各个数学模型及其Python实现

1.微分方程模型

①如求下列微分方程的特解，只需要改变eq中的表达式和con中的初值即可

$$\frac{d^2y}{dx^2}+2\frac{dy}{dx}+2y=0,$$
$$y(0)=0,\quad y'(0)=1.$$

```
from sympy.abc import x
from sympy import diff, dsolve, simplify, Function
y=Function('y')
eq=diff(y(x),x,2)+2*diff(y(x),x)+2*y(x)  #定义方程
con={y(0): 0, diff(y(x),x).subs(x,0): 1}  #定义初值条件
y=dsolve(eq, ics=con)
print(simplify(y))
# 输出结果为Eq(y(x), exp(-x)*sin(x))，即y(x) = exp(-x)*sin(x)
```

$$\frac{dx_1}{dt}=2x_1-3x_2+3x_3,\quad x_1(0)=1,$$
$$\frac{dx_2}{dt}=4x_1-5x_2+3x_3,\quad x_2(0)=2,$$
$$\frac{dx_3}{dt}=4x_1-4x_2+2x_3,\quad x_3(0)=3.$$

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

②绘制微分方程的曲线（包括符号解和数值解）

使用：plt中的表达式可以根据①的求解进行替换，即可绘制

$$
\begin{aligned}
y_1' &= y_2, & y_1(0) &= 0, \\
y_2' &= -2y_1 - 2y_2, & y_2(0) &= 1.
\end{aligned}
$$

```
from scipy.integrate import odeint
from sympy.abc import t
import numpy as np
import matplotlib.pyplot as plt
def Pfun(y,x):
    y1, y2=y;
    return np.array([y2, -2*y1-2*y2])
x=np.arange(0, 10, 0.1)  #创建时间点
sol1=odeint(Pfun, [0.0, 1.0], x)  #求数值解
plt.rc('font',size=16); plt.rc('font',family='SimHei')
plt.plot(x, sol1[:,0],'r*',label="数值解")
plt.plot(x, np.exp(-x)*np.sin(x), 'g', label="符号解曲线")
plt.legend(); plt.savefig("figure8_5.png"); plt.show()
```

③洛伦兹模型，用于判断初值相差甚小时，解的偏差演化曲线，即用于判断是否处于混沌态

使用方法：控制sol1和sol2中的初值即可

```
from scipy.integrate import odeint
import numpy as np
from mpl_toolkits import mplot3d
import matplotlib.pyplot as plt
def lorenz(w,t):
    sigma=10; rho=28; beta=8/3
    x, y, z=w;
    return np.array([sigma*(y-x), rho*x-y-x*z, x*y-beta*z])
t=np.arange(0, 50, 0.01)  #创建时间点
sol1=odeint(lorenz, [0.0, 1.0, 0.0], t)  #第一个初值问题求解
sol2=odeint(lorenz, [0.0, 1.0001, 0.0], t)  #第二个初值问题求解
plt.rc('font',size=16); plt.rc('text',usetex=True)
ax1=plt.subplot(121,projection='3d')
ax1.plot(sol1[:,0], sol1[:,1], sol1[:,2],'r')
ax1.set_xlabel('$x$'); ax1.set_ylabel('$y$'); ax1.set_zlabel('$z$')
ax2=plt.subplot(122,projection='3d')
ax2.plot(sol1[:,0]-sol2[:,0], sol1[:,1]-sol2[:,1], sol1[:,2]-sol2[:,2],'g')
ax2.set_xlabel('$x$'); ax2.set_ylabel('$y$'); ax2.set_zlabel('$z$')
plt.savefig("figure8_6.png", dpi=500); plt.show()
print("sol1=",sol1, '\n\n', "sol1-sol2=", sol1-sol2)
```

④Logistic模型

适用于自然增长且增长率不断下降直至0的情况

使用方法：在Pdata8_10_1.txt中，奇数行设置时间，偶数行设置人口数量（或称因变量），即可。其中popt的值为r和xm，r为初始的增长率，xm为能容纳的最大人口数

Pdata8_10_1.txt

```
1790    1800    1810    1820    1830    1840    1850    1860
3.9 5.3 7.2 9.6 12.9    17.1    23.2    31.4
1870    1880    1890    1900    1910    1920    1930    1940
38.6    50.2    62.9    76.0    92.0    106.5   123.2   131.7
1950    1960    1970    1980    1990    2000
150.7   179.3   204.0   226.5   251.4   281.4
```

```python
import numpy as np
from scipy.optimize import curve_fit
a=[]; b=[];
with open("Pdata8_10_1.txt") as f:    #打开文件并绑定对象f
    s=f.read().splitlines()  #返回每一行的数据
for i in range(0, len(s),2):  #读入奇数行数据
    d1=s[i].split("\t")
    for j in range(len(d1)):
        if d1[j]!="": a.append(eval(d1[j]))  #把非空的字符串转换为年代数据
for i in range(1, len(s), 2):  #读入偶数行数据
    d2=s[i].split("\t")
    for j in range(len(d2)):
        if d2[j] != "": b.append(eval(d2[j])) #把非空的字符串转换为人口数据
c=np.vstack((a,b))  #构造两行的数组
np.savetxt("Pdata8_10_2.txt", c)  #把数据保存起来供下面使用
x=lambda t, r, xm: xm/(1+(xm/3.9-1)*np.exp(-r*(t-1790)))
bd=((0, 200), (0.1,1000))  #约束两个参数的下界和上界
popt, pcov=curve_fit(x, a[1:], b[1:], bounds=bd)
print(popt); print("2010年的预测值为：", x(2010, *popt))
```

2.综合评价方法

①获得评价矩阵，分别用向量归一化法、比例变换法、极差变换法

使用方法：在Pdata9_1_1.txt里写上原始数据，对于极大值指标和极小值指标有不同的处理方法，格式如下

	最大速度x_1	飞行半径x_2	最大负载x_3	费用x_4	可靠性x_5	灵敏度x_6
A_1	2.0	1500	20000	5500000	0.5	1
A_2	2.5	2700	18000	6500000	0.3	0.5
A_3	1.8	2000	21000	4500000	0.7	0.7
A_4	2.2	1800	20000	5000000	0.5	0.5

Pdata9_1_1.txt

```
2.0 1500    20000   5500000 0.5 1
2.5 2700    18000   6500000 0.3 0.5
1.8 2000    21000   4500000 0.7 0.7
2.2 1800    20000   5000000 0.5 0.5
```

其中费用为极小值指标，位于第3列，因此要进行不同的处理

```python
import numpy as np
import pandas as pd
a=np.loadtxt("Pdata9_1_1.txt",)
R1=a.copy(); R2=a.copy(); R3=a.copy()  #初始化
```

```
#注意R1=a,它们的内存地址一样，R1改变时，a也改变
for j in [0,1,2,4,5]:
    R1[:,j]=R1[:,j]/np.linalg.norm(R1[:,j]) #向量归一化
    R2[:,j]=R1[:,j]/max(R1[:,j])      #比例变换
    R3[:,j]=(R3[:,j]-min(R3[:,j]))/(max(R3[:,j])-min(R3[:,j]));
R1[:,3]=1-R1[:,3]/np.linalg.norm(R1[:,3])
R2[:,3]=min(R2[:,3])/R2[:,3]
R3[:,3]=(max(R3[:,3])-R3[:,3])/(max(R3[:,3])-min(R3[:,3]))
np.savetxt("Pdata9_1_2.txt", R1); #把数据写入文本文件，供下面使用
np.savetxt("Pdata9_1_3.txt", R2); np.savetxt("Pdata9_1_4.txt", R3)
DR1=pd.DataFrame(R1)  #生成DataFrame类型数据
DR2=pd.DataFrame(R2); DR3=pd.DataFrame(R3)
f=pd.ExcelWriter('Pdata9_1_5.xlsx')  #创建文件对象
DR1.to_excel(f,"sheet1")  #把DR1写入Excel文件1号表单中,方便做表
DR2.to_excel(f,"sheet2"); DR3.to_excel(f, "Sheet3"); f.save()
```

②TOPSIS法、灰色关联度评价、熵值法、秩和比法进行综合评价

使用方法：在Pdata9_1_3.txt中放上评价矩阵，最后所计算的值越大，评价度越高

```
import numpy as np
from scipy.stats import rankdata
a=np.loadtxt("Pdata9_1_3.txt")

cplus=a.max(axis=0)   #逐列求最大值
cminus=a.min(axis=0)  #逐列求最小值
print("正理想解=",cplus,"负理想解=",cminus)
d1=np.linalg.norm(a-cplus, axis=1)  #求到正理想解的距离
d2=np.linalg.norm(a-cminus, axis=1) #求到负理想解的距离
print(d1, d2)   #显示到正理想解和负理想解的距离
f1=d2/(d1+d2); print("TOPSIS的评价值为：", f1)

t=cplus-a   #计算参考序列与每个序列的差
mmin=t.min(); mmax=t.max()  #计算最小差和最大差
rho=0.5  #分辨系数
xs=(mmin+rho*mmax)/(t+rho*mmax)  #计算灰色关联系数
f2=xs.mean(axis=1)  #求每一行的均值
print("\n关联系数=", xs,'\n关联度=',f2)  #显示灰色关联系数和灰色关联度

[n, m]=a.shape
cs=a.sum(axis=0)  #逐列求和
P=1/cs*a   #求特征比重矩阵
e=-(P*np.log(P)).sum(axis=0)/np.log(n)  #计算熵值
g=1-e   #计算差异系数
w = g / sum(g)  #计算权重
F = P @ w       #计算各对象的评价值
print("\nP={}\n,e={}\n,g={}\n,w={}\nF={}".format(P,e,g,w,F))

R=[rankdata(a[:,i]) for i in np.arange(6)]  #求每一列的秩
R=np.array(R).T   #构造秩矩阵
print("\n秩矩阵为：\n",R)
RSR=R.mean(axis=1)/n; print("RSR=", RSR)
```

③层次分析法

使用方法：给a赋值两两比较表，赋值B1~5，也是通过两两比较得出，最后得出的k值越大越好。需要注意的是，第一个表是各个因素的比较，第二个表的各个景点对同一因素的比较

具体的做法是通过相互比较, 假设各准则对目标的权重和各方案对每一准则的权重. 首先在准则层对目标层进行赋权, 认为费用应占最大的比重 (因为是学生), 其次是景色 (目的主要是旅游), 再者是旅途, 至于吃住对年轻人来说不太重要. 表 9.7 是采用两两比较判断法得到的数据.

表 9.7　旅游决策准则层对目标层的两两比较表

项目	景色	费用	饮食	居住	旅途
景色	1	1/2	5	5	3
费用	2	1	7	7	5
饮食	1/5	1/7	1	1/2	1/3
居住	1/5	1/7	2	1	1/2
旅途	1/3	1/5	3	2	1

CR小于0.1时，认为判断矩阵具有满意的一致性

```
from scipy.sparse.linalg import eigs
from numpy import array, hstack
a=array([[1,1/2,5,5,3],[2,1,7,7,5],[1/5,1/7,1,1/2,1/3],
        [1/5,1/7,2,1,1/2],[1/3,1/5,3,2,1]])
L,V=eigs(a,1);
CR=(L-5)/4/1.12  #计算矩阵A的一致性比率
W=V/sum(V); print("最大特征值为：",L)
print("最大特征值对应的特征向量W=\n",W)
print("CR=",CR)
B1=array([[1,1/3,1/2],[3,1,1/2],[2,2,1]])
L1,P1=eigs(B1,1); P1=P1/sum(P1)
print("P1=",P1)
B2=array([[1,3,2],[1/3,1,2],[1/2,1/2,1]])
t2,P2=eigs(B2,1); P2=P2/sum(P2)
print("P2=",P2)
B3=array([[1,4,3],[1/4,1,2],[1/3,1/2,1]])
t3, P3=eigs(B3,1); P3=P3/sum(P3)
print("P3=",P3)
B4=array([[1,3,2],[1/3,1,2],[1/2,1/2,1]])
t4, P4=eigs(B4,1); P4=P4/sum(P4)
print("P4=", P4)
B5=array([[1,2,3],[1/2,1,1/2],[1/3,2,1]])
t5, P5=eigs(B5,1); P5=P5/sum(P5)
print("P5=",P5)
K=hstack([P1,P2,P3,P4,P5])@W  #矩阵乘法
print("K=",K)
```

3.图论模型

①绘制邻接矩阵无向图

使用方法：改变a的赋值即可

$$A=\begin{bmatrix} 0 & 9 & 2 & 4 & 7 \\ 9 & 0 & 3 & 4 & 0 \\ 2 & 3 & 0 & 8 & 4 \\ 4 & 4 & 8 & 0 & 6 \\ 7 & 0 & 4 & 6 & 0 \end{bmatrix}.$$

```
import numpy as np
import networkx as nx
import pylab as plt
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]; #输入邻接矩阵的上三角元素
a[3,4]=6; print(a); np.savetxt("Pdata10_2.txt",a) #保存邻接矩阵供以后使用
i,j=np.nonzero(a)  #提取顶点的编号
w=a[i,j]  #提出a中的非零元素
edges=list(zip(i,j,w))
G=nx.Graph()
G.add_weighted_edges_from(edges)
key=range(5); s=[str(i+1) for i in range(5)]
s=dict(zip(key,s))  #构造用于顶点标注的字符字典
plt.rc('font',size=18)
plt.subplot(121); nx.draw(G,font_weight='bold',labels=s)
plt.subplot(122); pos=nx.shell_layout(G)  #布局设置
nx.draw_networkx(G,pos,node_size=260,labels=s)
w = nx.get_edge_attributes(G,'weight')
nx.draw_networkx_edge_labels(G,pos,font_size=12,edge_labels=w) #标注权重
plt.savefig("figure10_2.png", dpi=500); plt.show()
```

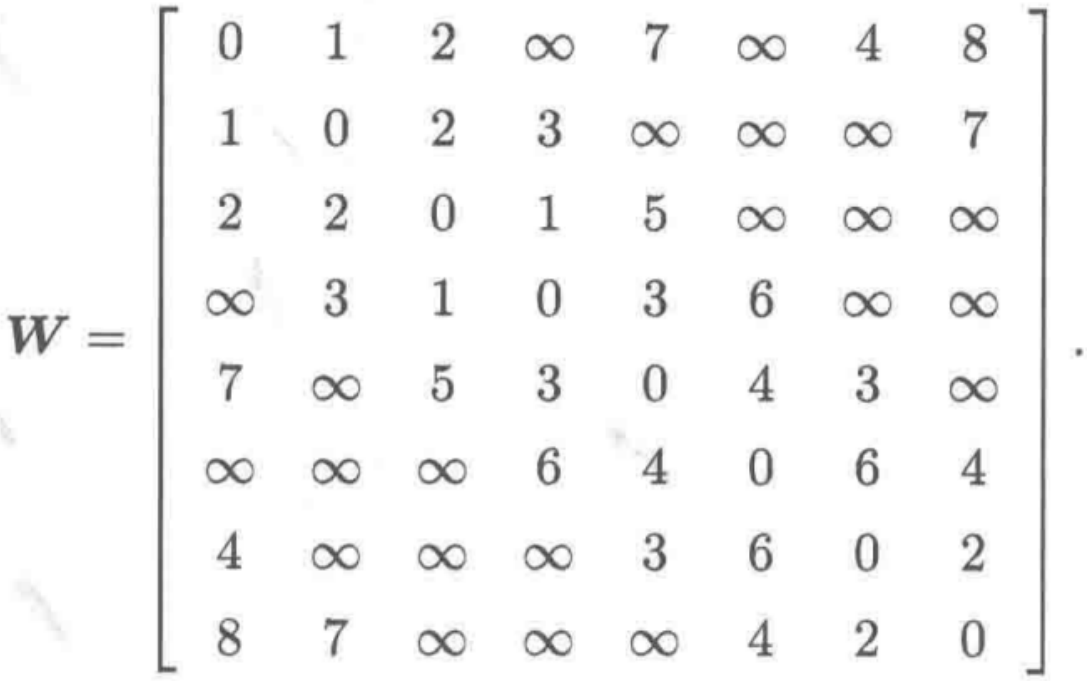
②绘制有向图

使用方法：在List中输入边集即可

```
import numpy as np
import networkx as nx
import pylab as plt
G=nx.DiGraph()
List=[(1,2),(1,3),(2,3),(3,2),(3,5),(4,2),(4,6),
      (5,2),(5,4),(5,6),(6,5)]
G.add_nodes_from(range(1,7))
G.add_edges_from(List)
plt.rc('font',size=16)
pos=nx.shell_layout(G)
nx.draw(G,pos,with_labels=True, font_weight='bold',node_color='r')
plt.savefig("figure10_3.png", dpi=500); plt.show()
```

③求顶点到其他各点的最短路径

使用方法：为a赋值邻接矩阵，在d中改变顶点号（从0开始）即可



$$W=\begin{bmatrix} 0 & 1 & 2 & \infty & 7 & \infty & 4 & 8 \\ 1 & 0 & 2 & 3 & \infty & \infty & \infty & 7 \\ 2 & 2 & 0 & 1 & 5 & \infty & \infty & \infty \\ \infty & 3 & 1 & 0 & 3 & 6 & \infty & \infty \\ 7 & \infty & 5 & 3 & 0 & 4 & 3 & \infty \\ \infty & \infty & \infty & 6 & 4 & 0 & 6 & 4 \\ 4 & \infty & \infty & \infty & 3 & 6 & 0 & 2 \\ 8 & 7 & \infty & \infty & \infty & 4 & 2 & 0 \end{bmatrix}.$$

```python
import numpy as np
inf=np.inf
def Dijkstra_all_minpath( matr,start): #matr为邻接矩阵的数组，start表示起点
    n=len( matr) #该图的节点数
    dis=[]; temp=[]
    dis.extend(matr[start])  #添加数组matr的start行元素
    temp.extend(matr[start]) #添加矩阵matr的start行元素
    temp[start] = inf  #临时数组会把处理过的节点的值变成inf
    visited=[start]  #start已处理
    parent=[start]*n   #用于画路径，记录此路径中该节点的父节点
    while len(visited)<n:
        i= temp.index(min(temp)) #找最小权值的节点的坐标
        temp[i]=inf
        for j in range(n):
            if j not in visited:
                if (dis[i]+ matr[i][j])<dis[j]:
                    dis[j] = temp[j] =dis[i]+ matr[i][j]
                    parent[j]=i  #说明父节点是i
        visited.append(i)  #该索引已经处理了
        path=[]  #用于画路径
        path.append(str(i))
        k=i
        while(parent[k]!=start):  #找该节点的父节点添加到path，直到父节点是start
            path.append(str(parent[k]))
            k=parent[k]
        path.append(str(start))
        path.reverse()   #path反序产生路径
        print(str(i)+':','->'.join(path))  #打印路径
    return dis
a=[[0,1,2,inf,7,inf,4,8],[1,0,2,3,inf,inf,inf,7],
  [2,2,0,1,5,inf,inf,inf],[inf,3,1,0,3,6,inf,inf],
```

```
  [7,inf,5,3,0,4,3,inf],[inf,inf,inf,6,4,0,6,4],
  [4,inf,inf,inf,3,6,0,2],[8,7,inf,inf,inf,4,2,0]]
d=Dijkstra_all_minpath(a,3)
print("v3到所有顶点的最短距离为：",d)
```

④求最短路径（更简洁的方法）

使用方法：在List中赋值边集和权重

```
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)]
G=nx.Graph()
G.add_weighted_edges_from(List)
A=nx.to_numpy_matrix(G, nodelist=range(8))  #导出邻接矩阵
np.savetxt('Pdata10_6.txt',A)
p=nx.dijkstra_path(G, source=3, target=7, weight='weight')  #求最短路径；
d=nx.dijkstra_path_length(G, 3, 7, weight='weight') #求最短距离
print("最短路径为：",p,"；最短距离为：",d)
```

⑤利用最短路径解决费用问题（目的是使路径最小）

如对于计算权值w

年份	1	2	3	4
年初购置价格/万元	25	26	28	31
当年维护费用/万元	10	14	18	26
年末剩余净值/万元	20	16	13	11

$$w_{ij}=p_i+\sum_{k=1}^{j-i}a_k-r_{j-i},$$

得到邻接矩阵

$$\boldsymbol{W}=\begin{bmatrix}0 & 15 & 33 & 54 & 82\\ \infty & 0 & 16 & 34 & 55\\ \infty & \infty & 0 & 18 & 36\\ \infty & \infty & \infty & 0 & 21\\ \infty & \infty & \infty & \infty & 0\end{bmatrix}.$$

```python
import numpy as np
import networkx as nx
import pylab as plt
p=[25,26,28,31]; a=[10,14,18,26]; r=[20,16,13,11];
b=np.zeros((5,5)); #邻接矩阵（非数学上的邻接矩阵）初始化
for i in range(5):
    for j in range(i+1,5):
        b[i,j]=p[i]+np.sum(a[0:j-i])-r[j-i-1];
print(b)
G=nx.DiGraph(b)
p=nx.dijkstra_path(G, source=0, target=4, weight='weight')  #求最短路径;
print("最短路径为:",np.array(p)+1)  #python下标从0开始
d=nx.dijkstra_path_length(G, 0, 4, weight='weight') #求最短距离
print("所求的费用最小值为: ",d)
s=dict(zip(range(5),range(1,6))) #构造用于顶点标注的标号字典
plt.rc('font',size=16)
pos=nx.shell_layout(G)  #设置布局
w=nx.get_edge_attributes(G,'weight')
nx.draw(G,pos,font_weight='bold',labels=s,node_color='r')
nx.draw_networkx_edge_labels(G,pos,edge_labels=w)
path_edges=list(zip(p,p[1:]))
nx.draw_networkx_edges(G,pos,edgelist=path_edges,
            edge_color='r',width=3)
plt.savefig("figure10_9.png",pdi=500); plt.show()
```

⑥处理给定一组点，选取一点作为中转站，使得到各个点路径总和最短的问题

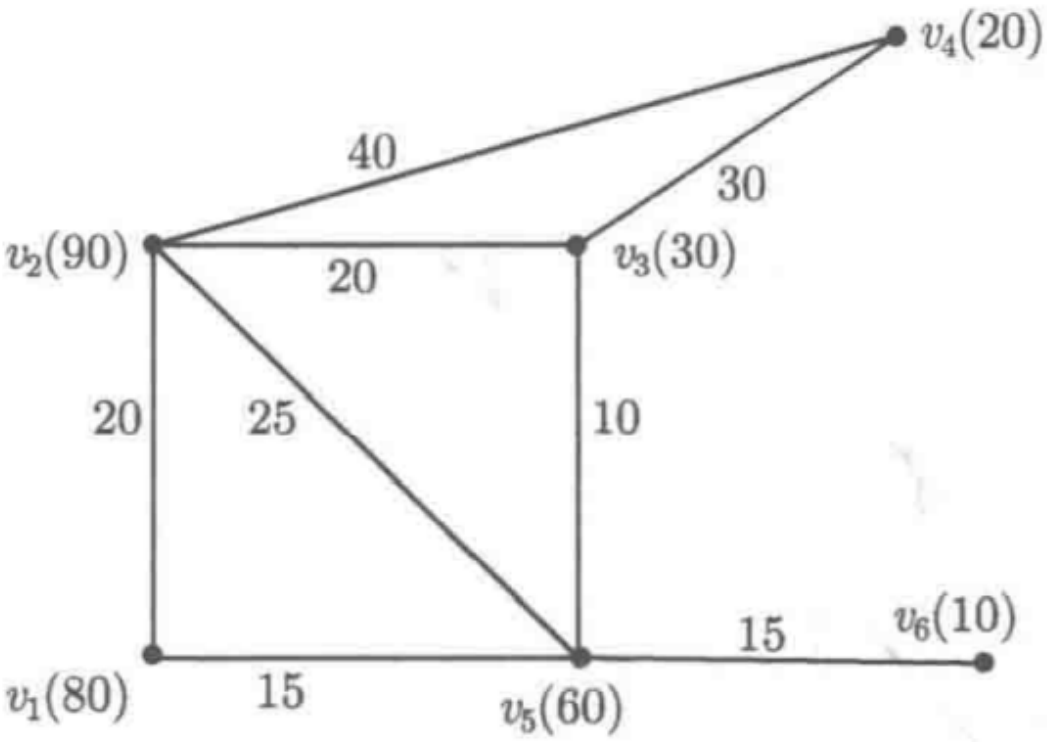


表 10.2　各顶点对之间的最短距离和总运力计算数据

产矿点	v_1	v_2	v_3	v_4	v_5	v_6	总运力 m_i
v_1	0	20	25	55	15	30	4850
v_2	20	0	20	40	25	40	4900
v_3	25	20	0	30	10	25	5250
v_4	55	40	30	0	40	55	11850
v_5	15	25	10	40	0	15	4700
v_6	30	40	25	55	15	0	8750

```python
import numpy as np
import networkx as nx
List=[(1,2,20),(1,5,15),(2,3,20),(2,4,40),
      (2,5,25),(3,4,30),(3,5,10),(5,6,15)]
G=nx.Graph()
G.add_nodes_from(range(1,7))
G.add_weighted_edges_from(List)
c=dict(nx.shortest_path_length(G,weight='weight'))
d=np.zeros((6,6))
for i in range(1,7):
    for j in range(1,7): d[i-1,j-1]=c[i][j]
print(d)
q=np.array([80,90,30,20,60,10])
m=d@q  #计算运力，这里使用矩阵乘法
mm=m.min()  #求运力的最小值
ind=np.where(m==mm)[0]+1  #python下标从0开始，np.where返回值为元组
print("运力m=",m,'\n最小运力mm=',mm,"\n选矿厂的设置位置为：",ind)
```

⑦求最小生成树

```python
import numpy as np
import networkx as nx
import pylab as plt
L=[(1,2,8),(1,3,4),(1,5,2),(2,3,4),(3,4,2),(3,5,1),(4,5,5)]
b=nx.Graph()
b.add_nodes_from(range(1,6))
b.add_weighted_edges_from(L)
T=nx.minimum_spanning_tree(b)  #返回可迭代对象
w=nx.get_edge_attributes(T,'weight') #提取字典数据
TL=sum(w.values())  #计算最小生成树的长度
print("最小生成树为:",w)
print("最小生成树的长度为：",TL)
pos=nx.shell_layout(b)
nx.draw(T,pos,node_size=280,with_labels=True,node_color='r')
nx.draw_networkx_edge_labels(T,pos,edge_labels=w)
plt.show()
```

⑧最小生成树处理给定一组点，选取一点作为中转站，使得到各个点路径总和最短的问题

使用方法：在xlsx中给出距离即可

表 10.4　各油井间距离　　(单位：n mile)

	2	3	4	5	6	7	8
1	1.3	2.1	0.9	0.7	1.8	2.0	1.5
2		0.9	1.8	1.2	2.6	2.3	1.1
3			2.6	1.7	2.5	1.9	1.0
4				0.7	1.6	1.5	0.9
5					0.9	1.1	0.8
6						0.6	1.0
7							0.5

Pdata10_14.xlsx

	A	B	C	D	E	F	G
1	1.30	2.10	0.90	0.70	1.80	2	1.50
2		0.90	1.80	1.20	2.60	2.30	1.10
3			2.60	1.70	2.50	1.90	1
4				0.70	1.60	1.50	0.90
5					0.90	1.10	0.80
6						0.60	1
7							0.50

```python
import numpy as np
import networkx as nx
import pandas as pd
import pylab as plt
a=pd.read_excel("Pdata10_14.xlsx",header=None)
b=a.values; b[np.isnan(b)]=0
c=np.zeros((8,8))  #邻接矩阵初始化
c[0:7,1:8]=b  #构造图的邻接矩阵
G=nx.Graph(c)
T=nx.minimum_spanning_tree(G)  #返回可迭代对象
d=nx.to_numpy_matrix(T)  #返回最小生成树的邻接矩阵
print("邻接矩阵c=\n",d)
W=d.sum()/2+5  #求油管长度
print("油管长度W=",W)
s=dict(zip(range(8),range(1,9))) #构造用于顶点标注的标号字典
plt.rc('font',size=16); pos=nx.shell_layout(G)
nx.draw(T,pos,node_size=280,labels=s,node_color='r')
w=nx.get_edge_attributes(T,'weight')
nx.draw_networkx_edge_labels(T,pos,edge_labels=w)
plt.savefig('figure10_14.png'); plt.show()
```

⑨解决匹配问题

使用方法：给出5个人做5个工作的效益

$$\begin{bmatrix} 3 & 5 & 5 & 4 & 1 \\ 2 & 2 & 0 & 2 & 2 \\ 2 & 4 & 4 & 1 & 0 \\ 0 & 2 & 2 & 1 & 0 \\ 1 & 2 & 1 & 3 & 3 \end{bmatrix}$$

```
import numpy as np
import networkx as nx
from networkx.algorithms.matching import max_weight_matching
a=np.array([[3,5,5,4,1],[2,2,0,2,2],[2,4,4,1,0],
            [0,2,2,1,0],[1,2,1,3,3]])
b=np.zeros((10,10)); b[0:5,5:]=a; G=nx.Graph(b)
s0=max_weight_matching(G)  #返回值为（人员，工作）的集合
s=[sorted(w) for w in s0]
L1=[x[0] for x in s]; L1=np.array(L1)+1  #人员编号
L2=[x[1] for x in s]; L2=np.array(L2)-4  #工作编号
c=a[L1-1,L2-1]  #提取对应的效益
d=c.sum()  #计算总的效益
print("工作分配对应关系为：\n人员编号：",L1)
print("工作编号：", L2); print("总的效益为：",d)
```

4.多元分析

①处理分类问题（利用马氏距离和欧氏距离）

使用方法：x0给出两种类型的原始数据，x给出预测数据，g给出类型

Af: (1.24, 1.27), (1.36, 1.74), (1.38, 1.64), (1.38, 1.82), (1.38, 1.90), (1.40, 1.70), (1.48, 1.82), (1.54, 1.82), (1.56, 2.08);

Apf: (1.14, 1.78), (1.18, 1.96), (1.20, 1.86), (1.26, 2.00), (1.28, 2.00), (1.30, 1.96).

```
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
x0=np.array([[1.24,1.27], [1.36,1.74], [1.38,1.64], [1.38,1.82],
[1.38,1.90], [1.40,1.70],
    [1.48,1.82], [1.54,1.82], [1.56,2.08], [1.14,1.78], [1.18,1.96],
[1.20,1.86],
    [1.26,2.00], [1.28,2.00], [1.30,1.96]])   #输入已知样本数据
x=np.array([[1.24,1.80], [1.28,1.84], [1.40,2.04]])  #输入待判样本点数据
g=np.hstack([np.ones(9),2*np.ones(6)])  #g为已知样本数据的类别标号
v=np.cov(x0.T)  #计算协方差
knn=KNeighborsClassifier(2,metric='mahalanobis',metric_params={'V': v}) #马
氏距离分类
knn.fit(x0,g); pre=knn.predict(x); print("马氏距离分类结果：",pre)
print("马氏距离已知样本的误判率为：",1-knn.score(x0,g))
```

```
knn2=KNeighborsClassifier(2)  #欧氏距离分类
knn2.fit(x0,g); pre2=knn2.predict(x); print("欧氏距离分类结果：",pre2)
print("欧氏距离已知样本的误判率为：",1-knn2.score(x0,g))
```

②马氏距离分类，处理多个数据的Excel

使用方法：给出Excel数据，控制程序中的下标

序号	x_1	x_2	x_3	x_4	x_5	类型
1	8.11	261.01	13.23	5.46	7.36	1
2	9.36	185.39	9.02	5.66	5.99	1
3	9.85	249.58	15.61	6.06	6.11	1
4	2.55	137.13	9.21	6.11	4.35	1
5	6.01	231.34	14.27	5.21	8.79	1
6	9.46	231.38	13.03	4.88	8.53	1
7	4.11	260.25	14.72	5.36	10.02	1
8	8.90	259.51	14.16	4.91	9.79	1
9	7.71	273.84	16.01	5.15	8.79	1
10	7.51	303.59	19.14	5.7	8.53	1
11	6.8	308.9	15.11	5.52	8.49	2
12	8.68	258.69	14.02	4.79	7.16	2
13	5.67	355.54	15.13	4.97	9.43	2
14	8.1	476.69	7.38	5.32	11.32	2
15	3.71	316.12	17.12	6.04	8.17	2

Pdata11_2.xlsx

	A	B	C	D	E	F	G
10	10	7.51	303.59	19.14	5.70	8.53	1
11	11	6.80	308.90	15.11	5.52	8.49	2
12	12	8.68	258.69	14.02	4.79	7.16	2
13	13	5.67	355.54	15.13	4.97	9.43	2
14	14	8.10	476.69	7.38	5.32	11.32	2
15	15	3.71	316.12	17.12	6.04	8.17	2
16	16	5.37	274.57	16.75	4.98	9.67	2
17	17	5.22	330.34	18.19	4.96	9.61	3
18	18	4.71	331.47	21.26	4.30	13.72	3
19	19	4.71	352.50	20.79	5.07	11	3
20	20	3.36	347.31	17.90	4.65	11.19	3
21	21	8.06	231.03	14.41	5.72	6.15	待判
22	22	9.89	409.42	19.47	5.19	10.49	待判

```
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
a=pd.read_excel("Pdata11_2.xlsx",header=None)
b=a.values
x0=b[:-2,1:-1].astype(float)  #提取已知样本点的观测值
y0=b[:-2,-1].astype(int)
x=b[-2:,1:-1]  #提取待判样本点的观察值
v=np.cov(x0.T)  #计算协方差
knn=KNeighborsClassifier(3,metric='mahalanobis',metric_params={'V': v}) #马氏距离分类
knn.fit(x0,y0); pre=knn.predict(x); print("分类结果：",pre)
print("已知样本的误判率为：",1-knn.score(x0,y0))
```

③Fisher判别法

使用方法：x0给出两种类型的原始数据，x给出预测数据，y0给出类型

Af: (1.24, 1.27), (1.36, 1.74), (1.38, 1.64), (1.38, 1.82), (1.38, 1.90), (1.40, 1.70), (1.48, 1.82), (1.54, 1.82), (1.56, 2.08);

Apf: (1.14, 1.78), (1.18, 1.96), (1.20, 1.86), (1.26, 2.00), (1.28, 2.00), (1.30, 1.96).

```
import numpy as np
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
x0=np.array([[1.24,1.27], [1.36,1.74], [1.38,1.64], [1.38,1.82], [1.38,1.90], [1.40,1.70],
    [1.48,1.82], [1.54,1.82], [1.56,2.08], [1.14,1.78], [1.18,1.96], [1.20,1.86],
    [1.26,2.00], [1.28,2.00], [1.30,1.96]])   #输入已知样本数据
x=np.array([[1.24,1.80], [1.28,1.84], [1.40,2.04]])  #输入待判样本点数据
y0=np.hstack([np.ones(9),2*np.ones(6)])  #y0为已知样本数据的类别
clf = LDA()
clf.fit(x0, y0)
print("判别结果为：",clf.predict(x))
print("已知样本的误判率为：",1-clf.score(x0,y0))
```

④Fisher判别法处理Excel

```
import numpy as np
import pandas as pd
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
a=pd.read_excel("Pdata11_2.xlsx",header=None)
b=a.values
x0=b[:-2,1:-1].astype(float)  #提取已知样本点的观测值
y0=b[:-2,-1].astype(int)
x=b[-2:,1:-1]  #提取待判样本点的观察值
clf = LDA()
clf.fit(x0, y0)
print("判别结果为：",clf.predict(x))
print("已知样本的误判率为：",1-clf.score(x0,y0))
```

⑤贝叶斯判别法

```
import numpy as np
from sklearn.naive_bayes import GaussianNB
x0=np.array([[1.24,1.27], [1.36,1.74], [1.38,1.64], [1.38,1.82], [1.38,1.90], [1.40,1.70],
    [1.48,1.82], [1.54,1.82], [1.56,2.08], [1.14,1.78], [1.18,1.96], [1.20,1.86],
    [1.26,2.00], [1.28,2.00], [1.30,1.96]])   #输入已知样本数据
x=np.array([[1.24,1.80], [1.28,1.84], [1.40,2.04]])  #输入待判样本点数据
y0=np.hstack([np.ones(9),2*np.ones(6)])  #y0为已知样本数据的类别
clf = GaussianNB()
clf.fit(x0, y0)
print("判别结果为：",clf.predict(x))
print("已知样本的误判率为：",1-clf.score(x0,y0))
```

⑥主成分分析

序号	身高 x_1/cm	胸围 x_2/cm	体重 x_3/kg	序号	身高 x_1/cm	胸围 x_2/cm	体重 x_3/kg
1	149.5	69.5	38.5	6	156.1	74.5	45.5
2	162.5	77	55.5	7	172.0	76.5	51.0
3	162.7	78.5	50.8	8	173.2	81.5	59.5
4	162.2	87.5	65.5	9	159.5	74.5	43.5
5	156.5	74.5	49.0	10	157.7	79	53.5

Pdata11_7.txt

```
1   1   149.5   69.5    38.5    6   156.1   74.5    45.5
2   2   162.5   77  55.5    7   172.0   76.5    51.0
3   3   162.7   78.5    50.8    8   173.2   81.5    59.5
4   4   162.2   87.5    65.5    9   159.5   74.5    43.5
5   5   156.5   74.5    49.0    10  157.7   79  53.5
```

```python
import numpy as np
from sklearn.decomposition import PCA
a=np.loadtxt("Pdata11_7.txt")
b=np.r_[a[:,1:4],a[:,-3:]]  #构造数据矩阵
md=PCA().fit(b)  #构造并训练模型
print("特征值为：",md.explained_variance_)
print("各主成分的贡献率：",md.explained_variance_ratio_)
print("奇异值为：",md.singular_values_)
print("各主成分的系数：\n",md.components_)  #每行是一个主成分
"""下面直接计算特征值和特征向量，和库函数进行对比"""
cf=np.cov(b.T)  #计算协方差阵
c,d=np.linalg.eig(cf) #求特征值和特征向量
print("特征值为：",c)
print("特征向量为：\n",d)
print("各主成分的贡献率为：",c/np.sum(c))
```

⑦利用主成分分析解决评价问题

表 11.4　安徽省各市大中型工业企业主要经济指标　　(单位：亿元)

地区	x_1	x_2	x_3	x_4	x_5	x_6
合肥市	1932.27	1900.53	653.83	570.95	1810.70	119.53
淮北市	367.05	366.08	186.16	252.07	395.43	32.82
亳州市	86.89	85.38	40.85	51.71	83.26	8.95
宿州市	154.27	147.07	30.68	57.96	146.30	−1.27
蚌埠市	197.21	193.28	104.56	90.15	182.60	7.85
阜阳市	244.17	231.55	56.37	121.96	224.04	26.49
淮南市	497.74	483.69	206.80	501.37	496.59	27.76
滁州市	308.91	296.99	118.65	76.90	277.42	19.32
六安市	191.77	189.05	70.19	62.31	191.98	23.08
马鞍山市	905.32	894.61	351.52	502.99	1048.02	53.88
巢湖市	254.99	242.38	106.66	75.48	234.76	19.65
芜湖市	867.07	852.34	418.82	217.76	806.94	37.01
宣城市	219.36	207.07	82.58	54.74	192.74	11.02
铜陵市	570.33	563.33	224.23	190.77	697.91	20.61
池州市	59.11	57.32	16.97	40.33	56.56	6.03
安庆市	430.58	426.25	103.08	147.05	442.04	0.79
黄山市	65.03	64.36	28.38	8.58	60.48	2.88

Pdata11_8.txt

```
1   1932.27 1900.53 653.83  570.95  1810.70 119.53
2   367.05  366.08  186.16  252.07  395.43  32.82
3   86.89   85.38   40.85   51.71   83.26   8.95
4   154.27  147.07  30.68   57.96   146.30  -1.27
5   197.21  193.28  104.56  90.15   182.60  7.85
6   244.17  231.55  56.37   121.96  224.04  26.49
7   497.74  483.69  206.80  501.37  496.59  27.76
8   308.91  296.99  118.65  76.90   277.42  19.32
9   191.77  189.05  70.19   62.31   191.98  23.08
10  905.32  894.61  351.52  502.99  1048.02 53.88
11  254.99  242.38  106.66  75.48   234.76  19.65
12  867.07  852.34  418.82  217.76  806.94  37.01
13  219.36  207.07  82.58   54.74   192.74  11.02
14  570.33  563.33  224.23  190.77  697.91  20.61
15  59.11   57.32   16.97   40.33   56.56   6.03
16  430.58  426.25  103.08  147.05  442.04  0.79
17  65.03   64.36   28.38   8.58    60.48   2.88
```

计算相关系数矩阵

$$R=\begin{bmatrix} 1.0000 & 1.0000 & 0.9754 & 0.8231 & 0.9914 & 0.9375 \\ 1.0000 & 1.0000 & 0.9758 & 0.8236 & 0.9920 & 0.9369 \\ 0.9754 & 0.9758 & 1.0000 & 0.8245 & 0.9712 & 0.9127 \\ 0.8231 & 0.8236 & 0.8245 & 1.0000 & 0.8502 & 0.8020 \\ 0.9914 & 0.9920 & 0.9712 & 0.8502 & 1.0000 & 0.9212 \\ 0.9375 & 0.9369 & 0.9127 & 0.8020 & 0.9212 & 1.0000 \end{bmatrix},$$

删除完全线性相关的指标

```python
import numpy as np
from scipy.stats import zscore
a=np.loadtxt("Pdata11_8.txt")
print("相关系数阵为：\n",np.corrcoef(a.T))
b=np.delete(a,0,axis=1) #删除第1列数据
c=zscore(b); r=np.corrcoef(c.T) #数据标准化并计算相关系数阵
d,e=np.linalg.eig(r) #求特征值和特征向量
rate=d/d.sum()  #计算各主成分的贡献率
print("特征值为：",d)
print("特征向量为：\n",e)
print("各主成分的贡献率为：",rate)
k=1; #提出主成分的个数
F=e[:,:k]; score_mat=c.dot(F) #计算主成分得分矩阵
score1=score_mat.dot(rate[0:k])  #计算各评价对象的得分
score2=-score1  #通过观测，调整得分的正负号
print("各评价对象的得分为：",score2)
index=score1.argsort()+1   #排序后的每个元素在原数组中的位置
print("从高到低各个城市的编号排序为：",index)
```

⑧聚类分析

表 11.11　7 个矽卡岩体数据

	1	2	3	4	5	6	7
Cu	2.9909	3.2044	2.8392	2.5315	2.5897	2.9600	3.1184
W	0.3111	0.5348	0.5696	0.4528	0.3010	3.0480	2.8395
Mo	0.5324	0.7718	0.7614	0.4893	0.2735	1.4997	1.9850

Pdata11_11.txt

```
1    2.9909  3.2044  2.8392  2.5315  2.5897  2.9600  3.1184
2    0.3111  0.5348  0.5696  0.4528  0.3010  3.0480  2.8395
3    0.5324  0.7718  0.7614  0.4893  0.2735  1.4997  1.9850
```

```
import numpy as np
from sklearn import preprocessing as pp
import scipy.cluster.hierarchy as sch
import matplotlib.pyplot as plt
a=np.loadtxt("Pdata11_11.txt")
b=pp.minmax_scale(a.T)   #数据规格化
d = sch.distance.pdist(b)  #求对象之间的两两距离向量
dd = sch.distance.squareform(d)  #转换为矩阵格式
z=sch.linkage(d); print(z) #进行聚类并显示
s=[str(i+1) for i in range(7)]; plt.rc('font',size=16)
sch.dendrogram(z,labels=s); plt.show()  #画聚类图
```

5.回归分析

①根据数据x1，x2，y确定线性回归模型

Pdata12_1.txt

1	7	26	78.5
2	1	29	74.3
3	11	56	104.3
4	11	31	87.6
5	7	52	95.9
6	11	55	109.2
7	3	71	102.7
8	1	31	72.5
9	2	54	93.1
10	21	47	115.9
11	1	40	83.8
12	11	66	113.3
13	10	68	109.4

```
import numpy as np
from sklearn.linear_model import LinearRegression
a=np.loadtxt("Pdata12_1.txt")   #加载表中x1,x2,y的13行3列数据
md=LinearRegression().fit(a[:,:2],a[:,2])    #构建并拟合模型
y=md.predict(a[:,:2])       #求预测值
b0=md.intercept_; b12=md.coef_   #输出回归系数
R2=md.score(a[:,:2],a[:,2])      #计算R^2
print("b0=%.4f\nb12=%.4f%10.4f"%(b0,b12[0],b12[1]))
print("拟合优度R^2=%.4f"%R2)
```

②岭回归

表 12.2　1949~1959 年法国进口总额与相关变量的数据

年份	x_1	x_2	x_3	y	年份	x_1	x_2	x_3	y
1949	149.3	4.2	108.1	15.9	1955	202.1	2.1	146.0	22.7
1950	171.5	4.1	114.8	16.4	1956	212.4	5.6	154.1	26.5
1951	175.5	3.1	123.2	19.0	1957	226.1	5.0	162.3	28.1
1952	180.8	3.1	126.9	19.1	1958	231.9	5.1	164.3	27.6
1953	190.7	1.1	132.1	18.8	1959	239.0	0.7	167.6	26.3
1954	202.1	2.2	137.7	20.4					

Pdata12_3.txt

```
 1   149.3   4.2 108.1   15.9
 2   171.5   4.1 114.8   16.4
 3   175.5   3.1 123.2   19.0
 4   180.8   3.1 126.9   19.1
 5   190.7   1.1 132.1   18.8
 6   202.1   2.2 137.7   20.4
 7   202.1   2.1 146.0   22.7
 8   212.4   5.6 154.1   26.5
 9   226.1   5.0 162.3   28.1
10   231.9   5.1 164.3   27.6
11   239.0   0.7 167.6   26.3
```

```python
import numpy as np; import matplotlib.pyplot as plt
from sklearn.linear_model import Ridge, RidgeCV
from scipy.stats import zscore
#plt.rc('text', usetex=True)  #没装LaTeX宏包把该句注释
a=np.loadtxt("Pdata12_3.txt")
n=a.shape[1]-1  #自变量的总个数
aa=zscore(a)  #数据标准化
x=aa[:,:n]; y=aa[:,n]  #提出自变量和因变量观测值矩阵
b=[]  #用于存储回归系数的空列表
kk=np.logspace(-4,0,100)  #循环迭代的不同k值
for k in kk:
    md=Ridge(alpha=k).fit(x,y)
    b.append(md.coef_)
st=['s-r','*-k','p-b']  #下面画图的控制字符串
for i in range(3): plt.plot(kk,np.array(b)[:,i],st[i]);
plt.legend(['$x_1$','$x_2$','$x_3$'],fontsize=15); plt.show()
mdcv=RidgeCV(alphas=np.logspace(-4,0,100)).fit(x,y);
print("最优alpha=",mdcv.alpha_)
```

```
#md0=Ridge(mdcv.alpha_).fit(x,y)  #构建并拟合模型
md0=Ridge(0.4).fit(x,y)  #构建并拟合模型
cs0=md0.coef_  #提出标准化数据的回归系数b1,b2,b3
print("标准化数据的所有回归系数为：",cs0)
mu=np.mean(a,axis=0); s=np.std(a,axis=0,ddof=1) #计算所有指标的均值和标准差
params=[mu[-1]-s[-1]*sum(cs0*mu[:-1]/s[:-1]),s[-1]*cs0/s[:-1]]
print("原数据的回归系数为：",params)
print("拟合优度：",md0.score(x,y))
```

③LASSO回归

```
import numpy as np; import matplotlib.pyplot as plt
from sklearn.linear_model import Lasso, LassoCV
from scipy.stats import zscore
plt.rc('font',size=16)
plt.rc('text', usetex=True)  #没装LaTeX宏包把该句注释
a=np.loadtxt("Pdata12_3.txt")
n=a.shape[1]-1  #自变量的总个数
aa=zscore(a)  #数据标准化
x=aa[:,:n]; y=aa[:,n]  #提出自变量和因变量观测值矩阵
b=[]  #用于存储回归系数的空列表
kk=np.logspace(-4,0,100)  #循环迭代的不同k值
for k in kk:
    md=Lasso(alpha=k).fit(x,y)
    b.append(md.coef_)
st=['s-r','*-k','p-b']  #下面画图的控制字符串
for i in range(3): plt.plot(kk,np.array(b)[:,i],st[i]);
plt.legend(['$x_1$','$x_2$','$x_3$'],fontsize=15); plt.show()
mdcv=LassoCV(alphas=np.logspace(-4,0,100)).fit(x,y);
print("最优alpha=",mdcv.alpha_)
#md0=Lasso(mdcv.alpha_).fit(x,y)  #构建并拟合模型
md0=Lasso(0.21).fit(x,y)  #构建并拟合模型
cs0=md0.coef_  #提出标准化数据的回归系数b1,b2,b3
print("标准化数据的所有回归系数为：",cs0)
mu=np.mean(a,axis=0); s=np.std(a,axis=0,ddof=1) #计算所有指标的均值和标准差
params=[mu[-1]-s[-1]*sum(cs0*mu[:-1]/s[:-1]),s[-1]*cs0/s[:-1]]
print("原数据的回归系数为：",params)
print("拟合优度：",md0.score(x,y))
```

6.差分方程模型

7.模糊数学

①计算贴近度处理分类问题

使用方法：a中给出3类的数据，b给出待测数据，计算贴近度，越大越贴近

```
import numpy as np
a=np.array([[0.4,0.3,0.5,0.3],[0.3,0.3,0.4,0.4],[0.2,0.3,0.3,0.3]])
b=np.array([0.2,0.3,0.4,0.3]); N=[]
for i in range(len(a)): N.append(1-(np.linalg.norm(a[i]-b))/2)
print("贴近度分别为：",N)
```

②评价问题

使用方法：给出原始数据和权重

表 14.3　对某教师的评价数据

	好	较好	一般	较差	差
政治表现	4	2	1	0	0
教学水平	6	1	0	0	0
科研水平	0	0	5	1	1
外语水平	2	2	1	1	1

$$\boldsymbol{W}_1 = [0.2,\ 0.5,\ 0.1,\ 0.2],\ \boldsymbol{W}_2 = [0.2,\ 0.1,\ 0.5,\ 0.2].$$

```
import numpy as np
d=np.array([[4,2,1,0,0],[6,1,0,0,0],[0,0,5,1,1],[2,2,1,1,1]])
r=d/7; w1=np.array([0.2,0.5,0.1,0.2])
w2=np.array([0.2,0.1,0.5,0.2])
a1=np.dot(w1,r); a2=np.dot(w2,r)
print(a1,'\n----------------------------------------\n',a2)
```

③更精细的评价问题，需给出隶属函数

表 14.5　5 个评价指标

分数	亩产量	产品质量	亩用工量	亩纯收入	生态影响
5	550～600	1	≤20	≥130	1
4	500～550	2	20～30	110～130	2
3	450～500	3	30～40	90～110	3
2	400～450	4	40～50	70～90	4
1	350～400	5	50～60	50～70	5
0	≤350	6	≥60	≤50	6

解　根据评价标准建立各指标的隶属函数如下.

(1) 亩产量的隶属函数：

$$c_1(x_1) = \begin{cases} 0, & x_1 \leqslant 350, \\ \dfrac{x_1 - 350}{600 - 350}, & 350 < x_1 < 600, \\ 1, & x_1 \geqslant 600. \end{cases}$$

(2) 产品质量的隶属函数：

$$c_2(x_2) = \begin{cases} 1, & x_2 \leqslant 1, \\ 1 - \dfrac{x_2 - 1}{6 - 1}, & 1 < x_2 < 6, \\ 0, & x_2 \geqslant 6. \end{cases}$$

(3) 亩用工量的隶属函数：

$$c_3(x_3) = \begin{cases} 1, & x_3 \leqslant 20, \\ 1 - \dfrac{x_3 - 20}{60 - 20}, & 20 < x_3 < 60, \\ 0, & x_3 \geqslant 60. \end{cases}$$

```
from numpy import array, piecewise, c_
d=array([[592.5,3,55,72,5],[529,2,38,105,3],[412,1,32,85,2]]); d=d.T
c1=lambda x: piecewise(x, [(350<x) & (x<600),  x>=600], [lambda x:(x-350)/250., 1])
c2=lambda x: piecewise(x, [(1.0<x) & (x<6.0), (x<=1.0)],[lambda x:1-(x-1)/5.0,1])
c3=lambda x: piecewise(x, [(20<x) & (x<60), x<=20], [lambda x:1-(x-20)/40., 1])
c4=lambda x: piecewise(x, [(50<x) & (x<130), x>=130], [lambda x:(x-50)/80., 1])
c5=lambda x: piecewise(x, [(1<x) & (x<6), x<=1], [lambda x:1-(x-1)/5., 1])
r=c_[c1(d[0]),c2(d[1]),c3(d[2]),c4(d[3]),c5(d[4])].T
w=array([0.2, 0.1, 0.15, 0.3, 0.25]);
A=w.dot(r); print('R=',r,'\n---------------------------\n','A=',A)
```

8.灰色系统预测

①GM(1,1)模型预测

```
import numpy as np
import sympy as sp
from matplotlib.pyplot import plot,show,rc,legend,xticks
rc('font',size=16); rc('font',family='SimHei')
x0=np.array([25723,30379,34473,38485,40514,42400,48337])
n=len(x0); jibi=x0[:-1]/x0[1:]  #求级比
bd1=[jibi.min(),jibi.max()]    #求级比范围
bd2=[np.exp(-2/(n+1)),np.exp(2/(n+1))]   #q求级比的容许范围
x1=np.cumsum(x0)  #求累加序列
z=(x1[:-1]+x1[1:])/2.0
B=np.vstack([-z,np.ones(n-1)]).T
u=np.linalg.pinv(B)@x0[1:] #最小二乘法拟合参数

sp.var('t'); sp.var('x',cls=sp.Function)  #定义符号变量和函数
eq=x(t).diff(t)+u[0]*x(t)-u[1]  #定义符号微分方程
xt=sp.dsolve(eq,ics={x(0):x0[0]})  #求解符号微分方程
xt=xt.args[1]  #提取方程中的符号解
xt=sp.lambdify(t,xt,'numpy')  #转换为匿名函数
t=np.arange(n+1)
xt1=xt(t)  #求模型的预测值
x0_pred=np.hstack([x0[0],np.diff(xt1)]) #还原数据
x2002=x0_pred[-1]  #提取2002年的预测值
cha=x0-x0_pred[:-1]; delta=np.abs(cha/x0)*100
print('1995~2002的预测值：',x0_pred)
print('\n------------------\n','相对误差',delta)
t0=np.arange(1995,2002); plot(t0,x0,'*--')
plot(t0,x0_pred[:-1],'s-'); legend(('实际值','预测值'));
xticks(np.arange(1995,2002)); show()
```

②GM(1,N)模型预测

Pdata15_3.txt

1	7.123	0.796	13.108	27.475
2	7.994	0.832	12.334	27.473
3	8.272	0.917	12.216	27.490
4	7.960	0.976	12.201	27.500
5	7.147	1.075	12.132	27.510
6	7.092	1.121	11.990	27.542
7	6.858	1.281	11.926	27.536
8	5.804	1.350	10.478	27.550
9	6.433	1.410	9.176	27.567
10	6.354	1.432	11.368	27.584
11	6.254	1.507	12.764	27.600
12	5.197	1.559	11.143	27.602
13	5.654	1.611	10.737	27.630

```python
import numpy as np
from scipy.integrate import odeint
a=np.loadtxt("Pdata15_3.txt")  #加载表中的后4列数据
n=a.shape[0]  #观测数据的个数
x10=a[:,0]; x20=a[:,1]; x30=a[:,2]; x40=a[:,3]
x11=np.cumsum(x10); x21=np.cumsum(x20)
x31=np.cumsum(x30); x41=np.cumsum(x40)
z1=(x11[:-1]+x11[1:])/2.; z2=(x21[:-1]+x21[1:])/2.
z3=(x31[:-1]+x31[1:])/2.; z4=(x41[:-1]+x41[1:])/2.
B1=np.c_[z1,np.ones((n-1,1))]
u1=np.linalg.pinv(B1).dot(x10[1:]); print(u1)
B2=np.c_[z1,z2]
u2=np.linalg.pinv(B2).dot(x20[1:]); print(u2)
B3=np.c_[z3,np.ones((n-1,1))];
u3=np.linalg.pinv(B3).dot(x30[1:]); print(u3)
B4=np.c_[z1,z3,z4]
u4=np.linalg.pinv(B4).dot(x40[1:]); print(u4)
def Pfun(x,t):
    x1, x2, x3, x4 = x;
    return np.array([u1[0]*x1+u1[1], u2[0]*x1+u2[1]*x2,
           u3[0]*x3+u3[1], u4[0]*x1+u4[1]*x3+u4[2]*x4])
t=np.arange(0, 14);
X0=np.array([7.1230,0.7960,13.1080,27.475])
s1=odeint(Pfun, X0, t); s2=np.diff(s1,axis=0)
xh=np.vstack([X0,s2])
cha=a-xh[:-1,:]  #计算残差
```

```
delta=np.abs(cha/a)  #计算相对误差
maxd=delta.max(0)  #计算每个指标的最大相对误差
pre=xh[-1,:]; print("最大相对误差：",maxd,"\n预测值为：",pre)
```

③GM(2,1)模型预测，适用于S型曲线

```
import numpy as np
import sympy as sp
x0=np.array([41,49,61,78,96,104])
n=len(x0)
lamda=x0[:-1]/x0[1:]  #计算级比
rang=[lamda.min(), lamda.max()]  #计算级比的范围
theta=[np.exp(-2/(n+1)),np.exp(2/(n+1))] #计算级比容许范围
x1=np.cumsum(x0)  #累加运算
z=0.5*(x1[1:]+x1[:-1])
B=np.vstack([-x0[1:],-z,np.ones(n-1)]).T
u=np.linalg.pinv(B)@np.diff(x0)  #最小二乘法拟合参数
print("参数u：",u)
sp.var('t'); sp.var('x',cls=sp.Function)  #定义符号变量和函数
eq=x(t).diff(t,2)+u[0]*x(t).diff(t)+u[1]*x(t)-u[2]
s=sp.dsolve(eq,ics={x(0):x0[0],x(5):x1[-1]})  #求微分方程符号解
xt=s.args[1]  #提取解的符号表达式
print('xt=',xt)
fxt=sp.lambdify(t,xt,'numpy')  #转换为匿名函数
yuce1=fxt(np.arange(n))  #求预测值
yuce=np.hstack([x0[0],np.diff(yuce1)])  #还原数据
epsilon=x0-yuce[:n]  #计算已知数据预测的残差
delta=abs(epsilon/x0)  #计算相对误差
print('相对误差：',np.round(delta*100,2))  #显示相对误差
```

9.Monte Carlo模拟

①求全局最优解

使用方法：改变fx的函数式和x0中的取值范围即可

$$f(x) = (1 - x^3)\sin(3x), \quad -2\pi \leqslant x \leqslant 2\pi.$$

```
import numpy as np
from matplotlib.pyplot import rc, plot, show
from scipy.optimize import fminbound, fmin
rc('font',size=16)
fx=lambda x:(1-x**3)*np.sin(3*x);
x0=np.linspace(-2*np.pi,2*np.pi,100);
y0=fx(x0); plot(x0,y0); show()
xm1=fminbound(lambda x:-fx(x),-2*np.pi,2*np.pi)
ym1=fx(xm1); print(xm1,ym1,'\n--------------')
xm2=fmin(lambda x:-fx(x), -2*np.pi)
ym2=fx(xm2); print(xm2,ym2,'\n--------------')
x=np.random.uniform(-2*np.pi,2*np.pi,100)
y=fx(x); ym=y.max()
```

```
xm=x[y==ym]; print(xm,ym)
```

10.智能算法

①退火算法求解最小值

使用方法：输出txt中的数据，这里处理成了弧度，可删除相关代码

Pdata17_1.txt

```
53.7121   15.3046   51.1758    0.0322   46.3253   28.2753   30.3313    6.9348
56.5432   21.4188   10.8198   16.2529   22.7891   23.1045   10.1584   12.4819
20.1050   15.4562    1.9451    0.2057   26.4951   22.1221   31.4847    8.9640
26.2418   18.1760   44.0356   13.5401   28.9836   25.9879   38.4722   20.1731
28.2694   29.0011   32.1910    5.8699   36.4863   29.7284    0.9718   28.1477
8.9586    24.6635   16.5618   23.6143   10.5597   15.1178   50.2111   10.2944
8.1519    9.5325    22.1075   18.5569    0.1215   18.8726   48.2077   16.8889
31.9499   17.6309    0.7732    0.4656   47.4134   23.7783   41.8671    3.5667
43.5474    3.9061   53.3524   26.7256   30.8165   13.4595   27.7133    5.0706
23.9222    7.6306   51.9612   22.8511   12.7938   15.7307    4.9568    8.3669
21.5051   24.0909   15.2548   27.2111    6.2070    5.1442   49.2430   16.7044
17.1168   20.0354   34.1688   22.7571    9.4402    3.9200   11.5812   14.5677
52.1181    0.4088    9.5559   11.4219   24.4509    6.5634   26.7213   28.5667
37.5848   16.8474   35.6619    9.9333   24.4654    3.1644    0.7775    6.9576
14.4703   13.6368   19.8660   15.1224    3.1616    4.2428   18.5245   14.3598
58.6849   27.1485   39.5168   16.9371   56.5089   13.7090   52.5211   15.7957
38.4300    8.4648   51.8181   23.0159    8.9983   23.6440   50.1156   23.7816
13.7909    1.9510   34.0574   23.3960   23.0624    8.4319   19.9857    5.7902
40.8801   14.2978   58.8289   14.5229   18.6635    6.7436   52.8423   27.2880
39.9494   29.5114   47.5099   24.0664   10.1121   27.2662   28.7812   27.6659
8.0831    27.6705    9.1556   14.1304   53.7989    0.2199   33.6490    0.3980
1.3496    16.8359   49.9816    6.0828   19.3635   17.6622   36.9545   23.0265
15.7320   19.5697   11.5118   17.3884   44.0398   16.2635   39.7139   28.4203
6.9909    23.1804   38.3392   19.9950   24.6543   19.6057   36.9980   24.3992
4.1591    3.1853    40.1400   20.3030   23.9876    9.4030   41.1084   27.7149
```

```
from numpy import loadtxt,radians,sin,cos,inf,exp
from numpy import array,r_,c_,arange,savetxt
from numpy.lib.scimath import arccos
from numpy.random import shuffle,randint,rand
from matplotlib.pyplot import plot, show, rc
a=loadtxt("Pdata17_1.txt")
x=a[:,::2]. flatten(); y=a[:,1::2]. flatten()
d1=array([[70,40]]); xy=c_[x,y]
xy=r_[d1,xy,d1]; N=xy.shape[0]
t=radians(xy)  #转化为弧度
d=array([[6370*arccos(cos(t[i,0]-t[j,0])*cos(t[i,1])*cos(t[j,1])+
  sin(t[i,1])*sin(t[j,1])) for i in range(N)]
       for j in range(N)]).real
savetxt('Pdata17_2.txt',c_[xy,d])  #把数据保存到文本文件，供下面使用
path=arange(N); L=inf
for j in range(1000):
    path0=arange(1,N-1); shuffle(path0)
    path0=r_[0,path0,N-1]; L0=d[0,path0[1]]  #初始化
    for i in range(1,N-1):L0+=d[path0[i],path0[i+1]]
    if L0<L: path=path0; L=L0
print(path,'\n',L)
e=0.1**30; M=20000; at=0.999; T=1
for k in range(M):
```

```
    c=randint(1,101,2); c.sort()
    c1=c[0]; c2=c[1]
    df=d[path[c1-1],path[c2]]+d[path[c1],path[c2+1]]-\
    d[path[c1-1],path[c1]]-d[path[c2],path[c2+1]]  #续行
    if df<0:
        path=r_[path[0],path[1:c1],path[c2:c1-1:-1],path[c2+1:102]]; L=L+df
    else:
        if exp(-df/T)>=rand(1):
            path=r_[path[0],path[1:c1],path[c2:c1-1:-1],path[c2+1:102]]
            L=L+df
    T=T*at
    if T<e: break
print(path,'\n',L)  #输出巡航路径及路径长度
xx=xy[path,0]; yy=xy[path,1]; rc('font',size=16)
plot(xx,yy,'-*'); show()  #画巡航路径
```

②遗传算法处理最小值问题

Pdata17_2.txt

```
7.00000000000000000e+01 4.00000000000000000e+01 9.49203968048095703e-05 3.16763112441
5.37120999999999951e+01 1.53046000000000065e+01 3.16763112441429757 4e+03 0.00000000000
5.11758000000000240e+01 3.21999999999999946e-02 4.83568492202109609 9e+03 1.72064859267
4.63252999999999859e+01 2.82753000000000143e+01 2.52670818634758507 0e+03 1.63009942631
3.03312999999999882e+01 6.93480000000000075e+00 5.38923535704372261 5e+03 2.71204222185
5.65431999999999879e+01 2.14188000000000095e+01 2.42695251966805108 2e+03 7.42428552125
1.08198000000000075e+01 1.62529000000000035e+01 6.24513634390456445 5e+03 4.58176465990
2.27891000000000124e+01 2.31045000000000159e+01 4.78512676676089813 6e+03 3.35287676474
1.01584000000000032e+01 1.24818999999999955e+01 6.56209462034364423 7e+03 4.70354401115
2.01050000000000043e+01 1.54562000000000083e+01 5.52430320415524511 2e+03 3.59882598350
1.94510000000000051e+00 2.05699999999999940e-01 8.14112755417013704 6e+03 5.92391276837
2.64951000000000076e+01 2.21220999999999965e+01 4.53810988053184610 2e+03 2.95953919815
3.14847000000000013e+01 8.96400000000000412e+00 5.13572252649689289 7e+03 2.51471113695
2.62418000000000135e+01 1.81759999999999838e+01 4.82867850255505436 5e+03 2.93926980694
4.40356000000000230e+01 1.35401000000000069e+01 3.88094197101168811 1e+03 1.06008875631
2.89835999999999914e+01 2.59878999999999978e+01 4.08622292095244756 6e+03 2.82647497468
3.84722000000000084e+01 2.01731000000000159e+01 3.71642657491429781 7e+03 1.70110549705
2.82694000000000097e+01 2.90011000000000099e+01 3.97384488367556286 9e+03 3.01940261040
3.21910000000000250e+01 5.86990000000000340e+00 5.34804687076525897 3e+03 2.57210751624
```

```
import numpy as np
from numpy.random import randint, rand, shuffle
from matplotlib.pyplot import plot, show, rc
a=np.loadtxt("Pdata17_2.txt")
xy,d=a[:,:2],a[:,2:]; N=len(xy)
w=50; g=10  #w为种群的个数，g为进化的代数
J=[];
for i in np.arange(w):
    c=np.arange(1,N-1); shuffle(c)
    c1=np.r_[0,c,101]; flag=1
    while flag>0:
        flag=0
        for m in np.arange(1,N-3):
            for n in np.arange(m+1,N-2):
                if d[c1[m],c1[n]]+d[c1[m+1],c1[n+1]]<\
                   d[c1[m],c1[m+1]]+d[c1[n],c1[n+1]]:
                    c1[m+1:n+1]=c1[n:m:-1]; flag=1
    c1[c1]=np.arange(N); J.append(c1)
J=np.array(J)/(N-1)
for k in np.arange(g):
```

```
    A=J.copy()
    c1=np.arange(w); shuffle(c1) #交叉操作的染色体配对组
    c2=randint(2,100,w)  #交叉点的数据
    for i in np.arange(0,w,2):
        temp=A[c1[i],c2[i]:N-1]  #保存中间变量
        A[c1[i],c2[i]:N-1]=A[c1[i+1],c2[i]:N-1]
        A[c1[i+1],c2[i]:N-1]=temp
    B=A.copy()
    by=[]  #初始化变异染色体的序号
    while len(by)<1: by=np.where(rand(w)<0.1)
    by=by[0]; B=B[by,:]
    G=np.r_[J,A,B]
    ind=np.argsort(G,axis=1)  #把染色体翻译成0,1，…，101
    NN=G.shape[0]; L=np.zeros(NN)
    for j in np.arange(NN):
        for i in np.arange(101):
            L[j]=L[j]+d[ind[j,i],ind[j,i+1]]
    ind2=np.argsort(L)
    J=G[ind2,:]
path=ind[ind2[0],:]; zL=L[ind2[0]]
xx=xy[path,0]; yy=xy[path,1]; rc('font',size=16)
plot(xx,yy,'-*'); show()  #画巡航路径
print("所求的巡航路径长度为：",zL)
```

③神经网络感知器分类

使用方法：在x0中输入原始数据矩阵，y0输入类型

```
from sklearn.linear_model import Perceptron
import numpy as np
x0=np.array([[-0.5,-0.5,0.3,0.0],[-0.5,0.5,-0.5,1.0]]).T
y0=np.array([1,1,0,0])
md = Perceptron(tol=1e-3)    #构造模型
md.fit(x0, y0)               #拟合模型
print(md.coef_,md.intercept_)  #输出系数和常数项
print(md.score(x0,y0))    #模型检验
print("预测值为：",md.predict(np.array([[-0.5,0.2]])))
```

④BP神经网络预测分析

Pdata17_5.txt

```
20.55   0.6 0.09    5126    1237
22.44   0.75    0.11    6217    1379
25.37   0.85    0.11    7730    1385
27.13   0.9 0.14    9145    1399
29.45   1.05    0.2 10460   1663
30.1    1.35    0.23    11387   1714
30.96   1.45    0.23    12353   1834
34.06   1.6 0.32    15750   4322
36.42   1.7 0.32    18304   8132
38.09   1.85    0.34    19836   8936
39.13   2.15    0.36    21024   11099
39.99   2.2 0.36    19490   11203
41.93   2.25    0.38    20433   10524
44.59   2.35    0.49    22598   11115
47.3    2.5 0.56    25107   13320
52.89   2.6 0.59    33442   16762
55.73   2.7 0.59    36836   18673
56.76   2.85    0.67    40548   20724
59.17   2.95    0.69    42927   20803
60.63   3.1 0.79    43462   21804
```

```
from sklearn.neural_network import MLPRegressor
from numpy import array, loadtxt
from pylab import subplot, plot, show, xticks,rc,legend
rc('font',size=15); rc('font',family='SimHei')
a=loadtxt("Pdata17_5.txt"); x0=a[:,:3]; y1=a[:,3]; y2=a[:,4];
md1=MLPRegressor(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=10)
md1.fit(x0, y1); x=array([[73.39,3.9635,0.988],[75.55,4.0975,1.0268]])
pred1=md1.predict(x); print(md1.score(x0,y1));
print("客运量的预测值为: ",pred1,'\n----------------');
md2=MLPRegressor(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=10)
md2.fit(x0, y2); pred2=md2.predict(x); print(md2.score(x0,y2));
print("货运量的预测值为: ",pred2); yr=range(1990,2010)
subplot(121); plot(yr,y1,'o'); plot(yr,md1.predict(x0),'-*')
xticks(yr,rotation=55); legend(("原始数据","网络输出客运量"))
subplot(122); plot(yr,y2,'o'); plot(yr,md2.predict(x0),'-*')
xticks(yr,rotation=55)
legend(("原始数据","网络输出货运量"),loc='upper left'); show()
```

11.时间序列分析

①一次移动平均法预测

使用方法：在y中改变已知参数

```
import numpy as np
y=np.array([423,358,434,445,527,429,426,502,480,384,427,446])
def MoveAverage(y,N):
    Mt=['*']*N
    for i in range(N+1,len(y)+2):
        M=y[i-(N+1):i-1].mean()
        Mt.append(M)
    return Mt
yt3=MoveAverage(y,3)
s3=np.sqrt(((y[3:]-yt3[3:-1])**2).mean())
yt5=MoveAverage(y,5)
s5=np.sqrt(((y[5:]-yt5[5:-1])**2).mean())
print('N=3时,预测值：',yt3,'，预测的标准误差：',s3)
print('N=5时,预测值：',yt5,'，预测的标准误差：',s5)
```

②一次指数平滑法预测

```
import numpy as np
import pandas as pd
y=np.array([4.81,4.8,4.73,4.7,4.7,4.73,4.75,4.75,5.43,5.78,5.85])
def ExpMove(y,a):
    n=len(y); M=np.zeros(n); M[0]=(y[0]+y[1])/2;
    for i in range(1,len(y)):
        M[i]=a*y[i-1]+(1-a)*M[i-1]
    return M
yt1=ExpMove(y,0.2); yt2=ExpMove(y,0.5)
yt3=ExpMove(y,0.8); s1=np.sqrt(((y-yt1)**2).mean())
s2=np.sqrt(((y-yt2)**2).mean())
s3=np.sqrt(((y-yt3)**2).mean())
d=pd.DataFrame(np.c_[yt1,yt2,yt3])
f=pd.ExcelWriter("Pdata18_2.xlsx");
d.to_excel(f); f.close()  #数据写入Excel文件，便于做表
print("预测的标准误差分别为：",s1,s2,s3)  #输出预测的标准误差
yh=0.8*y[-1]+0.2*yt3[-1]
print("下一期的预测值为：",yh)
```

③二次指数平滑法预测

使用方法：在Pdata18_3.txt中修改初始参数

Pdata18_3.txt

```
1    2031
2    2234
3    2566
4    2820
5    3006
6    3093
7    3277
8    3514
9    3770
10   4107
```

```python
import numpy as np
import pandas as pd
y=np.loadtxt('Pdata18_3.txt')
n=len(y); alpha=0.3; yh=np.zeros(n)
s1=np.zeros(n); s2=np.zeros(n)
s1[0]=y[0]; s2[0]=y[0]
for i in range(1,n):
    s1[i]=alpha*y[i]+(1-alpha)*s1[i-1]
    s2[i]=alpha*s1[i]+(1-alpha)*s2[i-1];
    yh[i]=2*s1[i-1]-s2[i-1]+alpha/(1-alpha)*(s1[i-1]-s2[i-1])
at=2*s1[-1]-s2[-1]; bt=alpha/(1-alpha)*(s1[-1]-s2[-1])
m=np.array([1,2])
yh2=at+bt*m
print("预测值为：",yh2)
d=pd.DataFrame(np.c_[s1,s2,yh])
f=pd.ExcelWriter("Pdata18_3.xlsx");
d.to_excel(f); f.close()
```

④季节性时间序列预测

Pdata18_4.txt

```
1    265 373 333 266
2    251 379 374 309
3    272 437 396 348
```

```python
import numpy as np
a=np.loadtxt('Pdata18_4.txt')
m,n=a.shape
amean=a.mean()  #计算所有数据的平均值
cmean=a.mean(axis=0)   #逐列求均值
r=cmean/amean   #计算季节系数
w=np.arange(1,m+1)
yh=w.dot(a.sum(axis=1))/w.sum()  #计算下一年的预测值
yj=yh/n   #计算预测年份的季度平均值
yjh=yj*r  #计算季度预测值
print("下一年度各季度的预测值为：",yjh)
```

12.支持向量机

①支持向量机解决分类问题

使用方法：将x，y替换为原始数据

```python
from sklearn import datasets, svm, metrics
from sklearn.model_selection import GridSearchCV
import numpy as np
iris=datasets.load_iris()
x=iris.data; y=iris.target
parameters = {'kernel':('linear','rbf'), 'C':[1,10,15]}
svc=svm.SVC(gamma='scale')
clf=GridSearchCV(svc,parameters,cv=5)  #cv为交叉验证参数，为5折
clf.fit(x,y)
print("最佳的参数值:", clf.best_params_)
print("score：",clf.score(x,y))
yh=clf.predict(x); print(yh) #显示分类的结果
print("预测准确率：",metrics.accuracy_score(y,yh))
print("误判的样本点为:",np.where(yh!=y)[0]+1)
```

②支持向量机回归分析

```python
import numpy as np
import pylab as plt
from sklearn.svm import SVR

np.random.seed(123)
x=np.arange(200).reshape(-1,1)
y=(np.sin(x)+3+np.random.uniform(-1,1,(200,1))).ravel()

model = SVR(gamma='auto'); print(model)
model.fit(x,y); pred_y = model.predict(x)
print("原始数据与预测值前15个值对比：")
for i in range(15): print(y[i],pred_y[i])

plt.rc('font',family='SimHei'); plt.rc('font',size=15)
plt.scatter(x, y, s=5, color="blue", label="原始数据")
plt.plot(x, pred_y, '-r*',lw=1.5, label="预测值")
plt.legend(loc=1)

score=model.score(x,y); print("score:",score)
```

```
ss=((y-pred_y)**2).sum()  #计算残差平方和
print("残差平方和：", ss)
plt.show()
```