数学实验复习笔记

2024 年春-张晓平

作者：FANG Changrui

组织：School of Mathematics and Statistics, Wuhan University

时间：Tuesday 11th June, 2024

超越人类极限，做宇宙的主人

目录

第 1 章 声明

本文档仅做学习交流使用，仅在武汉大学数学与统计学院内部传阅，禁止用于商业用途，如有侵犯老师或各方权益，请联系删除，邮箱：作者邮箱。

本文档限于水平不足，一定存在诸多问题，如遇到问题，欢迎指正，邮箱：作者邮箱。

本文档唯一网上版本见 Github：WHU_MATH_course。其上包含武汉大学数学与统计学院课程的学习笔记、历年复习题、经验分享等，如有需要，欢迎下载。

第 2 章 sympy/numpy/scipy 的使用

2.1 高数问题符号解

2.1.1 求极限sympy.limit

验证 $\lim\limits_{x\to 0}\frac{\sin x}{x}=1,\quad \lim\limits_{x\to+\infty}(1+\frac{1}{x})^x=e$

```
from sympy import *
x = symbols('x')
print(limit(sin(x)/x, x, 0))
print(limit((1+1/x)**x, x, oo))
```

2.1.2 求导数sympy.diff

已知 $z=\sin x+x^2e^y$，求 $\frac{\partial^2 z}{\partial x^2},\frac{\partial z}{\partial y},\frac{\partial^2 z}{\partial x\partial y}$

```
from sympy import *
x, y = symbols('x y')
z = sin(x) + x**2*exp(y)
print(diff(z, x, 2))
print(diff(z, y, 1))
print(diff(z, x, 1, y, 1))
```

2.1.3 求级数sympy.summation

验证 $\sum\limits_{k=1}^{n}k^2=\frac{n(n+1)(2n+1)}{6},\quad \sum\limits_{k=1}^{\infty}\frac{1}{k^2}=\frac{\pi^2}{6}$

```
from sympy import *
k, n = symbols('k, n')
series1 = summation(k**2, (k, 1, n))
print(factor(series1)) # factor: 合并同类项
series2 = summation(1/k**2, (k, 1, oo))
print(series2)
```

2.1.4 泰勒展开sympy.series

已知 $f(x)=\sin x$，求 $f(x)$ 在 $x=0$ 处的泰勒展开式

```
from sympy import *
x = symbols('x')
y = sin(x)
print(series(y,x,0,1))
print(series(y,x,0,3))
print(series(y,x,0,5).removeO()) # .removeO(): 去除余项 O
```

2.1.5 不定积分和定积分sympy.integrate

验证 $\int_0^{\pi} \sin(2x)dx = 0, \int_0^{+\infty} \frac{\sin x}{x}dx = \frac{\pi}{2}$

```
from sympy import *
x = symbols('x')
print(integrate(sin(2*x), (x, 0, pi)))
print(integrate(sin(x)/x, (x, 0, oo)))
```

2.1.6 求解代数方程/组sympy.solve

$x^3 = 1, \quad (x-2)^2(x-1)^3 = 0$

```
from sympy import *
x = symbols('x')
f1 = x**3-1
f2 = (x-2)**2*(x-1)**3
print(solve(f1,x))
print(solve(f2))
print(roots(f2, x)) # 根与重根次数
```

$$\begin{cases} x^2 + y^2 = 1 \\ x - y = 0 \end{cases}$$

```
from sympy import *
x, y = symbols('x, y')
f1 = x**2 + y**2 - 1
f2 = x - y
solve([f1, f2], (x, y))
```

2.1.7 函数驻点和最值sympy.solve

求 $f(x) = 2x^3 - 5x^2 + x$ 驻点和在 $[0,1]$ 上最大值

```
from sympy import *
x = symbols('x')
y = 2*x**3 - 5*x**2 + x
print(solve(diff(y, x), x)) # 求驻点
```

2.1.8 微分方程符号解sympy.dsolve

齐次：$y'' - 5y' + 6y = 0$，　非齐次：$y'' - 5y' + 6y = 2e^{3x}$

初值问题: $y'' - 5y' + 6y = 0, y(0) = 1, y'(0) = 0$，　边值问题: $y'' - 5y' + 6y = xe^{2x}, y(0) = 1, y(2) = 0$

```
from sympy import *
x = symbols('x')
y = symbols('y', cls=Function)
```

```
eq1 = diff(y(x), x, 2) - 5*diff(y(x), x, 1) + 6*y(x)
eq2 = diff(y(x), x, 2) - 5*diff(y(x), x, 1) + 6*y(x) - x*exp(2*x)
print(dsolve(eq1)) # 齐次问题
print(dsolve(eq2)) # 非齐次问题
print(dsolve(eq1, y(x), ics={y(0):1, diff(y(x), x).subs(x, 0):0})) # 初值问题
print(dsolve(eq1, y(x), ics={y(0):1, y(2):2})) # 边值问题
```

2.2 高数问题数值解

2.2.1 泰勒级数实现

实现泰勒展开 $\sin x = \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k+1}}{(2k+1)!}, \quad x \in (-\infty, \infty).$

```
import numpy as np
import matplotlib.pyplot as plt
def frac(k): return 1 if k==0 else frac(k-1)*k
def item(x, k): return (-1)**k*x**(2*k+1)/(frac(2*k+1))
def mysin(x, k): return x if k==0 else mysin(x, k-1) + item(x, k)

x = np.linspace(-2*np.pi, 2*np.pi, 100)
plt.figure(dpi=200, figsize=(7, 2))
plt.plot(x, np.sin(x))
for k in [1, 3, 5]: plt.plot(x, mysin(x, k))
plt.legend(['sin', 'n=1', 'n=3', 'n=5'])
```

2.2.2 数值导数

$f'(x) = \lim_{h\to 0} \frac{f(x+h)-f(x)}{h} \approx \frac{f(x+\Delta x)-f(x)}{\Delta x} \quad f''(x) \approx \frac{f(x+\Delta x)-2f(x)+f(x-\Delta x)}{(\Delta x)^2}$

2.2.3 数值积分`scipy.integrate`

2.2.3.1 一重积分`scipy.integrate.quad`

使用复化梯形公式、复化辛普森公式和`scipy.integrate.quad`计算 $\int_0^1 \sin(\sqrt{\cos x + x^2})dx$

基本梯形公式:

$$\int_{x_i}^{x_{i+1}} f(x)dx \approx \frac{h}{2}\left(f(x_i) + f(x_{i+1})\right)$$

复化梯形公式:

$$\int_a^b f(x)dx \approx \frac{h}{2}[f(a) + 2\sum_{i=1}^{n-1} f(x_i) + f(b)]$$

基本辛普森公式:

$$\int_{x_{2i}}^{x_{2i+2}} f(x)dx \approx \frac{h}{3}\left(f(x_{2i}) + 4f(x_{2i+1}) + f(x_{2i+2})\right)$$

复化辛普森公式:

$$\int_a^b f(x)dx \approx \frac{h}{3}\left(f(a) + 4\sum_{i=1,\text{odd}}^{2N-1} f(a+ih) + 2\sum_{i=2,\text{even}}^{2N-2} f(a+ih) + f(b)\right), h = \frac{b-a}{2N}$$

```python
import numpy as np
from scipy.integrate import quad # 引入的模块
def trapezoid(f, a, b, n): # 复化梯形公式
    xi = np.linspace(a, b, n+1); h = (b-a)/n
    return h*(np.sum(f(xi))-(f(a)+f(b))/2)
def simpson(f, a, b, n): # 复化辛普森公式
    xi, h = np.linspace(a, b, 2*n+1), (b-a)/n
    fo = [f(xi[i]) for i in range(len(xi)) if i%2 != 0] # 奇数次编号点值
    fe = [f(xi[i]) for i in range(len(xi)) if i%2 == 0] # 偶数次编号点值
    return h*(2*np.sum(fe)+4*np.sum(fo)-f(a)-f(b))/6
#=============================================================================#
f = lambda x: np.sin(np.sqrt(np.cos(x)+x**2))
s1 = trapezoid(f, 0, 1, 200) # 梯形公式
s2 = simpson(f, 0, 1, 100) # 辛普森公式
s3 = quad(f, 0, 1) # 直接使用scipy.integrate.quad
print(s1, s2, s3[0])
```

2.2.3.2 多重积分scipy.integrate.dblquad&scipy.integrate.tplquad

$I_1 = \int_0^2 dx \int_0^1 xy^3 dy \quad I_2 = \iint_{x^2+y^2\le 1} e^{-\frac{x^2}{2}} \sin(x^2+y) dxdy$

$I_3 = \iiint_\Omega z\sqrt{x^2+y^2+1} dxdydz$, 其中$\Omega$ 为柱面$x^2+y^2-2x=0$ 与$z=0, z=6$ 两平面所围成的空间区域

$\Rightarrow I = \int_0^2 dx \int_{-\sqrt{2x-x^2}}^{\sqrt{2x-x^2}} dy \int_0^6 z\sqrt{x^2+y^2+1} dz$

```python
from scipy.integrate import dblquad, tplquad
f1 = lambda y, x: x*y**3 # 注意lambda的变量顺序，这是函数的要求
I1 = dblquad(f1, 0, 2, 0, 1)

f2 = lambda y, x: np.exp(-x**2/2)*np.sin(x**2+y)
bd1 = lambda x: np.sqrt(1-x**2) # 积分边界自己化
I2 = dblquad(f2, -1, 1, lambda x: -bd1(x), lambda x: bd1(x))
#=============================================================================#
f3 = lambda z, y, x: z*np.sqrt(x**2+y**2+1) # 注意lambda的变量顺序，这是函数的要求
bd2 = lambda x: np.sqrt(2*x-x**2) # 积分边界自己化
I3 = tplquad(f3, 0, 2, lambda x: -bd2(x), lambda x: bd2(x), 0, 6)
```

2.2.4 非线性方程（组）数值解scipy.optimize.fsolve

二分法： 适用范围：1. 函数在给定区间内是连续的；2. 在给定区间两端函数值异号

优点： 1. 简单易实现；2. 有全局收敛性

缺点： 1. 收敛速度慢；2 无法求出多重根；3. 要求区间两端值异号

牛顿法： 适用范围：1. 连续可微函数；2. 合理的初始猜测 x_0

优点： 1. 收敛速度快 (二次收敛)

缺点： 1. 依赖初始猜测；2. 要计算导数 (高计算复杂度)；3. 无法求出多重根

推导： $f(x) \approx f(x_n) + f'(x_n)(x-x_n)$，希望 $f(x)=0$，则 $0 = f(x_n) + f'(x_n)(x_{n+1}-x_n)$，解得 $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$

1. 使用二分法、牛顿法和直接调用scipy库求 $x^3+1.1x^2+0.9x-1.4=0$ 根的近似值，使得误差不超过 10^{-6}

2. 求 $\begin{cases} 5x_2+3=0 \\ 4x_1^2-2\sin(x_2x_3)=0 \\ x_2x_3-1.5=0 \end{cases}$

```python
import numpy as np
def binary_search(f, a, b, eps): # 二分法
    c = (a+b)/2
    while np.abs(f(c)) > eps:
        if f(a)*f(c)<0: b = c
        else: a = c
        c = (a+b)/2
    return c

def newton(f, x0, dx, eps):
    def diff(f, x0, dx):
        return (f(x0+dx)-f(x0))/dx
    while np.abs(f(x0)) >= eps:
        x0 = x0 - f(x0)/diff(f, x0, dx)
    return x0

f = lambda x: x**3+1.1*x**2+0.9*x-1.4
x1 = binary_search(f, 0, 1, 1e-6)
x2 = newton(f, 1, 0.01, 1e-6)
#-----------------------------------------------------------------------------#
from scipy.optimize import fsolve
x3 = fsolve(f, 0)[0] # 这里的 0 是初始值, 非 f=0
#=============================================================================#
g = lambda x: [5*x[1]+3, 4*x[0]**2-2*np.sin(x[1]*x[2]), x[1]*x[2]-1.5]
x = fsolve(g, [1, 1, 1])
```

2.2.5 函数极值点数值求解scipy.optimize

2.2.5.1 一元函数区间极值点scipy.optimize.fminbound

$f(x)=e^x\cos(2x)$ 在 $[0,3]$ 上极小点

```python
import numpy as np
from scipy.optimize import fminbound

f = lambda x: np.exp(x)*np.cos(2*x)
x0 = fminbound(f, 0, 3)
x0, f(x0)
```

2.2.5.2 一元函数点附近极值点scipy.optimize.fmin

$f(x)=e^x\cos(2x)$ 在 0 附近的一个极小点

```python
import numpy as np
```

```
from scipy.optimize import fmin

f = lambda x: np.exp(x)*np.cos(2*x)
x0 = fmin(f, 0)
x0, f(x0)
```

2.2.5.3 多元函数的极值点scipy.optimize.minimize

求 $f(\boldsymbol{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$ 极小值

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

f = lambda x: 100*(x[1]-x[0]**2)**2+(1-x[0])**2
x0 = minimize(f, [0, 0], method='BFGS')
x0.x, f(x0.x), x0.fun
```

2.3 线代问题符号解

2.3.1 向量/矩阵相关计算sympy.Matrix

$\alpha = (1, 2, 3)^T, \quad \beta = (4, 5, 6)^T$, 求 $\|\alpha\|_2, \alpha^T, \alpha \cdot \beta, \alpha \times \beta$

```
import sympy
a = sympy.Matrix([[1], [2], [3]])
b = sympy.Matrix([[4], [5], [6]])
print(a.norm().evalf(3)) # 范数
print(a.T) # 转置
print(a.dot(b)) # 点乘
print(a.cross(b)) # 叉乘
#============================================================================#
A = sympy.Matrix(np.arange(1, 17, 1).reshape(4, 4))
B = sympy.diag(1, 2, 3, 4)
I = sympy.eye(4)

print(A.norm()) # 矩阵的范
print(sympy.det(A)) # 行列式
print(A.rank()) # 矩阵的秩
print(A.T) # 矩阵转置
print(A.transpose()) # 矩阵转置
print(B.inv()) # 矩阵求逆
print(A * B) # 矩阵乘法
print(A.row_join(B)) # 横向拼接(行拼接)
print(A.col_join(B)) # 纵向拼接(列拼接)
print(A[:2, :2]) # 取子矩阵
A1 = A.copy(); A1.row_del(3) # 删除第四行
```

2.3.2 解线性方程组sympy

2.3.2.1 唯一解A.inv()*B

```
# 唯一解
import sympy
A1 = sympy.Matrix ([[2,1,-5,1], [1,-3,0,-6], [0,2,-1,2], [1,4,-7,6]])
b = sympy.Matrix ([8,6,-2,2])
print(A.rank())
print(A1.inv()*b)
```

2.3.2.2 齐次方程基础解系A.nullspace()

```
# 基础解系 Ax=0
A2 = sympy.Matrix ([[1,-5,2,-3], [5,3,6,-1], [2,4,2,1]])
print(A2.nullspace())
```

2.3.2.3 无穷多解的行最简形C.rref()

```
# 无穷解
A3 = sympy.Matrix([[1,1,-3,-1], [3,-1,-3,4], [1,5,-9,-8]])
b3 = sympy.Matrix([1 ,4 ,0]); b3.transpose()
C = A2.row_join(b3)
print(C.rref())
```

2.3.3 矩阵的特征值问题

2.3.3.1 特征值A.eigenvals()与特征向量A.eigenvects()

求 $A=\begin{pmatrix} 0 & -2 & 2 \\ -2 & -3 & 4 \\ 2 & 4 & -3 \end{pmatrix}$ 的特征值与特征向量

```
import sympy
A = sympy.Matrix([[0, -2, 2], [-2, -3, 4], [2, 4, -3]])
print(A.eigenvals()) # 矩阵特征值
print(A.eigenvects()) # 矩阵特征向量
```

2.3.3.2 对角化A.diagonalize()

即求可逆矩阵 P，使得 $P^{-1}AP=D$

```
import sympy
A = sympy.Matrix([[0, -2, 2], [-2, -3, 4], [2, 4, -3]])
```

```
if A.is_diagonalizable():
    print(f"P={A.diagonalize()[0]}")
    print(f"D={A.diagonalize()[1]}")
else:
    print("A is not diagonalizable")
```

2.4 线代问题数值解numpy.linalg

2.4.1 NumPy 库中有关线代的重要函数

```
import numpy as np
A1 = np.eye(4) # 4*4单位阵
A2 = np.zeros((4, 3)) # 4*3全零矩阵
A3 = np.ones((4, 3)) # 4*3全一矩阵
A4 = np.vander((3,9,10), 4, increasing=True) # 第二列为(3, 9, 10), 4列, 从左往右递增的vanderment
A5 = np.outer(A1, A2) # 外积
A6 = np.inner(A2, A3) # 内积
A7 = np.dot(A1, A2) # 矩阵乘积
a1 = np.trace(A1) # 迹
A8 = np.transpose(A4) # 转置
A8 = A4.T # 转置
A9 = np.diag(A4) # 矩阵与一维数组的相互转换(仅对角部分)
```

2.4.2 numpy.linalg中有关线代的重要函数

```
import numpy as np
np.linalg.det(A1) # 行列式
np.linalg.eig(A1) # 方阵的特征值与特征向量
np.linalg.eigvals(A1) # 方阵的特征值
np.linalg.inv(A1) # 方阵的逆
np.linalg.pinv(A3) # 矩阵的伪逆
np.linalg.qr(A3) # 矩阵的QR分解
np.linalg.svd(A3) # 矩阵的奇异值分解
np.linalg.norm(A3) # 范数
np.linalg.matrix_rank(A3) # 秩
```

2.4.3 矩阵和向量运算

```
a = np.arange(1, 4); b = np.arange(5, 8)
np.linalg.norm(a) # 范数
np.dot(a, b) # 点乘
np.inner(a, b) # 内积
np.cross(a, b) # x乘
np.outer(a, b) # 外积
```

```
A, B, I = np.arange(1, 17).reshape(4 ,4), np.diag((1, 2, 3, 4)), np.eye(4)
np.linalg.det(A) # Determinant
np.linalg.matrix_rank(A) # 秩
np.linalg.inv(A+10*I) # 求逆
np.c_[A, B] # 按行拼接
np.r_[A, B] # 按列拼接
A1 = np.delete(A, 3, axis=0) # 删除第四行
```

2.4.4 线性方程组的数值解

2.4.4.1 齐次线性方程组基础解系np.linalg.nullspace()

```
import numpy as np
A = np.array ([[1,-5,2,-3],[5,3,6,-1],[2,4,2,1]])
np.linalg.null_space(A)
```

2.4.4.2 非齐次唯一解numpy.linalg.solve(A, b)/numpy.linalg.inv(A).dot(b)

```
import numpy as np
A = np.array([[2,1,-5,1],[1,-3,0,-6],[0,2,-1,2],[1,4,-7,6]])
b = np.array([8,6,-2,2]).reshape(4,1)
np.linalg.solve(A, b)
np.linalg.inv(A).dot(b)
np.linalg.pinv(A).dot(b)
```

2.4.4.3 非齐次无穷解numpy.linalg.lstsq()

```
A = np.array ([[1,1,-3,-1],[3,-1,-3,4],[1,5,-9,-8]])
b = np.array ([1 ,4 ,0]).reshape (3,1)
np.linalg.pinv(A).dot(b) # 最小二乘解
np.linalg.lstsq(A, b, rcond=-1) # 最小二乘解
```

2.4.4.4 矩阵的特征值问题

```
A = np.array ([[0,-2,2],[-2,-3,4],[2,4,-3]])
values, vectors = np.linalg.eig(A) # 特征值和特征向量
np.linalg.eigvals(A) # 特征值
np.allclose(A-vectors*values@np.linalg.inv(vectors), 0) # 检查分解
```

第 3 章 概率论与数理统计

3.1 随机变量的数字特征

期望： 离散型：$E[X]=\sum\limits_{k=1}^{\infty}x_kp_k$，连续型：$E[X]=\int_{-\infty}^{\infty}xf(x)dx$.

方差： $D(X)=\mathrm{Var}(X)=E\left[(X-E[X])^2\right]$

标准差： $\sqrt{D(X)}$

极差： $R(X)=X_{\max}-X_{\min}$

偏度： X 的标准化变量的三阶中心矩：$\nu_1=E\left[\left(\frac{X-E[X]}{\sqrt{D(X)}}\right)^3\right]=\frac{E\left[(X-E[X])^3\right]}{D(X)^{3/2}}$

峰度： X 的标准化变量的四阶中心矩：$\nu_2=E\left[\left(\frac{X-E[X]}{\sqrt{D(X)}}\right)^4\right]=\frac{E\left[(X-E[X])^4\right]}{D(X)^2}-3$

协方差： $\mathrm{Cov}(X,Y)=E\left[(X-E[X])\,(Y-E[Y])\right]$

相关系数： $\rho_{XY}=\frac{\mathrm{Cov}(X,Y)}{\sqrt{D(X)}\sqrt{D(Y)}}$

矩： k 阶原点矩：$E[X^k]$，k 阶中心矩：$E[(X-E[X])^k]$，$k+l$ 阶混合矩：$E[X^kY^l]$，$k+l$ 阶混合中心矩：$E[(X-E[X])^k\,(Y-E[Y])^l]$

协方差矩阵： $\boldsymbol{X}=(X_1,X_2,\cdots,X_n)$ 的协方差矩阵为 $\boldsymbol{C}$，$c_{ij}=\mathrm{Cov}(X_i,X_j)=E[(X_i-E[X_i])(X_j-E[X_j])]$

分布	参数	数学期望	方差
两点分布 $b(1,p)$	$0<p<1$	p	$p(1-p)$
二项分布 $b(n,p)$	$0<p<1,n\geq1$	np	$np(1-p)$
泊松分布 $\pi(\lambda)$	$\lambda>0$	λ	λ
均匀分布 $U(a,b)$	$a<b$	$\frac{a+b}{2}$	$\frac{(b-a)^2}{12}$
正态分布 $\mathcal{N}(\mu,\sigma^2)$	$\mu,\sigma>0$	μ	σ^2

表 3.1: 各种概率分布的数学期望和方差

3.2 描述性统计量

样本均值： $\bar{X}=\frac{1}{n}\sum\limits_{i=1}^{n}X_i$.

中位数： 将数据从小到大排序后位于中间位置的数 (奇数时) 或者中间两个数的均值 (偶数时)

样本方差： $S^2=\frac{1}{n-1}\sum\limits_{i=1}^{n}(X_i-\bar{X})^2$

样本标准差： $S=\sqrt{S^2}$

样本极差： $R=\max(X)-\min(X)$

样本矩： k 阶原点矩：$A_k=\frac{1}{n}\sum\limits_{i=1}^{n}X_i^k$，$k$ 阶中心矩：$B_k=\frac{1}{n}\sum\limits_{i=1}^{n}(X_i-\bar{X})^k$

样本偏度： $\nu_1=\frac{1}{S^3}\sum\limits_{i=1}^{n}(X_i-\bar{X})^3$，$v_1>0$ 为右偏态，位于均值右边的数据更多，反之同理

样本峰度： $\nu_2=\frac{1}{S^4}\sum\limits_{i=1}^{n}(X_i-\bar{X})^4-3$，正态分布的峰度为 0(若不减 3 则为 3)，$\nu_2\gg3$ 表示样本含有较多远离均值的数据

样本协方差： $\mathrm{Cov}(X,Y)=\frac{1}{n-1}\sum\limits_{i=1}^{n}(X_i-\bar{X})(Y_i-\bar{Y})$

样本相关系数： $\rho_{XY} = \frac{\mathrm{Cov}(X,Y)}{S_X S_Y} = \frac{\sum_{i=1}^{n}(X_i-\bar{X})(Y_i-\bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i-\bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i-\bar{Y})^2}}$

3.3 使用 Pandas 读取数据文件

```python
import pandas as pd
df_csv = pd.read_csv('path/to/your/file.csv') # 读取本地 CSV 文件
df_csv = pd.read_csv('https://example.com/your/file.csv') # 读取远程 CSV 文件
print(df_csv.head())

df_excel = pd.read_excel('path/to/your/file.xlsx', sheet_name='Sheet1') # 读取本地 Excel 文件
print(df_excel.head())
valuse = df_excel.values() # 赋值，转换为 numpy 数组
```

3.4 Python 计算统计量

3.4.1 numpy 库计算统计量

函数	mean	median	ptp	var	std	cov	corrcoef
功能	均值	中位数	极差	方差	标准差	协方差	相关系数
函数	count	mad	mode	skew	kurt	quantile	
功能	计数	中位数绝对偏差	众数	偏度	峰度	样本中位数 (默认 50%)	

表 3.2: NumPy 库中计算统计量的函数

3.5 统计图

3.5.1 直方图hist

```python
import matplotlib.pyplot as plt
plt.hist(x, bins=None ,range=None ,density=False, weights=None, cumulative=False , bottom=None ,
    histtype='bar', align='mid', orientation='vertical ', rwidth=None, log=False, color=None,
    label=None, stacked=False)
```

3.5.2 箱线图boxplot

p **分位数** x_p：至少有 np 个样本 $\leq x_p$，至少有 $(1-p)n$ 个样本 $\geq x_p$

```python
import matplotlib.pyplot as plt
boxplot(x, notch=None, sym=None, vert=None, whis=None, positions=None, widths=None, labels=None)
```

3.5.3 Q-Q 图scipy.stats.probplot

常用于检验**拟合优度**:： 将经验分布函数的分位数与分布模型的理论分位数作为一对数画在直角坐标系中，如果这些点看起来像一条直线，则说明观测数据与分布模型的拟合效果很好。

```
from scipy.stats import norm , probplot
import matplotlib.pyplot as plt

probplot(h, plot=plt)
```

第 4 章 规划问题

4.1 scipy.optimize模块linprog

标准型：

$$\min \quad z = c^T x,$$
$$\text{s.t.} \begin{cases} Ax \leq b, \\ A_{\text{eq}} x = b_{\text{eq}}, \\ l \leq x \leq u. \end{cases}$$

基本格式：

```
from scipy.opitmize import linprog
res = linprog(c, A, b, Aeq, beq, bounds, method) # 这里面的c, A等可以用列表[]
print(f"minimum of obj func: {res.fun:.3f}")
print(f"optimal solution: {res.x}")
```

```
from scipy.optimize import linprog
c = [-1, 2, 3]; A =[[-2, 1, 1], [3, -1, -2]]; b = [9, -4]; Aeq = [[4, -2, -1]]; beq = [-6]; bounds
     = [[-10, None], [0, None], [None, None]]
res = linprog(c, A, b, Aeq, beq, bounds)
print(f"{res.fun:.3f}")
print(f"{res.x}")
```

4.2 cvxopt.solvers&cvxopt.matrix模块

标准型：

$$\min \quad z = c^T x,$$
$$\text{s.t.} \quad \begin{cases} Ax \leq b, \\ A_{\text{eq}} x = b_{\text{eq}}. \end{cases}$$

基本格式：

```
from cvxopt import solvers, matrix
sol = solvers.lp(c, A, b, Aeq, beq)
print(f"minimum of obj func: {sol['x'][0], sol['x'][1]}")
print(f"optimal solution: {sol['primal objective']:.3f}")
```

```
from cvxopt import solvers, matrix
c = matrix([-1., 4]); A = matrix([[-3., 1], [1., 2]]); b = matrix([6., 4]); Aeq = matrix([[0.],
    [-1.]]); beq=matrix([3.])
sol = solvers.lp(c, A, b, Aeq, beq)
print(f"minimum of obj func: {sol['x'][0], sol['x'][1]}")
print(f"optimal solution: {sol['primal objective']:.3f}")
```

4.3 cvxpy模块

流程：

1. **读取数据**：pandas
2. **问题设置**：变量 (cp.Variable)，目标函数 (cp.Minimize)，约束 ([ ])
3. **问题求解**：problem.solve(solver='GLPK_MI',verbose=False)
4. **结果输出**：目标函数值 (problem.value)，变量值 (x.value)

```
import numpy as np
import cvxpy as cp
import pandas as pd

df = pd.read_excel("../各部分代码/05@data/data/ex05_06.xlsx", header=None)
data = df.values; c = data[:-1, :-1]; e = data[:-1,-1].reshape(-1,1); d = data[-1,:-1].reshape
    (1,-1);

x = cp.Variable((6, 8))
obj = cp.Minimize(cp.sum(cp.multiply(c, x)))
con = [cp.sum(x, axis=1, keepdims=True)<=e, cp.sum(x, axis=0, keepdims=True)==d, x>=0]
prob = cp.Problem(obj, con)
prob.solve(solver='GLPK_MI',verbose=False) # verbose: 是否输出过程
np.set_printoptions(precision=3)
print(f"optimal value of obj func: {prob.value :.3f}")
print(f"optimal solution :\n{x.value}")
```

4.4 整数规划问题

在cvxpy中，只需在cp.Variable中添加integer=True即可

```
import cvxpy as cp
import numpy

x = cp.Variable((1, 6), integer = True)
c = np.array([[-3, -5, -2, -4, -2, -3]])
weight =np.array([[8, 13, 6, 9, 5, 7]])
obj = cp.Minimize(cp.sum(cp.multiply(c, x)))
con = [0 <= x, x <= 1, cp.sum(cp.multiply(x, weight)) <= 24]
prob = cp.Problem(obj, con)
prob.solve(solver='GLPK_MI')
print(-prob.value)
print(x.value)
```

4.5 非线性规划

4.5.1 scipy.optimize.minimize

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

def obj(x): return (2+x[0])/(1+x[1])-3*x[0]+4*x[2]
LB = [0.1]*3; UB = [0.9]*3; bounds = tuple(zip(LB, UB))
res = minimize(obj, [1, 1, 1], bounds=bounds)
print(res.x, res.fun, res.success)
```

第 5 章 插值与拟合

5.1 插值

5.1.1 Lagrange 插值

构造一个 n 次多项式 L_n 满足 $L_n(x_i)=y_i, i=0,1,\cdots,n$. 得到

$$L_n(x)=\sum_{k=0}^{n} y_k l_k(x),\quad l_k(x)=\prod_{i=0}^{n}\frac{(x-x_i)}{(x_k-x_i)},$$

记 $\omega_{n+1}(x)=(x-x_0)(x-x_1)\cdots(x-x_n)=\prod_{k=0}^{n}(x-x_k)$,
则 $\omega'_{n+1}(x_k)=(x_k-x_0)\cdots(x_k-x_{k-1})(x_k-x_{k+1})\cdots(x_k-x_n)$，故

$$L_n(x)=\sum_{k=0}^{n} y_k\frac{\omega_{n+1}(x)}{(x-x_k)\omega'_{n+1}(x)}$$

5.1.2 分段低次插值

优点：1. 公式简单，计算量小 (如分段线性插值)；2. 有较好的收敛速度；3. 可避免计算机上做高次乘幂时常遇到的上溢和下溢的困难；4. 提高数值稳定性 (高次插值不稳定，如有龙格现象)

分段线性插值：基函数选用帽子函数 $l_i(x)=\begin{cases}\frac{x-x_{n-1}}{x_n-x_{n-1}}, & x_{n-1}\le x\le x_n\\ 0, & \text{其他}\end{cases}$

分段抛物线插值：在每个区间上为二次多项式，要用到 $y_{i+\frac{1}{2}}$ 的值

5.1.3 牛顿插值

5.1.3.1 差分

向前差分：$\Delta y_i=y_{i+1}-y_i$，**向后差分：**$\nabla y_i=y_i-y_{i-1}$
二阶差分：$\Delta^2 y_i=\Delta y_{i+1}-\Delta y_i=y_{i+2}-2y_{i+1}+y_i$，$\nabla^2 y_i=\nabla y_{i+1}-\nabla y_i=y_i-2y_{i-1}+y_{i-2}$
n 阶差分：$\Delta^n y_i=\Delta^{n-1}y_{i+1}-\Delta^{n-1}y_i$, $\nabla^n y_i=\nabla^{n-1}y_i-\nabla^{n-1}y_{i-1}$

5.1.3.2 差商

一阶差商：$f[x_i,x_{i+1}]=\frac{f(x_{i+1})-f(x_i)}{x_{i+1}-x_i}$
二阶差商：$f[x_i,x_{i+1},x_{i+2}]=\frac{f[x_{i+1},x_{i+2}]-f[x_i,x_{i+1}]}{x_{i+2}-x_i}$
k 阶差商：$f[x_i,x_{i+1},\cdots,x_{i+k}]=\frac{f[x_{i+1},x_{i+2},\cdots,x_{i+k}]-f[x_i,x_{i+1},\cdots,x_{i+k-1}]}{x_{i+k}-x_i}$

5.1.3.3 牛顿插值公式

一次牛顿插值：$f(x)=f(x_0)+f[x,x_0](x-x_0),\quad f[x,x_0]=f[x_0,x_1]+f[x,x_0,x_1](x-x_1)\Rightarrow$

$$f(x)=\underbrace{f(x_0)+f[x_0,x_1](x-x_0)}_{N_1(x)}+\underbrace{f[x,x_0,x_1](x-x_0)(x-x_1)}_{R_1^*(x)}$$

二次牛顿插值：$f(x)=f(x_0)+f[x_0,x_1](x-x_0)+f[x,x_0,x_1](x-x_0)(x-x_1)$
$f[x,x_0,x_1]=f[x_0,x_1,x_2]+f[x,x_0,x_1,x_2](x-x_2)\Rightarrow$

$$f(x)=\underbrace{f(x_0)+f[x_0,x_1](x-x_0)+f[x_0,x_1,x_2](x-x_0)(x-x_1)}_{N_2(x)}+\underbrace{f[x,x_0,x_1,x_2](x-x_0)(x-x_1)(x-x_2)}_{R_2^*(x)}$$

n 次牛顿插值： $f(x) = N_n(x) + R_n^*(x)$，其中 $N_n(x)$ 为 n 次牛顿插值多项式，$R_n^*(x)$ 为牛顿型插值余项。

$$N_n(x) = f(x_0)+f[x_0,x_1](x-x_0)+f[x_0,x_1,x_2](x-x_0)(x-x_1)+\cdots+f[x_0,x_1,\cdots,x_n](x-x_0)(x-x_1)\cdots(x-x_{n-1})$$

$$R_n^*(x) = f[x,x_0,x_1,\cdots,x_n](x-x_0)(x-x_1)\cdots(x-x_n).$$

5.1.4 样条插值

暂略

5.1.5 scipy.interpolate模块求解

一维插值函数： `interp1d: f = interp1d(x, y, kind='linear/nearest/zero/slinear/quadratic/cubic'); y = f(x_new)`

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d

x = np.arange(0, 25, 2); y = np.array ([12, 9, 9, 10, 18, 24, 28, 27, 25, 20, 18, 15, 13])
x_new = np.linspace(0, 24, 100)
f1 = interp1d(x, y); y1 = f1(x_new)
f2 = interp1d(x, y, kind='cubic'); y2 = f2(x_new)
fig = plt.figure(figsize=(11, 4))
plt.scatter(x, y, label='real data', c='r')
plt.plot(x_new, y1, label='$y_1$'); plt.plot(x_new, y2, label='$y_2$')
plt.legend()
```

二维插值函数： `interp2d: f = interp2d(x, y, z, 'cubic '); zn = f(xn , yn)`

多维插值函数： `interpn&interpnd:`

5.2 拟合

线性拟合： $f(x) = a_1\varphi_1(x) + a_2\varphi_2(x) + \cdots + a_n\varphi_n(x), \quad m(\text{数据点}) > n(\text{基函数})$

非线性拟合： 拟合函数 $f(x)$ 非线性，需要求解非线性函数的极小化问题

5.2.1 numpy.polyfit

调用： `polyfit(x, y, deg)` **预测：** `yhat = polyval(f, x_new)`

```
import numpy as np
from numpy import polyfit, polyval
import matplotlib.pyplot as plt

x0 = np.arange(0, 1.1, 0.1); x_new = np.linspace(0, 2, 100)
y0 = np.array ([ -0.447 , 1.978 , 3.28, 6.16, 7.08, 7.34, 7.66, 9.56, 9.48, 9.30, 11.2])
f = polyfit(x0, y0, deg=2) # deg为多项式阶数
yhat = polyval(f, [0.25, 0.35]) # 预测 x=0.25 和 0.35 时的 y
plt.plot(x0 , y0 , '*', x_new , polyval(f, x_new), '-')
```

5.2.2 scipy.optimize.leastsq/curve_fit

调用： curve_fit(func, xdata, ydata) **返回值：** popt 为拟合参数，pcov 为参数的协方差矩阵

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

f = lambda x, a, b, c: a*np.exp(b*x[0])+c*x[1]**2
x = np.array ([6, 2, 6, 7, 4, 2, 5, 9]); y = np.array ([4, 9, 5, 3, 8, 5, 8, 2]); z = np.array
    ([5, 2, 1, 9, 7, 4, 3, 3])
xy = np.vstack((x, y)) # 由于输入位置仅有一个故需要堆叠
popt , pcov = curve_fit(f, xy, z) # popt:拟合参数，pcov 为参数的协方差矩阵
```

5.3 线性/非线性最小二乘

5.3.1 线性最小二乘

令 $f(x)=a_1\varphi_1(x)+a_2\varphi_2(x)+\cdots+a_n\varphi_n(x),\quad m>n$，将数据 $\{(x_i,y_i)\}_{i=1}^m$ 代入得到**矛盾方程组** $(m\gg n)$

$$Ca=y,\quad C\in\mathbb{R}^{m\times n},a\in\mathbb{R}^n,y\in\mathbb{R}^m.$$

问题转化： 定义**偏差向量** $\delta=Ca-y$，根据最小二乘原则将其转化为求解优化问题

$$\boldsymbol{a}^*=\arg\min_{\boldsymbol{a}\in\mathbb{R}^n}Q$$

其中 $Q=\|\delta\|_2^2=\|Ca-y\|_2^2=a^TCa-2a^TC^Ty+y^Ty$，$\boldsymbol{a}^*$ 称为矛盾方程组的**最小二乘解**。

法方程组推导： Q 取得极值必要条件：$\frac{\partial Q}{\partial a}=\frac{\partial}{\partial a}\left(a^TC^TCa-2a^TC^Ty+y^Ty\right)=0\Rightarrow C^TCa=C^Ty$，其中

$$C=\begin{bmatrix}a_{11}&a_{12}&\cdots&a_{1n}\\a_{21}&a_{22}&\cdots&a_{2n}\\\vdots&\vdots&&\vdots\\a_{m1}&a_{m2}&\cdots&a_{mn}\end{bmatrix},\quad a=\begin{bmatrix}x_1\\x_2\\\vdots\\x_n\end{bmatrix},\quad y=\begin{bmatrix}b_1\\b_2\\\vdots\\b_m\end{bmatrix}$$

因此使用最小二乘方法求解矛盾方程组 $Ca=y$ 的步骤为：

1. 计算 C^TC 和 C^Ty 得**法方程组**
2. 求解法方程组 $C^TCa=C^Ty$ 得最小二乘解 $\boldsymbol{a}^*=(C^TC)^{-1}Cy=C^\dagger y$，其中 $C^\dagger=(C^TC)^{-1}C^T\in\mathbb{R}^{n\times m}$ 为 C 的 **Moore-Penrose 广义逆**

第 6 章 微分方程模型

6.1 求解方法

向前 Euler： $y'(t_n) \approx \frac{y(t_{n+1})-y(t_n)}{h} \Rightarrow y(t_{n+1}) \approx y(t_n) + hf(t_n, y(t_n))$

向后 Euler： $y'(t_{n+1}) \approx \frac{y(t_{n+1})-y(t_n)}{h} \Rightarrow y(t_{n+1}) \approx y(t_n) + hf(t_{n+1}, y(t_{n+1}))$，隐式格式，每一步迭代求解非线性方程，计算困难

两点 Euler： $y'(t_n) \approx \frac{y(t_{n+1})-y(t_{n-1})}{2h} \Rightarrow y(t_{n+1}) \approx y(t_{n-1}) + 2hf(t_n, y(t_n))$

梯形公式： $y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(t, y(t))dt \Rightarrow y(t_{n+1}) \approx y(t_n) + \frac{h}{2}[f(t_{n+1}, y(t_{n+1})) + f(t_n, y(t_n))]$

预估-校正格式：
$$\begin{cases} y_{n+1}^* = y_n + hf(t_n, y_n), \\ y_{n+1} = y_n + \frac{h}{2}\left[f(t_n, y_n) + f(t_{n+1}, y_{n+1}^*)\right] \end{cases}$$
实际上先使用向前 Euler 预估，再代入梯形公式计算 (校正)

6.2 符号解`sympy.fsolve`

求解微分方程组
$$\begin{cases} \frac{dx_1}{dt} = 2x_1 - 3x_2 + 3x_3, & x_1(0) = 1, \\ \frac{dx_2}{dt} = 4x_1 - 5x_2 + 3x_3, & x_2(0) = 2, \\ \frac{dx_3}{dt} = 4x_1 - 4x_2 + 2x_3, & x_3(0) = 3. \end{cases}$$

```
import sympy
t = sympy.symbols('t')
x1, x2, x3 = sympy.symbols('x1, x2, x3', cls=sympy.Function)
eq = [diff(x1(t), t)-2*x1(t)+3*x2(t)-3*x3(t), diff(x2(t), t)-4*x1(t)+5*x2(t)-3*x3(t), diff(x3(t),
    t)-4*x1(t)+4*x2(t)-2*x3(t)]
ics = {x1(0):1, x2(0):2, x3(0):3}
sympy.dsolve(eq, ics=ics)
```

```
import numpy as np
from sympy import *
import matplotlib.pyplot as plt

t = symbols('t'); x, y = symbols('x y', cls=Function)
fun = [diff(x(t), t)-x(t)+2*y(t), diff(y(t), t)-x(t)-2*y(t)]; ics = {x(0):1, y(0):1}
x1, y1 = dsolve(fun, ics=ics)
X = x1.rhs; x_fun = lambdify(t, X, 'numpy')
Y = y1.rhs; y_fun = lambdify(t, Y, 'numpy')

t1 = np.linspace(0, 2, 100)
x_hat = x_fun(t1); y_hat = y_fun(t1)
plt.plot(t1, x_hat, label='x(t)'); plt.plot(t1, y_hat, label='y(t)')
plt.legend()
```

6.3 数值解scipy.integrate.odeint

求数值解并绘图 $\begin{aligned} &\frac{d^2y}{dx^2}+2\frac{dy}{dx}+2y=0,\\ &y(0)=0,\quad y'(0)=1. \end{aligned}$ 注意下面代码进行了转化：$\frac{dy_1}{dx}=y_2, \frac{dy_2}{dx}=-2y_1-2y_2$

```
from scipy.integrate import odeint
import sympy
import numpy as np
import matplotlib.pyplot as plt

def f(y, x):
    y1 , y2 = y # y1 为函数值, y2 为导数值
    return np.array ([y2 , -2*y1 -2*y2])
x = np.arange(0, 10, 0.1)
sol = odeint(f, [0.0, 1.0], x) # 数值解
y = sympy.symbols('y', cls=sympy.Function)
f = diff(y(t), t, 2) + 2*diff(y(t), t, 1) + 2*y(t)
ics = {y(0):0, diff(y(t), t, 1).subs(t, 0):1}
sympy.dsolve(f, ics=ics)

f_real = lambda t: np.exp(-t) * np.sin(t)
plt.plot(x, sol[:,0],'r', label="numerical solution") # 函数值的数值解
plt.plot(x, f_real(x),'b', label="symbols solution") # 函数值的符号解
plt.legend()
```

2022-2023 考题：

```
from scipy.integrate import odeint
def f(xy, t): return np.array([0.2*xy[0]-0.005*xy[0]*xy[1], -0.5*xy[1]+0.01*xy[0]*xy[1]])

t = np.arange(0, 10, 0.01)
sol = odeint(f, [70, 40], t)
plt.plot(t, sol[:,0]); plt.plot(t, sol[:,1])
```

第 7 章 图论

7.1 基本知识点

无向图： $G=(V,E)$，其中顶点集 $V=\{v_1,v_2,\cdots,v_n\}$，边集 $E=\{e_1,e_2,\cdots,e_m\}$
有向图： $D=(V,A)$，其中顶点集 $V=\{v_1,v_2,\cdots,v_n\}$，弧集 $A=\{a_1,a_2,\cdots,a_m\}$
无向图的连通图： 任意两顶点 u,v 间存在道路
有向图的强连通图任意两顶点 u,v 间存在双向道路

关联矩阵： 用于图的表示。无向图：$m_{ij}=\begin{cases}1, & v_i \text{ 与} e_j \text{ 相关联},\\ 0, & v_i \text{ 与} e_j \text{ 不关联}.\end{cases}$，有向图：$m_{ij}=\begin{cases}1, & v_i \text{ 是} e_j \text{ 的起点},\\ -1, & v_i \text{ 是} e_j \text{ 的终点},\\ 0, & v_i \text{ 与} e_j \text{ 不关联}.\end{cases}$

邻接矩阵：用于图的表示。$w_{ij}=\begin{cases}1, & v_i \text{ 与} v_j \text{ 相邻},\\ 0, & v_i \text{ 与} v_j \text{ 不相邻}.\end{cases}$，赋权图：$w_{ij}=\begin{cases}\text{顶点}v_i \text{ 与}v_j \text{ 之间边的权}, & v_i \text{ 与} v_j \text{ 相邻},\\ 0(\text{或}\infty), & v_i \text{ 与} v_j \text{ 不相邻}.\end{cases}$

7.2 networkx库

```python
import numpy as np
import networkx

a = np.zeros ((5 ,5))
a[0 ,1:5] = [9, 2, 4, 7]; a[1 ,2:4] = [3, 4]; a[2 ,[3 ,4]] = [8, 4]; a[3, 4] = 6 # 由于对称故只表示一半
i, j = np.nonzero(a); w = a[i, j]
edges = list(zip(i, j, w)); pos = networkx.shell_layout(G) # 按顺序排列顶点
G = networkx.Graph(); G.add_weighted_edges_from(edges)
labels = dict(zip(range(5), [str(i+1) for i in range(5)]))
networkx.draw(G, pos, font_size=13, font_weight='bold', labels=labels) # 方法一
# networkx.draw_networkx(G, pos, labels=labels, font_size=13, font_weight='bold') # 方法二
w = networkx.get_edge_attributes(G, 'weight')
networkx.draw_networkx_edge_labels(G, pos, edge_labels=w) # 添加权重
```

7.3 最短路径

最短路径： 求解从一个顶点到另一个顶点的最短路径

7.3.1 Dijkstra 算法 (固定起点到其余各点)

流程：每一轮对**优先队列**最顶端 (最小) 做 DQ 操作，取出优先级最高的节点，寻找相邻顶点，更新下游节点的**最短路径表**，将发生改变的节点同距离加入优先队列。**直至优先队列清空**。

```python
import numpy as np
import networkx as nx

```

```
List = [(0,1,1),(0,2,2),(0,4,7),(0,6,4),(0,7,8),(1,2,2),(1,3,3),(1,7,7),(2,3,1),(2,4,5),(3,4,3)
    ,(3,5,6),(4,5,4),(4,6,3),(5,6,6),(5,7,4),(6,7,2)] # 各边权重
G = nx.Graph(); G.add_weighted_edges_from(List)
p = nx.dijkstra_path(G, source=3, target=7, weight='weight') # 最短路径
d = nx.dijkstra_path_length(G, 3, 7, weight='weight') # 最短距离
```

```
import numpy as np
import networkx as nx

List = [(0,1,1),(0,2,2),(0,4,7),(0,6,4),(0,7,8),(1,2,2),(1,3,3),(1,7,7),(2,3,1),(2,4,5),(3,4,3)
    ,(3,5,6),(4,5,4),(4,6,3),(5,6,6),(5,7,4),(6,7,2)] # 各边权重
G = nx.Graph()
G.add_weighted_edges_from(List)

# p, d = nx.single_source_dijkstra(G, source=3, target=7, weight='weight')
path = nx.single_source_dijkstra_path(G, source=3, weight='weight')
distance = nx.single_source_dijkstra_path_length(G, source=3, weight='weight')
print("shortest path:\n", path)
print("shortest distance :\n", distance)
```

7.3.2 Floyd 算法 (每对顶点间最短路径)

```
List = [(0,1,1),(0,2,2),(0,4,7),(0,6,4),(0,7,8),(1,2,2),(1,3,3),(1,7,7),(2,3,1),(2,4,5),(3,4,3)
    ,(3,5,6),(4,5,4),(4,6,3),(5,6,6),(5,7,4),(6,7,2)] # 各边权重
G = nx.Graph()
G.add_weighted_edges_from(List)
d = nx.shortest_path_length(G, weight='weight'); Ld = dict(d) # 转换为字典输出
print(f"The distance from vertex 0 to vertex 4:\n{Ld[3][7]}")
```

7.4 最小生成树

树： 连通的无圈图；**生成树：** 若图 G 的生成子图 H 是树，则称 H 为 G 的生成树 (可能不唯一)。
最小生成树： 赋权图 G 中，边权之和最小的生成树称为 G 的最小生成树。

7.4.1 Kruskal 算法

流程： 每次将权值最小的边加入生成树，并保正不形成圈，直至生成树中含有 $n-1$ 条边为止。

7.4.2 Prim 算法

流程： 初始化三个集合：顶点集 $P=\{v_1\}$(起始顶点)，边集 $Q=\emptyset$，剩余顶点集 $E=\{v_2,v_3,\cdots,v_n\}=P-V$。每一轮从所有 $p\in P, e\in E$ 形成的边中选取权最小的边，将 e 加入 P 中，将 (p,e) 加入 Q 中，直至 $P=V$。

7.4.3 代码实现networkx.minimum_spanning_tress

```
import numpy as np
import networkx as nx

L = [(1,2,8), (1,3,4), (1,5,2),(2,3,4),(3,5,1),(3,4,2),(4,5,5)]
G = nx.Graph(); G.add_nodes_from(range(1,6)); G.add_weighted_edges_from(L); pos = nx.shell_layout(G) # 形成图
T = nx.minimum_spanning_tree(G); w = nx.get_edge_attributes(T, 'weight')
nx.draw(T, pos, with_labels=True, node_color='r'); nx.draw_networkx_edge_labels(T, pos, edge_labels=w)
```

7.5 22-23 考题——图论

```
import networkx as nx
import numpy as np

List = [(1, 2, 7), (1, 4, 5), (2, 3, 8), (2, 4, 9), (2, 5, 7), (3, 5, 5), (4, 5, 15), (4, 6, 6), (5, 6, 8), (5, 7, 9), (6, 7, 11)]
G = nx.Graph()
G.add_weighted_edges_from(List)
#======================a======================#
path = nx.single_source_dijkstra(G, source = 1, target = 6, weight='weight')
print(f"shortest path: {path[1]}")
#======================b======================#
T = nx.minimum_spanning_tree(G); pos = nx.shell_layout(G)
nx.draw(T, pos=pos)
```

第 8 章 23-24 学年考题

第一题：条件：给定方程组 $Ax = b$ 中的 A, b(应该与 22-23 年一样的)
(a)：写出最小二乘法中的法方程组 (无需求解)
(b)：写出行最简行`.rref()`
(c)：写出最小二乘解`np.linalg.lstsq()`

第二题：给定一重积分式、二重积分式、三重积分式，各自积分区域给定
(a)：将重积分化为累次积分
(b)：分别使用`scipy.integrate.quad()`、`scipy.integrate.dblquad()`、`scipy.integrate.tplquad()`求解

第三题：给定一组数据，在一行中绘制三个子图 (`plt.subplot()`)，分别为：身高直方图 (`plt.hist()`)、体积直方图 (`plt.hist()`)、体积与身高的箱图 (`plt.boxplot()`)

第四题：条件：给定一个规划问题 (为 max 形式的)
(a)：写出规划问题的标准形式
(b)：使用`scipy.optimize.linprog()`求解

第五题：条件：给定一个微分方程组 (类似 2022-2023 学年的第 4 题)
(a)：使用`sympy.dsolve()`求解
(b)：使用`scipy.integrate.odeint()`求解
(c)：分别绘制两种解的图像，三个子图

第六题：条件：给定一个无向图
(a)：写出邻接矩阵
(b)：使用 **prime 算法**求解最小生成树，要求写出过程和最终的树，并能体现出先后顺序
(c)：使用`networkx.minimum_spanning_tree()`求解最小生成树

第七题：条件：给定一个整数规划应用题 (具体为 0-1 规划的成本收益最大化问题)
(a)：写出决策变量及如何施加约束
(b)：描述整个整数规划问题
(c)：使用`cvxpy`求解该整数规划问题