Applied and Computational Mechanics 12 (2018) 59–72

On the application of Sturm's theorem to analysis of dynamic pull-in for a graphene-based MEMS model

D. Omarov[a], D. Nurakhmetov[a,b], D. Wei[a,*], P. Skrzypacz[a]

[a] *School of Science and Technology, Nazarbayev University, 53 Kabanbay Batyr Ave., Astana 010000, Kazakhstan*
[b] *Department of Information System, S. Seifullin Kazakh Agrotechnical University, Zhenis Ave. 62, Astana, 010011, Kazakhstan*

Received 2 November 2017; accepted 22 May 2018

Abstract

A novel procedure based on the Sturm's theorem for real-valued polynomials is developed to predict and identify periodic and non-periodic solutions for 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 are no such solutions. This theoretical procedure is made practical by numerical implementations with Python scripts to verify the predicted behaviour of the solutions. Numerical simulations are performed with sample data to justify by this procedure the analytically predicted existence of periodic solutions.

Keywords: MEMS, graphene, pull-in, periodic solutions, singularity, Sturm's theorem, existence of solutions

1. Introduction

An important phenomenon in Micro-Electro-Mechanical Systems (MEMS) is the so called pull-in instability. For certain values of the voltage the system is in a stable operation regime in which a moving part, e.g., a charged plate, approaches a stable steady state, the so called static pull-in, and remains separate from the fixed part, e.g., a substrate. When the voltage is increased beyond a critical value (static pull-in voltage), the device is in the touch-down regime, i.e., the moving part collapses onto the fixed part. 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 first mass-spring model for an electrostatically actuated device has been introduced by Nathanson et al. [7]. The analysis of pull-in voltage of linear materials for MEMS have been throughly discussed in [8, 14]. For the mass-spring system, Zhang et al. [15] specify the dynamic pull-in and describe it as the collapse of the moving structure caused by the combination of kinetic and potential energies. In general, the dynamic pull-in requires a lower voltage to be triggered compared to the static pull-in threshold, see [2, 15]. The preliminary results for the static pull-in voltage have been stated in [12] by considering the quadratic stress-strain equation which is validated to be important for graphene with applications in MEMS. Exact conditions for the dynamic pull-in were discussed in [11]. However, the shortcoming of aforementioned paper is that it provides condition for mass lumped parameters only in the case of zero initial conditions.

The purpose of this work is to derive the conditions for the dynamic pull-in in the case of general initial conditions in the one degree of freedom (DOF) spring-mass system which

*Corresponding author. Tel.: +771 727 099 368, e-mail: dongming.wei@nu.edu.kz.
https://doi.org/10.24132/acm.2018.413

represents the graphene-based MEMS. Our mass-spring model can be also considered as one DOF approximations to solutions of MEMS problems which usually require applications of advanced finite element solvers, cf. [3, 4].

In Section 2, brief description of general model will be given, in Section 3 Sturm's theorem with its proof will be presented, in Section 4, application of Sturm's theorem to our model will be demonstrated, and in Section 5 conclusions will be drawn.

2. The model problem

We consider the graphene-based MEMS model for a parallel plate capacitor. The equation for the motion of the capacitor plate proposed by [12] reads as follows

$$m\frac{\mathrm{d}^2x}{\mathrm{d}t^2} + EA_c\frac{x}{L} - DA_c\left|\frac{x}{L}\right|\frac{x}{L} = \frac{\varepsilon_0 AV_{DC}^2}{2(d-x)^2}, \tag{1}$$

where m is mass of the flat plate, $x(t)$ the axial displacement, E the Young's modulus, A_c the cross-sectional area of graphene sheet, L the length of graphene sheet, D the third-order elastic stiffness constant, ε_0 is electric emissivity, A is the area of plate, V_{DC} is the applied voltage, and d is the gap between the plate and the substrate, see Fig. 1.

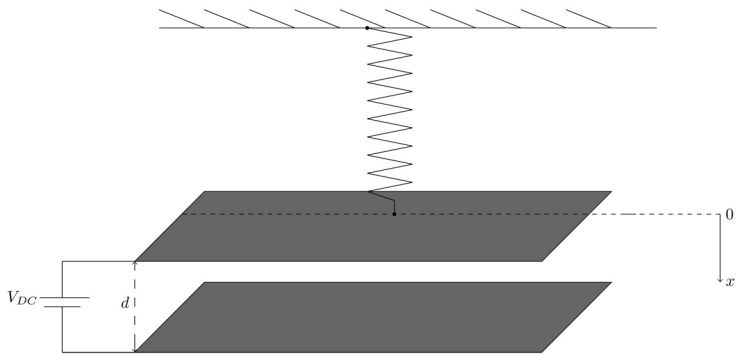

Fig. 1. The parallel capacitor MEMS model

The derivation of (1) follows from the standard framework of lumped-parameter modelling for longitudinal motion of elastic rod, see, e.g., [13, Ch. 9]. The third order elastic constant and the quadratic stress-strain equation which our model (1) is based on were validated by the experimental work of Zhou and Huang [16] which considered in plane deformation and then followed by the theoretical validation by Cadelano et al. [1]. Experimental validations were also done for the out-of-plane case by Lee et al. [6].

In order to transform the model equation (1) 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

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

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

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

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

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

which implies

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

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

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

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

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

where

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

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 $p_0(x)$ and $p_1(x)$ denote the polynomial $p(x)$ and its derivative $p'(x)$, respectively. Then, using the Euclidean algorithm we define the Sturm sequence

$$\begin{aligned} p_0(x) &= q_1(x) * p_1(x) - p_2(x)\,, \\ p_1(x) &= q_2(x) * p_2(x) - p_3(x)\,, \\ p_2(x) &= q_3(x) * p_3(x) - p_4(x)\,, \\ &\vdots \\ p_{k-2}(x) &= q_{k-1}(x) * p_{k-1}(x) - p_k(x)\,, \\ p_{k-1}(x) &= q_k(x) * p_k(x)\,. \end{aligned}$$

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

$$p_0(a), \ldots, p_{k-1}(a), p_k(a)$$

over the number of changes of the sign in the sequence

$$p_0(b), \ldots, p_{k-1}(b), p_k(b)\,.$$

Proof. We will follow the proof from [9]. Let $\sigma(a)$ is the number of sign changes in the sequence

$$p_0(a), \ldots, p_{k-1}(a), p_k(a)$$

and $\sigma(b)$ is the number of sign changes in the sequence

$$p_0(b), \ldots, p_{k-1}(b), p_k(b)\,.$$

Near any root c of $p(x)$, $p(x)$ is negative on one side of c and positive on another side of c. Hence $\sigma(x)$ can change only if it pass through a root of one of the $p_i(x)$ and $\sigma(x)$ looses one sign change. We have to study the following two cases:

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

Case 2. Let $p_0(x) = 0$, then $p_1(x)$ has constant sign in some interval sufficiently small interval $[c, d]$ such that:

- $p_1(x) > 0$: $p_1(c) < 0$ and $p_1(d) > 0$. Hence σ decreases by one.
- $p_1(x) < 0$: $p_1(c) > 0$ and $p_1(d) < 0$. Hence σ decreases by one.

Thus, σ looses one sign change if and only if x passes through a root of $p_0(x)$, which is initial $p(x)$. 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 $p(x)$.

□

4. Periodicity of solution

4.1. Zero initial conditions

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

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

Notice that $f(0) = 0$. In order to find periodic 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 the Euclidean algorithm we find the following Sturm sequence of polynomials

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

where

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

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 [10] to accomplish this task computationally for different α and K values. In addition, algorithm plots numerical solution solved using Runge-Kutta method (see [5]) for particular $\alpha > 0$ and $K > 0$ values in order to verify derived conclusion. Figs. 2–4 illustrate examples of periodic and non-periodic solutions.

Table 1. Sturm sequence for the case of zero initial conditions

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

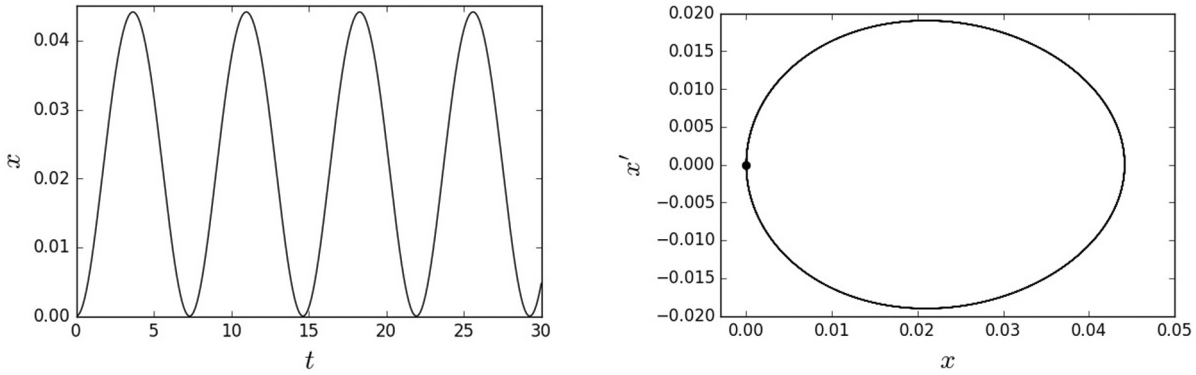

Fig. 2. Periodic solution $x(t)$ and its phase portrait for $\alpha = 5$ and $K = 0.018$

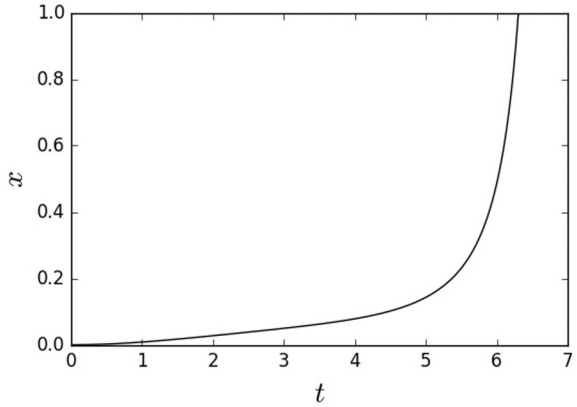

Fig. 3. Pull-in solution $x(t)$ for $\alpha = 14$ and $K = 0.018$

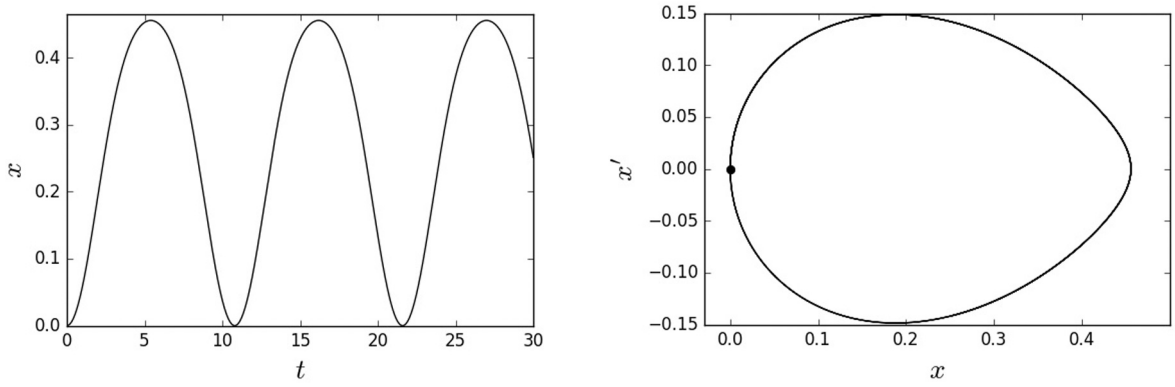

Fig. 4. Periodic solution $x(t)$ and its phase portrait for $\alpha = 0.001$ and $K = 0.124$

4.2. Non-zero initial conditions

In the case of non-zero initial conditions the Sturm polynomials for

$$p(s) = g_{pos}(s) = -2\alpha\, s^4 + (2\alpha + 3)s^3 - 3s^2 - 6Cs + 6(K + C)\,,$$

see Algorithm 2, are given as follows

$$\begin{aligned}
p_0(s) &= -2\alpha\, s^4 + (2\alpha + 3)s^3 - 3s^2 - 6Cs + 6(K + C)\,,\\
p_1(s) &= p_0'(s) = -8\alpha\, s^3 + 3(2\alpha + 3)s^2 - 6s - 6C\,,\\
p_2(s) &= \beta\, s^2 + g\, s + \tau\,,\\
p_3(s) &= \mu\, s + \rho\,,\\
p_4(s) &= \frac{\rho(\mu\, g - \rho\, \beta)}{\mu^2} - \tau\,,
\end{aligned}$$

where

$$\beta = \frac{3}{2} - \frac{3}{32\alpha}(2\alpha+3)^2\,, \quad g = \frac{9C}{2} + \frac{6}{32\alpha}(2\alpha+3)\,, \quad \tau = -6(K+C) + \frac{6}{32\alpha}(2\alpha+3)C\,,$$
$$\theta = 3(2\alpha+3) + \frac{8\alpha\, g}{\beta}\,, \quad \mu = 6 - \frac{8\alpha\,\tau}{\beta} + \frac{\theta\, g}{\beta}\,, \quad \varrho = 6C + \frac{\theta\,\tau}{\beta}\,.$$

The difference between the number of sign changes in the second column and the number of sign changes in third column of Table 2 states the number of roots of $p(s) = g_{pos}(s)$ in the interval $[0, 1]$. In the case of non-zero initial conditions, checking existence of root in the $(0, 1)$ interval is not enough. We have to perform two more steps.

Table 2. Sturm's polynomials for non-zero initial conditions

Sturm functions	$s = 0$	$s = 1$
$p_0(s)$	$6(K+C)$	$6K$
$p_1(s)$	$-6C$	$-2\alpha - 6C + 3$
$p_2(s)$	τ	$\beta + g + \gamma$
$p_3(s)$	ϱ	$\mu + \varrho$
$p_4(s)$	$\frac{\varrho(\mu\, g - \varrho\,\beta)}{\mu^2} - \tau$	$\frac{\varrho(\mu\, g - \varrho\,\beta)}{\mu^2} - \tau$

First, we will check the existence of a root in the interval $(0, 1)$ using Sturm table as it was done for the case of zero initial conditions. If there is no root, then pull-in occurs. If there is a root, then we go to the first additional step. In this step we check the sign of the function $p_0(s)$ at $s = 0$, see Algorithm 2 from Appendix. If the sign of $6(C + K)$ is negative, then there is pull-in. However, if the sign is positive, then we go to the second additional step. In this step we check the existence of a root in the interval $(-\infty, 0)$ using Sturm algorithm for

$$p(s) = g_{neg}(s) = 2\alpha\, s^4 + (3 - 2\alpha)s^3 - 3s^2 - 6Cs + 6(K + C)\,,$$

see Algorithm 2. If there is no root, then pull-in occurs. Otherwise, we conclude that there is periodic solution. Algorithm 2 from Appendix provides reader with Python code [10] 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, Figs. 5, 6 and 7 demonstrate the periodic and pull-in solutions for various parameters α, K and initial conditions.

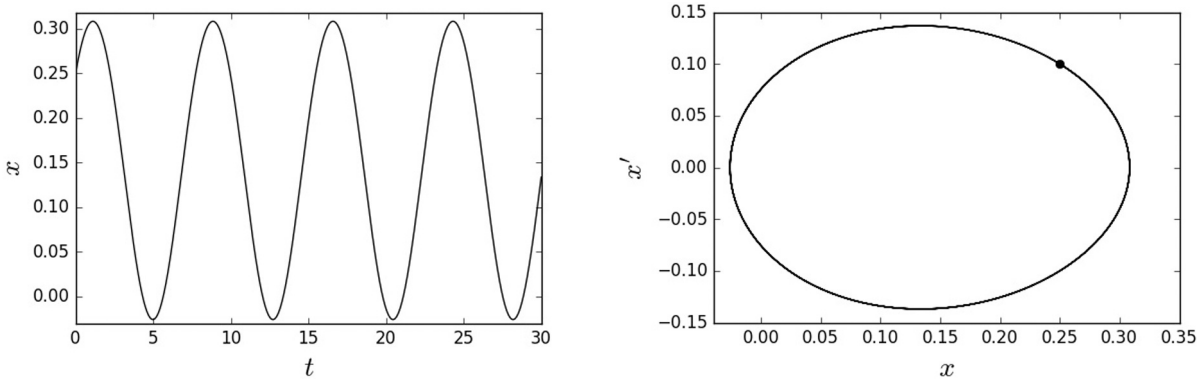

Fig. 5. Periodic solution $x(t)$ and its phase portrait for $\alpha = 0.01$, $K = 0.1$, $x_0 = 0.25$, and $x_0' = 0.1$

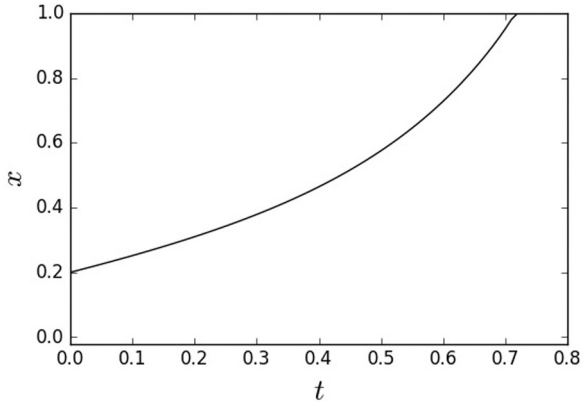

Fig. 6. Pull-in solution $x(t)$ for $\alpha = 14$, $K = 0.018$, $x_0 = 0.2$, and $x_0' = 0.5$

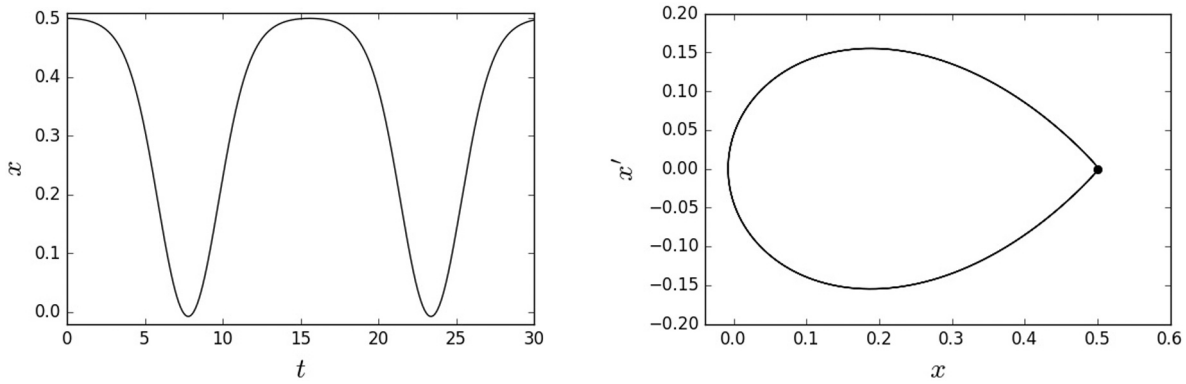

Fig. 7. Periodic solution $x(t)$ and its phase portrait for $\alpha = 0.001$, $K = 0.124$, $x_0 = 0.5$, and $x_0' = 0$

5. 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. Further research can be done to establish other analytical or numerical methods in order to determine solutions of the problem. Another research area is to find the methods for computing the periods of solutions when no pull-in occurs.

Acknowledgements

This research was supported by the Nazarbayev University ORAU grant "Modeling and Simulation of Nonlinear Material Structures for Mechanical Pressure Sensing and Actuation Applications".

References

[1] Cadelano, E., Palla, P. L., Giordano, S., Colombo, L., Nonlinear elasticity of monolayer graphene, Physical Review Letters 102 (2009) 235502. https://doi.org/10.1103/PhysRevLett.102.235502

[2] Gilberto, F., On the dynamic pull-in instability in a mass-spring model of electrostatically actuated MEMS devices, Journal of Differential Equations 262 (2017) 3597–3609. https://doi.org/10.1016/j.jde.2016.11.037

[3] Hannot, S. D. A., Rixen, D. J., A palette of methods for computing pull-in curves for numerical models of microsystems, Finite Elements in Analysis and Design 67 (2013) 76–90. https://doi.org/10.1016/j.finel.2013.01.001

[4] Kaneria, A. J., Sharma, D. S., Trivedi, R. R., Static analysis of electrostatically actuated micro cantilever beam, Procedia Engineering 51 (2013) 776–780. https://doi.org/10.1016/j.proeng.2013.01.111

[5] Landau, R. H., Manuel, J. P., Christian, C. B., Computational physics: Problem solving with Python, Weinheim, Germany: Wiley-VCH, 2015.

[6] Lee, C., Wei, X., Kysar, J. W., Hone, J., Measurement of the elastic properties and intrinsic strength of monolayer graphene, Science 321 (2008) 385–388. https://doi.org/10.1126/science.1157996

[7] Nathanson, H. C., Newell, W. E., Wickstrom, R. A., Davis, J. R., The resonant gate transistor, IEEE Transactions on Electron Devices 14 (1967) 117–133. https://doi.org/10.1109/T-ED.1967.15912

[8] Pelesko, J. A., Bernstein, D. H., Modeling MEMS and NEMS, Chapman & Hall/CRC, Boca Raton, USA, 2003.

[9] Prasolov, V. V., Polynomials, Berlin: Springer, 2004.

[10] Python Software Foundation, Python Language Reference, Version 2.7., http://www.python.org.

[11] Skrzypacz, P., Kadyrov, S., Nurakhmetov, D., Wei, D., Analysis of dynamic pull-in voltage of a nonlinear material MEMS model, Nonlinear Analysis: Modelling and Control (2018). (submitted)

[12] Wei, D., Kadyrov, S., Kazbek, Z., Periodic solutions of a graphene based model in micro-electromechanical pull-in device, Applied and Computational Mechanics 11 (2017) 81–90. https://doi.org/10.24132/acm.2017.322

[13] Woodson, H. H., Melcher, J. R., Electromechanical dynamics, Krieger Pub Co, 1985.

[14] Younis, M. I., MEMS linear and nonlinear statics and dynamics, Springer-Verlag New York, 2014.

[15] Zhang, W.-M., Yan, H., Peng, Z.-K., Meng, G., Electrostatic pull-in instability in MEMS/NEMS: A review, Sensors and Actuators A: Physical 214 (2014) 187–218. https://doi.org/10.1016/j.sna.2014.04.025

[16] Zhou, J., Huang, R., Internal lattice relaxation of single-layer graphene under in-plane deformation, Journal of the Mechanics and Physics of Solids 56 (2008) 1609–1623. https://doi.org/10.1016/j.jmps.2007.07.013

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 g_pos_0(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 g_pos_1(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 g_pos_2(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 g_pos_3(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 g_pos_4(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 g_neg_0(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 g_neg_1(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 g_neg_2(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 g_neg_3(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 g_neg_4(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 g_pos_0(alpha, k, x0, x1, 0) > 0: column0[0] = 1;
    else: column0[0] = 0;
    if g_pos_1(alpha, k, x0, x1, 0) > 0: column0[1] = 1;
    else: column0[1] = 0;
    if g_pos_2(alpha, k, x0, x1, 0) > 0: column0[2] = 1;
    else: column0[2] = 0;
    if g_pos_3(alpha, k, x0, x1, 0) > 0: column0[3] = 1;
    else: column0[3] = 0;
    if g_pos_4(alpha, k, x0, x1, 0) > 0: column0[4] = 1;
    else: column0[4] = 0;
    if g_pos_0(alpha, k, x0, x1, 1) > 0: column1[0] = 1;
    else: column1[0] = 0;
    if g_pos_1(alpha, k, x0, x1, 1) > 0: column1[1] = 1;
    else: column1[1] = 0;
    if g_pos_2(alpha, k, x0, x1, 1) > 0: column1[2] = 1;
    else: column1[2] = 0;
    if g_pos_3(alpha, k, x0, x1, 1) > 0: column1[3] = 1;
    else: column1[3] = 0;
    if g_pos_4(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;

   # Step 1
    if counter1 - counter2 == 0:
        print("There is a pull-in.");
        return 0;
    else:
     # Step 2
        if g_pos_0(alpha, k, x0, x1, 0) < 0:
            print("There is a pull-in.");
            return 0;
        else:
             # Step 3
            if check2(alpha, k, x0, x1) == 0:
                print("There is a pull-in.");
                return 0;
            else:
                print("There is a periodic solution.");
                return 1;

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

    if g_neg_0(alpha, k, x0, x1, 0) > 0: column0[0] = 1;
    else: column0[0] = 0;
    if g_neg_2(alpha, k, x0, x1, 0) > 0: column0[1] = 1;
    else: column0[1] = 0;
    if g_neg_2(alpha, k, x0, x1, 0) > 0: column0[2] = 1;
    else: column0[2] = 0;
    if g_neg_4(alpha, k, x0, x1, 0) > 0: column0[3] = 1;
    else: column0[3] = 0;
    if g_neg_4(alpha, k, x0, x1, 0) > 0: column0[4] = 1;
    else: column0[4] = 0;
    if g_neg_0(alpha, k, x0, x1, float('-inf')) > 0: column1[0] = 1;
    else: column1[0] = 0;
    if g_neg_1(alpha, k, x0, x1, float('-inf')) > 0: column1[1] = 1;
    else: column1[1] = 0;
    if g_neg_3(alpha, k, x0, x1, float('-inf')) > 0: column1[2] = 1;
    else: column1[2] = 0;
    if g_neg_3(alpha, k, x0, x1, float('-inf')) > 0: column1[3] = 1;
    else: column1[3] = 0;
    if g_neg_4(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();
```