5（Ex 11.179）

复现 Ex 11.178 的全部图像，并说明为什么一阶精度的向后Euler格式相比二阶精度的梯形法存在优势。

```
import matplotlib.pyplot as plt
import numpy as np
import math
from math import pi
```

Backward Euler method类的实现。

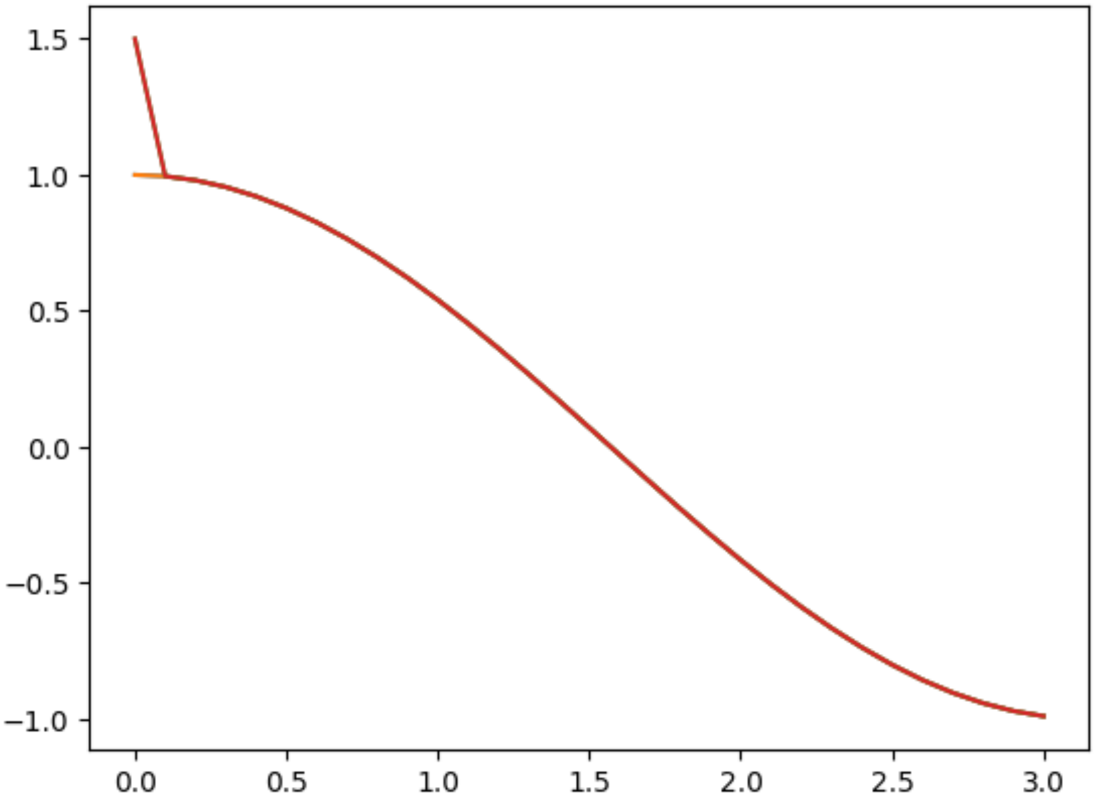
```
class Backward_Euler:
    def f(self, u, t):
        return self.lmd*(u-cos(t))-sin(t)
    def exact(self, t):
        return np.exp(self.lmd*t)*(self.u0-1)+np.cos(t)
    def step(self, u, t):
        t1 = t + self.k
        return (u - self.k*(self.lmd * np.cos(t1) + np.sin(t1))) / (1 -
self.k*self.lmd)
    def Solve(self, lmd, T, k, u0):
        self.lmd = lmd
        self.T = T
        self.k = k
        self.N = int(T/k+0.001)
        self.u0 = u0
        self.u = [u0]
        err = 0.0
        for i in range(self.N):
            self.u.append(self.step(self.u[i], i*self.k))
            err = max(err, np.abs(self.u[i+1] - self.exact((i+1)*self.k)))
        print("k = {:g}, u0 = {:g}, err = {:e}".format(k, u0, err))
    def plot(self):
        t = np.linspace(0, self.T, self.N+1)
        ans = [self.exact(x) for x in t]
        plt.plot(t, self.u)
        plt.plot(t, ans)
```

计算无穷范数下的误差并作图。误差很小，且在图像上完全没有体现。

这说明，对于这个问题，Backward_Euler 是稳定的。

```
IVPSolver = Backward_Euler()
IVPSolver.Solve(-1e6, 3, 0.2, 1)
IVPSolver.Solve(-1e6, 3, 0.1, 1)
IVPSolver.plot()
IVPSolver.Solve(-1e6, 3, 0.05, 1)
IVPSolver.Solve(-1e6, 3, 0.2, 1.5)
IVPSolver.Solve(-1e6, 3, 0.1, 1.5)
IVPSolver.plot()
IVPSolver.Solve(-1e6, 3, 0.05, 1.5)
```

```
k = 0.2, u0 = 1, err = 9.900173e-08
k = 0.1, u0 = 1, err = 4.987457e-08
k = 0.05, u0 = 1, err = 2.498388e-08
k = 0.2, u0 = 1.5, err = 2.400986e-06
k = 0.1, u0 = 1.5, err = 4.950075e-06
k = 0.05, u0 = 1.5, err = 9.974816e-06
```



trapezoidal method类的实现。

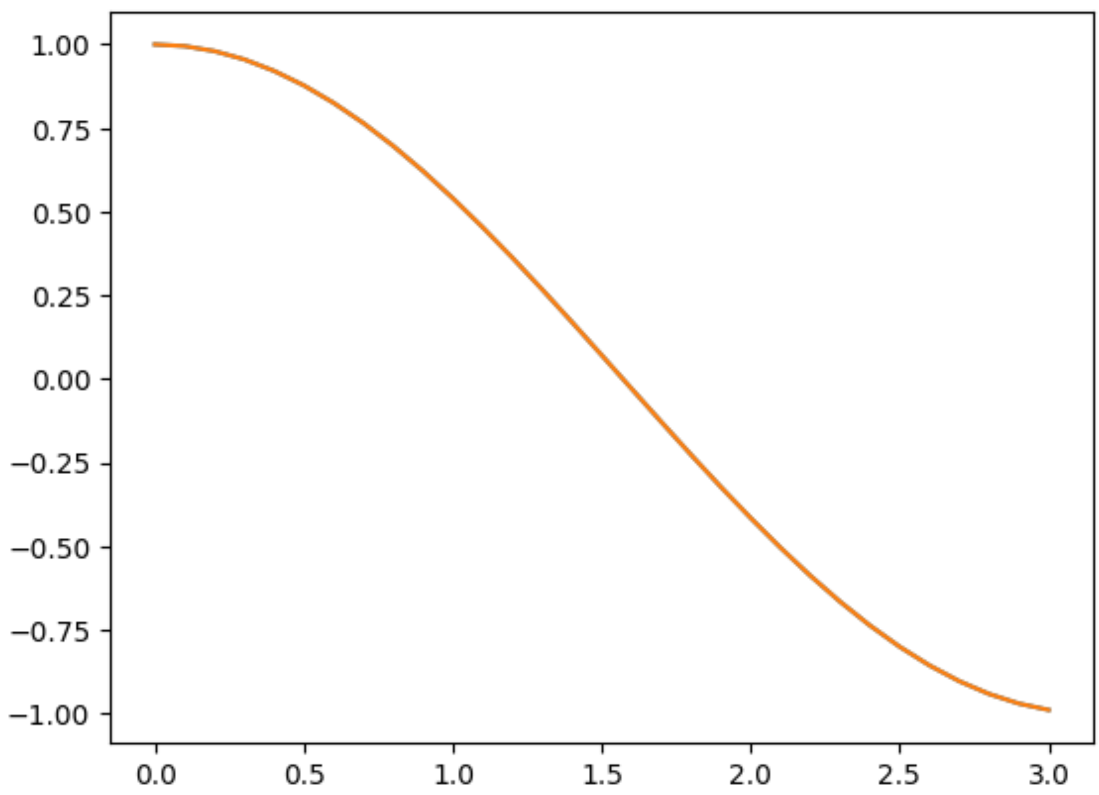
```
class trapezoidal:
    def f(self, u, t):
        return self.lmd*(u-np.cos(t))-np.sin(t)
    def exact(self, t):
        return np.exp(self.lmd*t)*(self.u0-1)+np.cos(t)
    def step(self, u, t):
        t1 = t + self.k
        return (u + self.k/2 * (self.f(u, t) - self.lmd * np.cos(t1))) / (1
- self.k*self.lmd / 2)
    def Solve(self, lmd, T, k, u0):
        self.lmd = lmd
        self.T = T
        self.k = k
        self.N = int(T/k+0.001)
        self.u0 = u0
        self.u = [u0]
        err = 0.0
        for i in range(self.N):
            self.u.append(self.step(self.u[i], i*self.k))
```

```python
                err = max(err, np.abs(self.u[i+1] - self.exact((i+1)*self.k)))
            print("k = {:g}, u0 = {:g}, err = {:e}".format(k, u0, err))
        def plot(self):
            t = np.linspace(0, self.T, self.N+1)
            ans = [self.exact(x) for x in t]
            plt.plot(t, self.u)
            plt.plot(t, ans)
            plt.show()
```

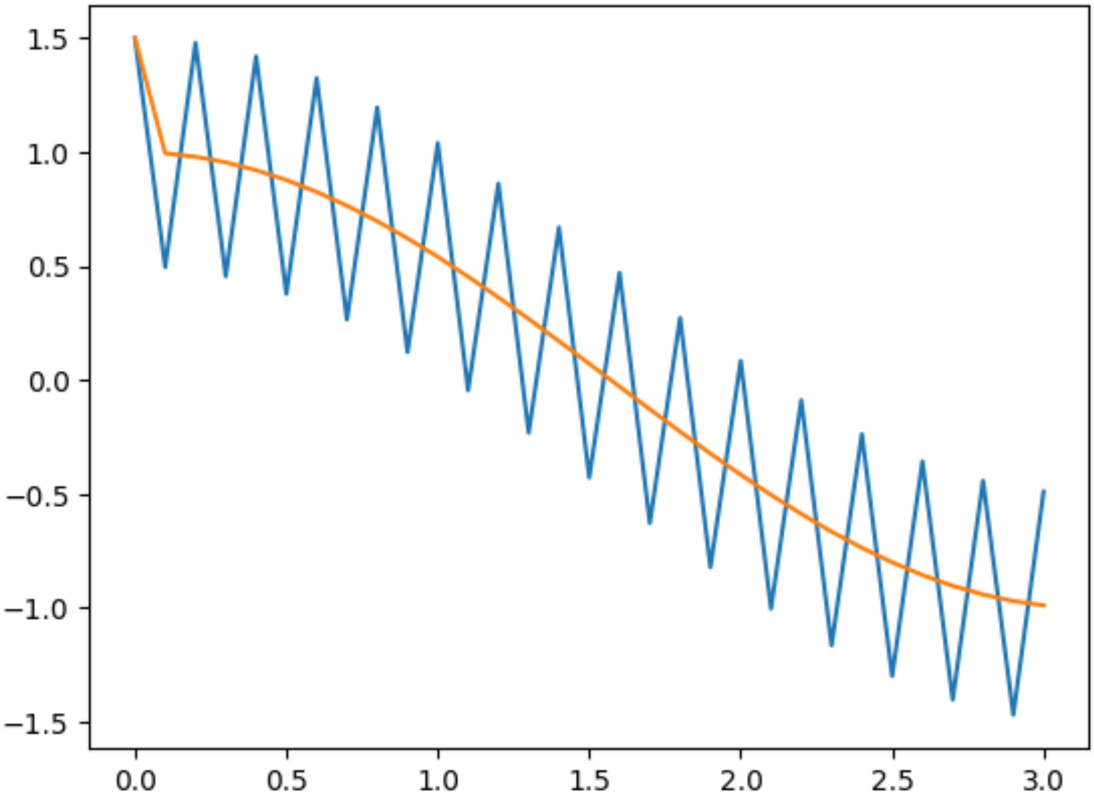
计算无穷范数下的误差并作图。当 $\eta = 1$ ，近似解误差很小，和真实解几乎相等；但当 $\eta = 1.5$ 时，近似解在真实解的曲线周围反复震荡，但不收敛。

```python
IVPSolver = trapezoidal()
IVPSolver.Solve(-1e6, 3, 0.2, 1)
IVPSolver.Solve(-1e6, 3, 0.1, 1)
IVPSolver.plot()
IVPSolver.Solve(-1e6, 3, 0.05, 1)
IVPSolver.Solve(-1e6, 3, 0.2, 1.5)
IVPSolver.Solve(-1e6, 3, 0.1, 1.5)
IVPSolver.plot()
IVPSolver.Solve(-1e6, 3, 0.05, 1.5)
```

```
k = 0.2, u0 = 1, err = 5.547095e-07
k = 0.1, u0 = 1, err = 5.263548e-07
```



```
k = 0.05, u0 = 1, err = 5.128314e-07
k = 0.2, u0 = 1.5, err = 4.999898e-01
k = 0.1, u0 = 1.5, err = 4.999799e-01
```




```
k = 0.05, u0 = 1.5, err = 4.999600e-01
```

虽然向后**Euler**法的精度比梯形法低一阶，但向后**Euler**法是**L**稳定的，梯形法不是**L**稳定的。**L**稳定性保证了向后**Euler**法在求解特征值很大的问题时对初值的敏感性不会过大。