



Homework cover page

Analytic and machine learning-based modeling of dynamical systems - 036064

Dr. Maor Farid

☒ Homework no. ____1____

☐ Final project

Students details:

Chenhao Yang	941170706
Full name	ID no.
Jonathan Oh	941170383
Full name	ID no.

Before submitting your homework, you are requested to fill in the suitable survey on the course website. Submitting the survey is mandatory. Your feedback is highly important for the course's staff and serves us for optimizing the course contents for you and for maximizing the contribution of the HW and assimilation of the material learned. Of course, your answers do not influence your grade and evaluation in any way.

☒ I confirm that I have filled in and submitted the HW survey on the course website

Remarks:

Our submission files contain several solution files (solution.js etc.), it accelerates your computing speed if you wish to run .ipynb, just put them in the same directory.

Thanks for your collaboration and good luck!

Problem 1 Conservative one DOF system (25 pts)

A reduced-order model of a string, in length $2l$, area of A , Young's modulus of E , and non-negligible mass of m is given in the figure below. The mass displacement with respect to the equilibrium point is denoted by the coordinate x (The gravitation is neglectable).

Answers:

a. (12 pts) Formulate the non-dimensional equation of motion (EOM) of the system using Newton's method.

Answers:

Define length of the string:

$$l_x = \sqrt{l^2 + x^2} \quad (1)$$

From equation:

$$\sigma = E\epsilon \quad (2)$$

We can derive the tension force applied on the ball:

$$T = \frac{EA(l_x - l)}{l} \quad (3)$$

After trigonometry analysis, the overall force exerted on the ball is:

$$F = -2T \sin(\theta) = -2T \frac{x}{l_x} = -2 \frac{EA(l_x - l)}{l} \frac{x}{l_x} \quad (4)$$

Where θ is the angle between the string and the vertical axis. Therefore, we can formulate equation of motion of the ball from Newton's Method:

$$m\ddot{x} = -\frac{2EA(\sqrt{l^2 + x^2} - l)}{l\sqrt{l^2 + x^2}}x \quad (5)$$

Nondimensionalize EOM by setting $u = x/l$, $\omega = \sqrt{\frac{2EA}{ml}}$, $\tau = t\omega$:

$$\begin{aligned} \frac{d}{dt^2} &= \frac{d}{d\tau^2} \omega^2 \\ u_{\tau\tau} &= \frac{1 - \sqrt{1 + (u)^2}}{\sqrt{1 + (u)^2}} u = ku \end{aligned} \quad (6)$$

b. (10 pts) Formulate the system approximate EOM under the assumption of small displacements

Answers:

Expand the stiffness term using Taylor series around $u = 0$, by Matlab symbolic function:

$$k = \frac{1 - \sqrt{1 + (u)^2}}{\sqrt{1 + (u)^2}} \approx -\frac{1}{2}u^2 + \frac{3}{8}u^4 + H.O.T. \quad (7)$$

Leaving first-order term approximation is:

$$u_{\tau\tau} = ku = -\frac{1}{2}u^3 \quad (8)$$

c. (3 pts) Is the system linear under the assumption of small displacements (in first order approximation)?

Answers:



If your first order approximation means first two term of Taylor series, then we have all zeros here -- linear. If your first order approximation means first term of non-zero Taylor series, then we have $-\frac{1}{2}u^3$ here -- nonlinear.

Problem 2 – Forced dynamical system (25 pts + 5 pts Bonus)

An inverted pendulum is held by a torsion spring as shown in figure 2. The pendulum mass is m , its length is l and the spring stiffness is k . The angle between the pendulum to the vertical axis is denoted by the coordinate θ .

Q2.a. (5 pts) Find the dimensional EOM of the system with the help of Lagrange's method.

Answers:

The moment of inertia I of the rotating mass is:

$$I = m\ell^2$$

The kinetic T of the system is:

$$T = \frac{1}{2}I\omega^2 = \frac{1}{2}m\ell^2\omega^2$$

If we consider gravity effect, the potential energy of the system is:

$$V = mg\ell \cos \theta + \frac{1}{2}k\theta^2$$

And largrangian will be:

$$L = T - V = \frac{1}{2}m\ell^2\dot{\theta}^2 - mg\ell \cos \theta - \frac{1}{2}k\theta^2$$

The generalized coordinates of the system are θ , the non-conservative term will be:

$$Q_{NC} = 0$$

And there's no torsional damping so the dissipation is negligible.

$$D = 0$$

According to the Lagrange's equations, the equations of motion are:

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{\theta}} - \frac{\partial L}{\partial \theta} + \frac{\partial D}{\partial \dot{\theta}} = Q_{NC,\theta},$$

Substituting L in these equations and simplifying leads to the equations that describe the motion of the inverted pendulum:

$$m\ell^2\ddot{\theta} - mgl \sin \theta + k\theta = 0$$

$$\Rightarrow \ddot{\theta} - \frac{g}{\ell} \sin \theta + \frac{k}{I} \theta = 0$$

Question 2.b. (3 pts) Find two frequencies that play a role in the system's dynamic behavior. What are their physical meanings?

Answers:

The two frequencies are respectively:

$$\omega_g = \sqrt{\frac{g}{\ell}}, \quad \omega_k = \sqrt{\frac{k}{I}}$$

ω_g is the natural frequency of the swinging pendulum, whereas ω_k is the natural frequency of the torsional spring. Thus the EOM can be written as:

$$\ddot{\theta} - \omega_g^2 \sin \theta + \omega_k^2 \theta = 0$$

Question 2.c. (3 pts) What is the natural frequency of the system?

Answers:

- ω_g is the natural frequency of the pendulum
- ω_k is the natural frequency of the torsional spring
- If $\theta \rightarrow 0 \Rightarrow \sin \theta \approx \theta$, then the natural frequency of the system is simplified as

$$\omega_{system} = \omega_g^2 - \omega_k^2 = (\omega_g + \omega_k)(\omega_g - \omega_k)$$

Question 2.d

Apply a time normalization with respect to $\omega_{pend} = \sqrt{g/l}$. Which nondimensional parameter dictates the system's dynamics? What is its physical meaning?

Answers:

$$\omega_{pend} = \omega_g$$

$$\tau = \omega_g t$$

$$\ddot{\theta} = \frac{d^2\theta}{dt^2} = \frac{d^2\theta}{d\tau^2} \frac{d\tau^2}{dt^2} = \omega_g^2 \theta_{\tau\tau}$$

Replacing EOM with non-dimensional $\ddot{\theta}$:

$$\omega_g^2 \theta_{\tau\tau} - \omega_g^2 \sin \theta + \omega_k^2 \theta = 0$$

$$\Rightarrow \theta_{\tau\tau} - \sin \theta + \left(\frac{\omega_k}{\omega_g}\right)^2 \theta = 0$$

We can define the characteristic parameter η_c as follows:

$$\eta_c = \left(\frac{\omega_k}{\omega_g}\right)^2 = \frac{k}{gml}$$

It's clear that η_c dictates the system's dynamics. It means the dominance ratio between the torsional stiffness and the natural frequency of the pendulum. When η_c is high, the structure is stiffer. When η_c is low, the structure is softer.

Question 2.e. (3 pts) Show the nondimensional EOM of the system.

Answers:

We already got this in the previous section.

$$\theta_{\tau\tau} - \sin \theta + \eta_c \theta = 0$$

Question 2.f. (3 pts)

Find the mathematical term which defines the equilibrium point of the system. Which nondimensional parameter dictates the value of the equilibrium points?

Answers:

The system is in equilibrium when $\theta_{\tau\tau} = \theta_\tau = 0$. Thus the EOM is reduced to:

$$\sin \theta = \eta_c \theta$$

The root of this equation is the equilibrium point. The nondimensional parameter η_c dictates the value of it. We can also express this in a reduced order representation.

$$\Theta = \begin{pmatrix} \theta \\ \theta_\tau \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

$$\Theta_\tau = \begin{pmatrix} \theta_\tau \\ \theta_{\tau\tau} \end{pmatrix} = \begin{pmatrix} x_2 \\ \sin x_1 - \eta_c x_1 \end{pmatrix}$$

The equilibrium point x_e is then:

$$\Theta_\tau = 0 \Rightarrow x_e = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

As shown in the figure, if we set initial point at $\theta = 0.3$, there are three main categories of system behaviors:

- when $\eta > 0.985$ the system behaves like a normal pendulum, bouncing back and forth, the frequency increases when the parameter is larger. This is because the spring force takes bigger effect and makes it going up instead of falling down. ;
- when $\eta \approx 0.985$, the pendulum stagnates at the initial point, that's because the gravity effect counteracts with spring force, making the pendulum stay at initial point;

- when $\eta < 0.985$, the pendulum still oscillates, but instead of backing to the equilibrium ($\theta = 0$), it falls down and back to the initial point of release. This is because the gravity takes bigger effect and makes it fall back instead of going up.

```
In [1]: # Importing relevant packages
import numpy as np
import matplotlib.pyplot as plt
import math
%matplotlib notebook
%matplotlib inline
from scipy.integrate import odeint

# Defining time vector, vector of initial conditions, and system parameter
tf, dt = 100, 0.1
t_vec = np.arange(0, tf, dt)

nd_params = [0.3, 0.5, 0.985, 1.5, 2]

# Defining the numerical solver for linear oscillator
def LO_EOM(x_vec, t):
    x1, x2 = x_vec
    x_vec_dot = [x2, math.sin(x1)-nd_para*x1]
    return x_vec_dot

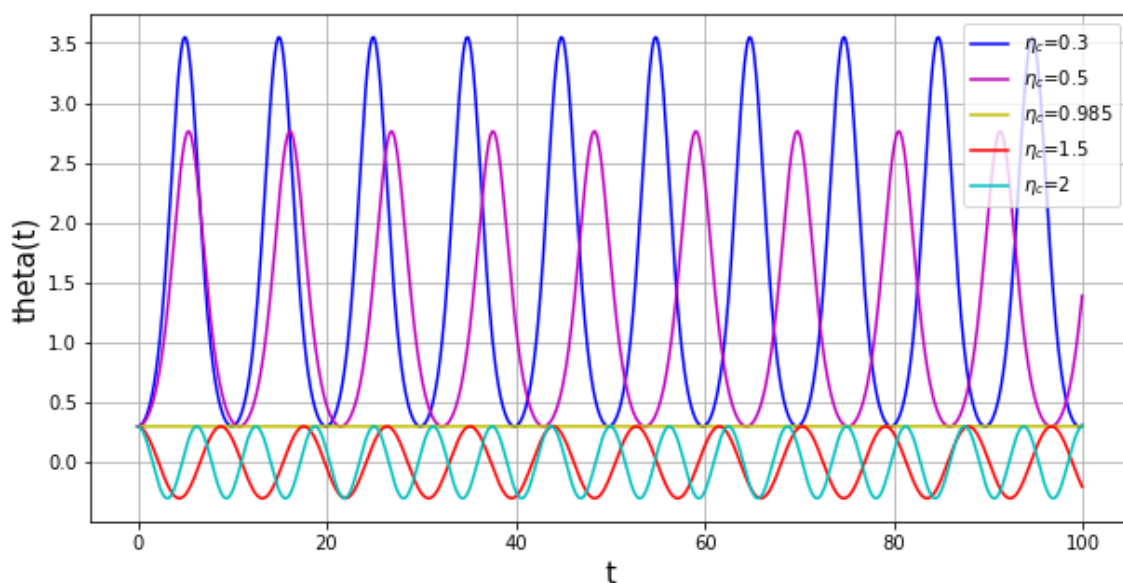
x0 = 0.3
v0 = 0

color_vec = ['b', 'm', 'y', 'r', 'c']
plt.figure(figsize=(10,5))
for i, nd_para in enumerate(nd_params):
    IC_vec = [x0, v0]
    sol = odeint(LO_EOM, IC_vec, t_vec)
    x, x_dot = sol[:, 0], sol[:, 1]
    plt.plot(t_vec, x, color_vec[i])

legend_lst = []
for i, nd_para in enumerate(nd_params):
    legend_lst.append('$\eta_c$='+str(nd_para))


plt.grid()
plt.xlabel('t', fontsize=15); plt.ylabel('theta(t)', fontsize=15)
plt.legend(legend_lst, loc='upper right')
```

Out[1]: <matplotlib.legend.Legend at 0x7fc295b34640>



Question 2.g. (5 pts + 5 pts Bonus)

Plot a bifurcation diagram that illustrates the evolution of the equilibrium point with respect to the bifurcation parameter. Hence, classify the equilibrium point types with a proper qualitative explanation. Refer only to the domain $\theta \in [-\pi, \pi]$. For which value of the bifurcation parameter a bifurcation occurred? Which kind of bifurcation occurs? Give a qualitative/intuitive physical explanation for the bifurcation- what really happens in the system when the bifurcation parameter changes?

Answers: The bifurcation point is when $\eta_c = 1$. It's a super critical pitchfork bifurcation point. 

- When $\eta_c < 1$, there are two stable equilibrium points (one positive, one negative). And there is a unstable equilibrium point at $\theta = 0$.
- When $\eta_c > 1$, there is only one stable equilibrium point at $\theta = 0$.

It can be explained physically. When $\eta_c < 1$, it means the gravitational force is more dominant than the spring stiffness. The stable equilibrium happens on both side because the spring is too soft to support the pendulum in the middle. When $\eta_c > 1$, it means the gravitational force is less dominant than the spring stiffness. The spring is stiff enough to support the pendulum in the middle.

The bifurcation diagram is plotted via python(`scipy.optimize.fsolve`) as follows:

In [2]:

```
import os.path
import math
import pandas as pd
import csv
from scipy.optimize import fsolve

# eta range
etas = np.linspace(0, 2, 2000)

# dotted line properties
dotted_line_length = 0.1
dotted_line_index = len(etas)*dotted_line_length/(etas[-1]-etas[0])/2

# create solution file if there's none
if not os.path.isfile('solution2.csv'):
    print('>>>No solution file found, calculating solutions.')
    df = None
    for i, eta in enumerate(etas):
        theta = np.linspace(-math.pi, math.pi, 201)
        func = lambda theta: eta*theta-math.sin(theta)
        # call function for calculating roots in domain
        if eta < 1:
            # calculate positive and negative roots
            theta_initail_guess1 = math.sqrt(6-6*eta)
            theta_solution1 = fsolve(func, theta_initail_guess1)
            theta_initail_guess3 = -math.sqrt(6-6*eta)
            theta_solution3 = fsolve(func, theta_initail_guess3)
            sol1 = {'eta':eta, 'theta':theta_solution1[0]}
            sol3 = {'eta':eta, 'theta':theta_solution3[0]}

            if df is None:
                df = pd.DataFrame(sol1, index=[0])
                df = df.append(sol3, ignore_index=True)
            else:
                df = df.append(sol1, ignore_index=True)
                df = df.append(sol3, ignore_index=True)

        # only calculate for dotted line region
        if i%(2*dotted_line_index)<dotted_line_index:
            # calculate zero roots
            theta_initail_guess2 = 0
            theta_solution2 = fsolve(func, theta_initail_guess2)
            sol2 = {'eta':eta, 'theta':theta_solution2[0]}
            df = df.append(sol2, ignore_index=True)
```

```

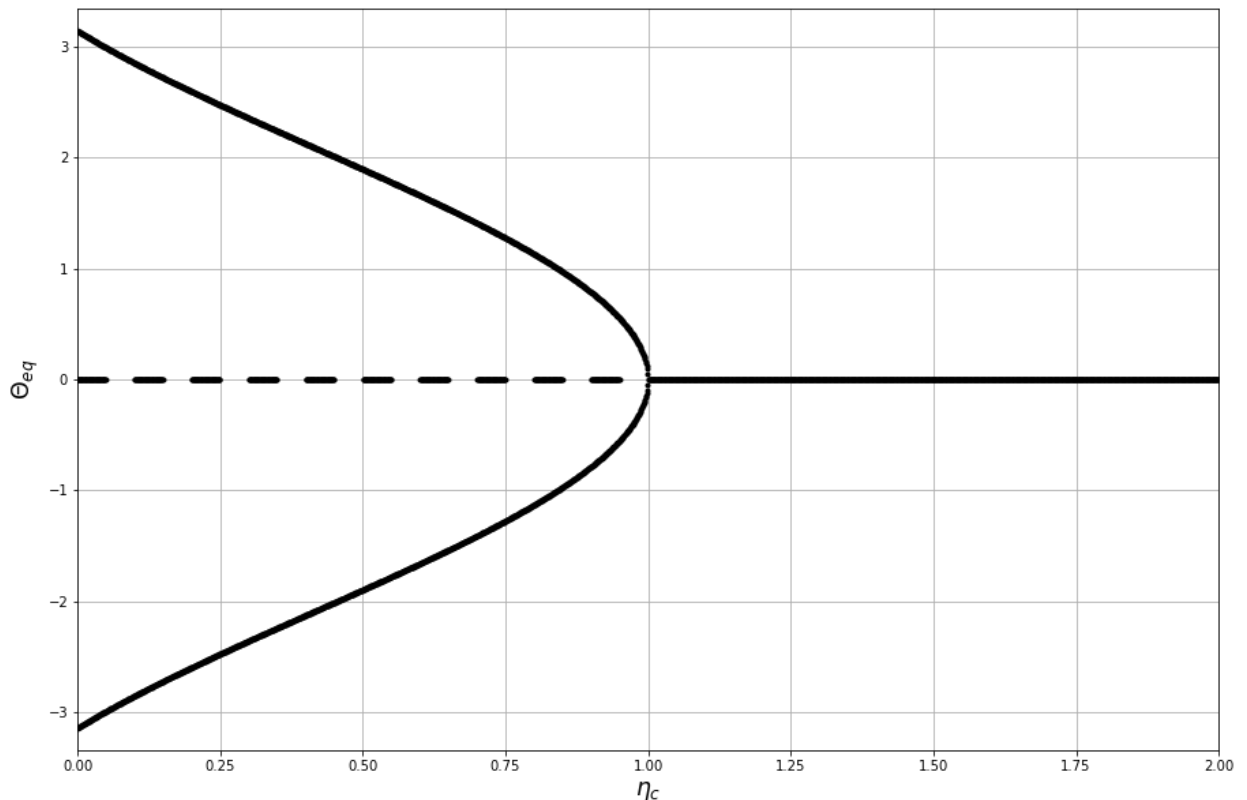
else:
    # eta > 1, there's one stable EQP at 0
    theta_initail_guess2 = 0
    theta_solution2 = fsolve(func, theta_initail_guess2)
    sol2 = {'eta':eta, 'theta':theta_solution2[0]}
    df = df.append(sol2, ignore_index=True)

print('>>Done Calculating! Saving the solutions to csv.')
data_save = df.to_csv('solution2.csv')
print('>>Solution Saved')

# load solution file
if os.path.isfile('solution2.csv'):
    print('>>Loading solution file.')
    # open output file for reading
    sols = pd.read_csv('solution2.csv')
    eta = sols['eta'].tolist()
    theta = sols['theta'].tolist()
    plt.figure(figsize=(15,10))
    plt.plot(eta, theta, 'o', color='gray',
             markersize=2, linewidth=1,
             markerfacecolor='black',
             markeredgecolor='black',
             markeredgewidth=2)
    plt.xlabel('$\eta_c$', size='xx-large')
    plt.ylabel('$\Theta_{eq}$', size='xx-large')
    plt.xlim([eta[0], eta[-1]])
    plt.grid(True)
    plt.ylim([-math.pi+0.2, math.pi+0.2])

```

>>Loading solution file.



Problem 3 Equilibrium and bifurcation analysis

Question 3.a (8 pts) For $r = 0$, find and classify all the equilibrium points and plot the obtained one-dimensional vector field.

Answers:

The equilibrium points are where:

$$\dot{x} = 0$$

For $r = 0$ the equation is written as follows:

$$\sin x = 0$$

The solution of this equation is $x = \pi k, k \in \mathbb{N}$

Classification:

Stable Equilibrium points:

*Marked in Green in graph.

$$x = 2k\pi, \quad k \in \mathbb{Z}$$

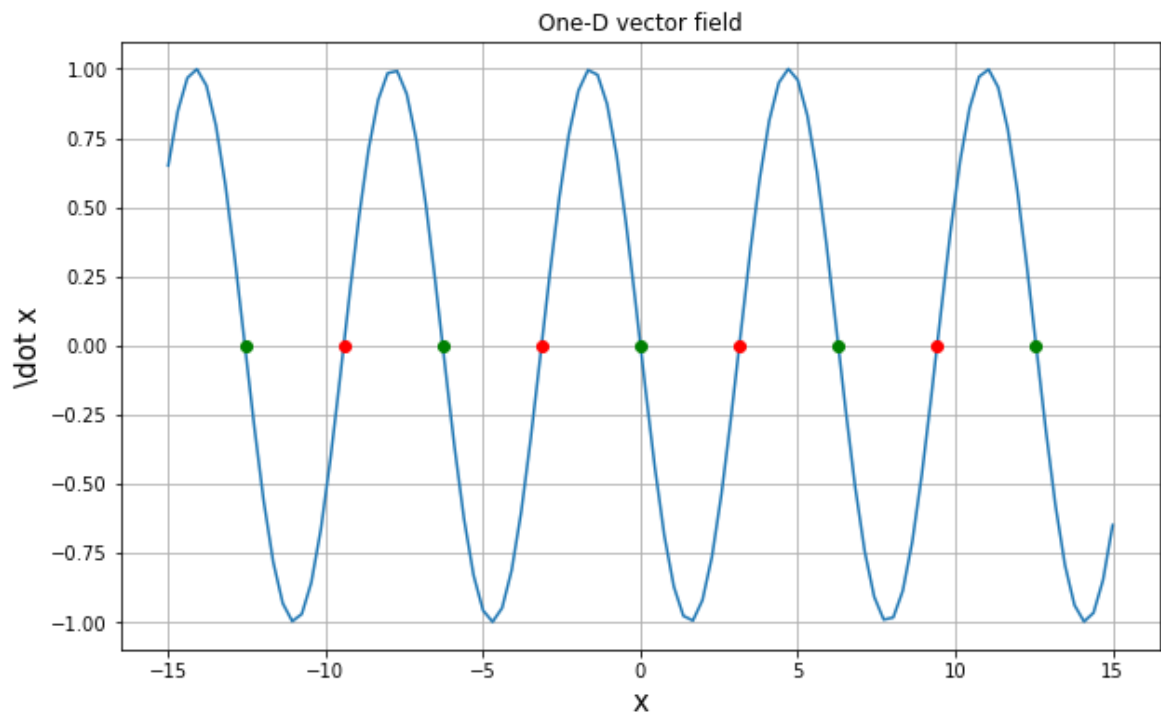
Unstable Equilibrium points:

*Marked in Red in graph.

$$x = (2k + 1)\pi, \quad k \in \mathbb{Z}$$

```
In [3]: plt.figure(figsize=[10,6])
x = np.linspace(-15,15,100)
y = - np.sin(x)
plt.plot(x,y, '-')
plt.plot([-4*3.141, -2*3.141,0, 2*3.141, 4*3.141],[0,0,0,0,0], 'o', markerfacecolor=
plt.plot([-3*3.141, -3.141, 3.141, 3*3.141],[0,0,0,0], 'o', markerfacecolor='red',m
plt.grid(True)
plt.xlabel('x', fontsize=15); plt.ylabel('\dot x', fontsize=15)
plt.title('One-D vector field')
# plt.legend(legend_lst, loc='upper right')
```

Out[3]: Text(0.5, 1.0, 'One-D vector field')



Question 3.b (8 pts) Explain why for $r > 1$ there is only one

equilibrium point? And classify its type.

Answers:

The equilibrium points are where:

$$\dot{x} = 0$$

The function for calculating equilibrium is :

$$f(x) = rx - \sin x = 0$$

And its derivative is :

$$f'(x) = r - \cos x$$

Remind that high school math class told us $\cos(x) \in [-1, 1]$ and $\cos(0) = 1$, we can conclude that when $r > 1$:

$$f'(x) > 0$$

$f(x)$ is monotonically increasing, leaving its only equilibrium point at $x = 0$. This equilibrium point is always unstable.

Question 3.c (9 pts)

Plot a bifurcation diagram that presents the values of equilibrium points as a function of the r parameter in the domain $0 \leq r < \infty$. Add a qualitative explanation that portrays bifurcation occurrence during the diminution of r from ∞ to 0.

Answers: The equilibrium points are where:

$$\dot{x} = 0$$

So we need to solve for function:

$$f(x) = rx - \sin x = 0$$

From WolframAlpha, we noticed that when $r \geq 1$, the values of equilibrium points stagnates at a single solution $x = 0$. As we decrease the value of r , the number of solutions increase significantly with speed faster than a linear scale.

To calculate the solution numerically, we used Python. Since all numerical solvers actually outputs only one solution, we use a solver that repeatedly search solutions in a certain interval in order to find all solutions.

```
In [4]: # Define a solver to find all roots in a certain interval
import math

def rootsearch(f,a,b,dx):
    x1 = a; f1 = f(a)
    x2 = a + dx; f2 = f(x2)
    while f1*f2 > 0.0:
        if x1 >= b:
            return None,None
        x1 = x2; f1 = f2
        x2 = x1 + dx; f2 = f(x2)
    return x1,x2

def bisection(f,x1,x2,switch=0,epsilon=1.0e-9):
    f1 = f(x1)
    if f1 == 0.0:
```

```

        return x1
    f2 = f(x2)
    if f2 == 0.0:
        return x2
    if f1*f2 > 0.0:
        print('Root is not bracketed')
        return None
    n = int(math.ceil(math.log(abs(x2 - x1)/epsilon)/math.log(2.0)))
    for i in range(n):
        x3 = 0.5*(x1 + x2); f3 = f(x3)
        if (switch == 1) and (abs(f3) > abs(f1)) and (abs(f3) > abs(f2)):
            return None
        if f3 == 0.0:
            return x3
        if f2*f3 < 0.0:
            x1 = x3
            f1 = f3
        else:
            x2 = x3
            f2 = f3
    return (x1 + x2)/2.0

def roots(f, a, b, eps=1e-6):
    while 1:
        x1,x2 = rootsearch(f,a,b,eps)
        if x1 != None:
            a = x2
            root = bisect(f,x1,x2,1)
            if root != None:
                pass
                sol.append(round(root,-int(math.log(eps, 10))))
            else:
                break

```

```

In [5]: import json
import os.path
rs = np.linspace(0.01, 2, 200)
plt.figure(figsize=(15,10))
sols = []
# define initial domain of search!
x_lower = 0
x_upper = 100
root_domain = [x_lower, x_upper]

if not os.path.isfile('solution.js'): # if solution file not exists
    print('>>>No solution file found, calculating solutions.')
    print('>>>Takes a long time...')
    for i, r in enumerate(rs):
        print ('Handling r = %f, %d more to go.' % (r,rs.size - i -1))
        f=lambda x:r*x-math.sin(x)
        sol = []
        # call function for calculating roots in domain
        roots(f, root_domain[0], root_domain[-1])

        # domain for next search
        root_domain = [0, math.ceil(sol[-1])]

        # mirror the answer
        sol_array = np.array(sol)
        sol_array = np.concatenate([-np.flip(sol[1:]), sol_array])
        sol = sol_array.tolist()
        sols.append(sol)

    # Plotting
    plt.plot(r * np.ones_like(sol_array), sol_array, 'o', color='gray',
             markersize=3, linewidth=1,
             markerfacecolor='white',
             markeredgecolor='gray',
             markeredgewidth=2)

```

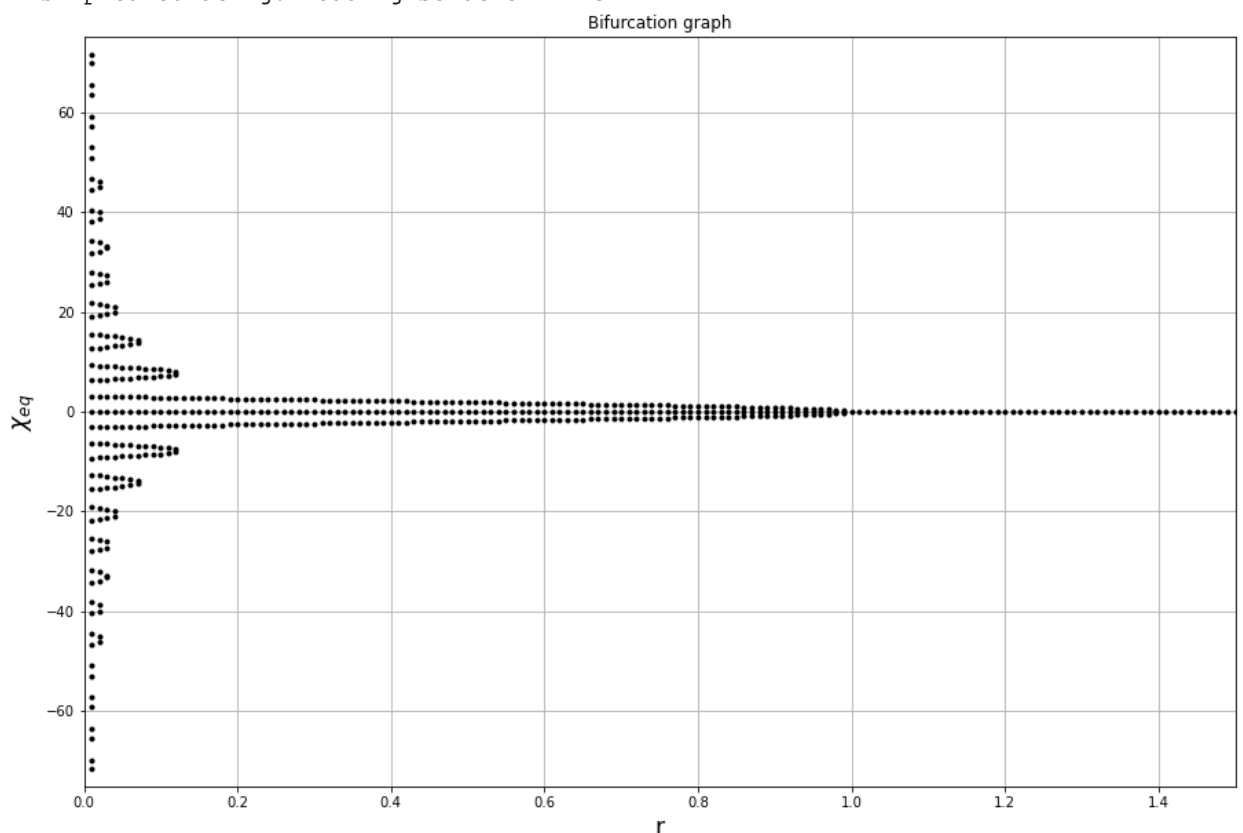
```

plt.xlim([0, 1.5])
print('Finnaly done!')
# open output file for writing
with open('solution.js', 'w') as filehandle:
    json.dump(sols, filehandle)
else:
    print('>>Skip calculating! Loading solution file.')
    # open output file for reading
    with open('solution.js', 'r') as filehandle:
        sols = json.load(filehandle)

    for i, r in enumerate(rs):
        plt.plot(r * np.ones_like(sols[i]), sols[i], 'o', color='gray',
                 markersize=2, linewidth=1,
                 markerfacecolor='black',
                 markeredgecolor='black',
                 markeredgewidth=2)
    plt.xlabel('r', size='xx-large')
    plt.ylabel('$\chi_{eq}$', size='xx-large')
    plt.title('Bifurcation graph')
    plt.xlim([0, 1.5])
    plt.grid(True)
    plt.ylim([-75, 75])

```

>>Skip calculating! Loading solution file.



The bifurcation diagram depicts as above, all zeros when $x \geq 1$ and bifurcates when $r : 1 \rightarrow 0$.

The graph of $rx - \sin x$ contains two parts, rx and $\sin x$: rx is monotone increasing and has only one zero at $x = 0$, while $\sin x$ is periodical and has numerous solutions in our interval $x \in [0, \infty]$.

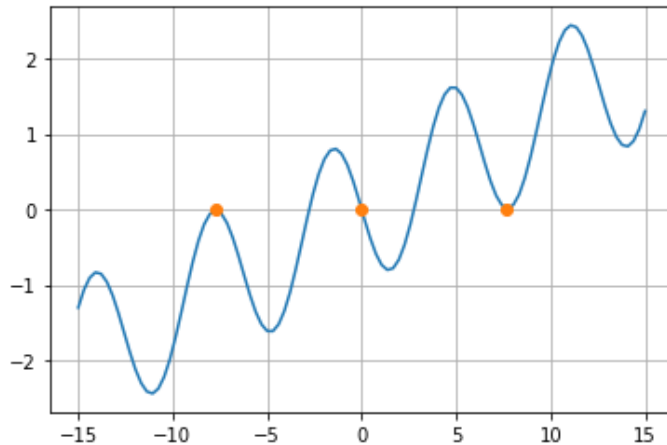
When $r > 1$, the linear part is more significant. Though the graph is still oscillating, it does not have chance to intersect with the zero line.

When $r = 1$, the function is tangent with zero line at $x = 0$, that's where bifurcation starts to take place.

We can further inspect on second bifurcation, you can see our figure below, when $r \approx 0.13$, that when the local minima of the second period of $\sin x$ is interacting with zero line.

As r continuously decreases, the whole graph becomes more "flat" because the rx term is less dominant and period term becomes more significant, thus it bifurcates and more solutions show up.

```
In [6]: # a plot showing 'bifurcation' phenomenon
x = np.linspace(-15,15,100)
y = 0.13*x - np.sin(x)
plt.plot(x,y, '-')
plt.plot([7.7,0,-7.7],[0,0,0], 'o')
plt.grid(True)
```



Problem 4 - (25 pts + 5 pts Bonus) Equilibrium and bifurcation analysis

A one-dimensional dynamical equation is given:

$$\dot{x} = r - x - e^{-x}$$

Question 4.a. (15 pts)

For $0 \leq r \leq 3$ plot by MATLAB bifurcation diagram which portrays the evolution of the equilibrium point x_{eq} versus the bifurcation parameter r . Describe the types of bifurcations and the r values of which the bifurcations took place. (5 pts Bonus for using python).

Just like what we did in question 3, let's solve equilibrium points x_{eq} and obtain bifurcation graph using Python !

```
In [7]: import json
import os.path
rs = np.linspace(0, 3, 300)
plt.figure(figsize=(15,10))
sols = []
# define initial domain of search!
x_lower = -3
x_upper = 3.5
root_domain = [x_lower, x_upper]

if not os.path.isfile('solution4.js'): # if solution file not exists
    print('>>>No solution file found, calculating solutions.')
    print('>>>Takes a long time...')
    for i, r in enumerate(rs):
        print ('Handling r = %f, %d more to go.' % (r,rs.size - i - 1))
        f=lambda x:r-x-math.exp(-x)
        sol = []
        # call function for calculating roots in domain
        roots(f, root_domain[0], root_domain[-1])
        sols.append(sol)
```

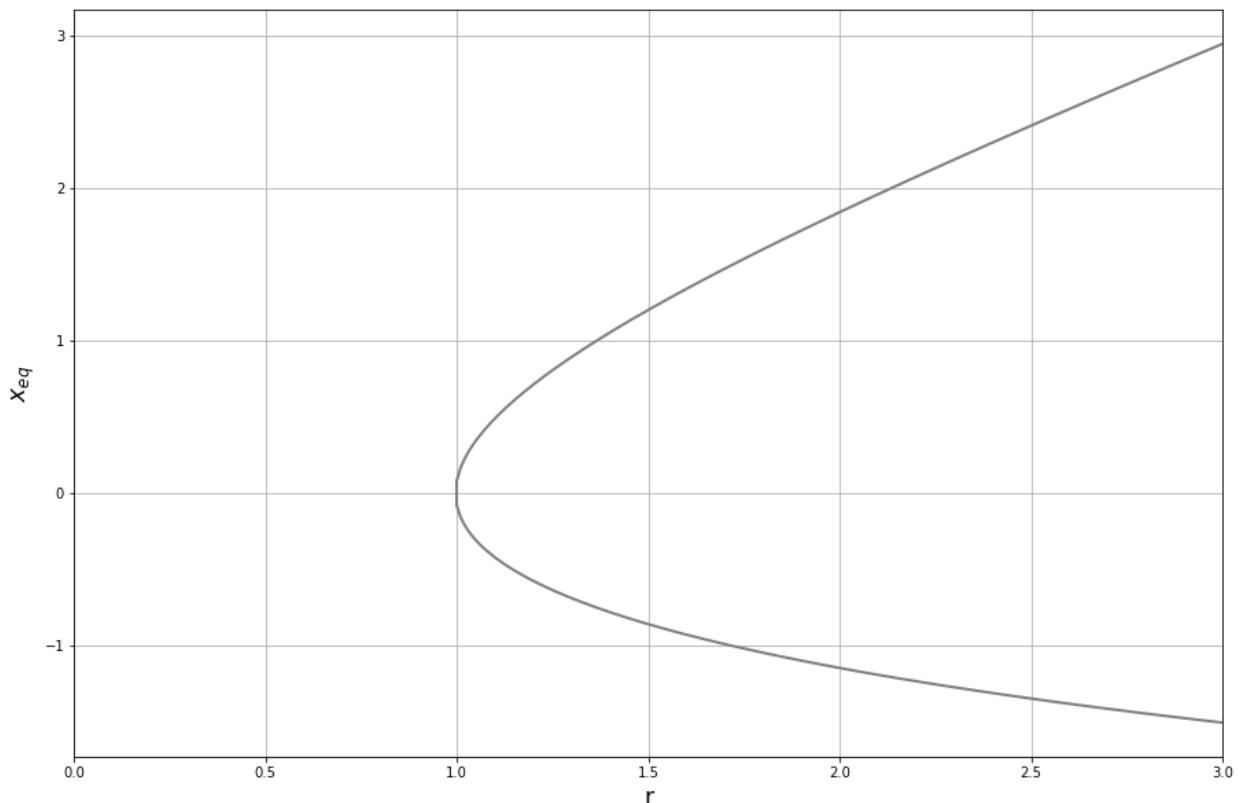
```

# Plotting
plt.plot(r * np.ones_like(sol), sol, 'o', color='gray',
         markersize=3, linewidth=1,
         markerfacecolor='white',
         markeredgecolor='gray',
         markeredgewidth=2)
print('Finally done!')
# open output file for writing
with open('solution4.js', 'w') as filehandle:
    json.dump(sols, filehandle)
else:
    print('Skip calculating! Loading solution file.')
    # open output file for reading
    with open('solution4.js', 'r') as filehandle:
        sols = json.load(filehandle)
    arr = np.array(sols[100:300])
    arr = np.transpose(arr)
    sols = arr.tolist()
    rs = np.linspace(1, 3, 200)

plt.plot(rs, sols[0], '-', color='gray',
         linewidth=2,
         markeredgecolor='gray',
         markeredgewidth=2)
plt.plot(rs, sols[1], '-', color='gray',
         linewidth=2,
         markeredgecolor='gray',
         markeredgewidth=2)
plt.plot([1,1], [sols[0][0], sols[1][0]], '-',
         color='gray',
         linewidth=2,
         markeredgecolor='gray',
         markeredgewidth=2)
plt.xlabel('r', size='xx-large')
plt.ylabel('$x_{eq}$', size='xx-large')
plt.xlim([0, 3])
plt.grid(True)

```

Skip calculating! Loading solution file.



Please note that the bottom half of the graph should be dotted(because they are unstable equilibrium points, explained later)



Then let's approximate the function with Taylor series:

$$\begin{aligned} p(x) &= r - x - e^{-x} = r - \sum_{k=0}^{\infty} \frac{(-x)^k}{k!} - x \\ &= r - 1 - \frac{x^2}{2} + \frac{x^3}{6} - \frac{x^4}{24} + \frac{x^5}{120} - \frac{x^6}{720} + \frac{x^7}{5040} - \frac{x^8}{40320} + O(x^9) \end{aligned}$$

Let's check how many order of approximation will be sufficient to obtain close roots.

```
In [8]: r = 3
num_order = 10
# create numpy array of values of true values
x = np.linspace(-5,5,100)
true_y = r - x - np.exp(-x)

# create numpy array of values of f(x)
ys = np.empty((0, x.size))
for i in range(num_order):
    y = np.zeros_like(x)
    y += r - x
    for k in range(i+1):
        y +=- np.power(-x,k)/math.factorial(k)
    y = np.expand_dims(y, 0)
    ys = np.append(ys, y, axis=0)

# Plotting
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15,5))

ax1.plot(x, true_y, '-', color='red', linewidth=2)
ax1.set(xlim=[-4,4], ylim=[-8,3])
ax2.set(xlim=[-4,4], ylim=[-8,3])
ax2.plot(x, true_y, '-', color='red', linewidth=2)

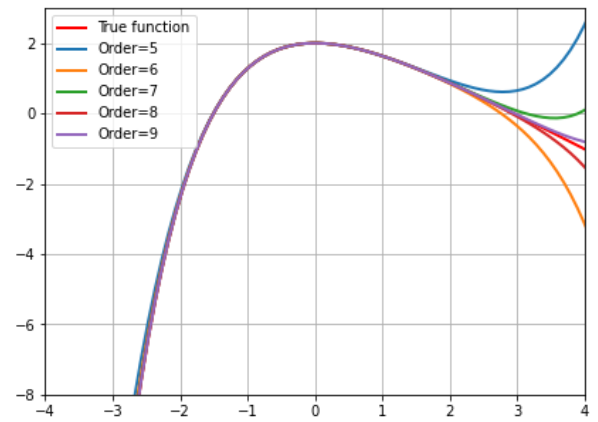
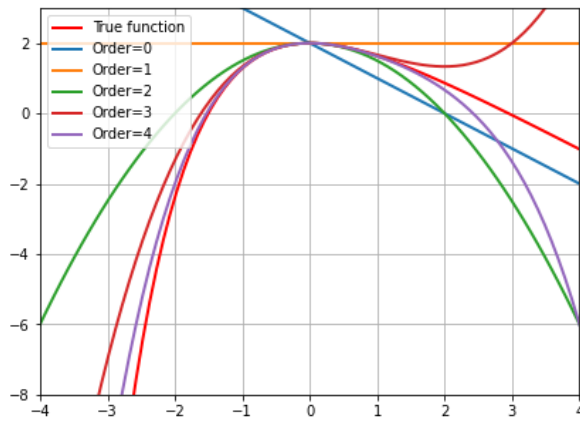
legend_lst = ['True function', 'Order=0', 'Order=1', 'Order=2', 'Order=3', 'Order=4']
for i in range(round(num_order/2)):
    ax1.plot(x, ys[i], '-',
            linewidth=2,
            markeredgewidth=2)
# legend_lst.append('Order='+str(i))

ax1.grid(True)
ax1.legend(legend_lst, loc='upper left')

legend_lst = ['True function', 'Order=5', 'Order=6', 'Order=7', 'Order=8', 'Order=9']
for i in range(round(num_order/2), num_order):
    ax2.plot(x, ys[i], '-',
            linewidth=2,
            markeredgewidth=2)
# legend_lst.append('Order='+str(i))

ax2.grid(True)
ax2.legend(legend_lst, loc='upper left')
```

Out[8]: <matplotlib.legend.Legend at 0x7fc297431af0>



From the graph, we can conclude that Taylor series with order up till 9 has a good approximation of $p(x)$.

```
In [9]: import warnings
warnings.filterwarnings('ignore', 'The iteration is not making good progress')
from scipy.optimize import fsolve

def func(x):
    return r -1-np.power(x,2)/2+ np.power(x, 3)/6-\
    np.power(x, 4)/24+np.power(x, 5)/120\
    - np.power(-x,6)/math.factorial(6)\
    - np.power(-x,7)/math.factorial(7)\
    - np.power(-x,8)/math.factorial(8)\
    - np.power(-x,9)/math.factorial(9)
```

```
In [10]: import json
import os.path
rs = np.linspace(1.001, 3, 200)
plt.figure(figsize=(15,10))
sols = []
sols_up=[]
sols_down=[]
# define initial domain of search!
rootsearch = [-3, 3]

if not os.path.isfile('solution4_9.js'): # if solution file not exists
    print('>>No solution file found, calculating solutions.')
    print('>>Takes a long time...')
    for i, r in enumerate(rs):
        print ('Handling r = %f, %d more to go.' % (r,rs.size - i -1))
        sol_up = 0
        sol_down = 0
        # call function for calculating roots
        sol_up = fsolve(func, rootsearch[0]).item()
        sol_down = fsolve(func, rootsearch[1]).item()
        sols_up.append(sol_up)
        sols_down.append(sol_down)

    sols = [sols_up, sols_down]
    # open output file for writing
    with open('solution4_9.js', 'w') as filehandle:
        json.dump(sols, filehandle)
    plt.plot(rs , sols_up, '-', color='red',
             linewidth=2,
             markeredgecolor='red',
             markeredgewidth=2)
    plt.plot(rs, sols_down, '-', color='red',
             linewidth=2,
             markeredgecolor='red',
             markeredgewidth=2)
else:
    print('Skip calculating! Loading solution file.')
    # open output file for reading
    with open('solution4_9.js', 'r') as filehandle:
```



```

sols = json.load(filehandle)
[sols_up, sols_down] = sols
plt.plot((1, 1), (sols_up[0], sols_down[0]), '-', color='red',
         linewidth=2,
         markeredgecolor='red',
         markeredgewidth=2)
plt.plot(rs, sols_up, '-', color='red',
         linewidth=2,
         markeredgecolor='red',
         markeredgewidth=2)
plt.plot(rs, sols_down, '-', color='red',
         linewidth=2,
         markeredgecolor='red',
         markeredgewidth=2)
plt.xlabel('r', size='xx-large')
plt.ylabel('$x_{eq}$', size='xx-large')
plt.xlim([0, 3])
plt.grid(True)
#
if os.path.isfile('solution4.js'): # if solution file not exists
    with open('solution4.js', 'r') as filehandle:
        sols = json.load(filehandle)
    arr = np.array(sols[100:300])
    arr = np.transpose(arr)
    sols = arr.tolist()
    rs = np.linspace(1, 3, 200)

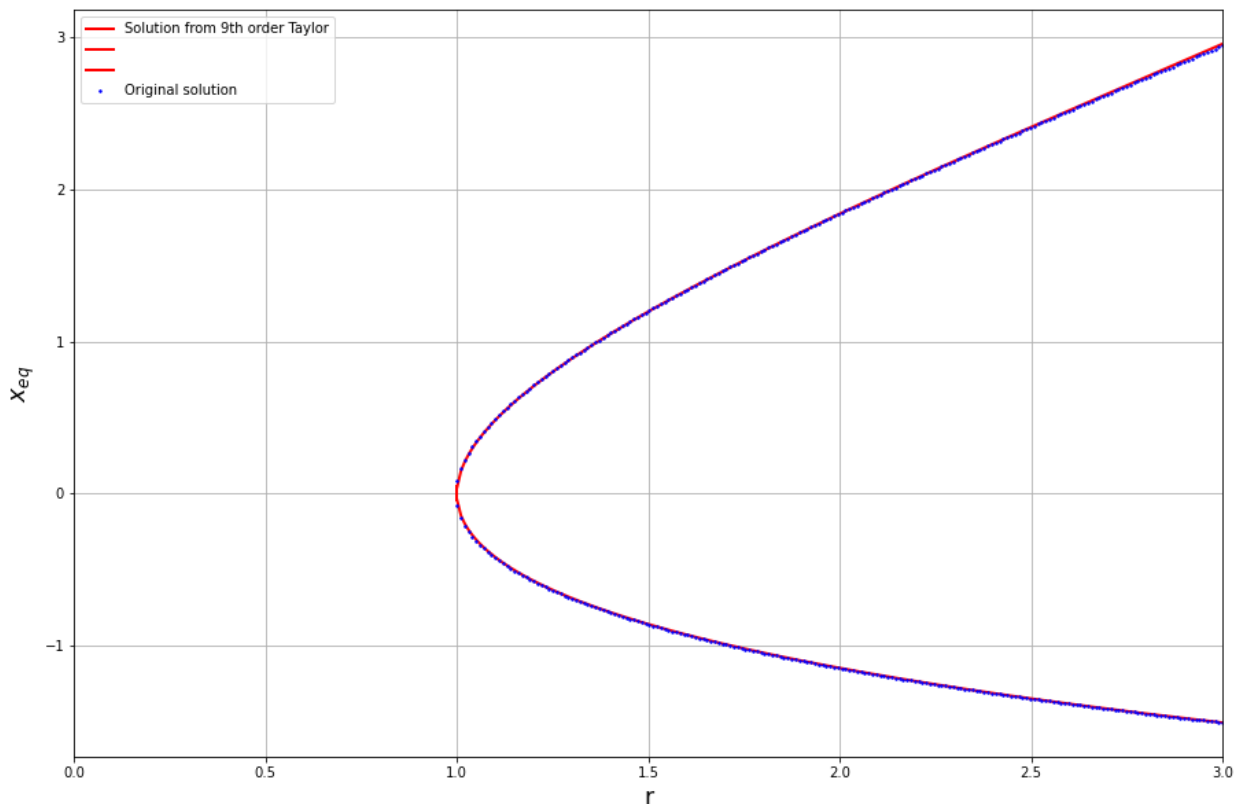
    plt.plot(rs, sols[0], '.', color='blue', markersize=3)
    plt.plot(rs, sols[1], '.', color='blue', markersize=3)

plt.legend(['Solution from 9th order Taylor', '', 'Original solution'])

```

Skip calculating! Loading solution file.

Out[10]: <matplotlib.legend.Legend at 0x7fc298174b20>



The bifurcation graph from original function and polynomial are almost colinear, we can conclude that the solutions of 9th order Taylor series is a very good approximation $p(x) = r - x - e^{-x}$ in range $x \in [-2, 3]$.

Question 4.b (10pts) Classify the equilibrium points

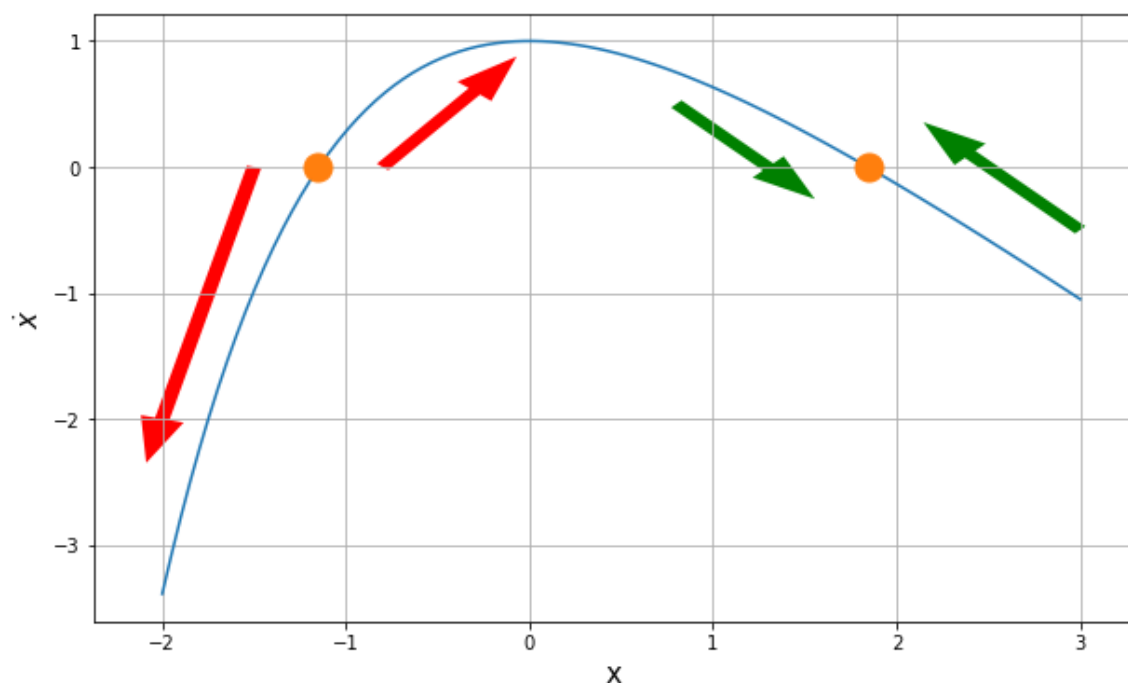
according to stability and explain your choice.

Answers:

If we choose $r = 2$ and write a small piece of code to visualize $f(x) = r - x - e^{-x}$, we can find out why two roots shows up:

```
In [11]: x = np.linspace(-2,3,100)
y = 2 - x - np.exp(-x)
plt.figure(figsize=[10,6])
plt.plot(x,y)
plt.plot([-1.15, 1.85],[0,0], 'o', markersize=15)
plt.grid(True)
plt.xlabel('x', fontsize=15); plt.ylabel('$\dot{x}$', fontsize=15)
plt.arrow(x=-1.5, y=0, dx=-0.5, dy=-2, width=.08, facecolor='red', edgecolor='none')
plt.arrow(x=-0.8, y=0, dx=0.5, dy=0.6, width=.08, facecolor='red', edgecolor='none')
plt.arrow(x=3, y=-0.5, dx=-0.6, dy=0.6, width=.08, facecolor='green', edgecolor='none')
plt.arrow(x=0.8, y=0.5, dx=0.5, dy=-0.5, width=.08, facecolor='green', edgecolor='none')
```

Out[11]: <matplotlib.patches.FancyArrow at 0x7fc29804da30>



There are two roots when $r = 2$ one is negative and the other is positive. The negative one is unstable because $f'(\chi_{eq}(-)) > 0$, if $x > \chi_{eq}(-)$, x will further increase since $\dot{x} > 0$.

The positive one is stable because $f'(\chi_{eq}(+)) < 0$, if $x > \chi_{eq}(+)$, x will go back to $\chi_{eq}(+)$ since $\dot{x} < 0$.

The above statements always hold when $r > 1$. because $f'(0) = 0$, the function $f(x) = r - x - e^{-x}$ always 'turns' at $x = 0$. So we can further conclude that when $r > 1$, the negative χ_{eq} is **UNSTABLE** and the positive χ_{eq} is **STABLE**