

上机2: Python数值分析

龚怡 (2167570874@qq.com)



上机2目录

- 理论教学2：Python编程基础
- 实验1：Python绘图模拟非线性方程求解过程
- 实验2：求解Hilbert系数矩阵方程组



理论教学2: Python编程基础

- 2.1 类与实例
- 2.2 迭代
- 2.3 NumPy基础数据结构
- 2.4 NumPy矩阵运算
- 2.5 NumPy线性代数
- 2.6 Python绘图



2.1 Python类与实例

- 类（class）是面向对象编程的基础
- 类由属性和方法两部分组成

定义一个类

```
class cat():
```

```
    def __init__(self, color, weight):  
        self.color = color  
        self.weight = weight
```

```
    def catch_mice(self):  
        """抓老鼠的方法"""  
        print('抓老鼠')
```

```
    def eat_mice(self):  
        """吃老鼠"""  
        print('吃老鼠')
```

__init__(self)初始化:

所有类都有

定义类包含的**属性**,
也可以在实例化类时
执行某些方法

这个类执行的**方法**,
通过**函数**来实现



类的定义与使用步骤

■ 1) 定义类

```
# 定义一个类
class cat():
    def __init__(self, color, weight):
        self.color = color
        self.weight = weight

    def catch_mice(self):
        """抓老鼠的方法"""
        print('抓老鼠')

    def eat_mice(self):
        """吃老鼠"""
        print('吃老鼠')
```

属性

方法

■ 2) 类的实例化

```
my_cat1 = cat('yellow', 10)    # 1号猫
my_cat2 = cat('black', 23)     # 2号猫
```

■ 3) 查看类的属性

```
print(my_cat1.color)
print(my_cat2.weight)
```

■ 4) 调用类的方法

```
my_cat1.catch_mice()           # 输出 抓老鼠
my_cat1.eat_mice()             # 输出 吃老鼠
```



2.2 Python中的迭代

- 迭代可以看作是循环的高效版本
- 提高编程效率和代码的可读性

```
# 列表迭代  
a = [1, 2, 3, 4, 5]
```

循环实现

```
# 循环版本  
b2 = []  
for x in a:  
    print('do some on ', x)  
    t = x + 1    #数值加 1  
    b2.append(t)  
print(b2)  
# 输出 [2, 3, 4, 5, 6]
```

迭代实现

```
def my_func(x):  
    print('do some on ', x)  
    return x + 1    #数值加 1  
  
# 迭代版本  
b = [my_func(x) for x in a]  
print(b)  
# 输出 [2, 3, 4, 5, 6]
```



2.3 NumPy基础数据结构

■ 数组 (array)

- 一维数组：行向量
- 二维数组：矩阵，或列向量
- 三维数组：在时间轴上的二维数组
- 高维数组

■ 常用代码

- 数组创建 `np.array([])`
- 显示维度 `.shape`
- 全1全0数组 `np.ones((x, y))` 和 `np.zeros((x, y))`
- 改变维度 `.ravel()` 和 `.reshape((x, y))`

*此处x和y代表行数和列数



操作实践1：编写以下代码

```
import numpy as np

# 创建一个二维数组
a = np.array([[1,2,3],
              [4,5,6]])

print('数组的维度是: ', a.shape) # 输出 (2, 3)

# 创建全是0或1的二维数组
a_one = np.ones((2,3))
print('创建全是1的数组: \n', a_one)
# 创建全是1的数组:
# [[1. 1. 1.]
#  [1. 1. 1.]]

a_zero = np.zeros((2,3))
print('创建全是0的数组: \n', a_zero)
# 创建全是0的数组:
# [[0. 0. 0.]
#  [0. 0. 0.]
```

```
# 重塑数组形状
import numpy as np
# 维度中的-1表示自动推断的意思
a = np.array([[1,2,3],
              [4,5,6]])

print('将二维数组转成一维数组', a.ravel())
# 输出 将二维数组转成一维数组 [1 2 3 4 5 6]

print('改变二维数组形状: 2*3 -> 3*2 \n', a.reshape((3,2)))
# 输出
# 改变二维数组形状: 2*3 -> 3*2
# [[1 2]
#  [3 4]
#  [5 6]]

print('将二维数组转成列向量: \n', a.reshape((-1,1)))
# 输出
# 将二维数组转成列向量:
# [[1]
#  [2]
#  [3]
#  [4]
#  [5]
#  [6]]
```




■ 常用代码

- 纵向拼接 `np.vstack([数组1, 数组2])`
- 横向拼接 `np.hstack([数组1, 数组2])`
- 提取子数组

```
a = np.array([[1, 2, 3],
              [4, 5, 6]])
b = np.array([[7, 8, 9],
              [10, 11, 12]])
```

```
print('纵向拼接: \n', np.vstack([a, b]))
# 输出
# 纵向拼接:
# [[ 1  2  3]
#  [ 4  5  6]
#  [ 7  8  9]
#  [10 11 12]]
```

```
print('横向拼接: \n', np.hstack([a, b]))
# 输出
# 横向拼接:
# [[ 1  2  3  7  8  9]
#  [ 4  5  6 10 11 12]]
```

切片, 就是截取子数组的意思

```
a = np.array([[1, 2, 3, 4, 5, 6],
              [4, 5, 6, 7, 8, 9],
              [7, 8, 9, 10, 11, 12],
              [10, 11, 12, 13, 14, 15]])
```

```
print('切取 1:3行, 2:4列的子数组: \n', a[0:3, 1:4])
```

输出 切取 1:3行, 2:4列的子数组:

```
# [[ 2  3  4]
#  [ 5  6  7]
#  [ 8  9 10]]
```

注意python数组从0位置开始

```
print('切取前3行, 后4列的子数组: \n', a[:3, -4:])
```

输出 切取前3行, 后4列的子数组:

```
# [[1 2 3]
#  [4 5 6]]
```

练习: 提取合并数组a第1,3列和数组b第2列



2.4 NumPy矩阵运算

```
import numpy as np

a1 = np.array([[4, 5, 6], [1, 2, 3]])
a2 = np.array([[6, 5, 4], [3, 2, 1]])

# 矩阵对应元素相加
print(a1 + a2)
# 输出
# [[10 10 10]
#  [ 4  4  4]]

# 矩阵对应元素相除, 如果都是整数则取商
print(a1 / a2)
# 输出
# [[0.66666667 1.          1.5         ]
#  [0.33333333 1.          3.         ]]

# 矩阵对应元素相除后取余数
print(a1 % a2)
# 输出
# [[4 0 2]
#  [1 0 0]]

# 矩阵每个元素都取n次方
print(a1 ** 3)
# 输出
# [[ 64 125 216]
#  [  1   8  27]]
```

```
# 矩阵点乘, 即对应元素相乘
print(a1 * a2)
# 输出
# [[24 25 24]
#  [ 3  4  3]]

# 矩阵点乘, 每个元素乘以一个数
print(a1 * 3)
# 输出
# [[12 15 18]
#  [ 3  6  9]]

# 矩阵相乘, (2*3) * (2*3) 是报错的, 维度不对应
# 需要先对a2转置
a3 = a2.T # 转置
print(np.dot(a1, a3)) # 矩阵相乘要用 np.dot 函数
# 输出
# [[73 28]
#  [28 10]]

# 矩阵转置
print(a1.T)
# 输出
# [[4 1]
#  [5 2]
#  [6 3]]
```

注意矩阵点乘与矩阵相乘的区别



2.4 NumPy矩阵运算

■ 矩阵求逆

```
a = np.array([[1, 2, 3], [4, 5, 6], [5, 4, 3]])
print(np.linalg.inv(a))
# 输出
# [[ 2.25179981e+15 -1.50119988e+15  7.50599938e+14]
#  [-4.50359963e+15  3.00239975e+15 -1.50119988e+15]
#  [ 2.25179981e+15 -1.50119988e+15  7.50599938e+14]]
```

■ 矩阵特征值与特征向量

```
a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
eigenValues, eigVector = np.linalg.eig(a)
# 得到三个特征值及其对应的特征向量
# 特征值是
# array([ 1.61168440e+01, -1.11684397e+00, -1.30367773e-15])
# 对应的特征向量是
# array([[ -0.23197069, -0.78583024,  0.40824829],
#        [ -0.52532209, -0.08675134, -0.81649658],
#        [ -0.8186735 ,  0.61232756,  0.40824829]])
```



2.5 NumPy求解方程组（直接法）

- 标准的线性代数方程形式为

$$Ax = b$$

$$A = \begin{bmatrix} 1 & -2 & 1 \\ 0 & 2 & -8 \\ -4 & 5 & 9 \end{bmatrix} \quad b = \begin{bmatrix} 0 \\ -8 \\ 9 \end{bmatrix}$$

```
import numpy as np

A = np.array([
    [1, -2, 1],
    [0, 2, -8],
    [-4, 5, 9]
])
B = np.array([0, -8, 9])
```

```
result = np.linalg.solve(A, B)
print('x=', result[0])
print('y=', result[1])
print('z=', result[2])
# 输出
# x= -29.0 y= -16.0 z= -3.0
```

检查答案正确性

```
print(np.allclose(np.dot(A, result), B))
```



2.6 Python绘图

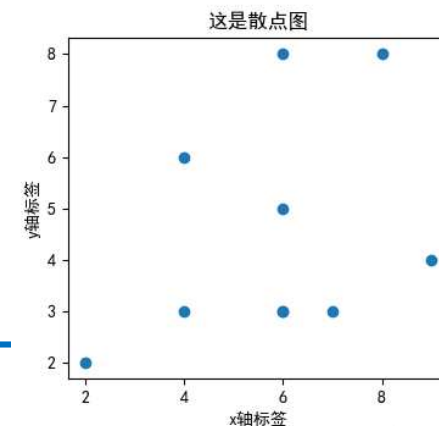
- Python科学计算的核心库——NumPy
- Python绘图库的核心库——Matplotlib
 - 知名绘图库如seaborn也是基于Matplotlib封装而来

1) 散点图 `plt.scatter(x, y)`

```
import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np

mpl.rcParams['font.sans-serif'] = ['SimHei'] # 指定默认字体
mpl.rcParams['axes.unicode_minus'] = False # 解决保存图像是负号 '-' 显示为方块的问题

# 绘制散点图
x = np.random.randint(low=2, high=10, size=10)
y = np.random.randint(low=2, high=10, size=10)
plt.scatter(x, y) # 绘制散点图
plt.title("这是散点图")
plt.xlabel("x轴标签")
plt.ylabel("y轴标签")
plt.show()
```

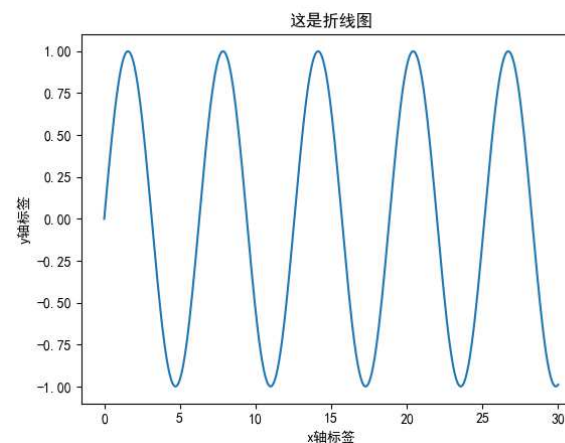




2.6 Python绘图

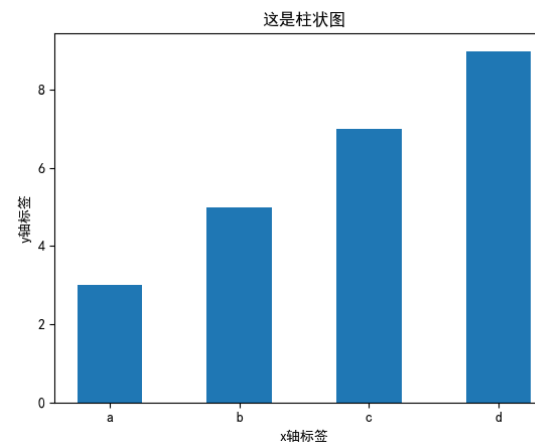
2) 折线图 `plt.plot(x, y)`

```
# 绘制折线图, 以sin函数为例
x = np.linspace(start=0, stop=30, num=300)
y = np.sin(x)
plt.plot(x, y)
plt.title("这是折线图")
plt.xlabel("x轴标签")
plt.ylabel("y轴标签")
plt.show()
```



3) 柱状图 `plt.bar(x, y)`

```
# 绘制柱状图
x = ['a', 'b', 'c', 'd']
y = [3, 5, 7, 9]
plt.bar(x, y, width=0.5)
plt.title("这是柱状图")
plt.xlabel("x轴标签")
plt.ylabel("y轴标签")
plt.show()
```

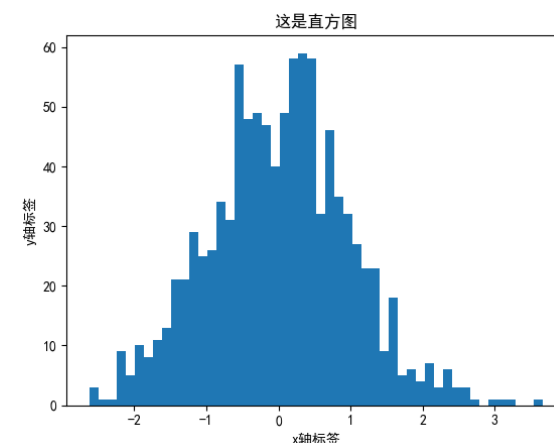




2.6 Python绘图

4) 直方图 `plt.hist(x=x, bins=n)`

```
x = np.random.normal(loc=0, scale=1, size=1000)
plt.hist(x=x, bins=50)
plt.title("这是直方图")
plt.xlabel("x轴标签")
plt.ylabel("y轴标签")
plt.show()
```



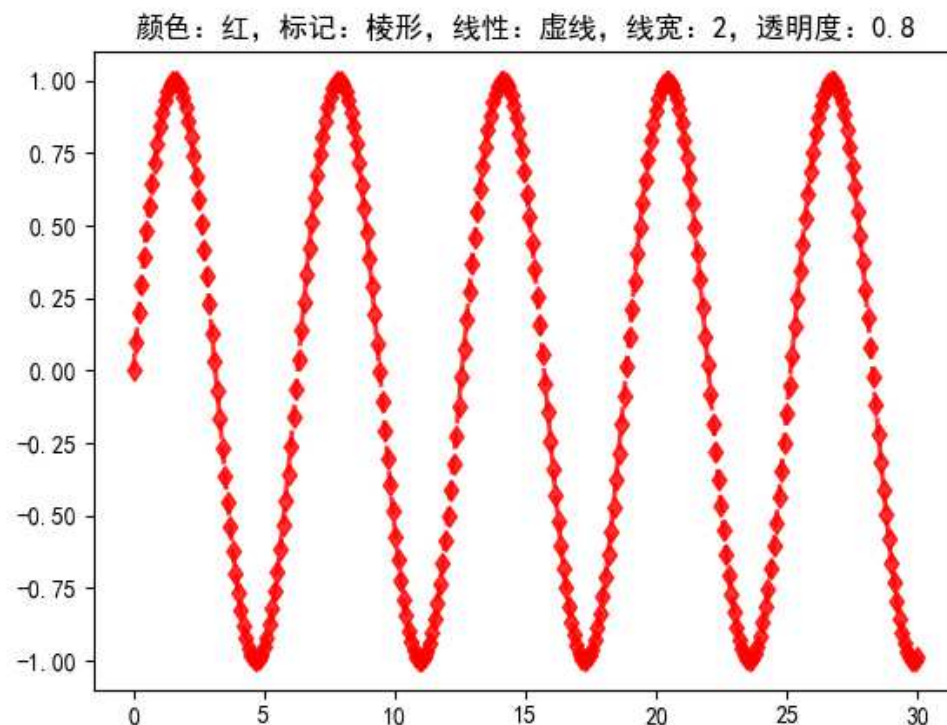
5) 图形属性

- 颜色 r-红色, b-蓝色, g-绿色, y-黄色
- o-圆圈, .-圆点, d-菱形
- -- (两横线) 虚线, - (一横线) 实线



2.6 Python绘图

```
# 绘制正弦曲线，并修改图形属性
x = np.linspace(start=0, stop=30, num=300)
y = np.sin(x)
plt.plot(x, y, color='r', marker='d', linestyle='--', linewidth=2, alpha=0.8)
plt.title('颜色: 红, 标记: 棱形, 线性: 虚线, 线宽: 2, 透明度: 0.8')
plt.show()
```





2.6 Python绘图

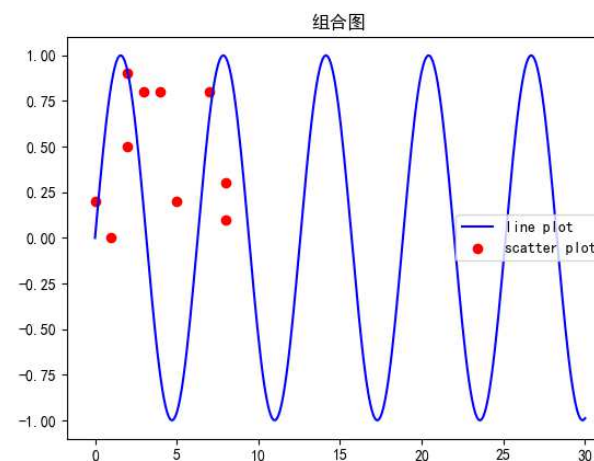
6) 组合图

画出多个图形，后面图形覆盖前面的，最后调用show函数完成。

```
# 绘制正弦曲线，并修改图形属性
x1 = np.linspace(start=0, stop=30, num=300)
y1 = np.sin(x1)
x2 = np.random.randint(low=0, high=10, size=10)
y2 = np.random.randint(low=0, high=10, size=10) / 10

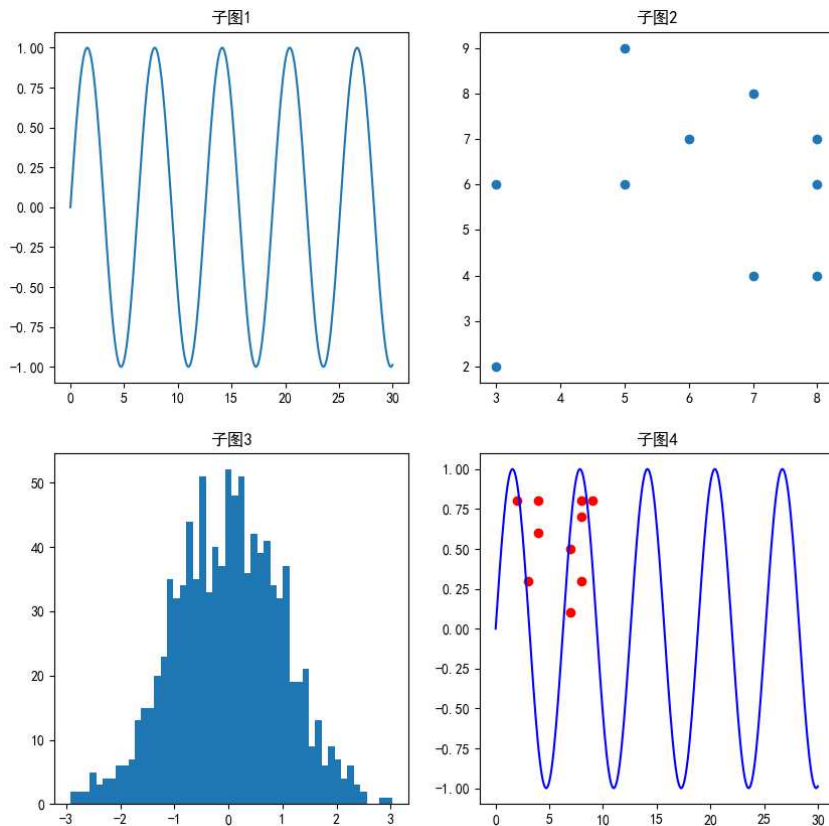
# 先绘制折线图，用蓝色
plt.plot(x1, y1, color='b', label='line plot')
# 再绘制散点图，用红色
plt.scatter(x2, y2, color='r', label='scatter plot')

plt.title("组合图")
plt.legend(loc='best') # 显示图例
plt.show()
```



2.6 Python绘图

7) 子图 `axi.plot()`



```
fig = plt.figure(figsize=(10, 10)) # 指定画布大小
```

```
ax1 = fig.add_subplot(2, 2, 1) # 添加一个子图, 返回子图句柄  
ax2 = fig.add_subplot(2, 2, 2)  
ax3 = fig.add_subplot(2, 2, 3)  
ax4 = fig.add_subplot(2, 2, 4)
```

```
# 子图1绘制sin图形
```

```
x = np.linspace(start=0, stop=30, num=300)  
y = np.sin(x)  
ax1.plot(x, y)  
ax1.set_title('子图1')
```

```
# 子图2绘制散点图
```

```
x = np.random.randint(low=2, high=10, size=10)  
y = np.random.randint(low=2, high=10, size=10)  
ax2.scatter(x, y) # 绘制散点图  
ax2.set_title('子图2')
```

```
# 子图3绘制直方图
```

```
x = np.random.normal(loc=0, scale=1, size=1000)  
ax3.hist(x=x, bins=50)  
ax3.set_title('子图3')
```

```
# 子图4绘制组合图
```

```
x1 = np.linspace(start=0, stop=30, num=300)  
y1 = np.sin(x1)  
x2 = np.random.randint(low=0, high=10, size=10)  
y2 = np.random.randint(low=0, high=10, size=10) / 10
```

```
# 绘制组合图
```

```
ax4.plot(x1, y1, color='b', label='line plot')  
ax4.scatter(x2, y2, color='r', label='scatter plot')  
ax4.set_title('子图4')
```

```
# 最后显示图形
```

```
plt.show()
```



2.6 Python绘图

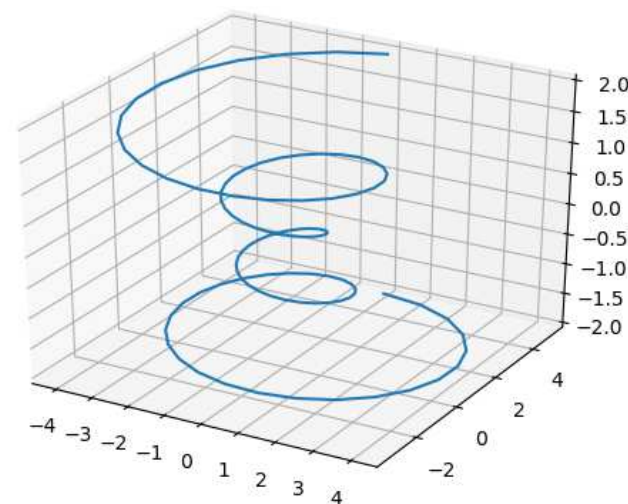
8) 三维曲线图 调入库mpl_toolkits.mplot3d

```
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as plt

# 生成画布
fig = plt.figure()
ax = fig.gca(projection='3d') # 指定为3D图形

# 生成(x,y,z)数据
theta = np.linspace(-4 * np.pi, 4 * np.pi, 100)
z = np.linspace(-2, 2, 100)
r = z ** 2 + 1
x = r * np.sin(theta)
y = r * np.cos(theta)

# 绘制图形
ax.plot(x, y, z) # 曲线图和2D一样使用plot函数
plt.show()
```





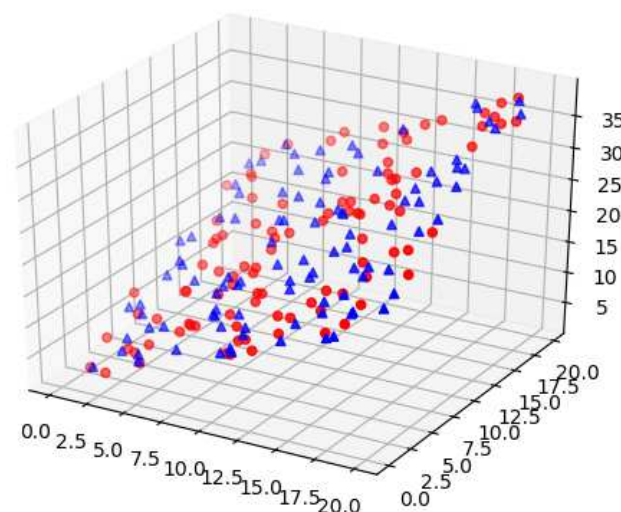
2.6 Python绘图

9) 三维散点图 调入库mpl_toolkits.mplot3d

```
# 绘制红色点100个
x1 = np.random.random(100) * 20
y1 = np.random.random(100) * 20
z1 = x1 + y1
ax.scatter(x1, y1, z1, c='r', marker='o')

# 绘制蓝色点100个
x2 = np.random.random(100) * 20
y2 = np.random.random(100) * 20
z2 = x2 + y2
ax.scatter(x2, y2, z2, c='b', marker='^')

plt.show()
```



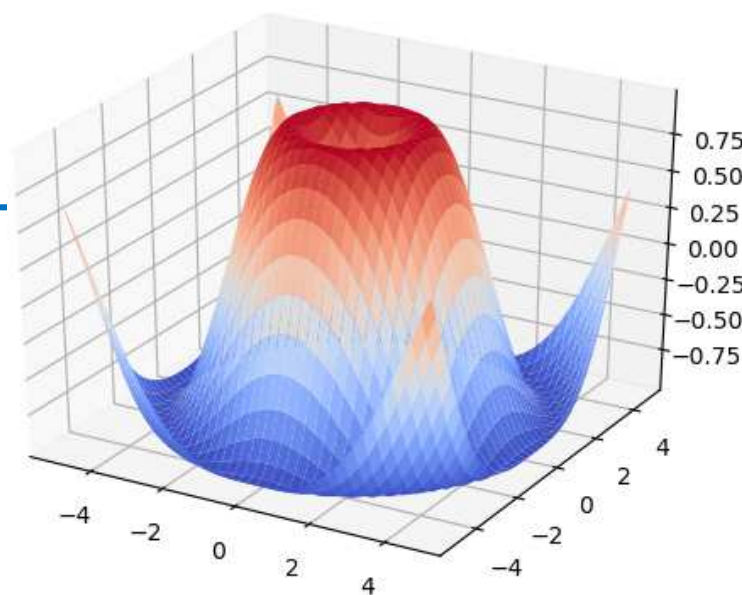


2.6 Python绘图

10) 三维曲面图 调入库mpl_toolkits.mplot3d

```
# 生成数据(x,y,z)
x = np.arange(-5, 5, 0.25)
y = np.arange(-5, 5, 0.25)
x, y = np.meshgrid(x, y) # 重点, 用np.meshgrid生成坐标网格矩阵
z = np.sin(np.sqrt(x ** 2 + y ** 2))

# 使用plot_surface函数
# cmap=cm.coolwarm 是颜色属性
surf = ax.plot_surface(x, y, z, cmap=cm.coolwarm)
plt.show()
```





2.6 Python绘图

11) 动态图 调入库animation

- ❑ 假设一个场景有3辆卡车，在一个广场上行驶。用动态图画出3辆卡车的位置。
- ❑ 卡车类由3个属性，x和y表示其位置，marker表示其形状

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation

# 这是一个卡车类
class car():
    def __init__(self, color):
        self.x = 1
        self.y = 1
        self.color = color

    def move(self):
        """在东南西北四个方向随机选一个方向走一步，然后更新坐标"""
        # 随机移动一步
        self.x = self.x + np.random.randint(low=-1, high=2, size=1)[0]
        self.y = self.y + np.random.randint(low=-1, high=2, size=1)[0]
        # 防止越界
        self.x = self.x if self.x > 0 else 0
        self.x = self.x if self.x < 10 else 10
        self.y = self.y if self.y > 0 else 0
        self.y = self.y if self.y < 10 else 10
```

定义卡车类



2.6 Python绘图

□ 实例化3辆车

```
# 实例化3辆车  
cars = [car(color='r'), car(color='b'), car(color='g')]
```

□ 模拟1000个时间点，在每个时间点操作图形对象

```
i = list(range(1, 1000)) # 模拟1000个时间点  
  
# update 是核心函数，在每个时间点操作图形对象  
def update(i):  
    plt.clf() # 清空图层  
    # 对每辆卡车进行操作  
    for c in cars:  
        c.move() # 移动1步  
        x = c.x  
        y = c.y  
        color = c.color  
        plt.xlim(0, 10) # 限制图形区域  
        plt.ylim(0, 10)  
        plt.scatter(x, y, color=color) # 绘制卡车  
    return
```



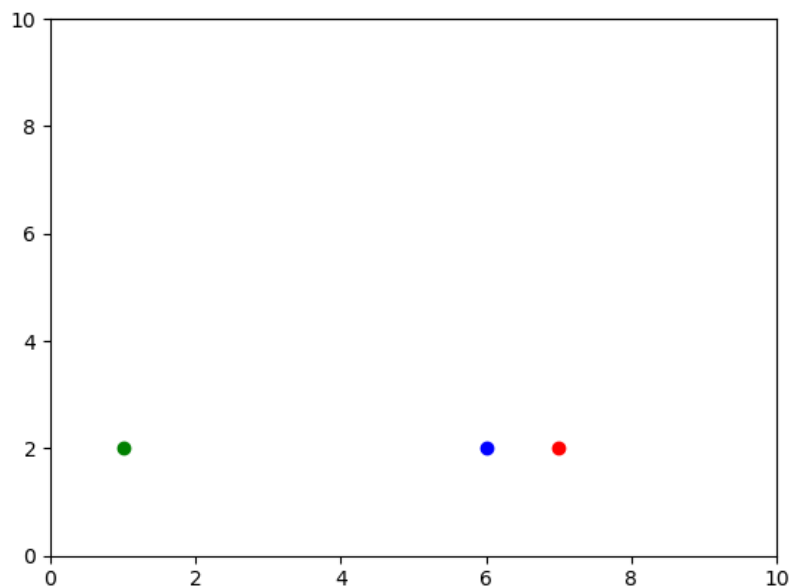
2.6 Python绘图

- 绘制一张画布

```
fig = plt.figure()
```

- 绘制动态图

```
ani = animation.FuncAnimation(fig, update)  
plt.show()
```





上机2目录

- 理论教学2: Python编程基础
- 实验1: Python绘图模拟非线性方程求解过程
- 实验2: 求解Hilbert系数矩阵方程组



实验1：Python绘图模拟非线性方程求解过程

■ 实验目的：

通过动态图显示求解迭代过程。

■ 实验要求：

对不动点迭代法fixPointFig.py进行修改，使得

- 1) 图形坐标根据迭代范围缩小自动调整(用plt.xlim, ylim, 在动图程序中有)；
- 2) 在图像上显示迭代序号，和对应迭代得到的值(通过修改title输出实现)



实验2：求解Hilbert系数矩阵方程组

■ 实验目的：

比较直接法和迭代法求解病态方程组的特点。

■ 实验要求：

系数矩阵为Hilbert阵，对解全为1的方程组，随着 $n = 2, 3, \dots$ 的增加，编写程序，测试和分析利用直接法和迭代法求解方程组的结果差别。

$$\text{Hilbert 阵 } H_n = \begin{bmatrix} 1 & \frac{1}{2} & \cdots & \frac{1}{n} \\ \frac{1}{2} & \frac{1}{3} & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{n} & \frac{1}{n+1} & \cdots & \frac{1}{2n-1} \end{bmatrix}$$



上机课任务

- 完成实验1和实验2内容，
写到“上机2实验报告.docx”，
在CG系统提交“上机2实验报告.pdf”文档。



参考文献

- 《Python最优化算法实战——从推公式到写代码》