孙雨欣-数值方法-2020-11-25

解非线性方程

方法综述

- 使用SymPy符号计算库中的solve()（解析解）/nsolve()（数值解）函数。
- 一元方程数值解可使用迭代法辅助求解。

问题分类

求解一元方程

问题引入：求该方程数值解

$$f(x)=\ln\left(\frac{513+0.6651x}{513-0.6651x}\right)-\frac{x}{1400\times 0.0918}=0.$$

解法一：SymPy.solve/nsolve函数求解

实际上SymPy内置解方程方法为迭代法。
输入：

```
import numpy as np
from sympy import *
var("x")
#方法一：求解解析解，进一步得出数值解
#solve()参数：表达式，自变量
result1=solve(x**3-x-1,x)
#求出解析解后，可根据解析解找出所需要的解，evalf()输出数值解
result1[2].evalf()
#方法二：对于没有解析解的方程，直接输出数值解
result2=nsolve(ln((513+0.6651*x)/(513-0.6651*x))-x/(1400*0.0918),0)
result2
```

输出：

```
1.32471795724475
0
```

解法二：迭代法

本方法仅提供一种思路，实际求解时可直接用解法一。实际上SymPy内置解方程方法为迭代法。
示例：双点弦截法
输入：

```
#解方程法二：迭代法
#方法一：弦截法
```

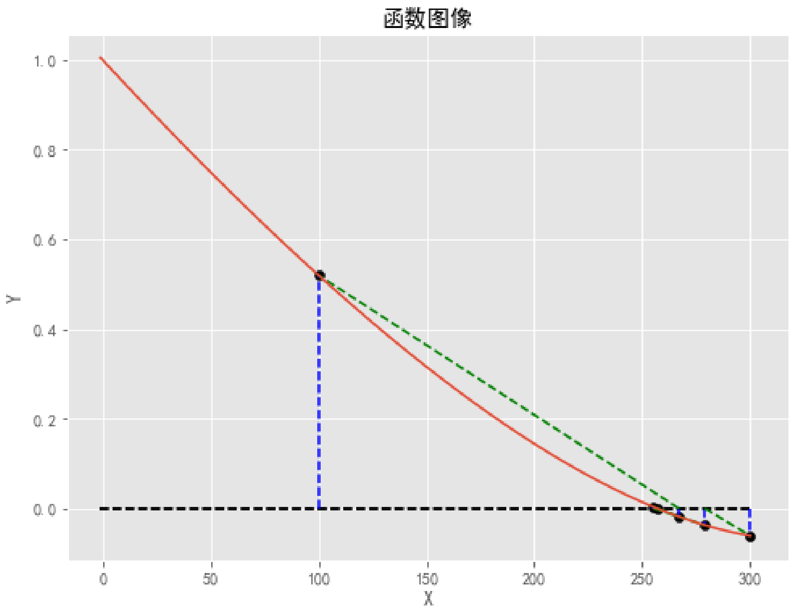
```python
import numpy as np
import matplotlib.pyplot as plt
#可以显示中文
plt.rcParams["font.sans-serif"] = ["SimHei"]
plt.rcParams['axes.unicode_minus'] = False
# 设置风格
plt.style.use('ggplot')
# 定义函数
init_fun = lambda x: (513+0.6651*x)/(513-0.6651*x)-x/(1400*0.0918)
# 导数
#deri_fun = lambda x: 2*x-4
# input
'''
x0:初始值1
x1:初始值2
theta:误差上界
'''
x0=float(input('输入初始点x0：较大值\n'))
x1=float(input('输入初始点x1：较小值\n'))
theta=1e-7
fig_1 = plt.figure(figsize = (8, 6))
plt.hlines(0,-1,x0,'black','--')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('函数图像')
# 函数图像
x=[]
if x0>0:
    x = np.arange(-1,x0,0.05)
    plt.hlines(0,-1,x0,'black','--')
else:
    x = np.arange(x0,10,0.05)
    plt.hlines(0,x0,10,'black','--')
y = init_fun(x)
def Secant(func = init_fun, x0 =x0, x1 = x1, theta = theta):
    number=0
    while True:
        x2=x0-func(x0)*(x1-x0)/(func(x1)-func(x0))
        plt.vlines(x0,0,init_fun(x0),'blue','--')
        plt.plot([x2,x0],[0,func(x0)],'r--',c='green')
        plt.scatter(x0,func(x0),c='black')
        if abs(func(x2))<theta:
            return x2,number
        x0=x1
        x1=x2
        number += 1

# 迭代法计算求解x0
xi,number = Secant(init_fun, x0, x1, theta)
```

```
print('迭代结果：'+str(xi))
print('迭代次数：'+str(number))
## 函数求解
plt.plot(x,y)
plt.show()
```

输出：

```
输入初始点x0：较大值
300
输入初始点x1：较小值
100
迭代结果：256.84798935747875
迭代次数：5
```



如果希望在lambda表达式输入对数、指数操作，只需引用np：

```
# 定义函数
init_fun = lambda x: np.log((513+0.6651*x)/(513-0.6651*x))-x/(1400*0.0918)
```

求解多元方程组

与求解一元方程同理。

方法一：运用SymPy

输入：

```
import numpy as np
from sympy import *
var("x,y")
#方法一：求解解析解，进一步得出数值解
#solve()参数：表达式，变量
result1=solve((x**2+x*y+1,y**2+x*y+2),x,y)
result1
#求出解析解后，可根据解析解找出所需要的解，evalf()输出数值解
result1[0][0].evalf()
#方法二：对于没有解析解的方程，直接输出数值解
```

```
var("x,y")
f1=x**2+y*x-1
f2=y**2+x+2
#初值要选好，否则会迭代不收敛
nsolve((f1,f2),(x,y),(I,I))
```

输出：

```
[(-sqrt(3)*I/3, -2*sqrt(3)*I/3), (sqrt(3)*I/3, 2*sqrt(3)*I/3)]
-0.577350269189626*I
Matrix([
[0.518912794385156 - 0.666609844932019*I],
[ 0.208223290106041 + 1.60070913439255*I]])
```

方法二：运用SciPy.optimize.fsolve()

输入：

```
from scipy.optimize import fsolve
from math import sin
def f(x):
    x0,x1,x2=x.tolist()
    return[
        5*x1+3,
        4*x0*x0-2*sin(x1*x2),
        x1*x2-1.5
    ]
#f计算方程组误差，[1,1,1]是初值
result=fsolve(f,[1,1,1])
print(result)
print(f(result))
```

输出：

```
[-0.70622057 -0.6        -2.5       ]
[0.0, -9.126033262418787e-14, 5.329070518200751e-15]
```

解线性方程组

方法：使用scipy.linalg的solve()方法
输入：

```
import numpy as np
from scipy.linalg import solve
a = np.array([[3, 1, -2], [1, -1, 4], [2, 0, 3]])
b = np.array([5, -2, 2.5])
x = solve(a, b)
x
```

输出：

```
array([0.5, 4.5, 0.5])
```

插值法

方法综述

插值是一种通过已知的离散数据来求未知数据的方法。与拟合不同的是，它要求曲线通过所有的已知数据。SciPy的interpolate模块提供了许多进行插值运算的函数。

问题分类

一元函数插值

其他插值方法的Python实现可参考：http://liao.cpython.org/scipy11/
文档仅演示最常用的B样条插值。

B样条插值

方法一：interp1d()
一维数据的B样条插值运算可以通过interp1d()完成。

```
interp1d(x,y,kind='cubic')
```

参数x和y：一系列已知的数据点。
kind：插值类型。

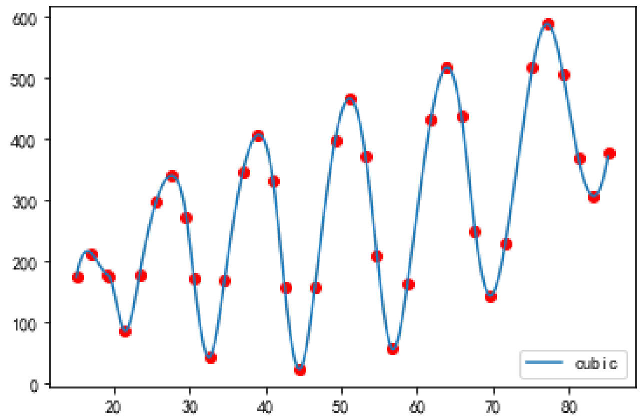
- ’zero’,’nearest’：阶梯插值，相当于0阶B样条曲线。
- ’slinear’,’linear’：线性插值，相当于1阶B样条曲线。
- ’quadratic’,’cubic’：2阶和3阶B样条曲线，更高阶的曲线可直接使用整数值指定。

实际常用三次B样条插值。

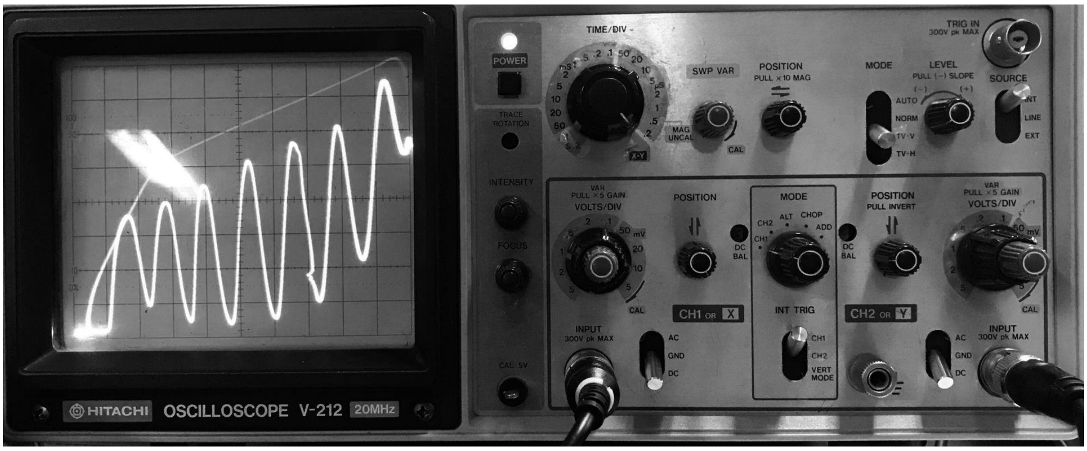
输入：（数据来源于本人物理实验BII弗兰克–赫兹实验数据）

```
#绘图工具：pylab
import numpy as np
from scipy import interpolate
import pylab as pl
x=[15.1,17.1,19.1, 19.4,21.4,23.4, 25.6,27.6,29.6, 30.6,32.6,34.6, 37.0,39.0,41.0, 42.5,44.5,46.5, 49.2,51.2,53.2, 54.7,56.7,58.7, 61.8,63.8,65.8, 67.6,69.6,71.6, 75.2,77.2,79.2, 81.2,83.2,85.2]
y=[175,213,179, 176,87,178, 298,340,272, 171,43,168, 345,407,332, 159,24,158, 399,466,371, 210,57,164, 433,517,437, 250,144,229, 518,588,505, 369,307,379]
xnew=np.arange(15.1,85.2,0.1)
pl.plot(x,y,"ro")
for kind in ["cubic"]:#插值方式
    #"nearest","zero"为阶梯插值
    #slinear 线性插值
    #"quadratic","cubic" 为2阶、3阶B样条曲线插值
    f=interpolate.interp1d(x,y,kind=kind)
    # ‘slinear’, ‘quadratic’ and ‘cubic’ refer to a spline interpolation of first, second or third order)
    ynew=f(xnew)
    pl.plot(xnew,ynew,label=str(kind))
pl.legend(loc="lower right")
pl.show()
```

输出：



与示波器输出的图像十分类似。



方法二：splrep()+splev()

该函数可以找到一维曲线的B-spline表示。

输入：

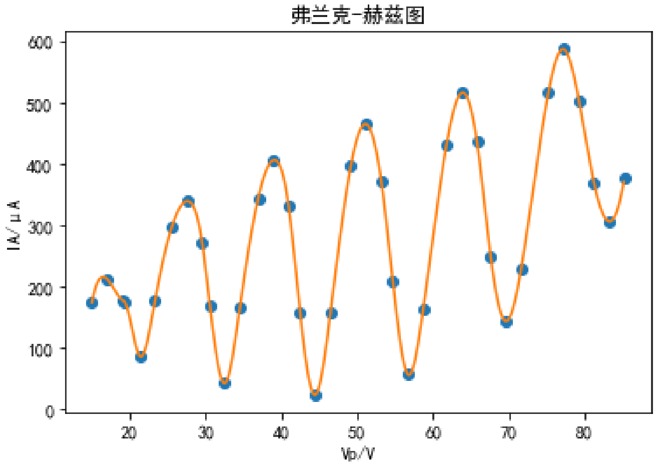
```
import numpy as np
import matplotlib.pyplot as plt
#进行样条插值
import scipy.interpolate as spi

plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
plt.rcParams['axes.unicode_minus']=False #用来正常显示负号
#数据准备
X=[15.1,17.1,19.1, 19.4,21.4,23.4, 25.6,27.6,29.6, 30.6,32.6,34.6,
37.0,39.0,41.0, 42.5,44.5,46.5, 49.2,51.2,53.2, 54.7,56.7,58.7,
61.8,63.8,65.8, 67.6,69.6,71.6, 75.2,77.2,79.2, 81.2,83.2,85.2]
Y=[175,213,179, 176,87,178, 298,340,272, 171,43,168, 345,407,332,
159,24,158, 399,466,371, 210,57,164, 433,517,437, 250,144,229,
518,588,505, 369,307,379]
#定义插值点
ix3=np.arange(15.1,85.2,0.1)

#进行三次样条拟合
ipo3=spi.splrep(X,Y,k=3) #样本点导入，生成参数
iy3=spi.splev(ix3,ipo3) #根据观测点和样条参数，生成插值
#作图
```

```
plt.plot(X,Y,'o',ix3,iy3)
plt.xlabel('Vp/V')
plt.ylabel('IA/μA')
plt.title('弗兰克-赫兹图')
plt.show()
```

输出：



二元函数插值

绘制2D图

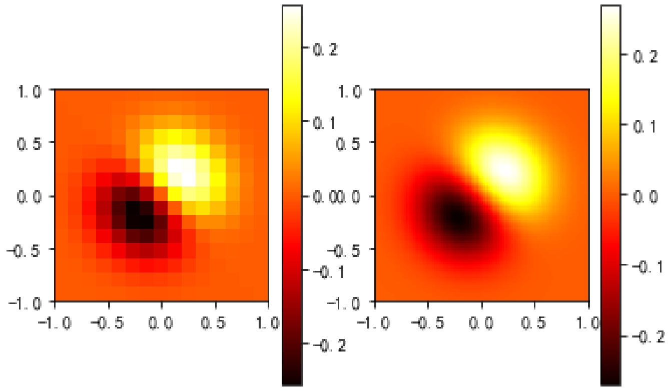
输入：

```
# -*- coding: utf-8 -*-
import numpy as np
from scipy import interpolate
import pylab as pl
import matplotlib as mpl
def func(x, y):
    return (x+y)*np.exp(-5.0*(x**2 + y**2))
# X-Y轴分为15*15的网格
y,x= np.mgrid[-1:1:15j, -1:1:15j]
fvals = func(x,y) # 计算每个网格点上的函数值  15*15的值
print(len(fvals[0]))
#三次样条二维插值
newfunc = interpolate.interp2d(x, y, fvals, kind='cubic')
# 计算100*100的网格上的插值
xnew = np.linspace(-1,1,100)#x
ynew = np.linspace(-1,1,100)#y
fnew = newfunc(xnew, ynew)#仅仅是y值   100*100的值
# 输出解得所求点的插值
print(newfunc(0.01,0.01))
# 绘图
# 为了更明显地比较插值前后的区别，使用关键字参数interpolation='nearest'
# 关闭imshow()内置的插值运算。
pl.subplot(121)
im1=pl.imshow(fvals, extent=[-1,1,-1,1], cmap=mpl.cm.hot, interpolation='nearest', origin="lower")#pl.cm.jet
#extent=[-1,1,-1,1]为x,y范围  favals为
```

```
pl.colorbar(im1)
pl.subplot(122)
im2=pl.imshow(fnew, extent=[-1,1,-1,1], cmap=mpl.cm.hot, interpolation='nearest', origin="lower")
pl.colorbar(im2)
pl.show()
```

输出：

```
15
[0.01985751]
```



绘制3D图

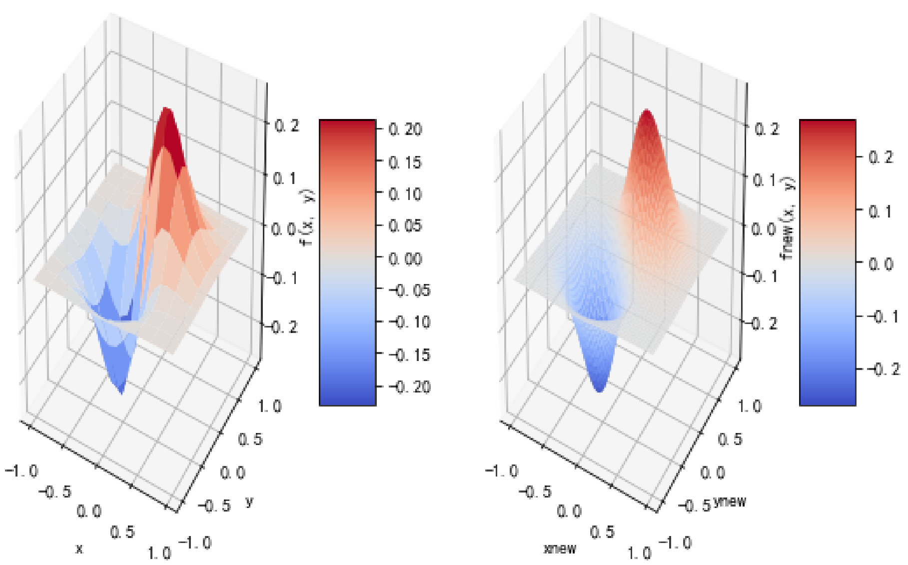
输入：

```
# -*- coding: utf-8 -*-
"""
演示二维插值。
"""
# -*- coding: utf-8 -*-
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib as mpl
from scipy import interpolate
import matplotlib.cm as cm
import matplotlib.pyplot as plt
def func(x, y):
    return (x+y)*np.exp(-5.0*(x**2 + y**2))
# X-Y轴分为20*20的网格
x = np.linspace(-1,1,20)
y = np.linspace(-1,1,20)
x, y = np.meshgrid(x, y)#20*20的网格数据
fvals = func(x,y) # 计算每个网格点上的函数值  15*15的值
fig = plt.figure(figsize=(9, 6))
#Draw sub-graph1
ax=plt.subplot(1, 2, 1,projection = '3d')
surf = ax.plot_surface(x, y, fvals, rstride=2, cstride=2, cmap=cm.coolwarm,linewidth=0.5, antialiased=True)
ax.set_xlabel('x')
ax.set_ylabel('y')
```

```
ax.set_zlabel('f(x, y)')
plt.colorbar(surf, shrink=0.5, aspect=5)#标注
#二维插值
newfunc = interpolate.interp2d(x, y, fvals, kind='cubic')#newfunc为一个函数
# 计算100*100的网格上的插值
xnew = np.linspace(-1,1,100)#x
ynew = np.linspace(-1,1,100)#y
fnew = newfunc(xnew, ynew)#仅仅是y值   100*100的值  np.shape(fnew) is 100*100
xnew, ynew = np.meshgrid(xnew, ynew)
# 输出解得所求点的插值
print(newfunc(0.01,0.01))
# 绘制3D图
ax2=plt.subplot(1, 2, 2,projection = '3d')
surf2 = ax2.plot_surface(xnew, ynew, fnew, rstride=2, cstride=2, cmap=cm.coolwarm,linewidth=0.5, antialiased=True)
ax2.set_xlabel('xnew')
ax2.set_ylabel('ynew')
ax2.set_zlabel('fnew(x, y)')
plt.colorbar(surf2, shrink=0.5, aspect=5)#标注
plt.show()
```

输出：

```
[0.01983083]
```



函数逼近（拟合）

最常用方法：最小二乘拟合

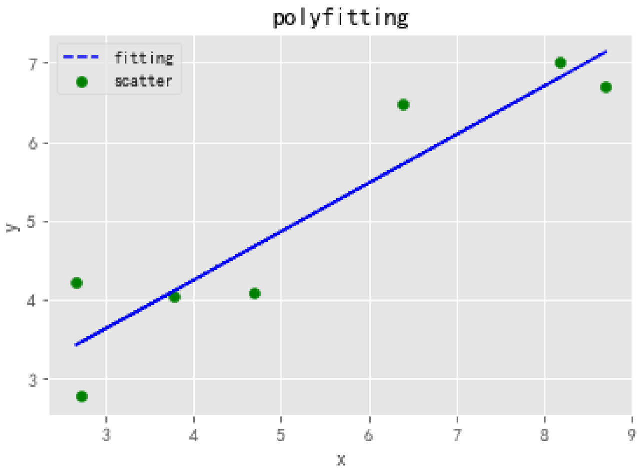
输入：

```
import numpy as np
from scipy.optimize import leastsq
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 residuals(p):
    "计算以p为参数的直线和原始数据误差"
    k,b=p#若不是线性拟合，则修改对应参数
    return y-(k*x+b)
#leastsq使residuals()输出数组的平方和最小，初值[1,0]
r=leastsq(residuals,[1,0])#若不是线性拟合，则修改对应参数
k,b=r[0]#若不是线性拟合，则修改对应参数
x1=np.arange(2.66,8.71,0.01)
y1=k*x+b#若不是线性拟合，则修改对应参数
print("k=",k,"b=",b)#若不是线性拟合，则修改对应参数
#画图
import matplotlib.pyplot as plt
plt.scatter(x,y,c='g',label='scatter')#散点图
plt.plot(x1,y1,'b--',label='fitting')
plt.title('polyfitting')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()#显示标签
plt.show()
```

输出：

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

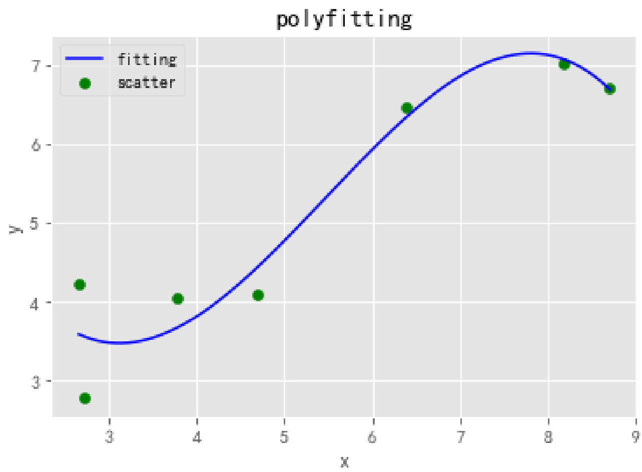


计算误差曲面函数：

```
def S(k,b):
  #计算直线y=k*x+b与原始数据x,y误差的平方和
  error=np.zeros(k,shape)
  for  x,y in zip(x,y)
    error+=(y-(k*x+b))**2
   return error
```

若使用其他函数进行拟合，只需将k*x+b替换为对应和函数（参数）即可。

如，三次多项式拟合图像如下：



微分方程数值解法

常微分方程

方法一：SymPy.dsolve()

$$f''(x) - 2f'(x) + f(x) = sin(x)$$

输入：

```
import numpy as np
from sympy import *
f = Function('f')
x = symbols('x')
eq = Eq(f(x).diff(x, x) - 2*f(x).diff(x) + f(x), sin(x))
print(dsolve(eq, f(x)))
```

输出：

```
Eq(f(x), (C1 + C2*x)*exp(x) + cos(x)/2)
```

方法二：scipy.integrate.odeint()

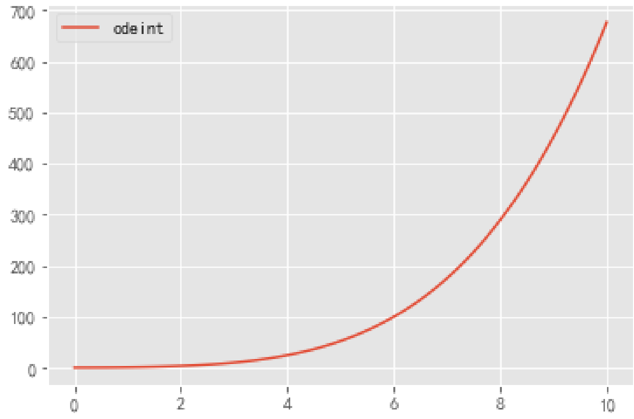
这个函数，要求微分方程必须化为标准形式，即

$$\frac{dy}{dt} = f(y, t,)$$

输入：

```
import math
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
def func(y, t):
    return t * math.sqrt(y)
YS=odeint(func,y0=1,t=np.arange(0,10.1,0.1))
t=np.arange(0,10.1,0.1)
plt.plot(t, YS, label='odeint')
plt.legend()
plt.show()
```

输出：

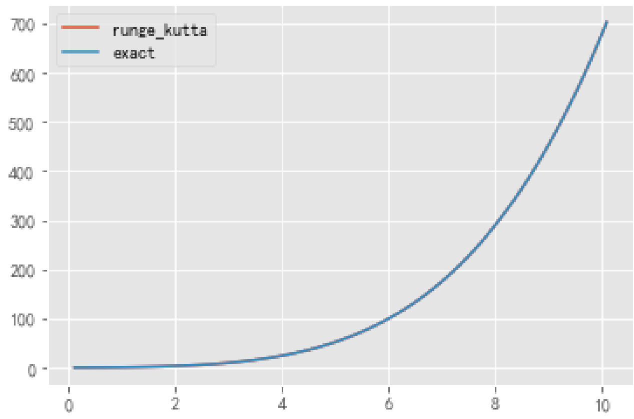


方法三：单个函数四阶龙格–库塔法

输入：

```
import math
import numpy as np
import matplotlib.pyplot as plt
def runge_kutta(y, x, dx, f):
    """ y is the initial value for y
        x is the initial value for x
        dx is the time step in x
        f is derivative of function y(t)
    """
    k1 = dx * f(y, x)
    k2 = dx * f(y + 0.5 * k1, x + 0.5 * dx)
    k3 = dx * f(y + 0.5 * k2, x + 0.5 * dx)
    k4 = dx * f(y + k3, x + dx)
    return y + (k1 + 2 * k2 + 2 * k3 + k4) / 6.
if __name__=='__main__':
    t = 0.
    y = 1.
    dt = .1
    ys, ts = [], []
def func(y, t):
    return t * math.sqrt(y)
while t <= 10:
    y = runge_kutta(y, t, dt, func)
    t += dt
    ys.append(y)
    ts.append(t)
exact = [(t ** 2 + 4) ** 2 / 16. for t in ts]
plt.plot(ts, ys, label='runge_kutta')
plt.plot(ts, exact, label='exact')
plt.legend()
#plt.show()
```

输出：



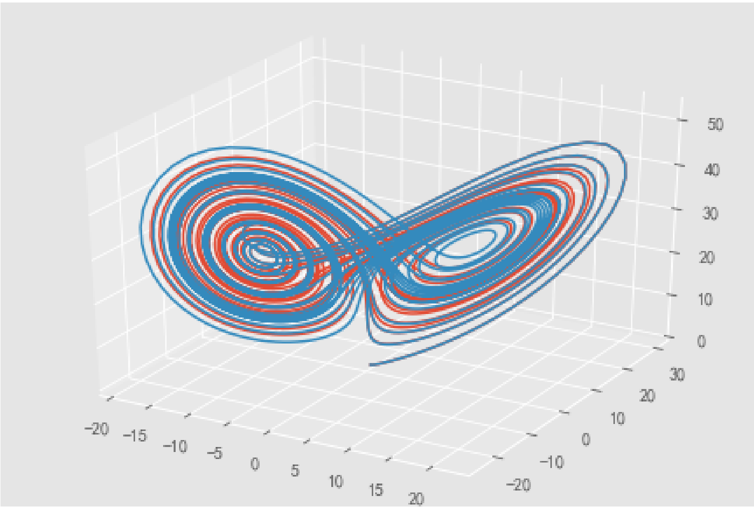
方法四：多个微分方程：欧拉法

输入：

```
import numpy as np
"""
移动方程：
t时刻的位置P(x,y,z)
steps：dt的大小
sets：相关参数
"""
def move(P, steps, sets):
    x, y, z = P
    sgima, rho, beta = sets
    # 各方向的速度近似
    dx = sgima * (y - x)
    dy = x * (rho - z) - y
    dz = x * y - beta * z
    return [x + dx * steps, y + dy * steps, z + dz * steps]
# 设置sets参数
sets = [10., 28., 3.]
t = np.arange(0, 30, 0.01)
# 位置1：
P0 = [0., 1., 0.]
P = P0
d = []
for v in t:
    P = move(P, 0.01, sets)
    d.append(P)
dnp = np.array(d)
# 位置2：
P02 = [0., 1.01, 0.]
P = P02
d = []
for v in t:
    P = move(P, 0.01, sets)
    d.append(P)
dnp2 = np.array(d)
```

```
"""
画图
"""
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
fig = plt.figure()
ax = Axes3D(fig)
ax.plot(dnp[:, 0], dnp[:, 1], dnp[:, 2])
ax.plot(dnp2[:, 0], dnp2[:, 1], dnp2[:, 2])
plt.show()
```

输出:



偏微分方程

与解常微分方程原理相同。

练习题目

（由于题目涉及知识方法较多，会在后续慢慢补全，本例仅演示数据预处理部分）

题目内容

```
淡水养殖池塘水华发生及池水净化处理
目前在我国水产养殖中，池塘养殖产量约占淡水养殖的70%。近年来，随着淡水生态系统水体污染和富营养化进程的加剧，经常导致有害蓝藻、轮虫等常见的浮游生物高密度发生，很容易诱发大面积水华。水华造成严重的环境污染及水体污染，对养殖业是一个严重的打击。
水华的发生不仅直接影响了养殖对象的正常生长发育，严重时大量排泄废水造成淡水资源污染，还会破坏养殖生态系统的平衡，导致养殖对象的不同程度死亡，造成巨大经济损失。为此我们通过研究淡水养殖池塘相关主要理化因子，主要浮游生物数据及鱼虾生成等数据分析水华发生的原因，控制并预测水华的发生，从而提高养殖产量，减小环境污染等。通过对水华发生的了解，加强大家环保意识。
根据附件1-8完成如下问题:
1）通过附件1中数据分析水体、底泥与间隙水中常见主要理化因子之间的关系，并分析原因。
2）通过附件2中数据对四个池塘水体质量进行评价及分类，分析虾池与鱼池对水体的影响。
3）建立主要理化因子和常见浮游生物致害密度发生关系的模型，给出水华发生时主要理化因子的范围，预测淡水养殖池塘水华发生 (1号池发生轻微水华)。
```

4）结合附件及以上分析，建立鱼类生长与体重相关模型。在养殖鲢鱼、鳙鱼等的生长过程中可以摄食浮游生物，净化某些藻类，构造一个与1号池相同大小的净化池，通过水循环，并放养鲢鱼或鳙鱼，放养多少才能净化1号池中的藻类，净化效果如何。
5）结合附件及通过查阅资料构建一种生态养殖模式，有利于池水养殖池塘水体的自净化。通过以上养殖从而使淡水养殖减少向江河湖海养殖废水排放。

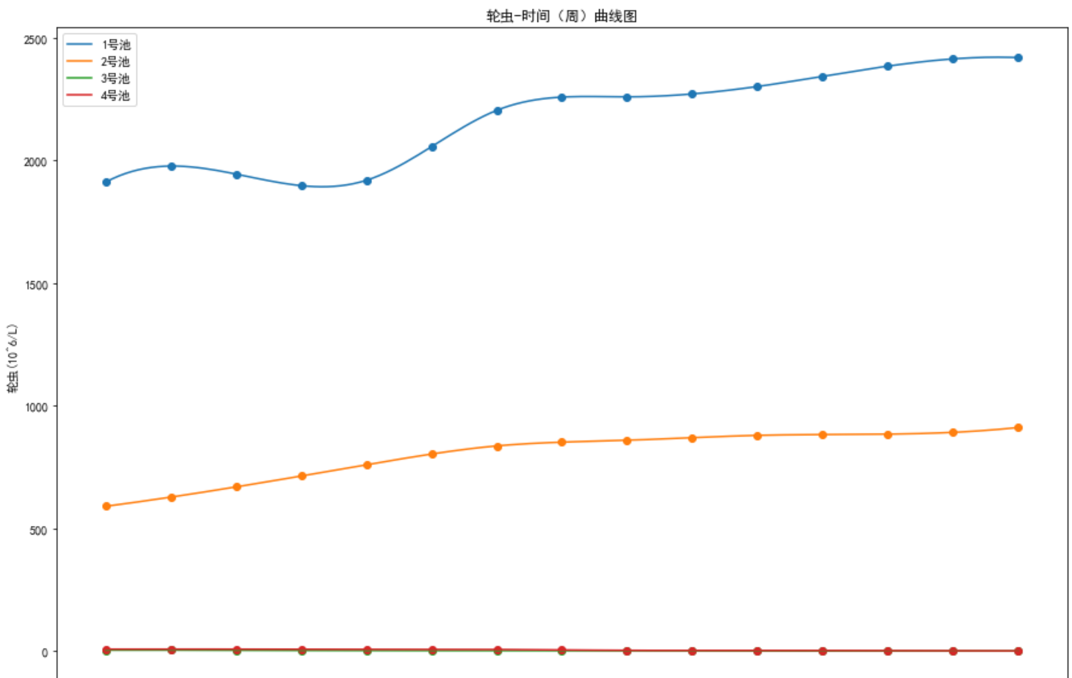
数据及资料见：
附件1 水体中常见理化因子
附件2 其它数据
附件3 吸光度及稀释倍数
附件4 浮游生物量
附件5 各池塘数据
附件6 鱼体重体长数据
附件7 鲢鱼鳙鱼相关数据
附件8 地表水环境质量标准

题目数据

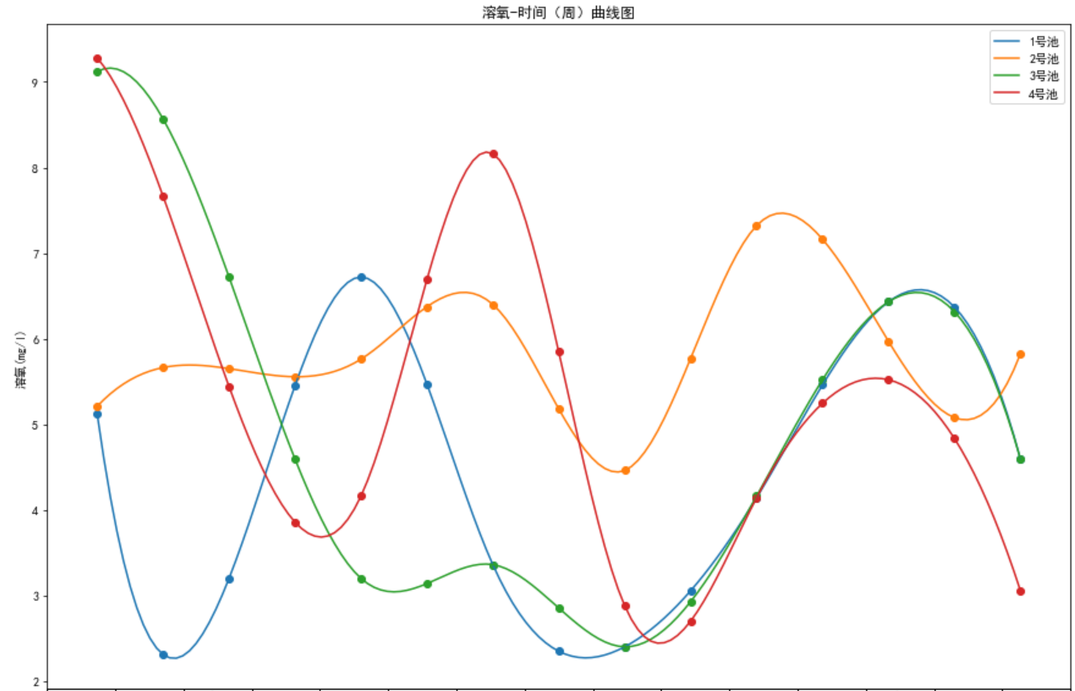
2016MathorCupA题.zip
0.8MB

数据预处理（只需翻译此部分）

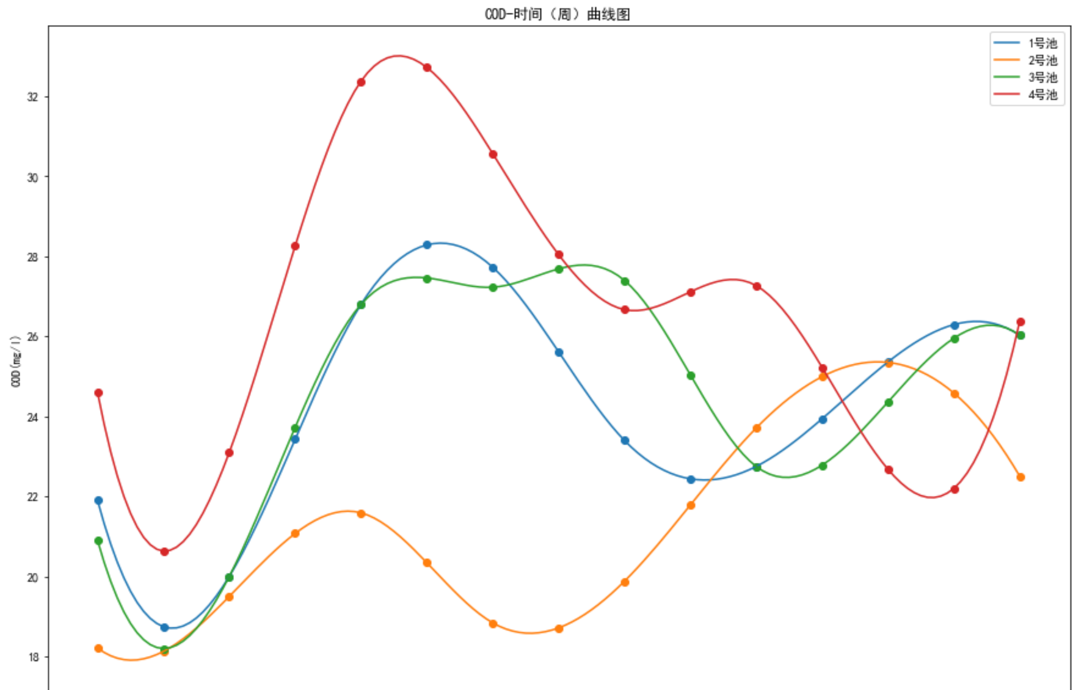
本文取每个水池中，A、B两个采样点各理化因子的实测值的均值作为各理化因子的计算值。总磷、总淡、氨氮15周的数据可以参考附件一。而附件二中COD、溶氧、PH值间隔两周采集一次，与附件一数据不对称，不足以建立合理的模型，因此考虑利用现有数据插值以补充数据。
插值方法选用三次B样条插值，该方法可以很好的保持数据光滑性和连续性，减少信息量的损失。最终得到的数据如下。



	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0	11.0	12.0	13.0	14.0	15.0
1号池	1913.0	1977.8	1945.0	1897.95	1920.0	2057.02	2205.0	2258.96	2260.0	2271.77	2302.0	2343.45	2385.0	2414.55	2420.0
2号池	591.0	628.41	670.0	714.34	760.0	803.99	837.0	852.34	860.0	870.53	880.0	883.28	885.0	892.22	912.0
3号池	4.46	4.51	3.94	3.16	2.61	2.55	2.66	2.54	2.12	1.48	1.02	1.06	1.35	1.5	1.12
4号池	8.49	8.64	8.45	8.13	7.92	7.84	7.13	5.27	3.49	3.01	3.12	2.9	2.38	1.84	1.56

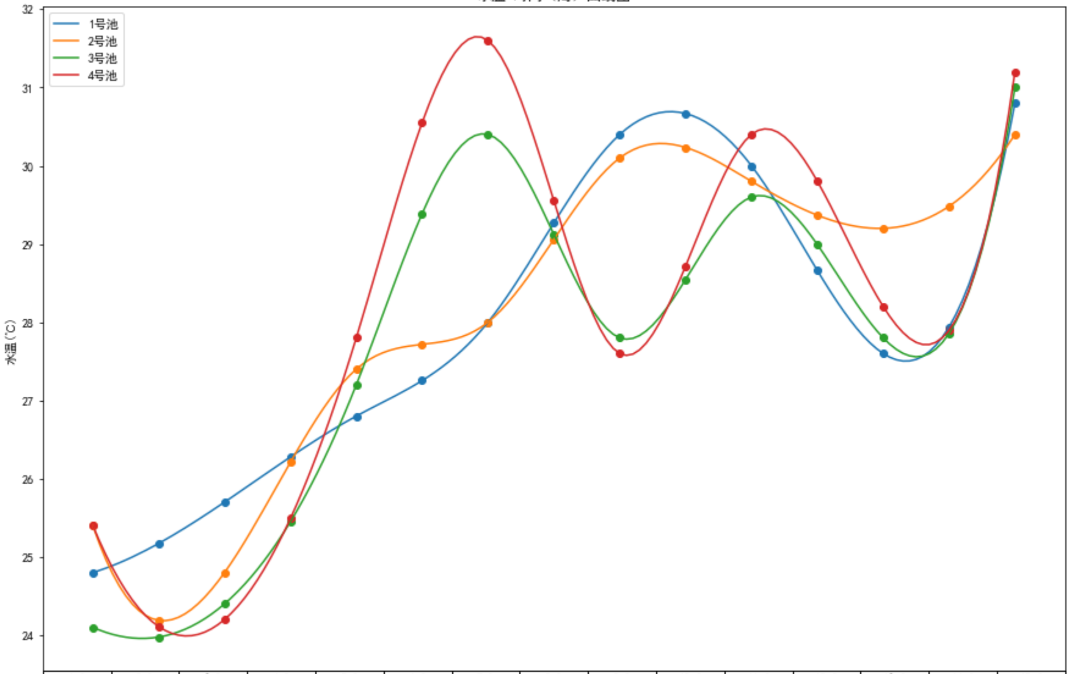


	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0	11.0	12.0	13.0	14.0	15.0
1号池	5.12	2.31	3.2	5.45	6.72	5.47	3.36	2.35	2.4	3.05	4.14	5.46	6.43	6.37	4.6
2号池	5.21	5.66	5.65	5.56	5.76	6.37	6.4	5.18	4.46	5.77	7.32	7.16	5.97	5.08	5.82
3号池	9.12	8.57	6.72	4.59	3.2	3.14	3.36	2.85	2.4	2.93	4.16	5.52	6.43	6.32	4.6
4号池	9.28	7.66	5.44	3.86	4.16	6.69	8.16	5.85	2.88	2.7	4.14	5.24	5.52	4.84	3.06



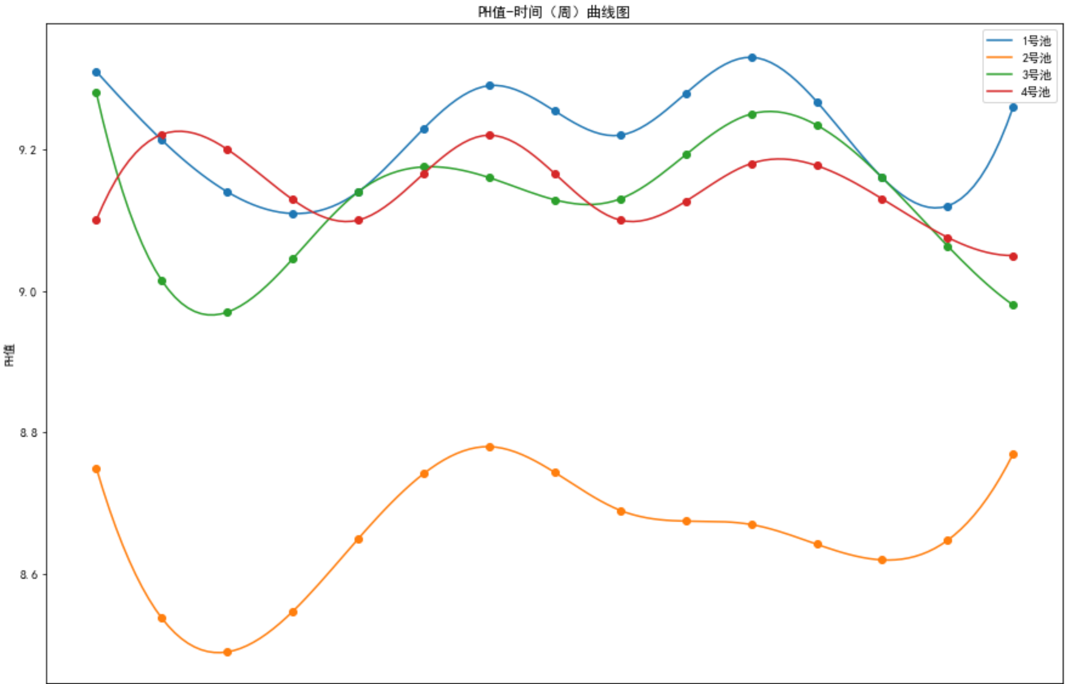
	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0	11.0	12.0	13.0	14.0	15.0
1号池	21.9	18.74	20.0	23.43	26.8	28.29	27.73	25.62	23.4	22.44	22.75	23.94	25.36	26.3	26.03
2号池	18.21	18.13	19.5	21.07	21.59	20.34	18.84	18.71	19.88	21.78	23.71	24.99	25.35	24.59	22.51
3号池	20.9	18.2	20.0	23.73	26.8	27.46	27.23	27.69	27.4	25.04	22.75	22.78	24.36	25.95	26.03
4号池	24.6	20.63	23.1	28.26	32.38	32.72	30.57	28.05	26.67	27.11	27.27	25.23	22.68	22.2	26.37

水温-时间（周）曲线图

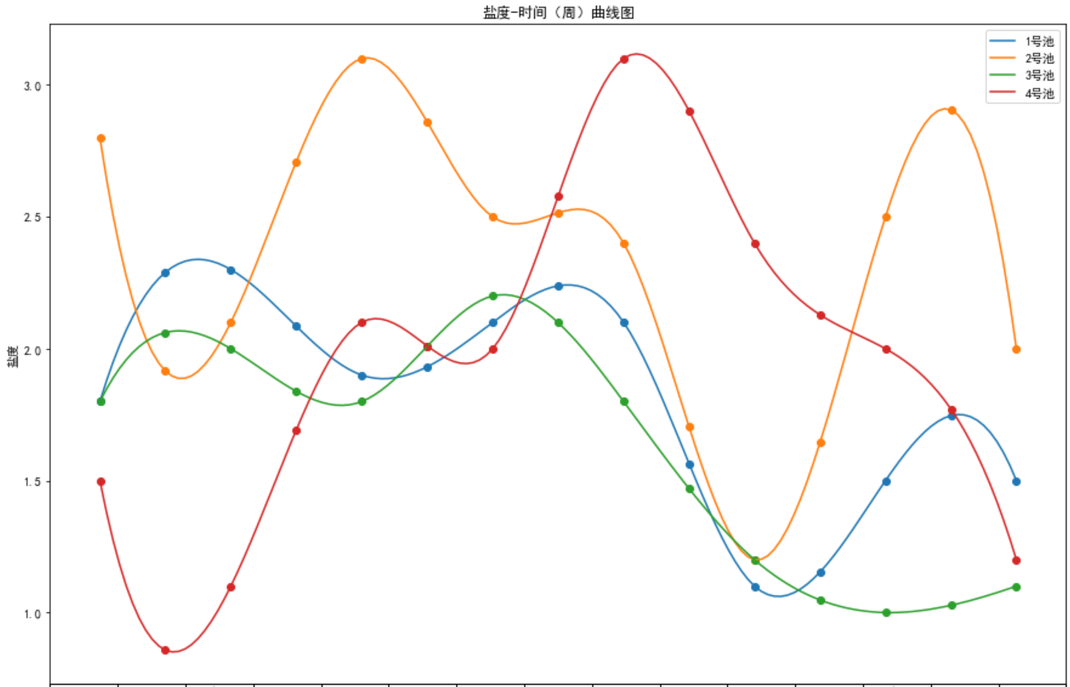


	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0	11.0	12.0	13.0	14.0	15.0
1号池	24.8	25.17	25.7	26.28	26.8	27.25	28.0	29.27	30.4	30.67	30.0	28.66	27.6	27.94	30.8
2号池	25.4	24.19	24.8	26.21	27.4	27.72	28.0	29.06	30.1	30.23	29.8	29.37	29.2	29.48	30.4
3号池	24.1	23.98	24.4	25.45	27.2	29.39	30.4	29.12	27.8	28.55	29.6	28.99	27.8	27.86	31.0
4号池	25.4	24.11	24.2	25.49	27.8	30.55	31.6	29.55	27.6	28.72	30.4	29.8	28.2	27.9	31.2

PH值-时间（周）曲线图



	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0	11.0	12.0	13.0	14.0	15.0
1号池	9.31	9.21	9.14	9.11	9.14	9.23	9.29	9.25	9.22	9.28	9.33	9.27	9.16	9.12	9.26
2号池	8.75	8.54	8.49	8.55	8.65	8.74	8.78	8.74	8.69	8.67	8.67	8.64	8.62	8.65	8.77
3号池	9.28	9.01	8.97	9.05	9.14	9.18	9.16	9.13	9.13	9.19	9.25	9.23	9.16	9.06	8.98
4号池	9.1	9.22	9.2	9.13	9.1	9.17	9.22	9.17	9.1	9.13	9.18	9.18	9.13	9.08	9.05



	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0	11.0	12.0	13.0	14.0	15.0
1号池	1.8	2.29	2.3	2.09	1.9	1.93	2.1	2.24	2.1	1.56	1.1	1.15	1.5	1.75	1.5
2号池	2.8	1.92	2.1	2.71	3.1	2.86	2.5	2.52	2.4	1.7	1.2	1.64	2.5	2.91	2.0
3号池	1.8	2.06	2.0	1.84	1.8	2.01	2.2	2.1	1.8	1.47	1.2	1.05	1.0	1.03	1.1
4号池	1.5	0.86	1.1	1.69	2.1	2.01	2.0	2.58	3.1	2.9	2.4	2.13	2.0	1.77	1.2

源代码：（以轮虫–时间为例）

```
#暂时手动导入数据，下周开始学习自动从EXCEL表读取数据
#轮虫-时间
import numpy as np
import matplotlib.pyplot as plt
#进行样条插值
import scipy.interpolate as spi

plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
plt.rcParams['axes.unicode_minus']=False #用来正常显示负号
#数据准备
X=[1,3,5,7,9,11,13,15]
Y1=[1913,1945,1920,2205,2260,2302,2385,2420]
Y2=[591,670,760,837,860,880,885,912]
Y3=[4.46,3.94,2.61,2.66,2.12,1.02,1.35,1.12]
Y4=[8.49,8.45,7.92,7.13,3.49,3.12,2.38,1.56]
#定义整周数点
xpoint=np.arange(1,15.1,1)
#定义曲线x点
xline=np.arange(1,15.1,0.1)

#进行三次样条拟合点
xpoint_r=spi.splrep(X,Y1,k=3) #样本点导入，生成参数
ypoint1=spi.splev(xpoint,xpoint_r) #根据观测点和样条参数，生成插值
xpoint_r=spi.splrep(X,Y2,k=3) #样本点导入，生成参数
ypoint2=spi.splev(xpoint,xpoint_r) #根据观测点和样条参数，生成插值
xpoint_r=spi.splrep(X,Y3,k=3) #样本点导入，生成参数
ypoint3=spi.splev(xpoint,xpoint_r) #根据观测点和样条参数，生成插值
xpoint_r=spi.splrep(X,Y4,k=3) #样本点导入，生成参数
ypoint4=spi.splev(xpoint,xpoint_r) #根据观测点和样条参数，生成插值
```

```
#进行三次样条拟合曲线
xline_r=spi.splrep(X,Y1,k=3) #样本点导入，生成参数
yline1=spi.splev(xline,xline_r) #根据观测点和样条参数，生成插值
xline_r=spi.splrep(X,Y2,k=3) #样本点导入，生成参数
yline2=spi.splev(xline,xline_r) #根据观测点和样条参数，生成插值
xline_r=spi.splrep(X,Y3,k=3) #样本点导入，生成参数
yline3=spi.splev(xline,xline_r) #根据观测点和样条参数，生成插值
xline_r=spi.splrep(X,Y4,k=3) #样本点导入，生成参数
yline4=spi.splev(xline,xline_r) #根据观测点和样条参数，生成插值
#输出数据表格
plt.figure(figsize=(15,10))
cell_text=[np.round(ypoint1,2),np.round(ypoint2,2),np.round(ypoint3,2),np.round(ypoint4,2)]
the_table=plt.table(cellText=cell_text,
                    rowLabels=["1号池","2号池","3号池","4号池"],
                    colLabels=xpoint,
                   )
#作图
plt.scatter(xpoint,ypoint1)
plt.plot(xline,yline1,label='1号池')
plt.scatter(xpoint,ypoint2)
plt.plot(xline,yline2,label='2号池')
plt.scatter(xpoint,ypoint3)
plt.plot(xline,yline3,label='3号池')
plt.scatter(xpoint,ypoint4)
plt.plot(xline,yline4,label='4号池')
plt.xticks([])
#plt.xlabel('时间（周）')
plt.ylabel('轮虫(10^6/L)')
plt.title('轮虫-时间（周）曲线图')
plt.legend()#显示标签
plt.show()
```

参考链接

- 《Python科学计算》-张若愚
- SymPy.solve()官方文档：https://docs.sympy.org/latest/tutorial/solvers.html
- SciPy.interpolate.splrep()官方文档（英文版）：https://docs.scipy.org/doc/scipy-0.19.0/reference/generated/scipy.interpolate.splrep.html
- SciPy.interpolate.splrep()官方文档（中文版）：https://vimsky.com/examples/usage/python-scipy.interpolate.splrep.html
- Python.SciPy实现Hermite插值：http://liao.cpython.org/scipy13/
- 常微分方程数值解：Python求解：https://www.jianshu.com/p/8d3671f9148d
- 微分方程在物理学中的应用：
  https://zhuanlan.zhihu.com/p/81488678
  https://zhuanlan.zhihu.com/p/164627678