# 《Python 程序设计综合训练》

# 实验报告

班级：

姓名：

学号：

# 1 爬虫练习

## 1.1 实验环境

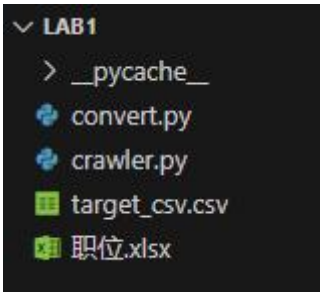
对于本实验，我采用的是 vscode+python3.11.1+Anaconda，同时需要利用 pip 语句进行爬虫所需要的库的安装：

```
pip install requests
pip install selenium
```

## 1.2 实验内容

本次实验，我在老师提供的爬虫代码的基础上，完成了 51job 的对于指定关键词 python 和西安的职位的寻找，并将所寻找到的岗位及其相关信息进行了提取，保存到了 excel 文件中。而对于优化方面，我利用 pandas 库完成了 xlsx 文件到 csv 文件的互相转换，从而方便不同的工作者进行数据处理。

因此，本实验包含两个 python 文件，分别是 crawler.py 和 convert.py，项目结构如下：




## 1.3 实验代码

### 1.3.1 爬虫

```
#Selenium 模块是一个自动化测试工具，能够驱动浏览器模拟人的操作，如单击、键盘输入等。
from selenium import webdriver
from selenium.webdriver.common.by import By
import re
import pandas as pd
```

```
import time
from convert import xlsx_to_csv

#从获取的网页源代码中提取目标数据
def extract_data(html_code):
    #目标数据的正则表达式
    p_job = 'class="jname at">(.*?)</span>'
    p_salary = 'class="sal">(.*?)</span>'
    p_needs_city =  'class="info">.*?class="d at">.*?>(.*?)</span>'
    p_needs_exp =   'class="info">.*?class="d at">.*?>.*?</span>.*?</span>.*?>(.*?)</span>'
    p_needs_xueli = 'class="info">.*?class="d at">.*?>.*?</span>.*?</span>.*?</span>.*?</span>.*?>(.*?)</span>'
    p_link = 'class="er">.*?href="(.*?)"'
    p_company = 'class="er">.*?title="(.*?)".*?</a>'

    #利用 findall()函数提取目标数据
    job = re.findall(p_job, html_code, re.S)
    salary = re.findall(p_salary, html_code, re.S)
    needs_city = re.findall(p_needs_city, html_code, re.S)
    needs_exp = re.findall(p_needs_exp, html_code, re.S)
    needs_xueli = re.findall(p_needs_xueli, html_code, re.S)
    link = re.findall(p_link, html_code, re.S)
    company = re.findall(p_company, html_code, re.S)

    #将几个目标数据列表转换为一个字典
    data_dt = {'职位名称': job, '月薪': salary, '岗位城市': needs_city, '要求经验': needs_exp, '要求学历': needs_xueli, '职位申请链接': link, '公司名称': company}
    #用上面的字典创建一个 DataFrame
    return pd.DataFrame(data_dt)
def get_pages(keyword, city, start, end):
    # 声明要模拟的浏览器是 Chrome,并启用无界面浏览模式
    chrome_options = webdriver.ChromeOptions()
     chrome_options.add_argument("--disable-blink-features=AutomationControlled")
    browser = webdriver.Chrome(options=chrome_options)
    browser.maximize_window()

    # 通过 get()函数控制浏览器发起请求，访问网址,获取源码
    url = 'https://www.51job.com/'
    browser.get(url)
    #模拟人操作浏览器，输入搜索关键词，点击搜索按钮
    browser.find_element(By.XPATH, '//*[@id="kwdselectid"]').clear()
```

```
    browser.find_element(By.XPATH, '//*[@id="kwdselectid"]').send_keys(keyword)
    browser.find_element(By.XPATH, '/html/body/div[3]/div/div[1]/div/button').click()

    time.sleep(10)
    all_data = pd.DataFrame()
    for page in range(start, end + 1):
        # 模拟人操作浏览器，输入搜索关键词，点击搜索按钮
        browser.find_element(By.XPATH, '//*[@id="jump_page"]').clear()
        browser.find_element(By.XPATH, '//*[@id="jump_page"]').send_keys(page)
        browser.find_element(By.XPATH, '//*[@id="app"]/div/div[2]/div/div/div[2]/div/div[2]/div/div[3]/div/div/span[3]').click()
        # 等待浏览器与服务器交互刷新数据，否则获取不到动态信息
        time.sleep(10)
        #将提取的目标数据添加到 DataFrame 中
        all_data = all_data.append(extract_data(browser.page_source))

    browser.quit()

    #将 DataFrame 保存为 Excel
    all_data.to_excel('职位.xlsx', index=False)
    xlsx_to_csv('职位.xlsx')

get_pages('python','西安', 1, 3)
```

### 1.3.2 文件转换

```
import pandas as pd

def xlsx_to_csv(filename):
    data_xls = pd.read_excel(filename, index_col=0)
    data_xls.to_csv('target_csv.csv', encoding='utf-8')

def csv_to_xlsx(filename):
    data_csv = pd.read_csv(filename,  encoding='utf-8')
    data_csv.to_excel('target_excel.xlsx', sheet_name='data')
```

# 2 办公自动化

## 2.1 分词

### 2.1.1 实验环境

对于本实验，我采用的是 vscode+python3.11.1+Anaconda，同时需要利用 pip 语句进行所需要的库的安装：

```
pip install jieba
pip install wordcloud
```

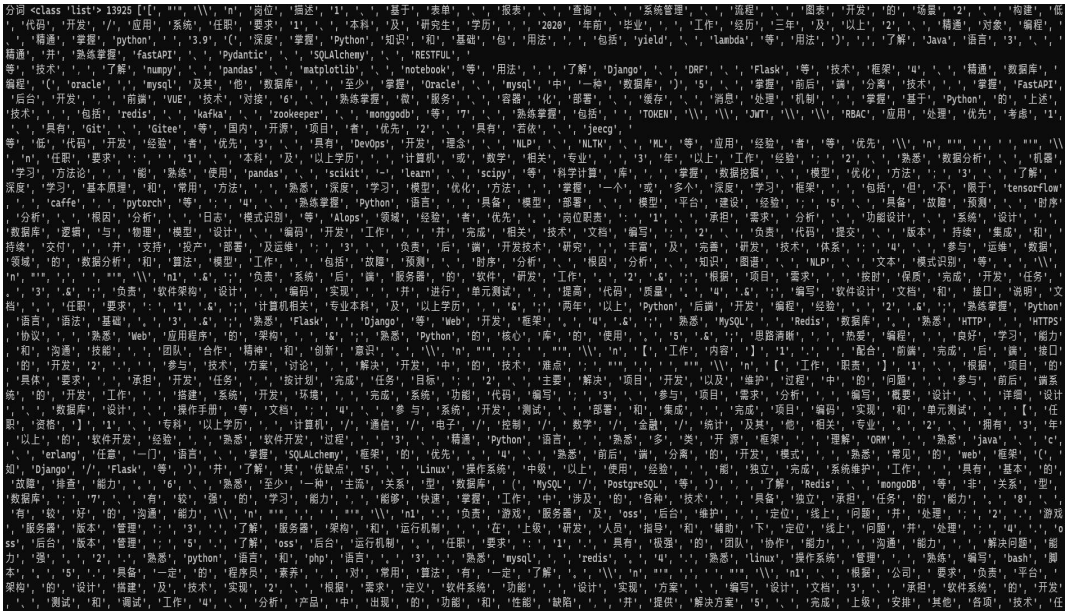
### 2.1.2 实验代码

```
#Selenium 模块是一个自动化测试工具，能够驱动浏览器模拟人的操作，如单击、键盘输入等。
import re
import jieba
from jieba.analyse import *
from openpyxl import load_workbook
from wordcloud import WordCloud
import matplotlib.pyplot as plt

#读取 excel 文件工作表 Sheet1
workbook = load_workbook('职位.xlsx')
worksheet = workbook['Sheet1']
#将 Sheet1 中 F 列内容写入 txt 文件
data = []
f = open('职位信息.txt', 'w')
for row in range(2, worksheet.max_row + 1):
    data.append(worksheet['F'+str(row)].value)
f.write(str(data))
f.close()

#去除 html 标签
f = open('职位信息.txt', 'r').read()
g = re.sub('<.*?>', '', f)
g = re.sub('nbsp', '', g)
g = re.sub('  ', '', g)
#清洗后数据写入新的 txt
h = open('职位信息-清洗后.txt', 'w')
h.write(g)
```

```
h.close()

#生成词云
#读取 txt
f = open('职位信息-清洗后.txt', 'r', encoding='gbk').read()
#分词
sep_list = jieba.lcut(f)
print('分词', type(sep_list), len(sep_list), sep_list)
#过滤停用词
# stopwords 为停用词 list，stop.txt 停用词一行一个
stopwords = [line.strip() for line in open('stop.txt', 'r', encoding='utf-8').readlines()]
print('停用词', type(stopwords), stopwords)
outstr = ''
for word in sep_list:
    if word not in stopwords:
        outstr += word
print('过滤停用词后', type(outstr), outstr)
#过滤后重新分词
outstr = jieba.lcut(outstr)
outstr = ' '.join(outstr)
print('再次分词后', type(outstr), outstr)
#生成词云图
#设置词云使用的字体
font = r'C:\Windows\Fonts\simsun.ttc'
wc = WordCloud(font_path=font, width=2400, height=1200, max_words=100)
wc.generate(outstr)
wc.to_file('词云.jpg')
plt.figure(dpi=100)
plt.imshow(wc, interpolation='catrom')
plt.axis('off')
plt.show()
plt.close()

#生成词频
for keyword, weight in extract_tags(outstr, topK=50, withWeight=True, allowPOS=()):
    print('%s %s' % (keyword, weight))
```

### 2.1.3 实验结果

在终端运行 python fenci.py 后，在终端的分词结果如下：




生成的词云图如下：


获得词频如下：

### 2.1.4 一点补充

尽管这个实验我仅仅是使用了老师提供的代码，但仍然有一些需要注意的点：

1. 导入包中有一个 wordcloud，因此对于 py 文件不能命名为 wordcloud.py，否则就会让 IDLE 认为循环导包。
2. 中文编码应设置成 utf-8.

## 2.2 图片处理

### 2.2.1 实验环境

利用如下语句安装图片处理所需的环境：

```
pip install opencv-python
```

### 2.2.2 实验代码

```
import cv2

#读取工程文件下的图片，也可使用绝对路径，cv2 读取的是 BGR 格式
img = cv2.imread('qianzi.jpg')
#BGR 转灰度图
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
#二值化，调整 thresh 值（125）直到获得较好结果
ret, img1 = cv2.threshold(img, 125, 255, cv2.THRESH_BINARY)
img4 = cv2.imwrite('qianzi-binary.jpg', img1)
#转为 BGRA，带透明度的四通道
img2 = cv2.cvtColor(img1, cv2.COLOR_GRAY2BGRA)
#图片 shape 属性是 tuple 类型，元组
print(img2.shape)
#遍历所有像素，白色像素（255）的透明度设置为透明（0）
for i in range(img2.shape[0]):
    for j in range(img2.shape[1]):
        if img2[i, j][0] == 255:
            img2[i, j][3] = 0
#保存为新图片
img3 = cv2.imwrite('qianzi-binary.png', img2)
```

### 2.2.3 实验结果

上述代码生成了两个文件，一个是 qianzi-binary.jpg，另一个是 qianzi-binary.png，如下

所示：

## 2.3 自动化处理 Word 文档

### 2.3.1 实验环境

利用如下语句安装图片处理所需的环境：

```
pip install python-docx
```

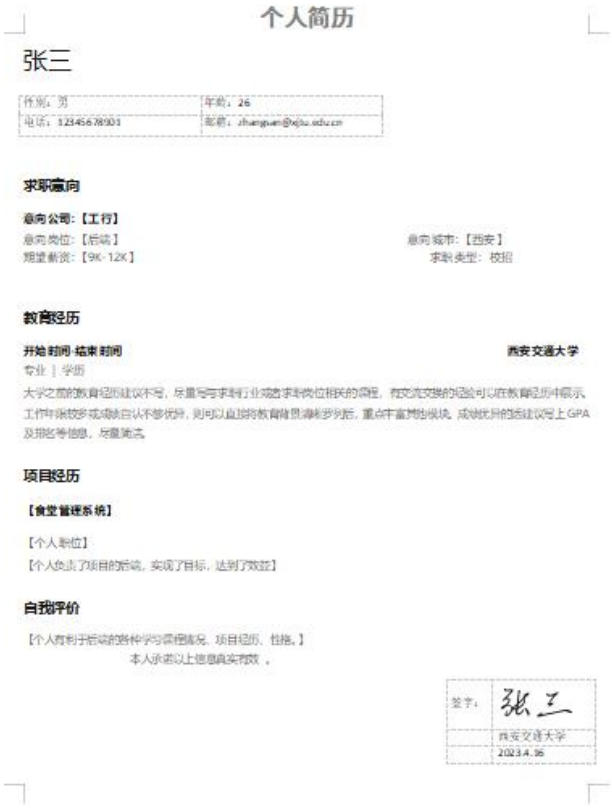
### 2.3.2 实验代码

```
from docx import Document
from docx.shared import Inches
from openpyxl import load_workbook

#替换 word 中的关键词
def info_update(doc, old_info, new_info):
    #遍历替换所有段落中的关键词
    for para in doc.paragraphs:
        for run in para.runs:
            run.text = run.text.replace(old_info, new_info)
    #遍历替换所有表格中的关键词
    for table in doc.tables:
        for row in table.rows:
            for cell in row.cells:
                cell.text = cell.text.replace(old_info, new_info)
#读取 excel 中的信息
wb = load_workbook('简历信息.xlsx')
ws = wb.active
#根据简历信息中的每行生成一个简历，即为每个公司生成一份专属简历
for row in range(2, ws.max_row + 1):
    doc = Document('简历模板.docx')
```

```
    for col in range(1, ws.max_column + 1):
        #简历信息中的标题，对应 word 中的关键词
        old_info = str(ws.cell(row=1, column=col).value)
        #简历信息中的替换的具体公司关键词
        new_info = str(ws.cell(row=row, column=col).value)
        #执行替换
        info_update(doc, old_info, new_info)
    #插入电子签名
    #在 word 中的第 2 个表格的（0，1）单元格插入图片
    table = doc.tables[1]
    run = table.cell(0,1).paragraphs[0].add_run('')
    run.add_picture('qianzi-binary.png', width=Inches(1), height=Inches(0.5))

    #基于更新后的简历模板生成以对应公司命名的简历
    com_name = str(ws.cell(row=row, column=1).value)
    doc.save(f'{com_name}简历.docx')
```

## 2.3.3 实验结果

上述代码实现了批量生成 word 简历，生成了百度，工行，腾讯，网易等按照模板的简历，如下图所示：



个人简历

张三

| | |
|---|---|
| 性别：男 | 年龄：26 |
| 电话：12345678901 | 邮箱：zhangsan@xjtu.edu.cn |

**求职意向**

意向公司：【工行】

意向岗位：【后端】　　意向城市：【西安】

期望薪资：【9K-12K】　　求职类型：校招

**教育经历**

开始时间-结束时间　　西安交通大学

专业 | 学历

大学之前的教育经历建议不写，尽量写与求职行业或者求职岗位相关的信息，有交流交换的经验可以在教育经历中提示。工作年限较多或项目认不够优异，则可以直接将教育背景缩减至两句话，重点丰富其他模块。成绩优异的话建议写上 GPA 及排名等信息，尽量简洁。

**项目经历**

【食堂管理系统】

【个人职位】

【个人负责了项目的后端，实现了目标，达到了效益】

**自我评价**

【个人有利于后端的各种学习课程情况、项目经历、性格。】

本人承诺以上信息真实有效，

签字：张三

西安交通大学

2023.4.16

# 3 科学计算

## 3.1 拟合与优化

### 3.1.1 实验环境

本次实验在 google colab 上进行编写，主要是为了逐行运行不同模块的代码的同时，对于一些全新的库的安装，不用在本地进行（主要是为了偷懒）

### 3.1.2 非线性方程组求解

非线性方程组求解

```
[1] # import packages
    from math import sin, cos
    from scipy import optimize
```

```
[2] def f(x):
      x0, x1, x2 = x.tolist()
      return [5 * x1 - 3,
              4 * x0 * x0 - 2 * sin(x1 * x2),
              x1 * x2 - 1.5]
```

```
res = optimize.fsolve(f, [1, 1, 1])
print(res)
print(f(res))
```

```
[0.70622057 0.6        2.5        ]
[0.0, 1.8729773287873286e-10, 7.549760816516482e-11]
```

### 3.1.3 最小二乘拟合

```
import numpy as np

X = np.array([8.19, 2.72, 6.39, 8.71, 4.7, 2.66, 3.78])
Y = np.array([7.01, 2.78, 6.47, 6.71, 4.1, 4.23, 4.05])

def residual(p):
  k, b = p
  return Y - (k * X + b)


r = optimize.leastsq(residual, [1, 0])
k, b = r[0]
print("k = ", k, "b = ", b)
```

```
k =  0.6134953491930442 b =  1.794092543259387
```

第二个拟合的实验是队带噪声的正弦波拟合，并且需要用 plt 库进行绘图，这里就需要引入一个 pylab 的库，并且引入相关的参数，如下所示：

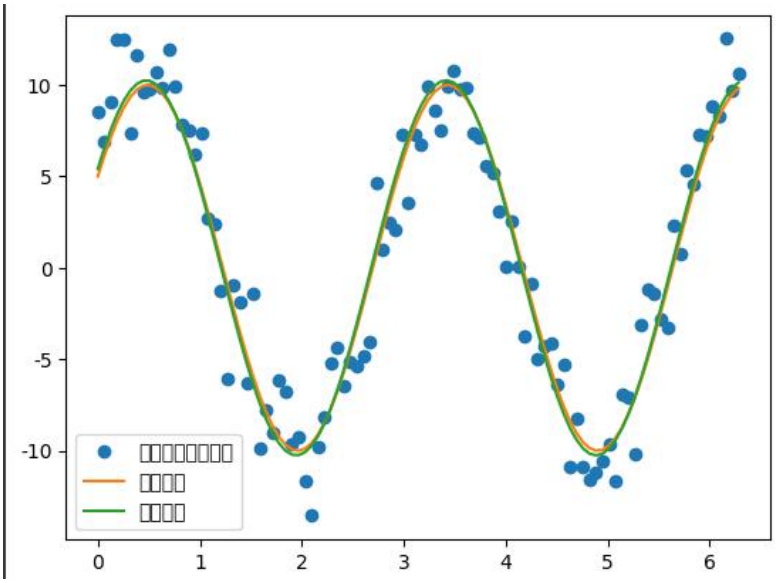
```
1. import pylab as pl
2.
3. pl.rcParams['font.sans-serif'] = [u'SimHei']
4. pl.rcParams['axes.unicode_minus'] = False
```

```
def func(x, p):
  A, K, theta = p
  return A * np.sin(2 * np.pi * k * x + theta)

def residual(p, y, x):
  return y - func(x, p)
```

```
[6] x = np.linspace(0, 2 * np.pi, 100)
A, k, theta = 10, 0.34, np.pi / 6
y0 = func(x, [A, k, theta])

# 加入噪声
np.random.seed(0)
y1 = y0 + 2 * np.random.randn(len(x))

p0 = [7, 0.40, 0]
```

```
plsq = optimize.leastsq(residual, p0, args=(y1, x))

print(u"真实函数: ", [A, k, theta])
print(u"拟合参数: ", plsq[0])

pl.plot(x, y1, "o", label=u"带噪声的实验数据")
pl.plot(x, y0, label=u"真实数据")
pl.plot(x, func(x, plsq[0]), label=u"拟合数据")
pl.legend(loc="best")
pl.show()
```

拟合曲线结果如下图：




# 3.2 特征值与特征向量

## 3.2.1 n*n 矩阵的特征向量

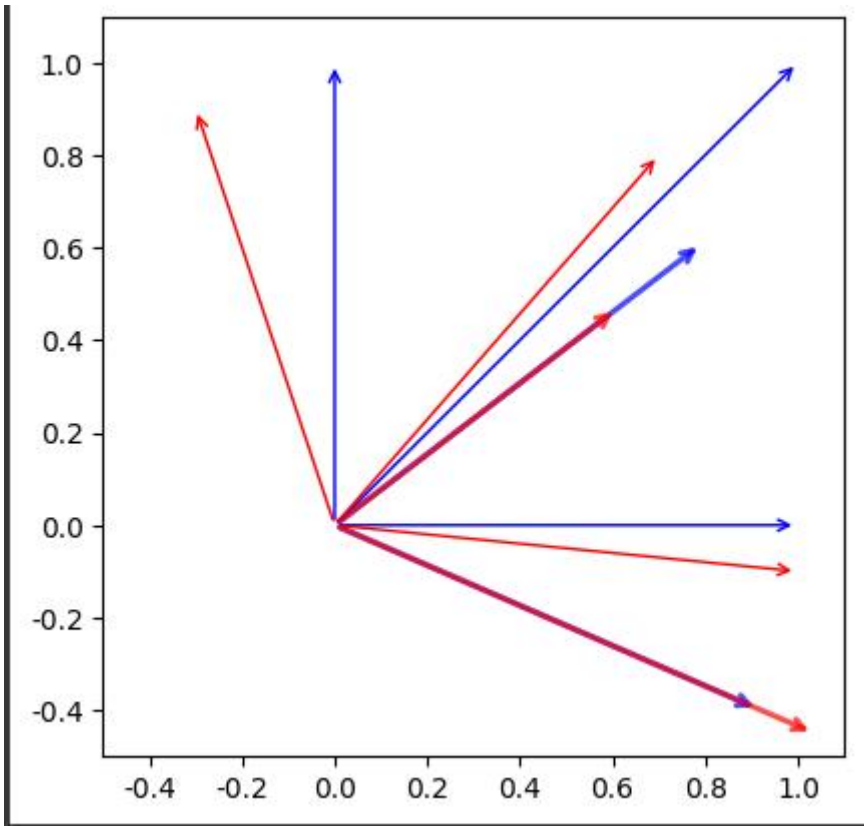
通过 linalg.eig(A)可以计算矩阵的特征值和特征向量，代码如下：

```
1.  A = np.array([[1, -0.3], [-0.1, 0.9]])
2.  evalues, evectors = linalg.eig(A)
```

接下来，构造 3 个普通向量做对比测试，为了可视化，需要定义一个 draw_arrows 的函数：

```
# 线性变换将蓝色箭头变换为红色箭头
points = np.array([[0, 1.0], [1.0, 0], [1, 1]])
def draw_arrows(points, **kw):
    props = dict(color="blue", arrowstyle="->")
    props.update(kw)
    for x, y in points:
        pl.annotate("",
                    xy=(x, y), xycoords='data',
                    xytext=(0, 0), textcoords='data',
                    arrowprops=props)

draw_arrows(points)
draw_arrows(np.dot(A, points.T).T, color="red")
draw_arrows(evectors.T, alpha=0.7, linewidth=2)
draw_arrows(np.dot(A, evectors).T, color="red", alpha=0.7, linewidth=2)

ax = pl.gca()
ax.set_aspect("equal")
ax.set_xlim(-0.5, 1.1)
ax.set_ylim(-0.5, 1.1)
pl.show()
```

实验结果如下：




### 3.2.2 广义特征值

对于广义特征值伪命题，需要另外一个矩阵 B，通过 linalg.eig(A, B)可以计算矩阵的广义特征值和特征向量

```
A = np.array([[1, -0.3], [-0.1, 0.9]])
B = np.array([[1, 0.5], [0.7, -0.9]])
evalues, evectors = linalg.eig(A, B)
print(evalues)
print(evectors)
```

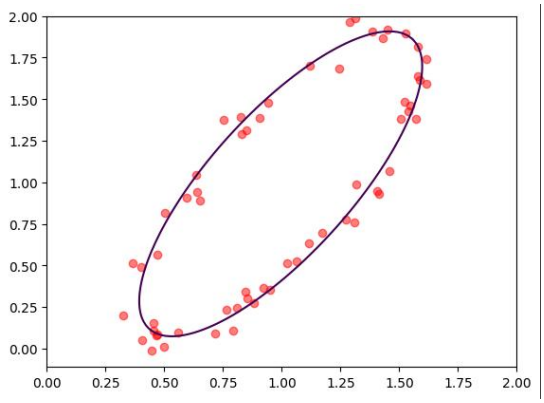
```
[ 0.7367235+0.j -0.9447235+0.j]
[[-0.93041671  0.08828439]
 [-0.36650341 -0.99609531]]
```

### 3.2.3 椭圆拟合求解

椭圆的拟合求解分为一下三步：

1. 计算广义特征向量，特征值
2. 将特征向量带入椭圆方程
3. 选择误差最小的特征向量作为椭圆参数

```
np.random.seed(42)
t = np.random.uniform(0, 2 * np.pi, 60)

alpha = 0.4
a = 0.5
b = 1.0
x = 1.0 + a * np.cos(t) * np.cos(alpha) - b * np.sin(t) * np.sin(alpha)
y = 1.0 + a * np.cos(t) * np.sin(alpha) - b * np.sin(t) * np.cos(alpha)
x += np.random.normal(0, 0.05, size=len(x))
y += np.random.normal(0, 0.05, size=len(y))

D = np.c_[x ** 2, x * y, y ** 2, x, y, np.ones_like(x)]
A = np.dot(D.T, D)
C = np.zeros((6, 6))
C[[0, 1, 2], [2, 1, 0]] = 2, -1, 2
evalues, evectors = linalg.eig(A, C)
evectors = np.real(evectors)
err = np.mean(np.dot(D, evectors) ** 2, 0)
p = evectors[:, np.argmin(err)]
print(p)

def ellipse(p, x, y):
    a, b, c, d, e, f = p
    return a * x ** 2 + b * x * y + c * y ** 2 + d * x + e * y + f


X, Y = np.mgrid[0:2:100j, 0:2:100j]
Z = ellipse(p, X, Y)
pl.plot(x, y, "ro", alpha=0.5)
pl.contour(X, Y, Z, levels=[0])
pl.show()
```



# 3.3 最短路径

## 3.3.1 实验环境

本实验不涉及到科学计算，就在本地 vscode+python3.11.1 中运行即可

## 3.3.2 实验代码

```
from scipy.sparse import csgraph
import subprocess

from scipy import sparse
if __name__ == '__main__':
    a = sparse.dok_matrix((10, 5))
    a[2, 3] = 1.0
    a[3, 3] = 2.0
    a[4, 3] = 3.0
    print(a.keys())
    print(a.values())

    b = sparse.lil_matrix((10, 5))
    b[2, 3] = 1.0
    b[3, 4] = 2.0
    b[3, 2] = 3.0
    print(b.data)
    print(b.rows)

    print("====================================")
    row = [2, 3, 3, 2]
    col = [3, 4, 2, 3]
    data = [1, 2, 3, 10]
    c = sparse.coo_matrix((data, (row, col)), shape=(5, 6))
    print(c.col, c.row, c.data)
    print(c.toarray())
    print("====================================")

    # Find the shortest path
    # Define the graph in the DOT language
    code = """
```

```
    digraph graph1{
        rankdir=LR;
        size="8,5"
        node [shape = circle];
        A -> B [ label = "10" ];
        B -> C [ label = "5" ];
        A -> C [ label = "3" ];
        C -> D [ label = "7" ];
        D -> A [ label = "4" ];
        D -> C [ label = "6" ];
    }
    """

    # Define the command and its arguments to run Graphviz
    dot_args = ["dot", "-T", "svg"]

    # Run Graphviz and pass the DOT code as input
    p = subprocess.Popen(dot_args, stdin=subprocess.PIPE, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
    stdout, stderr = p.communicate(code.encode('utf-8'))

    # Save the SVG image to a file
    with open("output.svg", "w") as f:
        f.write(stdout.decode("utf-8"))

    w = sparse.dok_matrix((4,4))
    edges = [(0, 1, 10), (1, 2, 5), (0, 2, 3),
             (2, 3, 7), (3, 0, 4), (3, 2, 6)]

    for i, j, v in edges:
        w[i, j] = v
    d, p = csgraph.dijkstra(csgraph=w, directed=True, indices=0, return_predecessors=True)

    print(d)
    print(p)
```

Dijsktra.py

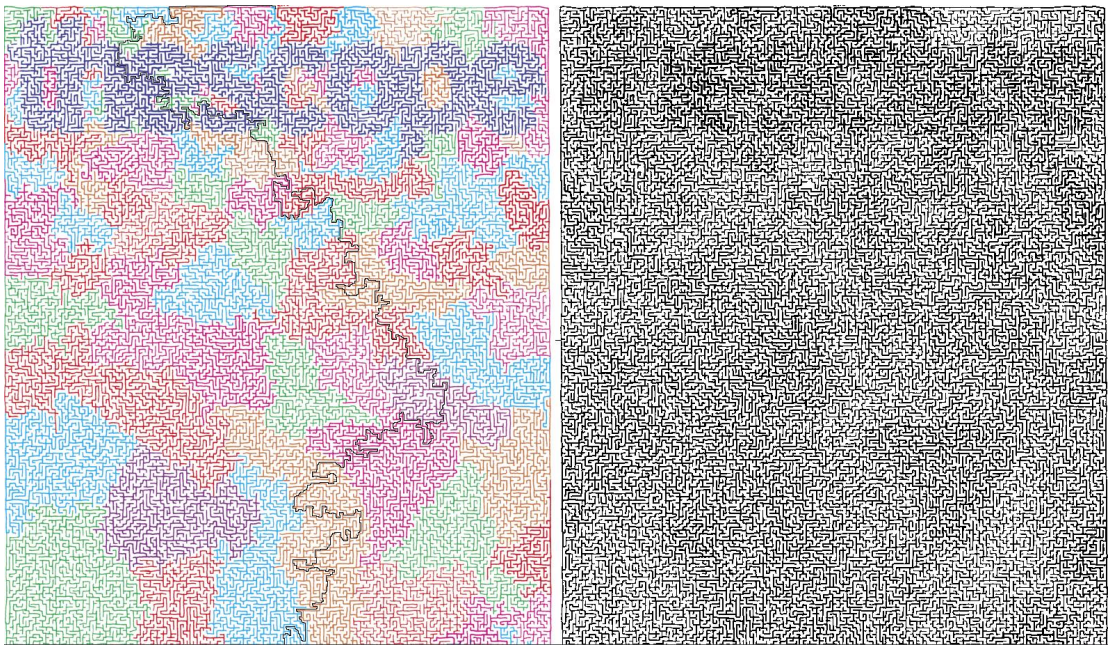
```
import numpy as np
import pylab as pl
from scipy import sparse
from scipy.sparse import csgraph

img = pl.imread("./maze.jpg")
```

```
sx, sy = (400, 979)
ex, ey = (398,  25)
bimg = np.all(img > 0.75*255, axis=2) #❶
H, W = bimg.shape

x0, x1 = np.where(bimg[H//2, :]==0)[0][[0, -1]] #❷
bimg[H//2, :x0] = 0
bimg[H//2, x1:] = 0

mask = (bimg[1:, :] & bimg[:-1, :])
idx = np.where(mask.ravel())[0]
vedge = np.c_[idx, idx + W]
pl.imsave("tmp.png", mask, cmap="gray")

#左右相邻白色像素
mask = (bimg[:, 1:] & bimg[:, :-1])
y, x = np.where(mask)
idx = y * W + x
hedge = np.c_[idx, idx + 1]

edges = np.vstack([vedge, hedge]) #❶

values = np.ones(edges.shape[0])
w = sparse.coo_matrix((values, (edges[:, 0], edges[:, 1])),  #❷
                      shape=(bimg.size, bimg.size))

startid = sy * W + sx
endid   = ey * W + ex
d, p = csgraph.dijkstra(w, indices=[startid], return_predecessors=True, dire
cted=False)

np.isinf(d[0]).sum()

path = []
node_id = endid
while True:
    path.append(node_id)
    if node_id == startid or node_id < 0:
        break
    node_id = p[0, node_id]
path = np.array(path)

x, y = path % W, path // W
img = img.copy()
```

```
img[y, x, :] = 0
fig, axes = pl.subplots(1, 2, figsize=(16, 12))
axes[0].imshow(img)
axes[1].imshow(bimg, cmap="gray")
for ax in axes:
    ax.axis("off")
fig.subplots_adjust(0, 0, 1, 1, 0, 0)
pl.show()
```

### 3.3.3 实验结果

```
dict_keys([(2, 3), (3, 3), (4, 3)])
dict_values([1.0, 2.0, 3.0])
[list([]) list([]) list([1.0]) list([3.0, 2.0]) list([]) list([]) list([])
 list([]) list([]) list([])]
[list([]) list([]) list([3]) list([2, 4]) list([]) list([]) list([])
 list([]) list([]) list([])]
==========================================
[3 4 2 3] [2 3 3 2] [ 1  2  3 10]
[[ 0  0  0  0  0  0]
 [ 0  0  0  0  0  0]
 [ 0  0  0 11  0  0]
 [ 0  0  3  0  2  0]
 [ 0  0  0  0  0  0]]
```



# 4 数据可视化

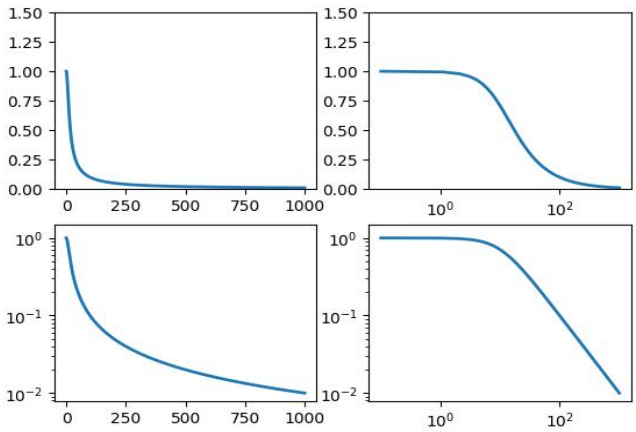
## 4.1 实验环境

本次实验在 google colab 上进行编写。

## 4.2 实验代码及其结果

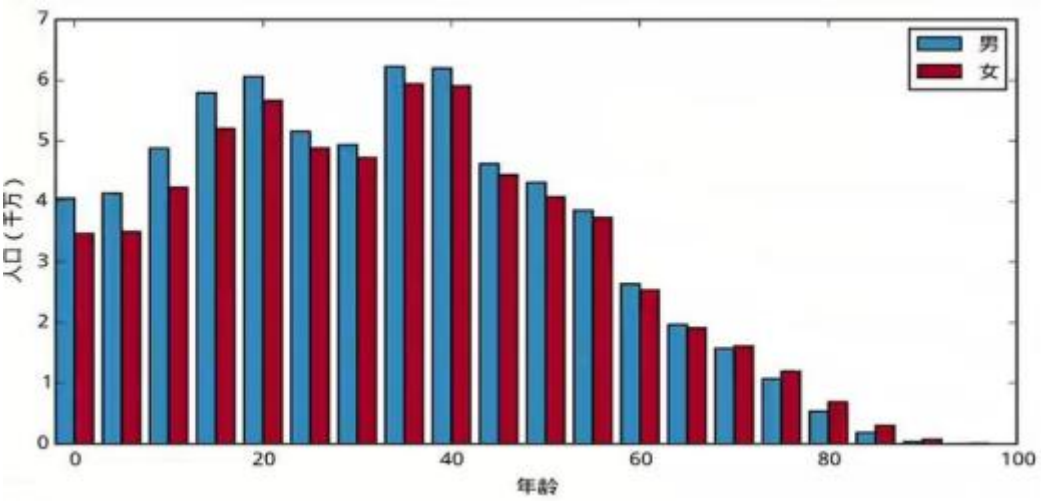
### 4.2.1 对数坐标

```
import numpy as np
from matplotlib import pyplot as plt
# ## 绘图函数简介

# ### 对数坐标图
#%fig=低通滤波器的频率响应：算术坐标（左上）、X 轴对数坐标（右上）、Y 轴对数坐标（左下）、双对数坐标（右上）
w = np.linspace(0.1, 1000, 1000)
p = np.abs(1/(1+0.1j*w)) # 计算低通滤波器的频率响应

fig, axes = plt.subplots(2, 2)
functions = ("plot", "semilogx", "semilogy", "loglog")

for ax, fname in zip(axes.ravel(), functions):
    func = getattr(ax, fname)
    func(w, p, linewidth=2)
    ax.set_ylim(0, 1.5)
plt.show()
```



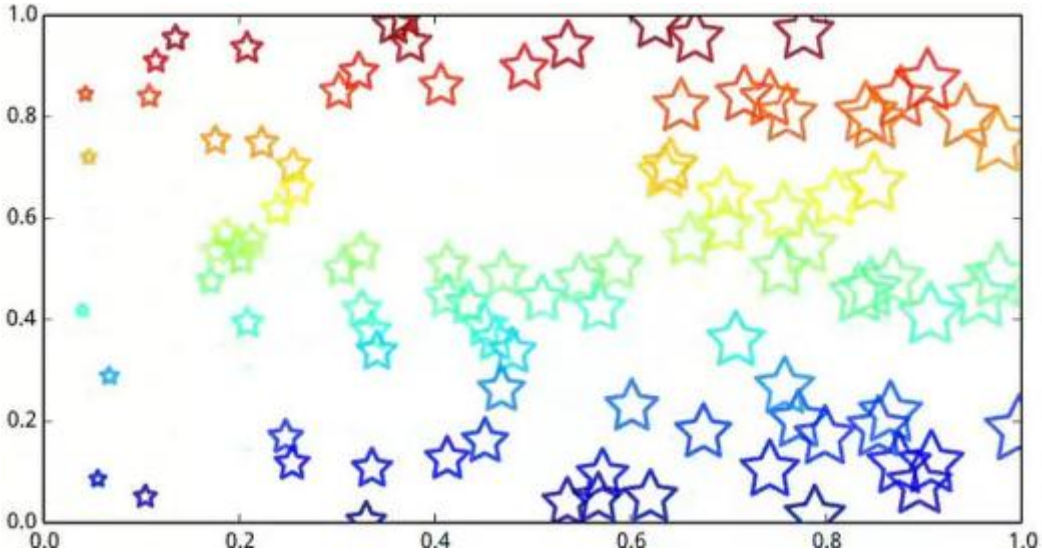

## 4.2.2 柱状图

```
1.  data = np.loadtxt("china_population.txt")
2.  width = (data[1,0] - data[0,0])*0.4 #❶
3.  plt.figure(figsize=(8, 4))
4.  plt.rcParams["font.sans-serif"] = ["Microsoft YaHei"]
5.  #c1, c2 = plt.rcParams['axes.prop_cycle'][:2]
6.  c1='red'
7.  c2='blue'
8.  plt.bar(data[:,0]-width, data[:,1]/1e7, width, color=c1, label=u"男") #❷
9.  plt.bar(data[:,0], data[:,2]/1e7, width, color=c2, label=u"女") #❸
10. plt.xlim(-width*1.5, 100)
11. plt.xlabel(u"年龄")
12. plt.ylabel(u"人口（千万）")
13. plt.legend()
14. plt.show()
```



### 4.2.3 散列图

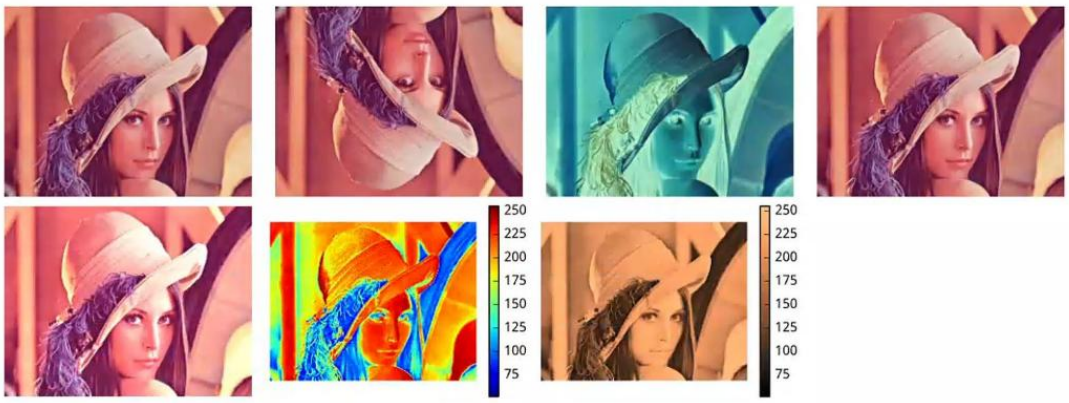
```
# ### 散列图
#%fig=可指定点的颜色和大小的散列
plt.figure(figsize=(8, 4))
x = np.random.random(100)
y = np.random.random(100)
plt.scatter(x, y, s=x*1000, c=y, marker=(5, 1),
            alpha=0.8, lw=2, facecolors="none")
plt.xlim(0, 1)
plt.ylim(0, 1)
plt.show()
```




### 4.2.4 图像

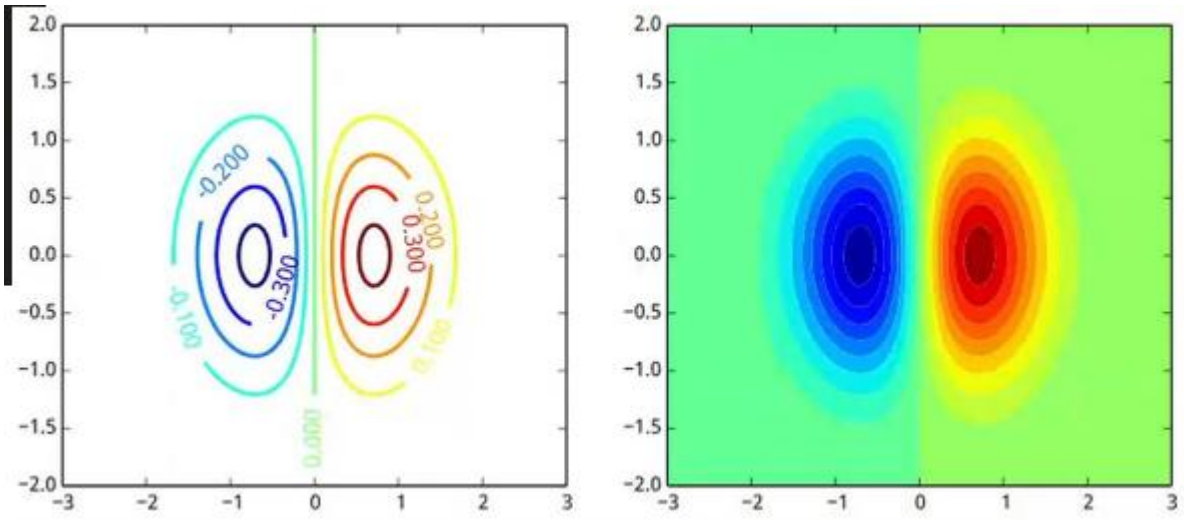
```
# ### 图像
img = plt.imread("lena.jpg")
print(img.shape, img.dtype)

#%fig=用 imread()和 imshow()显示图像
img = plt.imread("lena.jpg")
fig, axes = plt.subplots(2, 4, figsize=(11, 4))
fig.subplots_adjust(0, 0, 1, 1, 0.05, 0.05)
axes = axes.ravel()
axes[0].imshow(img)                          #❶
```

```
axes[1].imshow(img, origin="lower")         #❷
axes[2].imshow(img * 1.0)                   #❸
axes[3].imshow(img / 255.0)                 #❹
axes[4].imshow(np.clip(img / 200.0, 0, 1)) #❺

axe_img = axes[5].imshow(img[:, :, 0])      #❻
plt.colorbar(axe_img, ax=axes[5])

axe_img = axes[6].imshow(img[:, :, 0], cmap="copper") #❼
plt.colorbar(axe_img, ax=axes[6])

for ax in axes:
    ax.set_axis_off()

import matplotlib.cm as cm
cmap_names = list(cm.cmap_d.keys())
print(cmap_names[:5])

#%fig=使用 imshow()可视化二元函数
y, x = np.ogrid[-2:2:200j, -2:2:200j]
z = x * np.exp( - x**2 - y**2) #❶

extent = [np.min(x), np.max(x), np.min(y), np.max(y)] #❷

plt.figure(figsize=(10,3))
plt.subplot(121)
plt.imshow(z, extent=extent, origin="lower") #❷
plt.colorbar()
plt.subplot(122)
plt.imshow(z, extent=extent, cmap=cm.gray, origin="lower")
plt.colorbar()
plt.show()
```



### 4.2.5 等值图

```
# ### 等值线图
#%fig=用 contour(左)和 contourf(右)描绘等值线图
y, x = np.ogrid[-2:2:200j, -3:3:300j] #❶
z = x * np.exp( - x**2 - y**2)

extent = [np.min(x), np.max(x), np.min(y), np.max(y)]

plt.figure(figsize=(10,4))
plt.rcParams['font.sans-serif']=['Microsoft YaHei']
plt.subplot(121)
cs = plt.contour(z, 10, extent=extent) #❷
plt.clabel(cs) #❸
plt.subplot(122)
plt.contourf(x.reshape(-1), y.reshape(-1), z, 20) #❹;


#  **TIP**
#  如果需要对散列点数据绘制等值线图，可以先使用`scipy.interpolate`模块中提供的插值函数将散列点数据插值为网格数据。

#%fig=使用等值线绘制隐函数曲线（左），获取等值线数据并绘图（右）
y, x = np.ogrid[-1.5:1.5:200j, -1.5:1.5:200j]
f = (x**2 + y**2)**4 - (x**2 - y**2)**2

plt.figure(figsize=(9, 4))
plt.subplot(121)
extent = [np.min(x), np.max(x), np.min(y), np.max(y)]
c_s=["b", "r"]
cs = plt.contour(f, extent=extent, levels=[0, 0.1],    #❶
     colors=c_s, linestyles=["solid", "dashed"], linewidths=[2, 2])


plt.subplot(122)
i=0
for c in cs.collections: #❷
    data = c.get_paths()[0].vertices
    plt.plot(data[:,0], data[:,1],
        color=c_s[i], linewidth=c.get_linewidth()[0])
    i=i+1
plt.show()
```

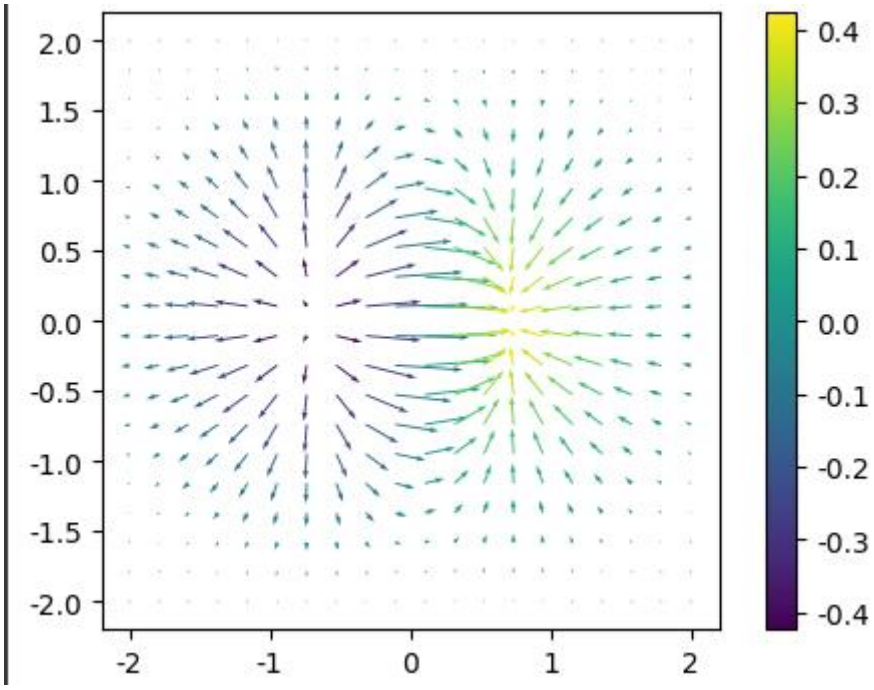


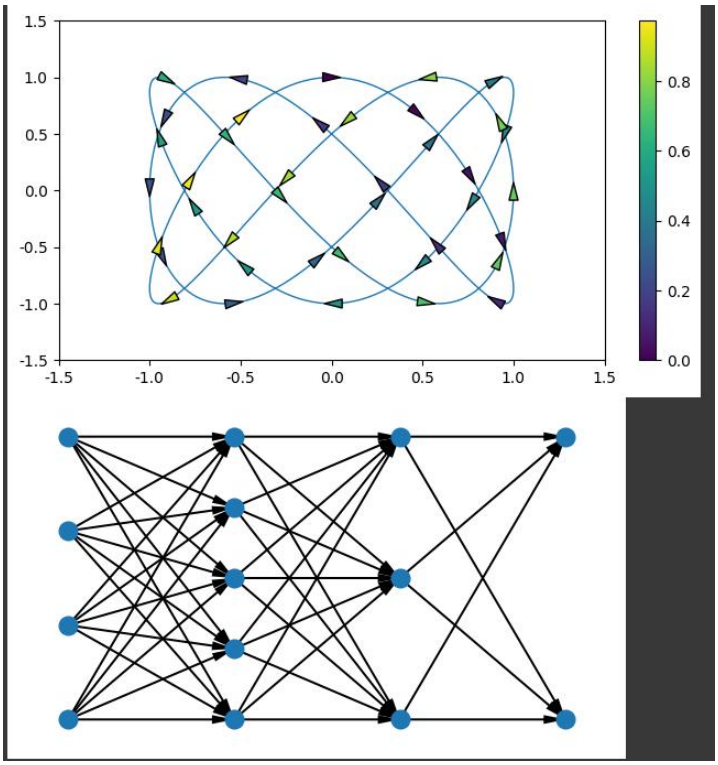

## 4.2.6 曲线图

```
# ### 箭头图
#%fig=用 quiver()绘制矢量场
def f(x, y):
    return x * np.exp(- x**2 - y**2)

def vec_field(f, x, y, dx=1e-6, dy=1e-6):
    x2 = x + dx
    y2 = y + dy
    v = f(x, y)
    vx = (f(x2, y) - v) / dx
    vy = (f(x, y2) - v) / dy
    return vx, vy
plt.figure(figsize=(6, 4))
X, Y = np.mgrid[-2:2:20j, -2:2:20j]
C = f(X, Y)
U, V = vec_field(f, X, Y)
plt.quiver(X, Y, U, V, C)
plt.colorbar();
plt.gca().set_aspect("equal")
plt.show()


#%fig=使用箭头表示参数曲线的切线方向
plt.figure(figsize=(8, 4))
n = 40
arrow_size = 16
t = np.linspace(0, 1, 1000)
```

```
x = np.sin(3*2*np.pi*t)
y = np.cos(5*2*np.pi*t)
line, = plt.plot(x, y, lw=1)

lengths = np.cumsum(np.hypot(np.diff(x), np.diff(y)))
length = lengths[-1]
arrow_locations = np.linspace(0, length, n, endpoint=False)
index = np.searchsorted(lengths, arrow_locations)
dx = x[index + 1] - x[index]
dy = y[index + 1] - y[index]
ds = np.hypot(dx, dy)
dx /= ds
dy /= ds
plt.quiver(x[index], y[index], dx, dy, t[index],
           units="dots", scale_units="dots",
           angles="xy", scale=1.0/arrow_size, pivot="middle",
           edgecolors="black", linewidths=1,
           width=1, headwidth=arrow_size*0.5,
           headlength=arrow_size, headaxislength=arrow_size,
           zorder=100)
plt.colorbar()
plt.xlim([-1.5, 1.5])
plt.ylim([-1.5, 1.5])
plt.show()

#%fig=使用 quiver()绘制神经网络结构示意图
plt.figure(figsize=(7, 4))
levels = [4, 5, 3, 2]
x = np.linspace(0, 1, len(levels))

for i in range(len(levels) - 1):
    j = i + 1
    n1, n2 = levels[i], levels[j]
    y1, y2 = np.mgrid[0:1:n1*1j, 0:1:n2*1j]
    x1 = np.full_like(y1, x[i])
    x2 = np.full_like(y2, x[j])
    plt.quiver(x1, y1, x2-x1, y2-y1,
               angles="xy", units="dots", scale_units="xy",
               scale=1, width=2, headlength=10,
               headaxislength=10, headwidth=4)

yp = np.concatenate([np.linspace(0, 1, n) for n in levels])
xp = np.repeat(x, levels)
plt.plot(xp, yp, "o", ms=12)
```

```
plt.gca().axis("off")
plt.margins(0.1, 0.1)
plt.show()
```



## 4.2.7 三维绘图

```
# ### 三维绘图
#%fig=使用 mplot3D 绘制的三维曲面图

x, y = np.mgrid[-2:2:20j, -2:2:20j] #❷
z = x * np.exp( - x**2 - y**2)

fig = plt.figure(figsize=(8, 6))
ax = plt.subplot(111, projection='3d') #❸
ax.plot_surface(x, y, z, rstride=2, cstride=1, cmap = plt.cm.Blues_r) #❹
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")
plt.show()
```