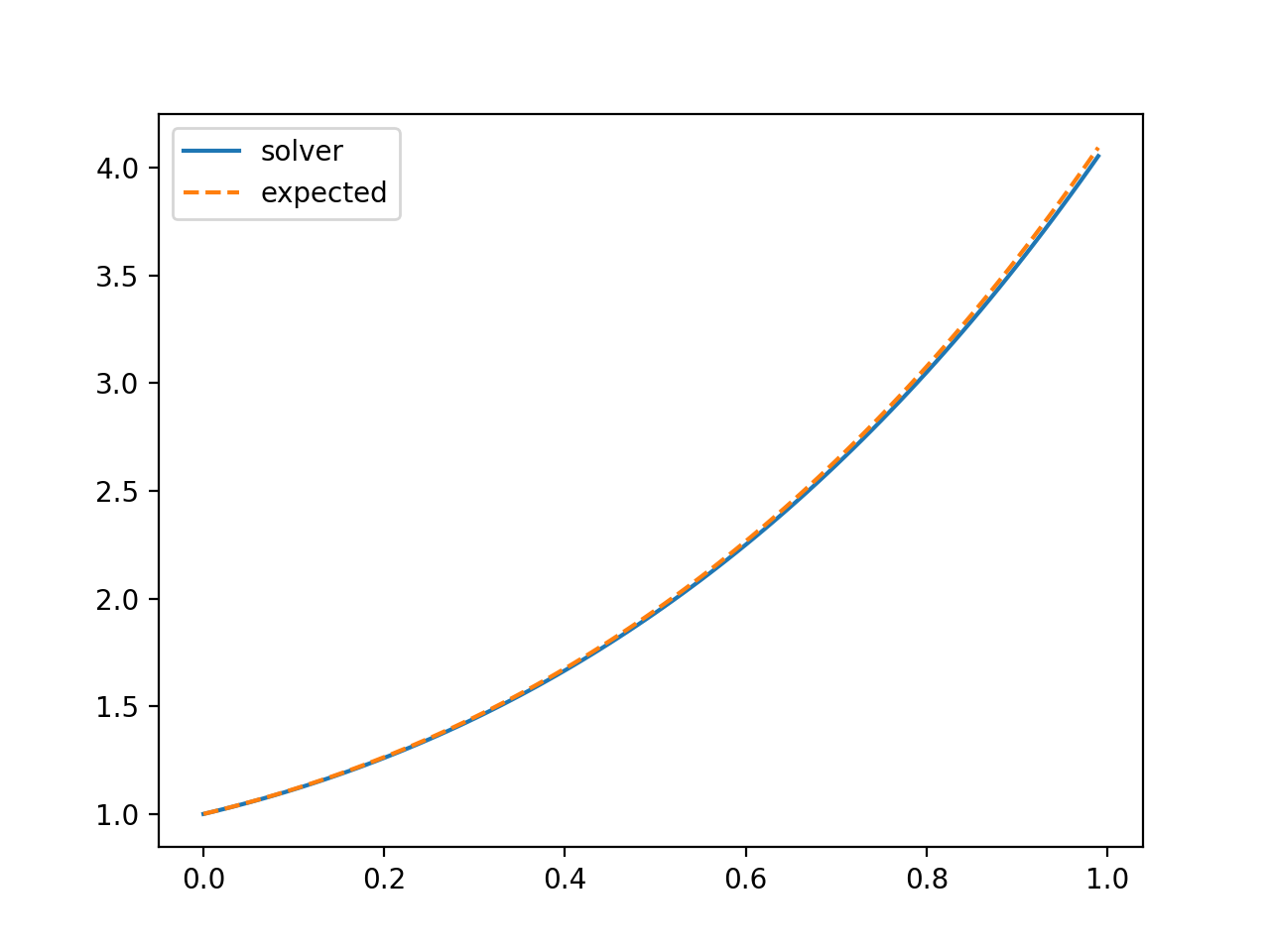
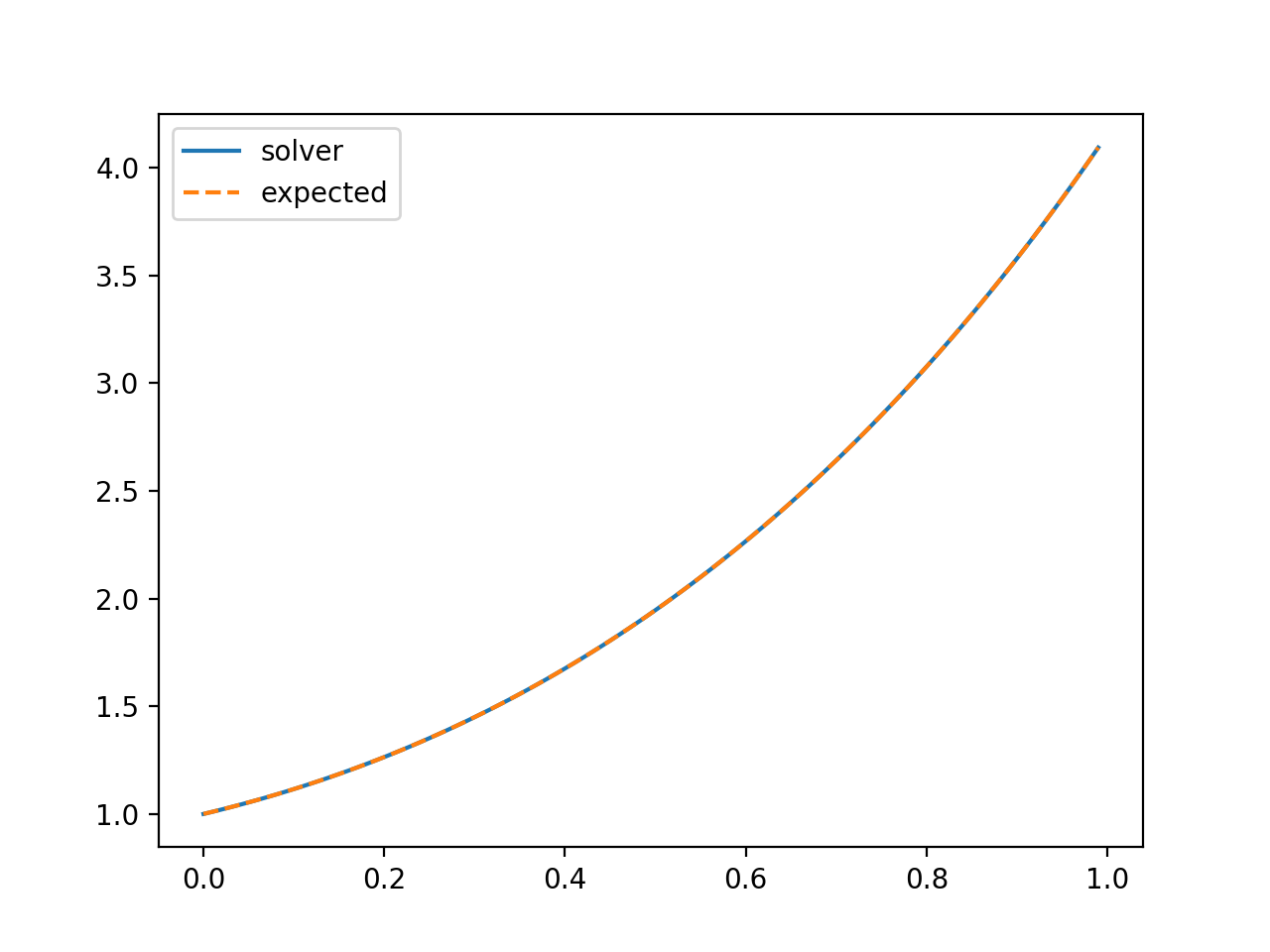
常微分方程初值问题的数值解 实验报告

计65 赖金霖 2016011377

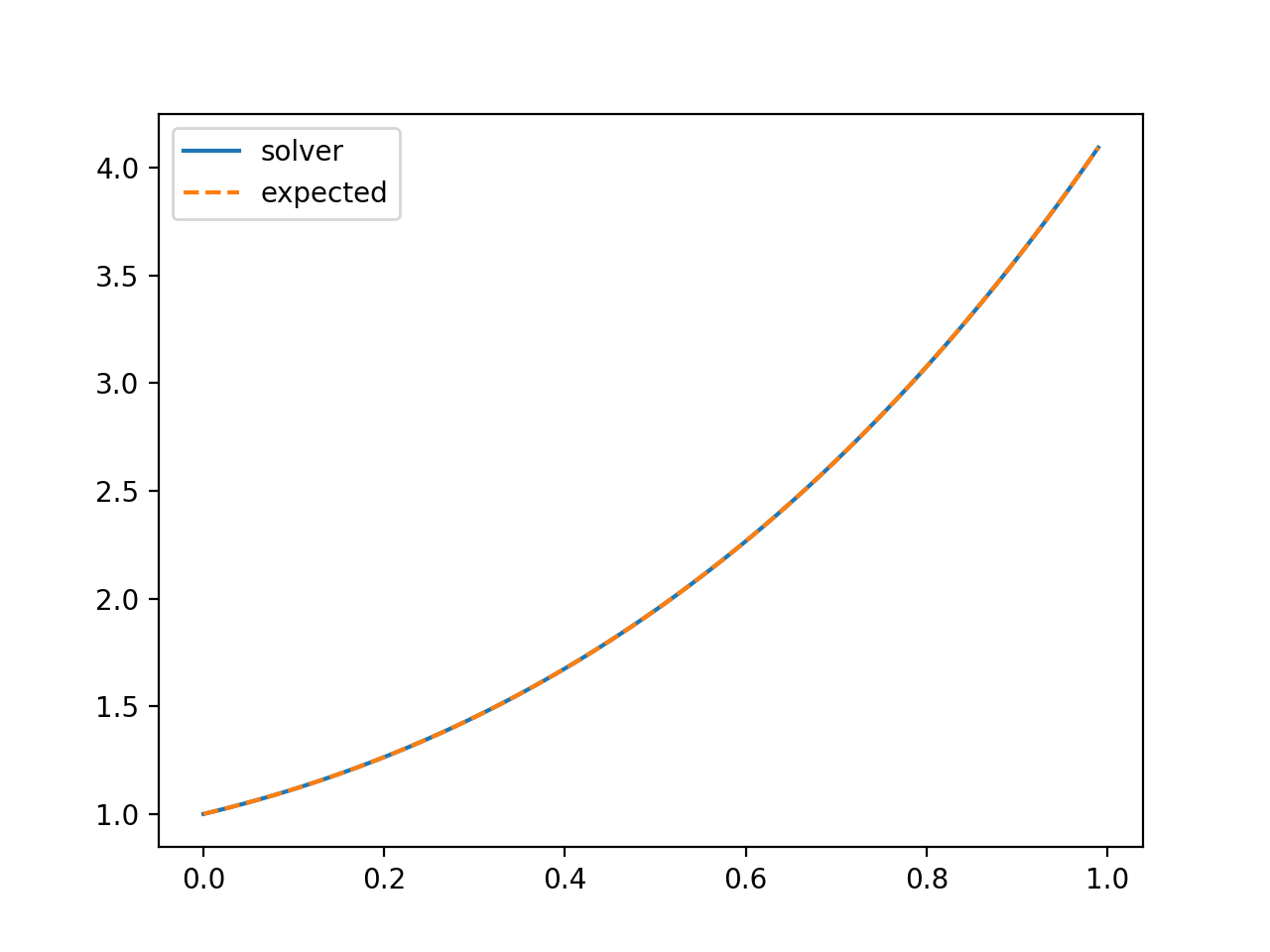
实验代码在https://github.com/lll6924/math\_exp/tree/master/exp2下可以找到，使用的函数在https://github.com/lll6924/math\_exp/blob/master/utils/odesolver.py内。

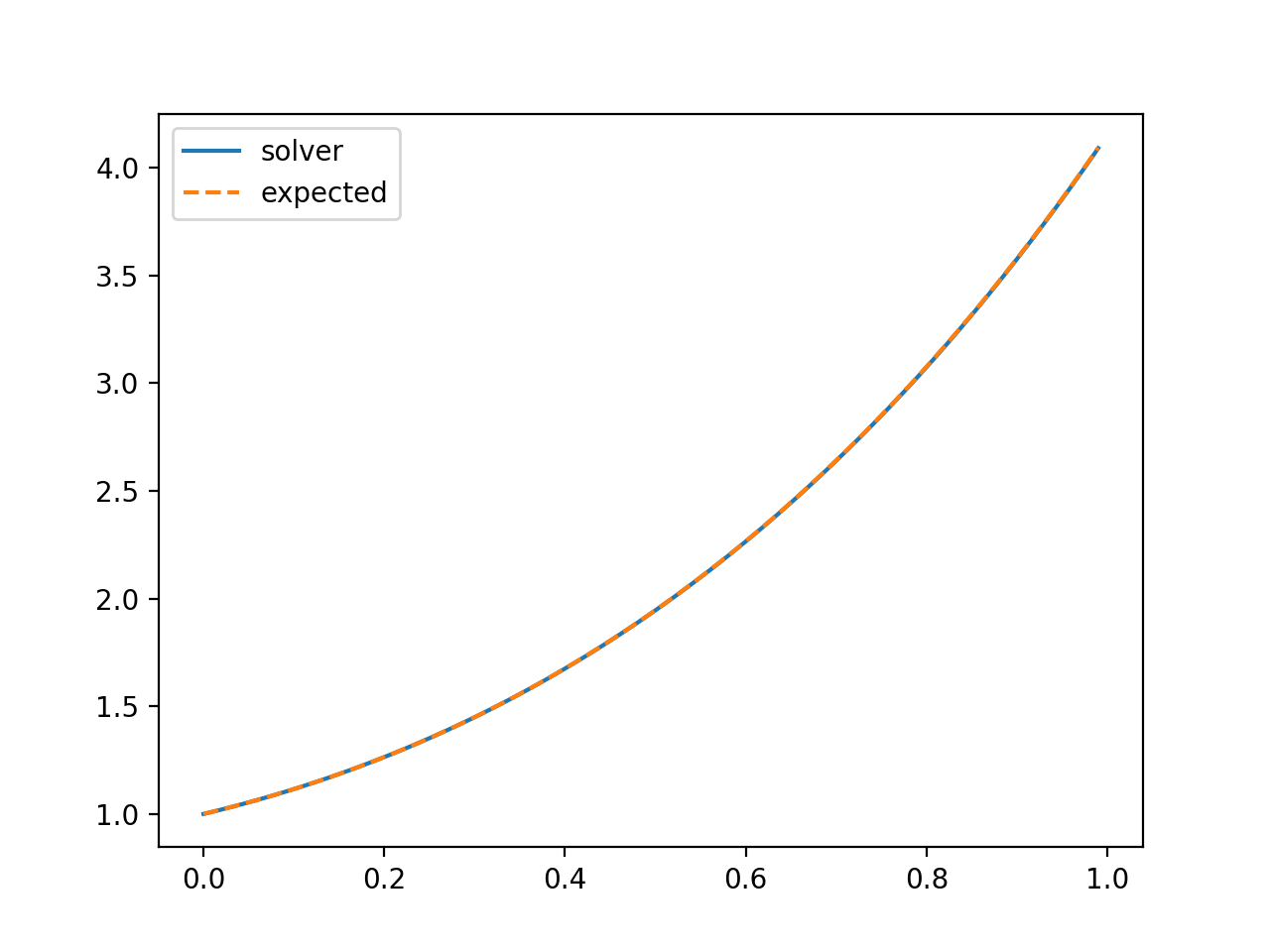
2.（1）步长为0.01

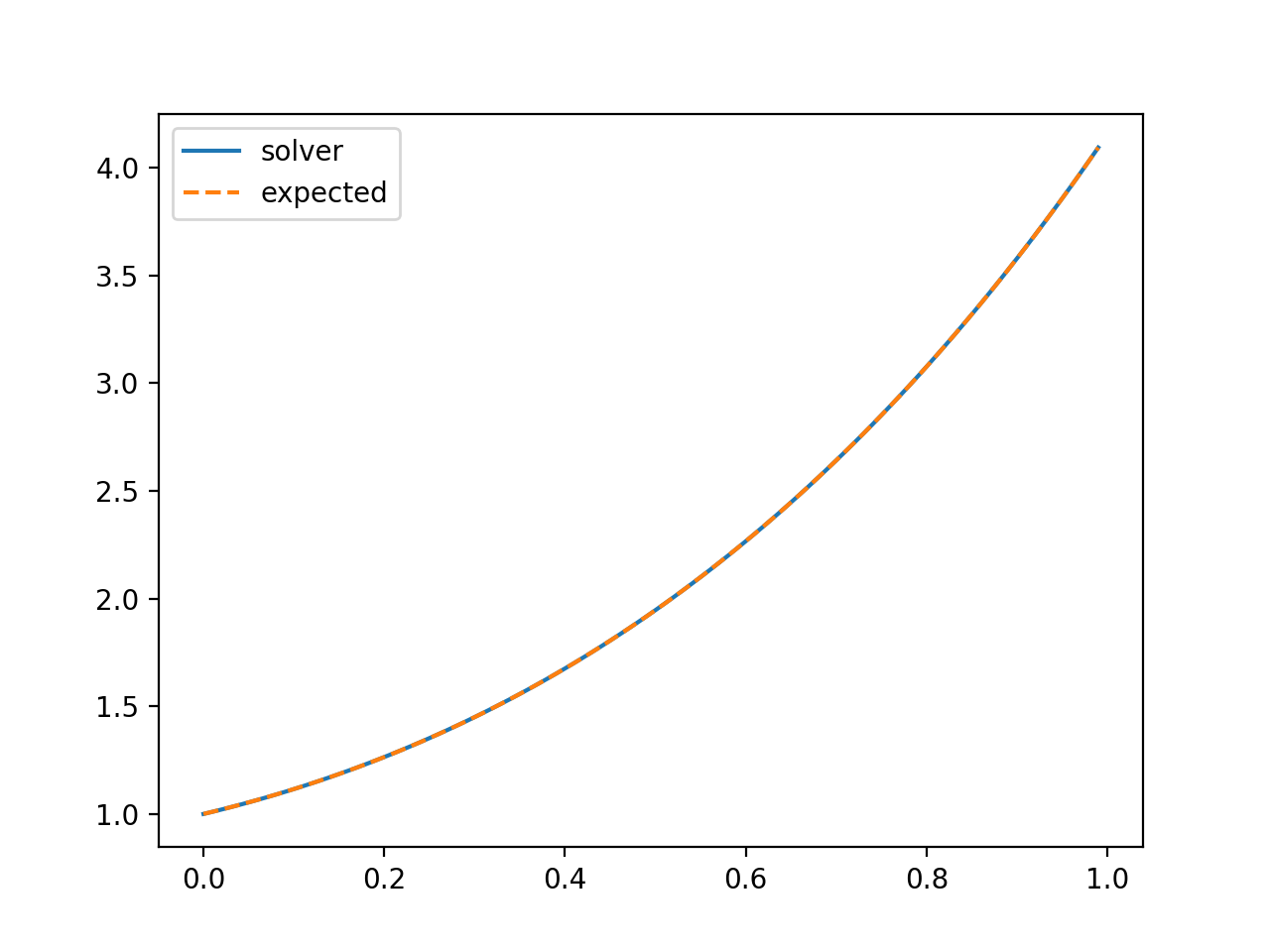
使用欧拉法的效果如下，在x=1处误差为0.10074500194685854，运行1000次耗时1.134s。

使用改进欧拉法如下，x=1处误差为0.0612742882286037，运行1000次耗时1.639s。

使用经典荣格-库塔法如下，x=1处误差为0.06114206898990027，运行1000次耗时2.806s。

使用rk23法如下，x=1处误差为0.06404871606413653，运行1000次耗时1.688s。



使用rk45方法如下，x=1处误差为0.061056342242031825，运行1000次耗时1.443s。

可以看出，库函数比手动实现要快许多，效果最好的是rk45方法。

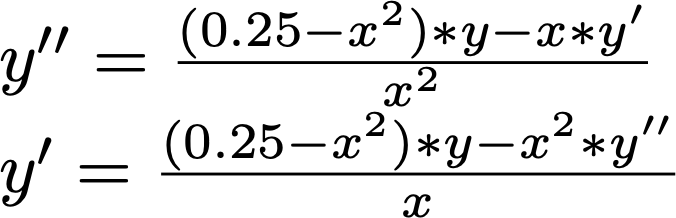
我的代码大致如下

samples\_x = np.arange(0.,1.,0.01)  
func = truefun1()  
solver = rk45(fun1(),0.,[1.],0.,1.,0.01,samples\_x,func(samples\_x))  
solver.plot(1)  
ys=solver.get\_y()  
print(np.abs(func(1.)-ys[0][-1]))

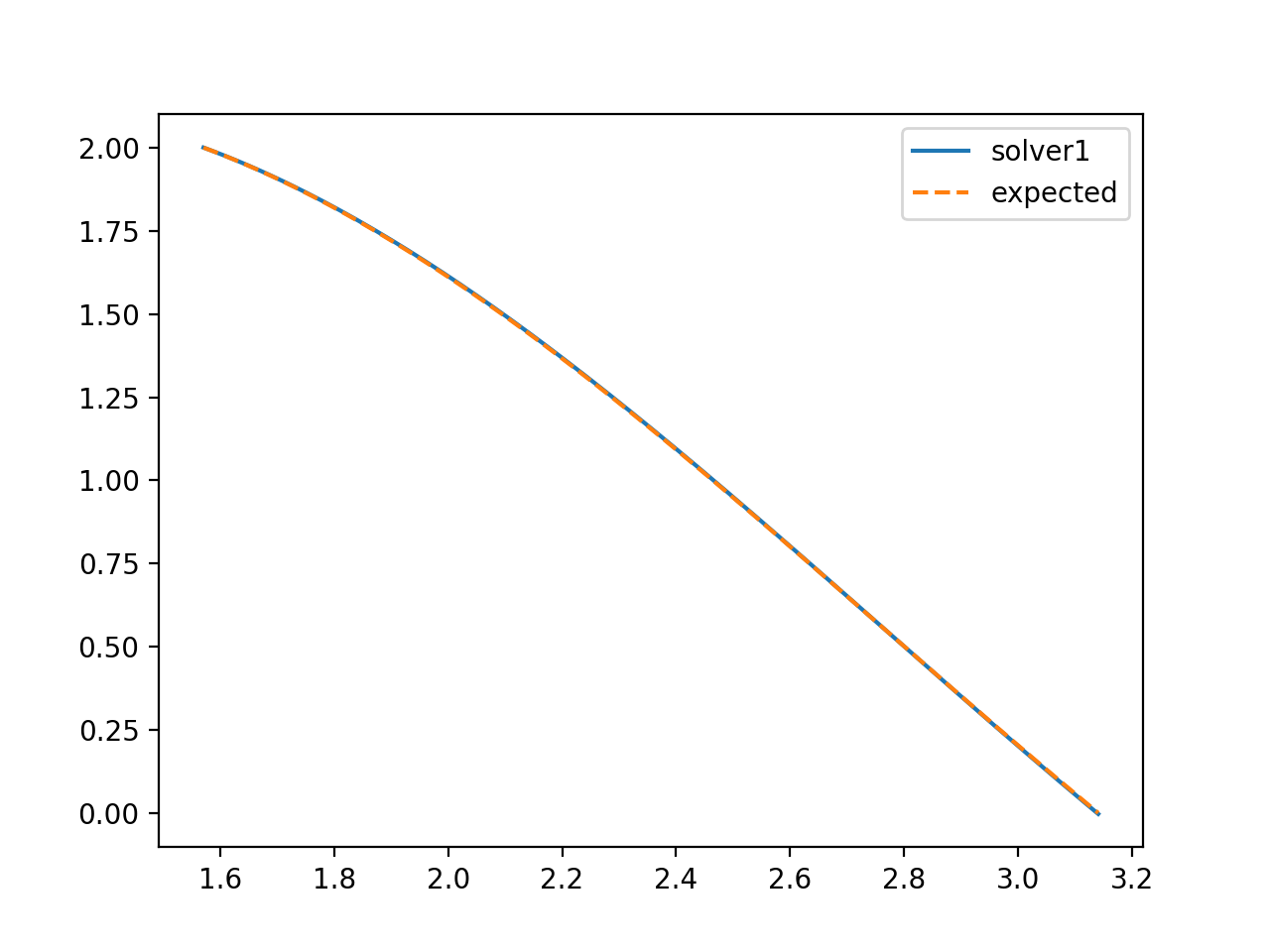
各函数定义和实现在utils/odesolver.py下。

（3）

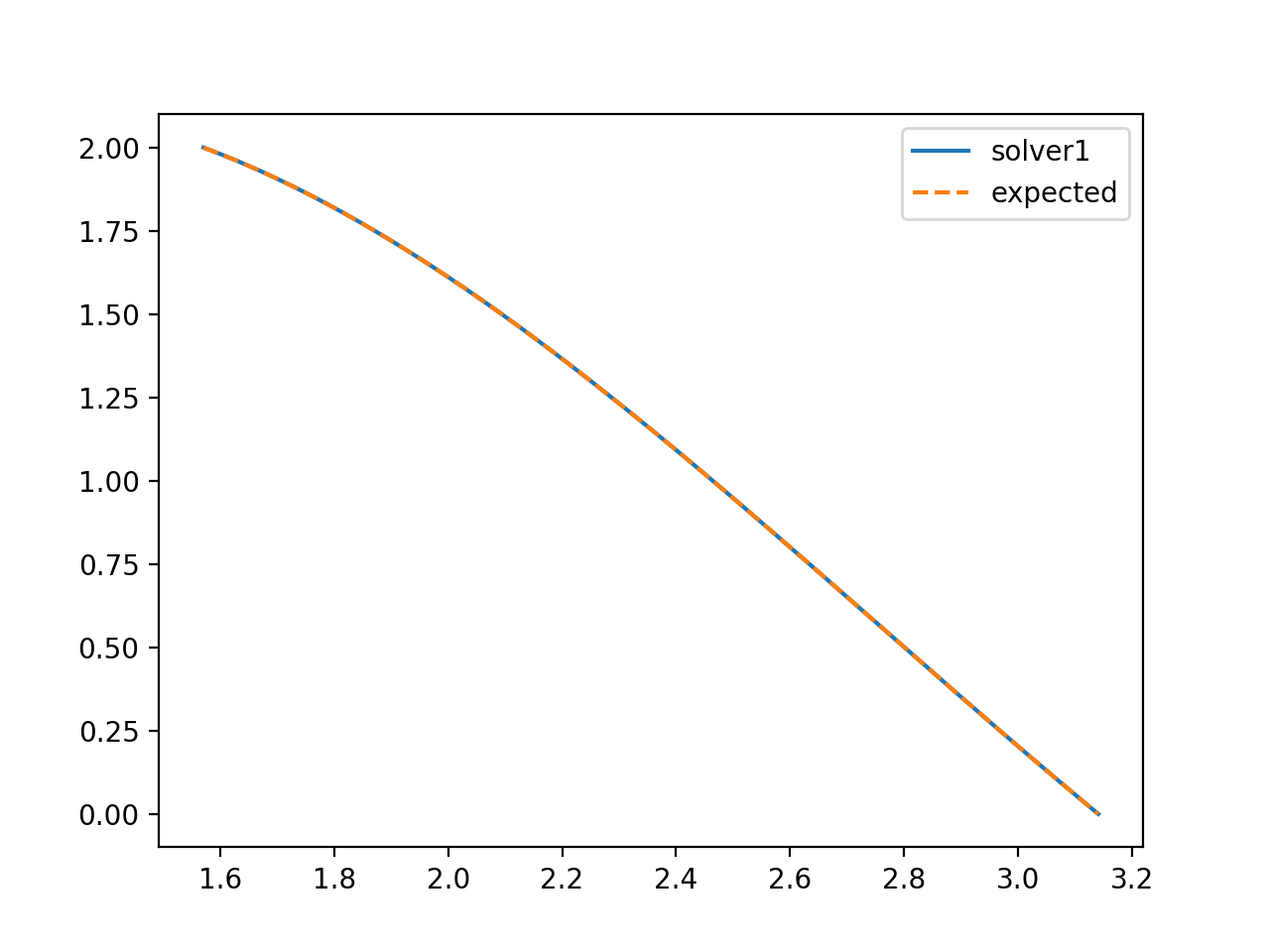
为了解二阶常微分方程，我采用了类似迭代的方法，经过变形，有



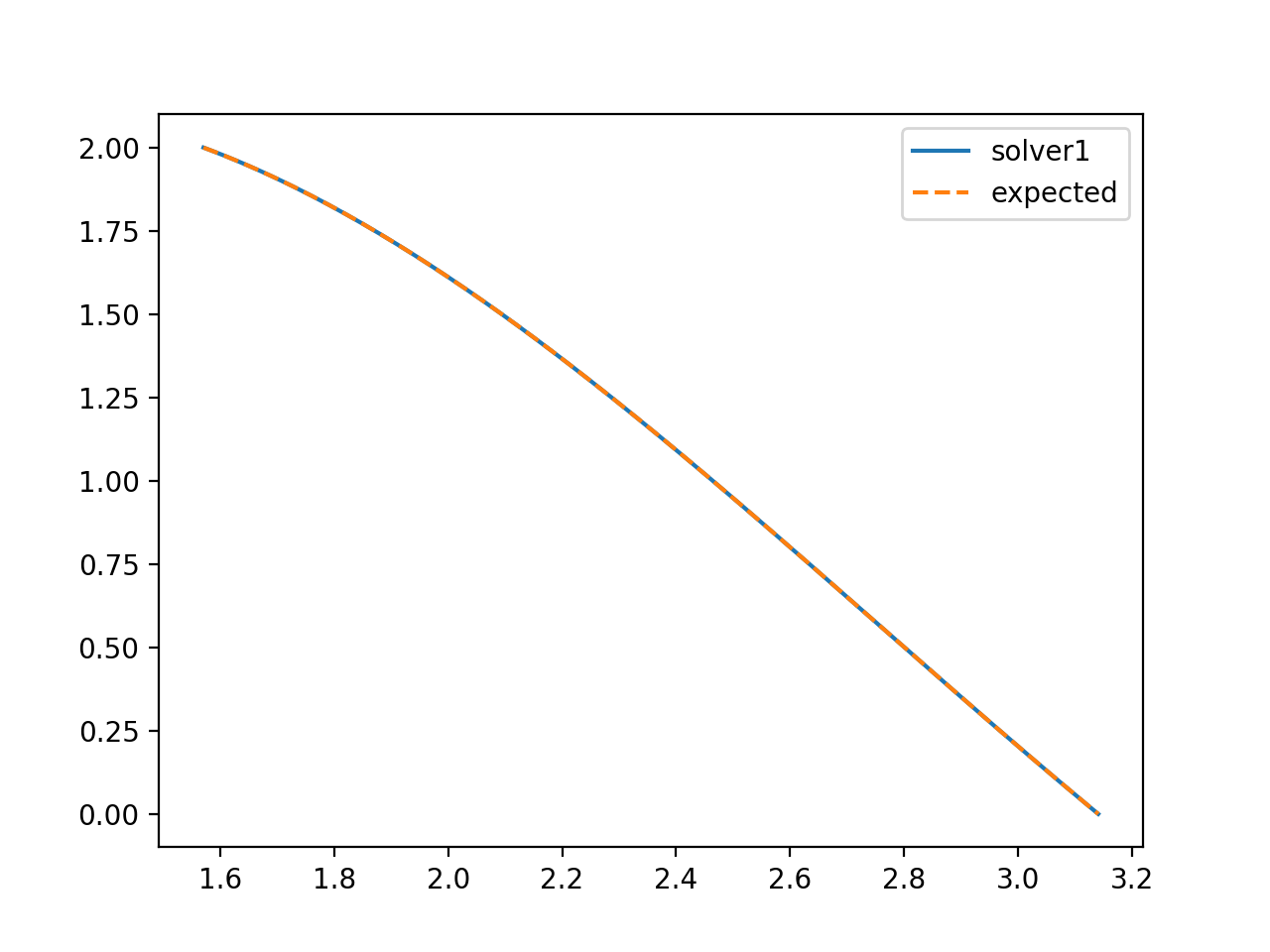
我们把y和y’作为两个同时求取的函数，导数如上定义，其中第二行要使用第一行计算的结果。我的实验区间是[π/2,π]，步长为0.01。

使用欧拉法的效果如下，在x=π处误差为0.001928599127587374，1000次实验耗时2.064s。

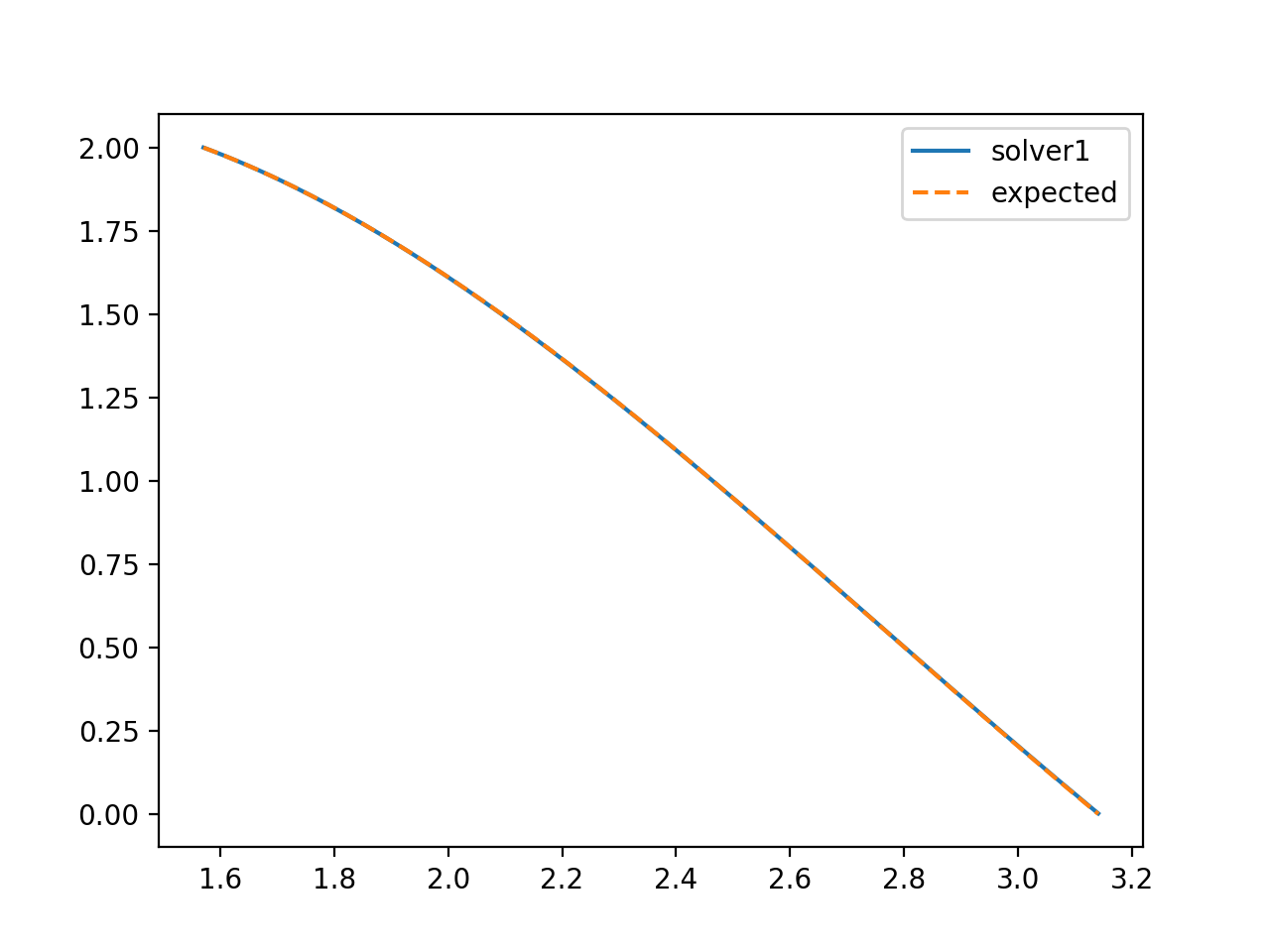
使用改进欧拉法如下，在x=π处误差为0.0011034178094134312，1000次实验耗时4.297s。



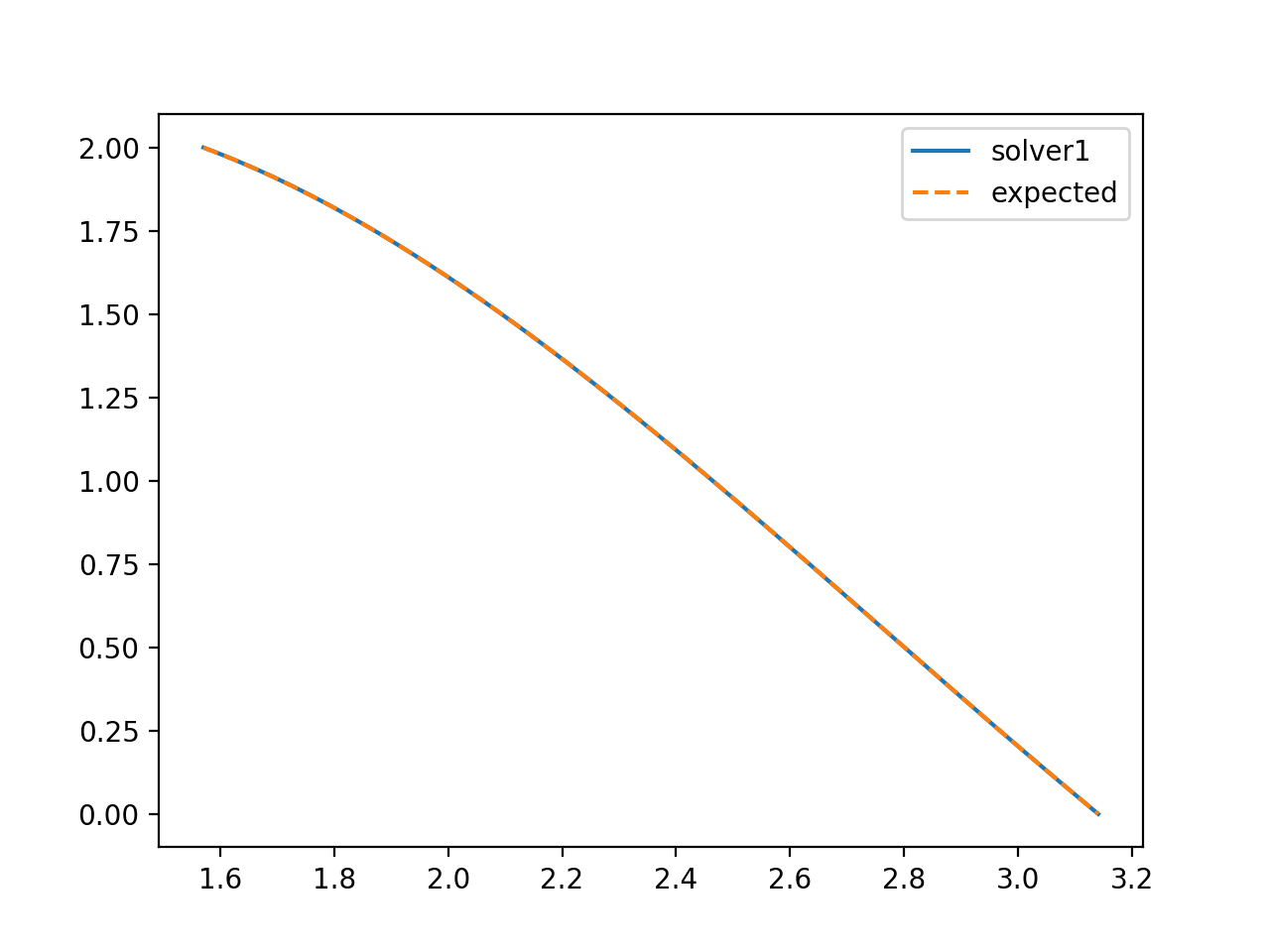
使用经典荣格-库塔方法如下，x=π处误差为0.0011263188524708488，1000次实验耗时6.454s。



使用rk23方法如下，x=π处误差为0.00264478140328988，1000次实验耗时1.932s。

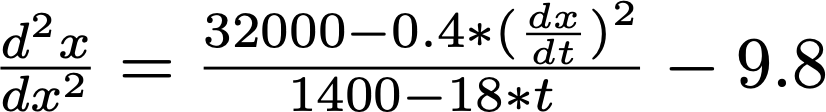


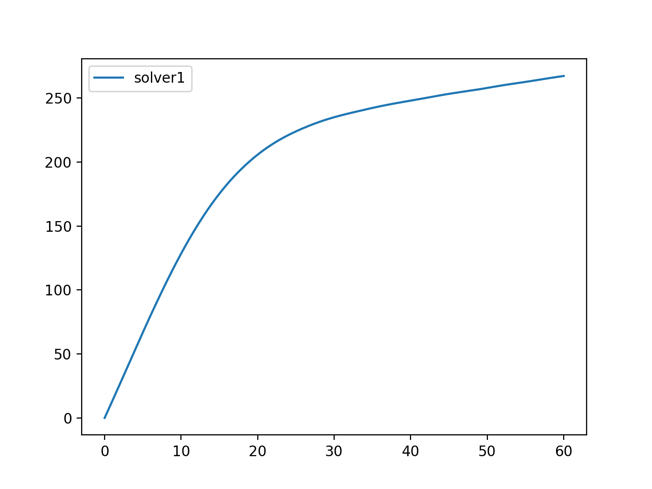
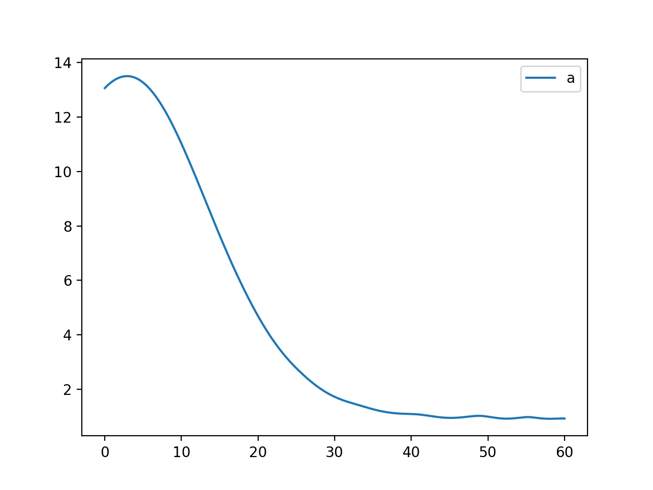
使用rk45方法如下，x=π处误差为0.0011135797070578812，1000次实验耗时1.488s。



可以看出，虽然改进欧拉法的误差最小，但库函数依然十分优秀。

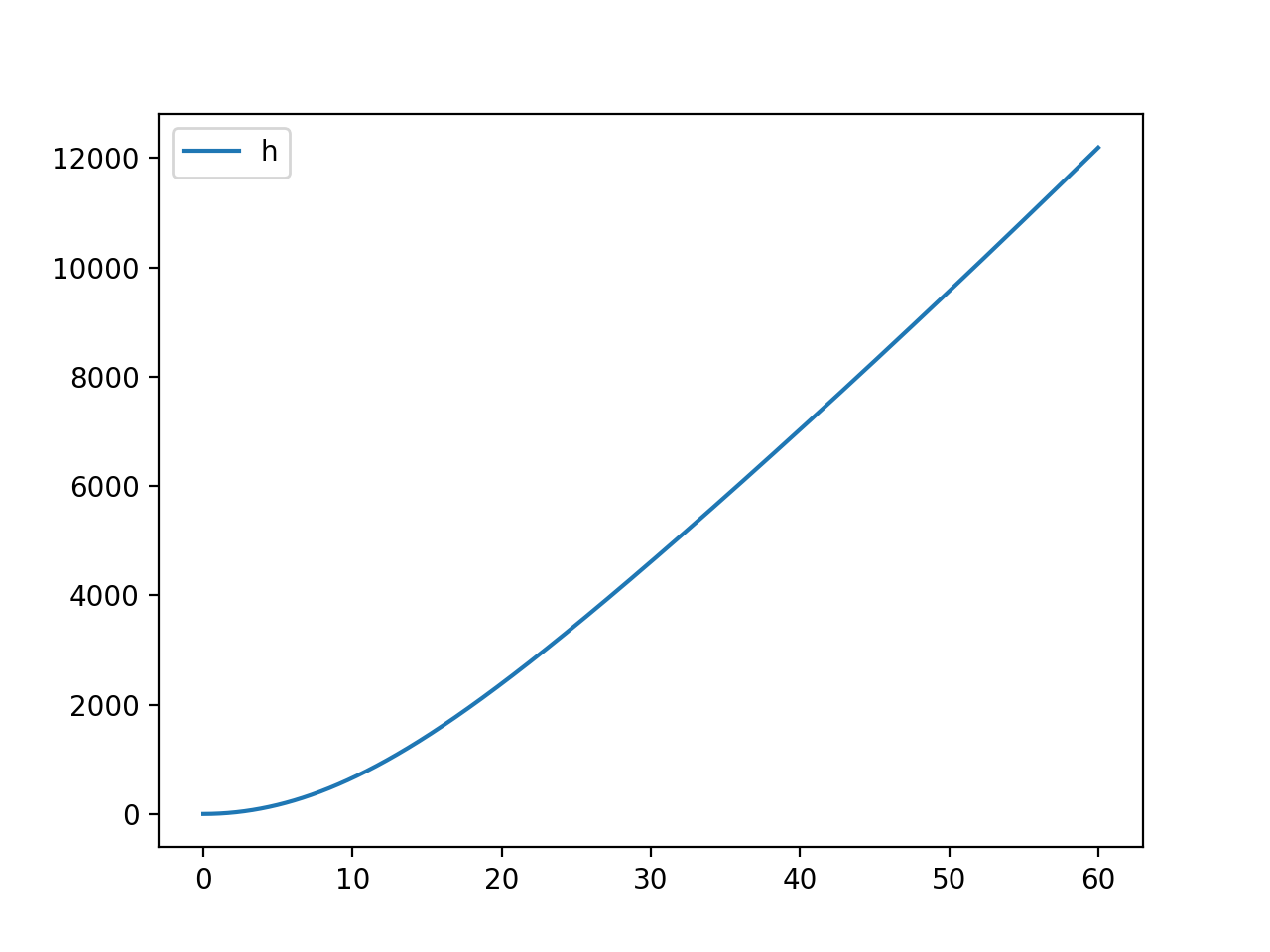
3.

可以列出如下的方程：

解此方程可得如下结果（左侧为速度曲线，右侧为加速度曲线）

在t=60时火箭燃料耗尽，此时高度经简单数值积分计算约为12190.745702660028m，速度为267.23889752532705m/s，加速度为0.9292145618076564。

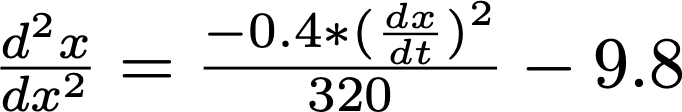
这个过程中高度随时间变化的曲线如下：



代码如下：

f3=fun3()  
solver = rk45(f3,0.,[0.],0.,60.,0.01)  
solver.plot(1)  
xs=solver.get\_x()  
ys=solver.get\_y()[0]  
ans\_x=[]  
ans=0  
**for** y **in** ys:  
 ans+=y  
 ans\_x.append(ans/100.)  
print(ans/100)  
print(ys[-1])  
print(f3(xs[-1],ys[-1]))  
plt.plot(xs, f3(xs,ys), **'-'**)  
plt.legend(**'acceleration'**, loc=**'best'**)  
plt.show()  
plt.plot(xs, ans\_x, **'-'**)  
plt.legend(**'height'**, loc=**'best'**)  
plt.show()

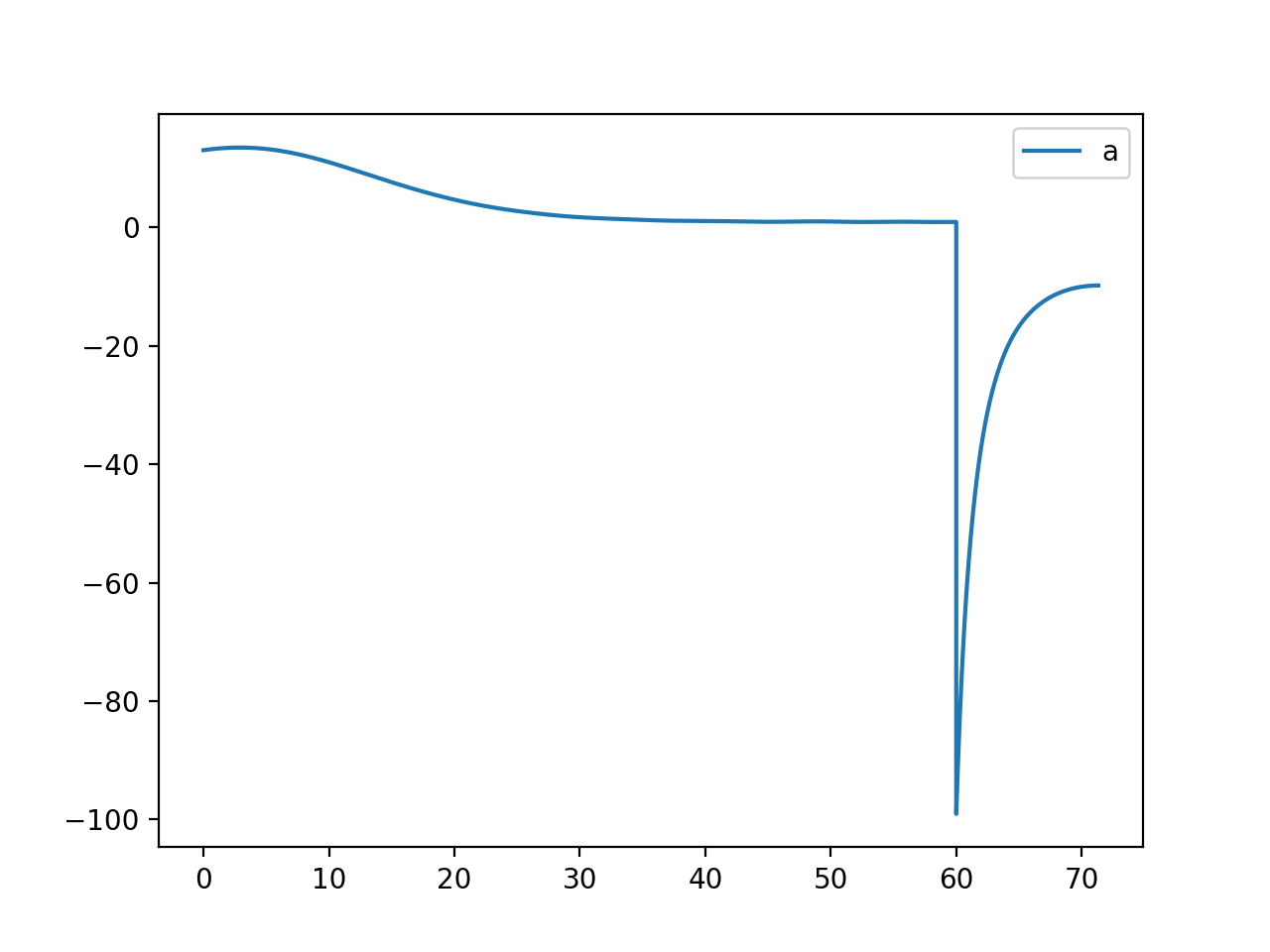
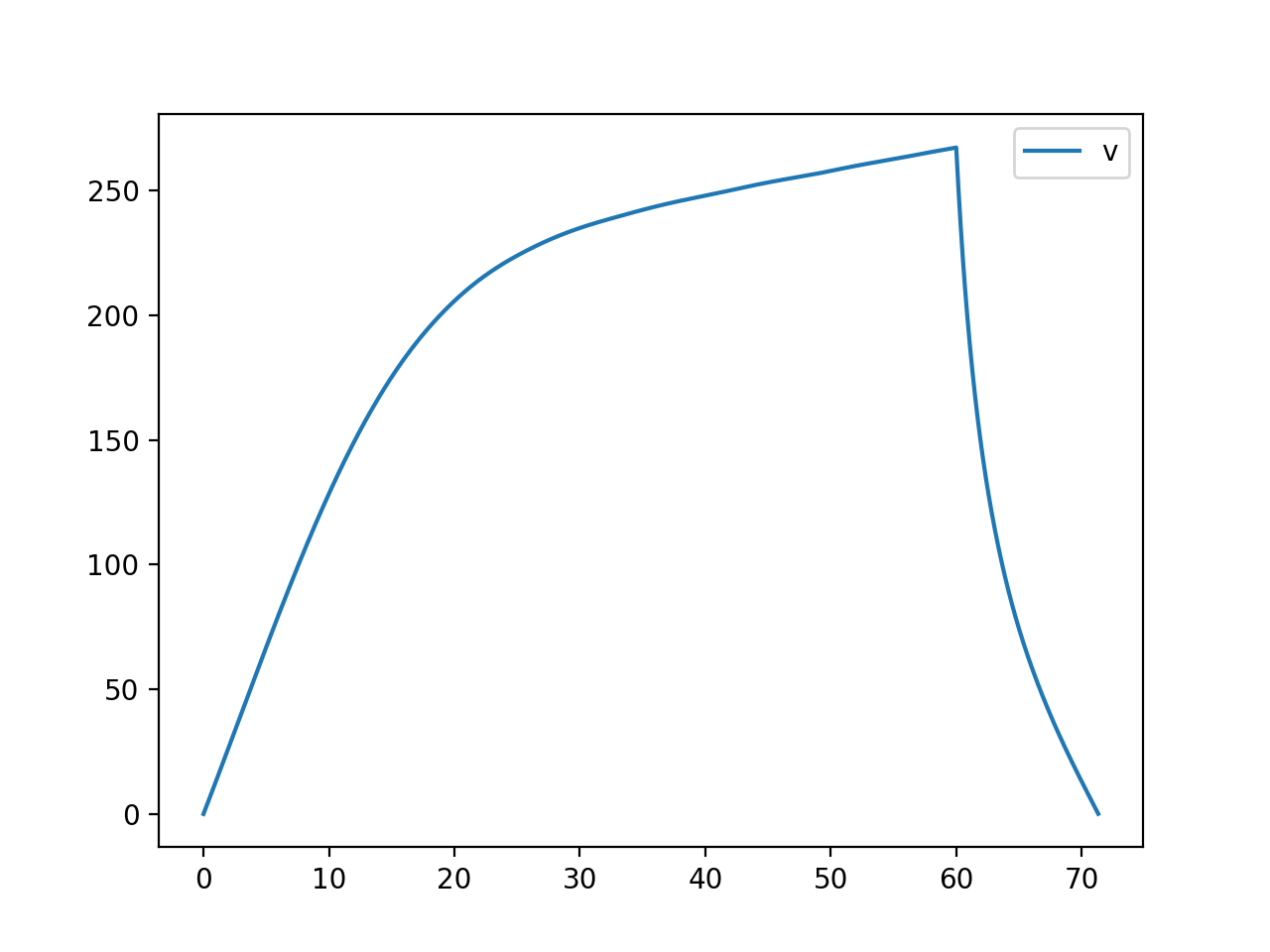
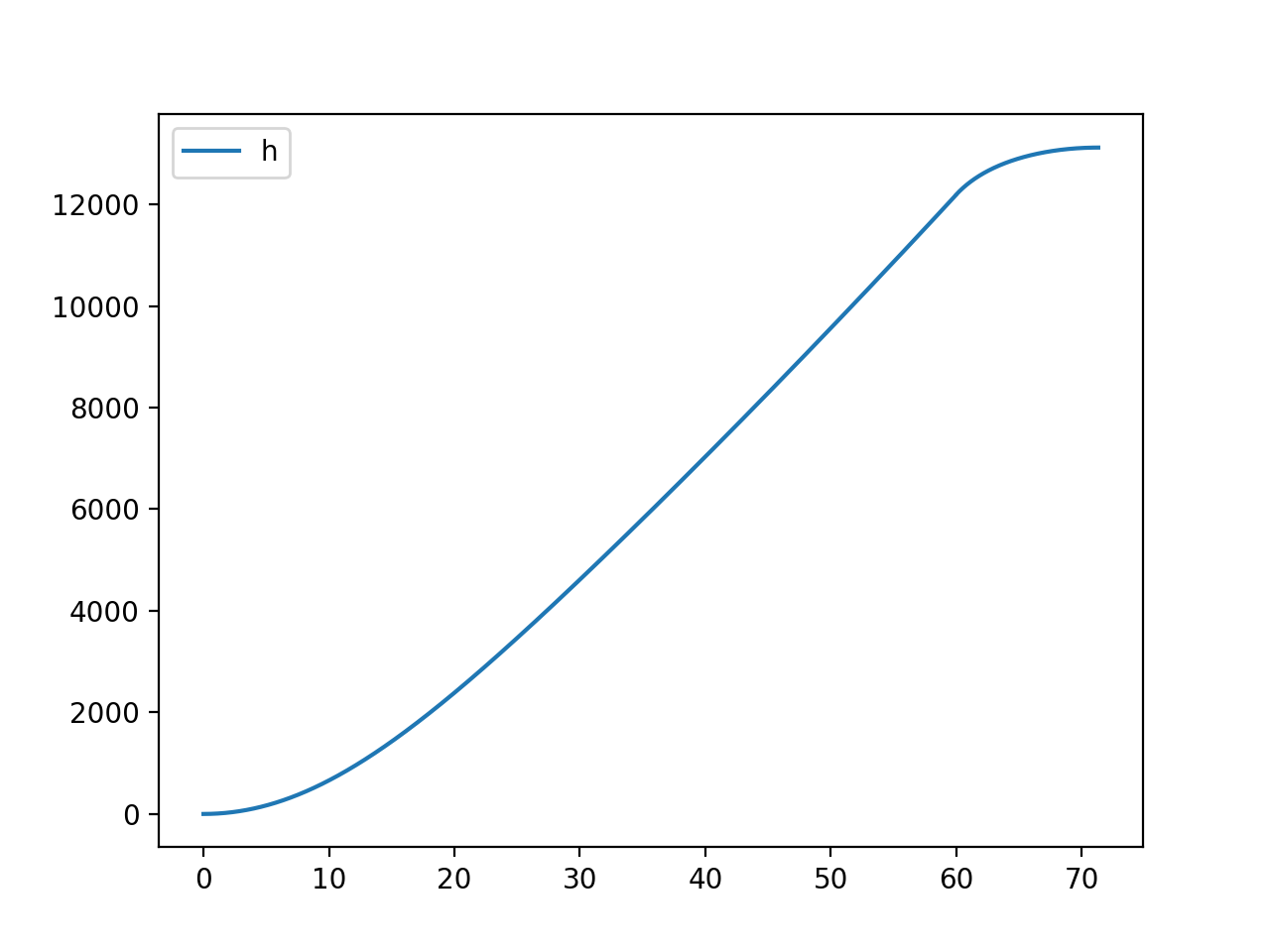
关闭引擎后，火箭继续飞行，模型方程变为：

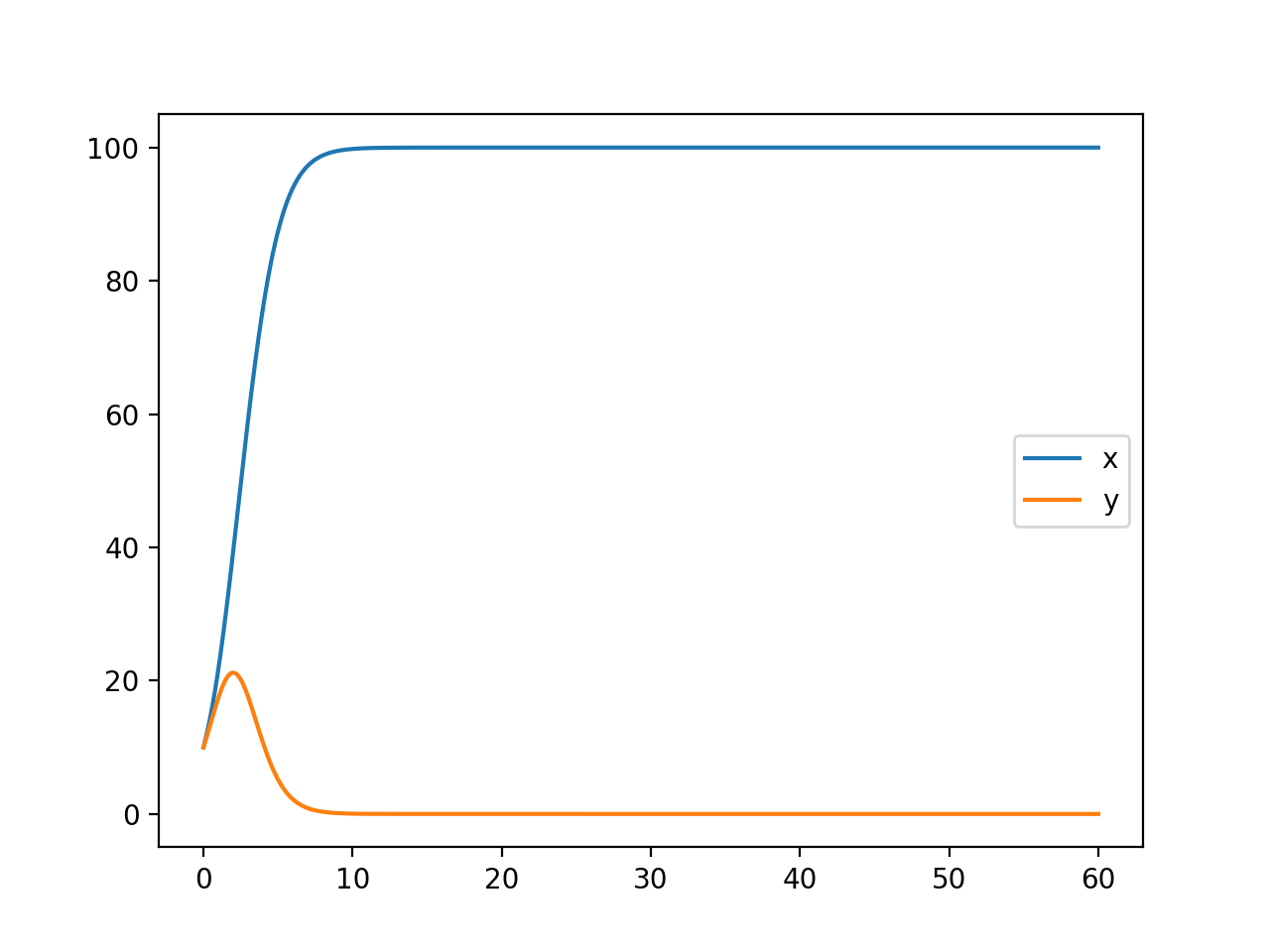


经计算在t=71.32左右到达最高点，高度为13118.81546776139m，加速度为-9.8m/s2。

这一部分代码如下：

f32 = fun3\_2()  
solver2 = rk45(f32,60.,[ys[-1]],60.,85,0.01)  
ans\_y = ans\_x  
ans\_x = xs.tolist()  
ans\_y\_1 = ys.tolist()  
ans\_y\_2 = f3(xs,ys).tolist()  
xs=solver2.get\_x()  
ys=solver2.get\_y()[0]  
  
**for** i **in** range(len(xs)):  
 **if**(ys[i]<=0):  
 **break** ans += ys[i]  
 ans\_y.append(ans / 100.)  
 ans\_y\_1.append(ys[i])  
 ans\_x.append(xs[i])  
 ans\_y\_2.append(f32(xs[i],ys[i]))  
ans\_x = np.asarray(ans\_x)  
ans\_y = np.asarray(ans\_y)  
ans\_y\_1 = np.asarray(ans\_y\_1)  
  
print(ans\_x[-1])  
print(f32(ans\_x[-1],ans\_y\_1[-1]))  
print(ans\_y\_1[-1])  
print(ans\_y[-1])  
  
plt.plot(ans\_x, ans\_y\_2, **'-'**)  
plt.legend(**'acceleration'**, loc=**'best'**)  
plt.show()  
plt.plot(ans\_x, ans\_y\_1, **'-'**)  
plt.legend(**'velocity'**, loc=**'best'**)  
plt.show()  
plt.plot(ans\_x, ans\_y, **'-'**)  
plt.legend(**'height'**, loc=**'best'**)  
plt.show()

全过程的加速度、速度、高度曲线如下：

9. (1)图形如下，当t充分大后，x(t)趋向于100，y(t)趋向于0。

代码如下：

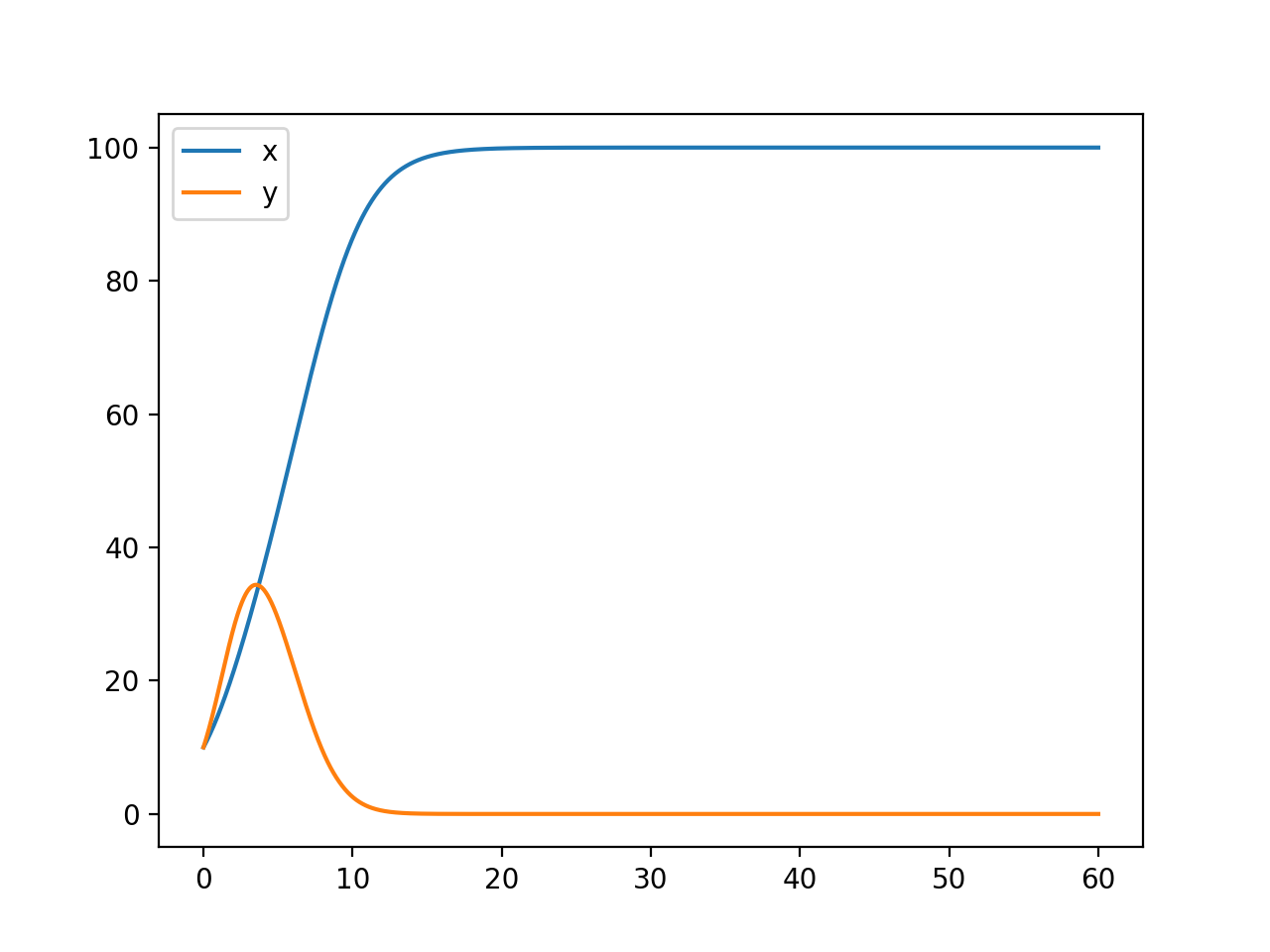
**class** fun9(function2d):  
 **def** \_\_init\_\_(self,r1,r2,n1,n2,s1,s2):  
 self.\_r1=r1  
 self.\_r2=r2  
 self.\_n1=n1  
 self.\_n2=n2  
 self.\_s1=s1  
 self.\_s2=s2  
 **def** get(self,x,y):  
 x=self.\_r1\*y[0]\*(1.-y[0]/self.\_n1-self.\_s1\*y[1]/self.\_n2)  
 y=self.\_r2\*y[1]\*(1.-self.\_s2\*y[0]/self.\_n1-y[1]/self.\_n2)  
 **return** np.asarray([x,y])

**def** p\_9():  
 f9=fun9(r1=1.,r2=1.,n1=100,n2=100,s1=0.5,s2=2)  
 solver = rk45(f9, 0., [10,10], 0., 60., 0.01)  
 solver.plot(2,[**'x'**,**'y'**])

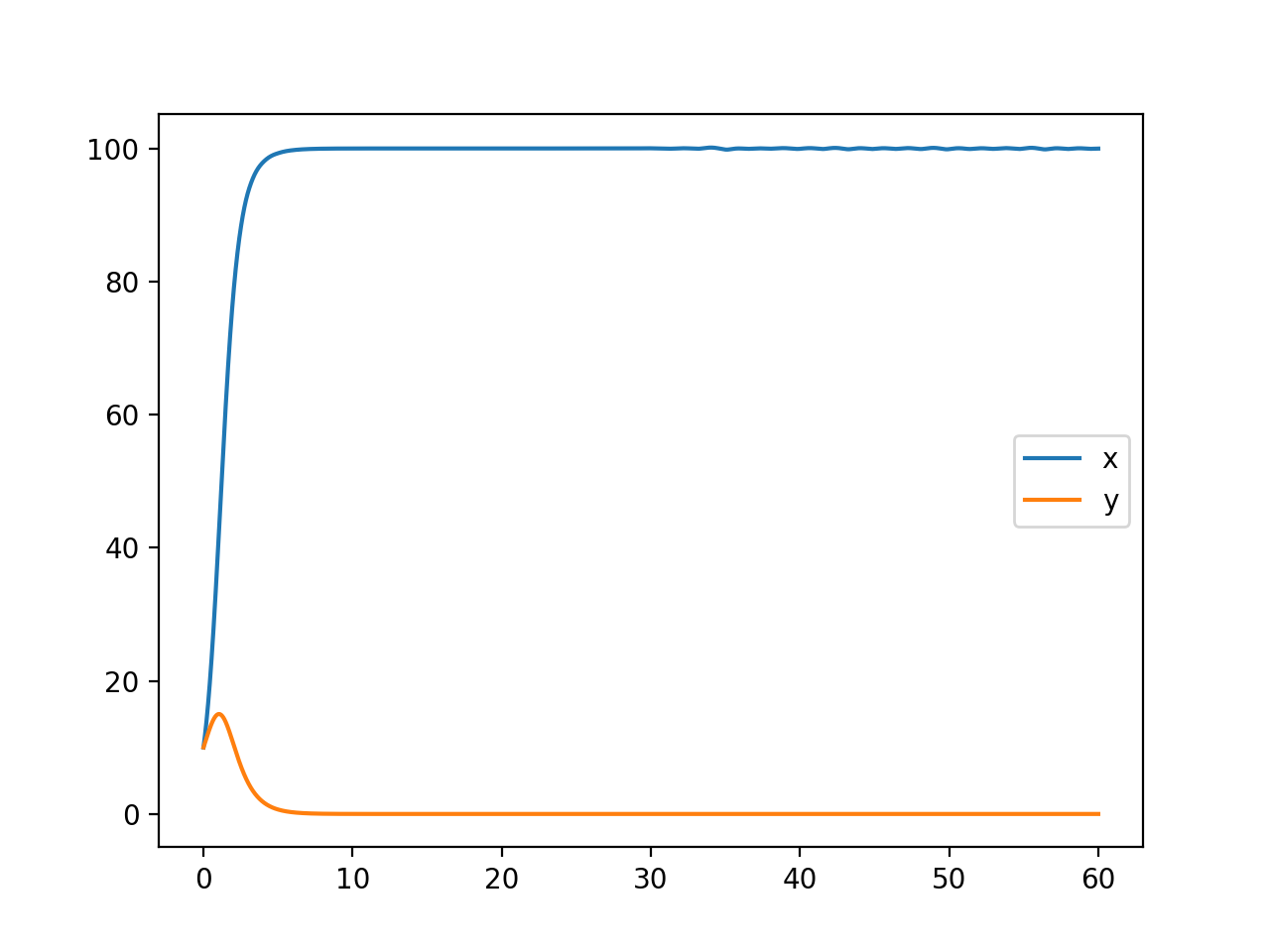
这个结果说明了甲消耗乙的资源导致乙灭绝。

（2）下面的结果每次只改变一个变量。

降低r1到0.5，结果如下：

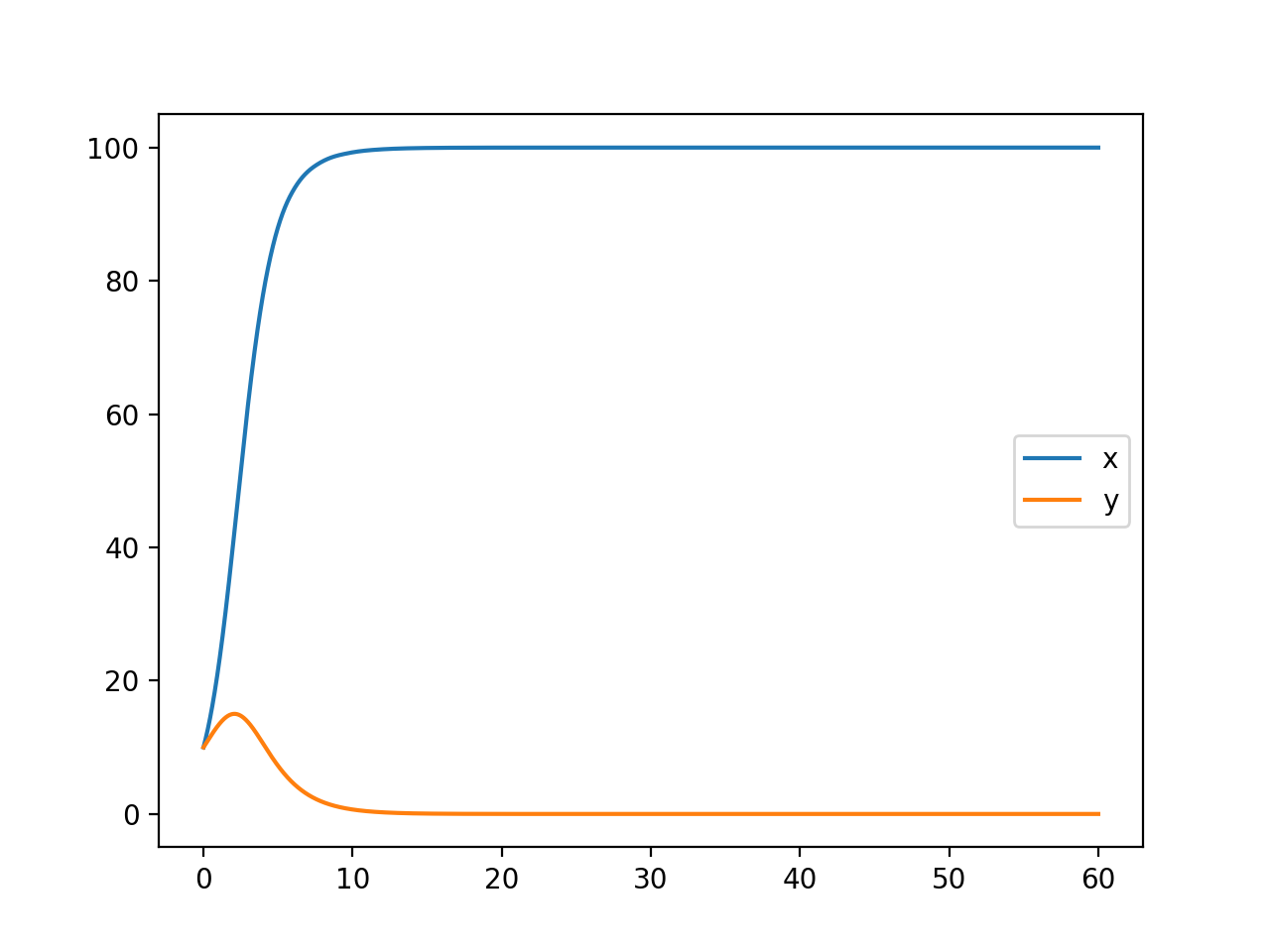


增加r1到2.0，结果如下：

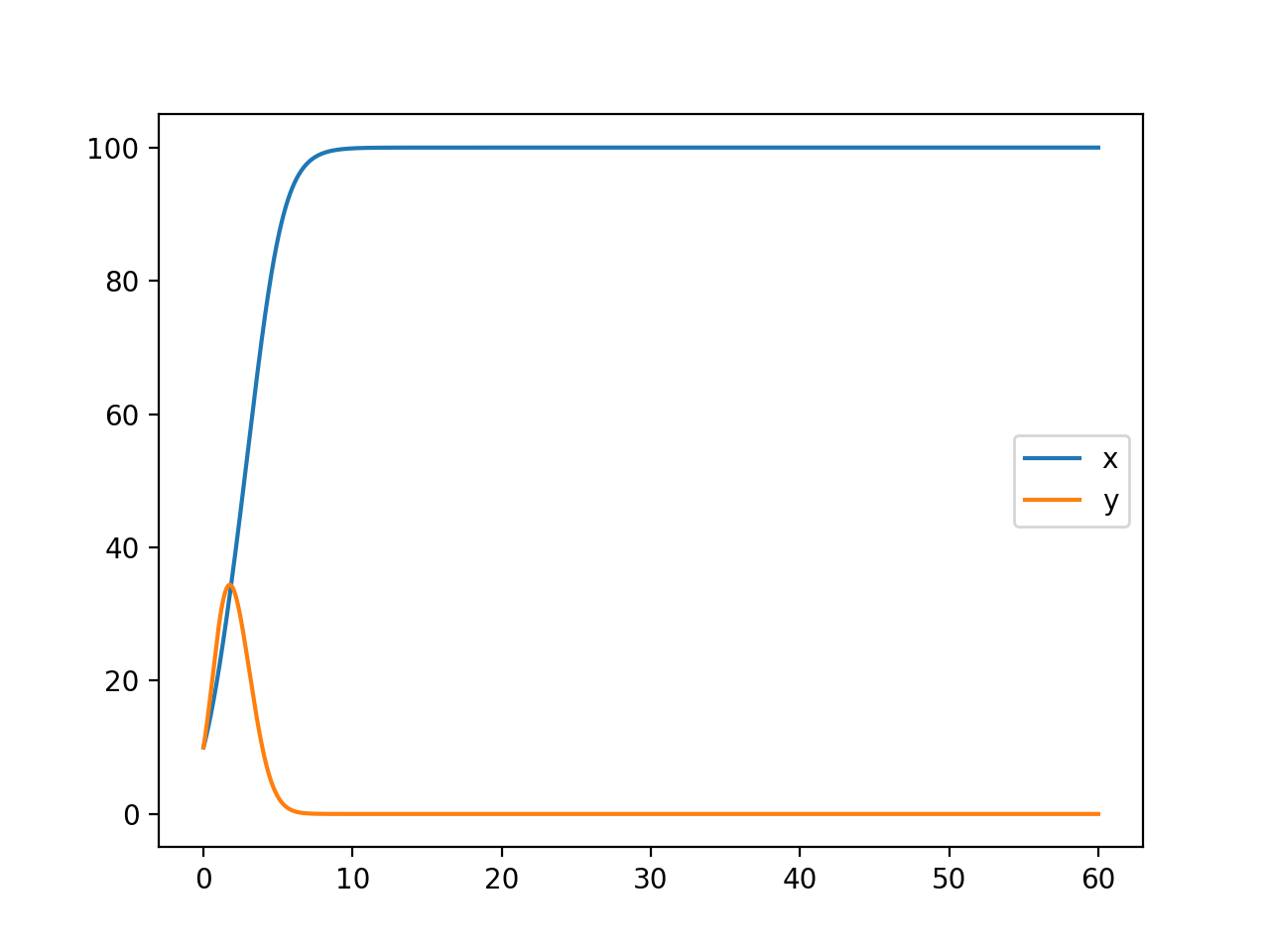


可以看出，甲增长率越高，甲到最大容量越快，乙灭绝的越快。

降低r2到0.5，结果如下：

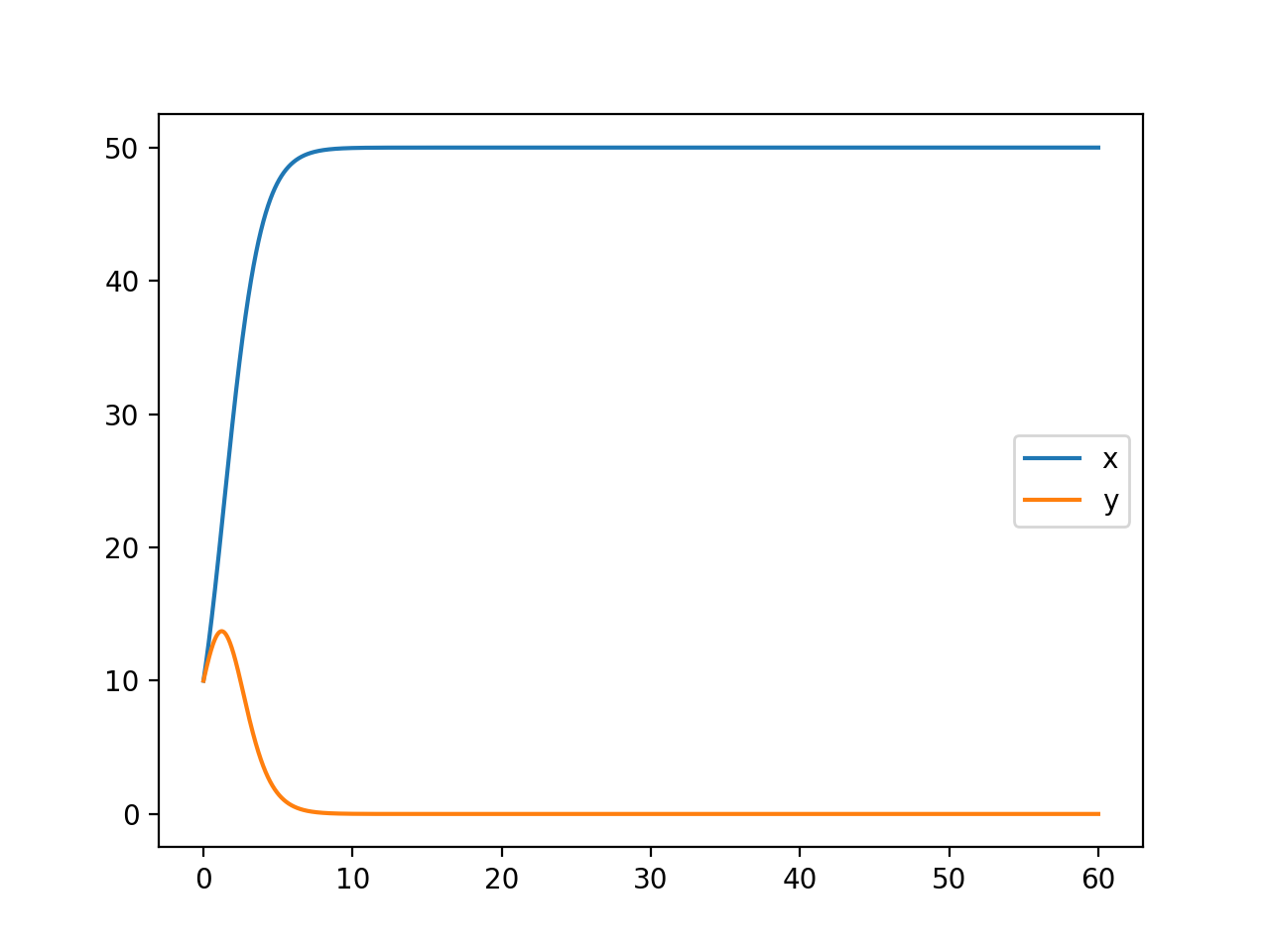


增加r2到2.0，结果如下：

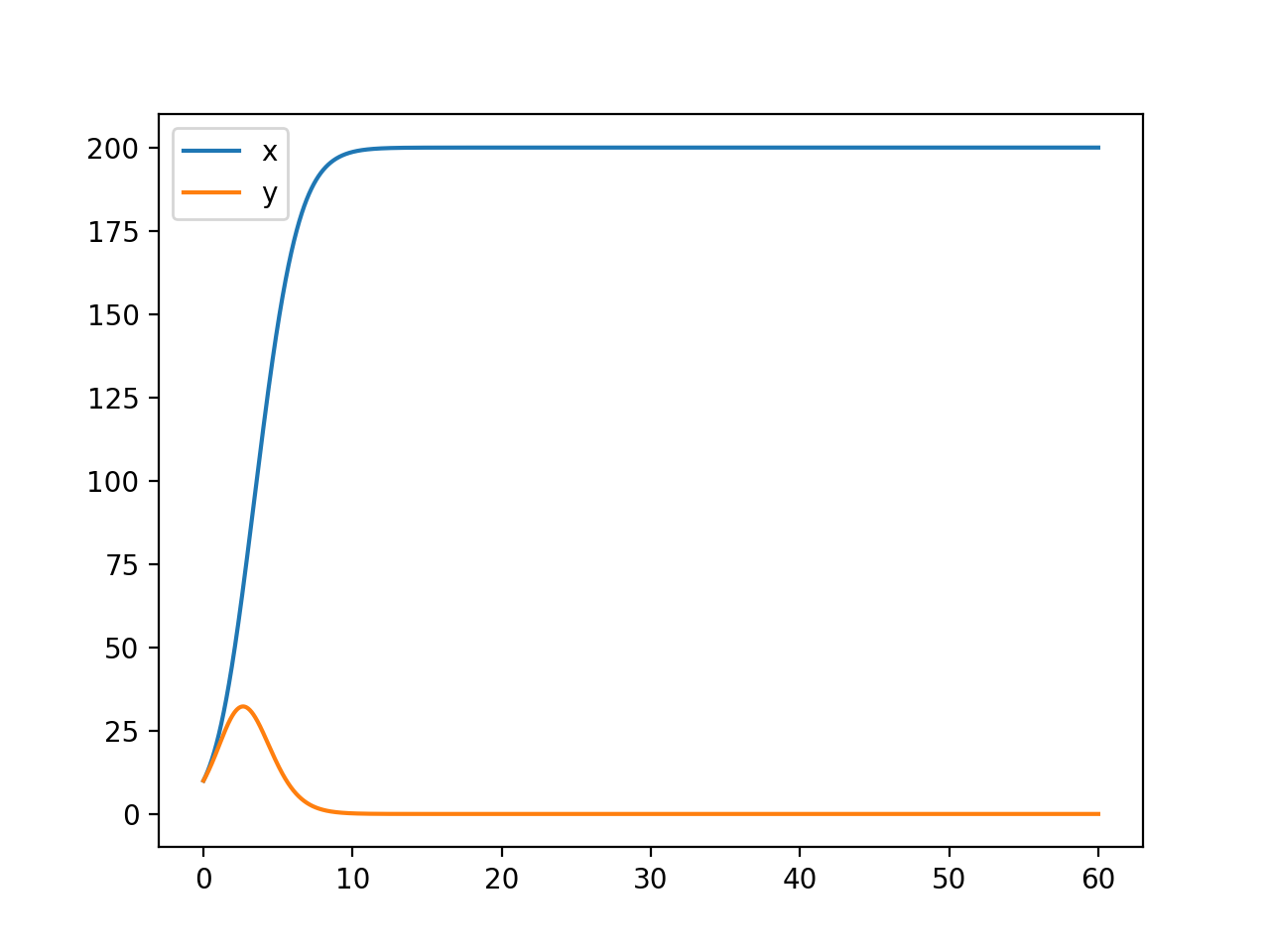


乙的增长率越高，灭绝之前到达的最大数量越高，但灭绝时间也越早，可能是消耗了过多的资源所致。

降低n1到50，结果如下：

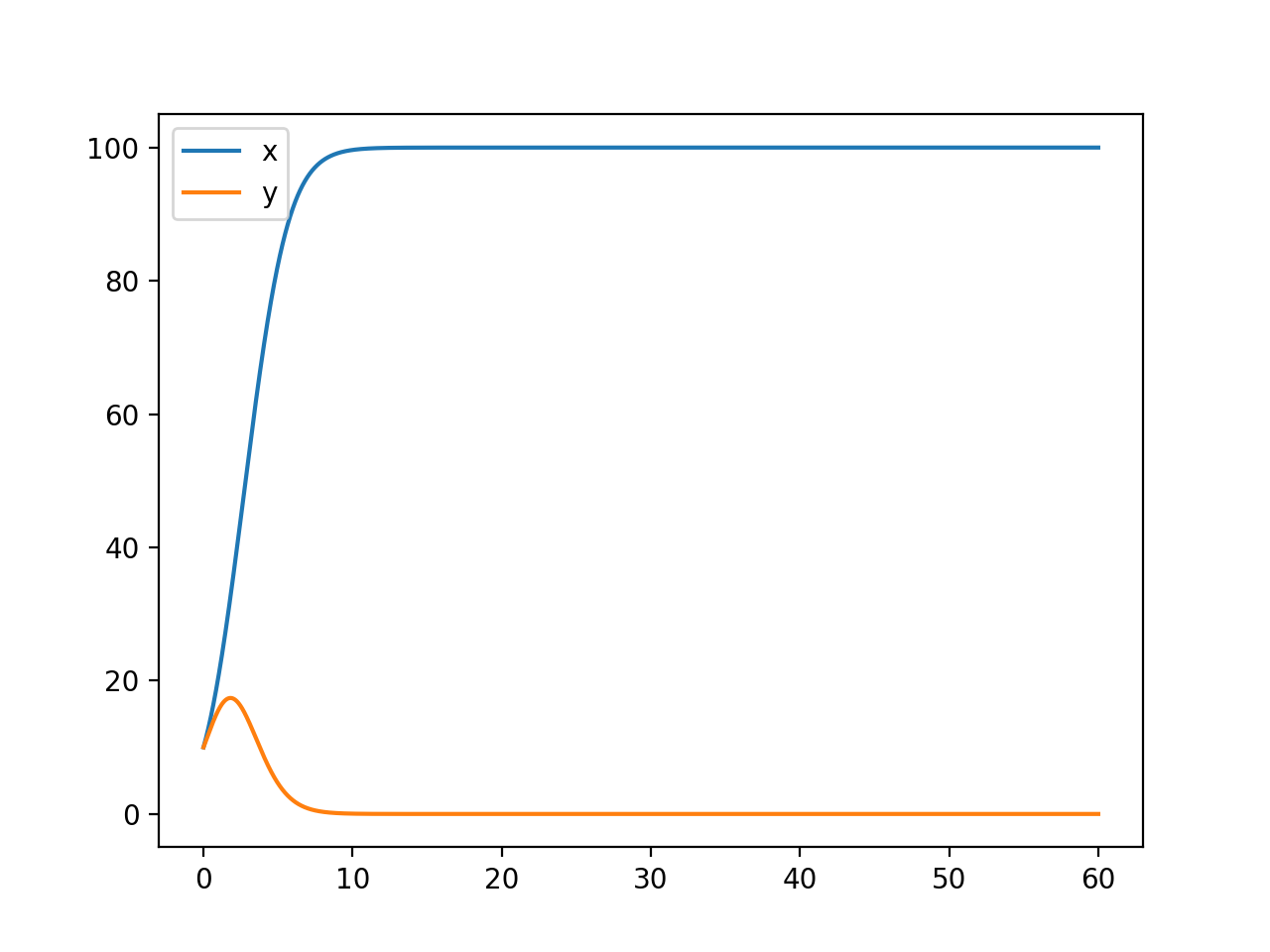


增加n1到200，结果如下：

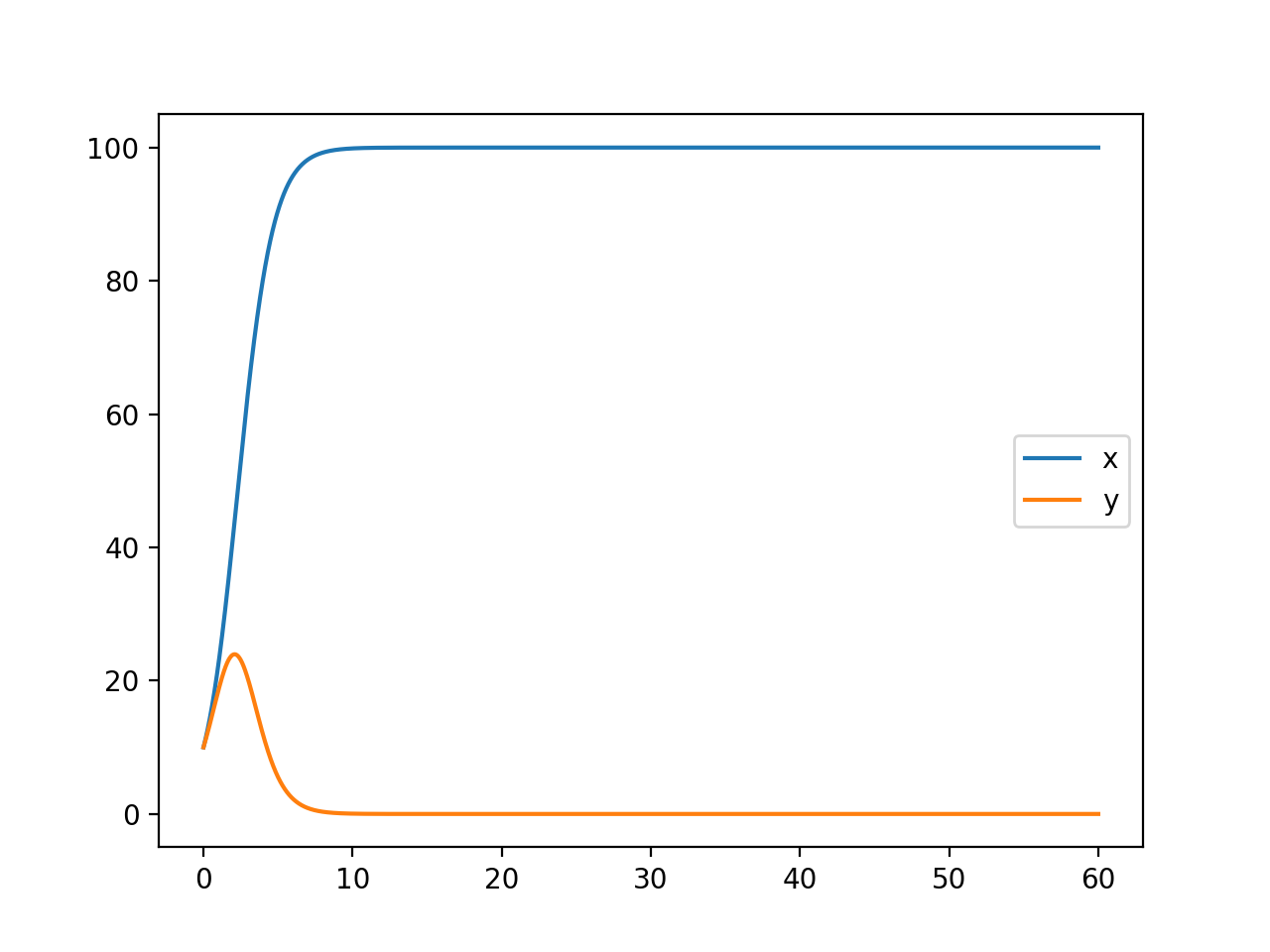


甲的最大容量越高，乙灭绝前达到的最大数量越高，原因是甲所消耗的乙资源和单位数量的甲有关。提高容量降低了单位数量的甲。

降低n2到50，结果如下：

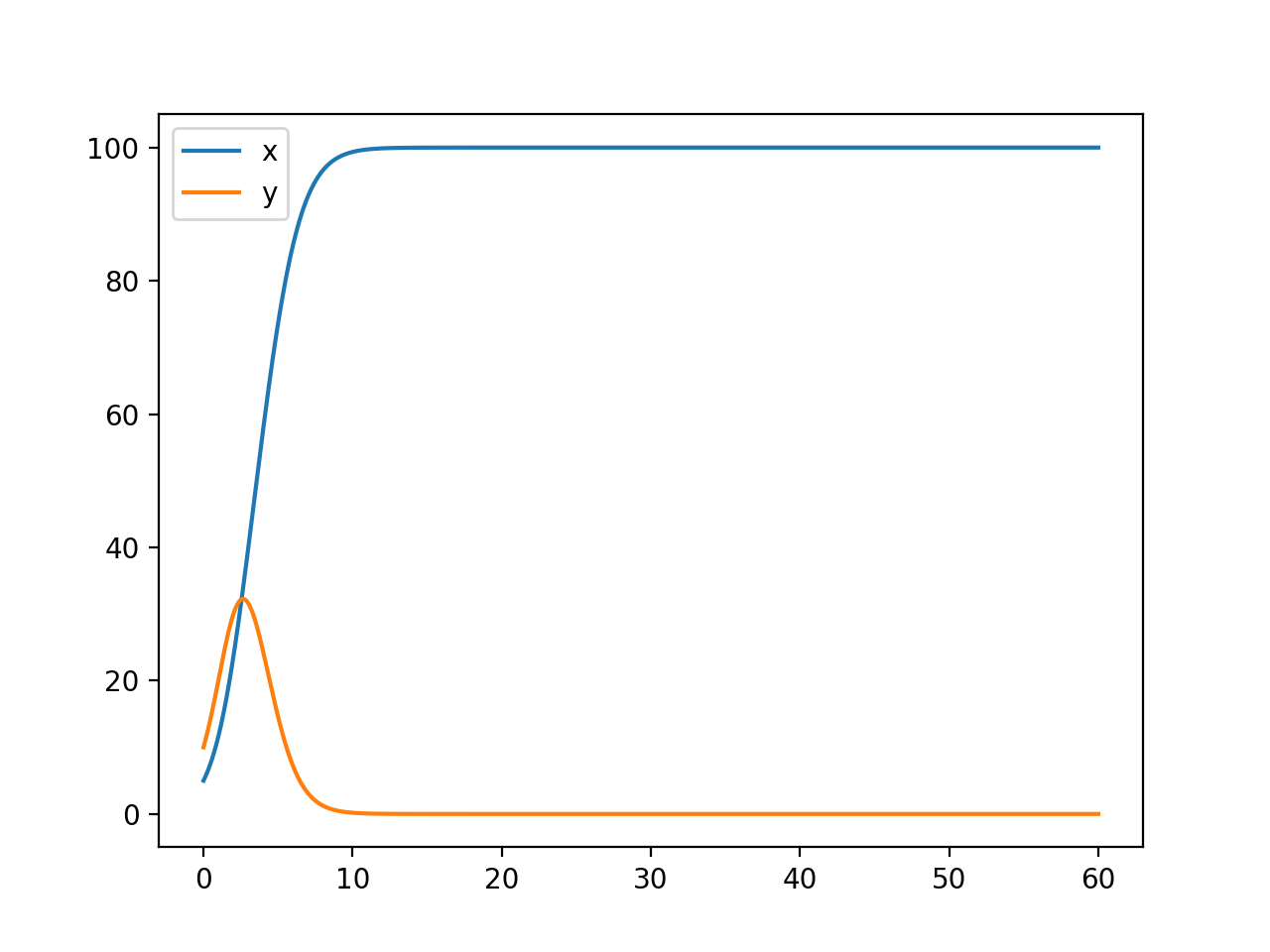


增加n2到200，结果如下：

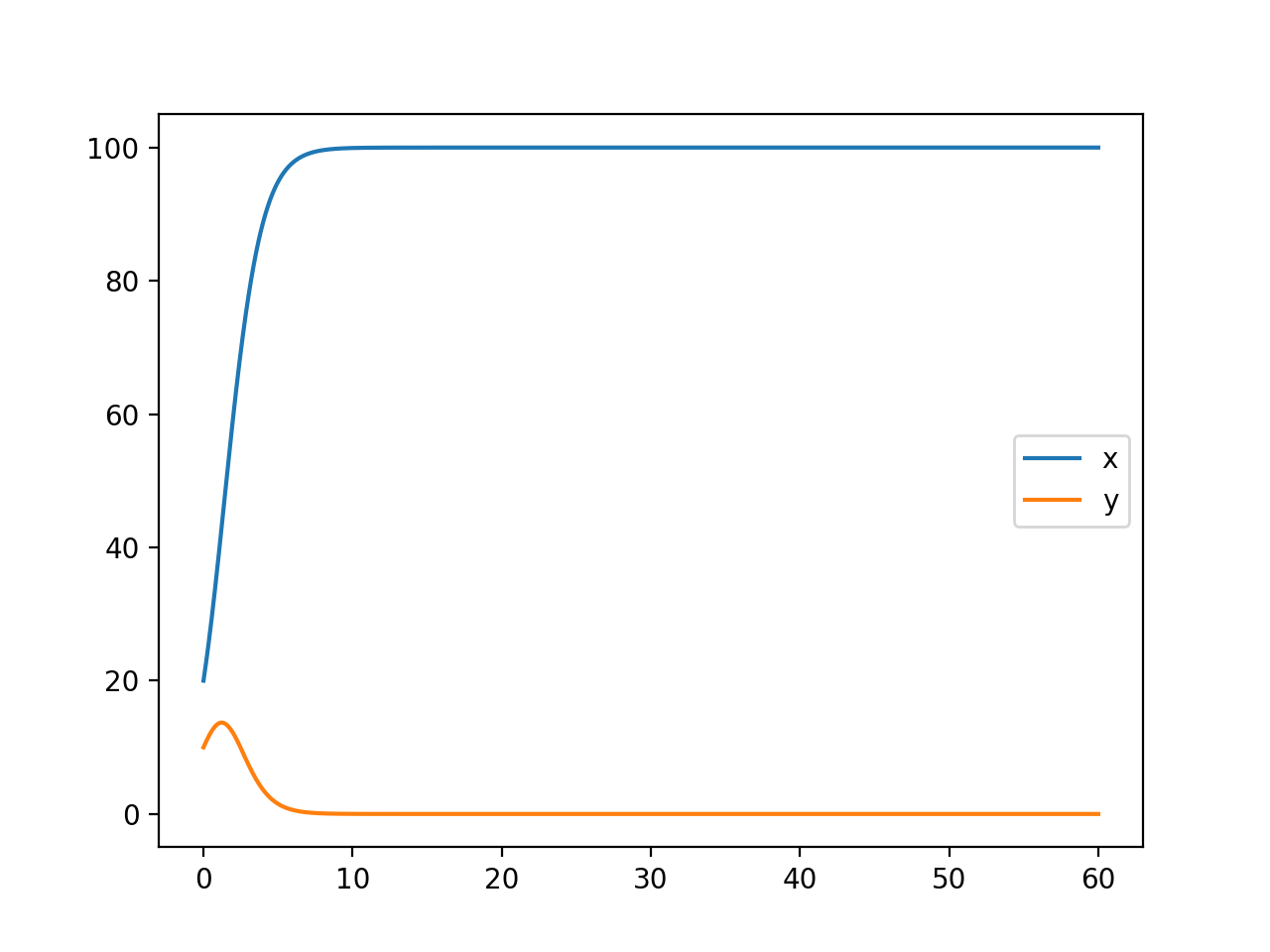


提高乙的容量降低了单位数量的乙，降低了所消耗的资源比例，故乙的容量越高，灭绝前达到的最大数量越高。

降低x0到5，结果如下：

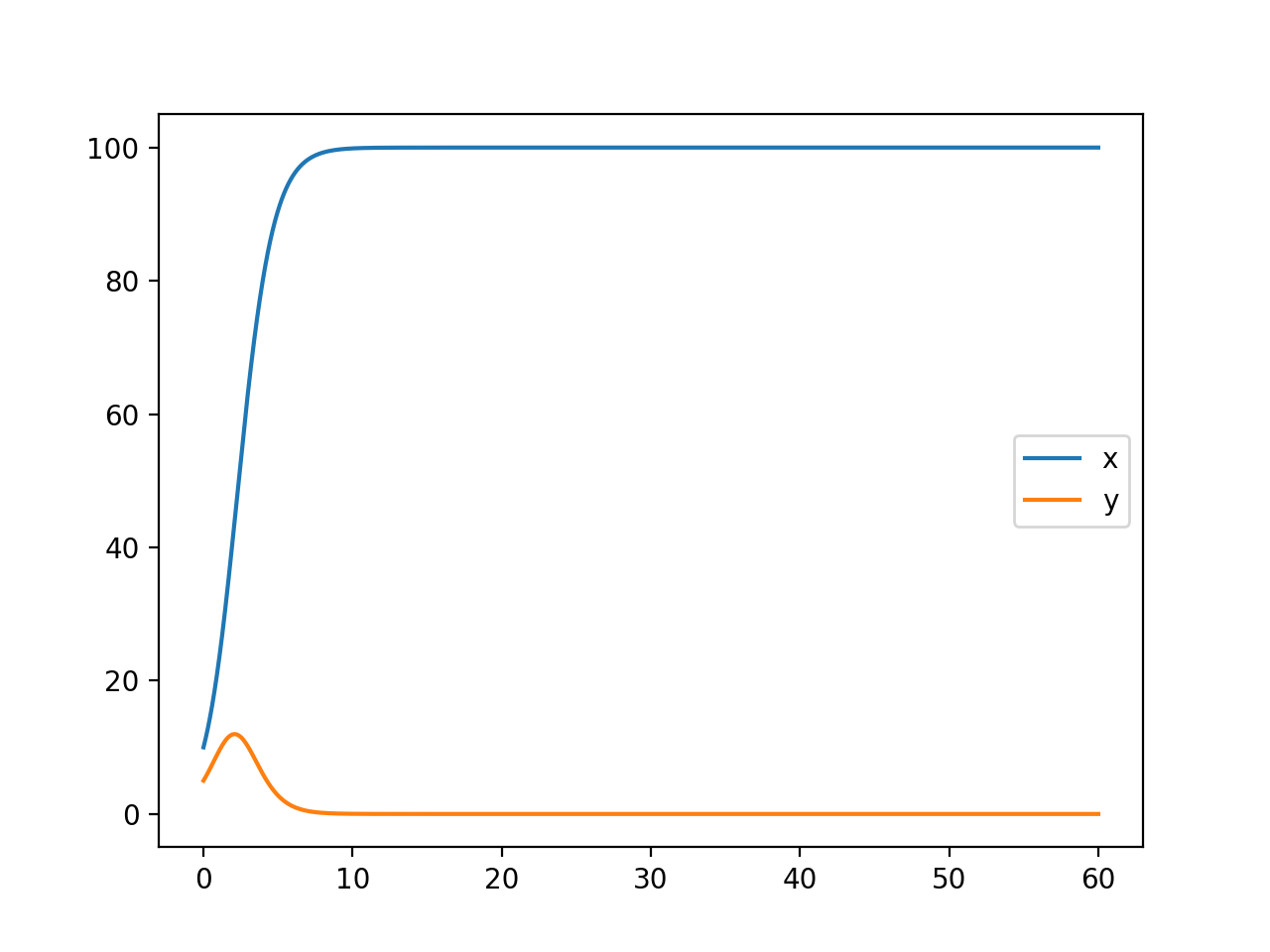


增加x0到20，结果如下：

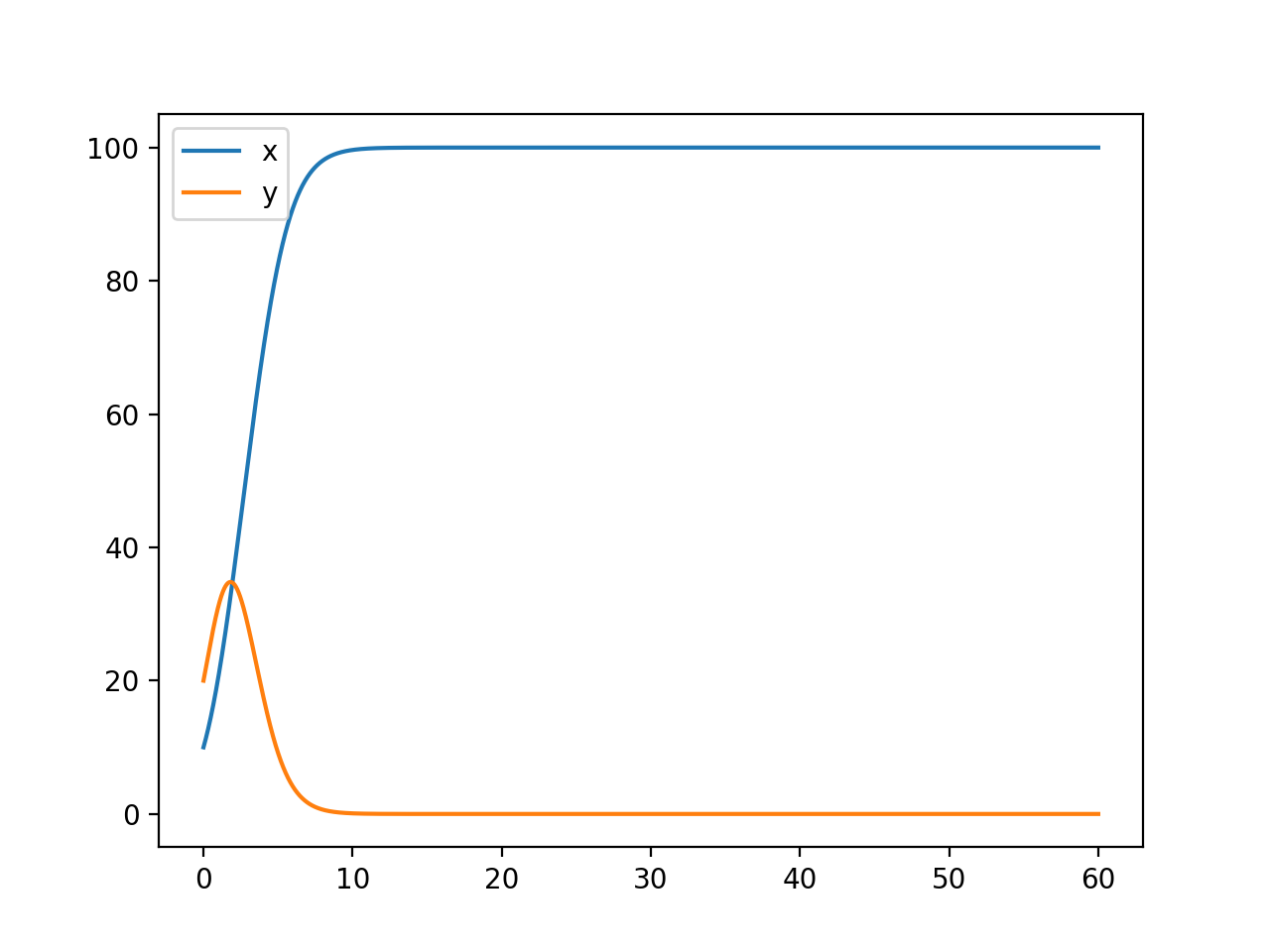


甲的初始数量越高，乙灭绝的越快。

降低y0到5，结果如下：

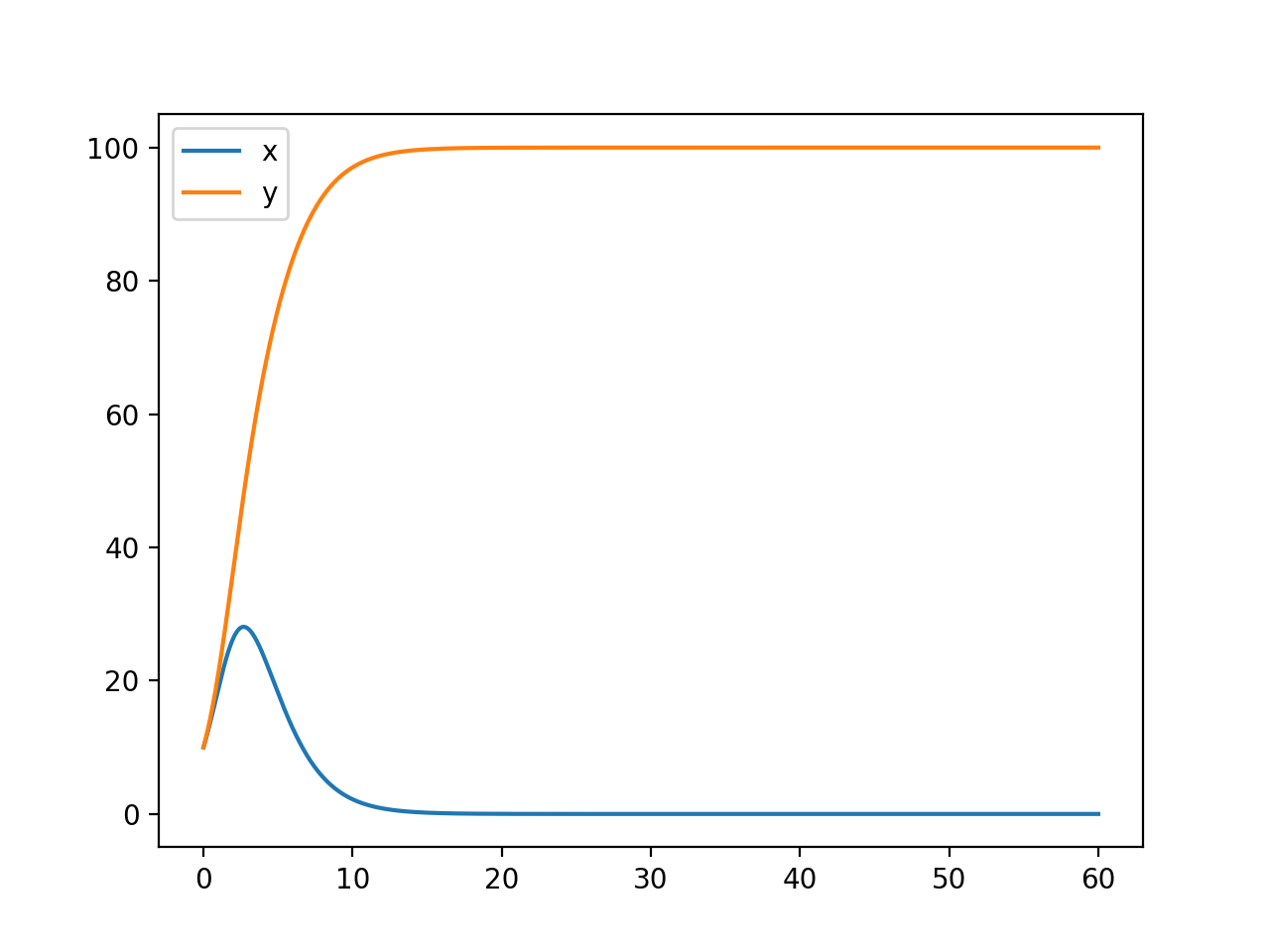


增加y0到20，结果如下：



乙的初始数量越高，灭绝得越慢。

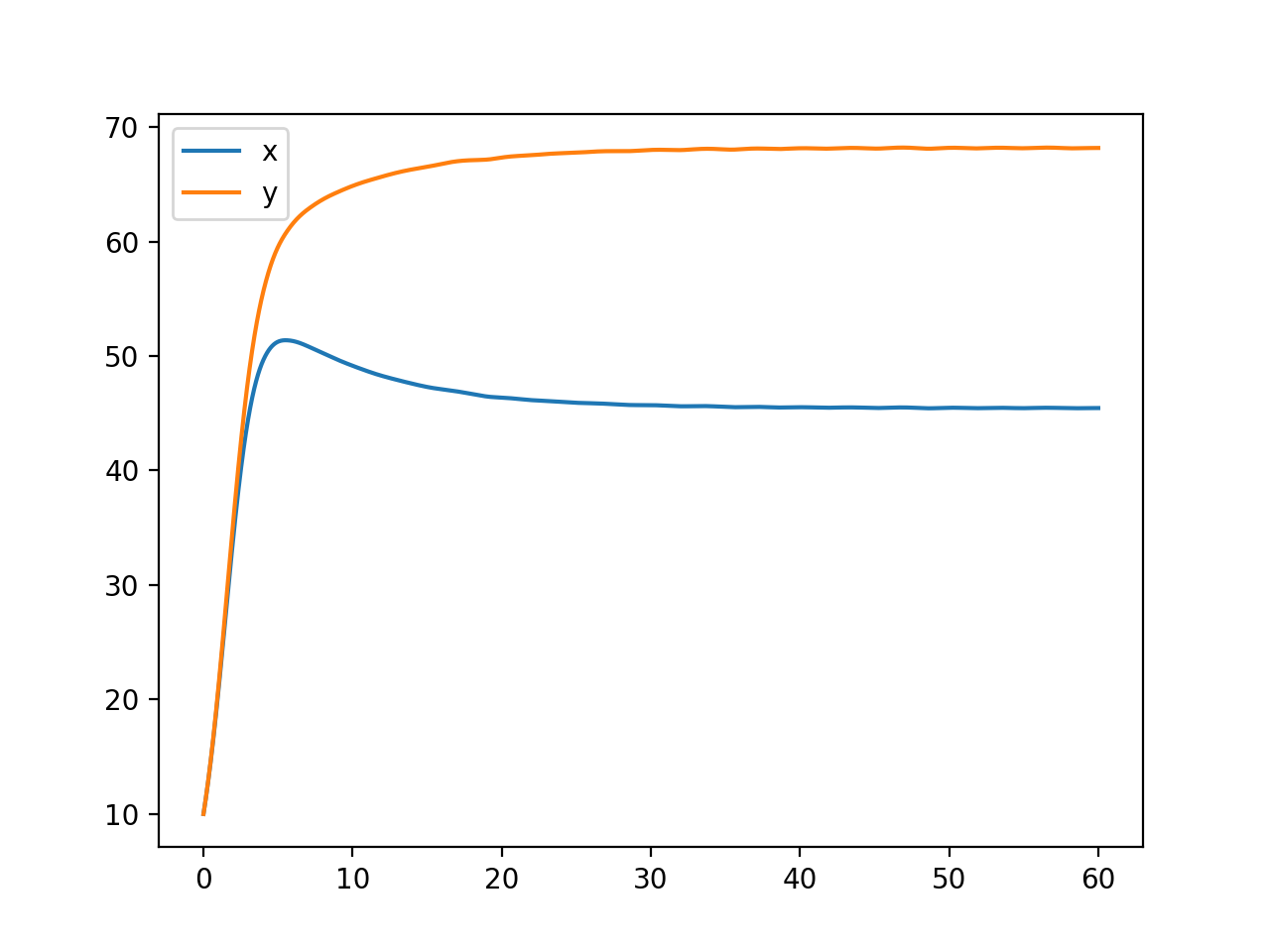
当s1=1.5,s2=0.7时，结果如下：



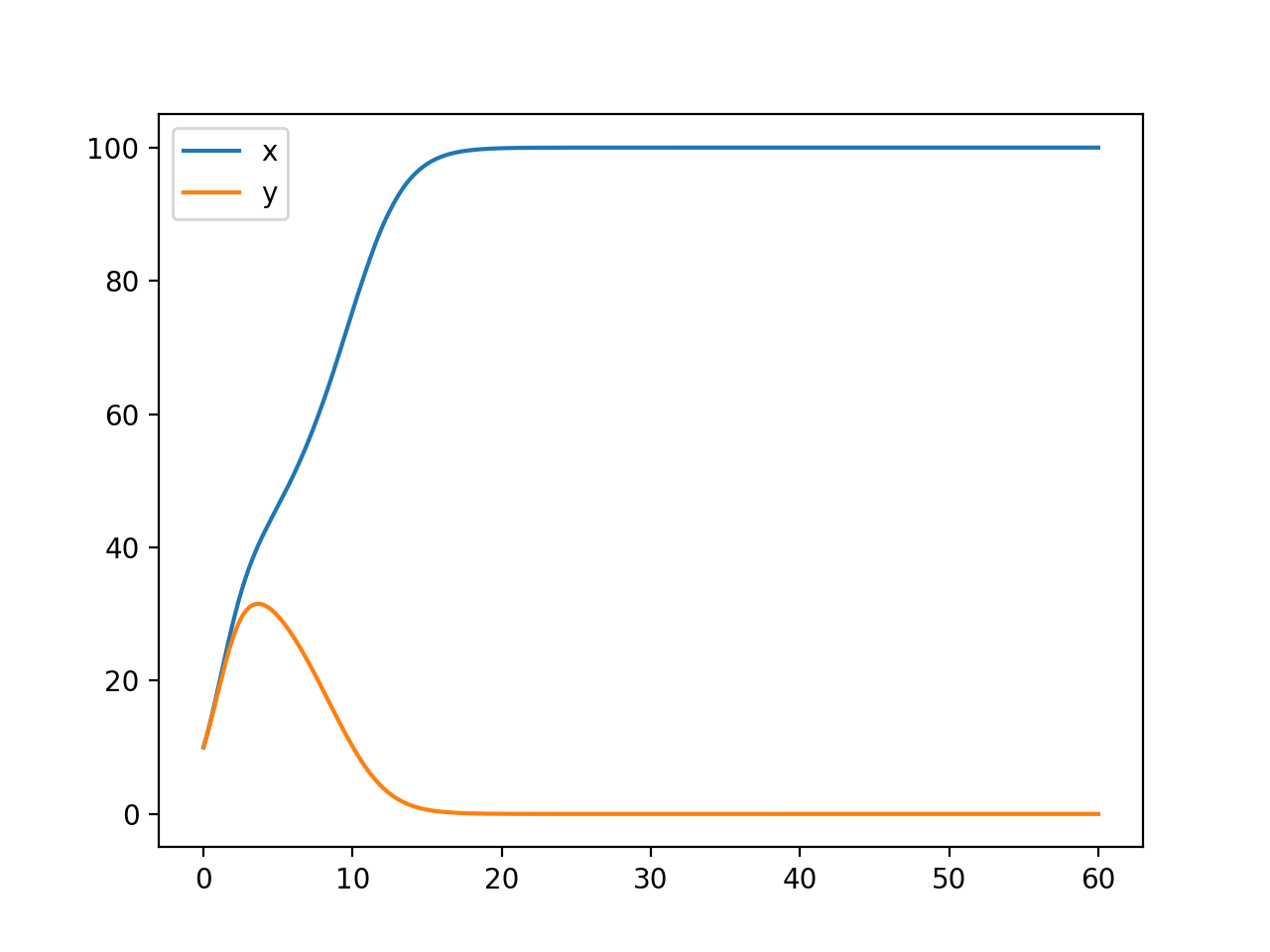
可以发现，无论如何调整数值，只要是一方的s＞1，另一方的s＜1，s＞1的一方在长期会使另一方灭绝。生物学上的解释是s>1的一方有绝对的竞争优势，能大量抢占另一方的资源，消耗的资源却不容易被另一方抢占。

这一部分的代码是直接在(1)中的代码修改参数，故略去。

（3）当s1=0.8，s2=0.7时，长期曲线如下：



当s1=1.5,s2=1.7时，长期曲线如下：



当s1和s2都＜1时，双方都不能使对方灭绝，在长期处于相对平衡状态，乙消耗甲的资源比甲消耗乙的资源多，故乙的数量更多。当s1和s2都＞1时，甲消耗乙的资源比乙消耗甲的资源多，导致了乙的灭绝。