, i), x)
for i in range(i):
    plt.text(xx[i]-1, x[i], str(x[i]))
plt.xticks(range(0, i))
plt.ylabel('n_iter')
plt.ylabel('value')
plt.grid()
plt.show()
print('迭代结果')
print(x)
```

6. 请用梯度法和拟牛顿法求解下列优化问题：

$$\min f(X)=\frac{3}{2}x_1^2+\frac{1}{2}x_2^2-x_1x_2-2x_1$$

取初始点 $X^{(1)}=[0,0]^T,\varepsilon=10^{-2}$，并对求解过程进行说明，给出前两次迭代计算结果。

梯度法：先求出函数的梯度

$$\mathrm{d}=\begin{pmatrix}3\mathrm{x}_1-\mathrm{x}_2-2\\ \mathrm{x}_2-\mathrm{x}_1\end{pmatrix}$$

，然后带入$\mathrm{x}_1=-2$ 和$\mathrm{x}_2=0$ 得到

$$\mathrm{d}=\begin{pmatrix}-2\\ 0\end{pmatrix}$$

再得到

$$x_{next}=\begin{pmatrix}x_1+\lambda d_1\\ x_2+\lambda d_2\end{pmatrix}$$

将x_{next}带入 f(x)中得

$$\mathrm{f(x)}=6\lambda^2+4\lambda$$

求解出使得 f(x)最小的λ，然后带入x_{next}得到下一次的 x.

拟牛顿法(DFP)

(1) 给定初始点$\mathrm{x}^{(0)}$，允许误差ϵ，令$D_0=I_n$(n 是 x 的维数)，t=0

(2) 计算搜索方向$d^{(t)}=-D_t^{-1}g_t$

(3) 从点x^{t}出发，沿着d^{t}做一维搜索，获得最优步长并更新参数：

$$\lambda_t=argmin_\lambda f(x^{(t)}+\lambda d^{(t)})$$

$$\mathrm{x}^{(\mathrm{t}+1)}=x^{(t)}+\lambda d^{(t)}$$

(4) 判断精度，若$|g_{t+1}|<\epsilon$则停止迭代，否则转(5)

(5) 计算$\Delta\mathrm{g}=\mathrm{g}_{\mathrm{t}+1}-\mathrm{g}_{\mathrm{t}}$，$\Delta\mathrm{x}=\mathrm{x}^{(\mathrm{t}+1)}-\mathrm{x}^{\mathrm{t}}$, 更新 D

$$\mathrm{D}_{\mathrm{t}+1}=\mathrm{D}_{\mathrm{t}}+\frac{\Delta\mathrm{x}\Delta\mathrm{x}^{\mathrm{T}}}{\Delta\mathrm{g}^{\mathrm{T}}\Delta\mathrm{x}}-\frac{\mathrm{D}_{\mathrm{t}}\Delta\mathrm{g}\Delta\mathrm{g}^{\mathrm{T}}\mathrm{D}_{\mathrm{t}}}{\Delta\mathrm{g}^{\mathrm{T}}\mathrm{D}_{\mathrm{t}}\Delta\mathrm{g}}$$

(6) t=t+1，转(2)

梯度法：

迭代	g	lambda	X_new
1	[-2, 0]	1/3	[2/3, 0]
2	[0, -2/3]	1	[2/3, 2/3]

```
D:\Python3.10\python.exe D:\PycharmProject\EngineeringMath\2.6.py
     fun: -0.9999999999999997
     jac: array([-2.23517418e-08,  1.49011612e-08])
 message: 'Optimization terminated successfully'
    nfev: 13
     nit: 4
    njev: 4
  status: 0
 success: True
       x: array([0.99999999, 0.99999998])
True
[0.666666666666667 0]
[0.666666666666667 0.666666666666667]
[0.888888888888889 0.666666666666667]
[0.888888888888889 0.888888888888889]
[0.962962962962963 0.888888888888889]
[0.962962962962963 0.962962962962963]
[0.987654320987654 0.962962962962963]
[0.987654320987654 0.987654320987654]
[0.995884773662552 0.987654320987654]
[0.995884773662551 0.995884773662550]
[0.998628257887517 0.995884773662551]
[0.998628257887517 0.998628257887516]
[0.999542752629172 0.998628257887516]
```

图 8 最速下降法的迭代

DFP 法：

迭代	H	G	d	a	X_new	y	s
1	E	[[3, -1], [-1, 1]]	[2, 0]	1/3	[2/3, 0]	[2, -2/3]	[2/3, 0]
2	[[0.4333, 0.3],[0.3, 0.9]]	[[3, -1], [-1, 1]]	[0.2, 0.6]	5/3	[1, 1]	1/3	[0, 2/3]

```
D:\Python3.10\python.exe D:\PycharmProject\EngineeringMath\2.6.1.py
第 1 次结果
[[0.66666667]
 [0.        ]]
[[0]
 [0]]
第 2 次结果
[[1.]
 [1.]]
[[0.66666667]
 [0.        ]]
[1, 1]
```

图 9 DFP 算法迭代

计算得到当 x=[1, 1]时取得最小值为-1.

代码:

最速下降法：

```python
def fun():
    v = lambda x: 3/2*x[0]**2 + 1/2*x[1]**2 - x[0]*x[1] - 2*x[0]
    return v
x0 = np.asarray((0, 0))  # 初始猜测值
res = minimize(fun(), x0, method='SLSQP')
print(res)
print(res.success)
```

```
def sinvar(fun):
    s_p = solve(diff(fun))  # stationary point
    return s_p
def value_enter(fun, x, i):
    value = fun[i].subs(x1, x[0]).subs(x2, x[1])
    return value
def grandient_l2(grand, x_now):
    grand_l2 = sqrt(pow(value_enter(grand, x_now, 0), 2) + pow(value_enter(grand, x_now, 1), 2))
    return grand_l2
def msd(x_init, obj):
    i = 1
    grandient_obj = np.array([diff(obj, x1), diff(obj, x2)])
    error = grandient_l2(grandient_obj, x_init)
    plt.plot(x_init[0], x_init[1], 'ro')
    while (error > 0.001):
        if i == 1:
            grandient_obj_x = np.array([value_enter(grandient_obj, x_init, 0), value_enter(grandient_obj, x_init, 1)])
            t = symbols('t')
            x_eta = x_init - t * grandient_obj_x
            eta = sinvar(obj.subs(x1, x_eta[0]).subs(x2, x_eta[1]))
            x_new = x_init - eta * grandient_obj_x
            print(x_new)
            i = i + 1
            error = grandient_l2(grandient_obj, x_new)
            plt.plot(x_new[0], x_new[1], 'ro')
        else:
            grandient_obj_x = np.array([value_enter(grandient_obj, x_new, 0), value_enter(grandient_obj, x_new, 1)])
            t = symbols('t')
            x_eta = x_new - t * grandient_obj_x
            eta = sinvar(obj.subs(x1, x_eta[0]).subs(x2, x_eta[1]))
            x_new = x_new - eta * grandient_obj_x
            print(x_new)
            i = i + 1
            error = grandient_l2(grandient_obj, x_new)
            plt.plot(x_new[0], x_new[1], 'ro')
    plt.show()
x_0 = np.array([0, 0], dtype=float)
x1 = symbols("x1")
x2 = symbols("x2")
result = msd(x_0, 3/2*x1**2 + 1/2*x2**2 - x1*x2 - 2*x1)
```

DFP 法：

```
import numpy as np
import sympy as sp


def jacobian(f, x):  # 雅可比矩阵，求一阶导数
    a, b = np.shape(x)  # 判断变量维度
    x1, x2 = sp.symbols('x1 x2')  # 定义变量，如果多元的定义多元的
    x3 = [x1, x2]  # 将 1 变量放入列表中，方便查找和循环。有几个变量放几个
    df = np.array([[0.00000], [0.00000]])  # 定义一个空矩阵，将雅可比矩阵的值放入，保留多少位小数，小数点后面就有几个 0。n 元变量就加 n 个[]
    for i in range(a):  # 循环求值

        df[i, 0] = sp.diff(f, x3[i]).subs({x1: x[0][0], x2: x[1][0]})  # 求导和求值,n 元的在 subs 后面补充

    return df


def hesse(f, x):  # hesse 矩阵
    a, b = np.shape(x)
    x1, x2 = sp.symbols('x1 x2')
    x3 = [x1, x2]
    G = np.zeros((a, a))
    for i in range(a):
        for j in range(a):
            G[i, j] = sp.diff(f, x3[i], x3[j]).subs({x1: x[0][0], x2: x[1][0]})  # n 元的在 subs 后面补充

    return G


def dfp_newton(f, x, iters):
    """
    实现 DFP 拟牛顿算法
    :param f: 原函数
    :param x: 初始值
    :param iters: 遍历的最大迭代次数
    :return: 最终更新完毕的 x 值
    """
    a = 1  # 定义初始步长

    H = np.eye(2)  # 初始化正定矩阵
    G = hesse(f, x)  # 初始化 Hesse 矩阵
```

```
41.
42.     epsilon = 1e-3  # 一阶导 g 的第二范式的最小值（阈值）
43.     for i in range(1, iters):
44.         g = jacobian(f, x)
45.
46.         if np.linalg.norm(g) < epsilon:
47.             xbest = []
48.             for a in x:
49.                 xbest.append(round(a[0]))  # 将结果从矩阵中输出放到列表中并四舍五入
50.             break
51.         # 下面的迭代公式
52.         d = -np.dot(H, g)
53.
54.         a = -(np.dot(g.T, d) / np.dot(d.T, np.dot(G, d)))
55.
56.         # 更新 x 值
57.         x_new = x + a * d
58.         print("第 %d 次结果" % i)
59.         print(x_new)
60.         g_new = jacobian(f, x_new)
61.         y = g_new - g
62.
63.         s = x_new - x
64.         # 更新 H
65.         H = H + np.dot(s, s.T) / np.dot(s.T, y) - np.dot(H, np.dot(y, np.dot(y.T, H)))
    / np.dot(y.T, np.dot(H, y))
66.         # 更新 G
67.         G = hesse(f, x_new)
68.         print(x)
69.         x = x_new
70.
71.     return xbest
72.
73.
74. x1, x2 = sp.symbols('x1 x2')  # 例子
75. x = np.array([[0], [0]])
76. f = 3/2*x1**2 + 1/2*x2**2 - x1*x2 - 2*x1
77. print(dfp_newton(f, x, 20))
```

7. 已知某公园地形图和界牌/界桩点坐标点如下图所示，请计算该公园边界线长度和近似面积，给出求解步骤说明。

界牌/界桩点坐标表

界桩(牌)编号	北　纬	东　经	高程(m)
J1	25° 31′ 06.72″	103° 45′ 57.60″	1870
J2	25° 31′ 07.42″	103° 45′ 57.22″	1871
J3	25° 31′ 09.30″	103° 46′ 00.80″	1871
J4	25° 31′ 12.96″	103° 46′ 02.84″	1871
J5	25° 31′ 14.89″	103° 46′ 03.52″	1871
J6	25° 31′ 19.96″	103° 46′ 05.18″	1871
J7	25° 31′ 20.10″	103° 46′ 05.60″	1871
J8	25° 31′ 21.32″	103° 46′ 06.22″	1871
J9	25° 31′ 21.88″	103° 46′ 05.86″	1817
J10	25° 31′ 21.48″	103° 46′ 07.59″	1873
J11	25° 31′ 31.08″	103° 46′ 07.82″	1874
J12	25° 31′ 31.10″	103° 46′ 08.69″	1874
J13	25° 31′ 36.46″	103° 46′ 09.20″	1878
J14	25° 31′ 35.85″	103° 46′ 16.55″	1877
J15	25° 31′ 34.26″	103° 46′ 16.14″	1877
J16	25° 31′ 33.90″	103° 46′ 18.14″	1877
J17	25° 31′ 35.52″	103° 46′ 18.50″	1877
J18	25° 31′ 34.20″	103° 46′ 23.29″	1879
J19	25° 31′ 28.63″	103° 46′ 18.91″	1879
J20	25° 31′ 29.11″	103° 46′ 17.12″	1879
J21	25° 31′ 28.43″	103° 46′ 15.58″	1887
J22	25° 31′ 28.64″	103° 46′ 13.58″	1887
J23	25° 31′ 29.53″	103° 46′ 12.99″	1884
J24	25° 31′ 24.18″	103° 46′ 11.08″	1878
J25	25° 31′ 09.94″	103° 46′ 06.08″	1871
J26	25° 31′ 08.44″	103° 46′ 04.62″	1871
J27	25° 31′ 05.76″	103° 46′ 01.62″	1870

注：界牌　界桩

(1)计算边界线长度。将坐标数据转为小数形式(度为整数，其余部分为小数，例如 25° 31′ 06.72″ 转为 25.310672)，然后将经纬度坐标数据转为以为 J1 为原点的 XY 坐标数据。最后求解每两个点之间的欧氏距离之和，得到边界线长度。

(2)因为图形形状不规则，所以使用投点的方式计算面积。首先求出能包含这个图形的最小矩形，求出该矩形的面积 S，然后在这个矩形内随机投 100000 颗针，统计在该图形内的针的数量 n，则该图形的面积为 $n\frac{n}{100000}S$。

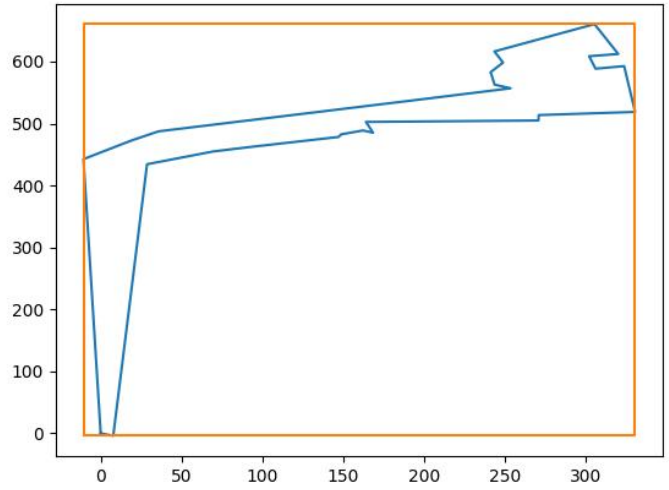


图 10 投点框

如上图所示，在黄色矩形框内均匀投 100000 个点，然后统计落在区域内的点的个数即可计算面积。

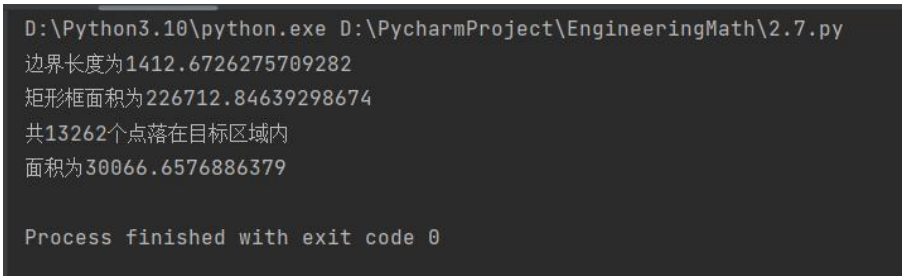


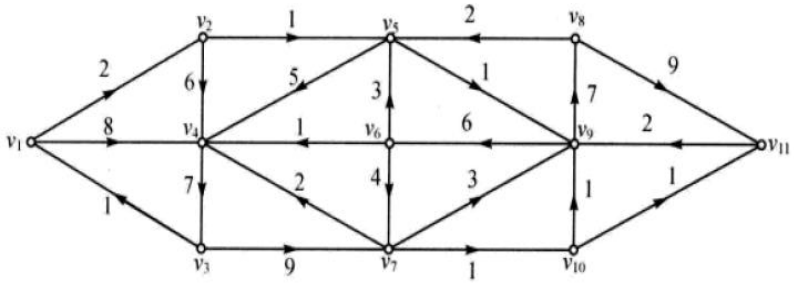
图 11 求解面积程序运行结果

```
import shapely.geometry

global CONSTANTS_RADIUS_OF_EARTH
CONSTANTS_RADIUS_OF_EARTH = 6371000

x = [25.310672, 25.310742, 25.310930, 25.311296, 25.311489, 25.311996, 25.312010, 25.312132, 25.312188, 25.312148,
     25.313108, 25.313110, 25.313646,
     25.313585, 25.313426, 25.313390, 25.313552, 25.313420, 25.312863, 25.312911, 25.312842, 25.312864, 25.312953,
     25.312418, 25.310994,
     25.310844, 25.310576]

y = [103.455760, 103.455722, 103.460080, 103.460284, 103.460352, 103.460518, 103.460560, 103.460622, 103.460586,
     103.460759,
     103.460782, 103.460869, 103.460920, 103.461655, 103.461614, 103.461814, 103.461850, 103.462329, 103.461891,
     103.461712,
     103.461558, 103.461358, 103.461299, 103.461108, 103.460608, 103.460462, 103.460162]


def GPStoXY(lat, lon, ref_lat, ref_lon):
    # input GPS and Reference GPS in degrees
    # output XY in meters (m) X:North Y:East
    lat_rad = math.radians(lat)
    lon_rad = math.radians(lon)
    ref_lat_rad = math.radians(ref_lat)
    ref_lon_rad = math.radians(ref_lon)
    sin_lat = math.sin(lat_rad)
    cos_lat = math.cos(lat_rad)
    ref_sin_lat = math.sin(ref_lat_rad)
    ref_cos_lat = math.cos(ref_lat_rad)
    cos_d_lon = math.cos(lon_rad - ref_lon_rad)
```

```
        arg = np.clip(ref_sin_lat * sin_lat + ref_cos_lat * cos_lat * cos_d_lon, -1.0, 1.0)

        c = math.acos(arg)
        k = 1.0
        if abs(c) > 0:
            k = (c / math.sin(c))
        x = float(k * (ref_cos_lat * sin_lat - ref_sin_lat * cos_lat * cos_d_lon) * CONSTANTS_RADIUS_OF_EARTH)
        y = float(k * cos_lat * math.sin(lon_rad - ref_lon_rad) * CONSTANTS_RADIUS_OF_EARTH)

        return x, y


    ref_x = x[0]
    ref_y = y[0]
    for i in range(len(x)):
        x[i], y[i] = GPStoXY(x[i], y[i], ref_x, ref_y)

    border = 0
    for i in range(len(x) - 1):
        border = border + np.sqrt((x[i + 1] - x[i]) ** 2 + (y[i + 1] - y[i]) ** 2)
    print(f'边界长度为{border}')

    x_max = max(x)
    y_max = max(y)
    x_min = min(x)
    y_min = min(y)

    x = np.array(x).reshape(27, 1)
    y = np.array(y).reshape(27, 1)
    point = np.concatenate([x, y], axis=1)
    # print(point)

    poly = shapely.geometry.Polygon(point)
    x, y = poly.exterior.xy
    plt.plot(x, y)
    ref = shapely.geometry.Polygon([[x_min, y_min], [x_min, y_max], [x_max, y_max], [x_max, y_min]])
    x1, y1 = ref.exterior.xy
    plt.plot(x1, y1)
    plt.show()


    def generate_point(n, x_max, y_max, x_min, y_min):
```

```
    p = np.zeros([n, 2])
    p[:, 0] = np.random.uniform(x_min, x_max, [n, 1]).reshape([n, ])
    p[:, 1] = np.random.uniform(y_min, y_max, [n, 1]).reshape([n, ])
    return p


n = 100000
sum = 0
Points = generate_point(n, x_max, y_max, x_min, y_min)
for i in range(n):
    x = Points[i, 0]
    y = Points[i, 1]
    p = shapely.geometry.Point(x, y)
    if poly.contains(p):
        sum = sum + 1
S = (x_max-x_min) * (y_max-y_min)
print(f'矩形框面积为{S}')
print(f'共{sum}个点落在目标区域内')
print(f'面积为{sum/n*S}')
```

8. 求下图所示网络从 v_1 点到 v_{11} 点的最短路线和距离，给出求解过程的计算描述。



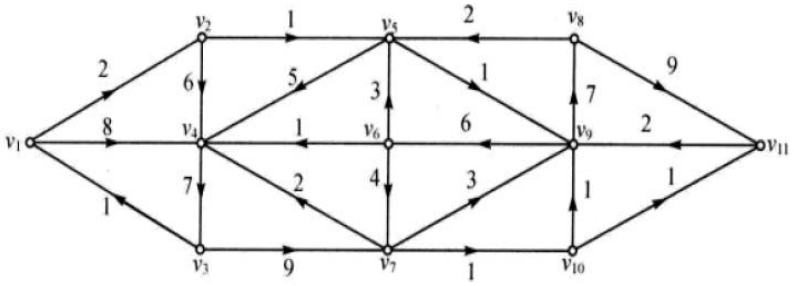
使用 Dijkstra 算法。

S	U	V1 点到各点距离
{v1}	{v2, v3, v4, v5, v6, v7, v8, v9, v10, v11}	v2(2), v3(*), v4(8), v5(*)
{v1, v2}	{v3, v4, v5, v6, v7, v8, v9, v10, v11}	v2(2), v3(*), v4(8), v5(3)
{v1, v2, v5}	{v3, v4, v6, v7, v8, v9, v10, v11}	v4(8), v9(4)
{v1, v2, v5, v9}	{v3, v4, v6, v7, v8, v10, v11}	v6(10), v8(11)
{v1, v2, v5, v9, v6}	{v3, v4, v7, v8, v10, v11}	v7(14)
{v1, v2, v5, v9, v6,	{v3, v4, v8, v10, v11}	v10(15)

v7}

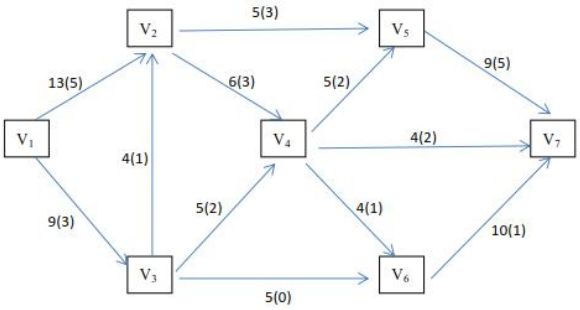
{v1, v2, v5, v9, v6, v7, v10}	{v3, v4, v8, v11}	v11(16)
{v1, v2, v5, v9, v6, v7, v10, v11}	{v3, v4, v8}	

得到最短路径为 16



通过 v1-v2-v5-v9-v6-v7-v10-v11

9. 求下图所示网络的最大流和最小割，给出求解过程的计算描述。



最大流：

(1) 初始化最大流 Flow=0；

(2) 在图中找一条通路，如果通路不存在，则结束；

(3) Flow = Flow+通路流量；

(4) 对通路上所有边的容量进行更新 Cij = Cij-通路流量；

(5) 跳转 Step2

```
D:\Python3.10\python.exe D:\PycharmProject\EngineeringMath\2.9.py
{'v1': {'v2': 5, 'v3': 6}, 'v2': {'v5': 2, 'v4': 3}, 'v3': {'v2': 0, 'v4': 1, 'v6': 5}, 'v5': {'v7': 2}, 'v4': {'v5': 0, 'v7': 2, 'v6': 2}, 'v6': {'v7': 7}, 'v7': {}}
(11, ({'v2', 'v1'}, {'v3', 'v5', 'v6', 'v4', 'v7'}))
83
```

通过 networkx 包求解得到的最大流为 83.

最小割：寻找最大流的过程中容量达到上限的边即为最小割。

10. 有一居民小区，其自来水是由一个圆柱形水塔提供，水塔高 12.2m，塔直径为 17.4m。水塔是由水泵根据水塔中的水位自动加水。按照设计，当水塔中的水位降至最低水位约 8.2m 时，水泵自动启动加水；当水位升高到最高水位约 10.8m 时，水泵停止工作。表 2 给出 28 个时刻的数据，由于水泵正在向水塔供水，有 4 个时刻无法测到水位（表中用“--”表示）。

时刻(t)/h	0	0.92	1.84	2.95	3.87	4.98	5.90
水位/m	9.68	9.48	9.31	9.13	8.98	8.81	8.69
时刻(t)/h	7.01	7.93	8.97	9.98	10.92	10.95	12.03
水位/m	8.52	8.39	8.22	--	--	10.82	10.5
时刻(t)/h	12.95	13.88	14.98	15.9	16.83	17.93	19.04
水位/m	10.21	9.94	9.65	9.41	9.18	8.92	8.66
时刻(t)/h	19.96	20.84	22.01	22.96	23.88	24.99	25.91
水位/m	8.43	8.22	--	--	10.59	10.35	10.18

请完成以下问题：

（1）绘制水位数据散点图；

（2）以下是一个计算居民用水速度和日总用水量的步骤，请按要求补充中间结果。

第一步：水塔中水的体积计算。要计算水的流量，首先需要计算出水塔中水的体积，体积计算公式及代码如下：

```
def get_volume(h):
    r = 17.4 / 2
    return np.pi * r ** 2 * h
```

第二步：水塔中水流速度的估计。居民用水速度就是水塔中的水流速度，水流速度就是水塔中水的体积对时间的导数。但由于没有每一时刻水体积的具体数学表达式，只能用差商近似导数。

由于在两个时段，水泵向水塔供水，无法确定水位的高度，因此在计算水塔中水流速度时要分三段计算，即：第一段从 0-8.97h，第二段从 10.95-20.94h，第三段从 23.88-25.91h。

上面计算仅给出流速的离散值，如果需要得到流速的连续曲线，需要按下列方式分三段进行三次多项式拟合：用前 6 个数据点，即在时间区间[0,4.98]上拟合三次多项式；用第 6 个数据点到第 10 个数据点，即在时间区间[4.98,12.03]上拟合第二个三次多项式；用第 10 个数据点到第 20 个数据点，即在时间区间[12.03,25.91]上拟合第三个三次多项式。计算过程如下：

```
t = [0, 0.92, 1.84, 2.95, 3.87, 4.98, 5.90, 7.01, 7.92,
     8.97, 9.98, 10.92, 10.95, 12.03, 12.95, 13.88,
     14.98, 15.9, 16.83, 17.93, 19.04, 19.96,
     20.84, 22.01, 22.96, 23.88, 24.99, 25.91]

h = [9.68, 9.48, 9.31, 9.13, 8.98, 8.81, 8.69,
     8.52, 8.39, 8.22, 0, 0, 10.82, 10.5, 10.21,
     9.94, 9.65, 9.41, 9.18, 8.92, 8.66, 8.43,
     8.22, 0, 0, 10.59, 10.35, 10.18]

diff_h_0 = np.zeros([len(h), 1])
for i in range(1, len(h)):
    diff_h_0[i - 1] = (h[i] - h[i - 1]) / (t[i] - t[i - 1])

diff_h_0[9:12] = 0
diff_h_0[22:25] = 0

print(diff_h_0)
```

```
plt.plot(range(0, len(diff_h_0)), diff_h_0)
plt.show()

y1 = diff_h_0[:9]
y2 = diff_h_0[12:22]
y3 = diff_h_0[25:27]

p0 = np.polyfit(t[:9], y1, 3)
p1 = np.polyfit(t[12:22], y2, 3)
p2 = np.polyfit(t[25:27], y3, 3)
```

得到的曲线方程为：

段	x^3	x^2	x	c
1	-1.79523438×10^{-3}	2.04849523×10^{-2}	-2.3222×10^{-1}	-2.3222
2	$-4.825800583\times10^{-3}$	1.05618461×10^{-1}	-9.1197×10^{-1}	1.1385
3	1.00367066×10^{-1}	-4.58350998×10^{-3}	68.8713	-3.3063

第三步：日总用水量的计算。日用水量是对水流速度作积分，其积分区间为$[0,24]$,利用数值积分方法计算，代码如下：

```
n = 100
x_0 = np.linspace(0, 8.97, n)
y_fit_0 = np.zeros([n, ])
for i in range(n):
    y_fit_0[i] = polynomial(p0, x_0[i])

x_1 = np.linspace(10.95, 20.84, n)
y_fit_1 = np.zeros([n, ])
for i in range(n):
    y_fit_1[i] = polynomial(p1, x_1[i])

x_2 = np.linspace(23.88, 24, 10)
y_fit_2 = np.zeros([10, ])
for i in range(10):
    y_fit_2[i] = polynomial(p2, x_2[i])

int_y0 = np.trapz(y_fit_0, x_0)
int_y1 = np.trapz(y_fit_1, x_1)
int_y2 = np.trapz(y_fit_2, x_2)

print(f'积分后得到{int_y0 + int_y1 + int_y2}')
print(f'日用水量为{get_volume(int_y0 + int_y1 + int_y2)}')
```

最终得到的日用水总量为：

图 12 日用水量

结果分析（不少于 150 字）：

我认为模型可以做如下改进：

1. 可以将拟合模型改为插值模型，增多数据点，更有利于后续的数值微分。
2. 数值微分的方式可以使用更先进的方式，而不是简单做一阶差分，可以采用二阶甚至更高阶的微商。
3. 数值积分也可以使用更好的方法，而不是简单的 trapz 算法。
4. 每组数据的前后两个数据可以分别采用后向和前向差商。
5. 实际上水位高度对水的流速也有一定的影响。

第三部分：综合应用题（请按要求完成以下问题）

1. 已知某物质有 8 个配送中心可以供货，有 15 个用户需要该物质。配送中心和用户之间单位物质的运费、用户物质需求量和配送中心的物质储备量见表 1。

表 1：数据

用户	单位物质运费								需求量
	1	2	3	4	5	6	7	8	
1	390.6	618.5	553	442	113.1	5.2	1217.7	1011	3000
2	370.8	636	440	401.8	25.6	113.1	1172.4	894.5	3100
3	876.3	1098.6	497.6	779.8	903	1003.3	907.2	40.1	2900
4	745.4	1037	305.9	725.7	445.7	531.4	1376.4	768.1	3100
5	144.5	354.5	624.7	238	290.7	269.4	993.2	974	3100
6	200.2	242	691.5	173.4	560	589.7	661.8	855.7	3400

7	235	205.5	801.5	326.6	477	433.6	966.4	1112	3500
8	517	541.5	338.4	219	249.5	335	937.3	701.8	3200
9	542	321	1104	576	896.8	878.4	728.3	1243	3000
10	665	827	427	523.2	725.2	813.8	692.2	284	3100
11	799	855.1	916.5	709.3	1057	1115.5	300	617	3300
12	852.2	798	1083	714.6	117.4	1216.8	40.8	898.2	3200
13	602	614	820	517.7	899.6	952.7	272.4	727	3300
14	903	1092.5	612.5	790	932.4	1034.9	777	152.3	2900
15	600.7	710	522	448	726.6	811.8	563	426.8	3100
储备量	18600	19600	17100	18900	17000	19100	20500	17200	

用 $i=1,2,\cdots,15$ 表示用户编号，$j=1,2,\cdots,8$ 表示配送中心编号，a_i 表示第 i 个用户的需求量，b_j 表示第 j 个配送中心的物质储备量，c_{ij} 表示第 j 个配送中心到第 i 个用户之间的单位物质运费，x_{ij} 表示第 j 个配送中心对第 i 个用户的物质配送量。现有以下两种调运方案：

（1）配送量无约束时，最小费用调运方案模型可表示为：

$$\min z=\sum_{i=1}^{15}\sum_{j=1}^{8}c_{ij}x_{ij}$$

$$\text{s.t.}\begin{cases}\sum\limits_{i=1}^{15}x_{ij}\le b_j, & j=1,2,\cdots,8\\ \sum\limits_{j=1}^{8}x_{ij}=a_i, & i=1,2,\cdots,15\\ x_{ij}\ge 0, & i=1,2,\cdots 15;\quad j=1,2,\cdots,8\end{cases}$$

（2）若每个配送中心，可以对某个用户配送物质，也可以不对某个用户配送物质；若配送物质，配送量要大于 1000 且小于等于 2000，此时的最小费用调运方案模型为：

$$\min z=\sum_{i=1}^{15}\sum_{j=1}^{8}c_{ij}x_{ij}$$

$$\text{s.t.}\begin{cases}\sum\limits_{i=1}^{15}x_{ij}\le b_j, & j=1,2,\cdots,8\\ \sum\limits_{j=1}^{8}x_{ij}=a_i, & i=1,2,\cdots,15\\ 1000\le x_{ij}\le 2000\text{或}x_{ij}=0 & i=1,2,\cdots 15;\quad j=1,2,\cdots,8\end{cases}$$

请分别对这两种方案进行求解，给出计算过程描述，将程序代码及计算结果截图附在题后。

答：使用 cvpyx 包进行求解第一问得到如下结果

```
(CVXPY) Feb 15 11:17:17 PM: Problem status: optimal
(CVXPY) Feb 15 11:17:17 PM: Optimal value: 9.245e+06
(CVXPY) Feb 15 11:17:17 PM: Compilation took 9.011e-03 seconds
(CVXPY) Feb 15 11:17:17 PM: Solver (including time spent in interface) took 9.971e-04 seconds
第一问最优值为： 9244730.0
第一问最优解为：
 [[  -0.   -0.   -0.   -0.   -0. 3000.   -0.   -0.]
 [  -0.   -0.   -0.   -0. 3100.   -0.   -0.   -0.]
 [  -0.   -0.    0.   -0.   -0.   -0.   -0. 2900.]
 [  -0.   -0. 3100.   -0.   -0.   -0.   -0.   -0.]
 [3100.   -0.   -0.   -0.   -0.   -0.   -0.   -0.]
 [  -0.   -0.   -0. 3400.   -0.   -0.   -0.   -0.]
 [  -0. 3500.   -0.   -0.   -0.   -0.   -0.   -0.]
 [  -0.   -0.   -0. 3200.   -0.   -0.   -0.   -0.]
 [  -0. 3000.   -0.   -0.   -0.   -0.   -0.   -0.]
 [  -0.   -0.   -0.   -0.   -0.   -0.   -0. 3100.]
 [  -0.   -0.   -0.   -0.   -0.   -0. 3300.   -0.]
 [  -0.   -0.   -0.   -0.   -0.   -0. 3200.   -0.]
 [  -0.   -0.   -0.   -0.   -0.   -0. 3300.   -0.]
 [  -0.   -0.   -0.   -0.   -0.   -0.   -0. 2900.]
 [  -0.   -0.   -0.   -0.   -0.   -0.   -0. 3100.]]
```

图 13 cvxpy 包计算结果

第二问需要实现 x=0 或 1000<x<=2000。令 a 为一个 0-1 分布的变量(要么为 0 要么为 1)，再令 1000<x<=2000，则

$$ax = 0 \text{ or } 1000 < ax \leq 2000$$

程序求解得到：

```
(CVXPY) Feb 15 11:24:40 PM: Problem status: optimal
(CVXPY) Feb 15 11:24:40 PM: Optimal value: 1.197e+07
(CVXPY) Feb 15 11:24:40 PM: Compilation took 7.980e-03 seconds
(CVXPY) Feb 15 11:24:40 PM: Solver (including time spent in interface) took 6.990e-03 seconds
第二问最优值为： 11966110.0
第二问最优解为：
 [[   0.   -0.   -0.   -0. 1000. 2000.   -0.    0.]
 [   0.    0.   -0.   -0. 2000. 1100.   -0.    0.]
 [  -0.    0. 1000.   -0.   -0.   -0.   -0. 1900.]
 [  -0.    0. 2000.   -0. 1100.   -0.   -0.   -0.]
 [2000.    0.   -0. 1100.   -0.   -0.   -0.   -0.]
 [1400.    0.   -0. 2000.   -0.   -0.   -0.   -0.]
 [1500. 2000.   -0.   -0.   -0.   -0.   -0.   -0.]
 [  -0.   -0.   -0. 2000. 1200.   -0.   -0.    0.]
 [1000. 2000.   -0.   -0.   -0.   -0.   -0.    0.]
 [  -0.   -0. 1100.   -0.   -0.   -0.   -0. 2000.]
 [  -0.   -0.   -0.   -0.   -0.   -0. 2000. 1300.]
 [  -0.   -0.   -0.   -0. 1200.   -0. 2000.    0.]
 [  -0.   -0.   -0. 1300.   -0.   -0. 2000.    0.]
 [  -0.   -0. 1000.   -0.   -0.   -0.   -0. 1900.]
 [   0.    0.    0. 1100.    0.    0.    0. 2000.]]
是否配送的选择为：
 [[0. 0. 0. 0. 1. 1. 1. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1. 1. 0. 1. 0. 0. 0. 0. 1. 0. 1.]
 [1. 1. 0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0.]
 [1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 1. 1.]]
```

图 14 第二问的计算结果

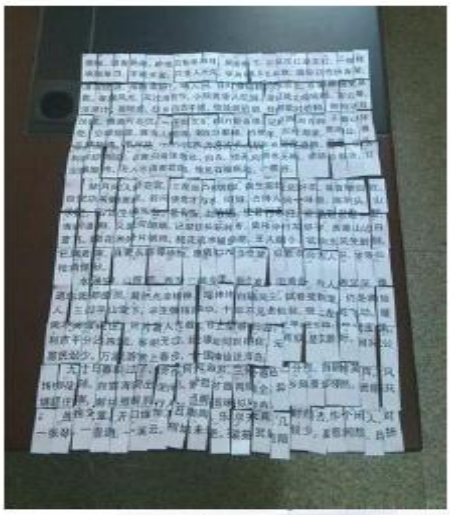
代码:

```
import numpy as np
import cvxpy as cp

c = np.mat(    [[390.6, 618.5 , 553  , 442  , 113.1, 5.2   , 1217.7, 1011],
                [370.8, 636   , 440  , 401.8, 25.6 , 113.1 , 1172.4, 894.5],
                [876.3, 1098.6, 497.6, 779.8, 903  , 1003.3, 907.2 , 40.1 ],
                [745.4, 1037  , 305.9, 725.7, 445.7, 531.4 , 1376.4, 768.1],
                [144.5, 354.5 , 624.7, 238  , 290.7, 269.4 , 993.2,  974],
                [200.2, 242   , 691.5, 173.4, 560  , 589.7 , 661.8 , 855.7],
```

```
                [235  , 205.5 , 801.5, 326.6, 477  , 433.6 , 966.4 , 1112],
                [517  , 541.5 , 338.4, 219  , 249.5, 335   , 937.3 , 701.8],
                [542  , 321   , 1104 , 576  , 896.8, 878.4 , 728.3 , 1243],
                [665  , 827   , 427  , 523.2, 725.2, 813.8 , 692.2 , 284],
                [799  , 855.1 , 916.5, 709.3, 1057 , 1115.5, 300   , 617],
                [852.2, 798   , 1083 , 714.6, 117.4, 1216.8, 40.8  , 898.2],
                [602  , 614   , 820  , 517.7, 899.6, 952.7 , 272.4 , 727],
                [903  , 1092.5, 612.5, 790  , 932.4, 1034.9, 777   , 152.3],
                [600.7, 710   , 522  , 448  , 726.6, 811.8 , 563   , 426.8]])

 a = np.mat([3000, 3100, 2900, 3100, 3100, 3400, 3500, 3200, 3000, 3100, 3300, 3200, 3300, 2900, 3100])
 b = np.mat([18600, 19600, 17100, 18900, 17000, 19100, 20500, 17200])


 d = a.reshape(1,-1)#需求量向量
 e = b.reshape(-1,1)#储存量向量
 x = cp.Variable((8,15))
 a = cp.Variable((8,15),integer = True) #0-1 变量,须有参数 integer = True 的设定才可定义 0-1 变量

 obj =cp.Minimize(cp.sum(cp.multiply(c.T, x)))
 # con1 = [cp.sum(x,axis = 1,keepdims = True) <= e,cp.sum(x,axis = 0,keepdims = True) == d,0<=x]
 #
 con2 = [cp.sum(x,axis = 1,keepdims = True) <= e,cp.sum(x,axis = 0,keepdims = True) == d, 0 <=a, a <= 1, 1000*a <= x,x <= 2000*a]  #约束条件，其中 0-1 变量就设为大于等 0+小于等于 1 即可
 # prob1 = cp.Problem(obj,con1)
 prob2 = cp.Problem(obj,con2)
 # prob1.solve(solver = 'GLPK_MI',verbose = True)
 prob2.solve(solver = 'GLPK_MI',verbose = True)
 print("第二问最优值为：",prob2.value)
 print("第二问最优解为：\n",x.value.T)
 print("是否配送的选择为：\n",a.value)
```

2. 破碎文件的拼接在司法物证复原、历史文献修复以及军事情报获取等领域都有着重要的应用。随着计算机技术的发展，人们试图开发碎纸片的自动拼接技术，以提高拼接复原效率。

现有一页纸质文档，被碎纸机纵横切成 $M \times N$ 个碎纸片（如下图所示），数据见附件 1，需要设计一个自动拼接复原算法。

现有两个拼接方法，描述如下：

方法一：一次性确定所有碎纸片的位置。

已知一张纸被切成 M 行、N 列，则每张纸片可用其所在的行、列编号来表述其位置。于是，将纸片依次编号为 $1,2,\cdots,MN$，令

$$x_{i,j,k}=\begin{cases}1, & \text{第}k\text{张纸片放在第 }i\text{行第}j\text{列}\\ 0, & \text{否则}\end{cases}$$

$$i=1,2,\cdots,M;\quad j=1,2,\cdots,N;\quad k=1,2,\cdots,MN$$

令 $c_{i,j}$ 表示碎纸片 i 和 j 左右拼接（i 左 j 右）的差异度指标值，$d_{i,j}$ 表示碎纸片 i 和 j 上下拼接（i 上 j 下）的差异度指标值。

定义目标函数为：所有碎片的左右和上下差异度指标值之和最小，即

$$\min z=\sum_{i=1}^{M}\sum_{j=1}^{N-1}\sum_{k=1}^{MN}\sum_{l=1,l\neq k}^{MN}c_{k,l}x_{i,j,k}x_{i,j+1,l}+\sum_{i=1}^{M-1}\sum_{j=1}^{N}\sum_{k=1}^{MN}\sum_{l=1,l\neq k}^{MN}d_{k,l}x_{i,j,k}x_{i+1,j,l}$$

$$\text{s.t.}\quad \sum_{i=1}^{M}\sum_{j=1}^{N}x_{i,j,k}=1,\quad k=1,2,\cdots,MN\ \text{（每个碎片只能放在一个位置上）}$$

$$\sum_{k=1}^{MN}x_{i,j,k}=1, i=1,2,\cdots,M; j=1,2,\cdots,N\ \text{（每个位置只能放一张碎片）}$$

方法二：分组确定碎纸片的位置。

（1）分行。假设每一行的最左边一块可以人工识别出来，依次记为 $p_i, i=1,2,\cdots,M$，其它碎片记为 $q_j, j=1,2,\cdots,N$，每一个 q_j 与 p_i 的相似度记为 $c_{i,j}$，令

$$x_{i,j}=\begin{cases}1, & \text{第}j\text{张纸片放在第 }i\text{行}\\ 0, & \text{否则}\end{cases}$$

定义目标函数为：碎片分组后的总相似度最大，即

$$\max z = \sum_i \sum_j c_{i,j} x_{i,j}$$

s.t. $\sum_{i=1}^{M} x_{i,j} = 1, \quad j = 1,2,\cdots,M(N-1)$ 每张碎片必在某一行

$\sum_{j=1}^{M(N-1)} x_{i,j} = N-1, \quad i = 1,2,\cdots,M$ 每行有 $N-1$ 张碎片

$x_{i,j} = 0$或1

（2）行内排序。将每一行的碎片依次编号为$1,2,\cdots,N$，定义两碎片之间的有向距离为$c_{i,j}$。

令 $x_{i,k} = \begin{cases} 1, & \text{第}i\text{块碎片在第}k\text{个位置上} \\ 0, & \text{否则} \end{cases}$

目标函数为： $\min z = \sum_{k=1}^{N-1} \sum_{i=1}^{N} \sum_{j=1, j\neq i}^{N} c_{i,j} x_{i,k} x_{j,k+1}$

s.t. $\sum_{i=1}^{N} x_{i,k} = 1, \quad k = 1,2,\cdots,N$

$x_{i,k} = 0$或1

请比较这两种方法的特点。如果需要你来完成，你选择哪个方法？请说明理由，并编程实现你选择的方法，给出方法描述、程序代码及计算结果。

答：我选择第二种方法，理由如下：

第一种方法需要同时考虑上下左右四个方向的差异度，变量较多，计算复杂度较高，共有 MN！=208！≈10^393 种可能。而第二种方法先确定了每一张碎片在哪一行，大约 MN*M 种可能，行内排序共 N*N！种可能性，如果分为 11 行，则共 M=11,N=18,共有 11*18*18*18*18！=6,402,373,705,728,000，复杂度比较小。

首先做第一步，先统计最左侧的 19 张碎片中文字的位置。分别统计每一张碎片图像的每一行是否有 0 元素，如果有则该行计为 1，反之则计为 0。得到一个一维向量 D。

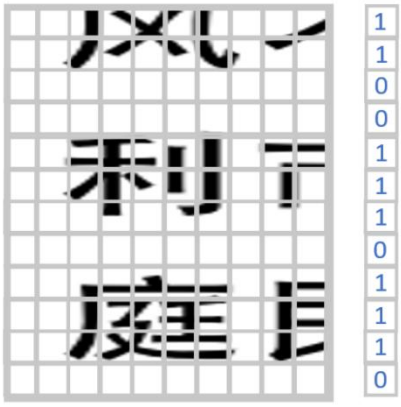


图 15 横向扫描

假设在相同行的碎片的文字上下位置基本一致，则可以认为在相同行的两幅碎片的 0 和 1 的位置接近。定义行差异度 Cr 为

$$Cr = abs(D0-D1)$$

根据 Cr 的定义求出 209 幅图片与最左侧的 19 张图片的差异度，得到一个 19X209 的矩阵 Crm。

由于直接使用 cvxpy 包无法求解第一步的最优化问题，故逐列扫描 Crm 矩阵，将每一列的最小值的下标作为该张碎片的行。

$$Row\_j = argmin(C[:,j])$$

完成行分类后，进行行内排序

首先定义差异度指数 Cij，表示第 i 张碎片右侧和第 j 张碎片左侧的差异度，为第 i 张碎片右侧与第 j 张碎片左侧的对应灰度值之差的绝对值的累和。公式如下:

$$C_{ij} = \left\{ \sum_{k=1}^{180} \left| P^{i}_{k,72} - P^{j}_{k,1} \right| \right.$$

其中$P^{i}_{k,72}$表示第 i 张碎片的第 k 行第 72 列的元素，$P^{j}_{k,1}$同理

因为已经确定了每一行的最左侧一张碎片和该行的所有碎片，可以按照根据 Cij 从左到右逐张拼接。

最后得到

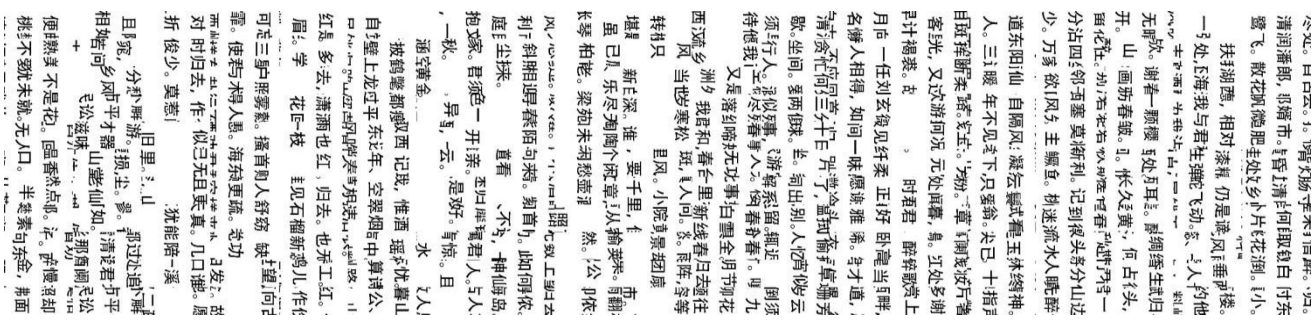


图 16 拼接结果

代码：

```
import numpy as np
import cv2
import os
import cvxpy as cp

path = "Final/Data/paper/"


def row_location(left, imgs):
    scores = inWhichRow(left, imgs)
    print(scores)
    return scores


def inWhichRow(left, imgs):
    scores = np.zeros([19, 209])
    for i in range(len(left)):
        element_location_left = start_end(left[i])
        for j in range(len(imgs)):
```

```
            scores[i, j] = compare_start_end(element_location_left, imgs[j])
    return scores


def compare_start_end(element_location_left, img1):
    element_location = start_end(img1)
    return np.sum(np.abs(element_location_left - element_location))


def getC(img0, img1):
    element_location_left = img0[:, -1]
    element_location_right = img1[:, 0]
    return np.sum(np.abs(element_location_left - element_location_right))


def getSim(imgs, left_img, right):
    c = np.zeros([len(right), 1])
    for i in range(len(right)):
        c[i] = getC(left_img, imgs[right[i]])
    index = np.argmin(c)
    return index, right[i]


def start_end(img0):
    element_location = np.zeros([180, 1])
    for i in range(180):
        if (img0[i, :] - 255).any():
            element_location[i] = 1
    return element_location


def read_img(path):
    file_name = os.listdir(path)
    imgs = []
    m = 0
    n = 0
    for i in file_name:
        name = path + i
        img = cv2.imread(name, cv2.IMREAD_GRAYSCALE)
        t, img = cv2.threshold(img, 128, 255, cv2.THRESH_BINARY)
        imgs.append(img)
    return imgs, file_name


```

```
imgs, file_name = read_img(path)
M = 19
N = 0
dictionary = dict()
base = imgs[0]
for img, name in zip(imgs, file_name):
    temp = (img[:, 0] - 255)
    n_nonzero = np.count_nonzero(temp)
    dictionary[name] = n_nonzero
    if not temp.any():
        N = N + 1
        base = np.concatenate([base, img], axis=0)
print(N)

cv2.imshow("title", base)
cv2.imwrite("img.jpg", base)
cv2.waitKey(0)

test = np.concatenate([imgs[0], imgs[7]], axis=1)
cv2.imshow("title", test)
cv2.imwrite("img.jpg", test)
cv2.waitKey(0)

left = []
new_sys2 = sorted(dictionary.items(), key=lambda d: d[1], reverse=False)
for i in range(19):
    index = int(new_sys2[i][0][:3])
    left.append(imgs[index])

scores = row_location(left, imgs)
print(scores.shape)

row_answer = []
for i in range(19):
    row_answer.append([])
    row_answer[i] = []

for i in range(209):
    data = scores[:, i]
    for j in range(1, 19):
        index = np.argsort(data)[j]
        if len(row_answer[index]) == 11:
            continue
        else:
```

```
            row_answer[index].append(i)
            break

print(row_answer)
# test = imgs[row_answer[15][0]]
# for i in range(1, 11):
#     test = np.concatenate([test, imgs[row_answer[15][i]]], axis=1)
# cv2.imshow("title", test)
# cv2.imwrite("img.jpg", test)
# cv2.waitKey(0)

col_answer = []
for i in range(19):
    col_answer.append([])
    col_answer[i] = []

for i in range(19):
    left_cur = left[i]
    while len(row_answer[i]) != 0:
        index, right = getSim(imgs, left_cur, row_answer[i])
        col_answer[i].append(row_answer[i][index])
        left_cur = imgs[row_answer[i].pop(index)]
print(len(col_answer[17]))
col_answer[17].append(col_answer[17][3])
for i in range(19):
    test = imgs[col_answer[i][0]]
    temp = col_answer[i]
    for j in range(1, 11):
        print(i, j)
        index = col_answer[i][j]
        test = np.concatenate([test, imgs[index]], axis=1)
    if i == 0:
        base = test
    else:
        base = np.concatenate([base, test], axis=0)

cv2.imshow("title", base)
cv2.imwrite("img.jpg", base)
cv2.waitKey(0)
```

3. 现有 95 个目标点的数据见附件 2。在每一行中第 1 列是目标点的编号，第 2、3 列分别是该目标点的 x, y 坐标，第 4 列是该目标点的重要性分类，“1”表示该点是第一类重要点，“2”表示该点是第二类重要点，未标明数字的是一般点，第 5、6、7 列标明该行目标点与这些点的连接关系。请完成以下问题：

（1）画出这些目标点的无向图，一类点用“☆”表示，二类点用“*”表示，一般点用“•”表示，

每个点的位置坐标也需要标出；

（2）当权重为距离时，求该无向图的最小生成树；

（3）分别求出点 L 到点 R3 和点 P 到 L2 的最短距离和最短路径，并画出最短路径。

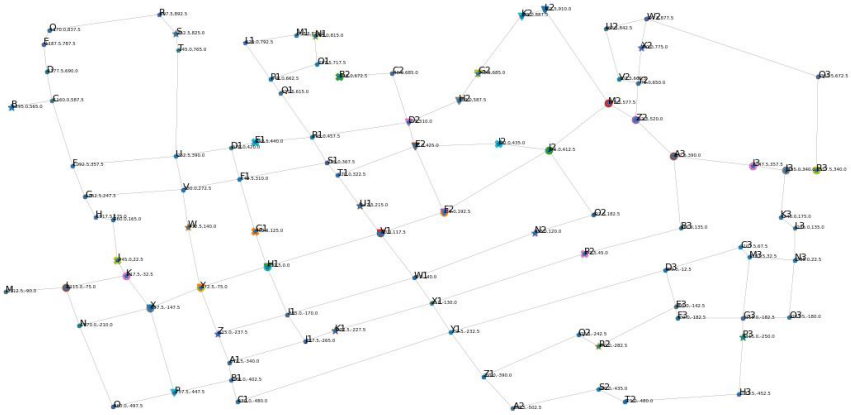


图 17 目标点(含坐标)

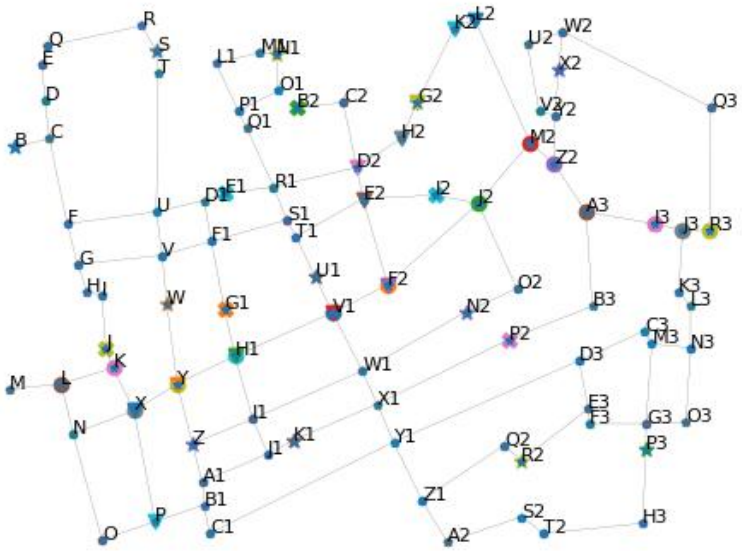


图 18 最短路径

图中圆形路径为 L 到 R3 的最短路径，三角形为 P 到 L2 的最短路径。

求解得到的最小生成树和最短路径如下图：

```
[95 rows x 7 columns]
最小生成树['B', 'C', 'D', 'F', 'E', 'Q', 'G', 'U', 'H', 'V', 'I', 'J', 'K', 'L', 'X', 'M', 'N', 'O', 'P', 'B1',
95

问题1: 无限制条件
L 到 R3 的最短加权路径:  ['L', 'K', 'X', 'Y', 'H1', 'V1', 'F2', 'J2', 'M2', 'Z2', 'A3', 'I3', 'J3', 'R3']
L 到 R3 的最短加权路径长度:  2795.4679098753263

问题1: 无限制条件
P 到 L2 的最短加权路径:  ['P', 'X', 'Y', 'H1', 'V1', 'F2', 'E2', 'D2', 'H2', 'G2', 'K2', 'L2']
P 到 L2 的最短加权路径长度:  2304.499008269838
```

图 19 最小生成树和最短路径求解结果

代码：

```python
import cv2
import numpy as np
import os
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
import cvxpy as cp

path = 'Final/Data\\附件 2-目标点数据.xlsx'


def distance(p0, p1):
    return np.sqrt((p0[0] - p1[0]) ** 2 + (p0[1] - p1[1]) ** 2)


def getPosDict(path):
    df = pd.read_excel(path)
    d_pos = dict()
    for row in df.index:
        pointName = df.at[row, '顶点']
        x = df.at[row, 'x 坐标']
        y = df.at[row, 'y 坐标']
        d_pos[pointName] = [x, y]
    return d_pos


def read_data(path, d_pos):
    G = nx.Graph()
    pos = dict()
    df = pd.read_excel(path)
    print(df)
```

```
    node_shape = []
    for row in df.index:
        pointName = df.at[row, '顶点']
        x = df.at[row, 'x 坐标']
        y = df.at[row, 'y 坐标']
        pos[pointName] = (x, y)

        cls = df.at[row, '顶点类别']
        if cls == 1:
            G.add_node(pointName)
            node_shape.append('*')
        elif cls == 2:
            G.add_node(pointName, node_shape='s')
            node_shape.append('X')
        else:
            G.add_node(pointName, node_shape='s')
            node_shape.append('.')
        plt.scatter(x, y, marker=node_shape[-1])
        plt.text(x, y, pointName, fontsize=8)
        # plt.text(x, y, str(x)+','+str(y), fontsize=4)
        near_p0 = df.at[row, '相邻的顶点 1']
        near_p0 = str(near_p0)
        if near_p0 != 'nan':
            d = distance(d_pos[near_p0], [x, y])
            G.add_edge(near_p0, pointName, weight=d)
        near_p1 = df.at[row, '相邻的顶点 2']
        near_p1 = str(near_p1)
        if near_p1 != 'nan':
            d = distance(d_pos[near_p1], [x, y])
            G.add_edge(near_p1, pointName, weight=d)
        near_p2 = df.at[row, '相邻的顶点 3']
        near_p2 = str(near_p2)
        if near_p2 != 'nan':
            d = distance(d_pos[near_p2], [x, y])
            G.add_edge(near_p2, pointName, weight=d)
    return G, pos, row, node_shape

d_pos = getPosDict(path)
G, pos, row, node_shape = read_data(path, d_pos)
T = nx.minimum_spanning_tree(G)
print(f'最小生成树{T.nodes}')
nx.draw(G, pos, width=0.1, alpha=1, node_shape='*', font_size=1, node_size=10)
plt.savefig("graph")
# plt.show()
```

```
print(len(G))

minWPath1 = nx.dijkstra_path(G, source='L', target='R3')  # 顶点 0 到 顶点 17 的最短加权路径
# 两个指定顶点之间的最短加权路径的长度
lMinWPath1 = nx.dijkstra_path_length(G, source='L', target='R3')  #最短加权路径长度
print("\n 问题 1: 无限制条件")
print("L 到 R3 的最短加权路径: ", minWPath1)
for point in minWPath1:
    plt.scatter(d_pos[point][0], d_pos[point][1], marker='o')
print("L 到 R3 的最短加权路径长度: ", lMinWPath1)

minWPath1 = nx.dijkstra_path(G, source='P', target='L2')  # 顶点 0 到 顶点 17 的最短加权路径
# 两个指定顶点之间的最短加权路径的长度
lMinWPath1 = nx.dijkstra_path_length(G, source='P', target='L2')  #最短加权路径长度
for point in minWPath1:
    plt.scatter(d_pos[point][0], d_pos[point][1], marker='v')
plt.savefig("cut")
plt.show()
print("\n 问题 1: 无限制条件")
print("P 到 L2 的最短加权路径: ", minWPath1)
print("P 到 L2 的最短加权路径长度: ", lMinWPath1)
```

4. 无人机在抢险救灾中的优化运用。

2017 年 8 月 8 日，四川阿坝州九寨沟县发生 7.0 级地震，造成了不可挽回的人员伤亡和重大的财产损失。由于预测地震比较困难，及时高效的灾后救援是减少地震损失的重要措施。无人机作为一种新型运载工具，能够在救援行动中发挥重要作用。

附件 3 给出了震区的高程数据，共有 2913 列，2775 行。第一行第一列表示(0,0)点处的海拔高度值（单位：米），相邻单元格之间的距离为 38.2 米，即第 m 行第 n 列单元格中的数据代表坐标(38.2(m-1), 38.2(n-1))处的高度值。震区 7 个重点区域的中心位置如下表所示（单位：千米）：

中心点	X 坐标	Y 坐标
A	30.3	89.8
B	66.0	84.7
C	98.4	76.7
D	73.7	61.0
E	57.9	47.6
F	86.8	22.0
G	93.6	48.8

假设无人机平均飞行速度 60 千米/小时，最大续航时间为 8 小时，飞行时的转弯半径不小于 100 米。所有无人机均按规划好的航路自主飞行，无须人工控制，完成任务后自动返回原基地。

大地震发生后，及时了解灾区情况是制订救援方案的重要前提。为此，使用无人机携带视频采集装

置巡查 7 个重点区域中心方圆 10 公里(并集记为 S)以内的灾情。假设无人机飞行高度恒为 4200 米，若所有无人机均从基地 H(110,0)(单位：千米)处派出，且完成任务后再回到 H，希望在 4 小时之内使区域 S 内海拔 3000 米以下的地方尽可能多地被巡查到。请完成以下问题：

（1）画出该震区三维地形图，标出 7 个重点区域位置；

（2）为在规定的时间内完成巡查任务，最少需要派出多少架无人机？建立相应的模型并求解；

（3）设计每架无人机的飞行路线，并画出相应的飞行路线图及巡查到的区域（不同的无人机的飞行路线图用不同的颜色表示）。

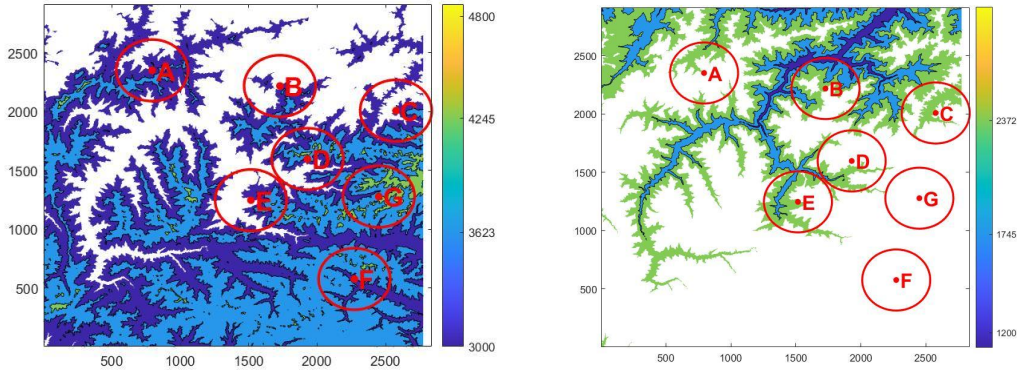


图 20 震区地图

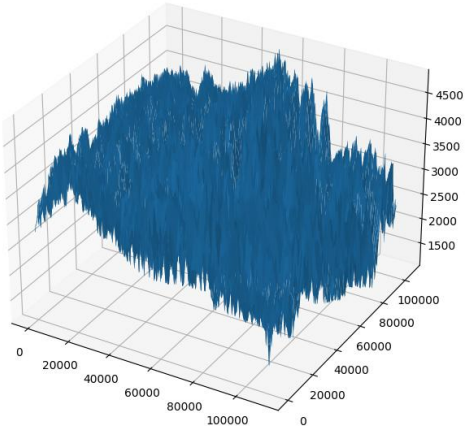


图 21 三维图

4. 围绕本课程所学某一方法，选择一篇你认为较好的期刊论文或会议论文，对论文工作进行概述，描述该方法在论文工作中所起的作用。

答：《ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION》发表于 ICLR2015，集中讨论了在高维参数空间中的随机梯度优化问题。文章提出的 ADAM 方法在非凸优化问题中具有实现简单，计算高效，占用内存小的特点。并具备梯度对角缩放的不变性，适合解决大规模数据和参数的优化问题。并通过实验验证了该方法适用于解决包含高噪声或稀疏梯度的问题以及非平稳目标问题。

文章主要工作包括：

1. 证明了 ADAM 算法的有界性

2. 证明了 ADAM 算法的梯度对角缩放不变性
3. 证明了 ADAM 算法的收敛性
4. 在三个实验(Logistic 回归，多层全连接神经网络和卷积神经网络)上与 Adagrad，RMSProp，SGDNestero 等优化方法进行对比，证明了 ADAM 在收敛速度上的优越性。
5. 讨论了 ADAM 方法的超参数设置，并针对 ADAM 方法收敛较快但精度不如 SGD 方法的问题提出了改进方向。