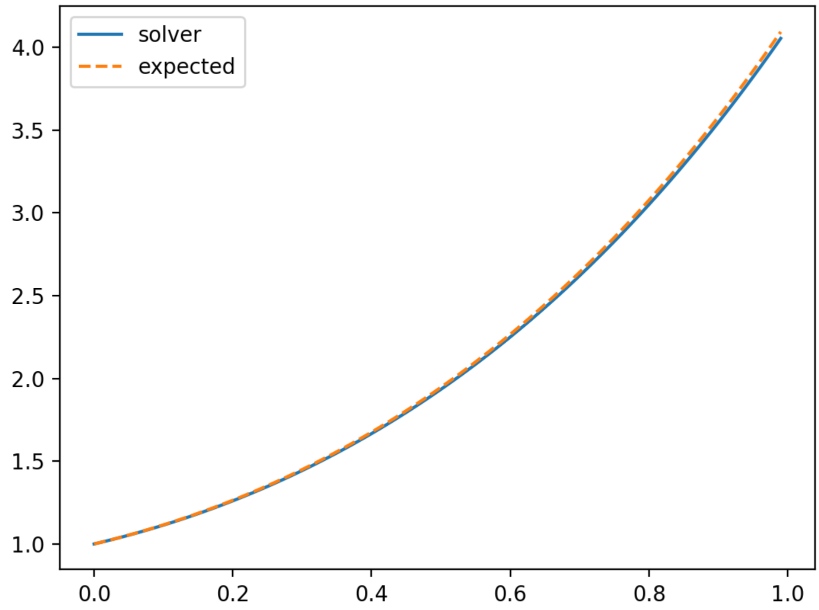
常微分方程初值问题的数值解 实验报告

计 65 赖金霖 2016011377

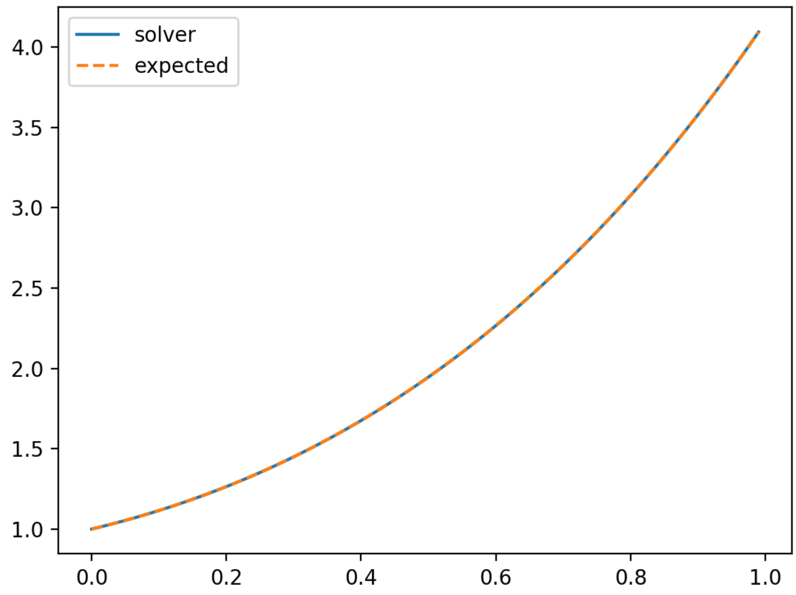
实验代码在 https://github.com/lll6924/math_exp/tree/master/exp2 下可以找到, 使用的函数在 https://github.com/lll6924/math_exp/blob/master/utils/odesolver.py 内。

2.（1）步长为 0.01

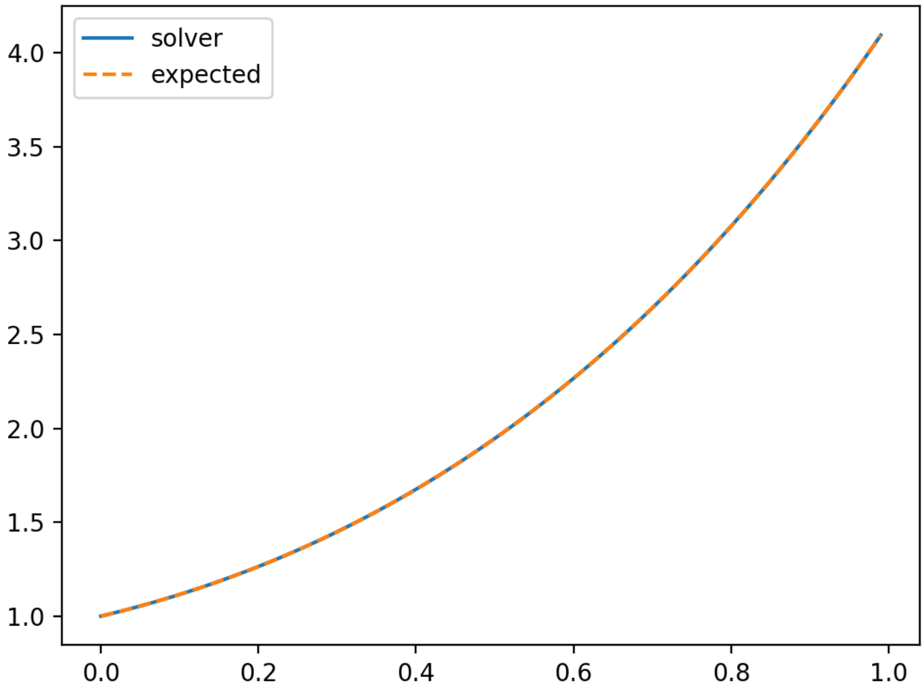
使用欧拉法的效果如下, 在 x=1 处误差为 0.10074500194685854, 运行 1000 次耗时 1.134s。



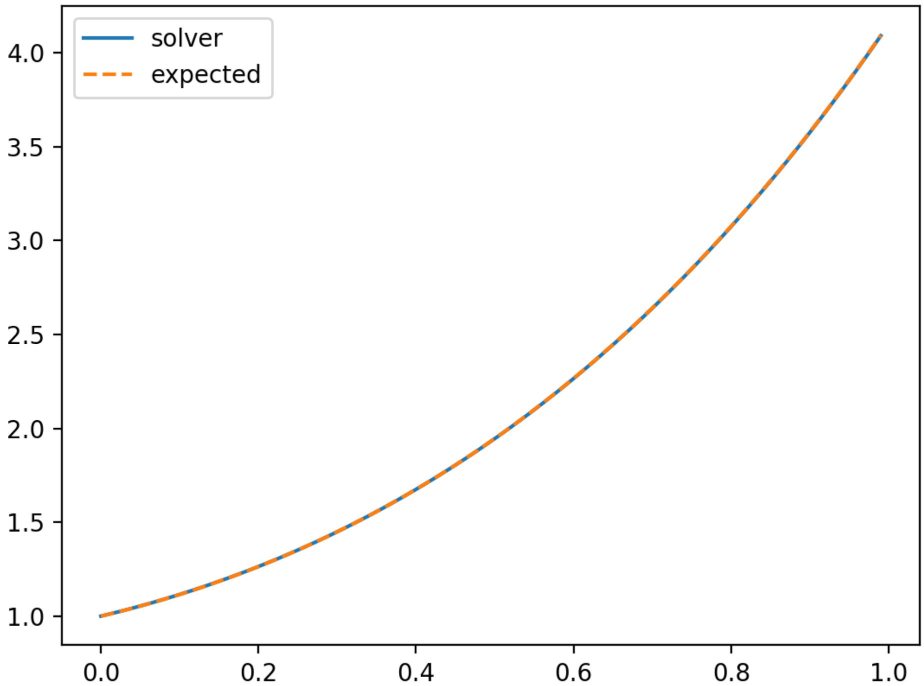
使用改进欧拉法如下，x=1 处误差为 0.0612742882286037，运行 1000 次耗时 1.639s。



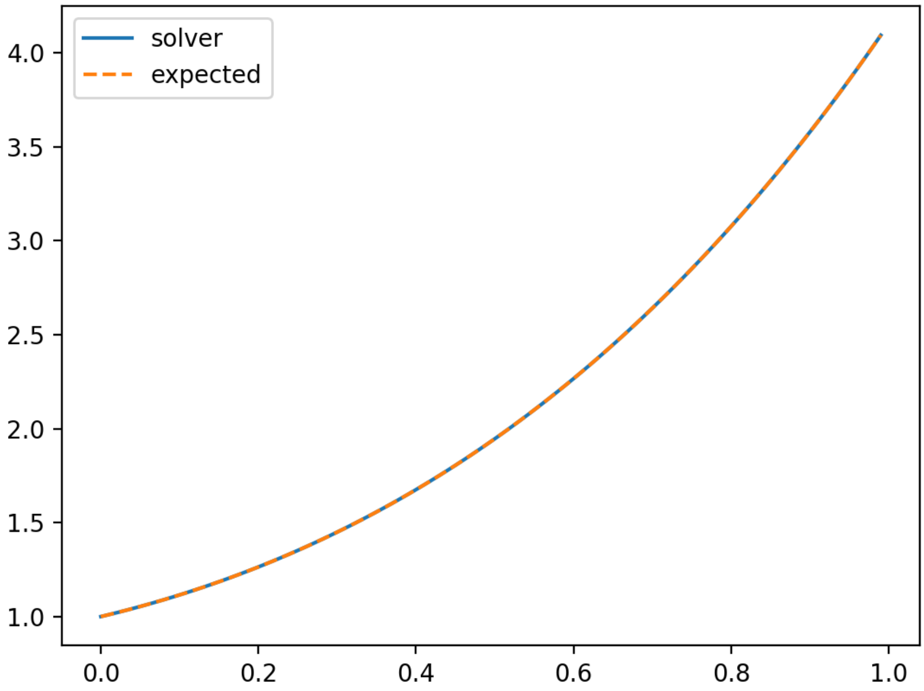
使用经典荣格-库塔法如下, x=1 处误差为 0.06114206898990027, 运行 1000 次耗时 2.806s。



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。

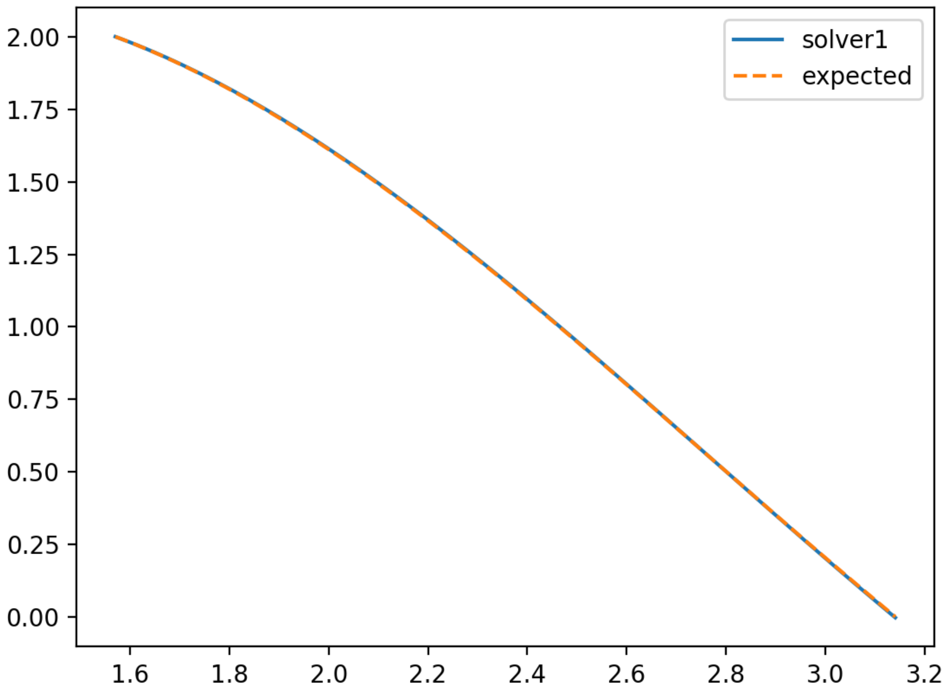
（3）
为了解二阶常微分方程，我采用了类似迭代的方法，经过变形，有

$$y'' = \frac{(0.25-x^2)*y-x*y'}{x^2}$$

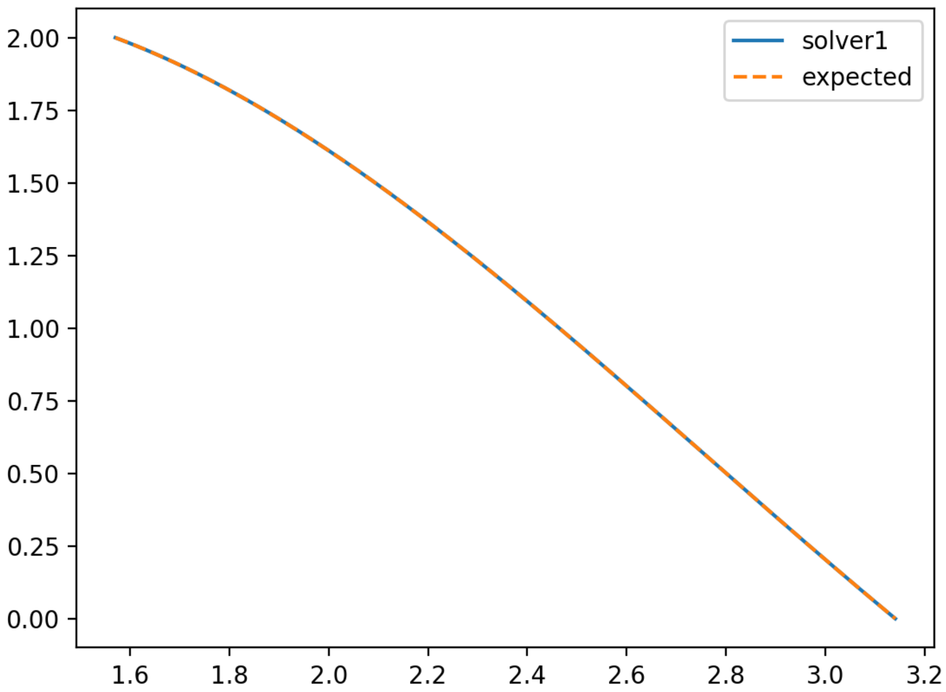
$$y' = \frac{(0.25-x^2)*y-x^2*y''}{x}$$

我们把 y 和 y' 作为两个同时求取的函数，导数如上定义，其中第二个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。

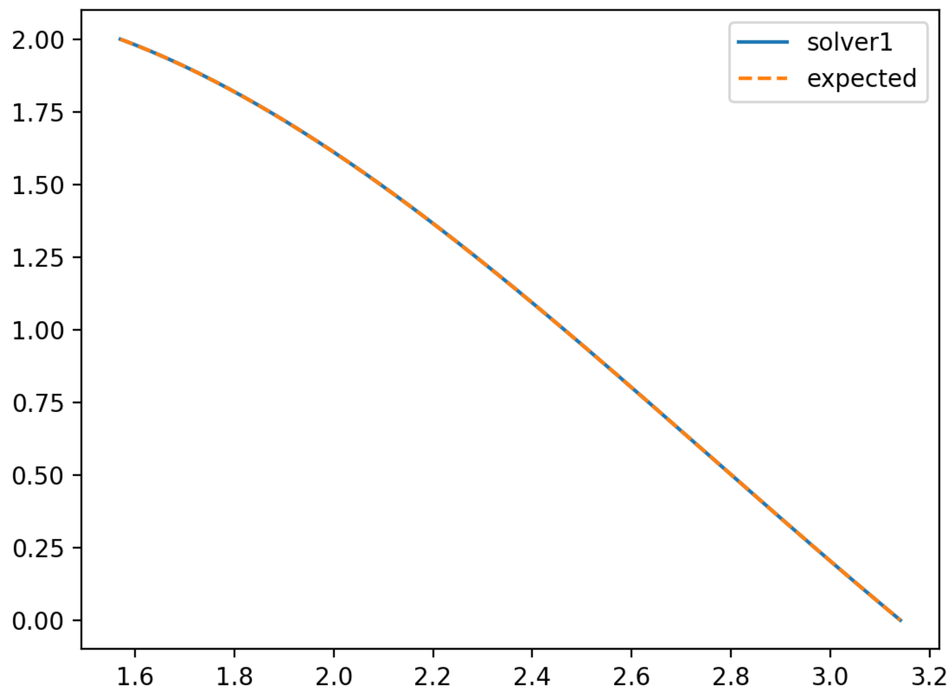
使用欧拉法的效果如下，在 x=π处误差为 0.001928599127587374，1000 次实验耗时 2.064s。



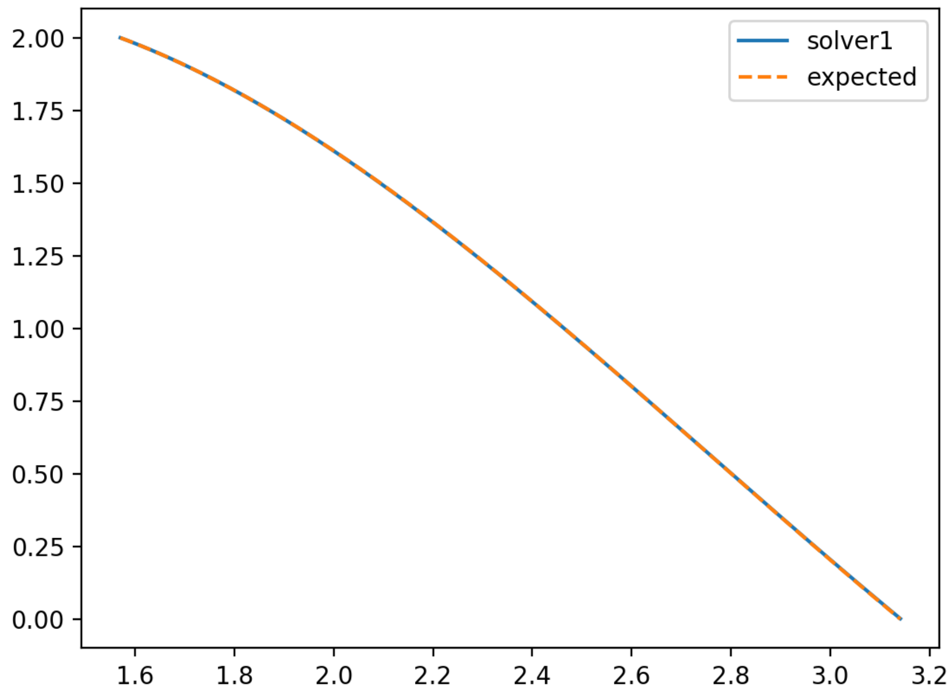
使用改进欧拉法如下，在 x=π处误差为 0.0011034178094134312，1000 次实验耗时 4.297s。



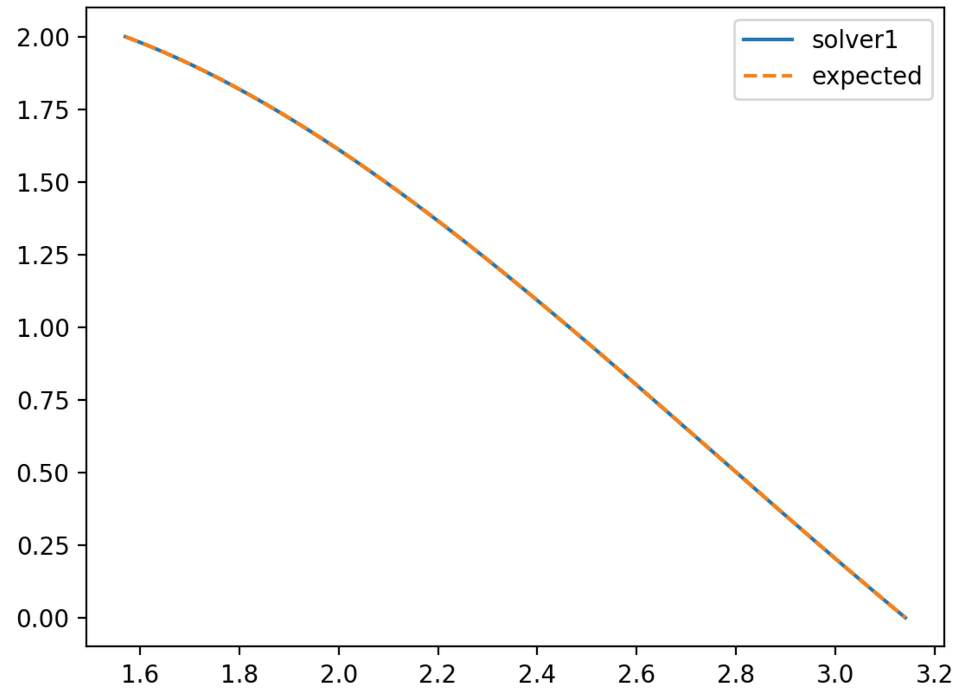
使用经典荣格-库塔方法如下，x=π处误差为 0.0011263188524708488，1000 次实验耗时 6.454s。



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。



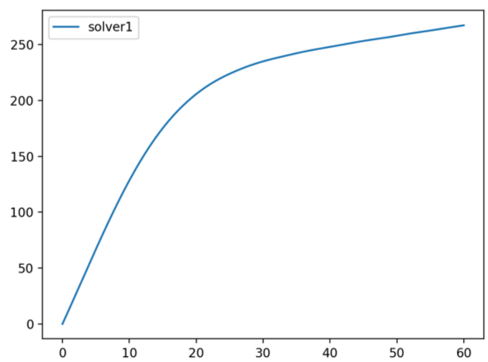
可以看出，虽然改进欧拉法的误差最小，但库函数依然十分优秀。

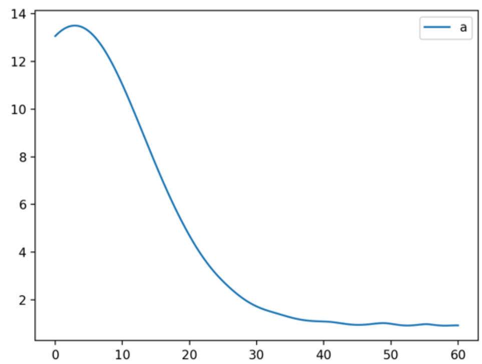
3.

可以列出如下的方程：

$$\frac{d^2x}{dx^2}=\frac{32000-0.4*(\frac{dx}{dt})^2}{1400-18*t}-9.8$$

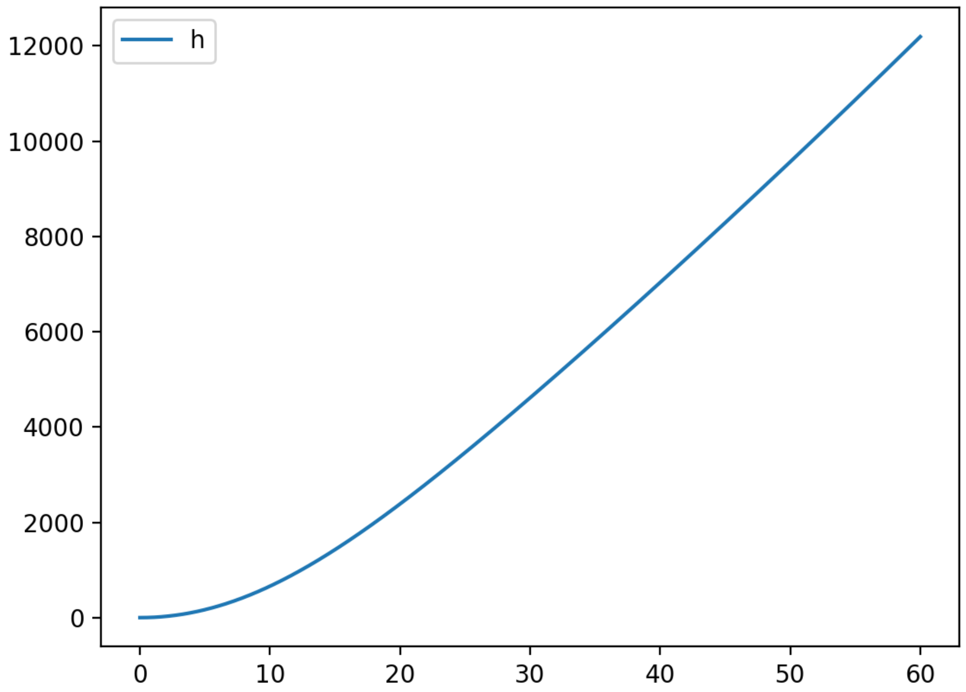
解此方程可得如下结果（左侧为速度曲线，右侧为加速度曲线）





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()
```

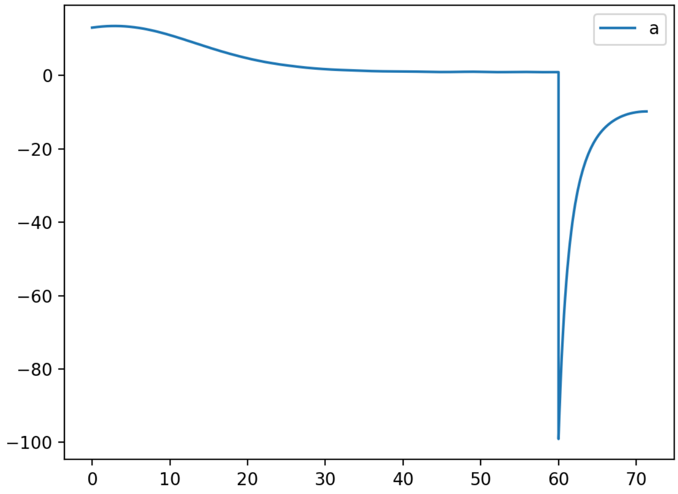
关闭引擎后，火箭继续飞行，模型方程变为：

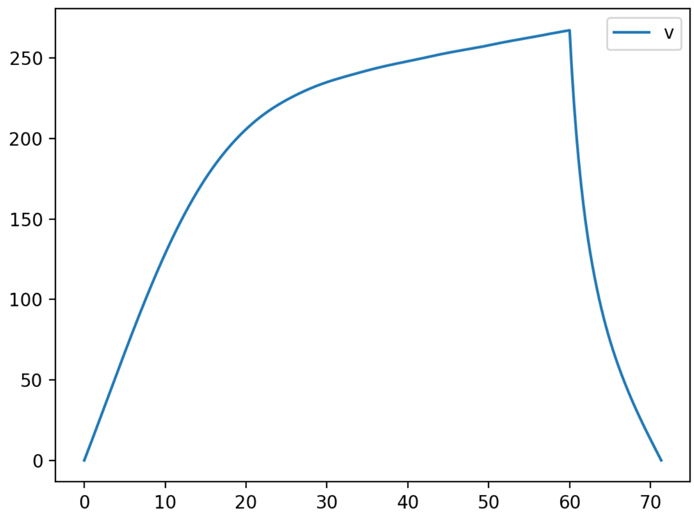
$$\frac{d^2x}{dx^2} = \frac{-0.4*(\frac{dx}{dt})^2}{320} - 9.8$$

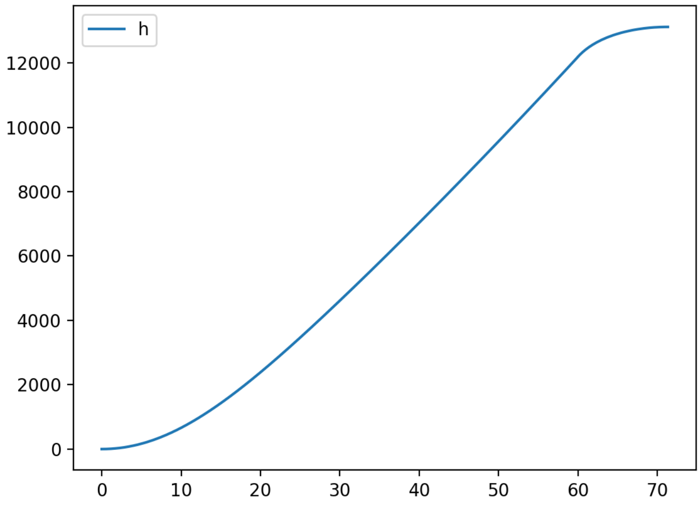
经计算在 t=71.32 左右到达最高点，高度为 13118.81546776139m，加速度为-9.8m/s2。这一部分代码如下：

```
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()
```

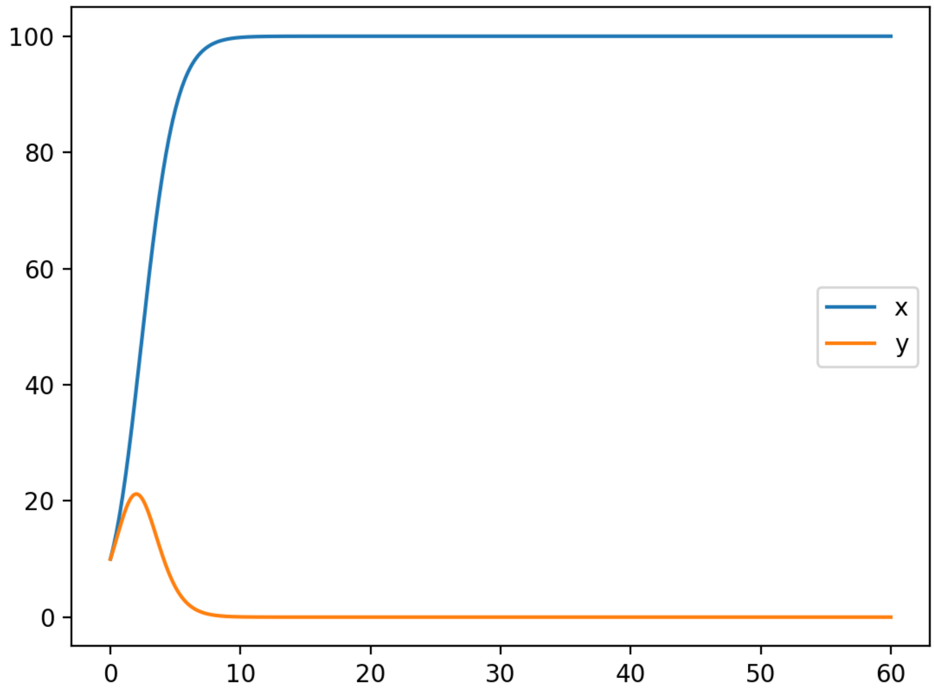
全过程的加速度、速度、高度曲线如下：







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

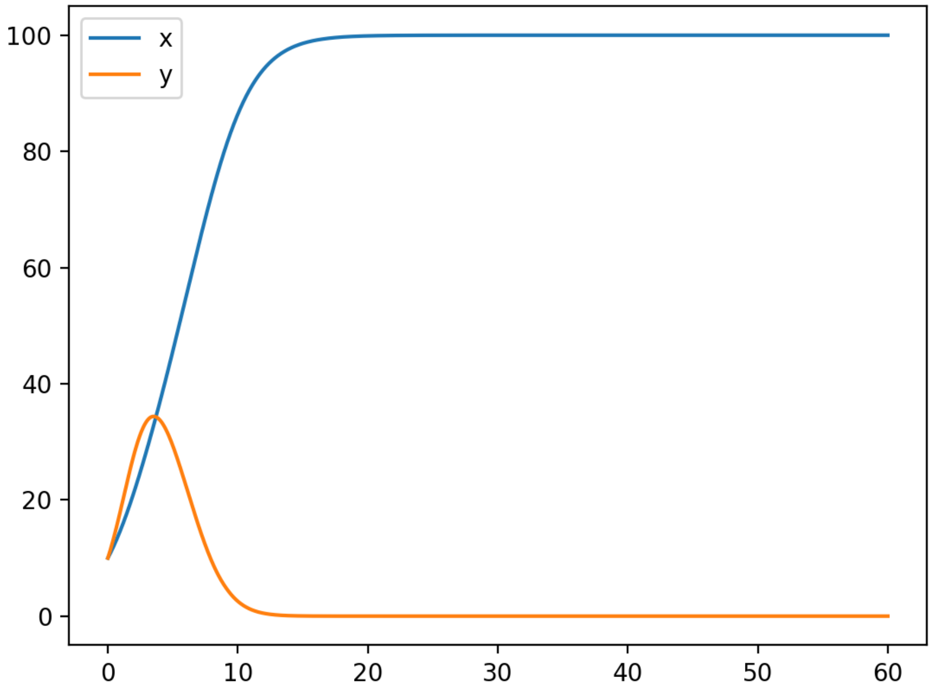


代码如下：

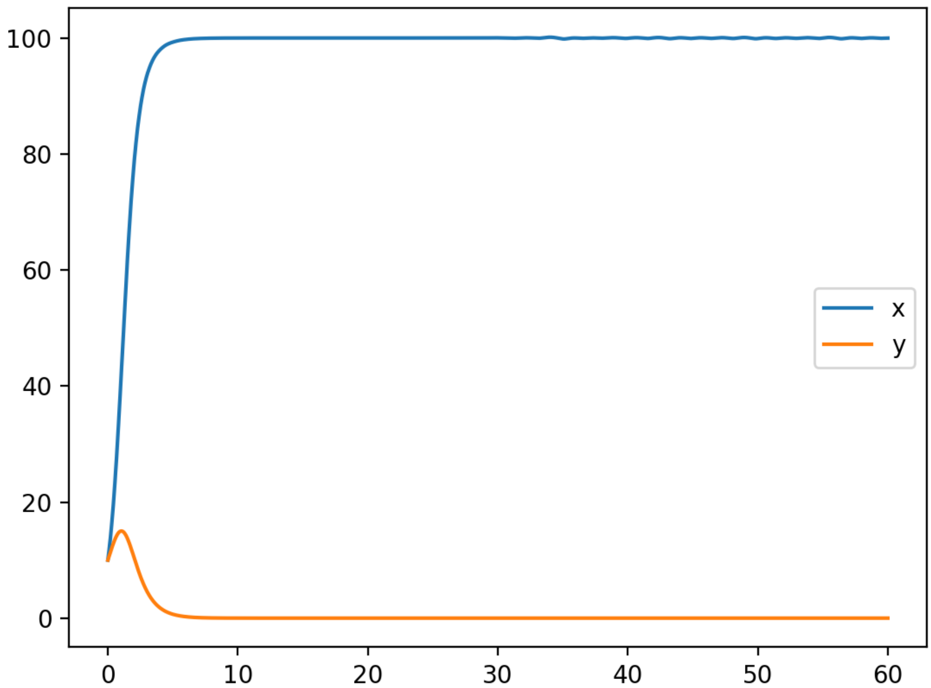
```
class fun9(function2d):
    def __init__(self,r1,r2,n1,n2,s1,s2):
        self._r1=r1
        self._r2=r2
        self._n1=n1
        self._n2=n2
        self._s1=s1
        self._s2=s2
    def get(self,x,y):
        x=self._r1*y[0]*(1.-y[0]/self._n1-self._s1*y[1]/self._n2)
        y=self._r2*y[1]*(1.-self._s2*y[0]/self._n1-y[1]/self._n2)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

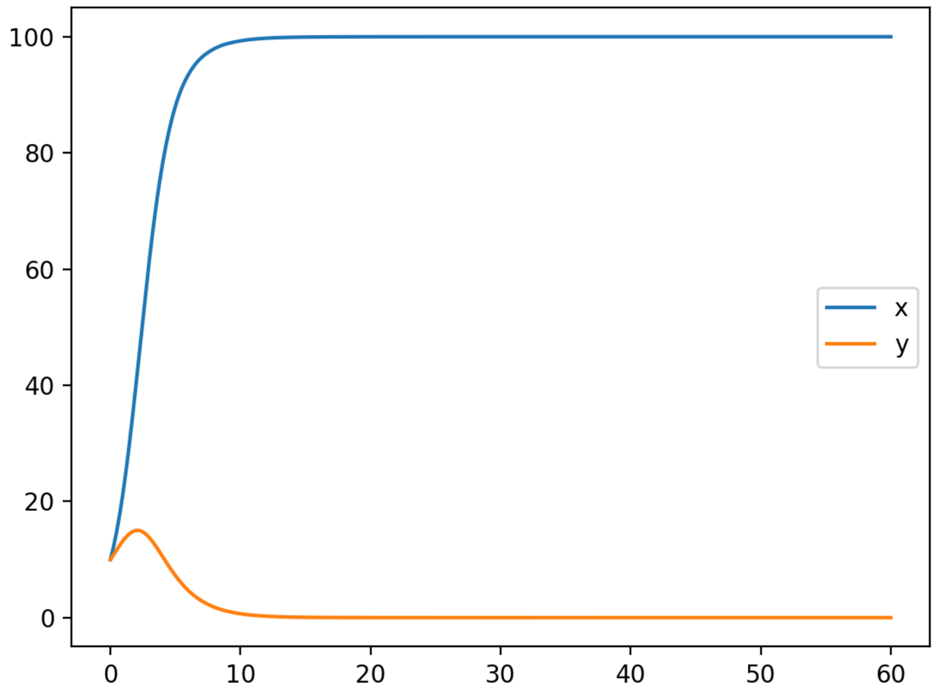


增加 r1 到 2.0，结果如下：

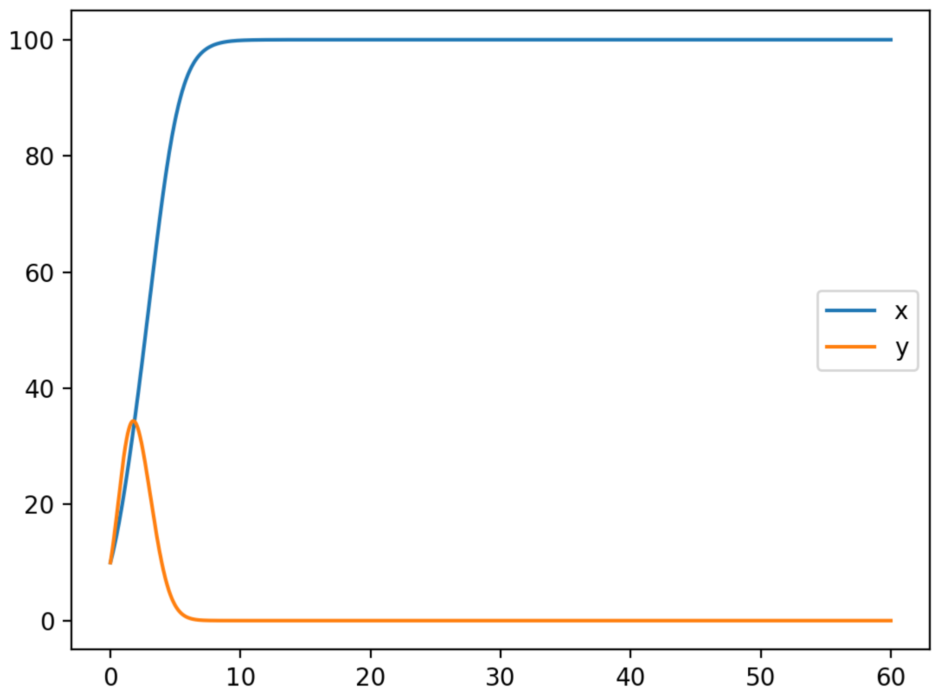


可以看出，甲增长率越高，甲到最大容量越快，乙灭绝的越快。

降低 r2 到 0.5，结果如下：

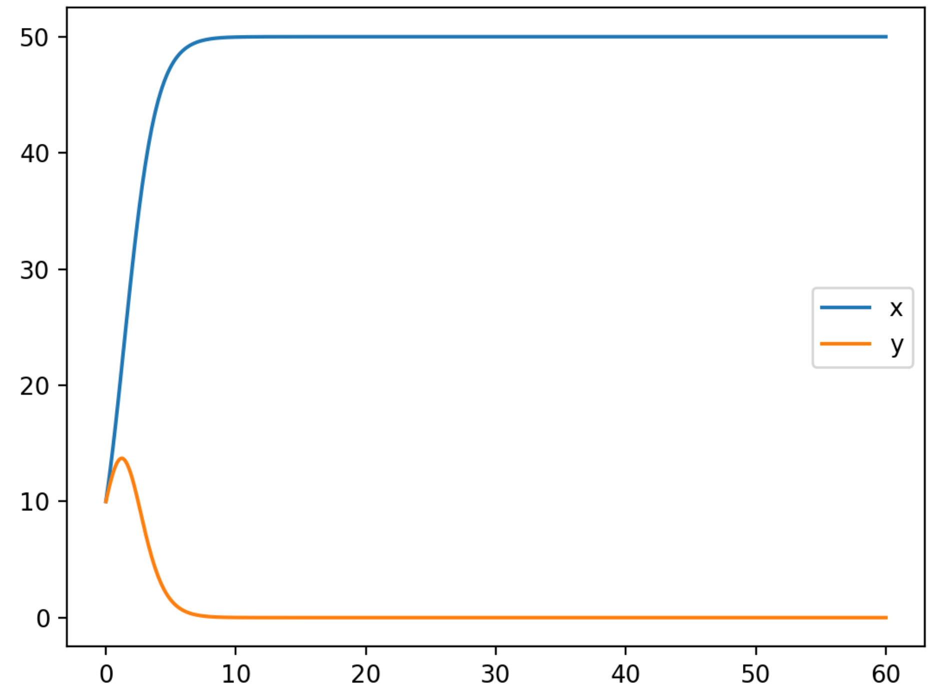


增加 r2 到 2.0，结果如下：

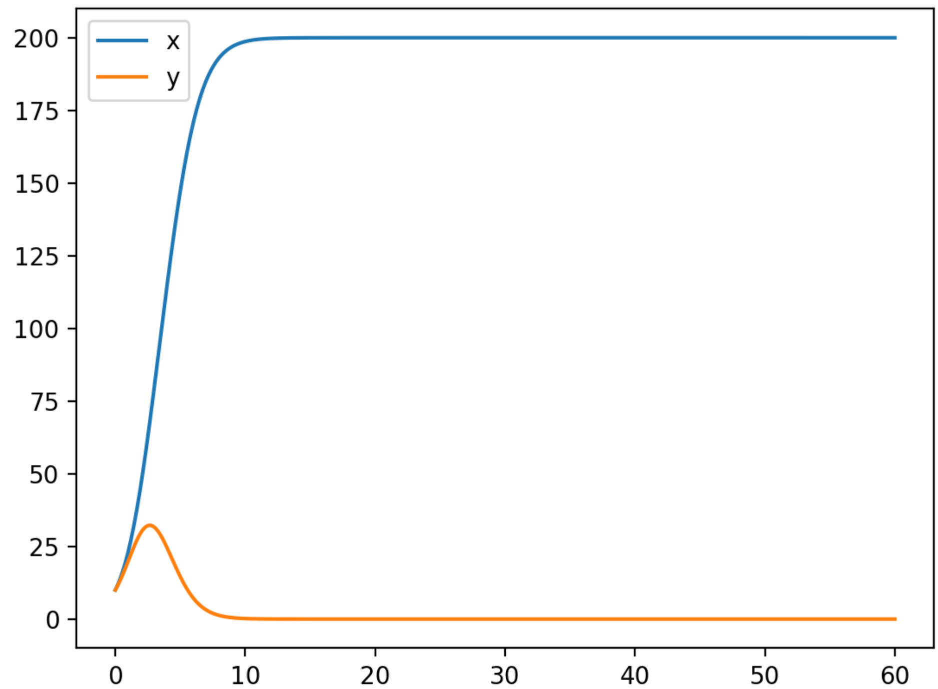


乙的增长率越高，灭绝之前到达的最大数量越高，但灭绝时间也越早，可能是消耗了过多的资源所致。

降低 n1 到 50，结果如下：

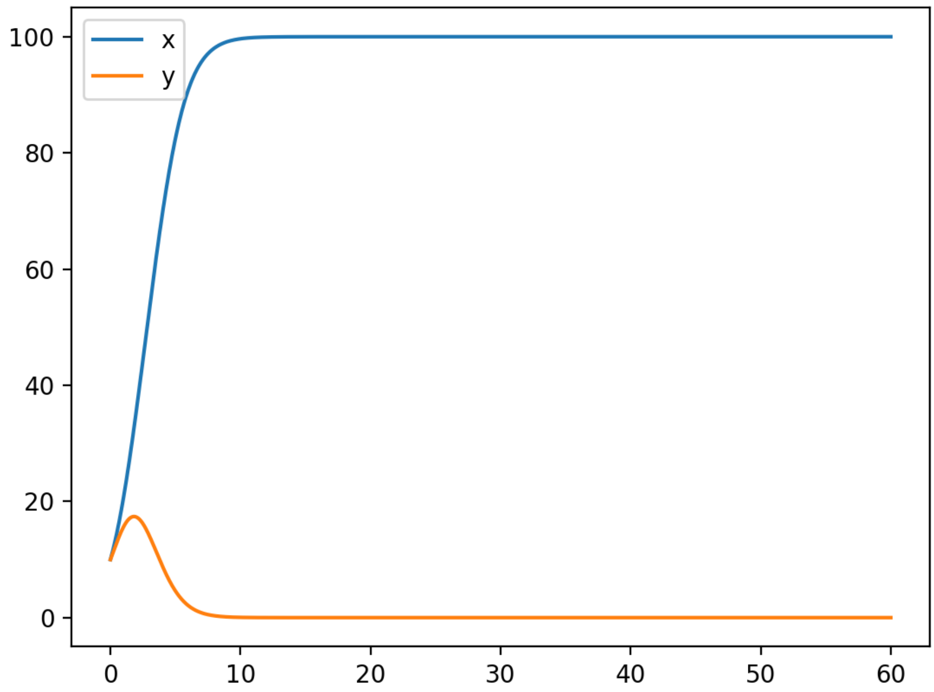


增加 n1 到 200，结果如下：

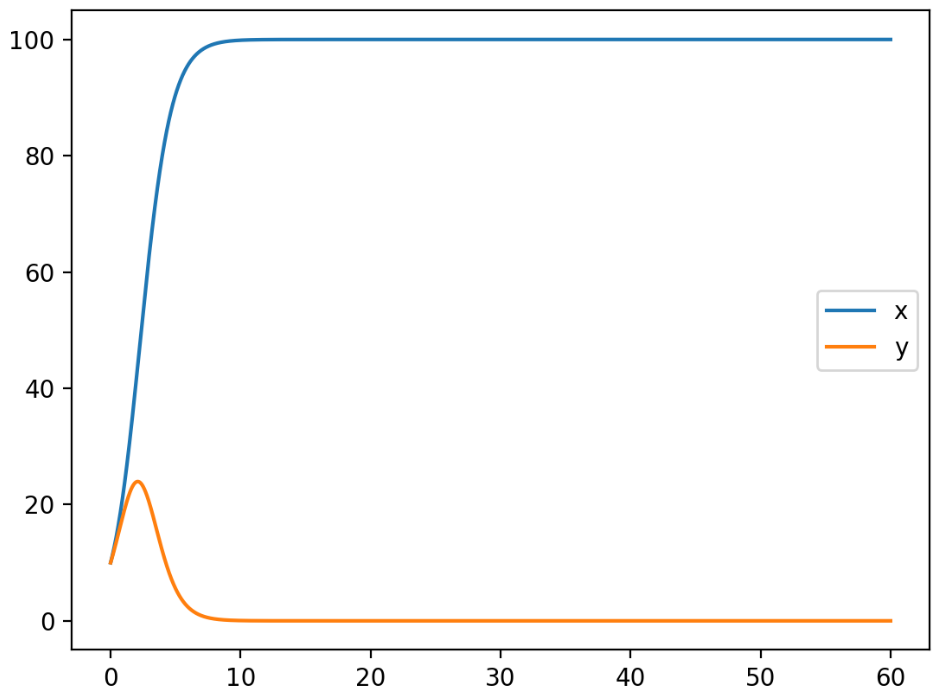


甲的最大容量越高，乙灭绝前达到的最大数量越高，原因是甲所消耗的乙资源和单位数量的甲有关。提高容量降低了单位数量的甲。

降低 n2 到 50，结果如下：



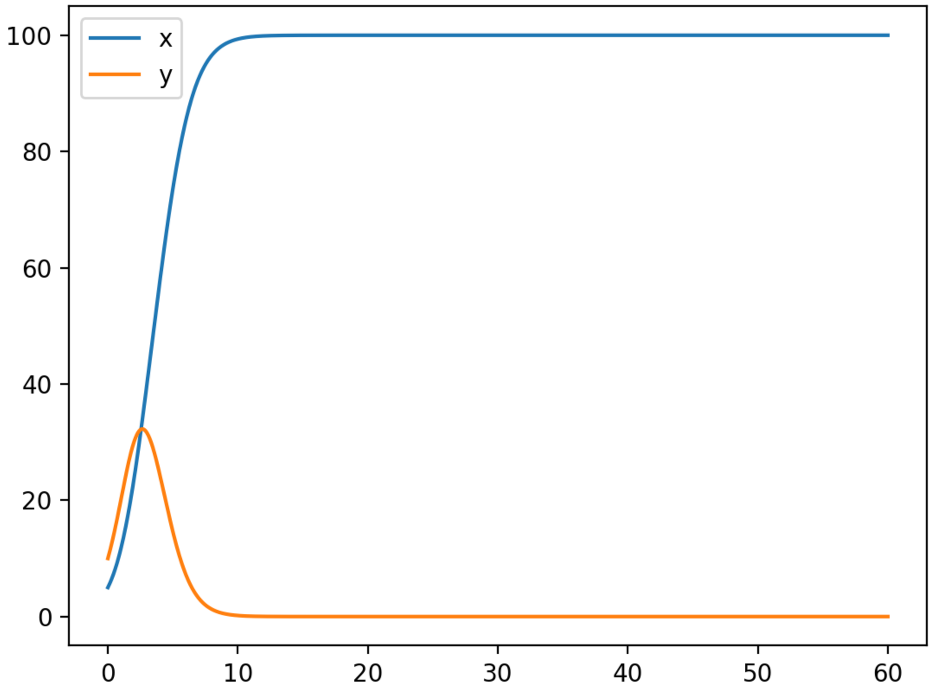
增加 n2 到 200，结果如下：



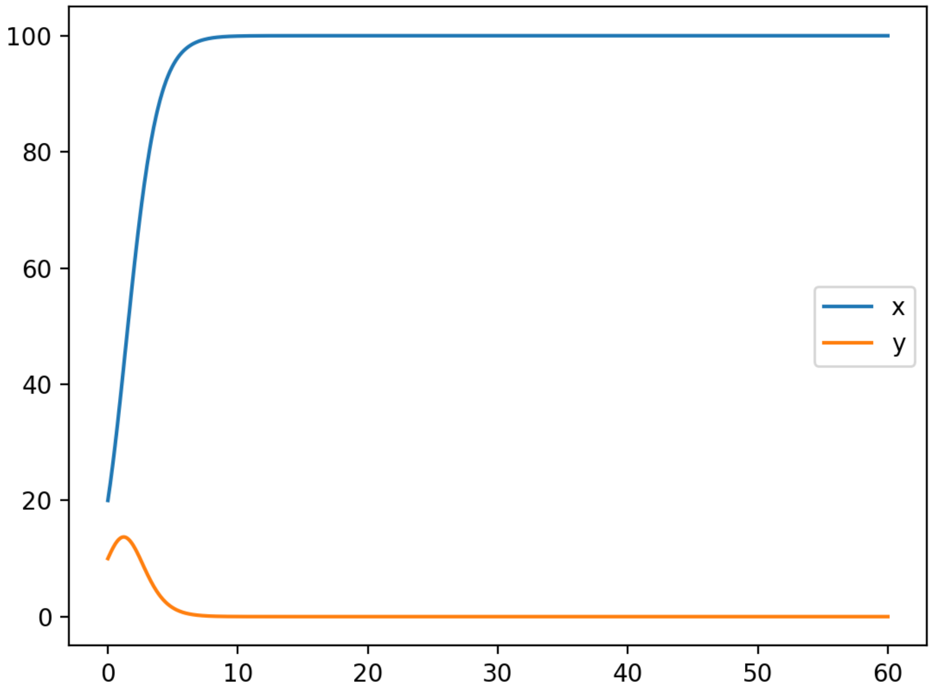
提高乙的容量降低了单位数量的乙，降低了所消耗的资源比例，故乙的容量越高，灭绝前达到的最大数量越高。

甲的初始数量越高，乙灭绝的越快。

降低 x0 到 5，结果如下：

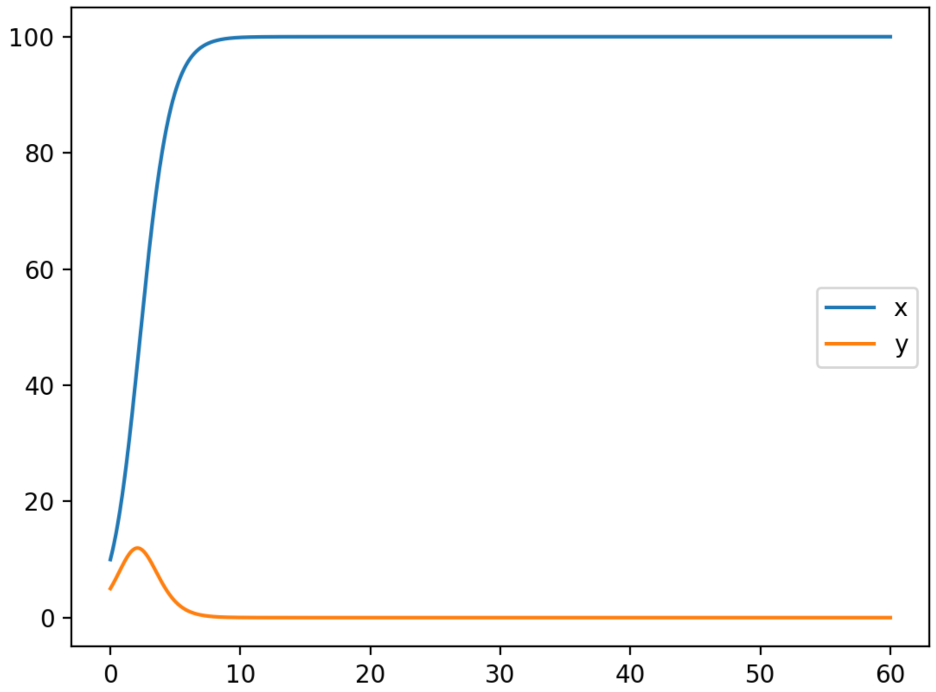


增加 x0 到 20，结果如下：

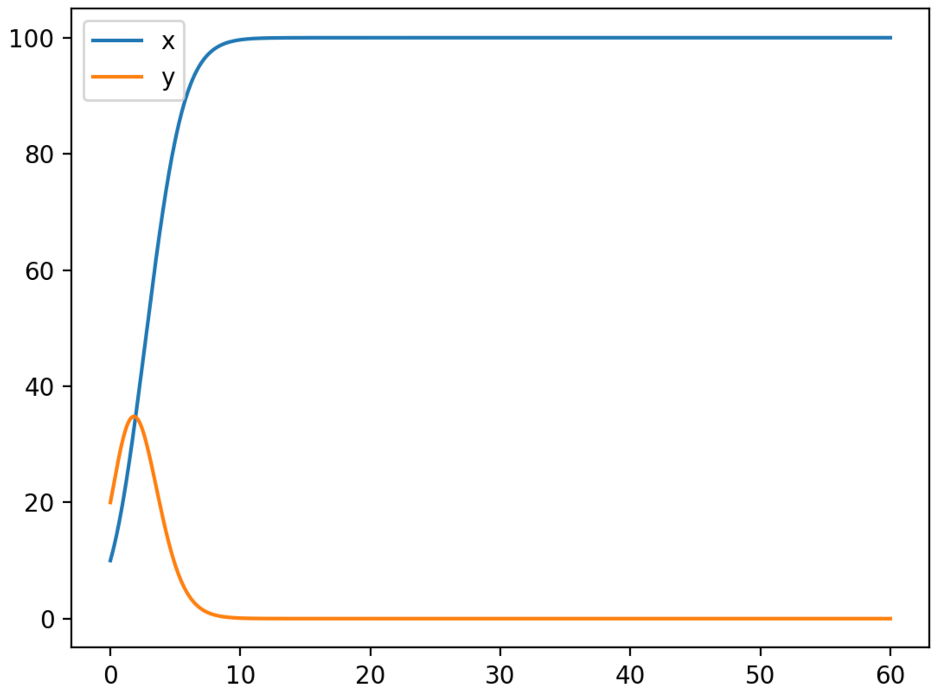


甲的初始数量越高，乙灭绝的越快。

降低 y0 到 5，结果如下：

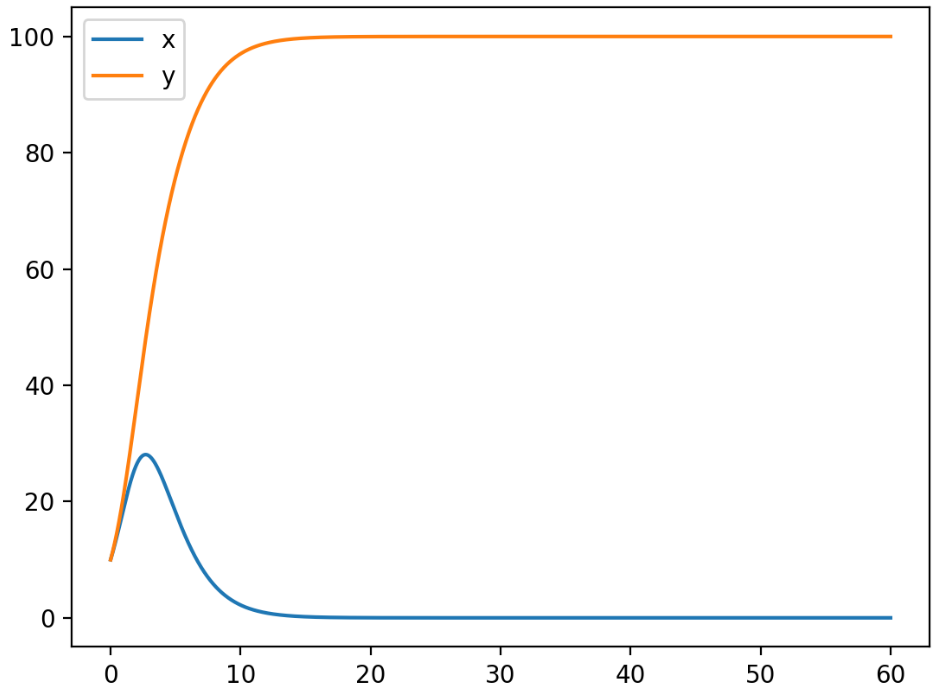


增加 y0 到 20，结果如下：



乙的初始数量越高，灭绝得越慢。

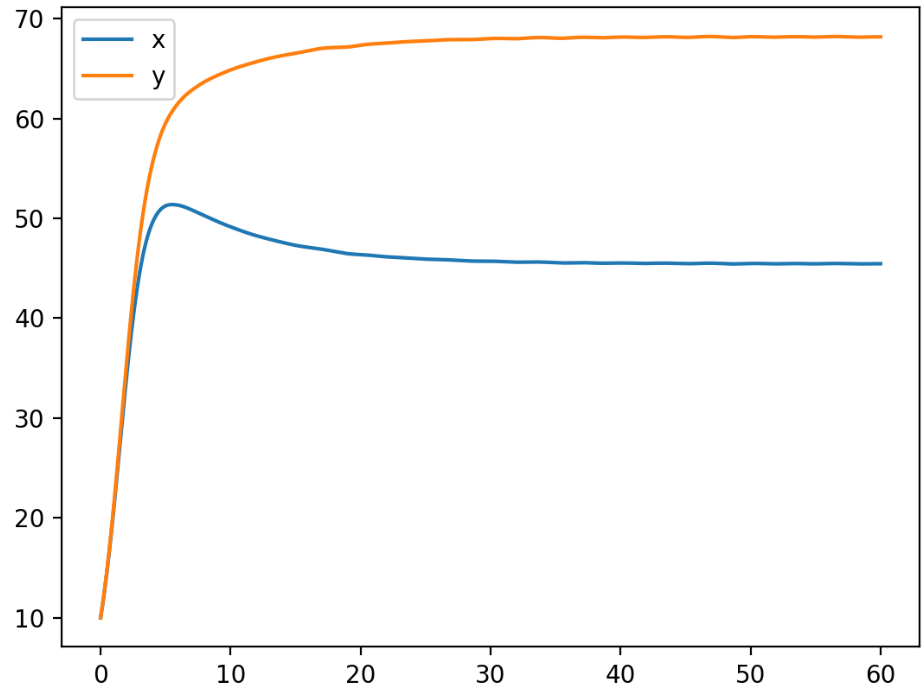
当 s1=1.5,s2=0.7 时，结果如下：



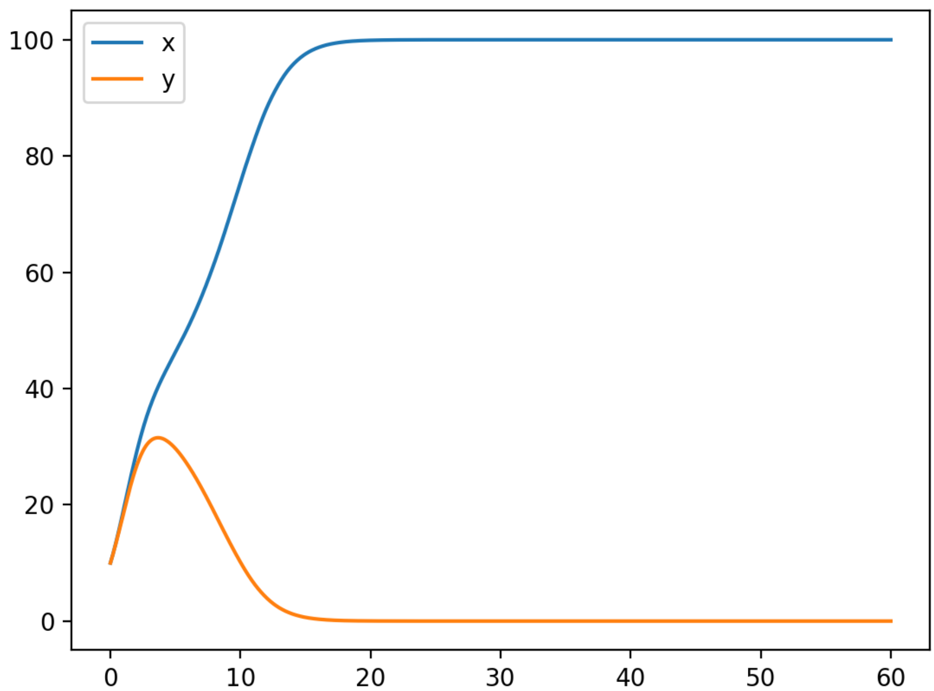
可以发现，无论如何调整数值，只要是一方的 s > 1，另一方的 s < 1，s > 1 的一方在长期会使另一方灭绝。生物学上的解释是 s>1 的一方有绝对的竞争优势，能大量抢占另一方的资源，消耗的资源却不容易被另一方抢占。

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。