Analysis of Dynamic Pull-in for a Graphene-based MEMS Model

Capstone Project by Daniyar Omarov

Supervisor: Dr. Piotr Skrzypacz, Second Reader: Dr. Dongming Wei

Abstract

A novel procedure based on the Sturm's theorem for real-valued polynomials is developed to predict and identify periodic solutions and non-periodic solutions in the pull-in analysis of a graphene-based MEMS lumped parameter model with general initial conditions. It is demonstrated that under specific conditions on the lumped parameters and the initial conditions, the model has certain periodic solutions and otherwise there is no such solutions. This theoretical procedure is made practical by numerical implementations with Python scripts to verify the predicted behaviour of the periodic solutions. Numerical simulations are performed with sample data to justify by this procedure the analytically predicted existence of periodic solutions. Also, Low Order Fourier Approximation is used to find the solution for the linear spring case. Comparison with the highly accurate Runge-Kutta method is done to verify derived values from the new numerical approximation.

Keywords: MEMS; graphene; pull-in; periodic solutions; singularity; Sturm's theorem; existence of solutions; Fourier Approximation; Composite Simpson Rule

1 Introduction

In the Micro-Electro-Mechanical Systems (MEMS) we often observe the pull-in phenomenon. Therefore, the analysis of pull-in voltage of such devices is very crucial for the right calibration and use of the MEMS devices. Numerous research results have been already obtained about pull-in conditions. The analysis of pull-in voltage of linear materials for MEMS have been throughly discussed in [1]. The preliminary results for the pull-in voltage in graphene-based MEMS device have been stated in [2]. Also, the stability analysis has been deeply investigated in [3]. Exact conditions for pull-in was discussed in [4]. However, the shortcoming of aforementioned paper is that it provided condition for mass lumped parameters only in the case of zero initial conditions.

The purpose of this work is to derive pull-in conditions for graphene material with more general initial conditions and new numerical approximation. In Section 2, brief description of general model will be given, Section 3 will present Sturm's theorem with its proof, in Section 4, application of Sturm's theorem for our model will be demonstrated. Section 5 will illustrate new low order numerical approximation for the linear case and Section 6 will draw the conclusions.

2 The Model Problem

We consider the following model equation for MEMS made of graphene:

$$m\frac{d^2x}{dt^2} + EA_c\frac{x}{L} - DA_c|\frac{x}{L}|\frac{x}{L} = \frac{\varepsilon_0 A V_{DC}^2}{2(d-x)^2}, \tag{1}$$

where m is mass of the flat plate, $x(t)$ the axial displacement, E the Young's modulus, A_c cross-sectional area of graphene sheet, L length of graphene sheet, D third-order elastic stiffness constant, ε_0 is electric emissivity, A is area of plate, V_{DC} is applied voltage, and d is gap between the plate and the substrate. The model equation (1) describes the motion of capacitor plate, see Figure 1, and we refer to [2] for its detailed derivation.

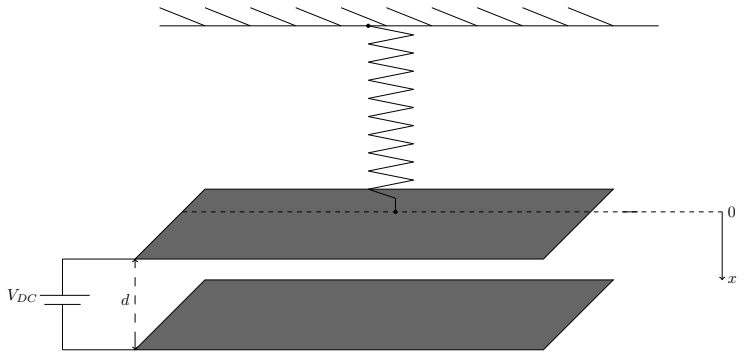

Figure 1. The parallel capacitor MEMS model.

In order to transform this equation into the dimensionless form let us introduce the following non-dimensional variables $\hat{x}$, $\hat{t}$ and the lumped parameters $K > 0$, $\alpha > 0$ are defined as

$$\hat{x} = \frac{x}{d}, \qquad \hat{t} = \frac{t}{\sqrt{\frac{mL}{EA_c}}}, \qquad \alpha = \frac{Dd}{EL}, \qquad K = \frac{\varepsilon_0 ALV_{DC}^2}{2d^3 EA_c}.$$

Hence, the model equation (1) can be written in the dimensionless form as follows

$$\frac{d^2\hat{x}}{d\hat{t}^2} + \hat{x} - \alpha|\hat{x}|\hat{x} = \frac{K}{(1-\hat{x})^2} \tag{2}$$

For the rest of the paper we will write x and t instead of $\hat{x}$ and $\hat{t}$, respectively. After multiplication of (2) by $x'(t)$ and integration with respect to time, we obtain the conservation of energy

$$\mathscr{E}'(t) = 0,$$

which implies

$$\mathscr{E}(t) = \frac{1}{2}(x'(t))^2 + \frac{1}{2}x^2 - \frac{\alpha}{3}|x|x^2 - \frac{K}{(1-x)} = C \tag{3}$$

for all $t \geq 0$, where

$$C = \frac{1}{2}(x_0')^2 + \frac{1}{2}x_0^2 - \frac{\alpha}{3}|x_0|x_0^2 - \frac{K}{(1-x_0)}.$$

Here, x_0 and x_0' are the initial conditions:

$$\begin{aligned} x(0) &= x_0, \\ x'(0) &= x_0'. \end{aligned} \tag{4}$$

Solving (3) for $x'(t)$ results in

$$x'(t) = \sqrt{-x^2 + \frac{2\alpha}{3}|x|x^2 + \frac{2K}{(1-x)} + 2C}.$$

From this expression the pull-in time can be found in terms of elliptic integral

$$t_{pull-in} = \int_{x_0}^{1} \frac{ds}{\sqrt{-s^2 + \frac{2\alpha}{3}|s|s^2 + \frac{2K}{(1-s)} + 2C}}.$$

In order to find the conditions for the existence of periodic solutions to problem (2) with initial conditions (4), we need to discuss the function

$$f(s) = -s^2 + \frac{2\alpha}{3}|s|s^2 + \frac{2K}{(1-s)} + 2C = \frac{s \cdot h(s)}{3(1-s)}, \tag{5}$$

where

$$h(s) = -2\alpha|s| * s^2 + 3s^2 + 2\alpha|s|s - 3s + \frac{6(C+K)}{s} - 6C. \tag{6}$$

Case 1. $f(s)$ is nonnegative for all $s \in [0,1]$. It means that there are no roots of $f(s)$ in interval $[0,1]$. Hence there is no periodic solution and pull-in happens.

Case 2. $f(s)$ is negative for some $s \in (0,1)$. It means that there are some roots of $f(s)$ in interval $(0,1)$. As a result, there is periodic solution and pull-in does not happen.

Therefore, it is crucial to know whether there is a root or not for different values of $\alpha > 0$ and $K \geq 0$. Sturm's theorem discussed in the next Section can be a very useful tool for solving this task.

3 Sturm's Theorem

Let $f_0(x)$ and $f_1(x)$ denote the polynomial $f(x)$ and its derivative $f'(x)$, respectively. Then, using Euclidean algorithm we define the Sturm sequence

$$\begin{aligned}
f_0(x) &= q_1(x) * f_1(x) - f_2(x), \\
f_1(x) &= q_2(x) * f_2(x) - f_3(x), \\
f_2(x) &= q_3(x) * f_3(x) - f_4(x), \\
&\vdots \\
f_{k-2}(x) &= q_{k-1}(x) * f_{k-1}(x) - f_k(x), \\
f_{k-1}(x) &= q_k(x) * f_k(x).
\end{aligned}$$

Theorem 1 (Sturm's Theorem) *The number of distinct real zeros of a polynomial $f(x)$ with real coefficients in $[a,b]$ is equal to the excess of the number of changes of sign in the sequence*

$$f_0(a), \ldots, f_{k-1}(a), f_k(a)$$

over the number of changes of the sign in the sequence

$$f_0(b), \ldots, f_{k-1}(b), f_k(b).$$

Proof. We will follow the proof from [5]. Let $\sigma(a)$ is the number of sign changes in the sequence $f_0(a),\ldots,f_{k-1}(a),f_k(a)$ and $\sigma(b)$ is the number of sign changes in the sequence $f_0(b),\ldots,f_{k-1}(b),f_k(b)$. Near any root c of $f(x)$, $f(x)$ is negative on one side of c and positive on another side of s. Hence $\sigma(x)$ can change only if it pass through a root of one of the $f_i(x)$ and $\sigma(x)$ looses one sign change. We have to study the following two cases:

Case 1. Let $f_i(x)=0, i\geq 1$: if one of the interior polynomials f_i has a root at a, then f_{i-1} and f_{i+1} are both nonzero and opposite signs. In addition, in a sufficiently small neighborhood f_{i-1} and f_{i+1} have constant signs.

Case 2. Let $f_0(x)=0$, then $f_1(x)$ has constant sign in some interval sufficiently small interval $[c,d]$ such that:

- $f_1(x)>0$: $f_1(c)<0$ and $f_1(d)>0$. Hence σ decreases by one.
- $f_1(x)<0$: $f_1(c)>0$ and $f_1(d)<0$. Hence σ decreases by one.

Thus, σ looses one sign change if and only if x passes through a root of $f_0(x)$, which is initial $f(x)$. Hence it is derived that the number of sign changes, i.e. losses in the interval $[a,b]$ counts the number of real roots of the polynomial. □

4 Periodicity of Solution

4.1 Zero Initial Conditions

In the first case initial conditions are set to zero, i.e. $x_0=x_0'=0$. As a result, $C=-K$ and (5) becomes:

$$f(s)=-s^2+\frac{2\alpha}{3}|s|s^2+\frac{2K}{(1-s)}-2K\,.$$

In order to find solution we need to analyze the condition when $f(s)=0$ for $s\in[0,1]$, that is the same as $h(s)=0$ from (6), namely

$$h(s)=-2\alpha s^3+(2\alpha+3)s^2-3s+6K\,.$$

Using Sturm's theorem and Euclidean algorithm we find the following Sturm sequence of polynomials

$$\begin{aligned}
h_0(s)&=-2\alpha s^3+(2\alpha+3)s^2-3s+6K\,,\\
h_1(s)&=-6\alpha s^2+2(2\alpha+3)s-3\,,\\
h_2(s)&=\beta s+\gamma,\\
h_3(s)&=\frac{3\beta+\gamma(2(2\alpha+3)+\frac{6\alpha\gamma}{\beta})}{\beta}\,,
\end{aligned}\tag{7}$$

where

$$\begin{aligned}
\beta&=\frac{2(18\alpha-(2\alpha+3)^2)}{18\alpha}\,,\\
\gamma&=\frac{-108\alpha K+6\alpha+9}{18\alpha}\,.
\end{aligned}$$

Table 1. Sturm algorithm.

Sturm Functions	s=0	s=1
$h_0(s)$	+	+
$h_1(s)$	-	$-2\alpha+3$
$h_2(s)$	γ	$\beta+\gamma$
$h_3(s)$	$\frac{3\beta+\gamma(2(2\alpha+3)+\frac{6\alpha\gamma}{\beta})}{\beta}$	$\frac{3\beta+\gamma(2(2\alpha+3)+\frac{6\alpha\gamma}{\beta})}{\beta}$

The number of roots of $h(s)$ in the interval $[0,1]$ is determined using Table 1. The difference between the number of sign changes in the second column and the number of sign changes in third column of Table 1 states the number of roots. Algorithm 1 in Appendix provides Python script [6] to accomplish this task computationally for different α and K values. In addition, algorithm plots numerical solution solved using Runge-Kutta method (see [7]) for particular $\alpha > 0$ and $K > 0$ values in order to verify derived conclusion. Figure 2, 3 and 4 illustrate examples of periodic and non-periodic solutions.

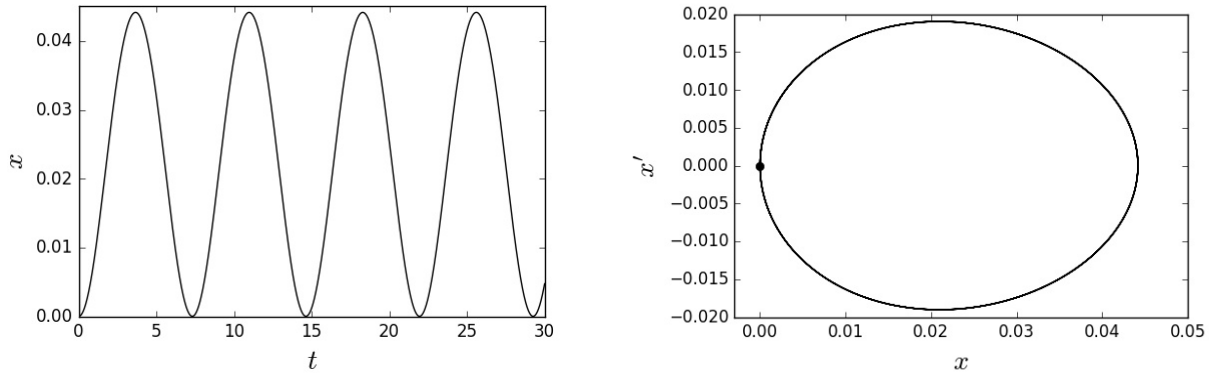

Figure 2. Periodic solution $x(t)$ and its phase portrait for $\alpha = 5$ and $K = 0.018$.

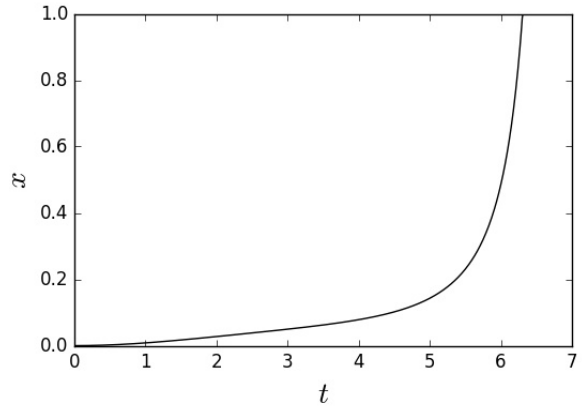

Figure 3. Pull-in solution $x(t)$ for $\alpha = 14$ and $K = 0.018$.

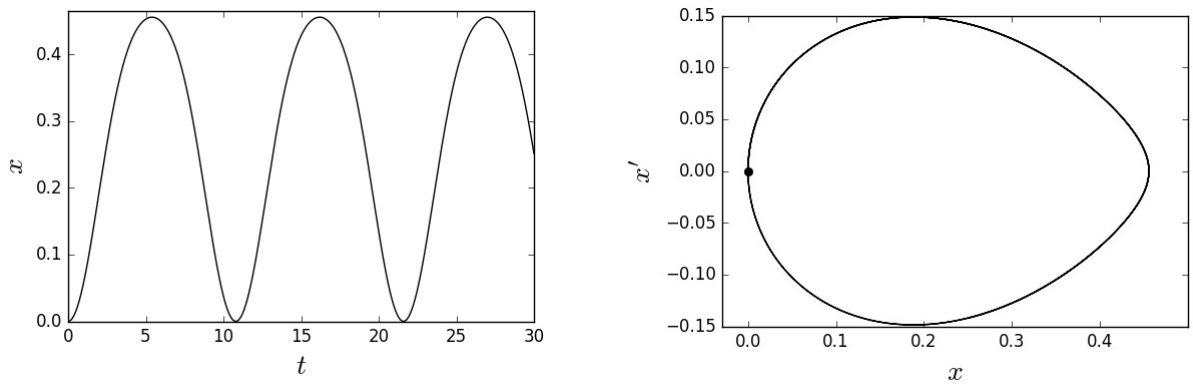

Figure 4. Periodic solution $x(t)$ and its phase portrait for $\alpha = 0.001$ and $K = 0.124$.

4.2 Non-zero Initial Conditions

In the case of non-zero initial conditions, checking existence of root in the $(0,1)$ interval is not enough. We will need to perform two more additional steps.

Firstly, we will check existence of root in the interval $(0,1)$ using Sturm table as it was done for zero initial conditions. If there is no root, then there is pull-in. If there is a root, then we proceed to the second step.

The second step is to check the sign of the $g(s)$ function at $s = 0$ from the Algorithm 2. from Appendix. If the sign is negative, then there is a pull-in and we need to stop the analysis. However, if the sign is positive, then we proceed to the last step.

The third step is to check the existence of the root in the interval $(-\infty, 0)$ using Sturm Algorithm. If there is no root, then there is pull-in. If there is a root, then we conclude that there is periodic solution.

However, due to nonzero initial conditions, polynomials become highly complicated. Hence, it is hard to construct Sturm tables manually. Algorithm 2. from Appendix provides reader with Python code [6] which performs the same analysis in the case of nonzero initial conditions for the given α, K, x_0, and x_0' values. For example, Figure 5, 6 and 7 demonstrate the periodic and pull-in solutions for various parameters.

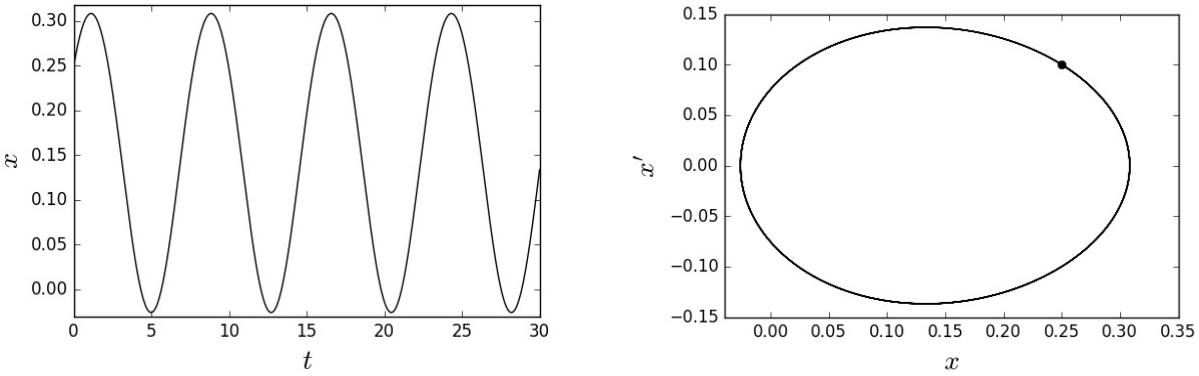

Figure 5. Periodic solution $x(t)$ and its phase portrait for $\alpha = 0.01$, $K = 0.1$, $x_0 = 0.25$, and $x_0' = 0.1$.

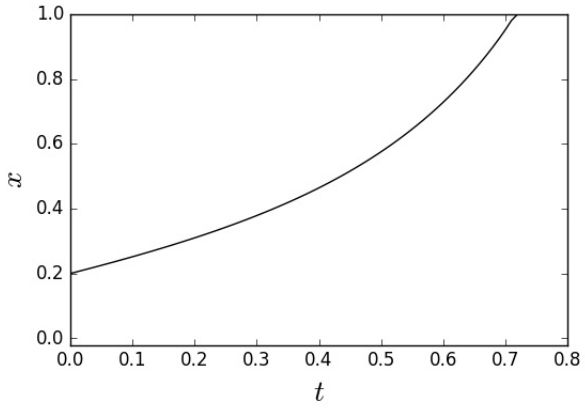

Figure 6. Pull-in solution $x(t)$ for $\alpha = 14$, $K = 0.018$, $x_0 = 0.2$, and $x_0' = 0.5$.

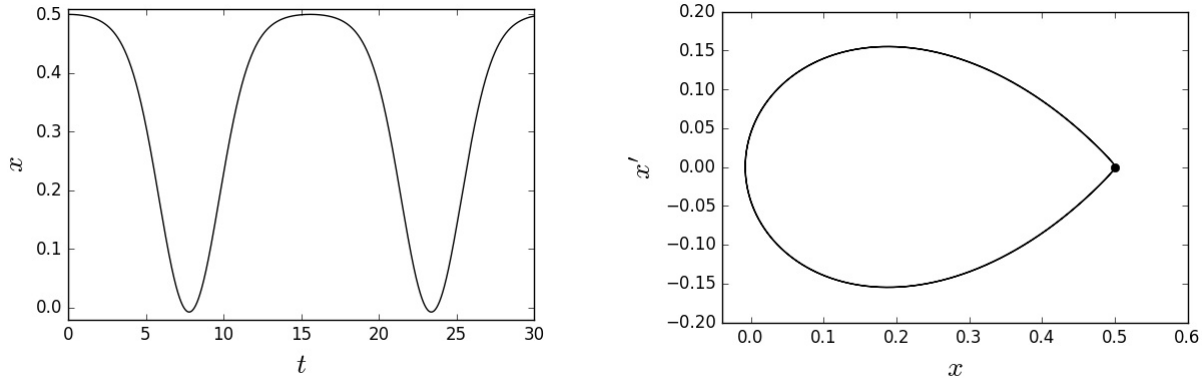

Figure 7. Periodic solution $x(t)$ and its phase portrait for $\alpha = 0.001$, $K = 0.124$, $x_0 = 0.5$, and $x_0' = 0$.

5 Numerical Approximation of Solution

In previous sections we demonstrated that model equation (2) can have periodic solution for particular choices of α and K. Hence this section will be dedicated to finding numerical approximation to that periodic solution. However, our analysis will be for Linear Spring Case (i.e. $\alpha = 0$) and Zero Initial Conditions (i.e. $x(0) = 0$ and $x'(0) = 0$).

$$\frac{d^2x}{dt^2} + x = \frac{K}{(1-x)^2}. \tag{8}$$

5.1 Low Order Fourier Approximation

For numerical approximation of solution will be used Fourier Series [8]:

$$f(x) = a_0 + \sum_{n=1}^{\infty} a_n \cos(nx) + \sum_{n=1}^{\infty} b_n \sin(nx).$$

For our analysis we will use only the first order Fourier approximation. Hence our approximation of solution will be in the following form:

$$x(t) = \frac{a_0}{2} + a_1 \cos(\omega t) + b_1 \sin(\omega t), \tag{9}$$

where a_0 is the amplitude of numerical solution and ω is angular frequency.
Firstly, we will find the values of a_1 and b_1 using the initial conditions:

$$a_1 = -\frac{a_0}{2},$$
$$b_1 = 0.$$

After substitution approximate solution becomes:

$$x(t) = \frac{a_0}{2}(1 - \cos(\omega t)) \tag{10}$$

Secondly, we will find the value of amplitude a_0, i.e. maximum value of $x(t)$. From Calculus we know that derivative of $x(t)$ (i.e. $x'(t) = 0$) must be zero at that point. Hence energy equation (3) can be rewritten as:

$$\frac{1}{2}x^2 - \frac{K}{(1-x)} = -K$$

After solving this equation we derive two roots:

$$a_0 = \frac{1 \pm \sqrt{1-8K}}{2}$$

We take the root with minus sign in order to get minimum energy. After finding the amplitude a_0, the only unknown variable in the equation (10) is angular frequency.

5.2 Finding the Period

We will find the angular frequency from Eq. (10) through finding period T of the approximate solution due to the relation:

$$\omega = \frac{2\pi}{T}$$

From the Eq. (3) we can derive the analytic value of the period:

$$\frac{T}{2} = \int_0^{a_0} \frac{ds}{\sqrt{-s^2 + \frac{2K}{1-s} - 2K}} = \int_0^{a_0} \sqrt{\frac{1-s}{s}} \frac{ds}{\sqrt{s^2 - s + 2K}} \tag{11}$$

This is Elliptic Integral [9] and has highly complicated exact solution. Hence we will approximate it using the Composite Simpson Rule [10].

Definition 1 (Composite Simpson Rule)

$$\int_a^b f(x)dx \approx \frac{b-a}{3n}[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \ldots + 4f(x_{n-1}) + f(x_n))]$$

where n is even number of subintervals, lower limit $a = x_0$, upper limit $b = x_n$.
For our case we will use $n = 4$ subintervals:

$$\int_a^b f(x)dx \approx \frac{b-a}{12}[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + f(x_4))] \tag{12}$$

However, Eq. (11) have singularities at the upper and lower limits, hence we cannot use Composite Simpson Rule directly. We will transform our integral (11) such that we will be able to use the rule. Firstly, we will divide the integral in two integrals:

$$\int_0^{a_0} \sqrt{\frac{1-s}{s}} \frac{ds}{\sqrt{s^2-s+2K}} = \int_0^{\frac{a_0}{2}} \sqrt{\frac{1-s}{s}} \frac{ds}{\sqrt{s^2-s+2K}} + \int_{\frac{a_0}{2}}^{a_0} \sqrt{\frac{1-s}{s}} \frac{ds}{\sqrt{s^2-s+2K}} = I_1 + I_2 \quad (13)$$

Secondly we will apply *Integration by Parts* method for each of integral from Eq. (13):

$$\begin{aligned} I_1 &= \int_0^{\frac{a_0}{2}} k(s) * l'(s) ds = k(s) * l(s)|_0^{\frac{a_0}{2}} - \int_0^{\frac{a_0}{2}} k'(s) * l(s) ds \\ I_2 &= \int_{\frac{a_0}{2}}^{a_0} m(s) * n'(s) ds = m(s) * n(s)|_{\frac{a_0}{2}}^{a_0} - \int_{\frac{a_0}{2}}^{a_0} m'(s) * n(s) ds \end{aligned} \quad (14)$$

where

$$\begin{aligned} k(s) = n'(s) &= \frac{1}{\sqrt{s^2-s+2K}} \\ l'(s) = m(s) &= \sqrt{\frac{1-s}{s}} \end{aligned}$$

Since after applying Integration by Parts we get rid of the the singularities at the lower and upper limits, we can apply Composite Simpson Rule for the integrals in the Eq. (14).

$$\begin{aligned} \int_0^{\frac{a_0}{2}} k'(s) * l(s) ds &= \frac{a_0}{24} [k'(0) * l(0) + 4k'(\frac{a_0}{8}) * l(\frac{a_0}{8}) \\ &\quad + 2k'(\frac{a_0}{4}) * l(\frac{a_0}{4}) + 4k'(\frac{3a_0}{8}) * l(\frac{3a_0}{8}) + k'(\frac{a_0}{2}) * l(\frac{a_0}{2})] \\ \int_{\frac{a_0}{2}}^{a_0} m'(s) * n(s) ds &= \frac{a_0}{24} [m'(\frac{a_0}{2}) * n(\frac{a_0}{2}) + 4m'(\frac{5a_0}{8}) * m(\frac{5a_0}{8}) \\ &\quad + 2m'(\frac{3a_0}{4}) * n(\frac{3a_0}{4}) + 4m'(\frac{7a_0}{8}) * n(\frac{7a_0}{8}) + m'(a_0) * n(a_0)] \end{aligned} \quad (15)$$

Finally, we substitute Eq. (15) in Eq. (14) and derive approximation for the angular frequency ω.

$$\omega = \frac{\pi}{\frac{T}{2}} = \frac{\pi}{I_1 + I_2}$$

5.3 Checking Goodness of Approximation

After finding the approximation for the angular frequency we will substitute it in the Eq. (10) and plot the our numerical approximation versus numerical solution solved using Runge-Kutta method (see [7]). As it can be seen from the Figure 8 Low Order Fourier Approximation have almost the same values as the highly accurate Runge-Kutta approximation. For more detailed information we can look at the absolute value of the error term. Figure 9 demonstrates that our approximation is very accurate for the first two periods (10^{-3}). However, as the time passes Fourier solution becomes worse. Hence we conclude that derived approximation is the most accurate for small time periods.

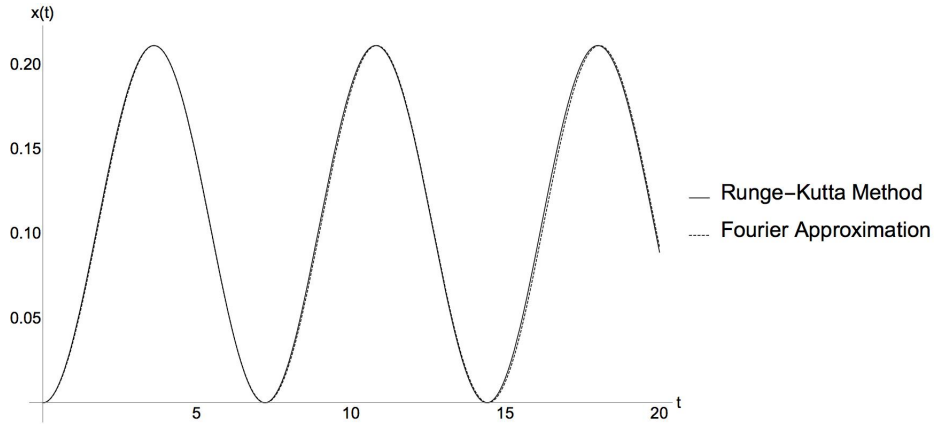

Figure 8. Low Order Fourier Approximation

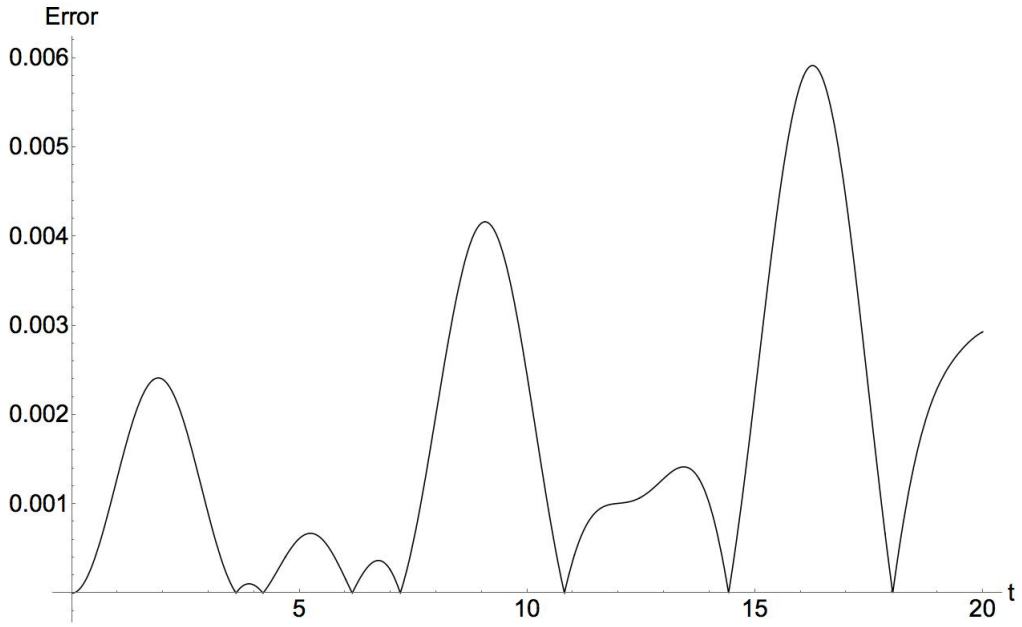

Figure 9. Absolute Value of the Error Term

6 Conclusions

We developed a simple and effective procedure to identify a priori the periodic and pull-in solutions to the MEMS problem of parallel plate capacitor. It was shown that the periodicity of solution depends on the lumped parameters α, K, and the initial conditions. The procedure is supplemented numerically by using Python codes. In general, this technique is useful for determination of the periodicity of solution to higher order differential equations and it can be applicable for analogous singular equations of dynamical systems.

In the second half of the paper we developed new approximation for the solution of the differential equation (8). We demonstrated that even Low Order (only first term of Fourier Series is used) Fourier Approximation gives the highly accurate values especially for small time intervals. Further research can be done to establish other analytical or numerical methods in order to determine solutions of the nonlinear problem (i.e. α is not zero). Another research area is to find the solution for nonzero initial conditions.

References

[1] Younis, Mohammad I. Mems linear and nonlinear statics and dynamics. Springer-Verlag New York, 2014.

[2] Wei, Dongming, S. Kadyrov, and Z. Kazbek. “Periodic Solutions of a Graphene Based Model in Micro-Electro-Mechanical Pull-in Device.” *Applied and Computational Mechanics* 11 (2017), pp.81-90.

[3] Zhang, Wen-Ming, Han Yan, Zhi-Ke Peng, and Guang Meng. “Electrostatic pull-in instability in MEMS/NEMS: A review.” Sensors and Actuators A: Physical 214 (2014): 187-218. doi:10.1016/j.sna.2014.04.025.

[4] Skrzypacz, Piotr, S. Kadyrov, D. Nurakhmetov, and D. Wei. “Analysis of Dynamic Pull-in Voltage of a Nonlinear Material NEMS Model.” accepted for publication in *Materials Today: Proceedings*.

[5] Prasolov, Viktor Vasil'evic. *Polynomials*. Berlin: Springer, 2004.

[6] Python Software Foundation. Python Language Reference, version 2.7. Available at http://www.python.org

[7] Landau, Rubin H., Manuel J. Paez, and Christian C. Bordeianu. *Computational physics: problem solving with Python*. Weinheim, Germany: Wiley-VCH, 2015.

[8] Wolfram MathWorld. *Fourier Series.* Accessed March 10, 2018. http://mathworld.wolfram.com/FourierSeries.html.

[9] Wolfram MathWorld. *Elliptic Integral.* Accessed March 10, 2018. http://mathworld.wolfram.com/EllipticIntegral.html.

[10] Süli, Endre, and David Francis. Mayers. *An introduction to numerical analysis.* Cambridge: Cambridge University Press, 2012.

Appendix

Algorithm 1 Python Script for Sturm Algorithm: Zero Initial Conditions.

```
import scipy.integrate as integrate; import numpy as np
import matplotlib.pyplot as plt; import matplotlib; %matplotlib inline

# Defining Sturm Functions
def beta(alpha, k): return 2*(18*alpha - (2*alpha + 3)**2)/(18*alpha)

def gamma(alpha, k): return (-108*alpha*k + 6*alpha + 9)/(18*alpha)

def func1(alpha,k): return (3*beta(alpha, k)+gamma(alpha, k)*(4*alpha+6+6*alpha*gamma(alpha,k)/beta(alpha,k)))/beta(alpha,k)

def func2(alpha, k): return (beta(alpha, k) + gamma(alpha, k))

# Defining Sturm Table
def zero_matrix(alpha, k):
    column0 = np.zeros(4, int); column0[0] = 1; column1 = np.zeros(4, int); column0[1] = 0;

    if gamma(alpha, k) > 0: column0[2] = 1
    else: column0[2] = 0

    if func1(alpha, k) > 0: column0[3] = 1; column1[3] = 1
    else: column0[3] = 0; column1[3] = 0

    column1[0] = 1

    if (-2*alpha + 3) > 0: column1[1] = 1
    else: column1[1] = 0

    if func2(alpha, k) > 0: column1[2] = 1
    else: column1[2] = 0

    counter1 = 0; counter2 = 0;

    for i in range(3):

        if column0[i] != column0[i+1]: counter1 = counter1 + 1
        if column1[i] != column1[i+1]: counter2 = counter2 + 1

    if counter1 - counter2 != 0: print("There is a periodic solution.")
    else: print("There is a pull-in.")

# Plotting for Verification
def model1(alpha, k):
    def model(x,t):
        y = x[0]; dy = x[1]; xdot = [[],[]]; xdot[0] = dy
        xdot[1] = -y + alpha*abs(y)*y + k/((1 - y)**2)
        return xdot
    time = np.linspace(0,10,1000); z1 = integrate.odeint(model,[0, 0],time)
    return z1;

def model2(alpha, k):
    z2 = np.zeros((1000,2),float); counter = 0;
    for i in range(0,1000):
        if model1(alpha, k)[i][0] > 1: break;
        z2[i][:] = model1(alpha, k)[i][:]; counter = counter + 1;
    z3 = np.zeros((counter,2), float);
    for j in range(0, counter): z3[j][:] = z2[j][:];
    return z3, counter

def plot1(alpha, k):
    z4, length = model2(alpha,k); time = np.linspace(0,length/100,length); plt.plot(time, z4[:,0],'r-');
    plt.ylabel('$x$', fontsize=25); plt.xlabel("$t$", fontsize=25); fig = matplotlib.pyplot.gcf(); fig.set_size_inches(6, 4);
    plt.xticks(fontsize = 12); plt.yticks(fontsize = 12); plt.ylim(ymin=0,ymax=(np.max(z4)+0.005));
    plt.savefig('1.jpg', dpi=100, bbox_inches = 'tight'); plt.show();

def plot2(alpha, k):
    z4, length = model2(alpha,k); plt.plot(z4[:,0],z4[:,1],'g-'); plt.ylabel('$x\'$', fontsize=25); plt.xlabel("$x$", fontsize=25);
    fig = matplotlib.pyplot.gcf(); fig.set_size_inches(6, 4); plt.xticks(fontsize = 12); plt.yticks(fontsize = 12);
    plt.plot(0,0,'ro'); plt.xlim(xmin=-0.005); plt.savefig('2.jpg', dpi=100, bbox_inches = 'tight'); plt.show();
```

Algorithm 2 Python Script for Sturm Algorithm: Non-zero Initial Conditions.

```
import scipy.integrate as integrate; import numpy as np
import matplotlib.pyplot as plt; import matplotlib %matplotlib inline

# Defining Sturm Functions 1
def C_value(alpha, k, x0, x1):
    return (x1**2)/2 + (x0**2)/2 - (alpha*x0**2)*abs(x0)/3 - k/(1 - x0)
def g0(alpha, k, x0, x1, s):
    return (6*C_value(alpha, k, x0, x1) + 6*k - 6*C_value(alpha, k, x0, x1)*s - 3*s**2 + (3 + 2*alpha)*s**3 - 2*alpha*s**4)
def g1(alpha, k, x0, x1, s):
    return (-6*C_value(alpha, k, x0, x1) - 6*s + 3*(3 + 2*alpha)*s**2 - 8*alpha*s**3)
def g2(alpha, k, x0, x1, s):
    return (- (3*(-6*C_value(alpha, k, x0, x1) + 60*alpha*C_value(alpha, k, x0, x1) + 64*alpha*k))/(32*alpha) - (3*(-6 -
            4*alpha - 48*alpha*C_value(alpha, k, x0, x1))*s)/(32*alpha) - (3*(9 - 4*alpha + 4*alpha**2)*s**2)/(32*alpha))
def g3(alpha, k, x0, x1, s):
    return (-(64*alpha*(-72*C_value(alpha, k, x0, x1) + 18*alpha*C_value(alpha, k, x0, x1) + 24*alpha**2*
            C_value(alpha, k, x0, x1) - 24*alpha**3*C_value(alpha, k, x0, x1) - 36*alpha*C_value(alpha, k, x0, x1)**2 +
            360*alpha**2*C_value(alpha, k, x0, x1)**2 - 81*k + 30*alpha*k + 20*alpha**2*k - 24*alpha**3*k + 384*alpha**2*
            C_value(alpha, k, x0, x1)*k))/(9 - 4*alpha + 4*alpha**2)**2 - (64*alpha*((3 - 2*alpha)**2 + 54*
            C_value(alpha, k, x0, x1) + 12*alpha*C_value(alpha, k, x0, x1) - 72*alpha**2*C_value(alpha, k, x0, x1) +
            48*alpha**3*C_value(alpha, k, x0, x1) - 288*alpha**2*C_value(alpha, k, x0, x1)**2 + 72*alpha*k -
            32*alpha**2*k + 32*alpha**3*k)*s)/(9 - 4*alpha + 4*alpha**2)**2)
def g4(alpha, k, x0, x1, s):
    return (3*(9 - 4*alpha + 4*alpha**2)**2*(216*(-1 - 6*alpha**2 + 4*alpha**3)*C_value(alpha, k, x0, x1)**3 +
            1296*alpha**2*C_value(alpha, k, x0, x1)**4 + 36*C_value(alpha, k, x0, x1)**2*(6 + 4*alpha**4 +
            alpha**2*(9 - 168*k) - 2*alpha*(2 + 3*k) + 12*alpha**3*(-1 + 14*k)) + k*(-54 + 24*alpha**2*(-1 + k) +
            729*k + 144*alpha**4*k - 72*alpha*(-1 + 9*k) + 32*alpha**3*k*(-9 + 128*k)) + 6*C_value(alpha, k, x0, x1)*
            (-9 + 162*k + 48*alpha**4*k - 6*alpha*(-2 + 23*k) + 24*alpha**3*k*(-5 + 64*k) + alpha**2*(-4 + 60*k -
            768*k**2))))/(32*alpha*(9 + 54*C_value(alpha, k, x0, x1) + 16*alpha**3*(3*C_value(alpha, k, x0, x1) +
            2*k) + 12*alpha*(-1 + C_value(alpha, k, x0, x1) + 6*k) - 4*alpha**2*(-1 + 18*C_value(alpha, k, x0, x1) +
            72*C_value(alpha, k, x0, x1)**2 + 8*k))**2)

# Defining Sturm Functions 2
def h0(alpha, k, x0, x1, s):
    return (6*C_value(alpha, k, x0, x1) + 6*k - 6*C_value(alpha, k, x0, x1)*s - 3*s**2 + (3 - 2*alpha)*s**3 + 2*alpha*s**4)
def h1(alpha, k, x0, x1, s):
    return (-6*C_value(alpha, k, x0, x1) - 6*s + 3*(3 - 2*alpha)*s**2 + 8*alpha*s**3)
def h2(alpha, k, x0, x1, s):
    return ((3*(-6*C_value(alpha, k, x0, x1) - 60*alpha*C_value(alpha, k, x0, x1) - 64*alpha*k))/(32*alpha) + (3*(-6 +
            4*alpha + 48*alpha*C_value(alpha, k, x0, x1))*s)/(32*alpha) + (3*(9 + 4*alpha + 4*alpha**2)*s**2)/(32*alpha))
def h3(alpha, k, x0, x1, s):
    return -(1/((9 + 4*alpha + 4*alpha**2)**2))*64*alpha*(72*C_value(alpha, k, x0, x1) + 18*alpha*C_value(alpha, k, x0, x1) -
            24*alpha**2*C_value(alpha, k, x0, x1) - 24*alpha**3*C_value(alpha, k, x0, x1) - 36*alpha*
            C_value(alpha, k, x0, x1)**2 - 360*alpha**2*C_value(alpha, k, x0, x1)**2 + 81*k + 30*alpha*k - 20*alpha**2*k -
            24*alpha**3*k - 384*alpha**2*C_value(alpha, k, x0, x1)*k) - (1/((9 + 4*alpha + 4*alpha**2)**2))*64*alpha*(-(3
            + 2*alpha)**2 - 54*C_value(alpha, k, x0, x1) + 12*alpha*C_value(alpha, k, x0, x1) + 72*alpha**2*
            C_value(alpha, k, x0, x1) + 48*alpha**3*C_value(alpha, k, x0, x1) + 288*alpha**2*C_value(alpha, k, x0, x1)**2 +
            72*alpha*k + 32*alpha**2*k + 32*alpha**3*k)*s
def h4(alpha, k, x0, x1, s):
    return -((3*(9 + 4*alpha + 4*alpha**2)**2*(-216*(1 + 6*alpha**2 + 4*alpha**3)*C_value(alpha, k, x0, x1)**3 +
            1296*alpha**2*C_value(alpha, k, x0, x1)**4 + 36*C_value(alpha, k, x0, x1)**2*(6 + 4*alpha**4 + alpha**2*
            (9 - 168*k) + alpha**3*(12 - 168*k) + alpha*(4 + 6*k)) + k*(-54 + 24*alpha**2*(-1 + k) + 729*k +
            144*alpha**4*k + 72*alpha*(-1 + 9*k) - 32*alpha**3*k*(-9 + 128*k)) + 6*C_value(alpha, k, x0, x1)*
            (-9 + 162*k + 48*alpha**4*k + 6*alpha*(-2 + 23*k) - 24*alpha**3*k*(-5 + 64*k) + alpha**2*(-4 + 60*k -
            768*k**2))))/(32*alpha*(-9*(1 + 6*C_value(alpha, k, x0, x1)) + 16*alpha**3*(3*C_value(alpha, k, x0, x1) +
            2*k) + 12*alpha*(-1 + C_value(alpha, k, x0, x1) + 6*k) + 4*alpha**2*(-1 + 18*C_value(alpha, k, x0, x1) +
            72*C_value(alpha, k, x0, x1)**2 + 8*k))**2))

# Defining Sturm Table 1
def check1(alpha, k, x0, x1):
    column0 = np.zeros(5, int); column1 = np.zeros(5, int)

    if g0(alpha, k, x0, x1, 0) > 0: column0[0] = 1;
    else: column0[0] = 0;
    if g1(alpha, k, x0, x1, 0) > 0: column0[1] = 1;
    else: column0[1] = 0;
    if g2(alpha, k, x0, x1, 0) > 0: column0[2] = 1;
    else: column0[2] = 0;
    if g3(alpha, k, x0, x1, 0) > 0: column0[3] = 1;
    else: column0[3] = 0;
    if g4(alpha, k, x0, x1, 0) > 0: column0[4] = 1;
    else: column0[4] = 0;
    if g0(alpha, k, x0, x1, 1) > 0: column1[0] = 1;
    else: column1[0] = 0;
    if g1(alpha, k, x0, x1, 1) > 0: column1[1] = 1;
    else: column1[1] = 0;
    if g2(alpha, k, x0, x1, 1) > 0: column1[2] = 1;
    else: column1[2] = 0;
    if g3(alpha, k, x0, x1, 1) > 0: column1[3] = 1;
    else: column1[3] = 0;
    if g4(alpha, k, x0, x1, 1) > 0: column1[4] = 1;
    else: column1[4] = 0;

    counter1 = 0; counter2 = 0;
    for i in range(4):
        if column0[i] != column0[i+1]: counter1 = counter1 + 1;
        if column1[i] != column1[i+1]: counter2 = counter2 + 1;

    if counter1 - counter2 == 0:
        print("There is a pull-in.");
        return 0;
    else:
        if g0(alpha, k, x0, x1, 0) < 0:
            print("There is a pull-in.");
            return 0;
        else:
            if check2(alpha, k, x0, x1) == 0:
                print("There is a pull-in.");
                return 0;
            else:
                print("There is a periodic solution.");
                return 1;
```

Continued

```
# Defining Sturm Table 2
def check2(alpha, k, x0, x1):
    column0 = np.zeros(5, int); column1 = np.zeros(5, int)

    if h0(alpha, k, x0, x1, 0) > 0: column0[0] = 1;
    else: column0[0] = 0;
    if h1(alpha, k, x0, x1, 0) > 0: column0[1] = 1;
    else: column0[1] = 0;
    if h2(alpha, k, x0, x1, 0) > 0: column0[2] = 1;
    else: column0[2] = 0;
    if h3(alpha, k, x0, x1, 0) > 0: column0[3] = 1;
    else: column0[3] = 0;
    if h4(alpha, k, x0, x1, 0) > 0: column0[4] = 1;
    else: column0[4] = 0;
    if h0(alpha, k, x0, x1, float('-inf')) > 0: column1[0] = 1;
    else: column1[0] = 0;
    if h1(alpha, k, x0, x1, float('-inf')) > 0: column1[1] = 1;
    else: column1[1] = 0;
    if h2(alpha, k, x0, x1, float('-inf')) > 0: column1[2] = 1;
    else: column1[2] = 0;
    if h3(alpha, k, x0, x1, float('-inf')) > 0: column1[3] = 1;
    else: column1[3] = 0;
    if h4(alpha, k, x0, x1, float('-inf')) > 0: column1[4] = 1;
    else: column1[4] = 0;

    counter1 = 0; counter2 = 0;
    for i in range(4):
        if column0[i] != column0[i+1]: counter1 = counter1 + 1;
        if column1[i] != column1[i+1]: counter2 = counter2 + 1;

    if counter1 - counter2 != 0:
        return 1;
    else:
        return 0;

# Plotting for Verification
def model1(alpha, k, x0, x1):

    def model(x,t):
        y = x[0] ;dy = x[1]
        xdot = [[],[]]; xdot[0] = dy
        xdot[1] = -y + alpha*abs(y)*y + k/((1 - y)**2)
        return xdot

    time = np.linspace(0,30,3000);
    z1 = integrate.odeint(model,[x0, x1],time);

    return z1

def model2(alpha, k, x0, x1):

    z2 = np.zeros((3000,2),float); counter = 0;

    for i in range(0,3000):
        if model1(alpha, k, x0, x1)[i][0] > 1: break;
        z2[i][:] = model1(alpha, k, x0, x1)[i][:];
        counter = counter + 1;

    z3 = np.zeros((counter,2), float);
    for j in range(0, counter): z3[j][:] = z2[j][:];

    return z3, counter

def plotper(alpha, k, x0, x1):

    z4, length = model2(alpha, k, x0, x1);
    time = np.linspace(0,length/100,length); plt.plot(time, z4[:,0],'k-');
    plt.ylabel('$x$', fontsize=20); plt.xlabel("$t$", fontsize=20);
    fig = matplotlib.pyplot.gcf(); fig.set_size_inches(6, 4);
    plt.xticks(fontsize = 12); plt.yticks(fontsize = 12);
    plt.ylim(ymin=-0.02,ymax=(np.max(z4)+0.01));
    plt.savefig('1.jpg', dpi=100, bbox_inches = 'tight'); plt.show();

def plotnonper(alpha, k, x0, x1):

    z4, length = model2(alpha, k, x0, x1);
    time = np.linspace(0,length/100,length); plt.plot(time, z4[:,0],'k-');
    plt.ylabel('$x$', fontsize=20); plt.xlabel("$t$", fontsize=20);
    fig = matplotlib.pyplot.gcf(); fig.set_size_inches(6, 4);
    plt.xticks(fontsize = 12); plt.yticks(fontsize = 12);
    plt.ylim(ymin=-0.02,ymax=1.0);
    plt.savefig('1.jpg', dpi=100, bbox_inches = 'tight'); plt.show();

def plot1(alpha, k, x0, x1):
    if check1(alpha, k, x0, x1) == 1:
        plotper(alpha,k, x0, x1);
    else:
        plotnonper(alpha, k, x0, x1);

def plot2(alpha, k, x0, x1):

    z4, length = model2(alpha, k, x0, x1); plt.plot(z4[:,0],z4[:,1],'k-');
    plt.ylabel('$x\'$', fontsize=20); plt.xlabel("$x$", fontsize=20);
    fig = matplotlib.pyplot.gcf(); fig.set_size_inches(6, 4);
    plt.xticks(fontsize = 12); plt.yticks(fontsize = 12);
    plt.plot(x0,x1,'ko'); plt.xlim(xmin=-0.7, xmax=1); plt.ylim(ymin=-0.02,ymax=1.5);
    plt.savefig('2.jpg', dpi=100, bbox_inches = 'tight'); plt.show();
```