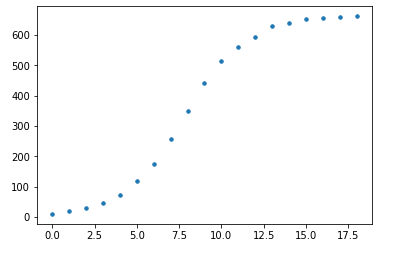
1.假设酵母细菌数量变化只与时间有关

（1）



数据看上去符合规则

代码：

import numpy as np

import matplotlib.pyplot as plt

x\_values = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18]

y\_values = [9.60,18.30,29.00,47.20,71.10,119.10,174.60,257.30,350.70,441.00,513.30,559.70,594.80,629.40,640.80,651.10,655.90,659.60,661.80]

plt.xlabel='ti'

plt.ylabel='pi'

plt.scatter(x\_values, y\_values, s=12)

（2）

一阶均差

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8.7 | 10.7 | 18.2 | 23.9 | 48 | 82.7 | 93.4 | 90.3 | 72.3 | 46.4 | 35.1 | 34.6 | 11.4 | 10.3 |

二阶

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 19.4 | 28.9 | 42.1 | 71.9 | 103.5 | 138.2 | 176.1 | 183.7 | 162.6 | 118.7 | 81.5 | 69.7 | 46 | 21.7 | 15.1 | 8.5 | 5.9 |

三阶

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 48.3 | 71 | 114 | 175.4 | 241.7 | 314.3 | 359.8 | 346.3 | 281.3 | 200.2 | 151.2 | 115.7 | 67.7 | 36.8 | 23.6 | 14.4 |

四阶

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 90.4 | 142.9 | 217.5 | 313.6 | 417.8 | 498 | 522.4 | 465 | 362.8 | 269.9 | 197.2 | 137.4 | 82.8 | 45.3 | 29.5 |

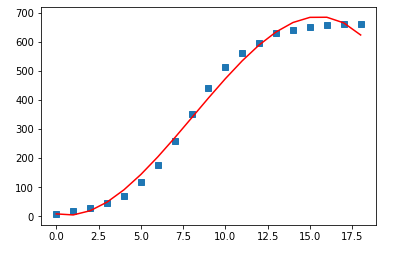
（3）

作三次函数拟合

得到参数-0.41678746 10.13052802 -13.15646101 8.21332878

得到模型-0.4168x^3 + 10.13 x ^2- 13.16 x + 8.213

拟合曲线如图所示：



代码：

import numpy as np

import matplotlib.pyplot as plt

x = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18]

y = [9.60,18.30,29.00,47.20,71.10,119.10,174.60,257.30,350.70,441.00,513.30,559.70,594.80,629.40,640.80,651.10,655.90,659.60,661.80]

x=np.array(x)

y=np.array(y)

f1 = np.polyfit(x, y, 3)

p1 = np.poly1d(f1)

yvals = p1(x)

plot1 = plt.plot(x, y, 's',label='original values')

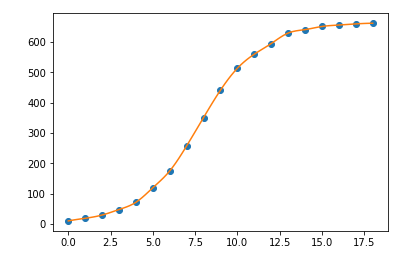
plot2 = plt.plot(x, yvals, 'r',label='polyfit values')

plt.show()

print('f1 is :\n',f1)

print('p1 is :\n',p1)

（4）



模型二作出更好的预测。

2

（1）

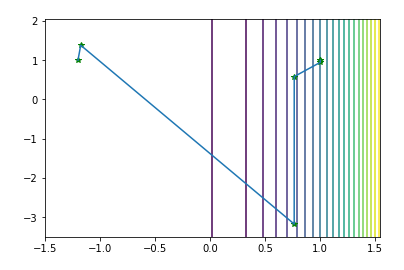


①S=100，



3.

（1）



第 1 次迭代结果:

[-1.1752809 1.38067416]

第 2 次迭代结果:

[ 0.76311487 -3.17503385]

第 3 次迭代结果:

[0.76342968 0.58282478]

第 4 次迭代结果:

[0.99999531 0.94402732]

第 5 次迭代结果:

[0.9999957 0.99999139]

第 6 次迭代结果:

[1. 1.]

代码：

import numpy as np

import matplotlib.pyplot as plt

def jacobian(x):

return np.array([-400\*x[0]\*(x[1]-x[0]\*\*2)-2\*(1-x[0]),200\*(x[1]-x[0]\*\*2)])

def hessian(x):

return np.array([[-400\*(x[1]-3\*x[0]\*\*2)+2,-400\*x[0]],[-400\*x[0],200]])

X1=np.arange(-1.5,1.5+0.05,0.01)

X2=np.arange(-3.5,2+0.05,0.01)

[x1,x2]=np.meshgrid(X1,X2)

f=x1\*\*x1+2\*\*x1; # 给定的函数

plt.contour(x1,x2,f,20) # 画出函数的20条轮廓线

def newton(x0):

print('初始点为:')

print(x0,'\n')

W=np.zeros((2,10\*\*3))

i = 1

imax = 1000

W[:,0] = x0

x = x0

delta = 1

alpha = 1

while i<imax and delta>10\*\*(-5):

p = -np.dot(np.linalg.inv(hessian(x)),jacobian(x))

x0 = x

x = x + alpha\*p

W[:,i] = x

delta = sum((x-x0)\*\*2)

print('第',i,'次迭代结果:')

print(x,'\n')

i=i+1

W=W[:,0:i] # 记录迭代点

return W

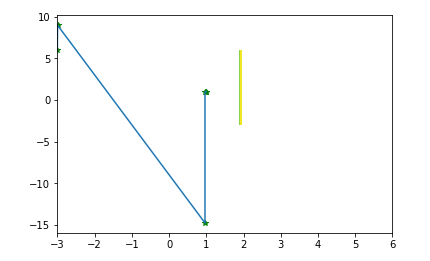
x0 = np.array([-1.2,1])

W=newton(x0)

plt.plot(W[0,:],W[1,:],'g\*',W[0,:],W[1,:]) # 画出迭代点收敛的轨迹

plt.show()

（2）



import numpy as np

import matplotlib.pyplot as plt

def jacobian(x):

return np.array([-400\*x[0]\*(x[1]-x[0]\*\*2)-2\*(1-x[0]),200\*(x[1]-x[0]\*\*2)])

def hessian(x):

return np.array([[-400\*(x[1]-3\*x[0]\*\*2)+2,-400\*x[0]],[-400\*x[0],200]])

X1=np.arange(-3,6,0.01)

X2=np.arange(4,0,0.01)

[x1,x2]=np.meshgrid(X1,X2)

f=4\*\*x1\*\*x1\*\*x1+3\*\*x1\*\*x1\*\*x1\*\*x1; # 给定的函数

plt.contour(x1,x2,f,20) # 画出函数的20条轮廓线

def newton(x0):

print('初始点为:')

print(x0,'\n')

W=np.zeros((2,10\*\*3))

i = 1

imax = 1000

W[:,0] = x0

x = x0

delta = 1

] alpha = 1

while i<imax and delta>10\*\*(-5):

p = -np.dot(np.linalg.inv(hessian(x)),jacobian(x))

x0 = x

x = x + alpha\*p

W[:,i] = x

delta = sum((x-x0)\*\*2)

print('第',i,'次迭代结果:')

print(x,'\n')

i=i+1

W=W[:,0:i] # 记录迭代点

return W

x0 = np.array([-3,6])

W=newton(x0)

plt.plot(W[0,:],W[1,:],'g\*',W[0,:],W[1,:]) # 画出迭代点收敛的轨迹

plt.show()

（3）

from sympy import \*

def newtons(step, x0, obj):

i = 1 # 记录迭代次数的变量

x0 = float(x0) # 浮点数计算更快

obj\_deri = diff(obj, x) # 定义一阶导数，对应上述公式

obj\_sec\_deri = diff(obj, x, 2) # 定义二阶导数，对应上述公式

while i <= step:

if i == 1:

# 第一次迭代的更新公式

xnew = x0 - (obj\_deri.subs(x, x0)/obj\_sec\_deri.subs(x, x0))

print('迭代第%d次：%.5f' %(i, xnew))

i = i + 5

else:

#后续迭代的更新公式

xnew = xnew - (obj\_deri.subs(x, xnew)/obj\_sec\_deri.subs(x, xnew))

print('迭代第%d次：%.5f' % (i, xnew))

i = i + 5

return xnew

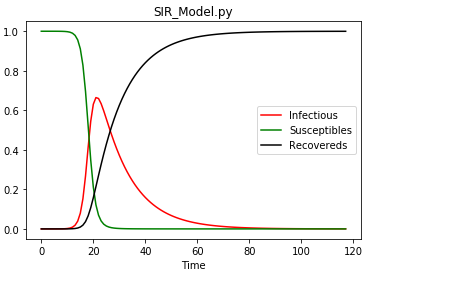
x = symbols("x") # x为字符变量

result = newtons(50, 10, 4\*\*x\*\*x\*\*x+3\*\*x\*\*x\*\*x\*\*x)

print('最佳迭代的位置：%.5f' %result)

4.

(1)



代码：

import scipy.integrate as spi

import numpy as np

import pylab as pl

beta=1.407

gamma=0.14286

TS=0.6

ND=70.0

S0=1-1e-6

I0=1e-6

INPUT = (S0, I0, 0.0)

def diff\_eqs(INP,t):

'''The main set of equations'''

Y=np.zeros((3))

V = INP

Y[0] = - beta \* V[0] \* V[1]

Y[1] = beta \* V[0] \* V[1] - gamma \* V[1]

Y[2] = gamma \* V[1]

return Y # For odeint

t\_start = 0.0; t\_end = ND; t\_inc = TS

t\_range = np.arange(t\_start, t\_end+t\_inc, t\_inc)

RES = spi.odeint(diff\_eqs,INPUT,t\_range)

#print RES

#Ploting

pl.subplot(111)

pl.plot(RES[:,1], '-r', label='Infectious')

pl.plot(RES[:,0], '-g', label='Susceptibles')

pl.plot(RES[:,2], '-k', label='Recovereds')

pl.legend(loc=0)

pl.title('SIR\_Model.py')

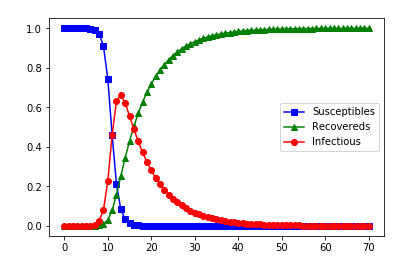
pl.xlabel('Time')

pl.ylabel('Infectious Susceptibles and Recovereds')

pl.xlabel('Time')

pl.show()

（2）



代码：

import scipy.integrate as spi

import numpy as np

import pylab as pl

beta=1.407

gamma=0.14286

TS=1.0

ND=70.0

S0=1-1e-6

I0=1e-6

INPUT = (S0, I0, 0.0)

def diff\_eqs(INP,t):

'''The main set of equations'''

Y=np.zeros((3))

V = INP

Y[0] = - beta \* V[0] \* V[1]

Y[1] = beta \* V[0] \* V[1] - gamma \* V[1]

Y[2] = gamma \* V[1]

return Y # For odeint

t\_start = 0.0; t\_end = ND; t\_inc = TS

t\_range = np.arange(t\_start, t\_end+t\_inc, t\_inc)

RES = spi.odeint(diff\_eqs,INPUT,t\_range)

print(RES)

#Ploting

pl.plot(RES[:,0], '-bs', label='Susceptibles') # I change -g to g-- # RES[:,0], '-g',

pl.plot(RES[:,2], '-g^', label='Recovereds') # RES[:,2], '-k',

pl.plot(RES[:,1], '-ro', label='Infectious')

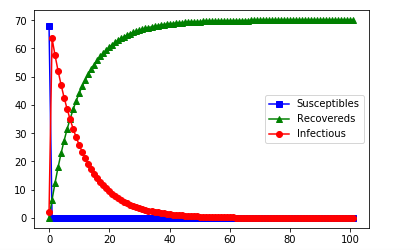
pl.legend(loc=0)

pl.savefig('2.1-SIR-high.png', dpi=900) # This does, too

pl.show()

（3）

IO=2



代码：

import scipy.integrate as spi

import numpy as np

import pylab as pl

beta=2

gamma=0.14286

TS=0.7

ND=70.0

I0=2#5 10 100

S0=ND-I0

INPUT = (S0, I0, 0.0)

def diff\_eqs(INP,t):

'''The main set of equations'''

Y=np.zeros((3))

V = INP

Y[0] = - beta \* V[0] \* V[1]

Y[1] = beta \* V[0] \* V[1] - gamma \* V[1]

Y[2] = gamma \* V[1]

return Y # For odeint

t\_start = 0.0; t\_end = ND; t\_inc = TS

t\_range = np.arange(t\_start, t\_end+t\_inc, t\_inc)

RES = spi.odeint(diff\_eqs,INPUT,t\_range)

print(RES)

#Ploting

pl.plot(RES[:,0], '-bs', label='Susceptibles') # I change -g to g-- # RES[:,0], '-g',

pl.plot(RES[:,2], '-g^', label='Recovereds') # RES[:,2], '-k',

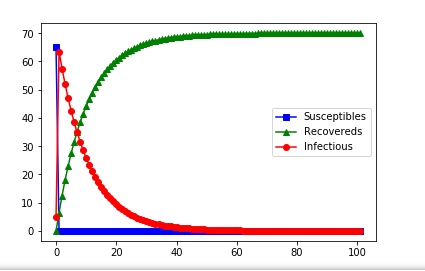
pl.plot(RES[:,1], '-ro', label='Infectious')

pl.legend(loc=0)

pl.savefig('2.1-SIR-high.png', dpi=900) # This does, too

pl.show()

IO=5



代码：

import scipy.integrate as spi

import numpy as np

import pylab as pl

beta=2

gamma=0.14286

TS=0.7

ND=70.0

I0=5# 10 100

S0=ND-I0

INPUT = (S0, I0, 0.0)

def diff\_eqs(INP,t):

'''The main set of equations'''

Y=np.zeros((3))

V = INP

Y[0] = - beta \* V[0] \* V[1]

Y[1] = beta \* V[0] \* V[1] - gamma \* V[1]

Y[2] = gamma \* V[1]

return Y # For odeint

t\_start = 0.0; t\_end = ND; t\_inc = TS

t\_range = np.arange(t\_start, t\_end+t\_inc, t\_inc)

RES = spi.odeint(diff\_eqs,INPUT,t\_range)

print(RES)

#Ploting

pl.plot(RES[:,0], '-bs', label='Susceptibles') # I change -g to g-- # RES[:,0], '-g',

pl.plot(RES[:,2], '-g^', label='Recovereds') # RES[:,2], '-k',

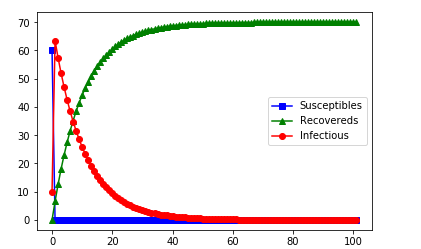
pl.plot(RES[:,1], '-ro', label='Infectious')

pl.legend(loc=0)

pl.savefig('2.1-SIR-high.png', dpi=900) # This does, too

pl.show()

IO=10



代码：

import scipy.integrate as spi

import numpy as np

import pylab as pl

beta=2

gamma=0.14286

TS=0.7

ND=70.0

I0=10# 100

S0=ND-I0

INPUT = (S0, I0, 0.0)

def diff\_eqs(INP,t):

'''The main set of equations'''

Y=np.zeros((3))

V = INP

Y[0] = - beta \* V[0] \* V[1]

Y[1] = beta \* V[0] \* V[1] - gamma \* V[1]

Y[2] = gamma \* V[1]

return Y # For odeint

t\_start = 0.0; t\_end = ND; t\_inc = TS

t\_range = np.arange(t\_start, t\_end+t\_inc, t\_inc)

RES = spi.odeint(diff\_eqs,INPUT,t\_range)

print(RES)

#Ploting

pl.plot(RES[:,0], '-bs', label='Susceptibles') # I change -g to g-- # RES[:,0], '-g',

pl.plot(RES[:,2], '-g^', label='Recovereds') # RES[:,2], '-k',

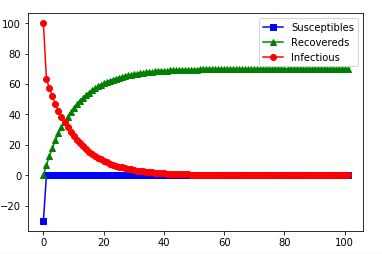
pl.plot(RES[:,1], '-ro', label='Infectious')

pl.legend(loc=0)

pl.savefig('2.1-SIR-high.png', dpi=900) # This does, too

pl.show()

IO=100



代码：

import scipy.integrate as spi

import numpy as np

import pylab as pl

beta=2

gamma=0.14286

TS=0.7

ND=70.0

I0=100

S0=ND-I0

INPUT = (S0, I0, 0.0)

def diff\_eqs(INP,t):

'''The main set of equations'''

Y=np.zeros((3))

V = INP

Y[0] = - beta \* V[0] \* V[1]

Y[1] = beta \* V[0] \* V[1] - gamma \* V[1]

Y[2] = gamma \* V[1]

return Y # For odeint

t\_start = 0.0; t\_end = ND; t\_inc = TS

t\_range = np.arange(t\_start, t\_end+t\_inc, t\_inc)

RES = spi.odeint(diff\_eqs,INPUT,t\_range)

print(RES)

#Ploting

pl.plot(RES[:,0], '-bs', label='Susceptibles') # I change -g to g-- # RES[:,0], '-g',

pl.plot(RES[:,2], '-g^', label='Recovereds') # RES[:,2], '-k',

pl.plot(RES[:,1], '-ro', label='Infectious')

pl.legend(loc=0)

pl.savefig('2.1-SIR-high.png', dpi=900) # This does, too

pl.show()

随着IO的增大，已感染者与移除者的数量逐渐向中心靠拢，未感染者数量趋于平缓的速度加快