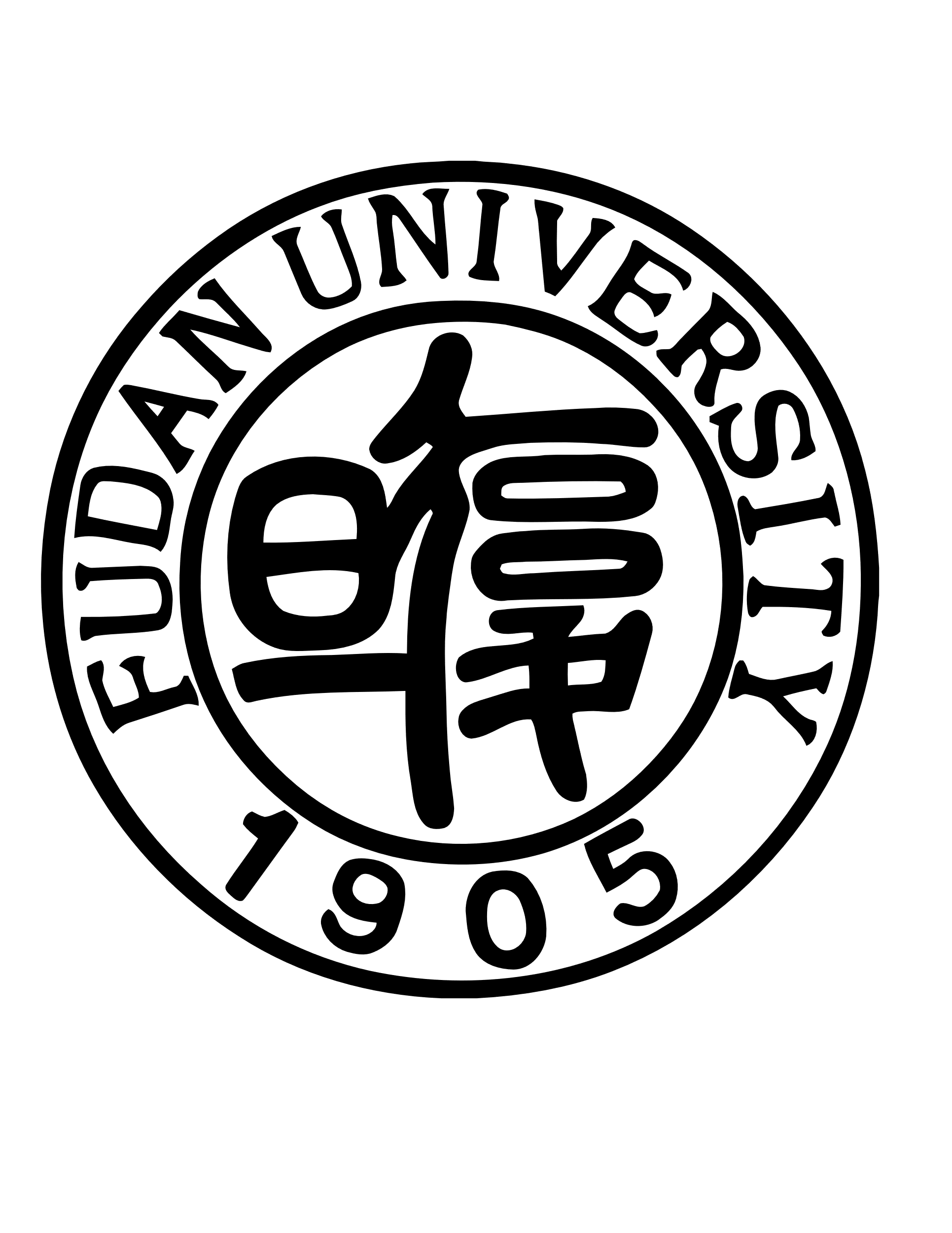
**电子系统导论实验报告**

**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

1. 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。

1. 实验原理：

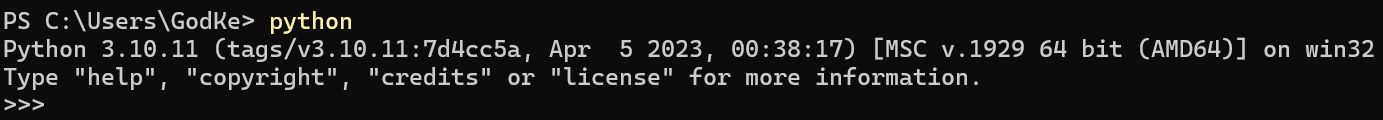
Python3相关教程内容；matplotlib等库的官方文档。

1. 实验内容：

# 1.python安装及IDE

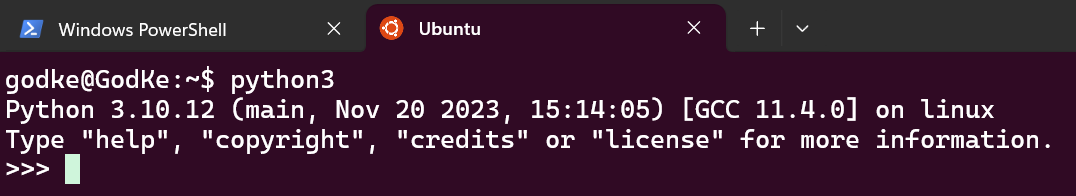
由于本人之前以及学习过python，且在ICS中使用python完成了课程项目的GUI设计，所以本人的python及IDE环境已经配置好了

## Windows下

  
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  
  
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

  
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  
01Classes/IES/img/Pasted image 20240308160043.png

# 3.matplotlib库 (个人学习笔记、可以跳过不看)

**[Ctrl+点击跳过](#课堂小实验)**

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  
Figue具有默认参数，可以不进行设置  


# plot(subplot)

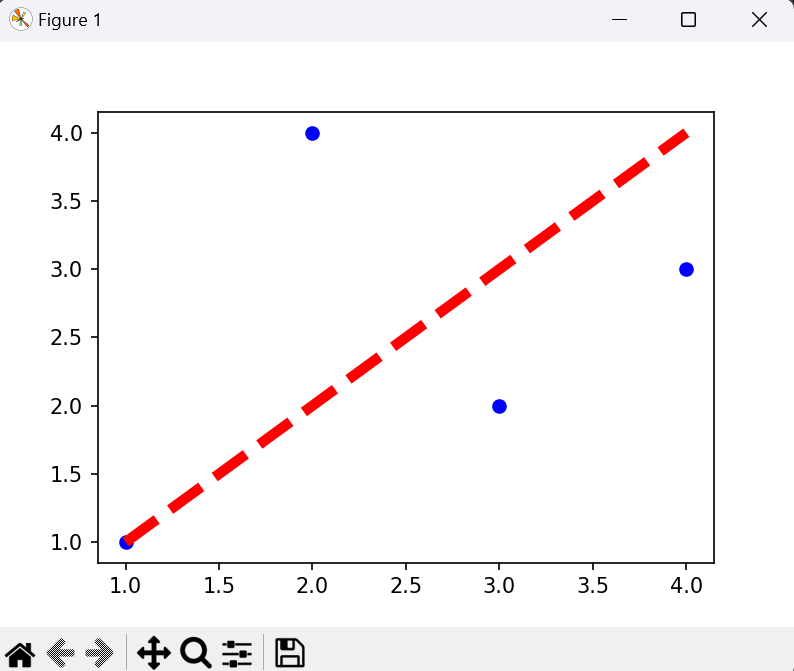
## plot

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和散点图

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

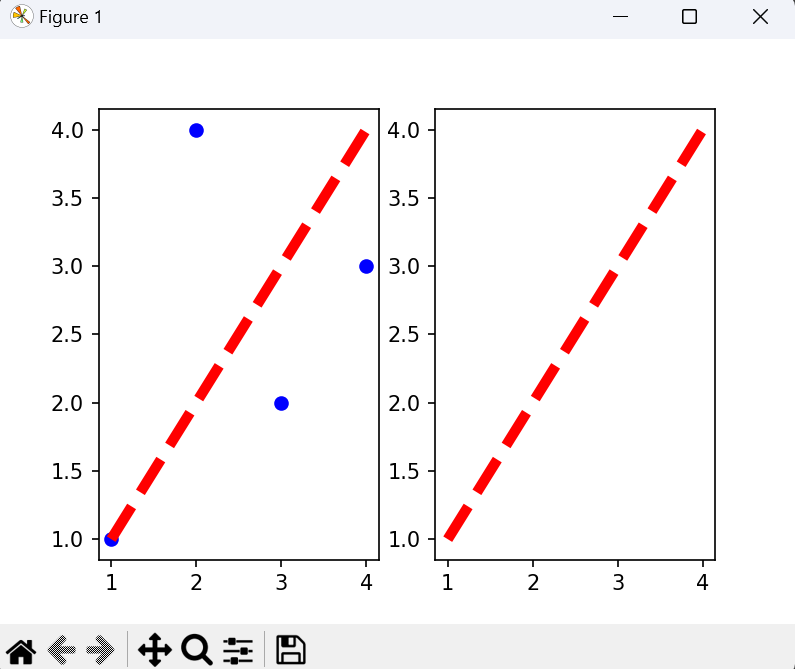
## subplot

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

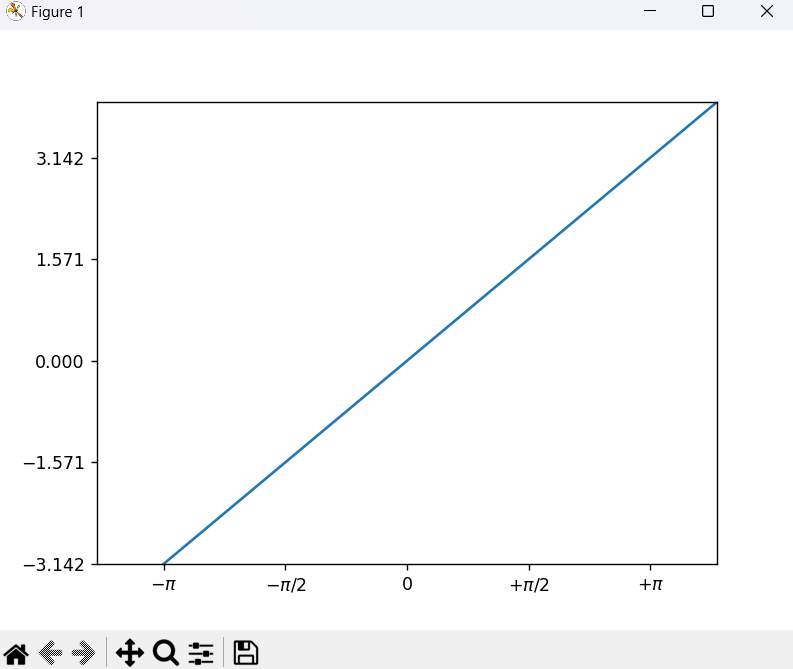
plt.xlim(left,right) # 设置横轴的上下限  
plt.xticks(Array) # 设置横轴记号,所有在列表中的点都会被标出  
plt.xlabel("string",loc="位置") #设置x轴名称和位置  
plt.ylim(left,right) # 设置纵轴的上下限  
plt.yticks(Array) # 设置纵轴记号,同上  
plt.ylabel("string",loc="位置") #设置y轴名称和位置  
复制

**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pi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)

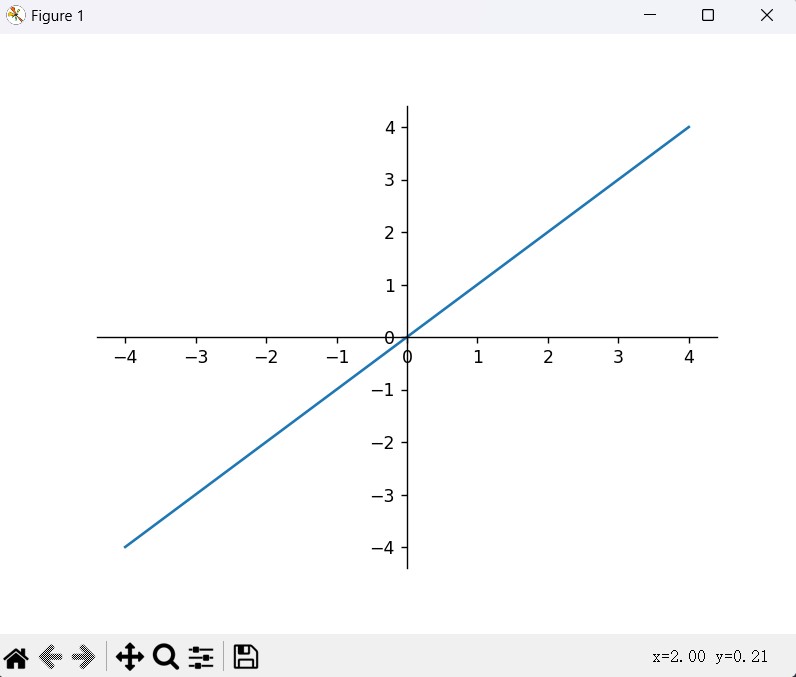
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],  
 [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])  
plt.yticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi])  
复制

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），如果要将脊柱放在图的中间，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

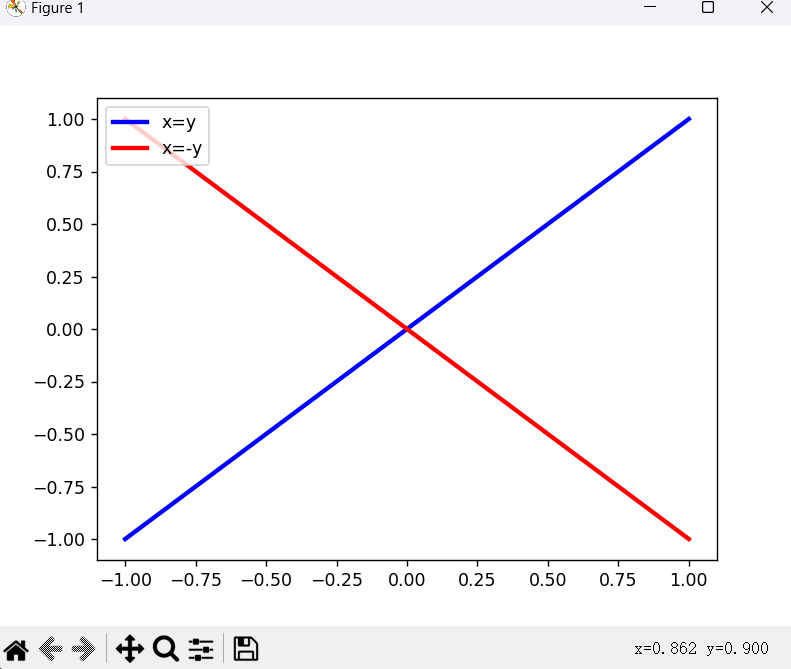
得到图像为  


### 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

matpltlib支持非常多种类的图样，可以自行探索

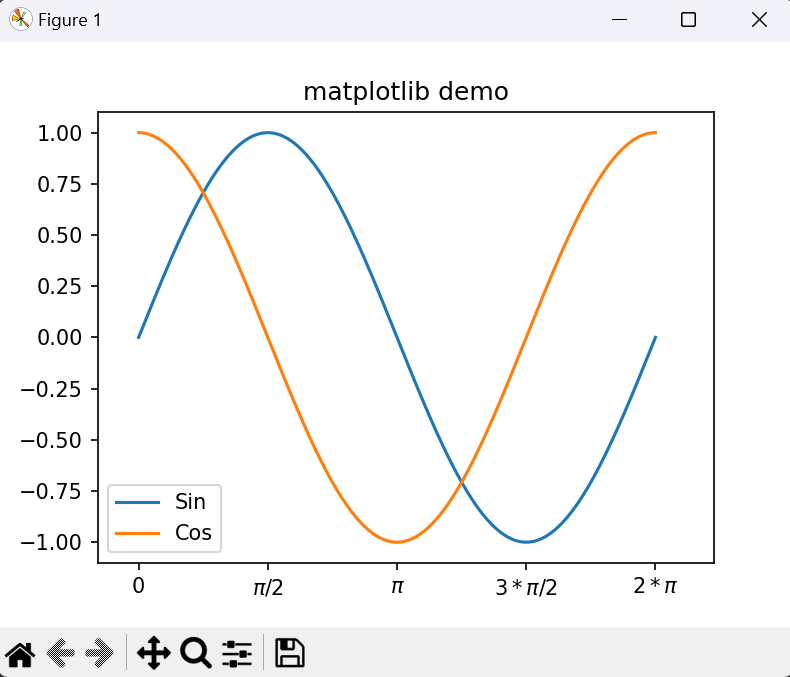
# 4.课堂小实验

***用numpy是用python的基本素养***

## 画Sin和Cos曲线

**简单写一段**

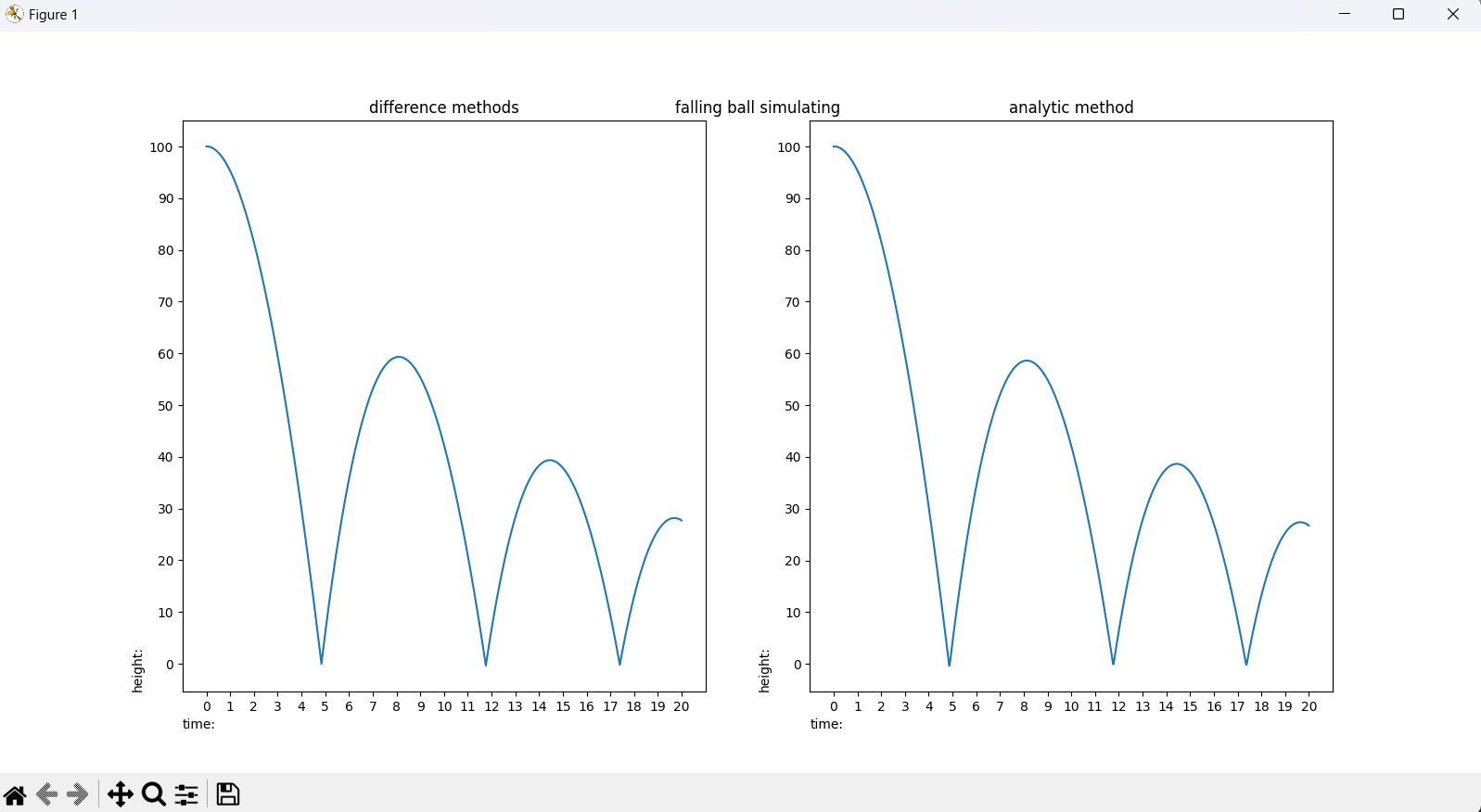
import numpy as np  
import matplotlib.pyplot as plt  
  
X = np.linspace(0,2\*np.pi,256) #linspace方法生成区间内等距点的坐标列表  
S = np.sin(X) # 直接调用numpy里的方法就行  
C = np.cos(X)  
  
plt.title("matplotlib demo") # 设置标题  
plt.xlim(-0.5,7) # 设计x轴  
plt.xticks([0,np.pi/2,np.pi,3\*np.pi/2,2\*np.pi],[0,r'$\pi/2$',r'$\pi$',r'$3\*\pi/2$',r'$2\*\pi$'])  
  
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，但是因为有常量更新，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  


1. 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
2. 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。