Applied Mathematics Assignment 03
TW244 Applied Differential Equations

https://github.com/BhekimpiloNdhlela/TW244AppliedDifferentialEquations

Bhekimpilo Ndhlela (18998712)

05 September 2018

NOTE: PLEASE REFER TO THE LAST PAGE FOR UTILITY FUNCTIONS OR THE FUNCTIONS THAT PLOT THE FIGURES

Question 1

1a.]

Python Source Code For Question 1a.)

```
f = lambda x, t: ( 3*x[0] + 3*x[0]*x[1], x[1]    2*x[1]*x[0])
T, h, x0, y0 = np.arange(0, 10, 1.0/100.0), 1.0/100.0, 0.3, 1.0
Y = eulers_method(f, T, x0, y0, h)
```

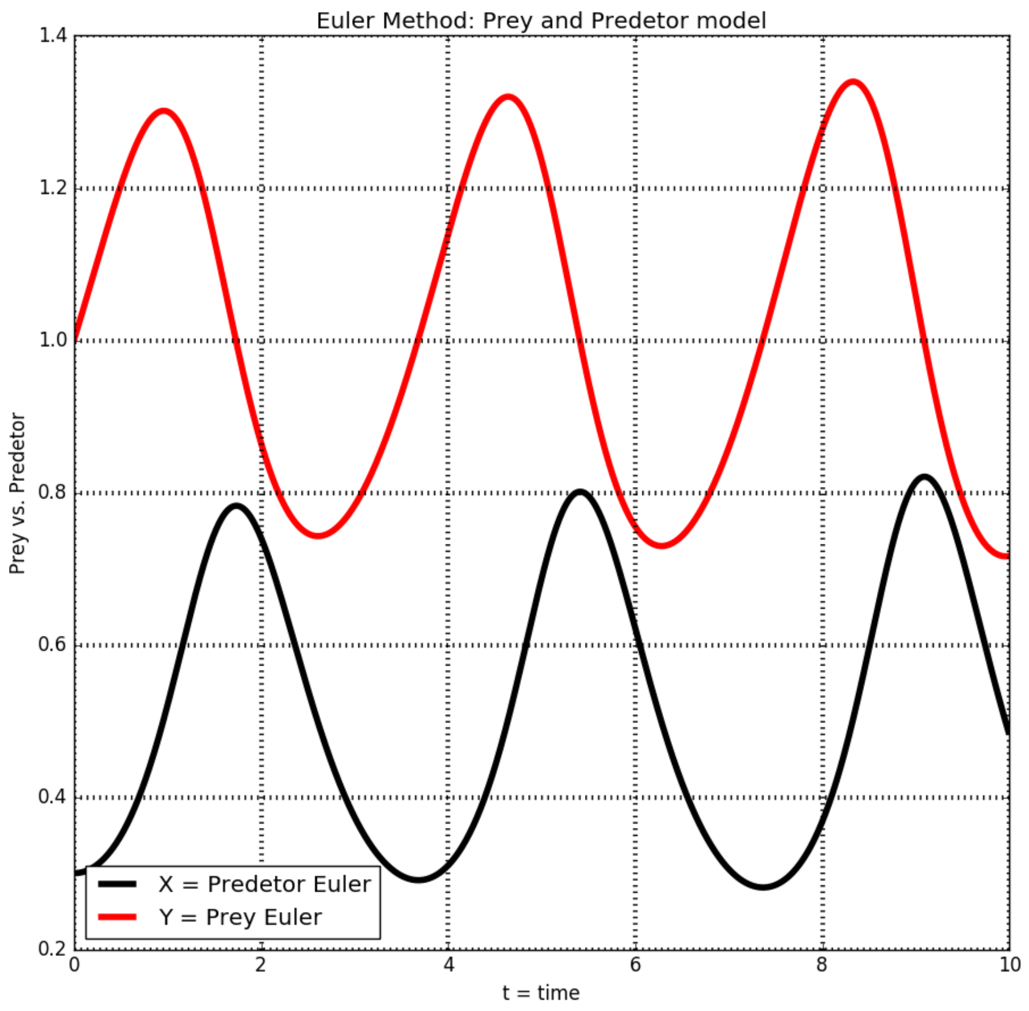

Figure 1: Resulting Plot of: $(x, f(x))$

1b.]

```
f = lambda x, t: ( 3*x[0] + 3*x[0]*x[1], x[1]    2*x[1]*x[0])
x0 = [0.3, 1.0]
odeint_sol = scipy.integrate.odeint(f, x0, T)
plot_graphs(Y, T, odeint_sol=odeint_sol)
```

From the above results the ODE solver is more accurate compared to the Euler's Method.

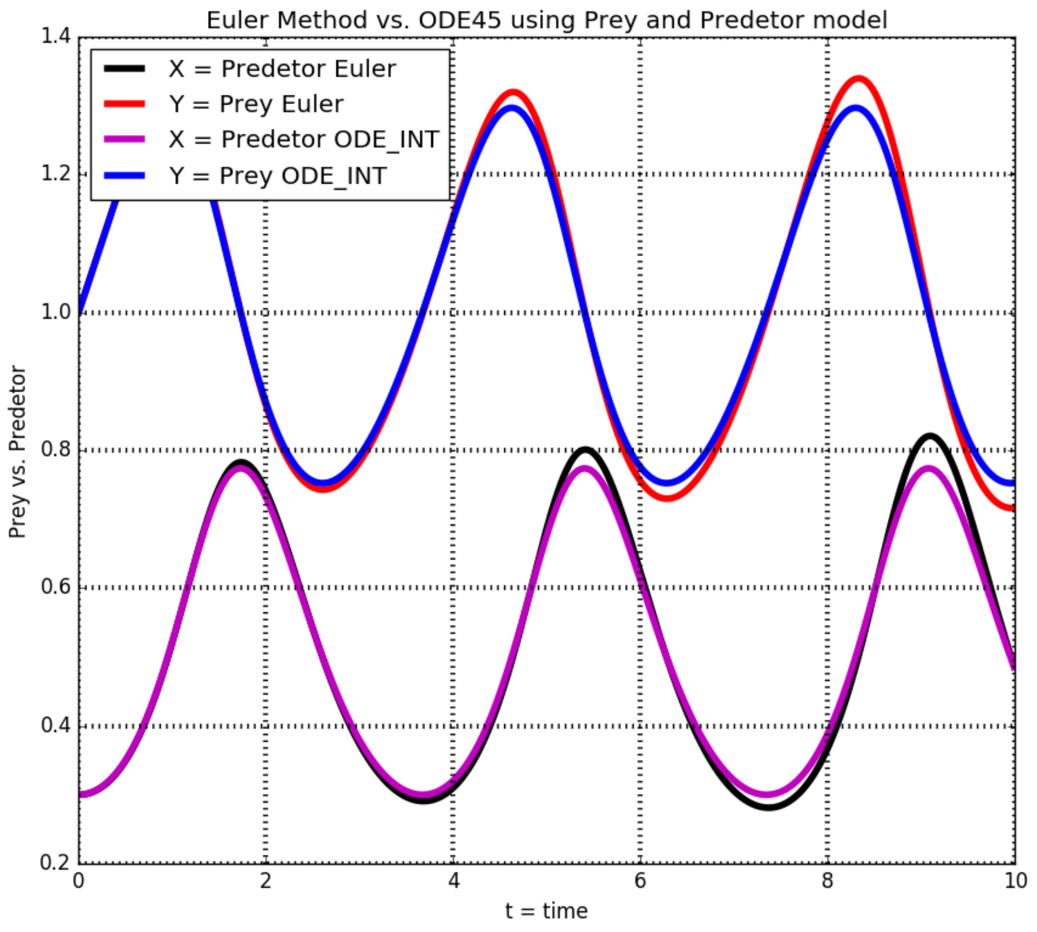

Figure 2: Euler and Ode45 Results

1c.]

	t (Days)	Predators (Thousands)
Maximum	9	821
Minimum	7	281

1d.] Optional

1e.]

Question 2

2a.]

I declare that I have studied these equations and also that I understand how and why they are modelling an infection.
And also I have managed to understand why: $S(t) + I(t) + R(t) = constant$

2b.]

```
B, G, T, x0 = .00083, .05, np.arange(0,28,1./100.), [999., 1., 0.]
f = lambda x, t: ( B*x[1]*x[0], B*x[1]*x[0]   G*x[1], G*x[1])
odeint_sol = scipy.integrate.odeint(f, x0, T)
plot_graphs(odeint_sol, T)
```

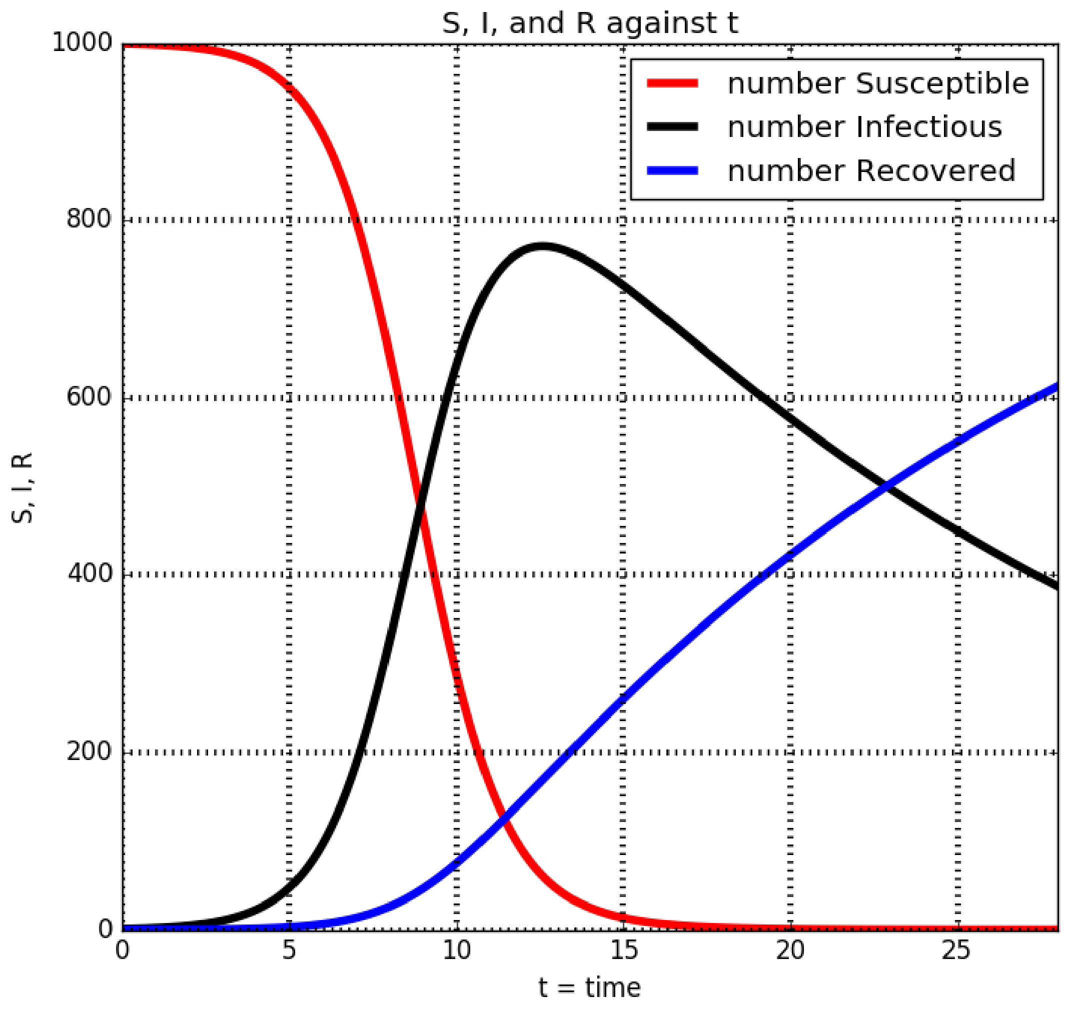

Figure 3: S, I, and R against t

2c.]

```
max_I = max(odeint_sol[:,1])      # i.] Maximum infected
I_28days = odeint_sol[:,1][ 1]    # ii.] Infected after 28 days
```

	Number Infectious
Maximum	771
After 28 Days	387

2d.]

```
G, T = 0.6, np.arange(0, 56, 1./100.)
odeint_sol = scipy.integrate.odeint(f, x0, T)
plot_graphs_Q2(odeint_sol, T)
```

By running a brute-force algorithm on the Amazon Web Services(AWS) I managed to obtain that:

$$\gamma = 0.6$$

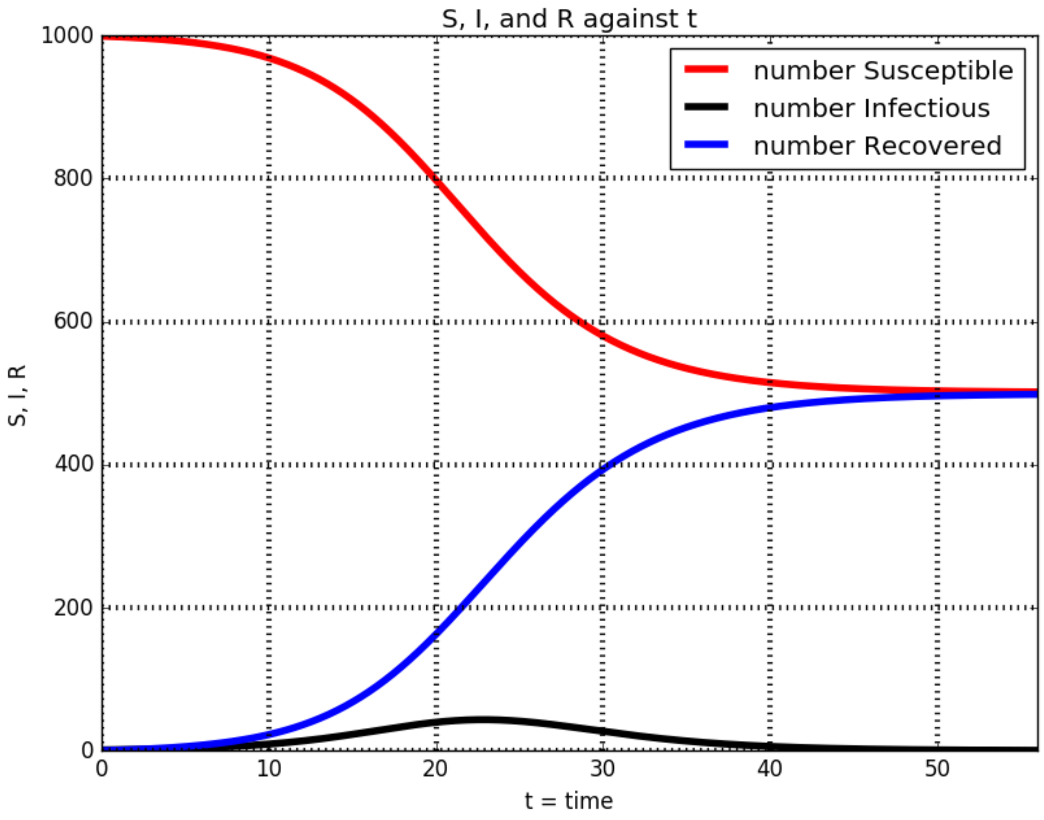

Figure 4: S, I, and R against t

2e.]

The model would be as folows:

$\frac{dS}{dt} = -\beta IS + \delta I,$

$\frac{dI}{dt} = \beta IS - \gamma I - \delta I$

$\frac{dR}{dt} = \gamma I$

Question 3

3a.]

```
X, x0 = arange(1, 4.01, 1./1000), [0, 1]
F_num = lambda x, t: (x[1], log(t)   2*x[1]   x[0])
Y0 = scipy.integrate.odeint(F_num, x0, X)[:,0]
plot_graphs(X, Y0)
```

IVP as a first-order system

$$z = y' \implies \frac{dy}{dx} = z$$

$$z' = y'' \implies \frac{dz}{dx} = \log(x) - 2z - y$$

$$y(1) = 0 \text{ , and } y'(1) = z(1) = 0$$

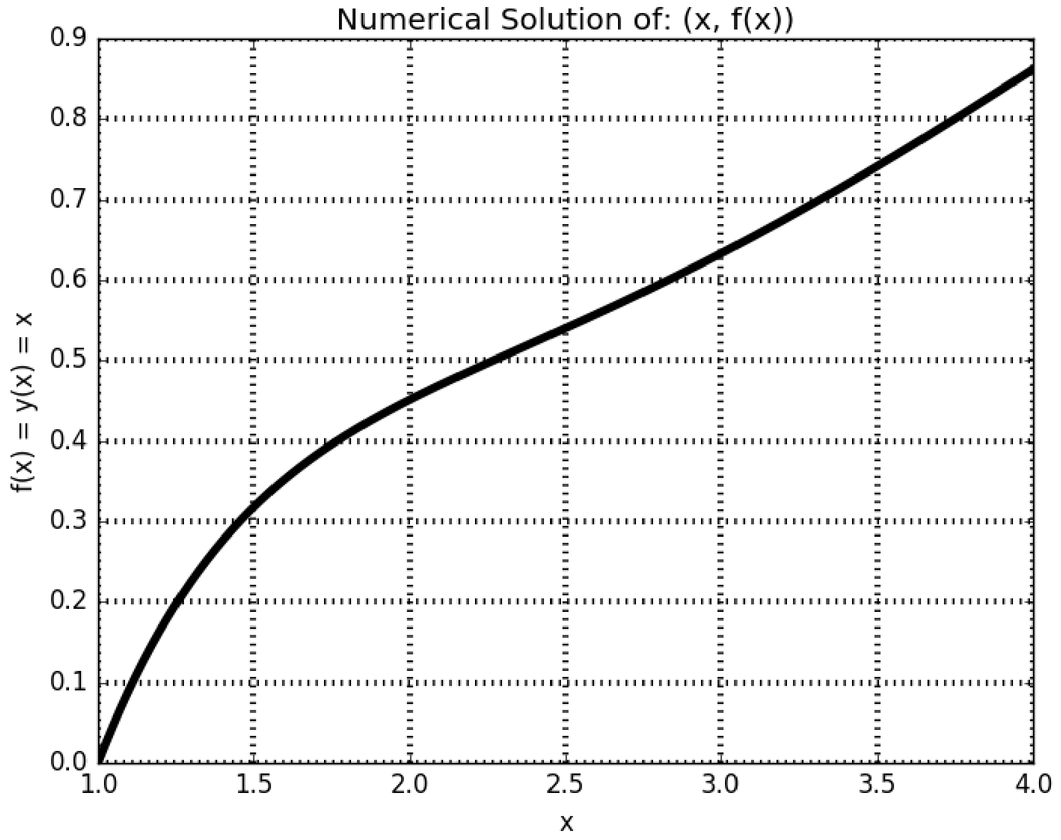

Figure 5: Numerical Solution

3b.] Supper Optional

3c.]

```
F_ana = lambda x: exp( x)*(( expi( 1)+expi( x))*(1+x)+(x 2)*exp(1))+1+log(x)
Y1 = array([F_ana(x) for x in X])
plot_graphs_Q3(X, Y0, y1=Y1)
```

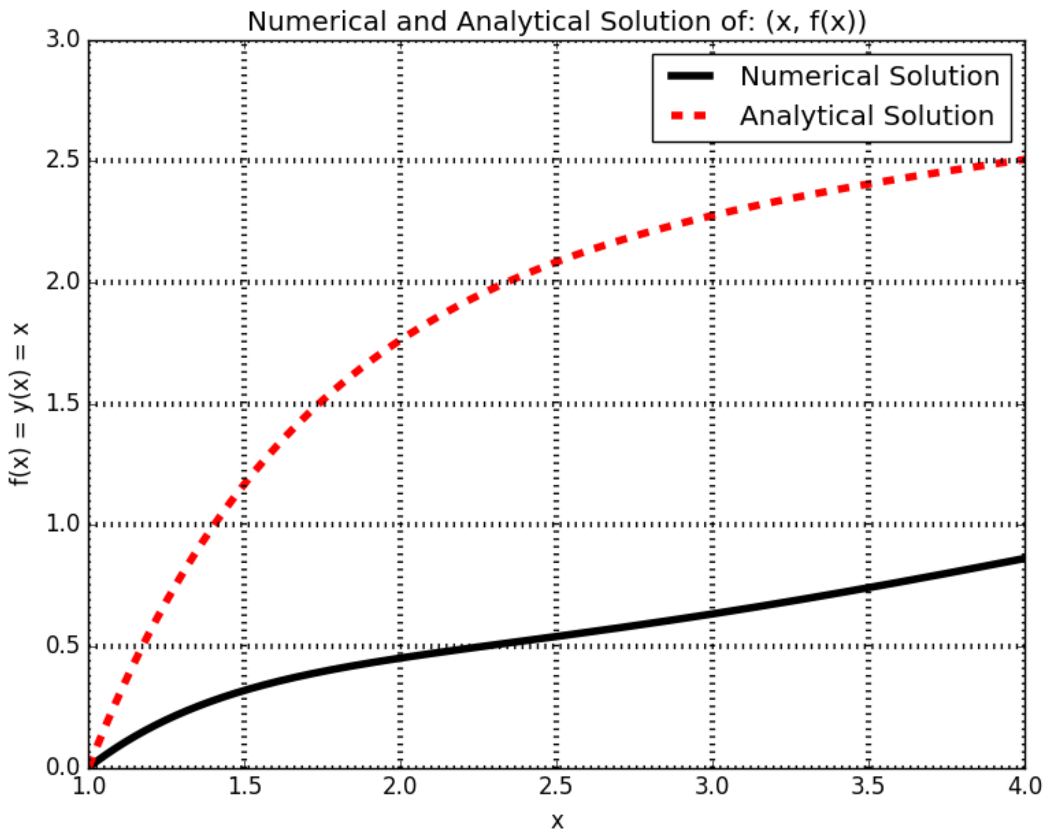

Figure 6: Numerical and Analytical Solution

Utility Functions for Question 1, 2 and 3

```
def plot_graphs_Q3(x, y, y1=[]):
    plt.xlabel('x')
    plt.ylabel('f(x)␣=␣y(x)␣=␣x')
    if y1 == []:
        plt.plot(x, y, ' k', linewidth=4)
        plt.title('Numerical␣Solution␣of:␣(x,␣f(x))')
    else:
        plt.title('Numerical␣and␣Analytical␣Solution␣of:␣(x,␣f(x))')
        plt.plot(x, y, ' k', linewidth=4, label='Numerical␣Solution')
        plt.plot(x, y1, '  r', linewidth=4, label='Analytical␣Solution')
        plt.legend(loc='best')
    plt.grid(True, linewidth=3)
    plt.xlim([1, 4])
    plt.show()

def plot_graphs_Q2(odeint_sol, t):
    plt.title('S,␣I,␣and␣R␣against␣t')
    plt.xlabel('t␣=␣time')
    plt.ylabel('S,␣I,␣R')
    plt.plot(t, odeint_sol[:,0], ' r', linewidth=4, label='number␣Susceptible')
    plt.plot(t, odeint_sol[:,1], ' k', linewidth=4, label='number␣Infectious')
    plt.plot(t, odeint_sol[:,2], ' b', linewidth=4, label='number␣Recovered')
    plt.legend(loc='best')
    plt.grid(True, linewidth=3)
    plt.xlim([0, t[ 1]])
    plt.show()

def plot_graphs(x, y, t, odeint_sol=None):
    plt.xlabel('t␣=␣time')
    plt.ylabel('Prey␣vs.␣Predetor')
    plt.plot(t, x, ' k', linewidth=4, label='X␣=␣Predetor␣Euler')
    plt.plot(t, y, ' r', linewidth=4, label='Y␣=␣Prey␣Euler')
    if odeint_sol is not None:
        plt.title('Euler␣Method␣vs.␣ODE45␣using␣Prey␣and␣Predetor␣model')
        plt.plot(t, odeint_sol[:,0], ' m', linewidth=4, label='X␣=␣Predetor␣ODE_INT')
        plt.plot(t, odeint_sol[:,1], ' b', linewidth=4, label='Y␣=␣Prey␣ODE_INT')
    else:
        plt.title('Euler␣Method:␣Prey␣and␣Predetor␣model')
    plt.legend(loc='best')
    plt.grid(True, linewidth=3)
    plt.show()

def plot_graph_Q1e(odeint_sol, t):
    plt.xlabel('t␣=␣time')
    plt.ylabel('Prey␣vs.␣Predetor')
    plt.title('Prey␣vs.␣Predetor␣model␣(10000␣as␣limiting␣Capacity␣for␣the␣Prey)')
    plt.plot(t, odeint_sol[:,0], ' m', linewidth=4, label='X␣=␣Predetor')
    plt.plot(t, odeint_sol[:,1], ' b', linewidth=4, label='Y␣=␣Prey')
    plt.legend(loc='best')
    plt.grid(True, linewidth=3)
    plt.show()
```