

电子系统导论实验报告

实验 1 【实验名】



指导教师: 万景

学生姓名: 彭堃

学 号: 22307110109

专 业: 保密技术

日 期: 2024. 3. 7

一、实验目的:

学习Python的多平台安装, IDE的选择与使用; python第三方库的安装方法; python的基本语法; python模块的导入和函数引入, matplotlib库的使用方法; 程序异常处理和调试的方法。以及numpy,pandas数据处理库和GUI相关库的介绍; python在芯片自动设计系统中的应用。

二、实验原理:

Python3相关教程内容; matplotlib等库的官方文档。

三、实验内容:

1.python 安装及 IDE

由于本人之前以及学习过 python, 且在 ICS 中使用 python 完成了课程项目的 GUI 设计, 所以本人的 python 及 IDE 环境已经配置好了

Windows 下

```
PS C:\Users\GodKe> python
Python 3.10.11 (tags/v3.10.11:7d4cc5a, Apr 5 2023, 00:38:17) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

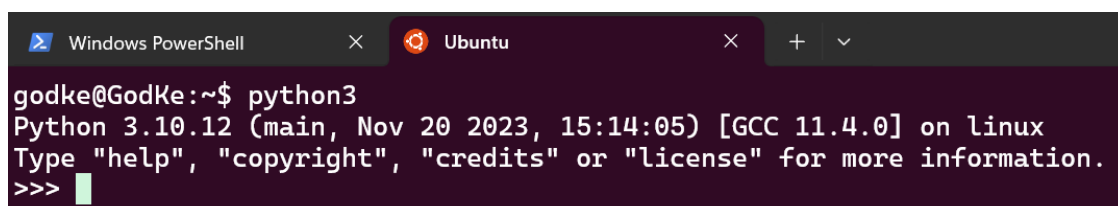
安装的是当时最新的 Python3.10, 这个版本最新支持了类 switch 的语法——match-case

使用 vscode 安装 python 插件即可实现 python 代码的运行 (自动配置) 而 Python 的调试则通过编写 launch.json 实现

```
{
    "name": "Python 调试程序: 当前文件",
    "type": "debugpy",
    "request": "launch",
    "program": "${file}",
    "console": "integratedTerminal"
},
```

由于本人之前是在 Linux 下开发, 所以已经习惯用 pip 命令了

Linux 下



```
godke@GodKe:~$ python3
Python 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Ubuntu 是自带 python 的, 但是依稀记得这个 python 版本是我主动升级的

由于是通过微软提供的 WSL2 (Windows Subsystem for Linux) 安装的 Linux, 所以可以直接通过 vscode 的远程开发组件实现在 Windows 上的 vscode 里编写和运行 Linux 上的代码

2.Python 基础语法

本人已经学过了, 就不扯了

2022-2023 1	COMP110042	COMP110042.0 7	Python程序设计	2	A-	3.7
-------------	------------	-------------------	------------	---	----	-----

3.matplotlib 库 (个人学习笔记、可以跳过不看)

Ctrl+点击跳过

本文由 markdown 文件转换而来, 要看建议直接看转出来的 pdf 文件

写在前面

python 最好的点在于: 当你乱输入参数时, 它会在报错里建议你输入一些参数所以尽情地尝试吧

matplotlib 对中文的支持极差, 懒得折腾就别用中文

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

plt.show() # 把你设计的图像显示出来（不然只是一个对象）
复制
```

Figure

Figure（图像）即程序运行后展现出的窗口，是 matplotlib 中的基本结构 Figure 具有默认参数，可以不进行设置

参数	默认值	描述
num	1	图像的数量
figsize	figure.figsize	图像的长和宽（英寸）
dpi	figure.dpi	分辨率（点/英寸）
facecolor	figure.facecolor	绘图区域的背景颜色
edgecolor	figure.edgecolor	绘图区域边缘的颜色
frameon	True	是否绘制图像边缘

plot(subplot)

plot

plot(图样)是 figure 上的元素，plot 方法的格式为

```
plt.plot(*args:ArrayLike, #接受多个类列表参数（默认第一个为 X 轴）
        scalex: bool = True, #x 轴自动调节
        scaley: bool = True, #y 轴自动调节
        data: Any | None = None, #可选择图的类型，如 ob 为散点图
        **Kwargs: Any #可选择的参数
) -> list[Line2D]
```

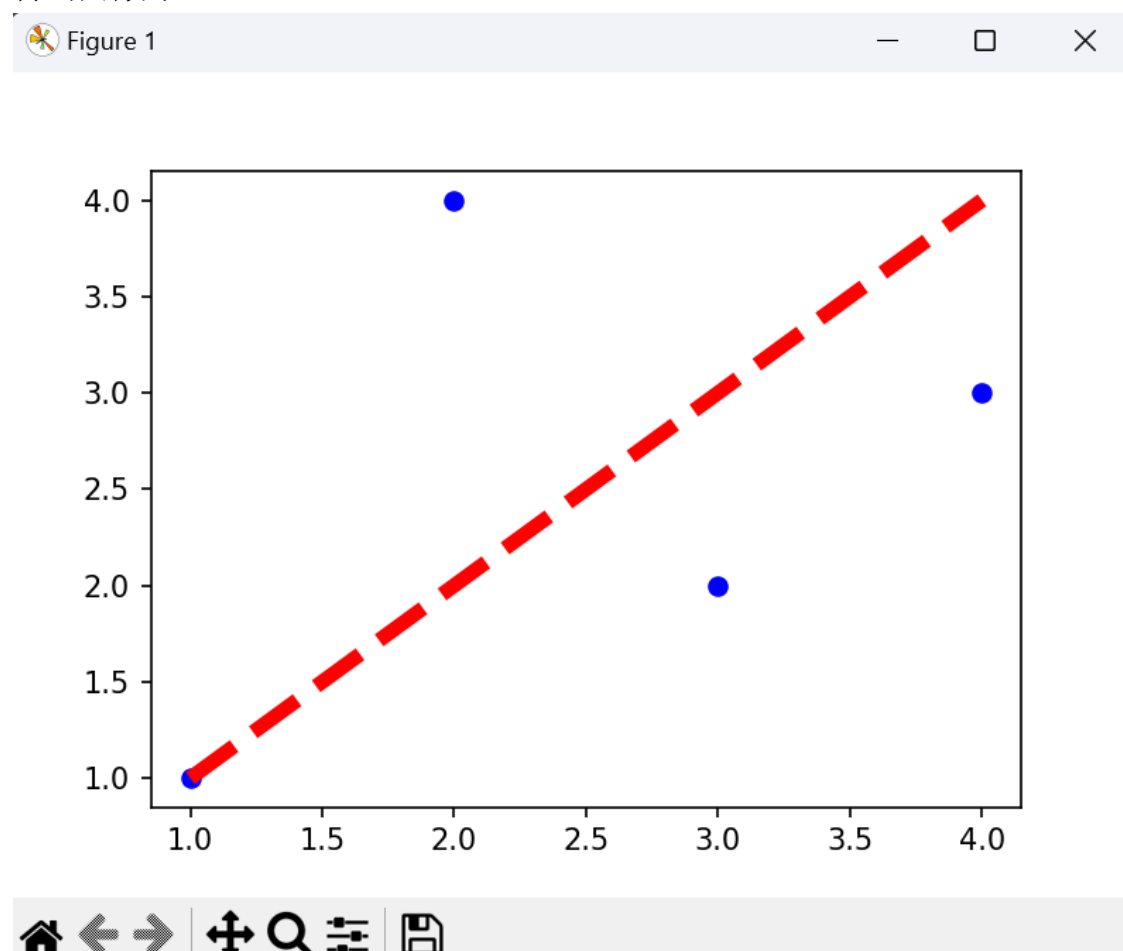
kwargs 包括: label="图像的标签"; color="图像的颜色，支持英文";linewidth=int(线的粗细); linestyle='线的风格，如'-'','--'等

复制

比如我们可以简单绘制一个折线图 and 散点图

```
plt.plot([1,2,3,4],[1,4,2,3],"ob",color="blue")  
plt.plot([1,2,3,4],[1,2,3,4],color="red",linewidth=5,linestyle='--')  
plt.show()  
复制
```

得到图像为



但是反复在一个 plot 里画图可能导致线互相重叠严重, 因为使用 plot 方法只有一个图像

subplot

subplot 可以在一个 figure 里创建多个图像, 但是需要自行设计分布, 方法为:

```
plt.subplot(nrows,ncols,index) # 创建 nrows 行, ncols 列的图像分布, 指定  
index  
默认为 plt.subplot(1,1,1)  
复制
```

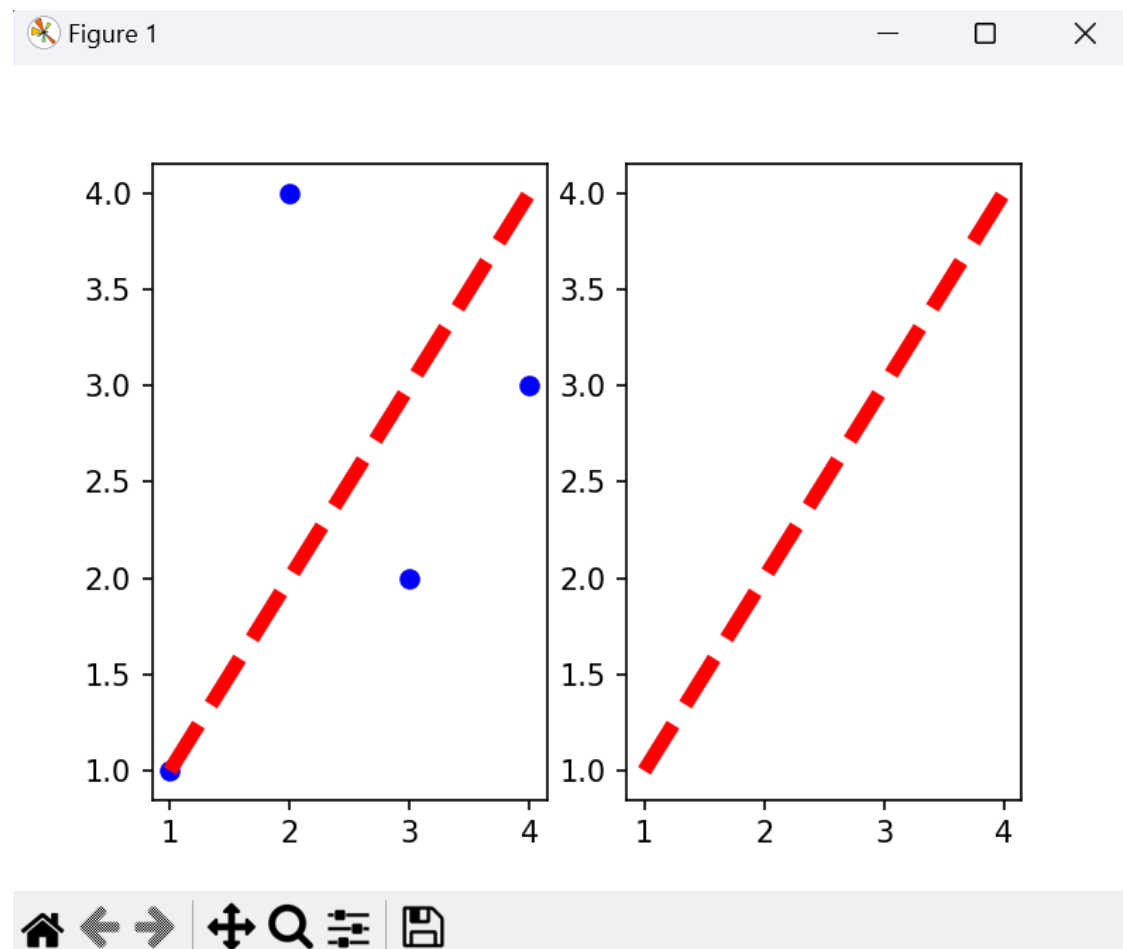
subplot()方法本质上是用来定位图像位置的, 在宣布完 index 后, 后续 plot 函数产生的图像都将展示在指定的图像上, 直到再次宣布 subplot 的 index

一个简单的例子

```
plt.subplot(1,2,1) # 宣布一个一行两列的图像分布，并指定当前绘制在第一个上
plt.plot([1,2,3,4],[1,4,2,3],"ob",color="blue")
plt.plot([1,2,3,4],[1,2,3,4],color="red",linewidth=5,linestyle='--')
plt.subplot(1,2,2) # 重新宣布绘制在第二个上
plt.plot([1,2,3,4],[1,2,3,4],color="red",linewidth=5,linestyle='--')
plt.show()
```

复制

得到的图像为



一些调整用的方法

针对单个图像的调整

就如 `subplot` 章节所言，我们调整某个图像或者往某个图像上画图时，都要指定是哪个图像，否则就会指定到默认图像上（虽然也无所谓）

标题

```
plt.title("标题", loc="位置, 支持英文")  
# 设置 plot 的标题, 如果你使用了 subplot, 那么你可以为每个子图设置一个标题  
# 并且可以在指定 subplot 前设置一次标题, 这个标题可以视为 figure 的大标题  
复制
```

坐标轴调整

```
plt.xlim(left, right) # 设置横轴的上下限  
plt.xticks(Array) # 设置横轴记号, 所有在列表中的点都会被标出  
plt.xlabel("string", loc="位置") # 设置 x 轴名称和位置  
plt.ylim(left, right) # 设置纵轴的上下限  
plt.yticks(Array) # 设置纵轴记号, 同上  
plt.ylabel("string", loc="位置") # 设置 y 轴名称和位置  
复制
```

注意: 当你设置了上下限时, `plot` 的 `xscale` 和 `yscale` 将不再工作
这意味着如果你输入的数据点超出了你自己设置的范围, 它将不再被显示

加强版坐标轴记号

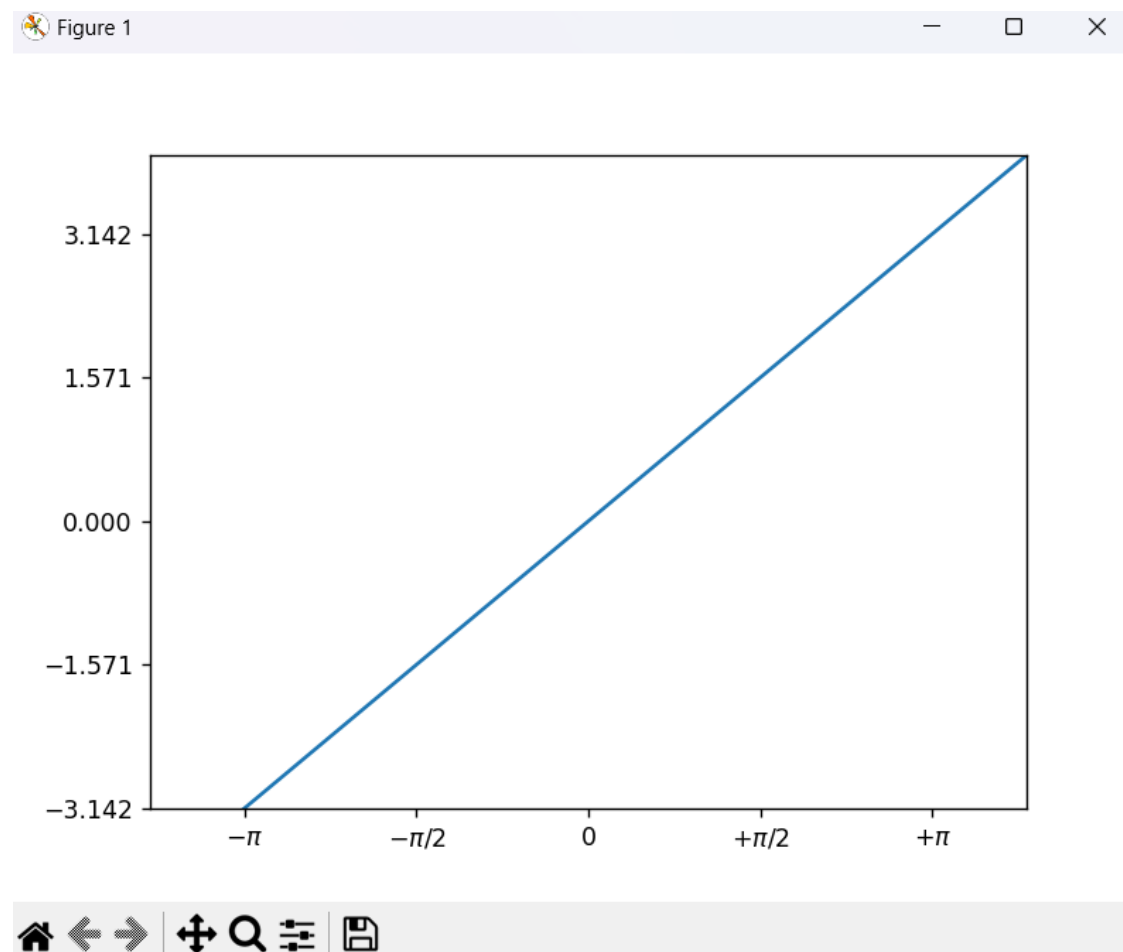
在使用的时候你会难受的发现记号方法用的是浮点数, 连分数都是转化成小数显示的, 还有精度

这实在是无法接受, 标个 π 都成了 3.142

于是我们可以使用 LaTeX 的语法实现给每个点取名字(只是取名字, 位置还是小数的位置)

```
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],  
           [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])  
plt.yticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi])  
复制
```

对比一下坐标轴的显示



这下舒服了

移动脊柱（Spines）——创建平面直角坐标系

- 坐标轴线和上面的记号连在一起就形成了脊柱（Spines），它记录了数据区域的范围。它们可以放在任意位置，默认情况下它们被放在图的四边。
- 实际上每幅图有四条脊柱（上下左右），如果要将其放在图的中间，我们必须将其中的两条（上和右）设置为无色，然后调整剩下的两条到合适的位置——数据空间的 0 点。

步骤很简单

```
[ax = plt.gca()](<plt.plot([-4,-3,-2,-1,1,2,3,4],[-4,-3,-2,-1,1,2,3,4])
```

```
ax = plt.gca() # 获取脊柱对象
```

```
ax.spines['right'].set_color('none') # 右脊柱设为无色
```

```
ax.spines['top'].set_color('none') # 上脊柱设为无色
```

```
ax.xaxis.set_ticks_position('bottom') # 调整 x 轴的记号位置
```

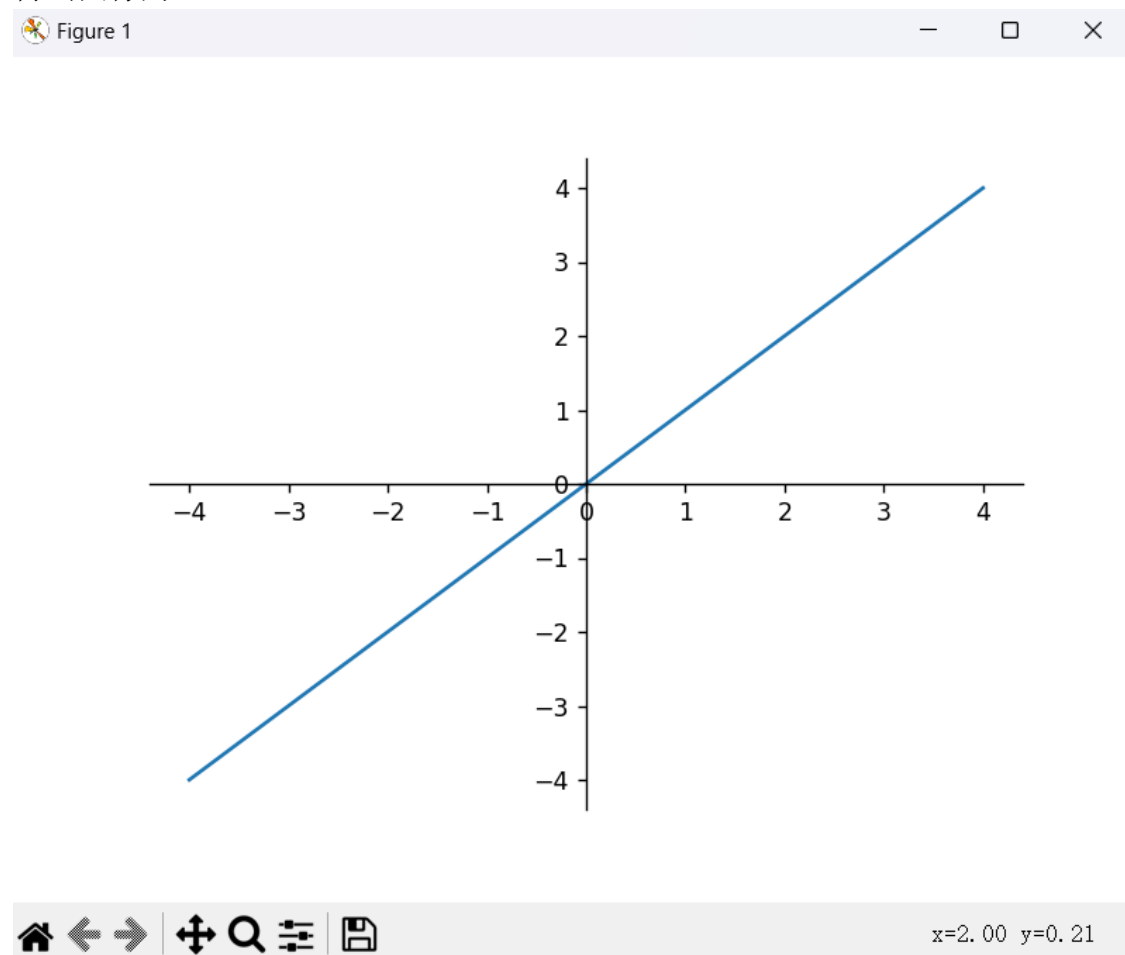
```
ax.spines['bottom'].set_position(('data',0)) # 调整下脊柱的位置
```

```
ax.yaxis.set_ticks_position('left')
```



```
ax.spines['left'].set_position(('data',0))  
plt.show()>  
# 位置参数与四个脊柱的代号相同  
复制
```

得到图像为



实际上通过 `gca()` 获取到的对象可以对坐标轴线进行大幅度的改造

添加图例

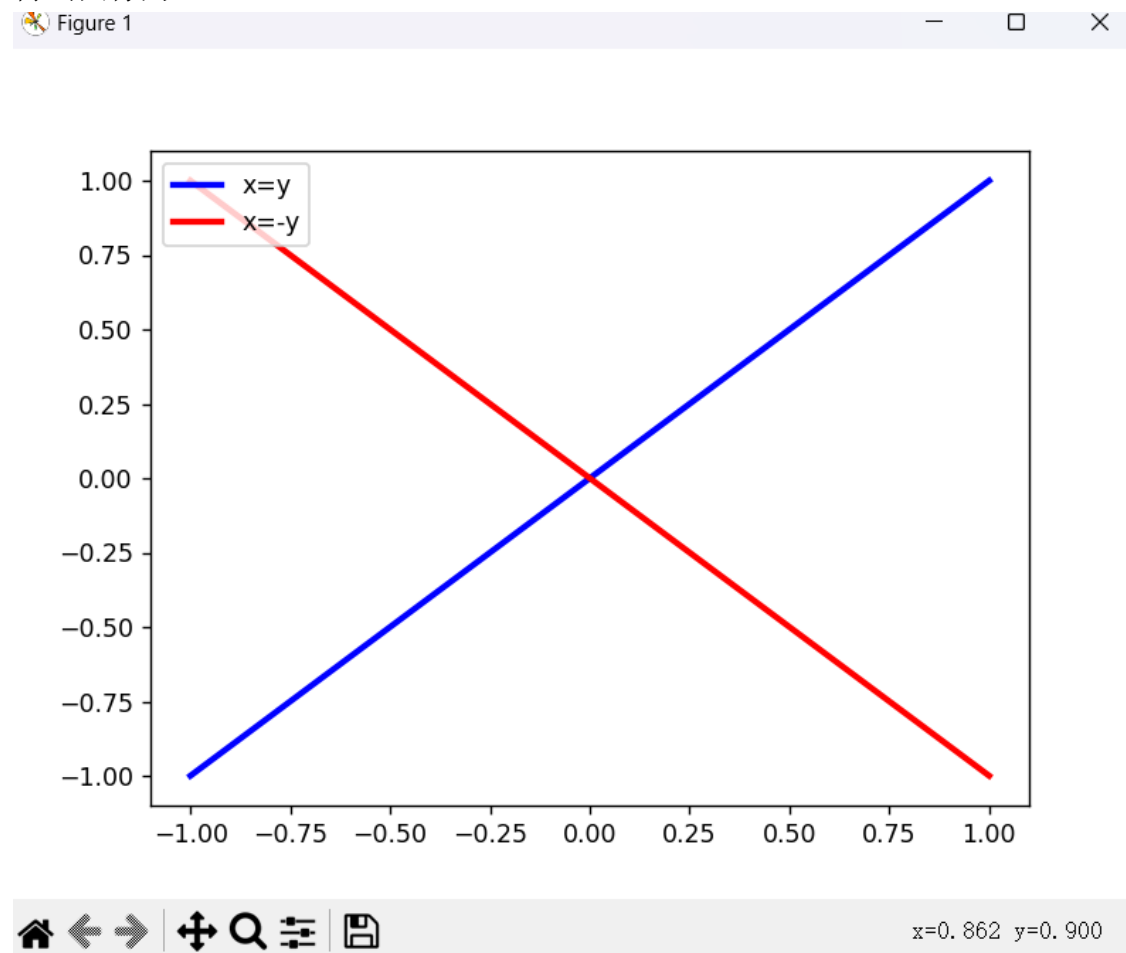
如果你在使用 `plot` 方法时设置了 `label` 参数，那么你就可以通过 `legend` 方法创建一个图例来展示 `label`

```
# 一个简单的例子  
plt.plot([-1,0,1], [-1,0,1], color="blue", linewidth=2.5, linestyle="-", label="x=y")  
plt.plot([-1,0,1], [1,0,-1], color="red", linewidth=2.5, linestyle="-", label="x=-y")  
plt.legend(loc='upper left') # loc(ation)参数可指定位置，可以用 best 参数自动抉择
```

```
plt.show()
```

复制

得到图像为



其他图样类型

matplotlib 支持非常多种类的图样，可以自行探索

4.课堂小实验

用 numpy 是用 python 的基本素养

画 Sin 和 Cos 曲线

简单写一段

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

```
X = np.linspace(0,2*np.pi,256) #linspace 方法生成区间内等距点的坐标列表  
S = np.sin(X) # 直接调用 numpy 里的方法就行  
C = np.cos(X)
```

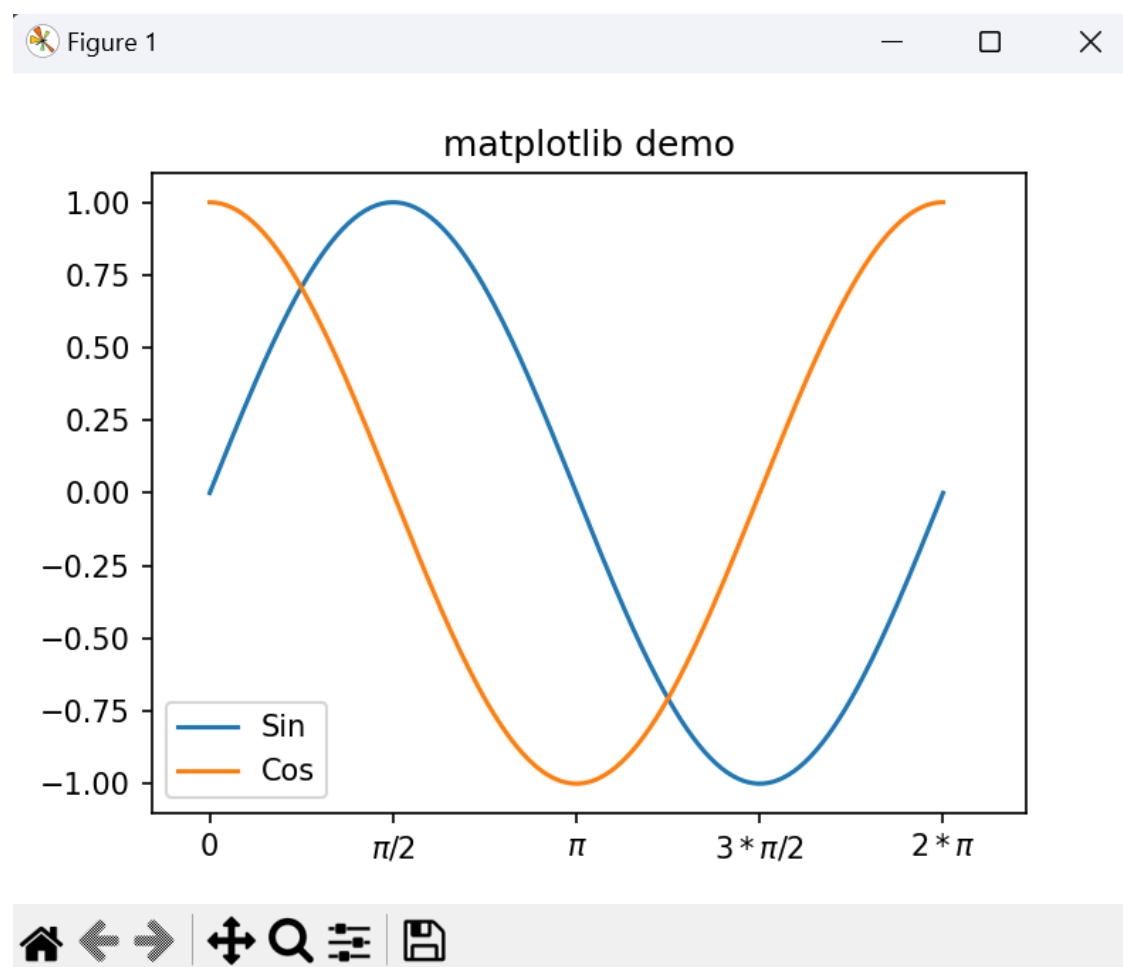
```
plt.title("matplotlib demo") # 设置标题  
plt.xlim(-0.5,7) # 设计 x 轴  
plt.xticks([0,np.pi/2,np.pi,3*np.pi/2,2*np.pi],[0,r'$\pi/2$',r'$\pi$'  
$,r'$3*\pi/2$',r'$2*\pi$'])
```

```
plt.plot(X,S,label="Sin") # 画 Sin 曲线  
plt.plot(X,C,label="Cos") # 画 Cos 曲线  
plt.legend(loc="lower left") # 设置一个 legend
```

```
plt.show() # show
```

复制

得到图像



5.课后作业

其实在课上就把差分法写出来了, 后面解析式法捣鼓了半天, 高数太差过程都写注释里了 ([建议直接看源代码](#))

```
# 1kg 重的弹性小球高处坠落, 已知初始高度为 100m, 初速度为 0m/s
# 重力加速度为 10m/s^2, 风阻系数为 r=0.1, 且小球碰撞地面后以原速反弹 (完全弹性碰撞)
# 利用 python 求解出小球在总世界 20s 内, 高度随时间变化的轨迹

#设置向上为正方向, 速度设为矢量, 此时的加速度计算公式
# a = -g - v*r/m

import numpy as np
import matplotlib.pyplot as plt

m = 1          #质量为 1kg
g = 10         #重力加速度为 10m/s^2
r = 0.1        #风阻系数

v = [0]        #初速度列表
h = [100]      #高度列表
t = [0]        #时间列表
n = 1000       #可调整迭代次数
dt = 20.0 / n  #计算 dt

#差分法
#h[n+1] = h[n] + v * dt
#v[n+1] = v[n] + a * dt

for i in range(n): #本质上就是根据第 i 个数据把第 i+1 个数据 append 进去
    if (h[i] < 0 and v[i] < 0): #检查高度, 如果小于 0 了就让速度反向
        v[i] = -v[i]
    a = -g - v[i] * r / m #即时计算加速度
    v.append(v[i] + a * dt) #计算各个数据并 append
    h.append(h[i] + v[i] * dt)
    t.append(t[i] + dt)

#也可以尝试直接解析
#dv/dt = -g - v*r/m -> -dt = dv/(g+v*r/m) -> -t = ln(g+v*r/m)/(r/m)+C
#g+v*r/m = e^-((t+C)r/m) -> 10 + 0.1v = e^-0.1(t+C) -> v = 10*e^-0.1
(t+C) - 100
#代入初始值 t=0,v=0 得到 C=-lng/(r/m) -> -t = ln(1+v*r/(m*g))/(r/m) -> 1
+ v*r/(m*g) = e^-(t*r/m)
```

```
#v = m*g/r(e^-(t*r/m)-1) = 100(e^(-0.1*t)-1) = 100*e^(-0.1*t)-100
#h = -e(-0.1*t)*1000 - 100t + C
#代入初值 t=0,h=100,得到 C = 1100 -> h = -1000*e(-0.1*t) - 100t + 1100
#考虑反弹, 第一个方程将运行到 h = 0, 之后 v 反弹, 需要先重设速度方程的常数
C, 再重设 h 的方程常数 C
#由 v = 10*e^-0.1(t+C) - 100 再积分, 得到 h = -100*e^-0.1(t+C1) - 100t
+ C2

Cv = -np.log(10) * 10    #初始速度方程常量
Ch = 1100    #初始高度方程常量

def tTov(t,v,flag): #速度方程抽象为一个函数, 且根据 flag 动态调整 global
的常量 Cv
    global Cv
    if flag:
        Cv = -t - np.log(10 + v*0.1) * 10
    return (10*np.e ** (-0.1*(t+Cv)) - 100)

def tToh(t,h,flag): #高度方程同样抽象, 并更新常量 Ch
    global Cv,Ch
    if flag:
        Ch = h + 100*t + 100*np.e ** (-0.1*(t+Cv))
    return (Ch - 100*t - 100*np.e ** (-0.1*(t+Cv)))

flag = 0    #flag 表征是否落地
T = t    #时间表复制一份差分法的, 这样坐标系可以对准
H = [100]    #高度列表
V = [0]    #速度列表
for i in range(n): #本质上是根据解析方程代入每个 t 计算, 但是因为常量更
新, 所以终归还是要循环
    if (H[i] < 0 and V[i] < 0): #检查是否落地
        V[i] = -V[i] #依旧速度反向
        flag = 1    #设置 flag
    V.append(tTov(T[i],V[i],flag)) #这里顺序很重要, 必须先更新 Cv 才能去
更新 Ch
    H.append(tToh(T[i],H[i],flag))
    flag = 0    #重设 flag

#创建图像
def init(): #设计一个初始化函数
    plt.xlabel("time:",loc="left") #轴名称和位置设置一下
    plt.ylabel("height:",loc="bottom")
    ax = plt.gca()
```

```
ax.xaxis.set_ticks([]) #删除原本的坐标线
ax.yaxis.set_ticks([])
x = np.linspace(0,20,21) #x 轴设计一下
plt.xticks(x)
y = np.linspace(0,100,11) #y 轴设计一下
plt.yticks(y)

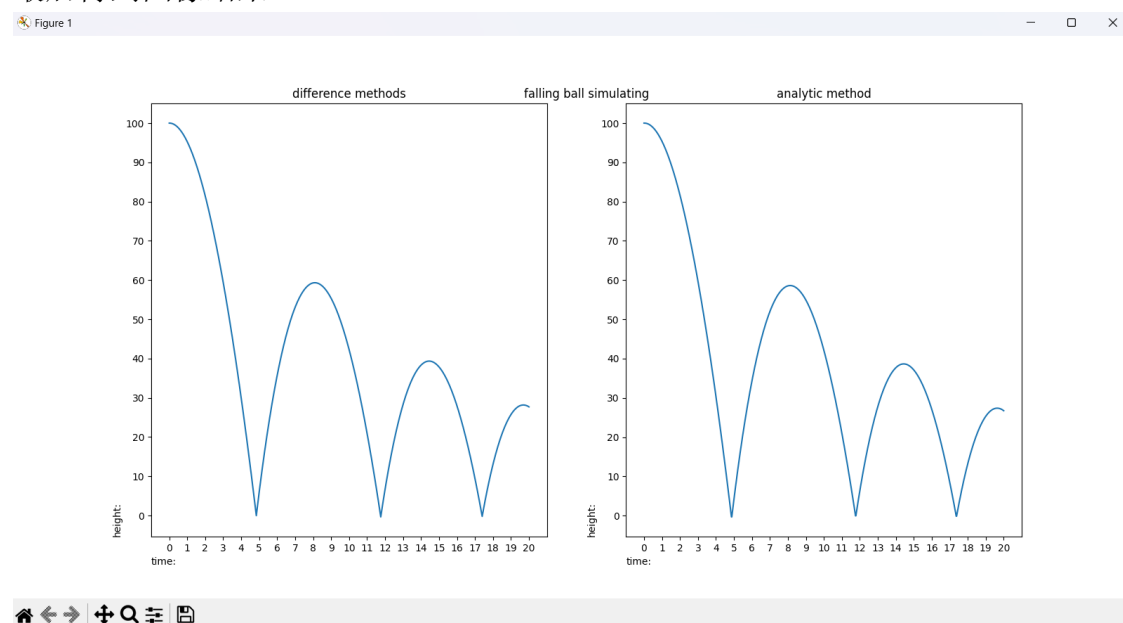
plt.figure(figsize=(16,8),dpi=80) #让整个画面舒服一点
plt.title("falling ball simulating") #设置一个大标题
ax = plt.gca()
ax.xaxis.set_ticks([]) #删除 figure 的坐标线
ax.yaxis.set_ticks([])
ax.spines["top"].set_color(None) #删除 figure 无用的线
ax.spines["bottom"].set_color(None)

plt.subplot(1,2,1) #开一下 subplot
init()
plt.title("difference methods") #差分法
plt.plot(t,h)

plt.subplot(1,2,2)
init()
plt.title("analytic method") #解析法
plt.plot(T,H)

plt.show() #show
复制
```

最后得到图像结果



四、实验分析:

在有 python 基础的情况下确实没有太大难度, 唯一要仔细考虑的只有解析法里方程常数的更新

五、总结与思考:

于本人而言, 本次主要学习了 matplotlib 的使用, 在能够画出图像的基础上进一步学习了如何设计更加好看的图像。