Using simulations to find the force profile of MEMS comb-drives with variable gap-profiles

J. Groenesteijn

September 25, 2009

Contents

1 Introduction **3**
- 1.1 Introduction 3

2 Analytical Approach **4**
- 2.1 Introduction 4
- 2.2 The electrostatic driving force on a comb-drive 5
- 2.3 Optimization 6

3 Force calculation methods **11**
- 3.1 Introduction 11
- 3.2 Maxwell's Stress Tensor 11
  - 3.2.1 Introduction 11
  - 3.2.2 The Maxwell Stress Tensor in Comsol 11
- 3.3 Virtual Work Method 12
  - 3.3.1 Introduction 12
  - 3.3.2 The Virtual Work Method in Comsol 13

4 Simulations **14**
- 4.1 Introduction 14
- 4.2 Problems 14
  - 4.2.1 Mesh deformation 14
  - 4.2.2 Bounding boxes 17
- 4.3 Simulation methods 17
  - 4.3.1 Mesh Refinement 18
  - 4.3.2 Different types of elements 20
- 4.4 Comparison with literature 20
  - 4.4.1 Geometries 20
- 4.5 Results 20
  - 4.5.1 Simulation vs. Theory 23
  - 4.5.2 Analysis of the simulations 40
  - 4.5.3 Conclusion 44

5 Scripts **45**
- 5.1 Execution Scripts 45
- 5.2 Simulation Scripts 45
- 5.3 Analyzing Scripts 46

6 Conclusion **47**

A MatLab scripts **49**
A.1 Execution Scripst . . . 49
A.1.1 Run_All . . . 49
A.1.2 Analyse_All . . . 51
A.2 Simulation Scripts . . . 54
A.2.1 Create_Sim_DB . . . 54
A.2.2 Do_Sim_lin . . . 55
A.2.3 Do_Sim_quad . . . 62
A.2.4 Build_Finger . . . 69
A.2.5 getapplmodes . . . 72
A.3 Analysing Scripts . . . 75
A.3.1 Analyse_CPUTime . . . 75
A.3.2 Analyse_Output_remesh . . . 76
A.3.3 Analyse_Result_Analyse_mat . . . 81
A.3.4 Analyse_SimQ . . . 86
A.3.5 Make_Overview . . . 87

Chapter 1

Introduction

1.1 Introduction

This report is the end result of a project which is part of the master course 'EM-Statics' at the University of Twente. The focus of the project is on a force analysis of comb-drives. Comb-drives are well known in micro electromechanical systems (MEMS) to be used as actuator or sensor. They can be easily adjusted to match different requirements. Some application might requires the displacement to be linear to the actuation voltage while another application might require that the displacement is easily adjustable to certain positions.

To optimize the comb-drive to certain applications, the shape of the fingers can be adjusted. An example of how an optimal shape can be found analytically, is found in [1]. The main focus of this project was to find a method to use simulations to find the force that a comb-drive can exert related to the amount of overlap of the fingers while the bias voltage stays constant. This is done using examples from literature that are designed to have a certain force profile.

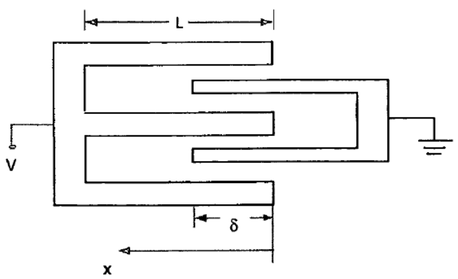
Chapter 2

Analytical Approach

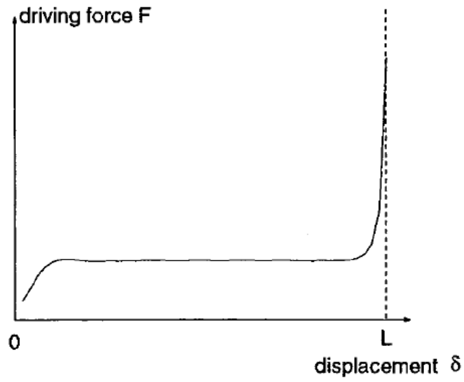
2.1 Introduction

Most comb-drives work by using the capacitance between the two fingers to exert a force that displaces one comb of the comb-drive, while the other comb usually is fixed. Other actuation methods are also possible, like the magnetic comb-drive in [2].

A standard comb-drive has finger of uniform width (Figure 2.1(a)), resulting in a capacitance which is linear with the displacement except at the ends of the operation range. This gives an electrostatic force that is independent of the displacement for the largest part of the operation range, as shown in Figure 2.1(b). However, it is possible that a different force-profile is preferred over the one mentioned earlier. The capacitance between the combs depends on the gap between the fingers and it can be shaped by making the gap variable. An altered force-profile can for instance be useful to counter the non-linearities of a spring at large displacements.



(a) A comb-drive with uniform finger-width



(b) Force-profile of a uniform comb-drive

Figure 2.1: A comb-drive with uniform finger-width and its force-profile [1]

2.2 The electrostatic driving force on a comb-drive

The comb-drives discussed here are driven by electrostatic forces. An ideal comb-drive can be modeled as a series of conductors in a lossless dielectric medium. When each conductor has a constant electrostatic potential, the charge in the conductor is distributed over the surface and satisfies:

$$q_i(\mathbf{r}) = \epsilon \frac{\partial \phi_i(\mathbf{r})}{\partial \mathbf{n}} \tag{2.1}$$

Here $q_i(\mathbf{r})$ is the surface charge density at point $\mathbf{r}$ on the surface of conductor i. $\epsilon = \epsilon_0 \epsilon_r$ is the dielectric constant of the medium, ϕ_i is the electrostatic potential of conductor i and $\mathbf{n}$ is the inward normal to a conductor at point $\mathbf{r}$. The electrostatic potential in the region outside the conductors satisfies the Laplace equation:

$$\nabla^2 \phi = 0 \tag{2.2}$$

An overview of the model can be seen in Figure 2.2. The electrostatic potential can be

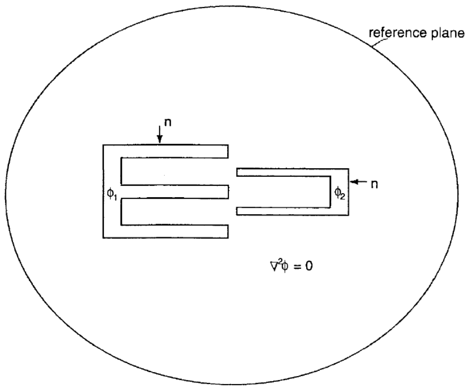


Figure 2.2: A system of ideal conductors embedded in a uniform lossless dielectric medium [1]

calculated using:

$$\phi_i = \sum_{j=1}^{m} \int_{\partial s_j} \frac{\partial \phi(\mathbf{r}')}{\partial \mathbf{n}} G(\mathbf{r}, \mathbf{r}') ds(\mathbf{r}') + C \tag{2.3}$$

And the total charge is given by:

$$Q = \sum_{j=1}^{m} \int_{\partial s_j} \frac{\partial \phi}{\partial n}(\mathbf{r}') ds(\mathbf{r}') \tag{2.4}$$

where m is the amount of finger pairs, $\mathbf{r}$ is the position vector of the source point, $\mathbf{r}'$ is the position vector of the field point, G is the Green's function given in Equation 2.5 (for 2D) or 2.6 (for 3D). ∂s_j is the surface of conductor j and C is a constant. The total charge of the model is 0.

$$G = \frac{1}{2\pi} \ln |\mathbf{r} - \mathbf{r}'| \tag{2.5}$$

$$G = \frac{1}{4\pi} \ln |\mathbf{r} - \mathbf{r}'| \tag{2.6}$$

The resulting electrostatic force density on the surface of a conductor is given by:

$$\mathbf{f} = -\frac{1}{2}\frac{q^2}{\epsilon}\mathbf{n} \tag{2.7}$$

This gives a total driving force on the moving comb equal to:

$$F_x = \int_\Gamma f_x ds \tag{2.8}$$

where Γ is the surface of the moving fingers.

2.3 Optimization

To find the optimal gap-profile that fits the desired force-profile, a optimization problem has to be formulated [1]. The goal is to minimize an objective function ($O(c_i)$) without violating the specified constraints ($g_i(c_i) \leq 0$). The objective function can be chosen as the integral of the square of the difference between the actual and desired force profiles over the range of operation of the comb drive:

$$O(c_i) = \int_{l_1}^{l_2} \left(F(c_i, x) - h(x)\right)^2 dx \tag{2.9}$$

Where c_i are the shape parameters of the comb drive, F is the driving force, x is the position of a moving finger, $h(x)$ is the desired force profile. l_1 and l_2 are the initial and final position of a moving finger. g_i are the constraints imposed by practical design issues, like minimum gap between the fingers.

The sensitivity of the objective function to a certain design variable (c) is given by the derivative to that design variable:

$$\frac{\partial O}{\partial c} = \int_{l_1}^{l_2} 2\left(F(c_i, x) - h(x)\right)\frac{\partial F}{\partial c}dx \tag{2.10}$$

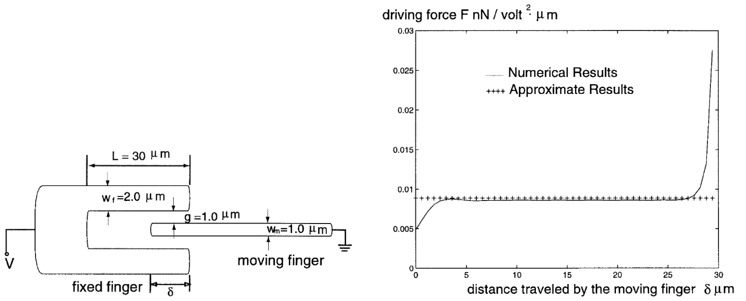
If fingers have a uniform width profile, Equation 2.11 can be used to find the approximate force. Ye et al. [1] compared this with a numerical simulation, the results can be seen in Figure 2.3.

$$F = \epsilon\frac{hV^2}{g} \tag{2.11}$$

where F is the driving force on the moving comb, h is the height of the fingers, V is the bias voltage and g is the gap between the fingers. Equation 2.11 shows that if the gap g is constant, that the force is constant too. However, if the gap would change with the displacement δ, the force would change with the displacement too, turning Equation 2.11 into Equation 2.12 where $g(x)$ is given by Equation 2.13. An example of this is shown in Figure 2.4. The gap is approximated by a sixth order polynomial function of the displacement δ.

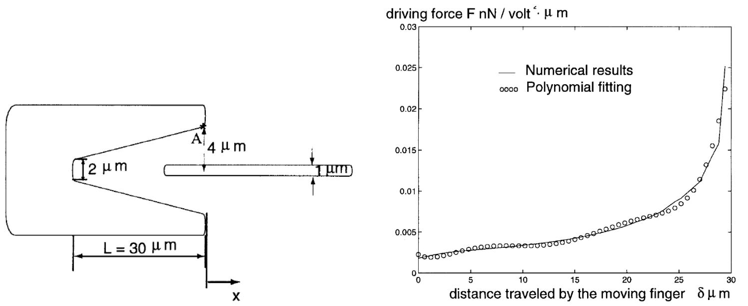
$$F(x) = \epsilon\frac{hV^2}{g(x)} \tag{2.12}$$

$$g(x) = \frac{1}{c_0 + c_1x + c_2x^2 + c_3x^3 + ... + c_nx^n} \tag{2.13}$$



(a) Model used in the numerical simulation (b) Force-profile of a uniform comb-drive

Figure 2.3: A comb-drive with uniform gap and its force-profile [1]



(a) Model used in the numerical simulation (b) Force-profile of the comb-drive

Figure 2.4: A comb-drive with linear gap and its force-profile [1]

The design parameters of the above equation are c_i with $i = 0, 1, 2, ..., n$. Ye et al. designed several comb-drives with linear, quadratic and cubic force profiles of which the moving comb had fingers with uniform width while the static comb had a variable shape. The results can be seen in Figure 2.5. It is quite clear that the comb-drives would take up a lot of space if these forms are used. The width of the gaps is modified by redefining the displacement x of the desired function $h(x)$. Instead of being proportional to x, x^2, etc, $h(x)$ is proportional to $x + x_0$, $(x + x_0)^2$, etc. The width of the comb-drive can then be reduced by choosing a suitable value for x_0.

A second method to decrease the total width of the comb-drive is by changing the shape of the fingers of both combs. However, the analysis of this becomes more complex and Equation 2.12 can no longer be used.[3] An analytical result can be obtained by going a few steps back from Equation 2.12.

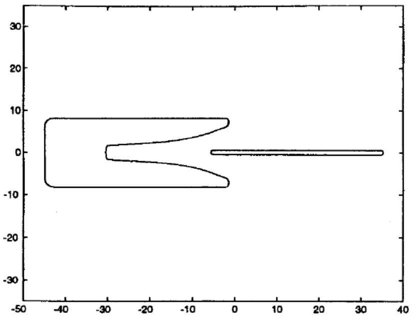
$$F_x = \frac{\partial E}{\partial x_1} = \frac{1}{2} V^2 \frac{\partial C}{\partial x_1} \tag{2.14}$$

$$C(x_1) = 2\epsilon h \int_{l_1}^{l_2} \frac{dx}{g_1(x) - g_2(x + L - x_0)} \tag{2.15}$$

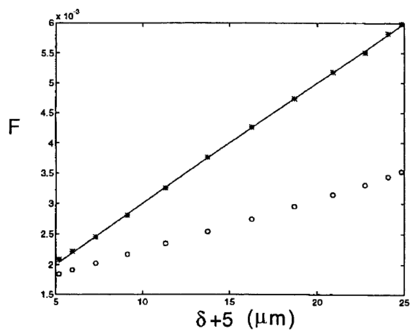
In the case that only 1 comb has shaped fingers, Equation 2.15 can be approximated by:

$$C = 2\epsilon h \int_{l_1}^{l_2} \frac{dx}{g(x)} \tag{2.16}$$

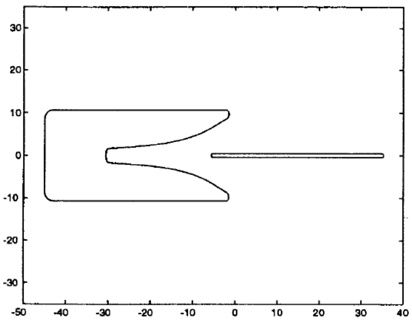
The derivative in Equation 2.14 will then result in Equation 2.12. To calculate the force when both combs have shaped fingers, a numerical solution is generally required, although an analytical solution can be found for some specific geometries. The results that were found by Ye et al. can be found in Figure 2.6. Not only did these comb-drives reduce the width of the gaps, they also improved the stability and increased the maximum force.



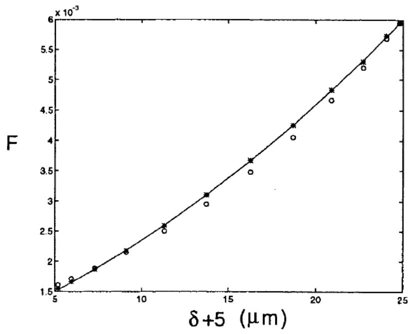
(a) Finger shape to get a linear force profile



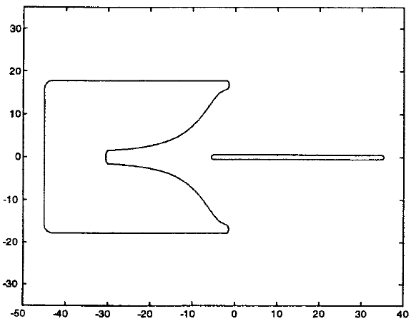
(b) Linear force profile of the comb-drive to the left



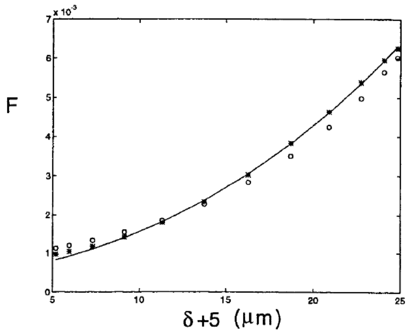
(c) Finger shape to get a quadratic force profile



(d) Quadratic force profile of the comb-drive to the left

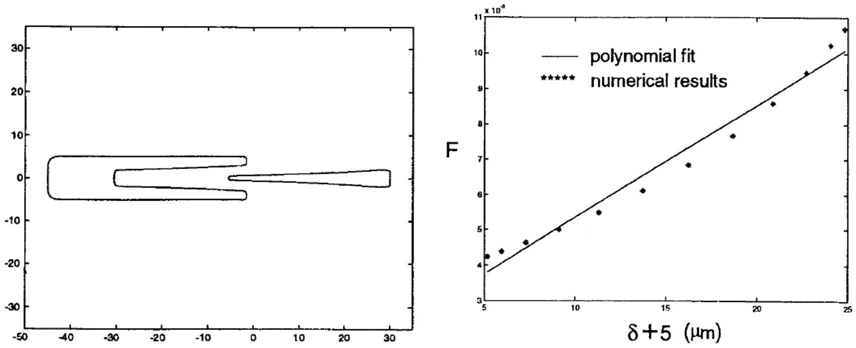


(e) Finger shape to get a cubic force profile



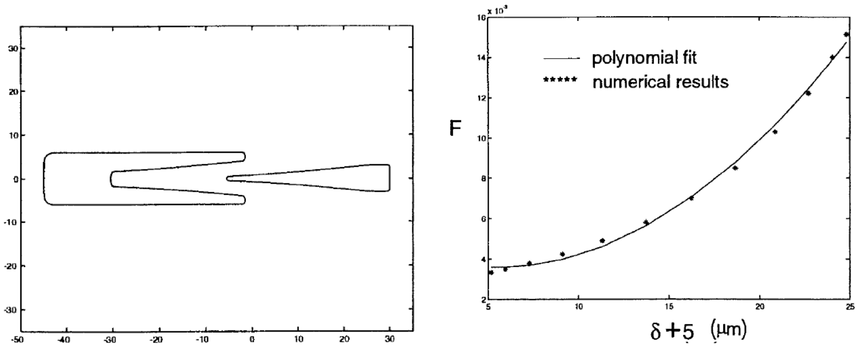
(f) Cubic force profile of the comb-drive to the left

Figure 2.5: Designs of variable comb drives with uniform moving fingers together with their force profiles. (The solid line is the desired function, the initial analytical force profile is marked with 'o' while the final analytical force profile is marked with '*') [1]



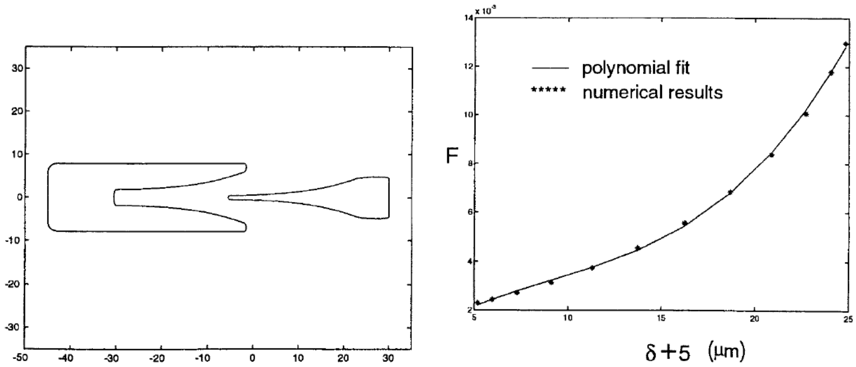
(a) Finger shape to get a linear force profile

(b) Linear force profile of the comb-drive to the left



(c) Finger shape to get a quadratic force profile

(d) Quadratic force profile of the comb-drive to the left



(e) Finger shape to get a cubic force profile

(f) Cubic force profile of the comb-drive to the left

Figure 2.6: Designs of variable comb drives with shaped moving fingers together with their force profiles. [1]

Chapter 3

Force calculation methods

3.1 Introduction

There are a lot of different ways to calculate the electromagnetic forces that are acting on an object. The most-used methods use 'Virtual Work' or use the Maxwell Stress Tensor. Virtual Work is the work resulting from either virtual forces acting through a real displacement or real forces acting through a virtual displacement. The Maxwell Stress Tensor is used to calculate the stresses on objects in magnetic or electric fields. These two methods are described below.

3.2 Maxwell's Stress Tensor

3.2.1 Introduction

The Maxwell Stress Tensor is used to calculated the stresses on objects in magnetic and electronical fields. The Maxwell Stress Tensor is defined as:

$$T_{ij} \equiv \epsilon_0 \left(E_i E_j - \frac{1}{2}\delta_{ij} E^2 \right) + \frac{1}{\mu_0} \left(B_i B_j - \frac{1}{2}\delta_{ij} B^2 \right) \tag{3.1}$$

Where T_{ij} is the Maxwell Stress Tensor in direction i, j. E is the electric field, B is the magnetic field and δ_{ij} is the Kronecker delta function. To find the force exerted by a surface, the tensor has to be integrated over that surface.

3.2.2 The Maxwell Stress Tensor in Comsol

The Maxwell Stress Tensor is the primary method in Comsol Multiphysics to calculate electromagnetic forces and torques [4]. In the application mode for electrostatics the forces are calculated by integrating equation 3.2.

$$\mathbf{n}_1 T_2 = -\frac{1}{2}\mathbf{n}_1(\mathbf{E} \cdot \mathbf{D}) + (\mathbf{n}_1 \cdot \mathbf{E})\mathbf{D}^T \tag{3.2}$$

Here, $\mathbf{E}$ is the electric field, $\mathbf{D}$ is the electric displacement and $\mathbf{n}_1$ is the outward normal from the object.

A force variable has to be defined on the appropriate subdomains. When this force variable is defined, several variables are generated automatically. They are all in the form:

Name	Dimension	Generated variable	Description
`<name>`	all	`<name>_nTx_emes`	Surface force density in the x direction
	all	`<name>_nTy_emes`	Surface force density in the y direction
	all	`<name>_forcex_emes`	Total force in the x direction
	all	`<name>_forcey_emes`	Total force in the y direction
	2D, 3D	`<name>_torquez_emes`	Total torque in the z direction, about the specified point
	3D	`<name>_torquex_emes`	Total torque in the x direction
	3D	`<name>_torquey_emes`	Total torque in the y direction
	3D	`<name>_torqueax_emes`	Total torque about the specified axis and point

Table 3.1: Variables using the Maxwell Stress Tensor in Comsol

`<variable name>_<type><independent variable name>_<application mode name>`
The resulting variables are given in Table 3.1.

`nTx` and `nTy` are defined on the exterior boundary of the selected subdomains. The `_emes` part relates to the Electrostatics application mode. These are the x and y components of Equation 3.2, together they represent the surface force density. These force densities can be visualized by using the Boundary Integration dialog box. `forcex` and `forcey` are the total force components in the x and y direction and can be visualized using the Global Data Display dialog box. The above variables are also accessible from Comsol Script and MatLab.

3.3 Virtual Work Method

3.3.1 Introduction

The Virtual Work Method calculates the work as a result from a virtual force acting through a real displacement or a real force acting through a virtual displacement. The virtual displacement is a displacement that can occur, but that will not actually occur. The same goes for the virtual force.

This can be represented by:

$$dU = \sum \mathbf{F} dr \cos\alpha \tag{3.3}$$

where dU is the virtual work, F are the forces that are considered, dr is the virtual displacement and α is the angle between the displacement and the force.

In electrostatics, the energy can be calculated by:

$$U = \frac{\epsilon}{2} \int_{\Omega_\mathrm{V}} |\mathbf{E}|^2 d\Omega_\mathrm{V} \tag{3.4}$$

where U is the electrostatic energy and $\mathbf{E}$ is the electric field.

Combining these equations gives us the force in the direction $\mathbf{p}$:

$$\mathbf{F}_\mathbf{p} = \frac{\epsilon}{2} \frac{\partial}{\partial \mathbf{p}} \int_{\Omega_\mathrm{V}} |\mathbf{E}|^2 d\Omega_\mathrm{V} \tag{3.5}$$

Using the chain rule, this becomes:

$$\mathbf{F}_\mathbf{p} = \epsilon \int_{\Omega_\mathrm{V}} \mathbf{E} \frac{\partial \mathbf{E}}{\partial \mathbf{p}} d\Omega_\mathrm{V} + \frac{\epsilon}{2} \int_{\Omega_\mathrm{V}} |\mathbf{E}|^2 \frac{\partial}{\partial \mathbf{p}} d\Omega_\mathrm{V} \tag{3.6}$$

This is the Virtual Work expression for the electrostatic force in direction **p**.

3.3.2 The Virtual Work Method in Comsol

Comsol has a built in command to calculate the force by using the Virtual Force Method. However, this command (`cemforce`) only gives a reliable result if linear Lagrange elements are used. The command `cemtorque` calculates the torque using the same method and is also restricted in the same way. Both commands are only available for backward compatibility and have been replaced by other methods to calculate the same, like the method described in 3.2.2.

Chapter 4

Simulations

4.1 Introduction

Finite Element simulations are usually an effective way to get an accurate idea of what a geometry will do. However, as usual, there are lots of things to look out for. When a comb-drive is simulated, there are several things to look out for. One of the biggest problems in simulating moving comb-drives are the high fields at the corners. Several methods to accurately simulate comb-drives are compared. The simulations are compared to the analytical and numerical results presented by Ye et al. in [1], Jensen et al. in [3] and to a comb-drive with uniform fingers and one with tapered fingers [5]. Table 4.1 gives all the different geometries used in the simulations.

4.2 Problems

When a different voltage is applied to two bodies, electrons want to from the body with the lowest electrical potential to the body with the highest electrical potential. When there is no conducting path between the bodies, the electrons try to get as close as possible to the body with the highest potential. As a result, the charge density at sharp corners at the ends of the fingers of the comb-drive will be very high, this can be seen in Figure 4.1. As can be seen, the sharp corners have very localized high density, while the charge density at the rounded corner is more spread out. The same distribution can be seen in Figure 4.2 which shows the electric field. It is easy to see that there are big differences in adjacent mesh-elements. This is not a very big problem when the mesh-elements stay more or less at the same place, however, this is not the case when one comb is moving with relation to the other. This would mean that the easiest solution would be to round the edges. However, round edges are harder to make in MEMS and usually generate a huge amount of mesh-elements when simulated which slows down the simulation enormously. When nothing is done to reduce or solve these effects, the simulation results will be very mesh dependent. This results in a 'force profile' like in Figure 4.3. Below several ways to solve this problem are discussed.

4.2.1 Mesh deformation

Comsol has several options that can be used to try to reduce the effect of a moving geometry. These application modes make sure that the mesh is not re-generated and thus that the

Sim_ID	Source	Description
1	Ye	Geometry for a linear force profile with one comb with shaped fingers and one comb with uniform fingers [1]
2	Ye	Geometry for a quadratic force profile with one comb with shaped fingers and one comb with uniform fingers [1]
3	Ye	Geometry for a cubic force profile with one comb with shaped fingers one comb with uniform fingers [1]
4	Ye	Geometry for a linear force profile with two combs with shaped fingers [1]
5	Ye	Geometry for a quadratic force profile with two combs with shaped fingers [1]
6	Ye	Geometry for a cubic force profile with two combs with shaped fingers [1]
7	Calculation	Geometry with only uniform fingers
8	Calculation	Geometry with only tapered fingers
9	Jensen	Geometry for a linear force profile with one comb with shaped fingers and one comb with uniform fingers. Derived by Jensen after Sim_ID 1 [3]
10	Jensen	Geometry for a cubic force profile with one comb with shaped fingers and one comb with uniform fingers. Derived by Jensen after Sim_ID 3 [3]
11	Jensen	Geometry for a linear force profile with two combs with shaped fingers. Derived by Jensen after Sim_ID 4 [3]
12	Jensen	Geometry for a cubic force profile with two combs with shaped fingers. Derived by Jensen after Sim_ID 6 [3]
13	Jensen	Geometry for a linear force profile but with higher force then the other shapes [3]

Table 4.1: Definition of the Simulation IDs

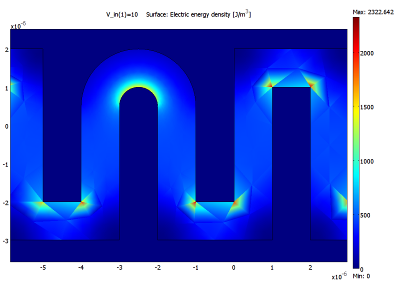


Figure 4.1: Electric energy-density (J/m^3) for round and sharp corners

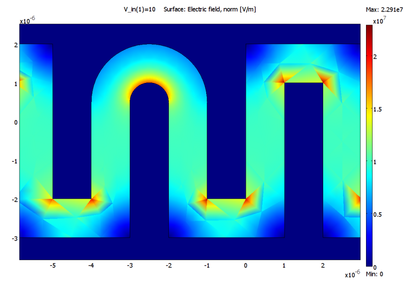


Figure 4.2: Electric field (V/m) for round and sharp corners

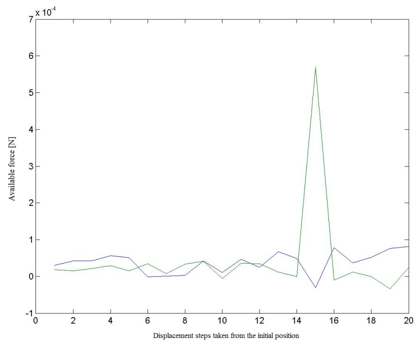


Figure 4.3: Unusable results caused by the electric fields at the sharp edges of the fingers.

amount of elements on a border remain the same. The problem with this however is that the mesh-elements have to deform a lot since the movements is very large (compared to the width of the gap). This can be seen in Figure 4.4. The figure also shows the high amount of mesh elements around the round finger. While the amount of mesh elements on the boundaries

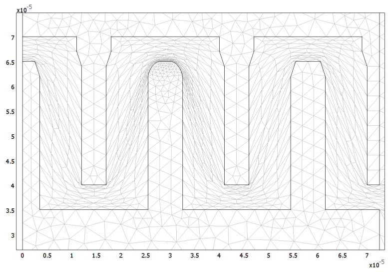
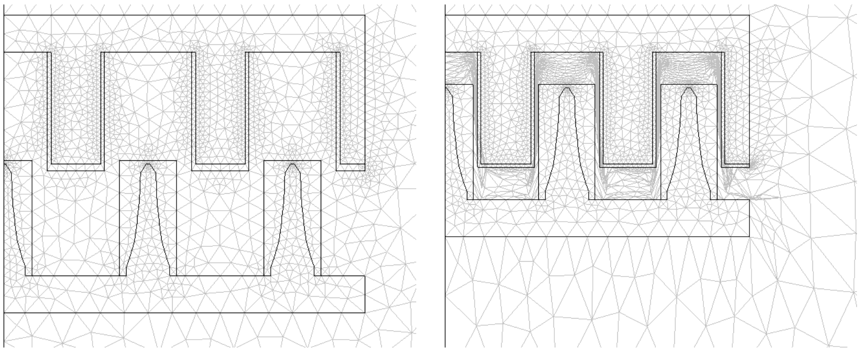


Figure 4.4: Deformed mesh when the fingers are at maximum displacement

stays the same, the mesh between the fingers still changes. This means that the localized fields still change from mesh element during the simulation. Because the combs have a large displacement, the mesh deforms a lot. This reduces the quality of the mesh and can even lead to situations where Comsol can not solve the equations for all the nodes of the mesh.

4.2.2 Bounding boxes

A different way to get better results at the edges is to surround them with a fake boundary. This boundary is used during meshing, but does not have an electrical or physical influence. The mesh inside this bounding box stays constant while the mesh outside the box still changes. The resulting mesh can be seen in Figure 4.5. The mesh-elements outside the bounding boxes still deform a lot, but inside, they remain the same. A problem with this method is that



(a) Mesh in the original state

(b) Deformed mesh after the combs move towards each other.

Figure 4.5: Outside the boundary boxes, the mesh deforms, inside, it stays the same

the bounding boxes of opposing fingers can not touch or cross each other. However, when the boxes are to small, they do not have any effect. To use this method properly, a reliable method has to be used to make the boxes as small as possible while still covering enough of the electric field. Figure 4.5 shows the bounding boxes used in combination with a deformed mesh. Because the area between the bounding boxes of opposing fingers is smaller then the area between the fingers, the mesh deforms more, leading to more situations where Comsol could not find a solution. However, since the bounding boxes solve most of the problem with the fringing fields, the deformed mesh is no longer needed. Most of the simulation steps resulted in a valid solution when a new mesh is generated after each displacement (though still not all), however, this at the cost of much more computing power and memory consumption. More about this in 4.3.

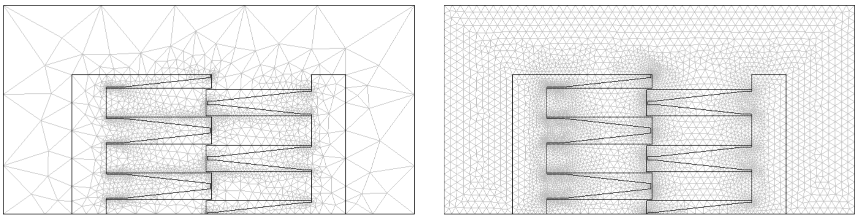
4.3 Simulation methods

A simulation is a numerical calculation to approximate the reality. To get an accurate result, the simulation needs to know about the real world and use those values. Obvious variables that it needs to know are physical dimensions and material properties. When Finite Element Analysis is used, some more variables are needed. A lot of these variables influence the way the mesh works and how calculations are done on that mesh. To see the influence of these variables, the mesh refinement (the size of the mesh elements) and the type of mesh elements are varied.

4.3.1 Mesh Refinement

The mesh refinement is a measure for the size of the mesh elements. To get the simulation result, a value is calculated for each point on the mesh. That value is used to interpolate the appropriate value on the borders of the mesh elements. How this is done depends on the type of element (see 4.3.2). If mesh elements are smaller, more calculations are done and the simulation result gets closer to the exact value defined by the used model. However, smaller mesh elements require more computing power and more available memory, thus making the simulation slower. Since changing the mesh refinement is influenced by a lot of variables, Comsol has several presets for the mesh refinement ranging from very fine (preset 1) to very course (preset 9).

An example of the mesh used in the simulation can be seen in Figures 4.6. The picture on the left shows a very course mesh refinement while the picture on the right shows a much finer mesh. Comsol Multiphysics optimizes the mesh so that it is finer on places where the geometry changes much while the mesh has larger elements at places where the geometry is very constant. Additional optimization can be done by making the mesh finer on places where the simulation results vary a lot and re-simulating the results.



(a) An example of a mesh with large elements (b) An example of a much finer mesh.

Figure 4.6: Examples of the same geometry with different mesh refinement

The influence of the mesh refinement can be seen in Figure 4.7. The simulation results are displayed for different values of the mesh refinement (presets 1 through 8). Even though all the results are within 5% of the average force (shows with the dashed lines), it can clearly be seen that the different mesh refinement presets give different results.

A finer mesh means more nodes and since an simulation result is calculated for each node and each line between adjacent nodes, this means more calculations need to be done to find all values. However, a finer mesh also means that the solver has a more accurate starting point for the next iteration in the simulation, which is then solved faster. Figure 4.8 shows the average simulation time per step of 1 μm displacement for the different mesh refinement presets. The finest mesh presets (1 through 3) need much more time to calculate the results then the courser meshes. Presets 4 through 8 need approximately the same time.

Simulation time and accuracy are not the only things that are influenced by the mesh refinement. Geometries with small features, like boundaries that are very close to each other, can cause problems for the mesher. Especially very course mesh presets and very fine mesh presets might not result in a valid mesh.

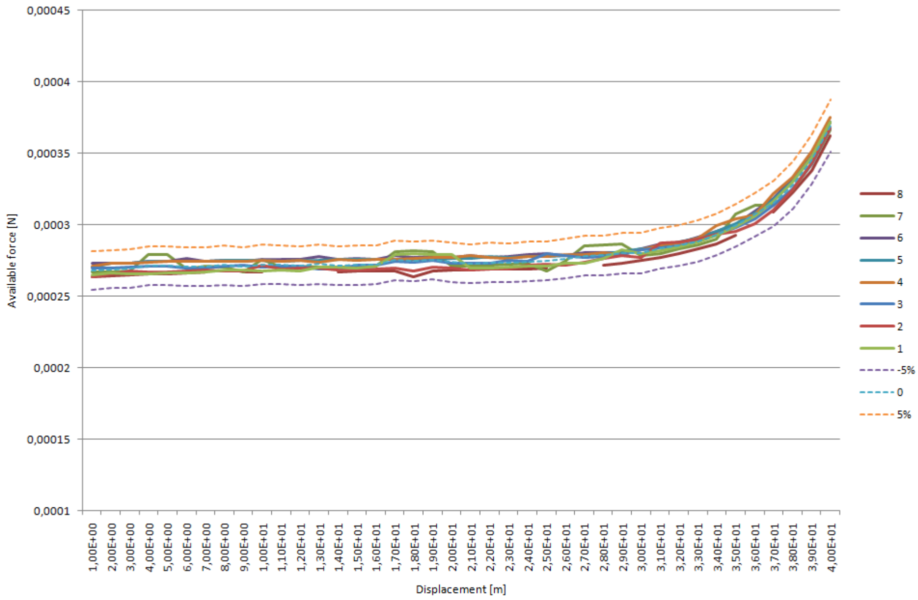


Figure 4.7: Varying simulation results because of different mesh refinements. The geometry used for these results has combs with uniform fingers.

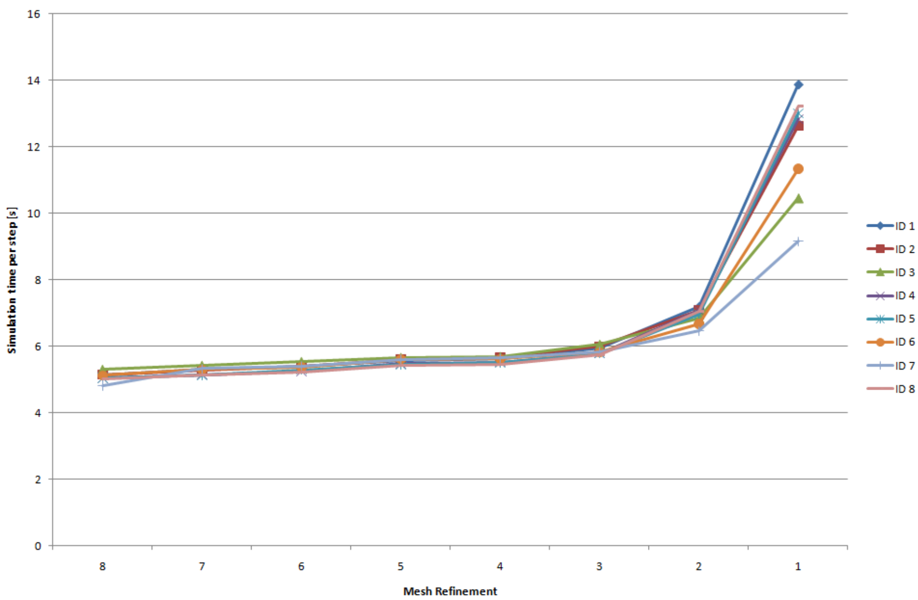


Figure 4.8: Simulation time per step of displacement in seconds vs. the mesh refinement for each simulated ID.

4.3.2 Different types of elements

The type of mesh element that is used, defines what kind of equations are solved at the nodes and on the borders of the mesh elements. An extensive explanation of the different kind of elements is given in [4]. The element type that is used the most in Comsol is the Lagrange element. A linear (or first order) Lagrange element defines that the simulated functions are linear between the nodes of the mesh. A higher order element gives a higher accuracy, but requires more computation power. The default setting for most application modes in Comsol is the quadratic Lagrange element (or second order Lagrange element).

4.4 Comparison with literature

Ye et al. [1] and Jensen et al. [3] analytically determined which shape finger would fit to which shape force-profile. Below, those shapes have been re-created to see if the simulations fit the analytical results. Ye did not give enough information to exactly re-create the geometries, sp the shapes have been copied from their graphical representations. Jensen gave better information about.

4.4.1 Geometries

To re-create the fingers, they were divided in 13 parts of 2.5μm length. The resulting geometries are shown in Figure 4.9. The geometries from [1] are shown at the left, their re-created counterparts are shown at the right.

Ye used numerical approximations to calculate the force profile of several geometries of which both combs had shaped fingers. These geometries have also been copied and can be seen in Figure 4.10.

The force profiles of Simulation ID's 7 and 8 have been analytically calculated. The geometries can be seen in Figure 4.11.

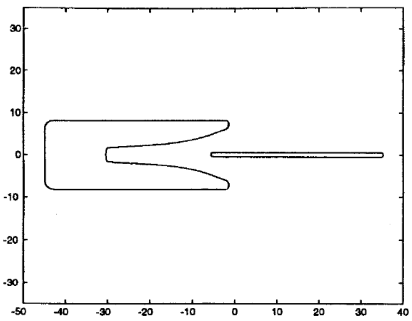
Jensen tried to reproduce the geometries Ye et al. made [3]. Equations were used to describe the shapes of the fingers, the equations for the static fingers are shown in Table 4.2 and the equations for the moving fingers are shown in Table 4.3. These geometries have been copied and can be seen in Figure 4.12.

Sim_ID	Equations
9	$\frac{1}{0.1507+0.3135\cdot10^{-2}x+0.1679\cdot10^{-2}x^2+0.8051\cdot10^{-4}x^3+0.1244\cdot10^{-5}x^4}$
10	$\frac{1}{0.06897-0.5754\cdot10^{-2}x+0.1419\cdot10^{-2}x^2+0.1411\cdot10^{-4}x^3-0.1879\cdot10^{-6}x^4}$
11	$\frac{1}{0.3188-0.7246\cdot10^{-2}x}$
12	$\frac{1}{0.1522-0.7262\cdot10^{-2}x+0.7469\cdot10^{-3}x^2+0.1893\cdot10^{-4}x^3}$
13	$\frac{6.5}{1.5+x}$

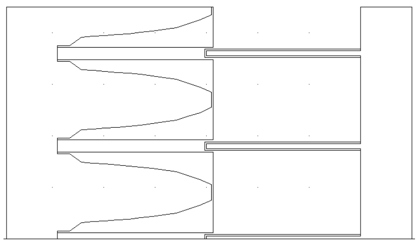
Table 4.2: Equations used for Jensens static fingers, x is the displacement

4.5 Results

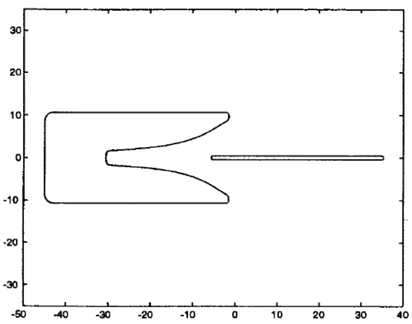
The simulation methods mentioned in 4.3 have been used to simulate the geometries defined in Table 4.1. Figure 4.13 shows how the displacement x is defined. An overview of the



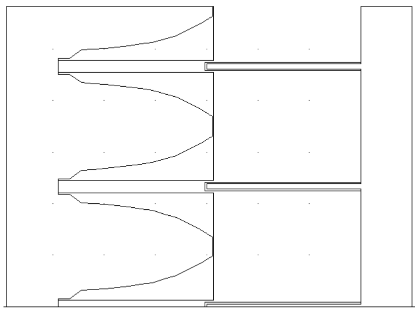
(a) Ye's model for Sim_ID 1



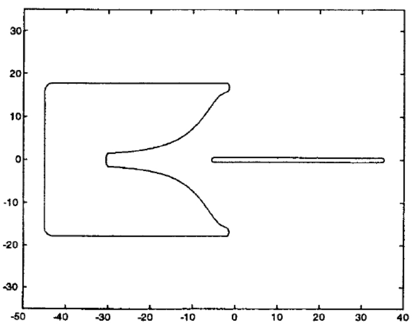
(b) The geometry that was used to simulate the model to the left.



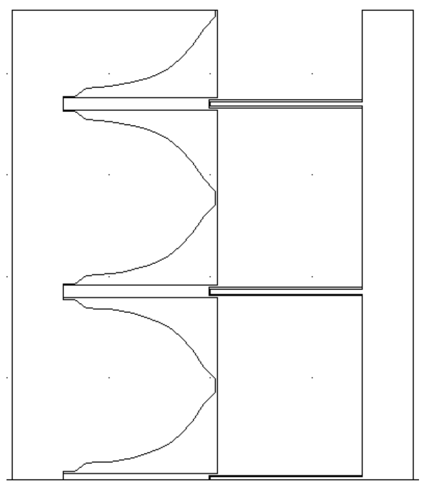
(c) Ye's model for Sim_ID 2



(d) The geometry that was used to simulate the model to the left.

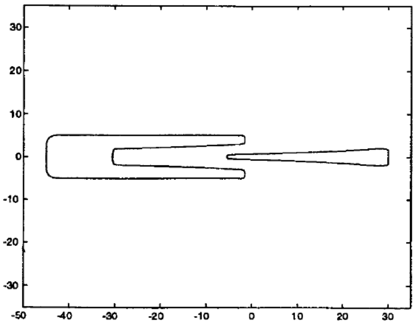


(e) Ye's model for Sim_ID 3

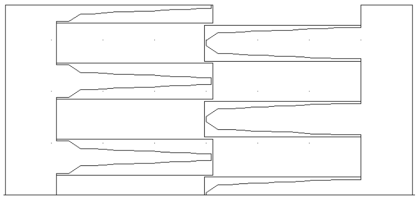


(f) The geometry that was used to simulate the model to the left.

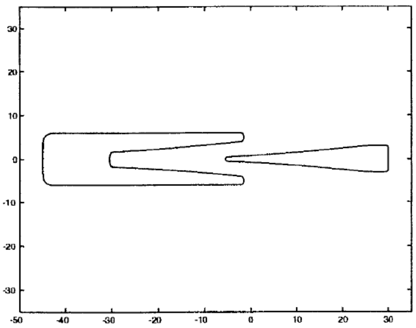
Figure 4.9: The models from Ye et al. [1] with one comb with straight fingers and the geometries used in simulation



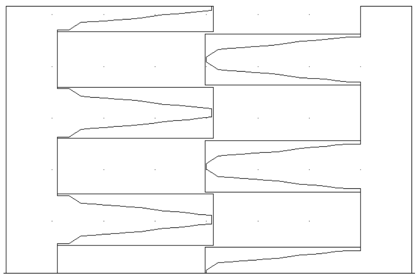
(a) Ye's model for Sim_ID 4



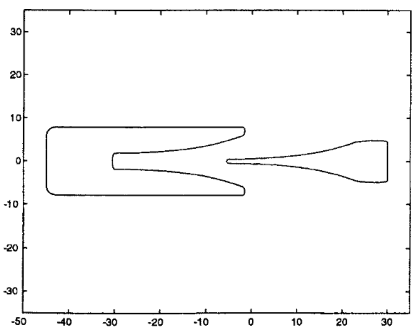
(b) The geometry that was used to simulate the model to the left.



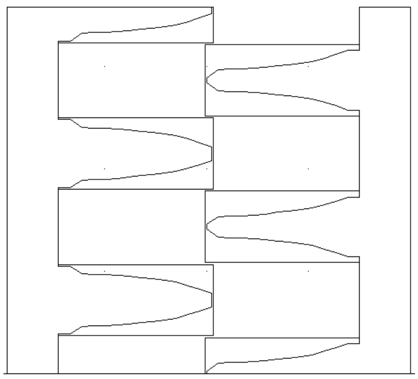
(c) Ye's model for Sim_ID 5



(d) The geometry that was used to simulate the model to the left.

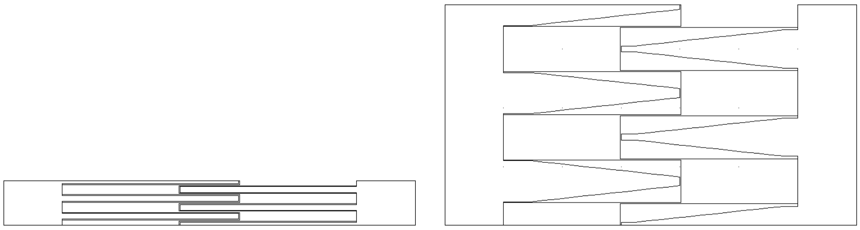


(e) Ye's model for Sim_ID 6



(f) The geometry that was used to simulate the model to the left.

Figure 4.10: The models from Ye et al. [1] with two shaped fingers and the geometries used in simulation



(a) A comb-drive with uniform fingers (b) A comb-drive with tapered fingers

Figure 4.11: The geometries of a comb-drive with uniform fingers and of a comb-drive with tapered fingers.

Sim_ID	Equations
9	uniform
10	uniform
11	$\frac{1}{0.3188-0.7246\cdot10^{-2}x}$
12	$\frac{1}{0.1522-0.7262\cdot10^{-2}x+0.7469\cdot10^{-3}x^2+0.1893\cdot10^{-4}x^3}$
13	uniform

Table 4.3: Equations used for Jensens moving fingers, x is the displacement

different simulation methods is given in table 4.4.

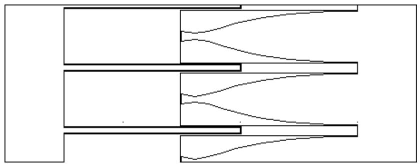
Method	Calculation Method	Displacement Method	Mesh Type
1	Maxwell Stress Tensor	Remeshing	Linear
2	Virtual Work Method	Remeshing	Linear
3	Maxwell Stress Tensor	Deformation	Linear
4	Maxwell Stress Tensor	Remeshing	Quadratic
5	Maxwell Stress Tensor	Deformation	Quadratic

Table 4.4: The different simulation methods

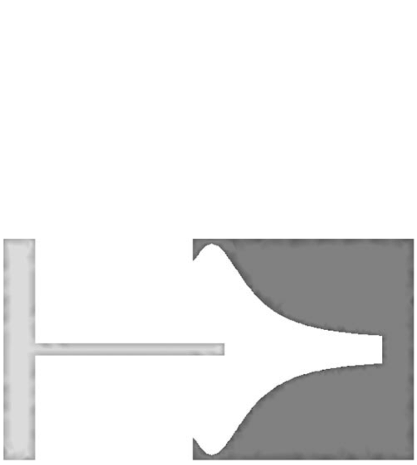
4.5.1 Simulation vs. Theory

The geometries that were used were mainly taken from literature [1, 3] (Sim_ID 1 through 6 and 9 through 13). Ye et al. did not specify the thickness of the fingers they use during their calculations and simulations and they use only one finger. Since both these effects are constant factors of which the force is linearly dependent, the results given by Ye et al. need to be scaled by a constant factor to match them to the results of my simulations. The results discussed below use a scaling factor of 20. Sim_ID 7 and 8 are compared to my own calculations. A polynomial fit has been made of each result to check the validity. The fit was made from $x = 0$ to $x = 11 \cdot 10^{-6}$ to avoid any influences from the large displacement. The equation used for the fit is shown in equation 4.1. For the geometries with a linear force profile, D and C were set to 0. The ones with a quadratic profile had D set to 0. The different

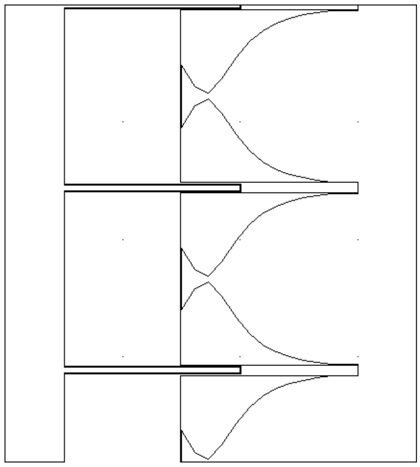
(a) Jensens model of Sim_ID 9



(b) The geometry that was used to simulate the model to the left.

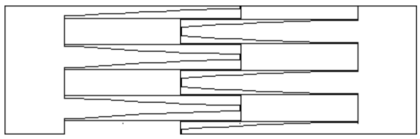


(c) Jensens model of Sim_ID 10

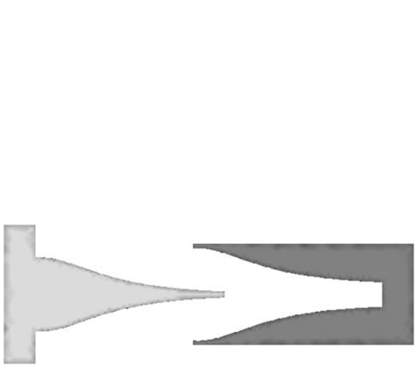


(d) The geometry that was used to simulate the model to the left.

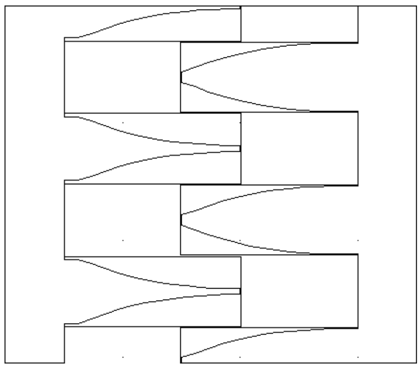
(e) Jensens model of Sim_ID 11



(f) The geometry that was used to simulate the model to the left.

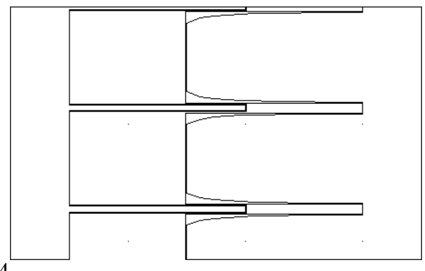


(g) Jensens model of Sim_ID 12



(h) The geometry that was used to simulate the model to the left.

(i) Jensens model of Sim_ID 13



(j) The geometry that was used to simulate the model to the left.

Figure 4.12: The models from Jensen et al. [3] and the geometries used in simulation

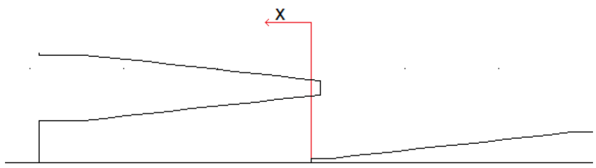


Figure 4.13: Basic geometry with the displacement defined

geometries are supposed to have purely linear/quadratic/cubic force profiles, but the results presented by Ye and Jensen were approximated by third order polynomials showing non-zero values for the other coefficients as well.

$$F(x) = Dx^3 + Cx^2 + Bx + A \tag{4.1}$$

Sim_ID 1

The first geometry calculated by Ye et al. had a finger with uniform width at one side and a shaped finger at the other comb. The finger was shaped in such a way that the force profile was linear. The model that was used during the simulations can be seen in Figure 4.9(b). Figure 4.14 shows the results of the simulation using the different methods that were discussed.

It can clearly be seen that the different simulation methods give different results. Pearson's χ-square test was used to find the accuracy of a linear fit to each of the simulation results up to a displacement of 10μm, which resulted in an χ^2 value of over 0.997 (1 being a perfect fit). Table 4.5 shows the parameters of the fit. The non-linear part of the force profile starts sooner compared to Ye's results (ca. 10μm versus ca. 12μm). The large increase in force at large displacements is caused by parasitic capacitances between the fingers and the base of the opposing comb. Why this effect is barely visible in the results of Ye is unknown.

	D	C	B	A	χ^2
1			1,9895E-06	1,0212E-04	9,9394E-01
2			3,8598E-06	2,3892E-04	9,9704E-01
3			2,3190E-06	1,0179E-04	9,7051E-01
4			2,6047E-06	1,4752E-04	9,9842E-01
5			2,3559E-06	1,4302E-04	9,9885E-01
Ye			4,9854E-06	1,4574E-04	9,9972E-01

Table 4.5: Coefficients of the fitted polynomials for Sim_ID1

Sim_ID 2

The second geometry also has one comb with fingers of uniform width while the other comb has shaped fingers. The force profile should be quadratic. The model that was used during the simulations can be seen in Figure 4.9(d). Figure 4.15 shows the simulation results and Table 4.6 shows the coefficients of the polynomial fit. All the simulation results have approximately the same shape. However, the force profile should be quadratic, but both the results from Ye as the results of the simulations look mainly linear at low displacement.

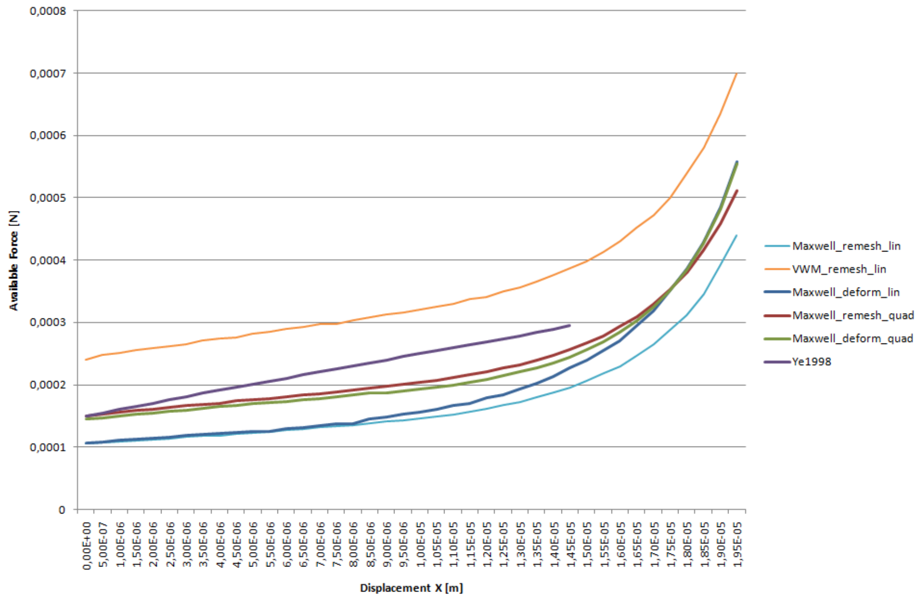


Figure 4.14: Simulation results of Sim_ID 1

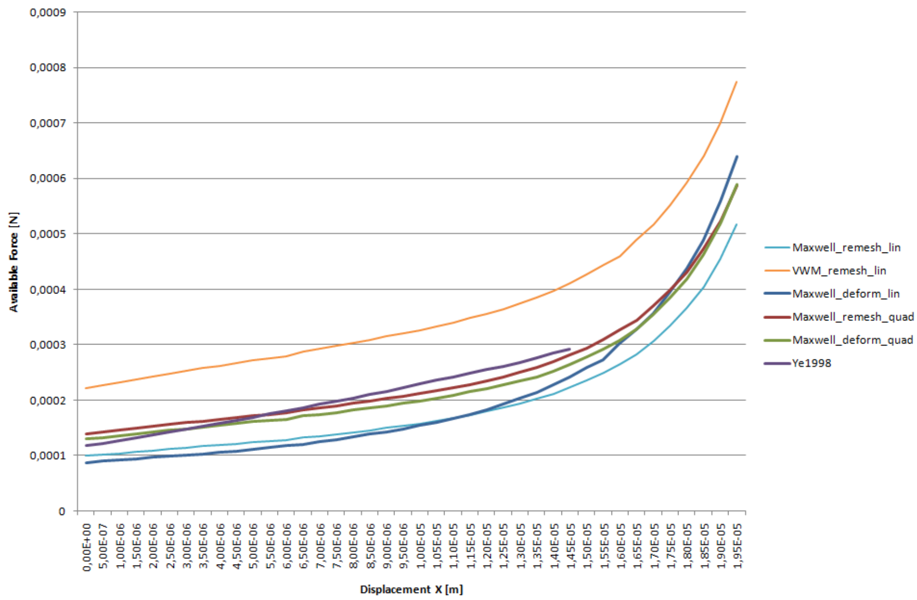


Figure 4.15: Simulation results of Sim_ID 2

	D	C	B	A	χ^2
1		5,4634E-08	1,6606E-06	9,8796E-05	9,9777E-01
2		3,4774E-08	4,3960E-06	2,1882E-04	9,9915E-01
3		1,0569E-07	9,0471E-07	8,8414E-05	9,9815E-01
4		4,2050E-08	2,6480E-06	1,3739E-04	9,9880E-01
5		4,4470E-08	2,4080E-06	1,2860E-04	9,9888E-01
Ye		4,2159E-08	4,6428E-06	1,1271E-04	9,9998E-01

Table 4.6: Coefficients of the fitted polynomials for Sim_ID2

Sim_ID 3

The geometry with Sim_ID 3 has one comb with fingers of uniform width while the other comb has shaped fingers too. The force profile should be cubic. The model that was used during the simulations can be seen in Figure 4.9(f). Figure 4.16 shows the simulation results and Table 4.7 shows the coefficients of the polynomial fit. The parameters of the fit show that the simulations are very much alike. However, the values of parameters C and D are very different from the ones of Ye's results.

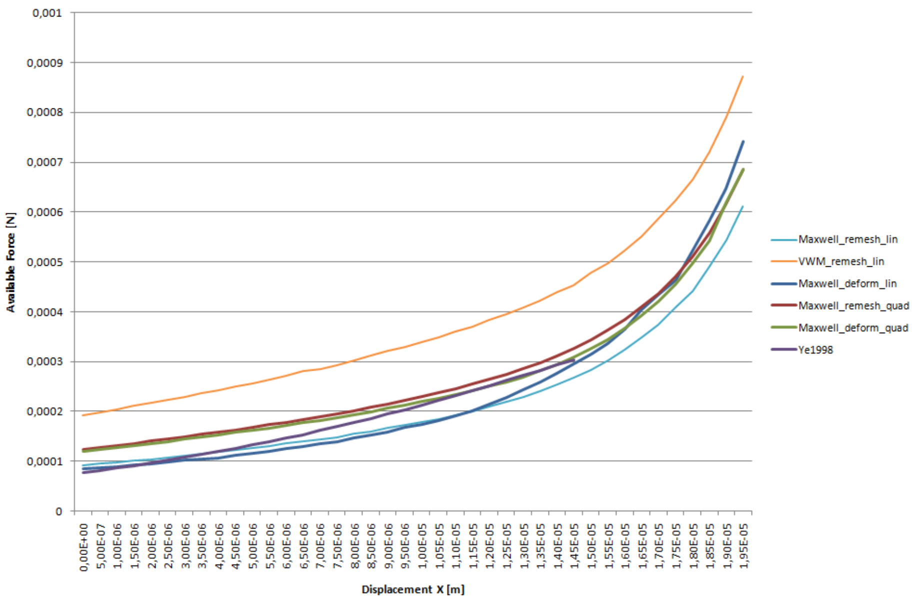


Figure 4.16: Simulation results of Sim_ID 3

Sim_ID 4

Sim_ID 4 has a geometry in which both combs have shaped fingers.. The force profile should be linear. The model that was used during the simulations can be seen in Figure 4.10(b). Figure 4.17 shows the simulation results and Table 4.8 shows the coefficients of the polynomial fit. The results show that the force is approximately 1.5 times as high as from the geometry with one comb with uniform finger width. While the simulations do show a linear force profile,

	D	C	B	A	χ^2
1	2,4709E-09	4,6840E-09	3,1037E-06	8,8367E-05	9,9989E-01
2	2,8944E-09	-4,2120E-09	6,0938E-06	1,8614E-04	9,9981E-01
3	3,3911E-09	2,5056E-08	2,3419E-06	8,2111E-05	9,9949E-01
4	2,7876E-09	-5,5447E-09	4,1170E-06	1,1946E-04	9,9996E-01
5	2,2606E-09	-1,5537E-09	3,9447E-06	1,1574E-04	9,9991E-01
Ye	-4,7495E-10	1,4098E-07	3,8786E-06	7,3238E-05	9,9996E-01

Table 4.7: Coefficients of the fitted polynomials for Sim_ID3

the linear coefficient is far lower then that of Ye's results. This can also be seen from the parameters of the polynomial fit.

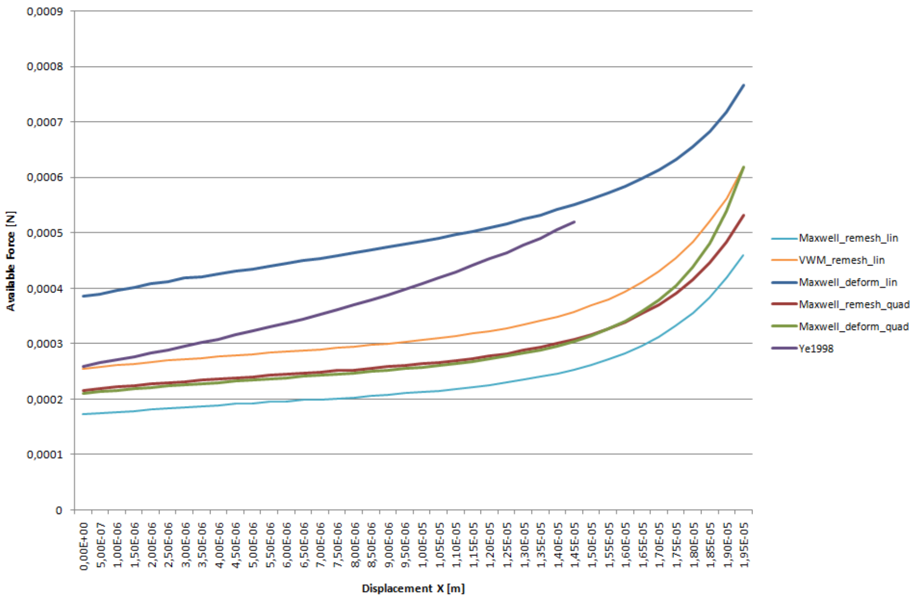


Figure 4.17: Simulation results of Sim_ID 4

Sim_ID 5

This geometry also has shaped fingers on both combs. The force profile should be quadratic. The model that was used during the simulations can be seen in Figure 4.10(d). Figure 4.18 shows the simulation results and Table 4.9 shows the coefficients of the polynomial fit. The results show that this geometry also has a 50% increase in force compared to its equivalent with one comb with uniform finger width. The polynomial fit shows that the quadratic parameter (C) of the simulation results is approximately 10 times lower than that of Ye's results.

	D	C	B	A	χ^2
1			1,9982E-06	1,7057E-04	9,9691E-01
2			2,5237E-06	2,5335E-04	9,9653E-01
3			4,9064E-06	3,8175E-04	9,9901E-01
4			2,3286E-06	2,1445E-04	9,9674E-01
5			2,3259E-06	2,0887E-04	9,9756E-01
Ye			7,6079E-06	2,4384E-04	9,9131E-01

Table 4.8: Coefficients of the fitted polynomials for Sim_ID4

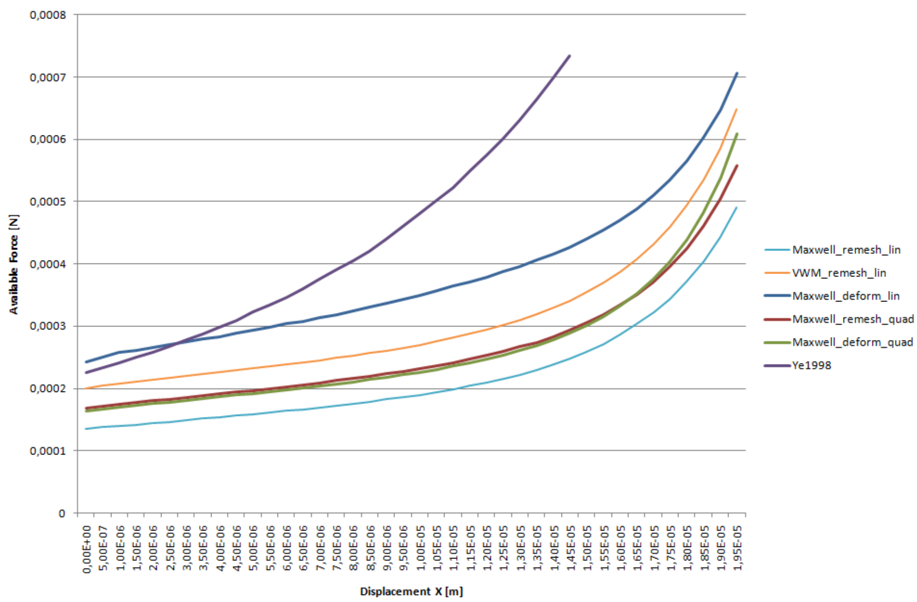


Figure 4.18: Simulation results of Sim_ID5

	D	C	B	A	χ^2
1		4,5009E-08	1,7042E-06	1,3425E-04	9,9881E-01
2		4,6984E-08	2,3686E-06	1,9988E-04	9,9855E-01
3		5,1267E-08	3,9616E-06	2,4353E-04	9,9812E-01
4		4,2716E-08	2,1386E-06	1,6782E-04	9,9909E-01
5		4,1131E-08	2,1266E-06	1,6345E-04	9,9879E-01
Ye		3,3607E-07	5,1630E-06	2,2302E-04	9,9958E-01

Table 4.9: Coefficients of the fitted polynomials for Sim_ID5

Sim_ID 6

The last geometry from Ye also has shaped fingers on both combs. The force profile should be cubic. The model that was used during the simulations can be seen in Figure 4.10(f). Figure 4.19 shows the simulation results and Table 4.10 shows the coefficients of the polynomial fit. The force is again approximately 1.5 times as high as the force calculated for Sim_ID 3. Again the parameters of the fit show that the simulation results are very different from Ye's results. The cubic parameter is about a factor 2 higher. The quadratic parameters of the simulation results are negative while the parameter is positive for Ye's results.

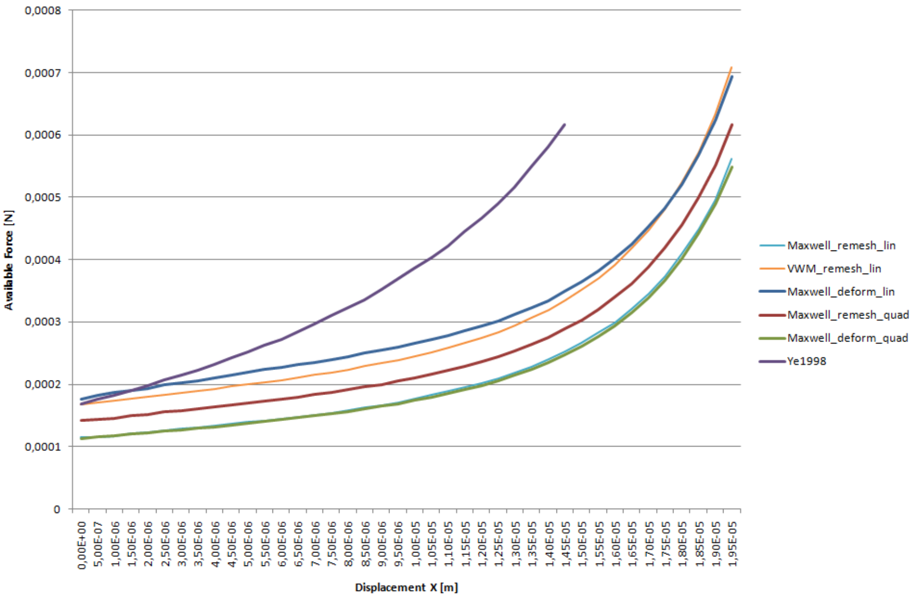


Figure 4.19: Simulation results of Sim_ID 6

	D	C	B	A	χ^2
1	4,0994E-09	-7,1651E-08	2,8625E-06	1,1026E-04	9,9946E-01
2	4,2846E-09	-7,4834E-08	3,5628E-06	1,6374E-04	9,9991E-01
3	4,1352E-09	-1,1039E-07	4,9808E-06	1,7164E-04	9,9946E-01
4	3,5415E-09	-5,7244E-08	3,0912E-06	1,3802E-04	9,9964E-01
5	3,3446E-09	-4,8241E-08	2,5831E-06	1,1008E-04	9,9995E-01
Ye	7,0367E-09	2,0036E-08	7,1721E-06	1,6066E-04	9,9992E-01

Table 4.10: Coefficients of the fitted polynomials for Sim_ID6

Sim_ID 7

Sim_ID 7 has uniform finger width on both combs, the model used in the simulations is shown in Figure 4.11(a). The simulated results are compared to a theoretical value calculated using equation 2.14 which is repeated below. [5]

$$F_x = \frac{1}{2}V^2\frac{\partial C}{\partial x} = \frac{1}{2}V^2\frac{\epsilon t}{g} \tag{4.2}$$

Where F_x is the x-component of the force per capacitor, V is the voltage on the two combs (1V during the simulations), x is the displacement as defined in Figure 4.13, ϵ is the electrical permittivity, t is the thickness of the combs (set to 1 during the simulations) and g is the gap between the fingers. The results that were calculated from the simulations using method 3 show a lot of noise, while the results from the other simulation methods show a very smooth flat line.

The theoretical value that is calculated is using the exact parameters that are used in the simulation as well and can thus be used to check the accuracy of the simulations. The results calculated using method 2 are nearly exactly equal to the theoretical values. However, method 3 has results that are nearly 2.5 times higher while the remaining methods have values 1.8 to 2.4 times lower then the theoretical force.

Figure 4.21 show that the fits are very close approximations of the simulation results, however, the χ^2 value that is found for each fit is very low. This is most likely because the fit was of a different 'shape' then the results (linear versus constant).

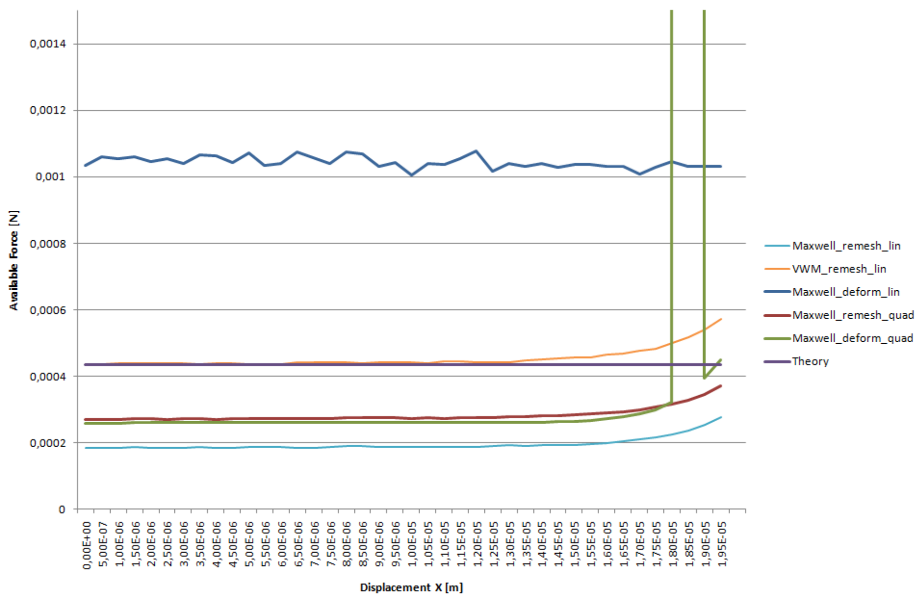


Figure 4.20: Simulation results of Sim_ID7

Sim_ID 8

ID 8 has tapered fingers on both combs, the model that is used during the simulations is shown in Figure 4.11(a). The simulation results are again compared to a theoretical value calculated using equation 4.3. This can be seen in Figure 4.23. The analytical calculation is not valid for high displacement because the capacitive effect between the fingers and the base of the opposite comb have not been taken into account. This results in a lower increase of

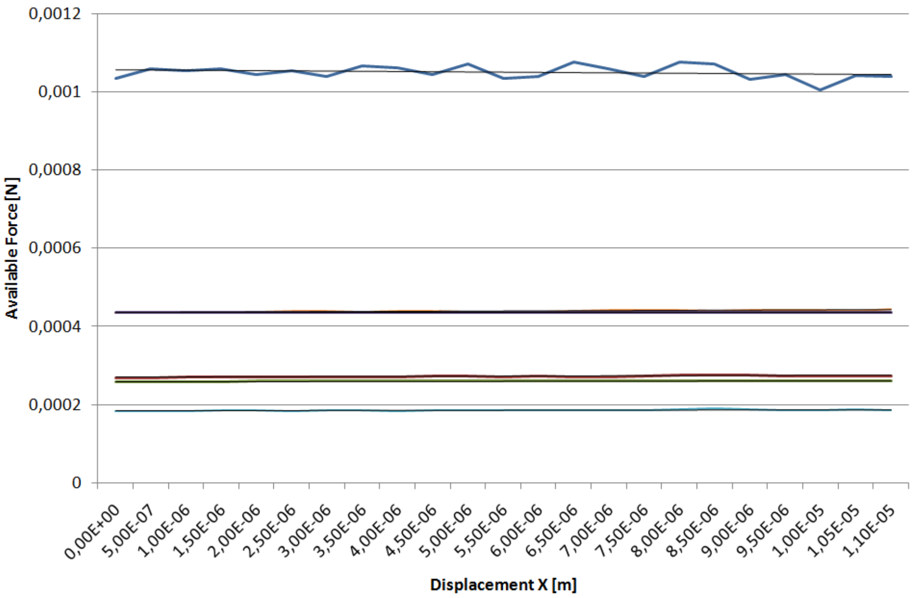


Figure 4.21: Fit on the simulation results of Sim_ID7 for low displacement

	D	C	B	A	χ^2
1			1,7859E-07	1,8346E-04	4,7788E-01
2			2,5684E-07	4,3607E-04	5,7446E-01
3			-5,7779E-07	1,0563E-03	5,2908E-02
4			2,3204E-07	2,6919E-04	7,1245E-01
5			9,1658E-08	2,5914E-04	3,6658E-01
Theory			-1,9079E-20	4,3557E-04	3,1259E-16

Table 4.11: Coefficients of the fitted polynomials for Sim_ID7

theoretical force then what the simulations will show at high displacement.

$$F_x = \frac{1}{2}V^2\frac{\partial C}{\partial x} = \frac{1}{2}V^2\frac{\partial}{\partial x}\left(\frac{\epsilon t l}{g_{max} - x\frac{g_{max}-g_{min}}{L}}\right) \tag{4.3}$$

The definition of the variables used in this equation can be found in Figure 4.22. l is given by equation 4.4.

$$l = \frac{x}{cos(\alpha)} \tag{4.4}$$

where

$$\alpha = tan^{-1}(\frac{g_{max} - g_{min}}{L}) \tag{4.5}$$

(4.4) and (4.5) are inserted into (4.4) and this gives equation 4.6.

$$F_x = \frac{1}{2}V^2\epsilon t\frac{1}{cos\left(tan^{-1}(\frac{g_{max}-g_{min}}{L})\right)}\frac{g_{max}}{g_{max}^2 - 2g_{max}\frac{g_{max}-g_{min}}{L}x + \left(\frac{g_{max}-g_{min}}{L}\right)^2 x^2} \tag{4.6}$$

When $g_{max}-g_{min} = 0$ is used, this equation reduces to equation 4.2. At low displacement, sim-

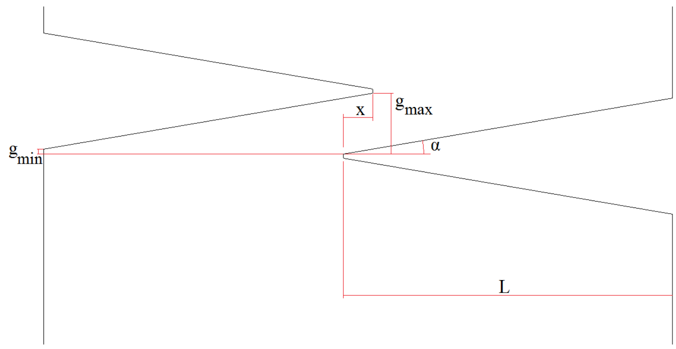


Figure 4.22: Variables used in equation 4.3

ulation method 1 gives approximately the same result as the theoretical calculation. Method 4 and 5 give approximately 30% more force then the theoretical value while method 2 and 3 give more then twice the theoretical force.

	D	C	B	A	χ^2
1	2,4292E-09	-5,6179E-08	2,6932E-06	9,6950E-05	9,9959E-01
2	3,0945E-09	-7,9017E-08	4,0168E-06	2,1739E-04	9,9983E-01
3	2,6756E-09	2,6968E-09	5,6134E-06	2,2689E-04	9,9993E-01
4	2,8715E-09	-7,9969E-08	3,3806E-06	1,3492E-04	9,9979E-01
5	5,8844E-10	1,8898E-08	2,0316E-06	1,2840E-04	9,9904E-01
Theory	1,3538E-09	2,5407E-08	2,3084E-06	9,4947E-05	1,0000E+00

Table 4.12: Coefficients of the fitted polynomials for Sim_ID8

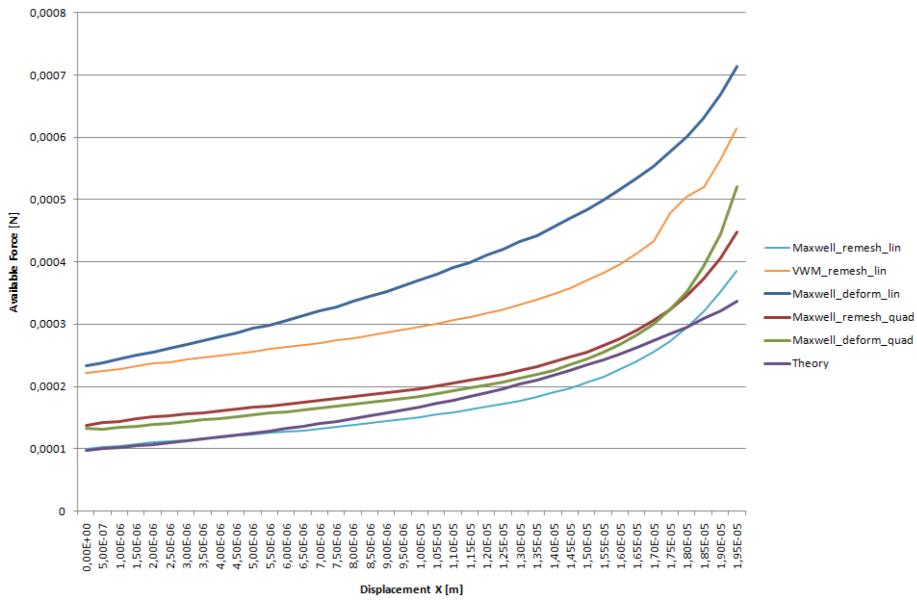


Figure 4.23: Simulation results of Sim_ID 8

Sim_ID 9

ID 9 is the version of Jensen that has a linear force profile and one comb with uniform finger width. The results from the simulations are compared to the analytical and simulation results from Jensen in [3]. The model used for the simulations is shown in Figure 4.12(b). This can be seen in Figure 4.25. The legend of this figure is shown in Figure 4.24. The results show

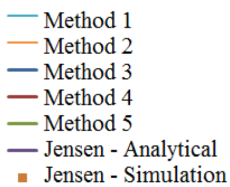


Figure 4.24: Legend of the figures with the results of Sim_ID 9 through 13.

that Jensens analytical calculations show a very linear behavior at high displacement, their simulation results are a lot less linear. The simulations results obtained using method 2 and 4 show results comparable to Jensens simulation results. However, simulation methods 1 and 3 show alternating behavior. This is because the results shown is an average of the different mesh refinements and these methods had a very low success rate, which means that each result that was calculated had a large impact on the average. Simulation method 5 had an extremely low success rate, results were only found for very low displacement for all mesh refinement presets. The parameters of the polynomial fit are shown in Table 4.13.

Sim_ID 10

ID 10 is the version of Jensen that has a cubic force profile and one comb with uniform finger width. The model used for the simulations is shown in Figure 4.12(d). The results from the

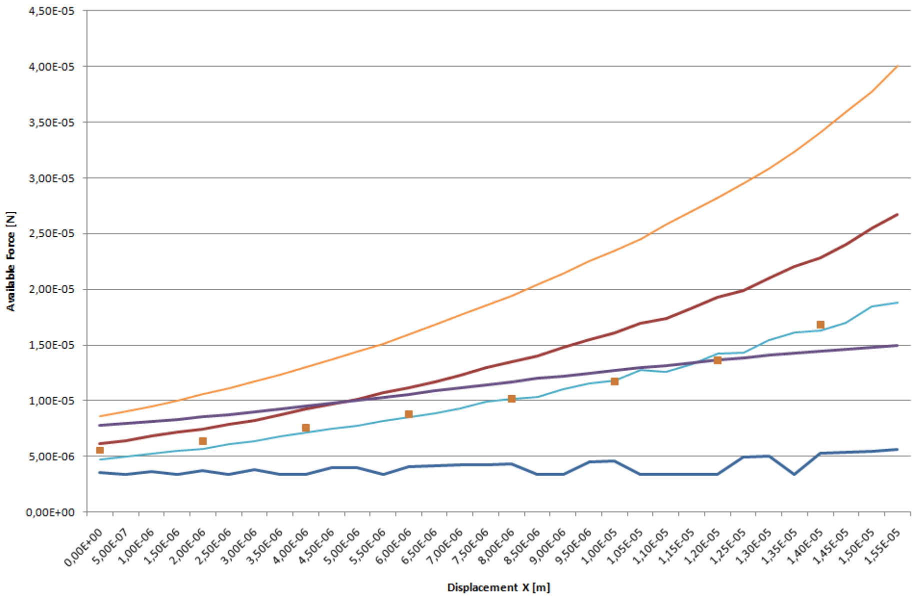


Figure 4.25: Simulation results of Sim_ID 9

	D	C	B	A	χ^2
1			3,2863E-07	4,2124E-06	9,9440E-01
2			6,4623E-07	7,4770E-06	9,9104E-01
3			4,5746E-08	3,3458E-06	4,3721E-01
4			4,4201E-07	5,3856E-06	9,9281E-01
5					
Jensen - analytical			4,8998E-07	1,4767E-05	9,9618E-01
Jensen - simulation			5,4878E-07	1,0290E-05	9,9246E-01

Table 4.13: Coefficients of the fitted polynomials for Sim_ID9

simulations are compared to the analytical and simulation results from Jensen in [3]. This can be seen in Figure 4.26. The legend of this figure is shown in Figure 4.24. The results

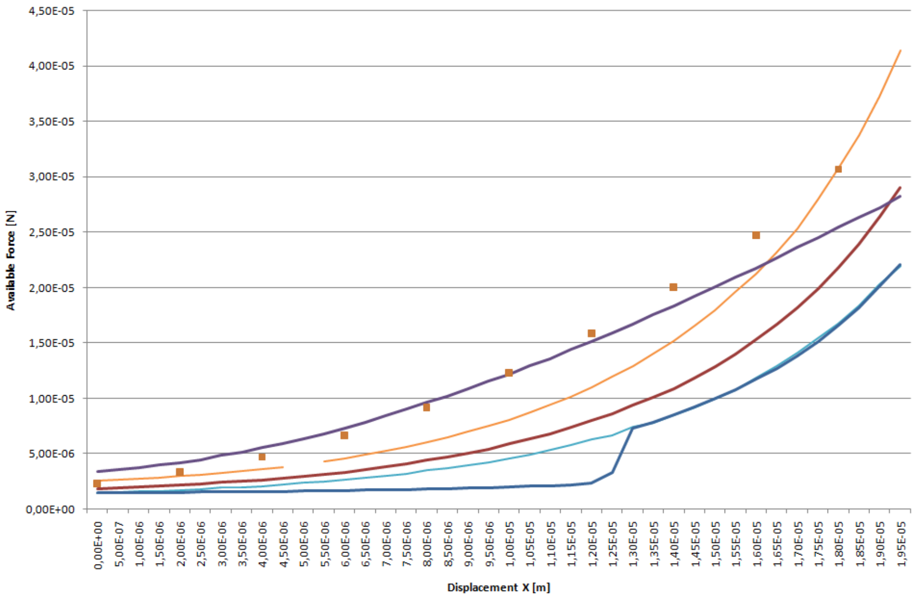


Figure 4.26: Simulation results of Sim_ID 10

show that Jensens analytical calculations show a very linear behavior at high displacement, their simulation results do not show this linear behavior. Except for method 3, the simulation results show alike behavior. Method 3 shows a 'jump', this is because before 12μm, all mesh refinement presets gave results, while after 13μm, only preset 1 gave results. When only the results from preset 1 is used, they are almost equal to the average results of simulation method 1. Inspection of the simulations showed that the mesh quality was rather low when other presets were used then 1. As a result, Comsol could not solve the FEM properly. This can also be seen from the needed simulation time. Mesh preset 8 did not give any results, but preset 2 through 7 needed a lot of time, meaning that Comsol needed to do a lot of iterations before it found results that were acceptable. The second 'fastest' preset needed nearly 7 times as much calculation time then preset 1. Mesh preset 2 even took over 50 times as long as preset 1. Simulation method 5 did not find a solution for any of the mesh presets. The parameters of the polynomial fit of the results are shown in Table 4.14.

	D	C	B	A	χ^2
1	2,1880E-10	-4,0333E-10	6,4593E-08	1,3847E-06	9,9958E-01
2	3,9240E-10	-4,0612E-10	1,0506E-07	2,4060E-06	9,9996E-01
3	3,0960E-11	4,1879E-11	1,0879E-08	1,4511E-06	9,9998E-01
4	2,8120E-10	-3,8529E-10	7,9433E-08	1,7508E-06	9,9992E-01
5					
Jensen - analytical	-1,3849E-10	1,8803E-08	9,0988E-08	3,2777E-06	9,9991E-01
Jensen - simulation	1,7487E-10	1,1043E-08	1,7558E-07	2,1223E-06	9,9996E-01

Table 4.14: Coefficients of the fitted polynomials for Sim_ID10

Sim_ID 11

ID 11 is the version of Jensen that has a linear force profile and two combs with shaped finger width. The model used for the simulations is shown in Figure 4.12(f). The results from the simulations are compared to the analytical and simulation results from Jensen in [3]. This can be seen in Figure 4.27. The legend of this figure is shown in Figure 4.24. The results of Jensen have been divided by 4 to get the results below. Again, method 3 gives strange

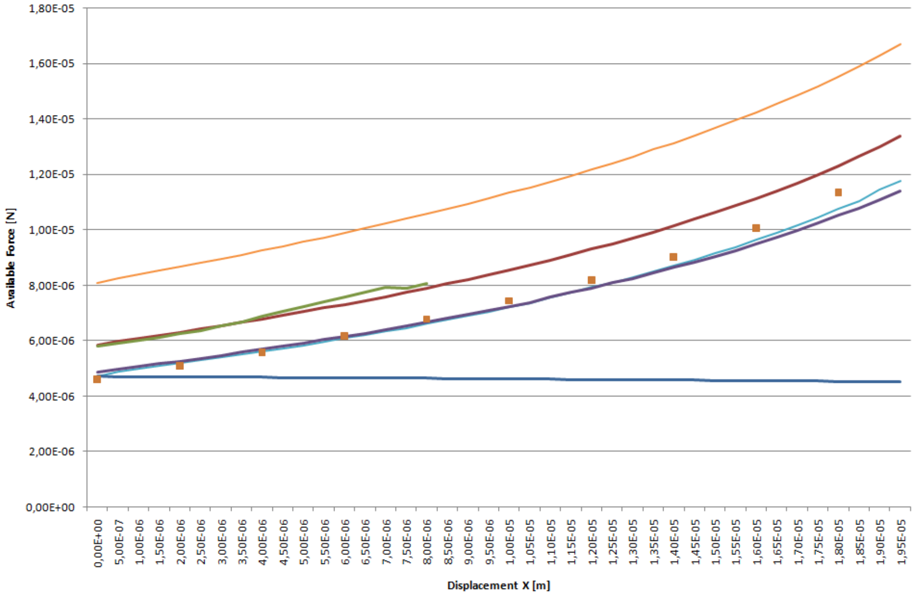


Figure 4.27: Simulation results of Sim_ID 11

results, while method 5 barely gives any results. The analytical results of Jensen is very close to the results from method 1 when Jensens results are divided by 4. Methods 2 and 4 produce results in the same shape as method 1. These similarities can be seen from the parameters of the polynomial fit that are shown in Table 4.15. The value for parameter B has approximately the same value for all the results except for method 3.

	D	C	B	A	χ^2
1			1,2415E-07	4,5401E-06	9,9389E-01
2			1,6320E-07	7,8428E-06	9,9644E-01
3			-4,6259E-09	4,7183E-06	9,9870E-01
4			1,3746E-07	5,6030E-06	9,9471E-01
5			1,5253E-07	5,5356E-06	9,9291E-01
Jensen - analytical			1,2081E-07	4,6513E-06	9,9423E-01
Jensen - simulation			1,4183E-07	4,3804E-06	9,9614E-01

Table 4.15: Coefficients of the fitted polynomials for Sim_ID11

Sim_ID 12

ID 12 is the version of Jensen that has a cubic force profile and two comb with shaped finger width. The model used for the simulations is shown in Figure 4.12(h). The results from the simulations are compared to the analytical and simulation results from Jensen in [3]. This can be seen in Figure 4.28. The legend of this figure is shown in Figure 4.24. The results of Jensen have been divided by 4 to get the results below. Simulation method 3 again gives

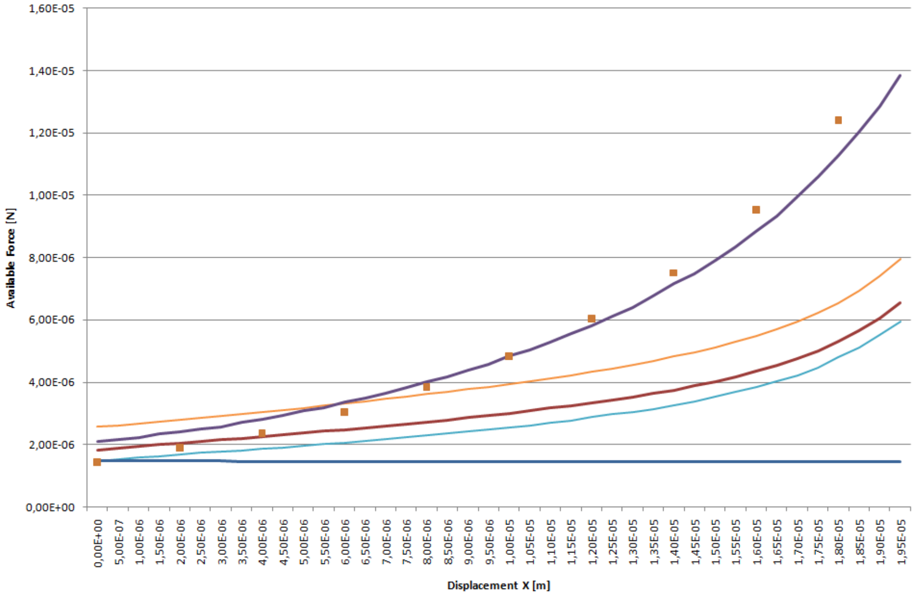


Figure 4.28: Simulation results of Sim_ID 12

different results from the rest and method 5 again does not give results. Method 1, 2 and 4 give results that are alike, but with a different offset. This is shown again by the parameters of the polynomial fit (see Table 4.16) which are very close together. The force profile should be cubic, but it seems from the simulation results that it is rather linear for low displacement.

	D	C	B	A	χ^2
1	4,4887E-11	-9,6051E-10	5,4183E-08	1,4238E-06	9,9977E-01
2	1,2692E-11	3,5536E-06	5,5525E-08	2,5072E-06	9,9996E-01
3	5,1166E-13	1,2900E-11	-1,9249E-09	1,4793E-06	1,0000E+00
4	3,9511E-11	-8,3249E-10	5,9171E-08	1,7654E-06	9,9996E-01
5					
Jensen - analytical	5,3764E-11	2,1749E-09	6,3446E-08	2,0282E-06	9,9986E-01
Jensen - simulation	6,0160E-11	2,4166E-09	8,8267E-08	1,3535E-06	9,9976E-01

Table 4.16: Coefficients of the fitted polynomials for Sim_ID12

Sim_ID 13

Jensen et al. designed simulation ID 13 to get a linear force profile with an increased force compared to the ones they copied from Ye et al. The model used for the simulations is shown in Figure 4.12(j). The results from the simulations are compared to the analytical and simulation results from Jensen in [3]. This can be seen in Figure 4.29. The legend of this figure is shown in Figure 4.24. The results of Jensen have been divided by 4 to get the results below. The figure shows that simulation methods 1 and 3 have a strange behavior,

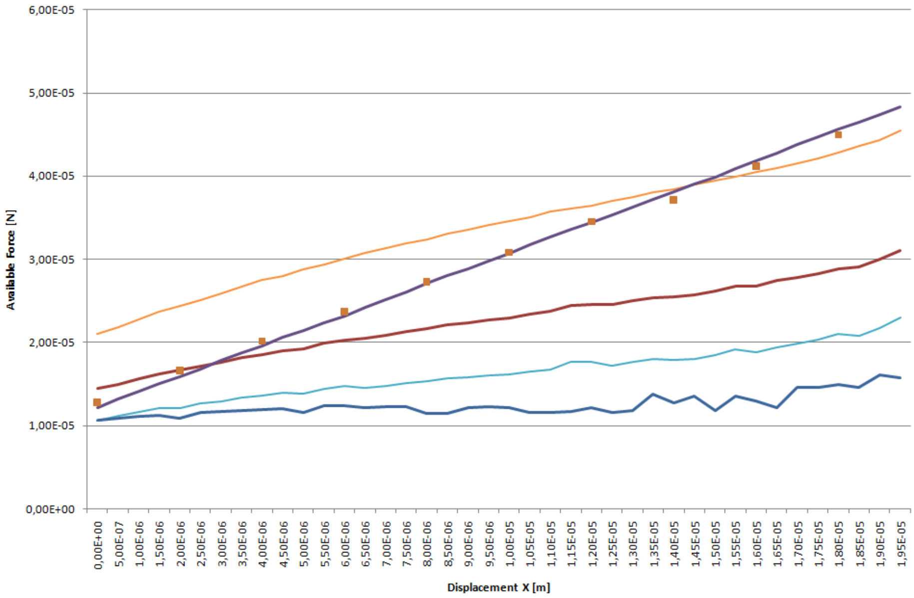


Figure 4.29: Simulation results of Sim_ID 13

this is again because they had a low success rate. Method 2 shows a smooth behavior, but is not very linear. Method 4 shows a smooth linear behavior, however, it is not as steep as the results from Jensen. Method 5 did not produce any results again. The parameters of the polynomial fit are shown in Table 4.17. This geometry was meant to have a higher force then Sim_ID 9 and 11. It turns out that at low displacements, the force is more then three times higher then that of Sim_ID 9, which is reduced to a bit more then one at high displacement. The force is about 2.5 times higher then that of Sim_ID 11 for the complete displacement.

	D	C	B	A	χ^2
1			2,5797E-07	1,0991E-05	9,7816E-01
2			6,6022E-07	2,1096E-05	9,9319E-01
3			4,5937E-08	1,1166E-05	3,4910E-01
4			4,1125E-07	1,4607E-05	9,9226E-01
5					
Jensen - analytical			9,2606E-07	1,1297E-05	9,9996E-01
Jensen - simulation			8,9749E-07	1,2022E-05	9,9985E-01

Table 4.17: Coefficients of the fitted polynomials for Sim_ID13

Conclusion

In each of the above results, the different simulation methods gave different results. For simulation id 1 through 3 (the ones with one comb with shaped fingers and one with straight fingers), method 2 always gives the highest result, while method 3 always has the lowest result at low displacement. At high displacement, method 1 always gives the lowest results. The order of results between is always in the same order as well. Method 1, 3, 4 and 5 are close together over the complete displacement. Simulation id 4 through 8 (the ones with combs with identical shaped fingers) also show a similar order (with the exception of method 6 at high displacement). Method 3 is always the highest while method 1 always gives the lowest results. Method 2 is second highest. Method 4 is higher then method 5 at low displacement, but method 5 gives a higher result at high displacement. Method 6 starts out in the same order, but is mixed up at higher displacement. Method 4 and 5 are usually very close together, while the other methods are spread out more. The geometries that were derived from Jensen (simulation id 9 through 13) had a set order which does not change with the displacement. The highest result always was method 2, followed by 4, then 1 and method 3 was always lowest. The only time method 5 gave results, it was very close to method 4. Why a certain kind of geometry has a certain order is not clear.

4.5.2 Analysis of the simulations

Simulation time

A lot of factors influence the processing time needed for the simulations. Below, the average simulation time for the different methods and mesh presets are compared. Figure 4.30 and 4.31 show the simulation time needed for the simulation of id 1 and 2. The other ids show similar behavior. The legend is shown in table 4.18. Simulation method 2 is not displayed separately because it is done during the same run as method 1.

Since the post-processing time for the different simulation methods varied greatly, each plot shows a graph with and without the post-processing time.

Simulation Method	Pure simulation time	Time incl. post-processing
1		
3		
4		
5		

Table 4.18: Legend for the time-simulation plots

The first thing that is noticed from the figures below, is that the simulation times of simulation methods 3 and 5 (the ones with the deformed mesh) have weird peaks. The reason behind this is, that some mesh presets generated unsolvable meshes at high displacement. Comsol Multiphysics tries several different ways to eventually still solve the mesh which takes a long time. When the mesh was regenerated for each displacement step, it is much easier for Comsol to solve the mesh resulting in less peaks. Methods 1 and 4 also show much more clearly how the simulation time increases for finer meshes.

If the peaks for simulation methods 3 and 5 are ignored, the simulation time is generally lower then for methods 1 and 4. This is because the meshing alone already takes considerable time. The methods that use a deformed mesh only generate one mesh which is adapted to the changed geometry, the methods that use remeshing have to generate a new mesh for each displacement step. This means that more of the simulation time is spend on meshing and also that more data is generated and thus more date needs to be saved.

The simulation run was done in several steps. First, all the different geometries were simulated and the results saved to disk. After that the relevant information was extracted from the saved files in the first post-processing step. The last step sorted all the data and saved in such a way that it can be used for presentation. The first post-processing step requires a lot of available physical memory (RAM). The amount needed for the post-processing of the data of methods 1 and 4 turned out to be more then was available and the simulation results had to be loaded in steps. This further increased the loading time since a large file had to be accessed several time for each simulation, something that takes a lot of time in MatLab. The post-processing time for methods 1 and 4 can be reduced a lot if a workstation would be available with more available physical memory so that the complete simulation results can be loaded at once.

The figures also show that the simulations using linear Lagrange elements take less time then their equivalent using quadratic Lagrange elements. Since quadratic elements uses more and more complex calculations, this was to be expected.

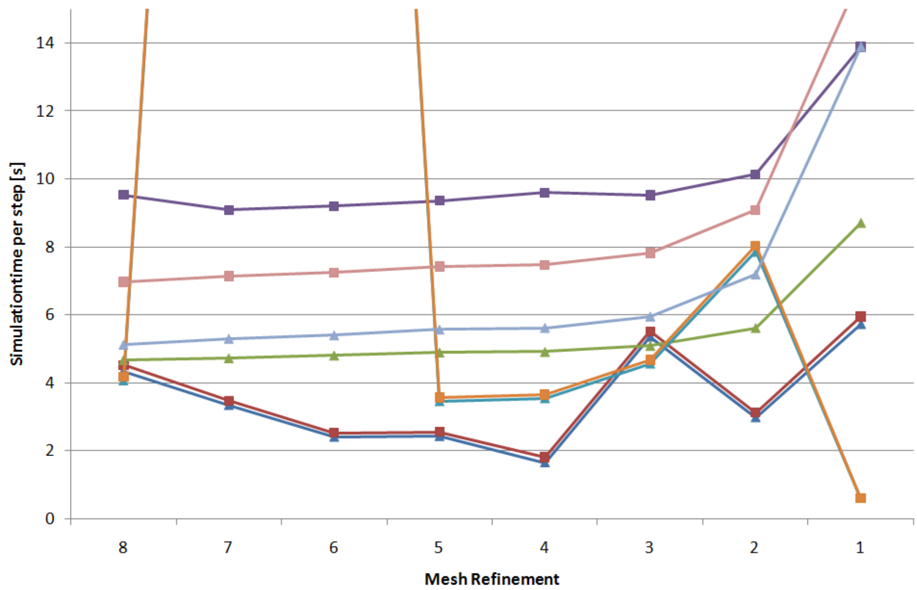


Figure 4.30: Simulation time per displacement step (of 0.5μm) for the different simulation methods for Sim_ID 1

Different amount of fingers

When a comb-drive is used on a chip, it usually has a lot of fingers. Simulating those amounts of fingers would need huge amounts of computing power and would thus take to long. To be able to decrease the amount of fingers, the effects of the outer fingers need to be small. During

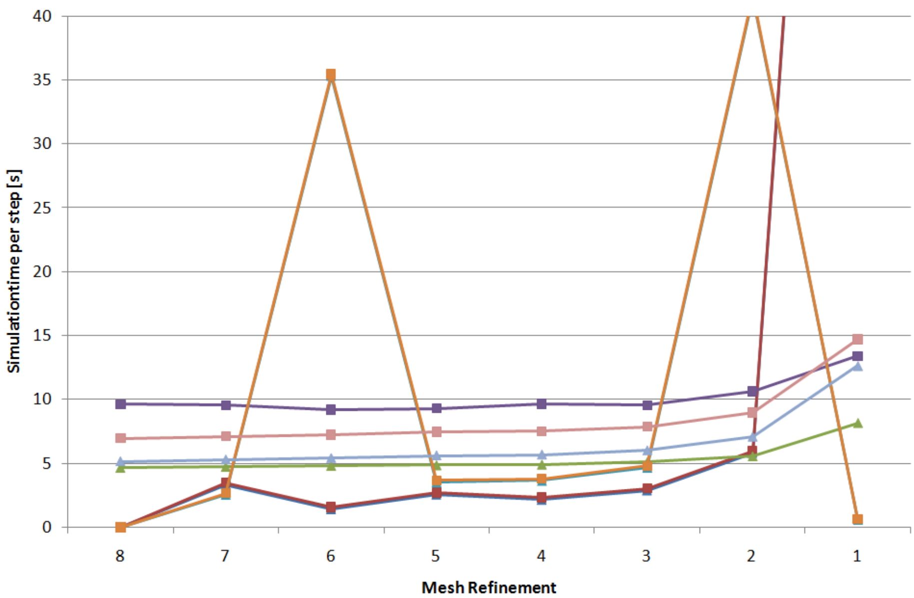


Figure 4.31: Simulation time per displacement step (of 0.5μm) for the different simulation methods for Sim_ID 2

the simulations the amount of capacitors in the comb-drive was five. To find the influence of the amount of fingers, simulations have been done for Sim_ID 1 through 8 with 9 capacitors. This means that the ratio between the two forces should be $\frac{9}{5} = 1.8$. Figure 4.32 shows the ratio of the average results per simulation ID when they were simulated using simulation method 1. As can be seen, all the ratio's are within 5% of 1.8, so it can be assumed that the effects of the end of the comb-drive are minimal. The other simulation methods gave similar results.

Errors during simulation

Accuracy of the results and simulation time are not the only things that matter when looking for a reliable simulation method. During the simulations, several different errors occurred at different settings. There errors terminated the simulation or resulted in incomplete results. The simulation methods that used remeshing mainly had an error where Comsol could not make a valid mesh of the geometry. The simulation methods that used a deformed mesh mainly had an error that was caused by a mesh with to much deformation.

To get an idea how likely it is that a simulation method gives a complete simulation result, the amount of displacements steps that were finished with each method for each mesh preset was divided by the total amount of displacement steps that would be done at a full simulation. During the simulation the combs moved over 20μm with 40 steps of 0.5μm, giving a total amount of steps of $40 \cdot 13 = 520$ per setting. An overview of the success rates is presented in table 4.19. Again method 1 and 2 are taken together because only the post-processing is different.

Especially the methods that used a deformed mesh had problems finishing the simulations. When linear Lagrange elements were used, the percentage of finished simulations is less then 80% for most mesh presets. Quadratic Lagrange elements show a big difference between the

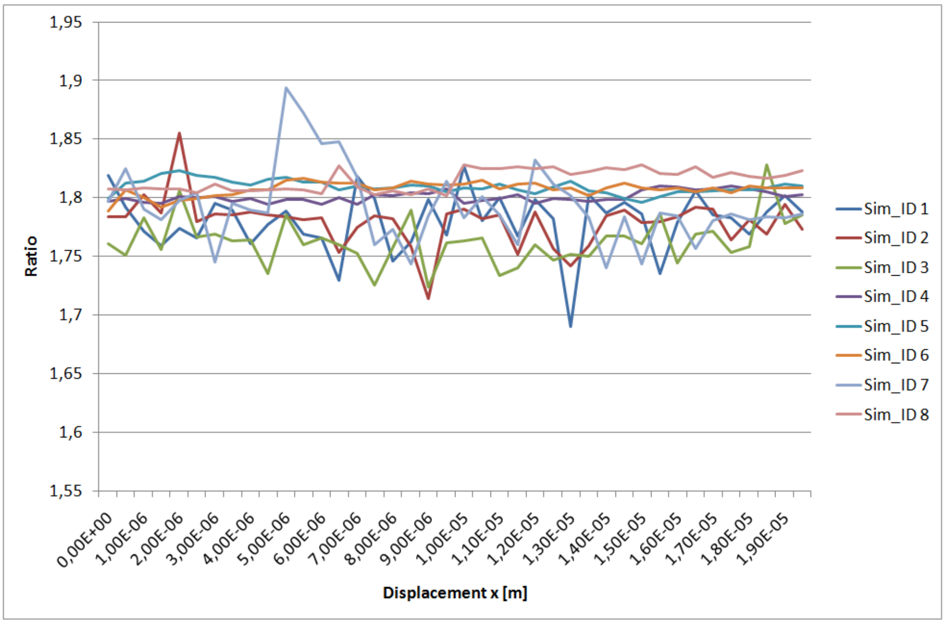


Figure 4.32: Ratio between the simulation results with 9 and 5 capacitors.

Mesh Preset	Method 1	Method 3	Method 4	Method 5
1	0,998	0,852	0,998	0,615
2	0,981	0,883	0,981	0,564
3	0,977	0,785	0,977	0,544
4	0,939	0,789	0,939	0,647
5	0,948	0,841	0,948	0,646
6	0,885	0,789	0,885	0,502
7	0,873	0,773	0,873	0,533
8	0,820	0,730	0,803	0,598

Table 4.19: Rates of successful simulation steps

different ID's. If only the geometries from Ye are evaluated, method 5 has a success rate of over 90%, for the geometries from Jensen it is less then 20%. The error that occurred during these simulations was mainly because the mesh had deformed to much and Comsol could not find reliable results. This big difference is most likely caused by the smaller gap that was used by Jensen et al.

The errors that occurred during simulations using method 1 and 4 were caused during meshing. This explains why the success rates are the same for both methods: the simulation process was the same up till the point where the error occurred.

4.5.3 Conclusion

An attempt has been made to recreate several finger shapes derived from papers. However, making an exact copy was very hard. Making a geometry in Comsol can only be done using certain kinds of shapes. Furthermore, the bounding box prevented fingers from opposite combs from coming to close to each other, while the geometries in the papers required that in certain cases.

Simulation ID 1 through 6 were based on shapes found by Ye et al. [1] They analytically calculated some of those shapes and used simulations to find the force profile of other shapes. The results from the simulations presented here are like the ones given by Ye, but there is still a large mismatch. The paper did not supply all the information that was needed for the the simulations and some of the information had to be read from small graphs. An attempt was made to re-create the geometries as accurate as possible, but they might not have been accurate enough.

Simulation ID 7 and 8 were compared to a analytically calculated force profile. Some of the simulation methods gave good results for low displacement for these geometries. At high displacement, the simulation results were different from the analytical model. This is because the analytical model does not take the extra capacities into account that have an effect when the fingers approach the base of the opposite comb.

Simulation ID 9 through 13 were based on shapes found by Jensen et al. [3] They define their shapes a lot better then Ye did and are more easily reproduced. The simulation results are very close to the results presented by Jensen. However, Jensens results were a factor four higher then the results from the simulations for ids 11, 12 and 13.

The geometries that were used during the simulations originally had round shapes. In Comsol, they were approximated by 13 straight boundaries. Increasing this number or finding a way to implement the round boundaries in Comsol would increase the quality of the model.

The different simulation methods gave different values for the force, however, the force profile was approximately the same with all methods. The most reliable methods seem to be method 1 and 4 in combination with a fine mesh: use the Maxwell stress tensor to find the force and make a new mesh after each displacement step.

Chapter 5

Scripts

Comsol can communicate with MatLab so that scripts can be used to perform simulations and do post-processing on the simulated data. The scripts that are used can be found in Appendix A.

The simulation scripts are roughly divided in three groups: execution scripts, simulation scripts and analyze scripts.

5.1 Execution Scripts

The 'Execution scripts' are used to run several of the other scripts after each other and to set constants. The 'main' script, `Run_All.m` (A.1.1) is used to set all the constants that are needed to run the simulations and to make a database of available finger shapes.

The second execution script is `Analyse_All.m` (A.1.2). This script executes the different scripts that are used to analyze the simulation results and it can make an overview of the simulation-runs that have been done.

5.2 Simulation Scripts

This group has all the scripts that are actually needed for the simulation. The first one that is executed is `Create_Sim_DB.m`. This creates a matrix with all the finger shapes that can be simulated. Using a variable, the database that is created consists of the finger shapes that are given in Table 4.1 or a list of all possible shapes that can be made using the constants set in `Run_All.m`. The result of this script is a file called `sim_db.mat` which contains the matrix.

The scripts that make the geometry and do the actual simulation are `Do_Sim_lin.m` (A.2.2) and `Do_Sim_quad.m` (A.2.3. The first does the simulations using linear Lagrange elements while the second uses quadratic Lagrange elements. The result of the simulations are saved in `.mat` files in a sub-folder. A unique name is generated for this sub-folder so that each simulation run is stored in a different place so that it will not be overwritten by a later simulation run.

The geometry that is to be simulated is build automatically. This is done with the help of the function `Build_Finger.m` (A.2.4) that creates one half of a finger. The boundary/subdomain settings that Comsol uses during the simulation are set by the function `getapplmodes` (A.2.5). This function returns a vector with the right values that is selected based on the constants set in `Run_All.m`.

5.3 Analyzing Scripts

The simulation results need to be analyzed so that the relevant information can be used. To do this, several different scripts are available that `Analyse_All` can execute.

The first script that can be executed is `Analyse_Output_remesh.m`. This script extracts the data from the `.mat` files that are the result of the simulation. Among the information that is extracted are the forces, the steps over which the lower comb of the geometry moves and the time it took to simulate. All this information is stored in the file `Result_Analyse.mat`. `Analyse_Result_Analyse_mat.m` (A.3.3) then takes this data and makes it into data that can be easily read. Among the results are several Excel sheets that put this data into plots that can be easily manipulated.

The script `Analyse_CPUTime.m` (A.3.1) gives a graphical overview of the data about the simulation time. `Analyse_SimQ.m` (A.3.4) checks the quality of the simulation by finding the amount of successfully completed simulations compared to the total simulations that needed to be done.

Finally the script `Make_Overview.m` (A.3.5) makes an overview of the simulations that have been done by making a matrix with the values of the constants that have been set by `Run_All.m` for each simulation run.

Chapter 6

Conclusion

A set of scripts has been made to simulate comb-drive with shaped fingers using MatLab and Comsol Multiphysics. Several different methods for the simulations have been analyzed and compared to analytical calculations. The simulations focused on a set of 13 different geometries. Six of these were taken from a paper by Ye et al. [1] who analytically calculated which shapes were needed for linear, quadratic and cubic force-profiles. Five were taken from a paper by Jensen et al [3] who tried to do the same as Ye et al. The two remaining geometries were a comb-drive with straight fingers and a comb-drive with tapered fingers. These two were compared to analytical calculations.

The comparison with Ye et al. could not successfully be made. Not all the required data was given in their paper which resulted in different simulation results. The comparison to Jensen's geometries was a lot more successful. The shape of the force profile matched in most cases, but the exact values were still different with a factor up to 10. A comparison between analytical values and simulations of standard comb-drive shapes (uniform finger width and tapered fingers) shows that the simulation result are close to the theoretical value and have approximately the same force profile. Simulation method 2 has results within 10% from the theoretical value for Sim_ID 7 except at high displacement and method 1 stays within 10% of the theoretical value for Sim_ID 8 for low displacements. The simulated values are different from the theoretical values at high displacements because the analytical model does not take the base of the opposite comb into account when the force is calculated. An explanation for the differences between the results from the papers and the results from the simulations can be that the geometries were not copied exact enough. Better results might be achieved if the geometries could be improved, however, the simulation process might need to be adjusted to be able to do this.

Five different methods have been used to do the simulations and the two methods that are most reliable use the Maxwell Stress Tensor and do not use a deformed mesh. A fine mesh gave a successful simulation result in 99.8% of the displacement steps when these methods were used. However, the different methods did not give the same values for the force with a differense up to a factor of 3, they do give the same shape to the force profile. Why this is and how an accurate result can be found needs further investigation.

Bibliography

[1] W. Ye, S. Mukherjee, and N.C. Macdonald. Optimal shape desisgn of an electrostatic comb drive in microelectromechanical systems. *Journal of microelectromechanical systems*, 7(1), 1998.

[2] S. Schonhardt, J.G. Korvink, J. Mohr, U. Hollenbach, and U. Wallrabe. Magnetic comb drive actuator. In *Micro Electro Mechanical Systems, 2008. MEMS 2008. IEEE 21st International Conference on*, pages 479–482, Jan. 2008.

[3] B.D. Jensen, S. Mutlu, S. Miller, K. Kurabayashi, and J.J. Allen. Shaped comb fingers for tailored electromechanical restoring force. *Microelectromechanical Systems, Journal of*, 12(3):373–383, June 2003.

[4] *Comsol Multiphysics Manual*.

[5] J.B.C. Engelen, H.E. Rothuizen, U. Drechsler, R. Stutz, M. Despont, L. Abelmann, and M.A. Lantz. A mass-balanced through-wafer electrostatic x/y-scanner for probe data storage. *Microelectronic Engineering*, In Press, Corrected Proof:–, 2008.

Appendix A

MatLab scripts

A.1 Execution Scripst

A.1.1 Run_All

```
close all;
clear all;
disp('Starting Run_All');
reset = 1;
do_Create_Sim_DB = 1;
do_Do_Sim = 1;
do_Analyse_All = 1;
do_Ye1998 = 0;
save do_what.mat do_*
if reset == 1
    disp('Defining and saving constants');
    reset = 0;
    run_counter = 0;
    date_of_start = date;   %Saving the date
    finger_length = 30e-6;  %Length of the fingers (y-direction)
    finger_width = 1e-6;    %Width of the fingers (x-direction)
    amount_of_fingers = 3;  %Amount of fingers to be simulated
    init_foot_thickness = 10e-6;    %Thickness at the foot of the finger

    uniform_fingers = 1;    %Are the static fingers uniform in width

    step_x = 1e-6;  %Change per step in x-direction in amounts of 0.1um
    steps_x = 3;    %Amount of steps in x-direction
    steps_y = 13; %Amount of steps in y-direction
    step_y = finger_length/steps_y;  %Stepsize in y-direction
    max_change = 3; %Amount of difference in steps in change (x-direction)

    min_gap = 1e-6; %Minimum width of the gap
    max_gap = min_gap + 2*steps_x*step_x;   %Maximum gap width

    init_x = 5e-6;  %Initial overlap of the fingers
    sim_step = 1e-6;    %Stepsize of the comb-movement in y-dircetion
    end_x = finger_length-step_y;  %Maximum overlap of the fingers
    sim_steps = floor((end_x-init_x)/(sim_step));  %Steps in the simulation
    num_sims = steps_x^steps_y; %Amount of different possibilities
```

```
    sim_plus_voltage = 10;   %Voltage on the positive comb
    sim_min_voltage = 0;    %Voltage on the negative comb (Grounded)
    save sim_const.mat run_counter date_of_start finger_length ...
        finger_width amount_of_fingers init_foot_thickness step_x ...
        steps_x step_y steps_y max_change min_gap max_gap init_x ...
        sim_step end_x sim_steps num_sims sim_plus_voltage ...
        sim_min_voltage uniform_fingers
else
    disp('Loading constants');
    load('sim_const.mat');
end

if exist('Temp','dir') == 0
    mkdir('Temp')
end

while run_counter ≥0
    if run_counter == 0
        load('do_what.mat');
        if do_Create_Sim_DB == 1
            % Create a database with the finger shapes that need to be
            % simulated
            Create_Sim_DB
        end
        run_counter = 1;
        save sim_const.mat run_counter date_of_start finger_length ...
        finger_width amount_of_fingers init_foot_thickness step_x ...
        steps_x step_y steps_y max_change min_gap max_gap init_x ...
        sim_step end_x sim_steps num_sims sim_plus_voltage ...
        sim_min_voltage uniform_fingers
    end
    if run_counter == 1
        load('do_what.mat');
        % Do the actual simulations, the shapes are taken from sim_db.mat
        if do_Do_Sim == 1
            %Linear or quadratic Lagrange elements
            Do_Sim_quad
            %Do_Sim_lin
        end
        run_counter = 2;
        save sim_const.mat run_counter date_of_start finger_length ...
        finger_width amount_of_fingers init_foot_thickness step_x ...
        steps_x step_y steps_y max_change min_gap max_gap init_x ...
        sim_step end_x sim_steps num_sims sim_plus_voltage ...
        sim_min_voltage uniform_fingers
    end
    if run_counter == 2
        load('do_what.mat');
        % Analyse the simulation results
        if do_Analyse_All == 1
            Analyse_All
        end
        run_counter = 3;
        save sim_const.mat run_counter date_of_start finger_length ...
        finger_width amount_of_fingers init_foot_thickness step_x ...
        steps_x step_y steps_y max_change min_gap max_gap init_x ...
```

```
        sim_step end_x sim_steps num_sims sim_plus_voltage ...
        sim_min_voltage uniform_fingers
        run_counter = -1;
    end

end
run_counter = 0;
save sim_const.mat run_counter date_of_start finger_length ...
        finger_width amount_of_fingers init_foot_thickness step_x ...
        steps_x step_y steps_y max_change min_gap max_gap init_x ...
        sim_step end_x sim_steps num_sims sim_plus_voltage ...
        sim_min_voltage uniform_fingers
```

A.1.2 Analyse_All

```
clear all
close all
%Clear temp files from previous runs
delete('Temp/*.*');

disp('Starting Analyse_All.m');
dir_path = 'Results';
filelist = dir(dir_path);
dirlistcounter = 0;
%Generate a list of directories with results
for i = 1:length(filelist)
    if filelist(i).isdir == 1
        dirlistcounter = dirlistcounter+1;
        dirlist(dirlistcounter) = filelist(i);
    end
end

%Variables to force a re-analysis
force_Analyse_Output_remesh = 0;
force_Analyse_Result_Analyse_mat = 0;
force_Analyse_CPUTime = 0;
force_Analyse_SimQ = 0;
force_Make_Overview = 0;
force_New_Overview = 0;
only_do_last = 1;
clear dirlistcounter
%Start at 3 because of /. and /.. directories
dirlist_vector = 3:length(dirlist);
if only_do_last == 1
    highest_counter = 0;
    for i = 1:length(dirlist_vector)
        dir_name1 = [dir_path '/' dirlist(dirlist_vector(i)).name];
        load([dir_name1 '/counter.mat'], 'simulation_number');
        if simulation_number > highest_counter
            highest_counter = simulation_number;
            highest_id = i;
        end
    end
    tempdir = dirlist_vector(highest_id);
    clear dirlist_vector
```

```
    dirlist_vector = tempdir;
    clear tempdir highest_counter simulation_number
end

%Give directories that should be excluded from analysis if needed
%At least one should be given
dir_Excluded(1) = {[dir_path '/_not_needed_now']};

timer = zeros(5,length(dirlist_vector));
stop_counter = length(dirlist_vector);
for repeat_counter = 1:stop_counter
    timer_Start1 = tic;
    dir_name1 = [dir_path '/' dirlist(dirlist_vector(repeat_counter)).name];
    dir_identifier=strrep(dirlist(dirlist_vector(repeat_counter)).name, ...
        'Simulation results - ','');
    disp(['Analysing dir ' dir_name1]);

    %Check to see if the directory should be analysed or not
    dont_analyse = 0;
    for i = 1:length(dir_Excluded(:))
        if strcmp(dir_name1, dir_Excluded(i))==1
            dont_analyse = 1;
        end
    end
    if dont_analyse==0
        %Check if all the proper directories exist
        dir_name2 = [dir_name1 '/Figures'];
        dir_name3 = [dir_name1 '/Models'];
        if (exist(dir_name1, 'dir') == 0)
            mkdir(dir_name1);
        end
        if (exist(dir_name2, 'dir') == 0)
            mkdir(dir_name2);
        end
        if (exist(dir_name3, 'dir') == 0)
            mkdir(dir_name3);
        end
        %Save all the relevant settings to be reloaded after the clear all
        save('Temp/Analyse_All_temp.mat', 'dir*', 'repeat_counter', ...
            'stop_counter', 'force_*', 'timer*');

        if exist([dir_name1 '/Result_Analyse.mat'],'file') == 0 || ...
                force_Analyse_Output_remesh == 1
            timer_Start2 = tic;
            save('Temp/Analyse_All_temp.mat', 'dir*', 'repeat_counter', ...
            'stop_counter', 'force_*', 'timer*');
            clear all
            close all
            %Analyse the simulated fem-structures from dir_name1/Models
            load('Temp/Analyse_All_temp.mat');
            Analyse_Output_remesh
            clear all
            close all
            load('Temp/Analyse_All_temp.mat');
            timer(2,repeat_counter) = toc(timer_Start2);
            disp(['Finished in: ' num2str(timer(2,repeat_counter)) 'sec']);
        end
```

```
        if exist([dir_name1 '/Result_Analyse.xls'],'file') == 0 || ...
                force_Analyse_Result_Analyse_mat == 1
            timer_Start3 = tic;
            save('Temp/Analyse_All_temp.mat', 'dir*', 'repeat_counter', ...
            'stop_counter', 'force_*', 'timer*');
            clear all
            close all
            %Analyse the collected results to make plots and xls sheets
            load('Temp/Analyse_All_temp.mat');
            if force_Analyse_Result_Analyse_mat == 1 && repeat_counter == 1
                %delete previous results to rebuild the result-database
                delete([dir_name1 '/../Results.mat']);
            end
            Analyse_Result_Analyse_mat
            clear all
            close all
            load('Temp/Analyse_All_temp.mat');
            timer(3,repeat_counter) = toc(timer_Start3);
            disp(['Finished in: ' num2str(timer(3,repeat_counter)) 'sec']);
        end
        if exist([dir_name1 '/TimeSave.xls'],'file') == 0 || ...
                force_Analyse_CPUTime == 1
            timer_Start4 = tic;
            save('Temp/Analyse_All_temp.mat', 'dir*', 'repeat_counter', ...
            'stop_counter', 'force_*', 'timer*');
            clear all
            close all
            %Collect the simulation time and save it
            load('Temp/Analyse_All_temp.mat');
            Analyse_CPUTime
            clear all
            close all
            load('Temp/Analyse_All_temp.mat');
            timer(4,repeat_counter) = toc(timer_Start4);
            disp(['Finished in: ' num2str(timer(4,repeat_counter)) 'sec']);
        end
        if force_Analyse_SimQ == 1
            timer_Start5 = tic;
            save('Temp/Analyse_All_temp.mat', 'dir*', 'repeat_counter', ...
            'stop_counter', 'force_*', 'timer*');
            clear all
            close all
            %Make an overview of the simulation quality
            load('Temp/Analyse_All_temp.mat');
            Analyse_SimQ
            clear all
            close all
            load('Temp/Analyse_All_temp.mat');
            timer(5,repeat_counter) = toc(timer_Start5);
            disp(['Finished in: ' num2str(timer(5,repeat_counter)) 'sec']);
        end
        if force_Make_Overview == 1
            timer_Start6 = tic;
            save('Temp/Analyse_All_temp.mat', 'dir*', 'repeat_counter', ...
            'stop_counter', 'force_*', 'timer*');
            if force_New_Overview == 1 && repeat_counter == 1
                %Delete previous overviews to make a fresh database
```

```
                delete([dir_name1 '/../Overview.mat']);
                delete([dir_name1 '/../Overview.xls']);
            end
            clear all
            close all
            %Make an overview of all the simulations
            load('Temp/Analyse_All_temp.mat');
            Make_Overview
            clear all
            close all
            load('Temp/Analyse_All_temp.mat');
            timer(6,repeat_counter) = toc(timer_Start6);
            disp(['Finished in: ' num2str(timer(6,repeat_counter)) 'sec']);
        end
    end
    timer(1,repeat_counter) = toc(timer_Start1);
    disp(['Analysing dir finished in: ' ...
        num2str(timer(1,repeat_counter)) 'sec']);
    disp([num2str(repeat_counter) '/' num2str(stop_counter) ' done']);
end
```

A.2 Simulation Scripts

A.2.1 Create_Sim_DB

```
clear all;
close all;
disp('Starting Create_Sim_DB.m');
disp('Loading constants')
load('sim_const.mat');
load('do_what.mat');


disp('Creating full DB');
x_temp = zeros(steps_y,num_sims,'uint8');
%uint8 is used to reduce the memory usage of the matrix
for i = 1:steps_y
    temp_num = zeros(1,steps_x^i,'uint8');
    for k = 1:steps_x
        start_num = (k-1)*steps_x^(i-1)+1;
        stop_num = k*steps_x^(i-1);
        temp_num(start_num:stop_num) = k;
    end
    repetitions = num_sims/length(temp_num);
    for j = 1:repetitions
        start_this_one = (j-1)*length(temp_num)+1;
        stop_this_one = j*length(temp_num);
        x_temp(i,start_this_one:stop_this_one) = temp_num;
    end
end
clear temp_num;
disp('Deleting illegal entries');
for i = 1:length(x_temp(1,:))
    counter = 0;
```

```
    for j = 1:length(x_temp(:,1))−1
        stepsize = abs(x_temp(j,i)−x_temp(j+1,i));
        if stepsize > max_change
            counter = counter+1;
        end
    end
    if counter ≠ 0
        x_temp(:,i) = 0;
    end
end

if do_Ye1998 == 1
    %If the geometries of Ye et al. are to be used, these settings need to
    %be used
    clear x_temp
    tapered = [0 1 2 3 4 5 6 7 8 9 10 11 12]/4;
    straight = [0 0 0 0 0 0 0 0 0 0 0 0 0];
    uniform_linear = [10 11 12 13 14 15.5 17 19 21 25.5 30 36 42]/6;
    uniform_quadratic = [10 11 12 13.5 15 17 19 23 27 34 41 49.5 58]/6;
    uniform_cubic = [10 11 12 14 17 21 26 33 44 60 80 94 99 ]/6;
    shaped_linear = [9 9 10 11 11 12 12 13 14 14 15 15 16]/6;
    shaped_quadratic = [9 10 11 12 13 14 16 18 19 20 22 23 25]/6;
    shaped_cubic = [9 10 10 11 12 14 15 17 19 23 27 32 37]/6;

    x_temp(:,1) = uniform_linear;
    x_temp(:,2) = uniform_quadratic;
    x_temp(:,3) = uniform_cubic;
    x_temp(:,4) = shaped_linear;
    x_temp(:,5) = shaped_quadratic;
    x_temp(:,6) = shaped_cubic;
    x_temp(:,7) = straight;
    x_temp(:,8) = tapered;
    x_vars = x_temp;
else
    nonzero_rows = find(x_temp(1,:));
    disp('Saving remaining entries');
    x_vars = x_temp(:,nonzero_rows);
end

save sim_db.mat x_vars
save sim_db_full.mat x_temp
num_sims = length(x_vars(1,:));
disp(['Number of remaining entries: ' num2str(num_sims)]);
disp('Done Create_Sim_DB.m');
```

A.2.2 Do_Sim_lin

```
clear all;
close all;
delete 'Temp/temp*.mat';

resume_previous = 0;    %Resume from previous simulation?
                        %0 for no
                        %1 for yes
sim_counter_direction = 1; %Count up or down
```

```
                            %1 for up
                            %-1 for down

errors = [];
error_count = 0;

if (exist('counter.mat','file') == 2)
    load('counter.mat');
else
    simulation_number = 0;
end
if resume_previous == 0
    simulation_number = simulation_number+1;
else
    %Resume from previous results
    files = dir([dir_name3 '/FEM Model - ID *.mat']);
    id(length(files)) = 0;
    for i = 1:length(files)
        filename = files(i).name;
        id(i) = str2double(strrep(strrep(filename,'.mat',''), ...
            'FEM Model - ID ',''));
    end
    started_id  = min(id);
    stopped_id = max(id);
end

disp('Starting Do_Sim.m');
load('sim_db.mat');
load('sim_const.mat');

dir_name1 = ['Results/Simulation results - ' num2str(simulation_number) ...
    ' - ' date_of_start];
dir_name2 = [dir_name1 '/Figures'];
dir_name3 = [dir_name1 '/Models'];
if (exist(dir_name1, 'dir') == 0)
    mkdir(dir_name1);
end
if (exist(dir_name2, 'dir') == 0)
    mkdir(dir_name2);
end
if (exist(dir_name3, 'dir') == 0)
    mkdir(dir_name3);
end
save 'counter.mat' simulation_number dir_*;

copyfile('*.m', dir_name1);
copyfile('*.mat', dir_name1);

flclear fem
fem = [];

% COMSOL version
clear vrsn
vrsn.name = 'COMSOL 3.5';
vrsn.ext = '';
vrsn.major = 0;
vrsn.build = 494;
```

```
vrsn.rcs = '$Name:  $';
vrsn.date = '$Date: 2008/09/19 16:09:48 $';

%Simulation settings
min_sim_counter = 1;
max_sim_counter = max(min_sim_counter,length(x_vars(1,:)));
Y_disp_start = 0;
Y_disp_step = 5e-7;
Y_disp_stop = finger_length-2.1*init_x;

%Mesh settings
% mesh refinement from 1 to 9
% 1 being the finest mesh,9 the coarsest
% more info type "help meshinit"
mesh_start = 3;
mesh_end = 3;
mesh_direction = 1;
if mesh_start ≥ mesh_end
    mesh_direction = -1;
end


do_simulation = 1;

time_save = zeros(max_sim_counter,1);
time_save2 = zeros(max_sim_counter,1);


sim_counts = max_sim_counter-min_sim_counter;
onepercent = (sim_counts)/100;
time_counter = 1;

disp('Starting simulations...');
disp(['Results will be saved in: ' dir_name1]);
%Fix the beginning and ending of the simulation
if sim_counter_direction == 1 && resume_previous == 1
    min_sim_counter = max(min_sim_counter, stopped_id+1);
    max_sim_counter = max_sim_counter;
elseif sim_counter_direction == 1 && resume_previous == 0
    min_sim_counter = min_sim_counter;
    max_sim_counter = max_sim_counter;
elseif sim_counter_direction == -1 && resume_previous == 1
    temp_counter = min_sim_counter;
    min_sim_counter = min(max_sim_counter, started_id-1);
    max_sim_counter = temp_counter;
    max_sim_counter = temp_counter;
elseif sim_counter_direction == -1 && resume_previous == 0
    temp_counter = min_sim_counter;
    min_sim_counter = max_sim_counter;
    max_sim_counter = temp_counter;
    clear temp_counter
end
%Matrix if certain id's should not be simulated
%Format: a_matrix(:,:) = [id mesh_refinement]
a_matrix(1,:) = [0 0];

tic;
```

```
for mesh_refinement = mesh_start:mesh_direction:mesh_end
for sim_counter = min_sim_counter:sim_counter_direction:max_sim_counter
    dontdosim = 0;
    for a_counter = 1:length(a_matrix(:,1))
        if sim_counter == a_matrix(a_counter,1)
            if mesh_refinement == a_matrix(a_counter,2)
                dontdosim = 1;
            end
        end
    end
    if dontdosim == 0
        clear femsave
        tStart = tic;
        Simulated_id = sim_counter;
        disp(['Simulating ID ' num2str(sim_counter) ...
            '@' num2str(mesh_refinement)]);
        Y_max_displacement(Simulated_id) = 0;
        Y_steps_finished(Simulated_id) = 0;
        for Y_displacement=Y_disp_start:Y_disp_step:Y_disp_stop
            flclear fem
            fem = [];
            save('sim_counter.mat','sim_counter', 'mesh_refinement');

            fem.version = vrsn;
            offset_x_0 = 0;
            offset_y_0 = 0;
            offset_x = 0;
            offset_y = 0;
            femshape = solid2;
            foot_thickness = init_foot_thickness;
            X1 = zeros(4,amount_of_fingers);
            X2 = zeros(4,amount_of_fingers);
            Y1 = zeros(4,amount_of_fingers);
            Y2 = zeros(4,amount_of_fingers);
            num_finger = sim_counter;  %ID of the selected fingershape
            clear tempshape femshape;
            tempshape = solid2;
            femshapea = solid2;
            femshapeb = solid2;
            %Build the geometry using the function 'Build_Finger'
            for i = 1:amount_of_fingers
            if i == 1
            %The first finger
            offset_x = offset_x_0;
            [ femshape1a femshape1b  X1(3,i) Y1(3,i) X2(3,i) Y2(3,i) ] ...
                = Build_Finger( double(x_vars(:,num_finger))*step_x, ...
                offset_x, offset_y_0, 1, 1, foot_thickness );
            offset_y_2 = offset_y_0+2*init_foot_thickness+...
                2*finger_length-init_x-Y_displacement;
            [ femshape2a femshape2b X1(4,i) Y1(4,i) X2(4,i) Y2(4,i) ] ...
                = Build_Finger( double(x_vars(:,num_finger))*step_x, ...
                X2(3,i), offset_y_2, 2, 2, init_foot_thickness );
            femshapea = geomcomp({femshapea, femshape1a, femshape2a},...
                'ns',{'femshapea', 'femshape1a','femshape2a'},'sf',...
                'femshapea+femshape1a+femshape2a','edge','all');
            femshapeb = geomcomp({femshapeb, femshape1b, femshape2b},...
                'ns',{'femshapeb', 'femshape1b','femshape2b'},'sf',...
```

```
            'femshapeb+femshape1b+femshape2b','edge','all');
        else
        %All the other fingers
        offset_x = X2(1,i-1);
        [ femshape1a femshape1b X1(1,i) Y1(1,i) X2(1,i) Y2(1,i) ] ...
            = Build_Finger( double(x_vars(:,num_finger))*step_x, ...
            X2(4,i-1), Y2(4,i-1), 1, 2, init_foot_thickness );
        [ femshape2a femshape2b X1(2,i) Y1(2,i) X2(2,i) Y2(2,i) ] ...
            = Build_Finger( double(x_vars(:,num_finger))*step_x, ...
            X2(1,i), Y1(3,i-1), 2, 1, foot_thickness );
        [ femshape3a femshape3b X1(3,i) Y1(3,i) X2(3,i) Y2(3,i) ] ...
            = Build_Finger( double(x_vars(:,num_finger))*step_x, ...
            X2(2,i), Y1(3,i-1), 1, 1, foot_thickness );
        [ femshape4a femshape4b X1(4,i) Y1(4,i) X2(4,i) Y2(4,i) ] ...
            = Build_Finger( double(x_vars(:,num_finger))*step_x, ...
            X2(3,i), Y2(4,i-1), 2, 2, init_foot_thickness );
        femshapea = geomcomp({femshapea, femshape1a, femshape2a, ...
            femshape3a, femshape4a},'ns',{'femshape1a','femshape2a',...
            'femshape3a','femshape4a','femshapea'},'sf',...
            'femshape1a+femshape2a+femshape3a+femshape4a+femshapea',...
            'edge','all');
        femshapeb = geomcomp({femshapeb, femshape1b, femshape2b, ...
            femshape3b, femshape4b},'ns',{'femshape1b','femshape2b',...
            'femshape3b','femshape4b','femshapeb'},'sf',...
            'femshape1b+femshape2b+femshape3b+femshape4b+femshapeb',...
            'edge','all');
        end     %if i == 1
        end     %for sim_step_counter = 1:max_sim_step_counter
        boundingbox_Lx = abs(max(max(X2))-min(min(X1)));
        boundingbox_Ly = abs(max(max(Y2))-min(min(Y1)));
        boundingbox = rect2(boundingbox_Lx*1.5, ...
            boundingbox_Ly*1.5, 'base', 'corner', 'pos', ...
            [min(min(X1)), min(min(Y1))-boundingbox_Ly*0.25]);
        tempshape = geomcomp({femshapea, femshapeb, boundingbox}, ...
            'ns',{'femshapea', 'femshapeb', 'boundingbox'},'sf', ...
            'femshapea+femshapeb+boundingbox','edge','none');

        clear femsha* boundingbox;
        fem = geomanalyze(fem,{tempshape},'ns',{'tempshape'});
        fem.geom=geomcsg(fem);
        if do_simulation == 1
            try
                % Initialize mesh
                fem.mesh=meshinit(fem, ...
                          'hauto',mesh_refinement, ...
                          'report','off');

                % Application mode 1
                clear appl
                appl.mode.class = 'SmePlaneStress';
                appl.module = 'MEMS';
                appl.gporder = 2;
                appl.cporder = 1;
                appl.border = 'on';
                appl.assignsuffix = '_smps';
                clear prop
                prop.elemdefault='Lag1';
```

```
appl.prop = prop;
clear bnd
bnd.loadtype = {'length','length','length','area'};
bnd.name = {'','Fixed','Symmetry','Es_Force'};
bnd.Fx = {0,0,0,'Fes_nTx_emes'};
bnd.Fy = {0,0,0,'Fes_nTy_emes'};
bnd.constrcond = {'free','fixed','sym','free'};
bnd.ind=getapplmodes(amount_of_fingers, 1, 2, steps_y);
appl.bnd = bnd;
clear equ
equ.constrcond = {'free','fixed','free','fixed'};
equ.Hx = {0,1,0,0};
equ.usage = {1,1,1,1};
equ.ind=getapplmodes(amount_of_fingers, 1, 3, steps_y);
appl.equ = equ;
fem.appl{1} = appl;

% Application mode 2
clear appl
appl.mode.class = 'EmElectrostatics';
appl.module = 'MEMS';
appl.border = 'on';
appl.assignsuffix = '_emes';
clear prop
prop.elemdefault='Lag1';
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm3'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear bnd
bnd.V0 = {0,0,10,0};
bnd.type = {'nD0','cont','V','V0'};
bnd.name = {'Zero charge/Symmetry','','V_in','Ground'};
bnd.ind=getapplmodes(amount_of_fingers, 2, 2, steps_y);
appl.bnd = bnd;
clear equ
equ.epsilonr = {1,4.5};
equ.maxwell = {{},'Fes'};
equ.ind=getapplmodes(amount_of_fingers, 2, 3, steps_y);
appl.equ = equ;
fem.appl{2} = appl;
fem.border = 1;
clear units;
units.basesystem = 'SI';
fem.units = units;

% Multiphysics
fem=multiphysics(fem);

% Extend mesh
fem.xmesh=meshextend(fem,...
    'report', 'off');

%Save a backup in case something goes wrong
%simulating
fem1 = fem;
```

```

                    % Solve problem
                    fem.sol=femstatic(fem, ...
                                      'solcomp',{'v','u','V'}, ...
                                      'outcomp',{'v','u','V'}, ...
                                      'blocksize','auto', ...
                                      'maxiter',1000,...
                                      'report', 'off');
                    Y_max_displacement(Simulated_id) = Y_displacement;
                    Y_steps_finished(Simulated_id) = ...
                        Y_displacement/Y_disp_step+1;
                    filename1 = ['Temp/temp' ...
                        num2str(Y_displacement/Y_disp_step) '.mat'];
                    clear A
                    A = genvarname(['femsave' ...
                        num2str(Y_displacement/Y_disp_step)]);
                    eval([A '=fem;']);
                    save(filename1,eval('A'), 'mesh_refinement', ...
                    'Y_displacement', 'min_gap', 'amount_of_fingers', ...
                    'finger_length', 'Simulated_id');
                    clear femsave*
                catch me
                error_count = error_count + 1;
                errors(error_count).identifier = me.identifier;
                errors(error_count).message = me.message;
                errors(error_count).stack = me.stack;
                errors(error_count).cause = me.cause;
                errors(error_count).id = sim_counter;
                disp(['Error in ' me.stack(length(me.stack),1).name ...
                '.m on line ' num2str(me.stack(length(me.stack),1).line)...
                ' for ID = ' num2str(sim_counter)]);
                if length(me.message) >= 120
                    disp(['Error: ' me.message(73:120)]);
                else
                    disp(['Error: ' me.message]);
                end
                end     %try
            end     %if do_simulation == 1
        end     %for Y_displacement
        CPUTime = toc(tStart);
        now_done = sim_counter-min_sim_counter;
        tElapsed = toc(tStart);
        time_save(Simulated_id) = tElapsed;
        time_save2(Simulated_id) = toc;
        if floor(now_done/onepercent) == (now_done/onepercent)
            disp([num2str(now_done/onepercent) '% done']);
            disp(['Average time per ID: ' ...
                num2str(mean(time_save(find(time_save)))) ' seconds']);
        end     %if floor(now_done/onepercent) == (now_done/onepercent)
        numstr = num2str(Simulated_id);
        %Add a number of 0's to make uniform filenames
        for temp_counter = 6:-1:1
            if (Simulated_id < 10^temp_counter)
                numstr = ['0' numstr];
            end   %if
        end   %for temp_counter
        filename1 = [dir_name3 '/FEM Model - ID ' numstr '.mat'];
```

```
        files = dir('Temp/temp*.mat');
        id = 1;
        for savecounter = 1:length(files)
            loadfilename = files(savecounter).name;
            id1 = strrep(strrep(loadfilename,'.mat',''),'temp','');
            id = str2double(id1)+1;
            load(['Temp/' loadfilename]);
            Y_disp(id) = Y_displacement;
        end
        try
            save(filename1, 'femsave*', 'CPUTime', 'mesh_refinement', ...
                'min_gap', 'Y_disp', 'id', 'sim_counter');
            delete 'Temp/temp*.mat';
        catch me
            error_count = error_count + 1;
            errors(error_count).identifier = me.identifier;
            errors(error_count).message = me.message;
            errors(error_count).stack = me.stack;
            errors(error_count).cause = me.cause;
            errors(error_count).id = sim_counter;
            disp(['Error in ' me.stack(length(me.stack),1).name ...
                '.m on line ' num2str(me.stack(length(me.stack),1).line)...
                ' for ID = ' num2str(sim_counter)]);
            disp(['Error: ' me.message]);
            femsave = fem1;
            %In case of error: save backup to prevent errors with analysing
            save(filename1, 'femsave', 'CPUTime', 'mesh_refinement', ...
                'min_gap', 'id');
        end
    end     %if dontdosim
end     %for sim_counter = etc
end     %for mesh_refinement = max_refinement:-1:min_refinement
total_num_of_sims = ...
(abs(max_sim_counter-min_sim_counter)+1)*(abs(mesh_end-mesh_start)+1);
filename1 = [dir_name1 '/errors.mat'];
save(filename1, 'errors');
filename1 = [dir_name1 '/time_save.mat'];
save(filename1, 'time_save', 'time_save2');
```

A.2.3 Do_Sim_quad

```
clear all;
close all;
delete 'Temp/temp*.mat';

resume_previous = 0;    %Resume from previous simulation?
                        %0 for no
                        %1 for yes
sim_counter_direction = 1; %Count up or down
                        %1 for up
                        %-1 for down

errors = [];
error_count = 0;

```

```
if (exist('counter.mat','file') == 2)
    load('counter.mat');
else
    simulation_number = 0;
end
if resume_previous == 0
    simulation_number = simulation_number+1;
else
    %Resume from previous results
    files = dir([dir_name3 '/FEM Model - ID *.mat']);
    id(length(files)) = 0;
    for i = 1:length(files)
        filename = files(i).name;
        id(i) = str2double(strrep(strrep(filename,'.mat',''), ...
            'FEM Model - ID ',''));
    end
    started_id  = min(id);
    stopped_id = max(id);
end

disp('Starting Do_Sim.m');
load('sim_db.mat');
load('sim_const.mat');

dir_name1 = ['Results/Simulation results - ' num2str(simulation_number) ...
    ' - ' date_of_start];
dir_name2 = [dir_name1 '/Figures'];
dir_name3 = [dir_name1 '/Models'];
if (exist(dir_name1, 'dir') == 0)
    mkdir(dir_name1);
end
if (exist(dir_name2, 'dir') == 0)
    mkdir(dir_name2);
end
if (exist(dir_name3, 'dir') == 0)
    mkdir(dir_name3);
end
save 'counter.mat' simulation_number dir_*;

copyfile('*.m', dir_name1);
copyfile('*.mat', dir_name1);

flclear fem
fem = [];

% COMSOL version
clear vrsn
vrsn.name = 'COMSOL 3.5';
vrsn.ext = '';
vrsn.major = 0;
vrsn.build = 494;
vrsn.rcs = '$Name:  $';
vrsn.date = '$Date: 2008/09/19 16:09:48 $';

%Simulation settings
min_sim_counter = 1;
max_sim_counter = max(min_sim_counter,length(x_vars(1,:)));
```

```
Y_disp_start = 0;
Y_disp_step = 5e−7;
Y_disp_stop = finger_length−2.1*init_x;

%Mesh settings
% mesh refinement from 1 to 9
% 1 being the finest mesh,9 the coarsest
% more info type "help meshinit"
mesh_start = 3;
mesh_end = 3;
mesh_direction = 1;
if mesh_start ≥ mesh_end
    mesh_direction = −1;
end


do_simulation = 1;

time_save = zeros(max_sim_counter,1);
time_save2 = zeros(max_sim_counter,1);


sim_counts = max_sim_counter−min_sim_counter;
onepercent = (sim_counts)/100;
time_counter = 1;

disp('Starting simulations...');
disp(['Results will be saved in: ' dir_name1]);
%Fix the beginning and ending of the simulation
if sim_counter_direction == 1 && resume_previous == 1
    min_sim_counter = max(min_sim_counter, stopped_id+1);
    max_sim_counter = max_sim_counter;
elseif sim_counter_direction == 1 && resume_previous == 0
    min_sim_counter = min_sim_counter;
    max_sim_counter = max_sim_counter;
elseif sim_counter_direction == −1 && resume_previous == 1
    temp_counter = min_sim_counter;
    min_sim_counter = min(max_sim_counter, started_id−1);
    max_sim_counter = temp_counter;
    max_sim_counter = temp_counter;
elseif sim_counter_direction == −1 && resume_previous == 0
    temp_counter = min_sim_counter;
    min_sim_counter = max_sim_counter;
    max_sim_counter = temp_counter;
    clear temp_counter
end
%Matrix if certain id's should not be simulated
%Format: a_matrix(:,:) = [id mesh_refinement]
a_matrix(1,:) = [0 0];

tic;
for mesh_refinement = mesh_start:mesh_direction:mesh_end
for sim_counter = 1:1:3%min_sim_counter:sim_counter_direction:max_sim_counter
    dontdosim = 0;
    for a_counter = 1:length(a_matrix(:,1))
        if sim_counter == a_matrix(a_counter,1)
            if mesh_refinement == a_matrix(a_counter,2)
```

```
                dontdosim = 1;
            end
        end
    end
    if dontdosim == 0
        clear femsave
        tStart = tic;
        Simulated_id = sim_counter;
        disp(['Simulating ID ' num2str(sim_counter) ...
            '@' num2str(mesh_refinement)]);
        Y_max_displacement(Simulated_id) = 0;
        Y_steps_finished(Simulated_id) = 0;
        for Y_displacement=Y_disp_start:Y_disp_step:Y_disp_stop
            flclear fem
            fem = [];
            save('sim_counter.mat','sim_counter', 'mesh_refinement');

            fem.version = vrsn;
            offset_x_0 = 0;
            offset_y_0 = 0;
            offset_x = 0;
            offset_y = 0;
            femshape = solid2;
            foot_thickness = init_foot_thickness;
            X1 = zeros(4,amount_of_fingers);
            X2 = zeros(4,amount_of_fingers);
            Y1 = zeros(4,amount_of_fingers);
            Y2 = zeros(4,amount_of_fingers);
            num_finger = sim_counter;  %ID of the selected fingershape
            clear tempshape femshape;
            tempshape = solid2;
            femshapea = solid2;
            femshapeb = solid2;
            %Build the geometry using the function 'Build_Finger'
            for i = 1:amount_of_fingers
            if i == 1
            %The first finger
            offset_x = offset_x_0;
            [ femshape1a femshape1b  X1(3,i) Y1(3,i) X2(3,i) Y2(3,i) ] ...
                = Build_Finger( double(x_vars(:,num_finger))*step_x, ...
                offset_x, offset_y_0, 1, 1, foot_thickness );
            offset_y_2 = offset_y_0+2*init_foot_thickness+...
                2*finger_length-init_x-Y_displacement;
            [ femshape2a femshape2b X1(4,i) Y1(4,i) X2(4,i) Y2(4,i) ] ...
                = Build_Finger( double(x_vars(:,num_finger))*step_x, ...
                X2(3,i), offset_y_2, 2, 2, init_foot_thickness );
            femshapea = geomcomp({femshapea, femshape1a, femshape2a},...
                'ns',{'femshapea', 'femshape1a','femshape2a'},'sf',...
                'femshapea+femshape1a+femshape2a','edge','all');
            femshapeb = geomcomp({femshapeb, femshape1b, femshape2b},...
                'ns',{'femshapeb', 'femshape1b','femshape2b'},'sf',...
                'femshapeb+femshape1b+femshape2b','edge','all');
            else
            %All the other fingers
            offset_x = X2(1,i-1);
            [ femshape1a femshape1b X1(1,i) Y1(1,i) X2(1,i) Y2(1,i) ] ...
                = Build_Finger( double(x_vars(:,num_finger))*step_x, ...
```

```
            X2(4,i-1), Y2(4,i-1), 1, 2, init_foot_thickness );
        [ femshape2a femshape2b X1(2,i) Y1(2,i) X2(2,i) Y2(2,i) ] ...
            = Build_Finger( double(x_vars(:,num_finger))*step_x, ...
            X2(1,i), Y1(3,i-1), 2, 1, foot_thickness );
        [ femshape3a femshape3b X1(3,i) Y1(3,i) X2(3,i) Y2(3,i) ] ...
            = Build_Finger( double(x_vars(:,num_finger))*step_x, ...
            X2(2,i), Y1(3,i-1), 1, 1, foot_thickness );
        [ femshape4a femshape4b X1(4,i) Y1(4,i) X2(4,i) Y2(4,i) ] ...
            = Build_Finger( double(x_vars(:,num_finger))*step_x, ...
            X2(3,i), Y2(4,i-1), 2, 2, init_foot_thickness );
        femshapea = geomcomp({femshapea, femshape1a, femshape2a, ...
            femshape3a, femshape4a},'ns',{'femshape1a','femshape2a',...
            'femshape3a','femshape4a','femshapea'},'sf',...
            'femshape1a+femshape2a+femshape3a+femshape4a+femshapea',...
            'edge','all');
        femshapeb = geomcomp({femshapeb, femshape1b, femshape2b, ...
            femshape3b, femshape4b},'ns',{'femshape1b','femshape2b',...
            'femshape3b','femshape4b','femshapeb'},'sf',...
            'femshape1b+femshape2b+femshape3b+femshape4b+femshapeb',...
            'edge','all');
        end     %if i == 1
        end     %for sim_step_counter = 1:max_sim_step_counter
        boundingbox_Lx = abs(max(max(X2))-min(min(X1)));
        boundingbox_Ly = abs(max(max(Y2))-min(min(Y1)));
        boundingbox = rect2(boundingbox_Lx*1.5, ...
            boundingbox_Ly*1.5, 'base', 'corner', 'pos', ...
            [min(min(X1)), min(min(Y1))-boundingbox_Ly*0.25]);
        tempshape = geomcomp({femshapea, femshapeb, boundingbox}, ...
            'ns',{'femshapea', 'femshapeb', 'boundingbox'},'sf', ...
            'femshapea+femshapeb+boundingbox','edge','none');

        clear femsha* boundingbox;
        fem = geomanalyze(fem,{tempshape},'ns',{'tempshape'});
        fem.geom=geomcsg(fem);
        if do_simulation == 1
            try
                % Initialize mesh
                fem.mesh=meshinit(fem, ...
                          'hauto',mesh_refinement, ...
                          'report','off');

                % Application mode 1
                clear appl
                appl.mode.class = 'SmePlaneStress';
                appl.sdin = {'X','Y','Z'};
                appl.module = 'MEMS';
                appl.gporder = 4;
                appl.cporder = 2;
                appl.border = 'on';
                appl.assignsuffix = '_smps';
                clear prop
                prop.analysis = 'para';
                prop.deformfram = 'ref';
                prop.fram='ref';
                appl.prop = prop;
                clear bnd
                bnd.loadtype = {'length','length','length','area'};
```

```
bnd.name = {'','Fixed','Symmetry','Es_Force'};
bnd.Fx = {0,0,0,'Fes_nTx_emes'};
bnd.Fy = {0,0,0,'Fes_nTy_emes'};
bnd.constrcond = {'free','fixed','sym','free'};
bnd.ind=getapplmodes(amount_of_fingers,1,2,steps_y);
appl.bnd = bnd;
clear equ
equ.constrcond = {'free','fixed','free','fixed'};
equ.Hx = {0,1,0,0};
equ.usage = {1,1,1,1};
equ.ind=getapplmodes(amount_of_fingers,1,3,steps_y);
appl.equ = equ;
fem.appl{1} = appl;

% Application mode 2
clear appl
appl.mode.class = 'EmElectrostatics';
appl.module = 'MEMS';
appl.border = 'on';
appl.assignsuffix = '_emes';
clear prop
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm5'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear bnd
bnd.V0 = {0,0,10,0};
bnd.type = {'nD0','cont','V','V0'};
bnd.name = {'Zero charge/Symmetry','','V_in','Ground'};
bnd.ind = getapplmodes(amount_of_fingers,2,2,steps_y);
appl.bnd = bnd;
clear equ
equ.epsilonr = {1,4.5};
equ.maxwell = {{},'Fes'};
equ.ind = getapplmodes(amount_of_fingers,2,3,steps_y);
appl.equ = equ;
fem.appl{2} = appl;
fem.sdin = {{'Xm','Ym'},{'X','Y'},{'x','y'}};
fem.fram = {'mesh','ref','ale'};
fem.border = 1;
clear units;
units.basesystem = 'SI';
fem.units = units;

% Multiphysics
fem=multiphysics(fem);

% Extend mesh
fem.xmesh=meshextend(fem,...
    'report', 'off');

%Save a backup in case something goes wrong
%simulating
fem1 = fem;

% Solve problem
```

```
                    fem.sol=femstatic(fem, ...
                                      'solcomp',{'v','u','V'}, ...
                                      'outcomp',{'v','u','V'}, ...
                                      'blocksize','auto', ...
                                      'maxiter',1000,...
                                      'report', 'off');
                    Y_max_displacement(Simulated_id) = Y_displacement;
                    Y_steps_finished(Simulated_id) = ...
                        Y_displacement/Y_disp_step+1;
                    filename1 = ['Temp/temp' ...
                        num2str(Y_displacement/Y_disp_step) '.mat'];
                    clear A
                    A = genvarname(['femsave' ...
                        num2str(Y_displacement/Y_disp_step)]);
                    eval([A '=fem;']);
                    save(filename1,eval('A'), 'mesh_refinement', ...
                    'Y_displacement', 'min_gap', 'amount_of_fingers', ...
                    'finger_length', 'Simulated_id');
                    clear femsave*
                catch me
                error_count = error_count + 1;
                errors(error_count).identifier = me.identifier;
                errors(error_count).message = me.message;
                errors(error_count).stack = me.stack;
                errors(error_count).cause = me.cause;
                errors(error_count).id = sim_counter;
                disp(['Error in ' me.stack(length(me.stack),1).name ...
                '.m on line ' num2str(me.stack(length(me.stack),1).line)...
                ' for ID = ' num2str(sim_counter)]);
                if length(me.message) ≥ 120
                    disp(['Error: ' me.message(73:120)]);
                else
                    disp(['Error: ' me.message]);
                end
                end    %try
            end    %if do_simulation == 1
        end    %for Y_displacement
        CPUTime = toc(tStart);
        now_done = sim_counter-min_sim_counter;
        tElapsed = toc(tStart);
        time_save(Simulated_id) = tElapsed;
        time_save2(Simulated_id) = toc;
        if floor(now_done/onepercent) == (now_done/onepercent)
            disp([num2str(now_done/onepercent) '% done']);
            disp(['Average time per ID: ' ...
                num2str(mean(time_save(find(time_save)))) ' seconds']);
        end    %if floor(now_done/onepercent) == (now_done/onepercent)
        numstr = num2str(Simulated_id);
        %Add a number of 0's to make uniform filenames
        for temp_counter = 6:-1:1
            if (Simulated_id < 10^temp_counter)
                numstr = ['0' numstr];
            end  %if
        end  %for temp_counter
        filename1 = [dir_name3 '/FEM Model - ID ' numstr '.mat'];
        files = dir('Temp/temp*.mat');
        id = 1;
```

```
        for savecounter = 1:length(files)
            loadfilename = files(savecounter).name;
            id1 = strrep(strrep(loadfilename,'.mat',''),'temp','');
            id = str2double(id1)+1;
            load(['Temp/' loadfilename]);
            Y_disp(id) = Y_displacement;
        end
        try
            save(filename1, 'femsave*', 'CPUTime', 'mesh_refinement', ...
                'min_gap', 'Y_disp', 'id', 'sim_counter');
            delete 'Temp/temp*.mat';
        catch me
            error_count = error_count + 1;
            errors(error_count).identifier = me.identifier;
            errors(error_count).message = me.message;
            errors(error_count).stack = me.stack;
            errors(error_count).cause = me.cause;
            errors(error_count).id = sim_counter;
            disp(['Error in ' me.stack(length(me.stack),1).name ...
                '.m on line ' num2str(me.stack(length(me.stack),1).line)...
                ' for ID = ' num2str(sim_counter)]);
            disp(['Error: ' me.message]);
            femsave = fem1;
            %In case of error: save backup to prevent errors with analysing
            save(filename1, 'femsave', 'CPUTime', 'mesh_refinement', ...
                'min_gap', 'id');
        end
    end     %if dontdosim
end     %for sim_counter = etc
end     %for mesh_refinement = max_refinement:-1:min_refinement
total_num_of_sims = ...
(abs(max_sim_counter-min_sim_counter)+1)*(abs(mesh_end-mesh_start)+1);
filename1 = [dir_name1 '/errors.mat'];
save(filename1, 'errors');
filename1 = [dir_name1 '/time_save.mat'];
save(filename1, 'time_save', 'time_save2');
```

A.2.4 Build_Finger

```
function [ finger_geom finger_container min_x min_y max_x max_y ] = ...
    Build_Finger( x_var, offset_x, offset_y, face_x, face_y, foot_thickness)
%BUILD_FINGER Summary of this function goes here
% Input:
%     x_var = x-values of the sections of the fingers
%     offset_x = X-position of the middle of the finger
%     offset_y = Y-position of the back side of the foot of the finger
%     face_x = x-direction of the finger. 1 means the foot is to the right
%       of the finger (L-shape), 2 means the foot is to the left of the
%       finger (mirrored L-shape)
%     face_y = y-direction of the finger. 1 means the finger points
%       upwards, 2 means the finger points downwards
%     foot_thickness = Thickness of the part at the foot of the finger
% Output:
%     finger_geom = geometry shape of the finger+foot
%     min_x = left-most x-coordinate of the shape
```

```
%      min_y = bottom-most y-coordinate of the shape
%      max_x = right-most x-coordinate of the shape
%      max_y = top-most y-coordinate of the shape

load('sim_const.mat');
load('sim_counter.mat');

min_x = offset_x;
min_y = offset_y;
max_x = offset_x;
max_y = offset_y;

g1 = solid2;
g2 = solid2;
y_steps = length(x_var);

width_finger = finger_width/2 + max(x_var);
height_finger = foot_thickness + y_steps*step_y;
if face_y == 1 && uniform_fingers == 1 && sim_counter <= 3
    x_var = x_var * 0;
end
width_finger2 = finger_width/2 + max(x_var);
if face_y == 2 && uniform_fingers == 1 && sim_counter <= 3
    width_finger2 = finger_width/2;
end
width12 = width_finger + width_finger2;


%Make the lower finger
if face_y == 1
    x_1 = offset_x;
    x_2 = offset_x + finger_width/2;
    x_3 = offset_x + finger_width/2;
    x_4 = offset_x;
    y_1 = offset_y + y_steps*step_y + foot_thickness;
    y_2 = y_1;
    for Build_Finger_Counter = 1:y_steps-1
        x_2 = x_3;
        x_3 = offset_x + finger_width/2 + x_var(Build_Finger_Counter);
        y_1 = y_2;
        y_2 = y_1-step_y;
        temp_curve = {curve2([x_1, x_2],[y_1, y_1],[10e-6,10e-6]),...
            curve2([x_2,x_3],[y_1,y_2],[10e-6,10e-6]),...
            curve2([x_3,x_4],[y_2,y_2],[10e-6,10e-6]),...
            curve2([x_4,x_1],[y_2,y_1],[10e-6,10e-6])};
        g4 = geomcoerce('solid',temp_curve);
        g1 = geomcomp({g1, g4},'ns',{'g1','g4'},'sf','g1+g4','edge','none');
    end
    %build last segment
    x_2 = x_3;
    x_3 = x_3;
    y_1 = y_2;
    y_2 = y_1-step_y;
    temp_curve = {curve2([x_1, x_2],[y_1, y_1],[10e-6,10e-6]),...
        curve2([x_2,x_3],[y_1,y_2],[10e-6,10e-6]),...
        curve2([x_3,x_4],[y_2,y_2],[10e-6,10e-6]),...
        curve2([x_4,x_1],[y_2,y_1],[10e-6,10e-6])};
```

```
    g4 = geomcoerce('solid',temp_curve);
    g1 = geomcomp({g1, g4},'ns',{'g1','g4'},'sf','g1+g4','edge','none');

    %build foot
    temp_rect=rect2(width12+min_gap, foot_thickness, 'base', ...
        'corner', 'pos', [offset_x, offset_y]);
    finger_geom = geomcomp({g1, temp_rect},'ns',{'g1','temp_rect'},'sf',...
        'g1+temp_rect','edge','none');

    %build finger container
    temp_rect=rect2(width_finger2+min(min_gap*0.3, 2e-6), ...
        height_finger + min(min_gap*0.3, 2e-6), 'base', 'corner', ...
        'pos', [offset_x, offset_y]);
    finger_container = geomcomp({finger_geom, temp_rect},'ns',...
        {'finger_geom','temp_rect'},'sf','temp_rect-finger_geom',...
        'edge','all');
end

%Make upper finger
if face_y == 2
    x_1 = offset_x;
    x_2 = offset_x + width_finger;
    x_3 = offset_x + width_finger;
    x_4 = offset_x;
    y_1 = offset_y - foot_thickness;
    y_2 = y_1;
    %build first segment
    x_2 = x_3;
    x_3 = x_3;
    y_1 = y_2;
    y_2 = y_1-step_y;
    temp_curve = {curve2([x_1, x_2],[y_1, y_1],[10e-6,10e-6]),...
        curve2([x_2,x_3],[y_1,y_2],[10e-6,10e-6]),...
        curve2([x_3,x_4],[y_2,y_2],[10e-6,10e-6]),...
        curve2([x_4,x_1],[y_2,y_1],[10e-6,10e-6])};
    g4 = geomcoerce('solid',temp_curve);
    g1 = geomcomp({g1, g4},'ns',{'g1','g4'},'sf','g1+g4','edge','none');


    for Build_Finger_Counter = 1:y_steps-1
        x_2 = x_3;
        x_3 = offset_x + width_finger - x_var(Build_Finger_Counter);
        y_1 = y_2;
        y_2 = y_1 - step_y;
        temp_curve = {curve2([x_1, x_2],[y_1, y_1],[10e-6,10e-6]),...
            curve2([x_2,x_3],[y_1,y_2],[10e-6,10e-6]),...
            curve2([x_3,x_4],[y_2,y_2],[10e-6,10e-6]),...
            curve2([x_4,x_1],[y_2,y_1],[10e-6,10e-6])};
        g4 = geomcoerce('solid',temp_curve);
        g1 = geomcomp({g1, g4},'ns',{'g1','g4'},'sf','g1+g4','edge','none');
    end

    %build foot
    temp_rect=rect2(width12+min_gap, foot_thickness, 'base', 'corner', ...
        'pos', [offset_x, offset_y-foot_thickness]);
    finger_geom = geomcomp({g1, temp_rect},'ns',{'g1','temp_rect'},...
        'sf','g1+temp_rect','edge','none');
```

```

    %build finger container
    temp_rect=rect2(width_finger+min(min_gap*0.3, 2e-6), ...
        height_finger - min_gap, 'base', 'corner', 'pos', [offset_x, ...
        offset_y-(height_finger+min(min_gap*0.3 , 2e-6))]);
    finger_container = geomcomp({finger_geom, temp_rect},'ns',...
        {'finger_geom','temp_rect'},'sf','temp_rect-finger_geom',...
        'edge','all');
end
min_x = offset_x;
max_x = offset_x + width12 + min_gap;
min_y = offset_y;
max_y = offset_y + height_finger + min_gap;
if face_x == 2
    finger_geom = mirror(finger_geom,[offset_x offset_y], [1 0]);
    finger_container = mirror(finger_container,[offset_x offset_y], [1 0]);
    min_x = offset_x - width12 - min_gap;
    max_x = offset_x;
end
if face_y == 2
    min_y = offset_y - height_finger - min_gap;
    max_y = offset_y;
end
end
```

A.2.5 getapplmodes

```
function [ outputvector ] = getapplmodes( number_of_fingers,appl_mode,element_type,subelement
%GETAPPLMODES Summary of this function goes here
%   Output:
%   outputvector gives the vector containing the numbers for the
%   boundaries/subdomains
%
%   Input:
%   number_of_fingers is the amount of fingers used in the simulation, it
%      is used to select the appropriate number of elements
%   appl_mode is the application mode for which the vector needs to be
%      given:
%      1 = smps = MEMS plane stress
%      2 = emes = MEMS electrostatics
%      3 = inte = boundaries for Maxwell tensor integration
%   element_type is the type of element for which the vector needs to be
%       given:
%       1 = point
%       2 = boundary
%       3 = subdomain
%   subelements is the amount of elements used to build up the fingers
clear smps emes outputvector
%The below vectors are generated by Comsol when the boundaries/subdomains
%had been given the right properties manually
smps.boundary(3).numsubelements(13,:) = [1,1,3,2,3,3,3,3,3,3,3,3,3,...
    3,3,3,3,1,4,1,1,3,4,1,4,1,4,4,4,4,4,4,4,4,4,4,4,4,4,4,1,4,1,1,4,...
    4,4,4,4,4,4,4,4,4,4,4,4,4,4,2,4,1,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,...
    4,1,4,1,4,1,4,4,4,4,4,4,4,4,4,4,4,4,4,4,2,4,1,4,4,4,4,4,4,4,4,4,...
    4,4,4,4,4,4,4,1,4,1,1,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,2,4,1,4,4,4,...
```

```
    4,4,4,4,4,4,4,4,4,4,4,4,4,1,4,1,4,1,4,4,4,4,4,4,4,4,4,4,4,4,4,4,...
    2,4,1,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,1,4,1,1,4,4,4,4,4,4,4,4,4,...
    4,4,4,4,4,4,4,1,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,1];
smps.subdomains(3).numsubelements(13,:) = [1,2,2,2,2,2,2,2,2];
smps.boundary(5).numsubelements(13,:) = [1,1,3,2,3,3,3,3,3,3,3,3,3,...
    3,3,3,3,3,4,1,1,3,4,1,2,1,4,4,4,4,4,4,4,4,4,4,4,4,4,4,1,4,1,1,4,...
    4,4,4,4,4,4,4,4,4,4,4,4,4,4,2,4,1,4,2,4,4,4,4,4,4,4,4,4,4,4,4,4,...
    4,1,4,1,4,1,4,4,4,4,4,4,4,4,4,4,4,4,4,4,2,4,1,4,2,4,4,4,4,4,4,4,...
    4,4,4,4,4,4,4,1,4,1,1,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,2,4,1,4,2,4,...
    4,4,4,4,4,4,4,4,4,4,4,4,4,1,4,1,4,1,4,4,4,4,4,4,4,4,4,4,4,4,4,4,...
    2,4,1,4,2,4,4,4,4,4,4,4,4,4,4,4,4,4,4,1,4,1,1,4,4,4,4,4,4,4,4,4,...
    4,4,4,4,4,4,2,4,1,4,2,4,4,4,4,4,4,4,4,4,4,4,4,4,4,1,4,1,4,1,4,4,...
    4,4,4,4,4,4,4,4,4,4,4,4,2,4,1,4,2,4,4,4,4,4,4,4,4,4,4,4,4,4,4,1,...
    4,1,1,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,2,4,1,4,2,4,4,4,4,4,4,4,4,4,...
    4,4,4,4,4,1,4,1,4,1,4,4,4,4,4,4,4,4,4,4,4,4,4,4,2,4,1,4,2,4,4,4,...
    4,4,4,4,4,4,4,4,4,4,4,1,4,1,1,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,1,...
    4,4,4,4,4,4,4,4,4,4,4,4,4,4,1];
smps.subdomains(5).numsubelements(13,:) = [1,2,3,4,3,3,3,3,3,3,3,3,3];
emes.boundary(3).numsubelements(13,:) = [1,1,1,3,1,1,1,1,1,1,1,1,1,...
    1,1,1,1,1,3,1,2,1,4,1,4,1,3,3,3,3,3,3,3,3,3,3,3,3,3,2,3,2,2,4,...
    4,4,4,4,4,4,4,4,4,4,4,4,4,4,3,3,2,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,...
    4,2,4,2,3,2,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,2,4,4,3,3,3,3,3,3,3,...
    3,3,3,3,3,3,3,2,3,2,2,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,3,3,2,4,4,4,...
    4,4,4,4,4,4,4,4,4,4,4,4,4,2,4,2,3,2,3,3,3,3,3,3,3,3,3,3,3,3,3,3,...
    3,3,2,4,4,3,3,3,3,3,3,3,3,3,3,3,3,3,3,2,3,2,2,4,4,4,4,4,4,4,4,4,...
    4,4,4,4,4,4,3,2,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,1];
emes.subdomains(3).numsubelements(13,:) = [1,2,1,2,1,1,1,1,1];
emes.boundary(5).numsubelements(13,:) = [1,1,1,3,1,1,1,1,1,1,1,1,1,...
    1,1,1,1,1,3,1,2,1,4,1,4,1,3,3,3,3,3,3,3,3,3,3,3,3,3,2,3,2,2,4,...
    4,4,4,4,4,4,4,4,4,4,4,4,4,4,3,3,2,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,...
    4,2,4,2,3,2,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,2,4,4,3,3,3,3,3,3,3,...
    3,3,3,3,3,3,3,2,3,2,2,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,3,3,2,4,4,4,...
    4,4,4,4,4,4,4,4,4,4,4,4,4,2,4,2,3,2,3,3,3,3,3,3,3,3,3,3,3,3,3,3,...
    3,3,2,4,4,3,3,3,3,3,3,3,3,3,3,3,3,3,3,2,3,2,2,4,4,4,4,4,4,4,4,4,...
    4,4,4,4,4,4,3,3,2,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,2,4,2,3,2,3,3,...
    3,3,3,3,3,3,3,3,3,3,3,3,3,3,2,4,4,3,3,3,3,3,3,3,3,3,3,3,3,3,3,2,...
    3,2,2,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,3,3,2,4,4,4,4,4,4,4,4,4,4,...
    4,4,4,4,4,2,4,2,3,2,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,2,4,4,3,3,3,...
    3,3,3,3,3,3,3,3,3,3,3,2,3,2,2,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,3,2,...
    4,4,4,4,4,4,4,4,4,4,4,4,4,4,1];
emes.subdomains(5).numsubelements(13,:) = [1,2,1,2,1,1,1,1,1,1,1,1,1];
inte.boundary(3).numsubelements(13,:) = [19 27:40 42 61 82 84:97 99 ...
    103:116 118 137 158 160:173 175 179:192 194 212];
inte.boundary(5).numsubelements(13,:) = [19,27,28,29,30,31,32,33,34,...
    35,36,37,38,39,40,42,61,82,84,85,86,87,88,89,90,91,92,93,94,95,...
    96,97,99,103,104,105,106,107,108,109,110,111,112,113,114,115,...
    116,118,137,158,160,161,162,163,164,165,166,167,168,169,170,171,...
    172,173,175,179,180,181,182,183,184,185,186,187,188,189,190,191,...
    192,194,213,234,236,237,238,239,240,241,242,243,244,245,246,247,...
    248,249,251,255,256,257,258,259,260,261,262,263,264,265,266,267,...
    268,270,289,310,312,313,314,315,316,317,318,319,320,321,322,323,...
    324,325,327,331,332,333,334,335,336,337,338,339,340,341,342,343,...
    344,346,364];
inte.boundary(1).numsubelements(13,:) = [4 19 28 29 30 31 32 33 34 ...
    35 36 37 38 39 40 41 53 54 55 56 71 72 73 74 75 76 77 78 79 80 ...
    81 82 83 84 86 89 90 91 92 93 94 95 96 97 98 99 100 101 102 115 ...
    116 129 130 131 132 133 134 135 136 137 138 139 140 141 142 144 ...
```

```
    145 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 166];

try
if appl_mode == 1
    %Select the MEMS Plane Stress application mode
    switch element_type
        case 1
            %Select Point elements
            outputvector = ...
                smps.point(number_of_fingers).numsubelements(subelements,:);
        case 2
            %Select Boundary elements
            outputvector = ...
                smps.boundary(number_of_fingers).numsubelements(subelements,:);
        case 3
            %Select Subdomain elements
            outputvector = ...
                smps.subdomains(number_of_fingers).numsubelements(subelements,:);
        otherwise
            %Error
            disp(['No valid input: element_type= ' ...
                num2str(element_type) ' @ appl_mode= ' ...
                num2str(appl_mode)]);
            outputvector = [];
    end
elseif appl_mode == 2
    %Select the MEMS Electrostatics application mode
    switch element_type
        case 1
            %Select Point elements
            outputvector = ...
                emes.point(number_of_fingers).numsubelements(subelements,:);
        case 2
            %Select Boundary elements
            outputvector = ...
                emes.boundary(number_of_fingers).numsubelements(subelements,:);
        case 3
            %Select Subdomain elements
            outputvector = ...
                emes.subdomains(number_of_fingers).numsubelements(subelements,:);
        otherwise
            %Error
            disp(['No valid input: element_type= ' ...
                num2str(element_type) ' @ appl_mode= ' ...
                num2str(appl_mode)]);
            outputvector = [];
    end
elseif appl_mode == 3
    %Select the domains for Maxwell Tensor integration
    switch element_type
        case 1
            %Select Point elements
            outputvector = ...
                inte.point(number_of_fingers).numsubelements(subelements,:);
        case 2
            %Select Boundary elements
            outputvector = ...
```

```
                inte.boundary(number_of_fingers).numsubelements(subelements,:);
        case 3
            %Select Subdomain elements
            outputvector = ...
                inte.subdomains(number_of_fingers).numsubelements(subelements,:);
        otherwise
            %Error
            disp(['No valid input: element_type= ' ...
                num2str(element_type) ' @ appl_mode= ' ...
                num2str(appl_mode)]);
            outputvector = [];
    end
else
    %Error
    disp(['No valid input: appl_mode= ' num2str(appl_mode)]);
    outputvector = [];
end
catch errormessage
disp(['Error, no valid selection']);
disp(errormessage.message);
outputvector = [];
end
end
```

A.3 Analysing Scripts

A.3.1 Analyse_CPUTime

```
disp('Starting: Analyse_CPUTime');
load([dir_name1 '/sim_db.mat']);
load([dir_name1 '/sim_const.mat']);
extratimer = 0;
if exist([dir_name1 '/Timer.mat'],'file')==2
    %Check to see if seperate timing data is saved for post-processing
    load([dir_name1 '/Timer.mat']);
    extratimer = 1;
end
files = dir([dir_name3 '/FEM Model - ID *.mat']);

TimeSave = zeros(6,1);
for analyse_counter = 1:length(files)
    clear filecontent
    filename = files(analyse_counter).name;
    id = str2double(strrep(strrep(filename,'.mat',''),'FEM Model - ID ',''));
    filepath = [dir_name3 '/' filename];
    if exist([dir_name3 '/' filename], 'file') == 0
        disp(['File ' filename ' does not exist']);
        fileloaded = 0;
    elseif files(analyse_counter).bytes < 1e6
        disp(['File ' filename ' is to small to be right']);
        fileloaded = 0;
    else
        try
            loadvars = {'mesh_refinement','Y_disp', 'CPUTime', 'sim_counter'};
```

```
            filecontent = load(filepath, loadvars{:});
            Sim_ID = filecontent.sim_counter;
            Y_displacement = filecontent.Y_disp;
            CPUTime = filecontent.CPUTime;
            if extratimer == 1
                CPUTime2 = CPUTime + analyse_timer(Sim_ID,...
                    filecontent.mesh_refinement);
            else
                CPUTime2 = 0;
            end
            meanCPUTime = CPUTime/length(Y_displacement);
            meanCPUTime2 = CPUTime2/length(Y_displacement);
            TimeSave(1,id) = Sim_ID;
            TimeSave(2,id) = meanCPUTime;
            TimeSave(3,id) = CPUTime;
            TimeSave(4,id) = filecontent.mesh_refinement;
            TimeSave(5,id) = meanCPUTime2;
            TimeSave(6,id) = CPUTime2;
        catch errormessage
            disp(['Error during loading of ' filename ', file not loaded']);
        end
    end     %if exist filename & filesize
end     %for analyse_counter
save([dir_name1 '/Result_Analyse.mat'], 'TimeSave', '-append');
if exist([dir_name1 '/TimeSave.xls'],'file') == 2
    delete([dir_name1 '/TimeSave.xls']);
end
copyfile('empty/TimeSave_empty.xls',[dir_name1 '/TimeSave.xls']);
xlswrite([dir_name1 '/TimeSave.xls'], TimeSave, 'TimeSave');
xlswrite([dir_name1 '/../TimeSave.xls'], TimeSave, dir_identifier);
xlswrite([dir_name1 '/Result_Analyse.xls'], TimeSave, 'TimeSave');
xlswrite([dir_name1 '/Result_AnalyseY_remesh.xls'], TimeSave, 'TimeSave');
```

A.3.2 Analyse_Output_remesh

```
disp('Starting: Analyse_Output_remesh');
load([dir_name1 '/sim_db.mat']);
load([dir_name1 '/sim_const.mat']);

files = dir([dir_name3 '/FEM Model - ID *.mat']);

failedids(1,:) = [0 0];
%Failedids:
%[ID Error_code]
%   Error #     Error
%   -1      Everything Ok
%   1       File does not exist
%   2       Filesize is to small (<1MB)
%   3       Error during loading
%   4       Error during analysing

TimeSave = zeros(3,1);
id_counter = 0;
analyse_timer = zeros(8,9);

```

```
for analyse_counter = 1:length(files)
    time_start = tic;
    clear filecontent
    clear fem femsave
    filename = files(analyse_counter).name;
    id=str2double(strrep(strrep(filename,'.mat',''),'FEM Model - ID ',''));
    filepath = [dir_name3 '/' filename];
    if exist([dir_name3 '/' filename], 'file') == 0
        disp(['File ' filename ' does not exist']);
        failedids(length(find(failedids)),:) = [id 1];
        fileloaded = 0;
        ResultAnalyse.Errorcode = [1];
    elseif files(analyse_counter).bytes < 1e6
        disp(['File ' filename ' is to small to be right']);
        failedids(length(find(failedids)),:) = [id 2];
        fileloaded = 0;
        ResultAnalyse.Errorcode = [2];
    else
        try
            loadvars = {'mesh_refinement','Y_disp', 'id','CPUTime'};
            femsaves = whos('-file', filepath, 'femsave*');
            numfemsaves = length(femsaves);
            filecontent = load(filepath, loadvars{:});
            %Simulation id if a mesh-sweep is done with the models of Ye
            %id_counter=id-(length(x_vars(1,:))*(9-filecontent.mesh_refinement));
            %Simulation id if a sweep is done for all possible shapes
            id_counter = id;
            ResultAnalyse.Sim_ID = id_counter;
            ResultAnalyse.Mesh_Refinement = filecontent.mesh_refinement;
            fileloaded = 1;
        catch errormessage
            fileloaded = 0;
            disp(['Error during loading of ' filename ', file not loaded']);
            failedids(length(find(failedids)),:) = [id 3];
            ResultAnalyse.Errorcode = [3];
        end
        if fileloaded == 1
            ResultAnalyse.Errorcode = [0];
            disp(['Analysing ' filename]);
            max_Y = 0;
            % Integrate maxwell tensor y-direction
            clear Boundary_Int
            Boundary_Int = getapplmodes(amount_of_fingers, 3, 2, steps_y);
            clear tempfem Es_Force_*
            Es_Force_Y_Maxwell = zeros(1,numfemsaves);
            Es_Force_X_Maxwell = zeros(1,numfemsaves);
            for femsavecounter = 1:numfemsaves
                timer_start(femsavecounter) = tic;
                try
                clear fem
                loadedfem = femsaves(femsavecounter).name;
                load([dir_name3 '/' filename],loadedfem);
                fem = eval(loadedfem);
                Y_counter = str2double(strrep(loadedfem,'femsave',''))+1;
                clear(eval('loadedfem'));
                if (length(fem.sol) > 0)
                %Check to see if a solution is available in the
```

```
                %saved file
                tempfem = fem;
                Es_Force_Y_Maxwell(Y_counter)=postint(fem,'Fes_nTEy_emes',...
                       'unit','N/m', ...
                       'recover','off', ...
                       'dl',Boundary_Int, ...
                       'edim',1, ...
                       'phase',0, ...
                       'geomnum',1, ...
                       'intorder',4, ...
                       'solnum','end');
                Es_Force_X_Maxwell(Y_counter)=postint(fem,'Fes_nTEx_emes',...
                       'unit','N/m', ...
                       'recover','off', ...
                       'dl',Boundary_Int, ...
                       'edim',1, ...
                       'phase',0, ...
                       'geomnum',1, ...
                       'intorder',4, ...
                       'solnum','end');
                %Try to use the virtual displacement method
                %Works with Linear Lagrange elements only!!!!
                try
                    Es_Force_VWM(:,Y_counter)=-cemforce(fem, 'We_emes', ...
                               'dl',2,...
                               'solnum','all');
                catch error
                    Es_Force_VWM(:,Y_counter)=[0 0];
                end
                else
                Es_Force_Y_Maxwell(Y_counter) = 0;
                Es_Force_X_Maxwell(Y_counter) = 0;
                end
                Y_displacement(Y_counter) = filecontent.Y_disp(Y_counter);
                ResultAnalyse.Errorcode = [-1];
                catch error
                    disp(error.message)
                    disp(error.stack)
                    Es_Force_Y_Maxwell(Y_counter) = 0;
                    Es_Force_X_Maxwell(Y_counter) = 0;
                    ResultAnalyse.Errorcode = [4];
                end
            end
            ResultAnalyse.Maxwell.Es_Force_Y = Es_Force_Y_Maxwell;
            ResultAnalyse.Maxwell.Es_Force_X = Es_Force_X_Maxwell;
            ResultAnalyse.VWM.Es_Force_Y = Es_Force_VWM(2,:);
            ResultAnalyse.VWM.Es_Force_X = Es_Force_VWM(1,:);
            ResultAnalyse.Y_displacement = Y_displacement;
            ResultAnalyse.Finger_Shape = x_vars(:,ResultAnalyse.Sim_ID);
            ResultAnalyse.CPUTime = filecontent.CPUTime;
            meanCPUTime = filecontent.CPUTime/length(Y_displacement);
            ResultAnalyse.meanCPUTime = meanCPUTime;
            TimeSave(1,analyse_counter) = ResultAnalyse.Sim_ID;
            TimeSave(2,analyse_counter) = meanCPUTime;
            TimeSave(3,analyse_counter) = filecontent.CPUTime;
            clear Es_Force_* filename1 meanCPUTime
            filename1 = ['Temp/Temp ' num2str(id) '.mat'];
```

```
                clear fem
                fem = tempfem;
                clear tempfem
                save(filename1, 'ResultAnalyse', 'id', 'fem');
            end
        end     %if exist filename & filesize
        a1 = ResultAnalyse.Sim_ID;
        a2 = ResultAnalyse.Mesh_Refinement;
        analyse_timer(a1,a2) = toc(time_start);
        clear ResultAnalyse a1 a2
        clear filecontent
    end     %for analyse_counter
    save('Temp/Temp.mat', 'TimeSave', 'failedids', 'dir_*', 'analyse_timer');
    clear all

    %Clear memory and reload things to avoid memory problems
    load('Temp/Temp.mat');
    copyfile('Temp/Temp.mat',[dir_name1 '/Timer.mat']);
    files = dir('Temp/Temp *.mat');
    stop_id = length(files);
    for i = 1:length(files)
        close all
        load(['Temp/' files(i).name]);
        %Check for different lengths of results for the axis of the plots
        if length(ResultAnalyse.Y_displacement) ≥ ...
                length(ResultAnalyse.Maxwell.Es_Force_Y)
        Y_displacement = linspace(min(ResultAnalyse.Y_displacement), ...
        ResultAnalyse.Y_displacement(length(ResultAnalyse.Maxwell.Es_Force_Y)),...
        length(ResultAnalyse.Maxwell.Es_Force_Y));
        else
            Y_displacement = linspace(min(ResultAnalyse.Y_displacement), ...
                max(ResultAnalyse.Y_displacement), ...
                length(ResultAnalyse.Maxwell.Es_Force_Y));
        end
        Result_Analyse(id) = ResultAnalyse;
        %Plot the ES forces
        figure(1)
        filename1 = [dir_name2 '/Es_Force_Y - ID ' num2str(id) '.jpg'];
        plot(Y_displacement,ResultAnalyse.Maxwell.Es_Force_Y, Y_displacement, ...
            ResultAnalyse.VWM.Es_Force_Y);
        legend('EM Force on lower comb in Y-direction using Maxwell Tensor',...
            'EM Force on lower comb in Y-direction using virtual displacement');
        title(['Electrostatic force @ 10V - ID = ' ...
            num2str(ResultAnalyse.Sim_ID) '@' ...
            num2str(ResultAnalyse.Mesh_Refinement)]);
        print ('-djpeg',filename1);
        figure(2)
        filename1 = [dir_name2 '/Es_Force_X - ID ' num2str(id) '.jpg'];
        plot(Y_displacement,ResultAnalyse.Maxwell.Es_Force_X, ...
            Y_displacement, ResultAnalyse.VWM.Es_Force_X);
        legend('EM Force on lower comb in X-direction using Maxwell Tensor',...
            'EM Force on lower comb in X-direction using virtual displacement');
        title(['Electrostatic force @ 10V - ID = ' ...
            num2str(ResultAnalyse.Sim_ID) '@' ...
            num2str(ResultAnalyse.Mesh_Refinement)]);
        print ('-djpeg',filename1);
        %Plot of the potential field
```

```
    try
        figure(3)
        filename1 = [dir_name2 '/Potential - ID ' num2str(id) '.jpg'];
        fem.xmesh=meshextend(fem, 'report','off');
        postplot(fem, ...
             'tridata',{'V','cont','internal','unit','V'}, ...
             'trimap','jet(1024)', ...
             'solnum','end', ...
             'title',['Electric potential [V] - ID = ' ...
             num2str(ResultAnalyse.Sim_ID) '@' ...
             num2str(ResultAnalyse.Mesh_Refinement)]);
        print ('-djpeg',filename1);
    catch errormessage
        disp('No V-plot, error');
    end
end
clear ResultAnalyse

%Make a matrix of the results to export
Y_length_temp = 0;
Y_max_length = 0;
Y_max_length_id = 0;
for Y_length_counter = 1:length(Result_Analyse(:))
    Y_length_temp = length(Result_Analyse(Y_length_counter).Y_displacement);
    if Y_length_temp > Y_max_length
        Y_max_length = Y_length_temp;
        Y_max_length_id = Y_length_counter;
    end     %if Y_length_temp > Y_max_length
end     %for Y_length_counter
Result_AnalyseYM = zeros(length(Result_Analyse(:))+1, ...
    length(Result_Analyse(Y_max_length_id).Y_displacement)+1);
Result_AnalyseXM = zeros(length(Result_Analyse(:))+1, ...
    length(Result_Analyse(Y_max_length_id).Y_displacement)+1);
Result_AnalyseYM(1,:) = [0 Result_Analyse(Y_max_length_id).Y_displacement];
Result_AnalyseXM(1,:) = [0 Result_Analyse(Y_max_length_id).Y_displacement];
Result_AnalyseYV(1,:) = [0 Result_Analyse(Y_max_length_id).Y_displacement];
Result_AnalyseXV(1,:) = [0 Result_Analyse(Y_max_length_id).Y_displacement];

for to_xls_counter = 1:length(Result_Analyse(:))
    if Result_Analyse(to_xls_counter).Errorcode == -1
        clear lineY lineX
        lineY = [Result_Analyse(to_xls_counter).Sim_ID ...
            Result_Analyse(to_xls_counter).Maxwell.Es_Force_Y];
        lineX = [Result_Analyse(to_xls_counter).Sim_ID ...
            Result_Analyse(to_xls_counter).Maxwell.Es_Force_X];
        xmax = length(lineY);
        Result_AnalyseYM(to_xls_counter+1,1:xmax) = lineY;
        Result_AnalyseXM(to_xls_counter+1,1:xmax) = lineX;
        clear lineY lineX
        lineY = [Result_Analyse(to_xls_counter).Sim_ID ...
            Result_Analyse(to_xls_counter).VWM.Es_Force_Y];
        lineX = [Result_Analyse(to_xls_counter).Sim_ID ...
            Result_Analyse(to_xls_counter).VWM.Es_Force_X];
        xmax = length(lineY);
        Result_AnalyseYV(to_xls_counter+1,1:xmax) = lineY;
        Result_AnalyseXV(to_xls_counter+1,1:xmax) = lineX;
    end
```

```
end

save([dir_name1 '/Result_Analyse.mat'], 'Result_Analyse');
save([dir_name1 '/Failed_Ids.mat'], 'failedids');
save([dir_name1 '/Result_AnalyseX_Maxwell.xls'], 'Result_AnalyseXM', ...
    '-ASCII', '-DOUBLE', '-TABS');
save([dir_name1 '/Result_AnalyseY_Maxwell.xls'], 'Result_AnalyseYM', ...
    '-ASCII', '-DOUBLE', '-TABS');
save([dir_name1 '/Result_AnalyseX_VWM.xls'], 'Result_AnalyseXV', ...
    '-ASCII', '-DOUBLE', '-TABS');
save([dir_name1 '/Result_AnalyseY_VWM.xls'], 'Result_AnalyseYV', ...
    '-ASCII', '-DOUBLE', '-TABS');
```

A.3.3 Analyse_Result_Analyse_mat

```
disp('Starting: Analyse_Result_Analyse_mat.m');

load([dir_name1 '/sim_db.mat']);
load([dir_name1 '/sim_const.mat']);
load([dir_name1 '/Result_Analyse.mat']);
%delete previous output of this m-file
delete([dir_name1 '/Result_Analyse.xls']);
delete([dir_name1 '/Result_AnalyseY_remesh_Maxwell.xls']);
delete([dir_name1 '/Result_AnalyseY_remesh_VWM.xls']);
delete([dir_name1 '/Result_AnalyseY_remesh.xls']);
%copy empty prepared Excel sheets to the correct folder
copyfile('empty/Result_AnalyseY_remesh_empty.xls', ...
    [dir_name1 '/Result_AnalyseY_remesh_Maxwell.xls'])
copyfile('empty/Result_AnalyseY_remesh_empty.xls', ...
    [dir_name1 '/Result_AnalyseY_remesh_VWM.xls'])
copyfile('empty/Result_AnalyseY_remesh_empty2.xls', ...
    [dir_name1 '/Result_AnalyseY_remesh.xls'])

%Put all the available simulation variables in a matrix
available_vars = whos('-file', [dir_name1 '/sim_const.mat']);
numvars = length(available_vars);
Result_Analyse_Consts(numvars,:) = {'',''};
for const_counter = 1:numvars
    Result_Analyse_Consts(const_counter,:) = ...
        {available_vars(const_counter).name, ...
        num2str(eval(available_vars(const_counter).name))};
end
warning off MATLAB:xlswrite:AddSheet %Supress warning in xlswrite
xlswrite([dir_name1 '/Result_Analyse.xls'],Result_Analyse_Consts,'Variables');

%Find the simulation result with the most results
Y_length_temp = 0;
Y_max_length = 0;
Y_max_length_id = 0;
for Y_length_counter = 1:length(Result_Analyse(:))
    Y_length_temp = length(Result_Analyse(Y_length_counter).Y_displacement);
    if Y_length_temp > Y_max_length
        Y_max_length = Y_length_temp;
        Y_max_length_id = Y_length_counter;
    end     %if Y_length_temp > Y_max_length
```

```
end     %for Y_length_counter
try
    Result_AnalyseYM = zeros(length(Result_Analyse(:))+1, ...
        length(Result_Analyse(Y_max_length_id).Y_displacement)+2);
    Result_AnalyseXM = zeros(length(Result_Analyse(:))+1, ...
        length(Result_Analyse(Y_max_length_id).Y_displacement)+2);
    Result_AnalyseYM(1,:)=[0 0 Result_Analyse(Y_max_length_id).Y_displacement];
    Result_AnalyseXM(1,:)=[0 0 Result_Analyse(Y_max_length_id).Y_displacement];
    average_counterY = zeros(length(x_vars(1,:)),...
        length(Result_Analyse(Y_max_length_id).Y_displacement));
    average_counterX = zeros(length(x_vars(1,:)),...
        length(Result_Analyse(Y_max_length_id).Y_displacement));
    average_counterY_num = zeros(length(x_vars(1,:)),...
        length(Result_Analyse(Y_max_length_id).Y_displacement));
    average_counterX_num = zeros(length(x_vars(1,:)),...
        length(Result_Analyse(Y_max_length_id).Y_displacement));
    for to_xls_counter = 1:length(Result_Analyse(:))
        if Result_Analyse(to_xls_counter).Errorcode == -1
        for mean_counter=...
                1:length(Result_Analyse(to_xls_counter).Maxwell.Es_Force_Y)
            if (Result_Analyse(to_xls_counter).Maxwell.Es_Force_Y(mean_counter) > 0)
                average_counterY(Result_Analyse(to_xls_counter).Sim_ID,mean_counter) ...
                    = average_counterY(Result_Analyse(to_xls_counter).Sim_ID,mean_counter) ..
                    + Result_Analyse(to_xls_counter).Maxwell.Es_Force_Y(mean_counter);
                average_counterY_num(Result_Analyse(to_xls_counter).Sim_ID,mean_counter) ...
                    = average_counterY_num(Result_Analyse(to_xls_counter).Sim_ID,mean_counter
                    + 1;
                average_counterX(Result_Analyse(to_xls_counter).Sim_ID,mean_counter) ...
                    = average_counterX(Result_Analyse(to_xls_counter).Sim_ID,mean_counter) ..
                    + Result_Analyse(to_xls_counter).Maxwell.Es_Force_X(mean_counter);
                average_counterX_num(Result_Analyse(to_xls_counter).Sim_ID,mean_counter) ...
                    = average_counterX_num(Result_Analyse(to_xls_counter).Sim_ID,mean_counter
                    + 1;
            end
        end
        clear line*
        lineY = [Result_Analyse(to_xls_counter).Mesh_Refinement ...
            Result_Analyse(to_xls_counter).Sim_ID ...
            Result_Analyse(to_xls_counter).Maxwell.Es_Force_Y];
        lineX = [Result_Analyse(to_xls_counter).Mesh_Refinement ...
            Result_Analyse(to_xls_counter).Sim_ID ...
            Result_Analyse(to_xls_counter).Maxwell.Es_Force_X];
        xmax = length(lineY);
        Result_AnalyseYM(to_xls_counter+1+3*length(x_vars(1,:)),1:xmax) ...
            = lineY;
        Result_AnalyseXM(to_xls_counter+1+3*length(x_vars(1,:)),1:xmax) ...
            = lineX;
        end
    end
    for mean_counter = 1:length(average_counterY(:,1))
        averageY = zeros(1,length(average_counterY_num(1,:)));
        averageX = zeros(1,length(average_counterY_num(1,:)));
        for average_counter = 1:length(average_counterY_num(1,:))
            averageY(1,average_counter) = ...
                average_counterY(mean_counter,...
                average_counter)/average_counterY_num(mean_counter,...
                average_counter);
```

```
            averageX(1,average_counter) = ...
                average_counterX(mean_counter,...
                average_counter)/average_counterX_num(mean_counter,...
                average_counter);
        end
        clear line*
        lineYmin = [-0.10 mean_counter averageY(1,:).*0.9];
        lineXmin = [-0.10 mean_counter averageX(1,:).*0.9];
        lineYavg = [0 mean_counter averageY(1,:)];
        lineXavg = [0 mean_counter averageX(1,:)];
        lineYmax = [0.10 mean_counter averageY(1,:).*1.1];
        lineXmax = [0.10 mean_counter averageX(1,:).*1.1];
        xmax = length(lineYmin);
        Result_AnalyseYM(1+mean_counter,1:xmax) = lineYmin;
        Result_AnalyseXM(1+mean_counter,1:xmax) = lineXmin;
        Result_AnalyseYM(1+mean_counter+...
            length(average_counterY(:,1)),1:xmax) = lineYavg;
        Result_AnalyseXM(1+mean_counter+...
            length(average_counterX(:,1)),1:xmax) = lineXavg;
        Result_AnalyseYM(1+mean_counter+...
            2*length(average_counterY(:,1)),1:xmax) = lineYmax;
        Result_AnalyseXM(1+mean_counter+...
            2*length(average_counterX(:,1)),1:xmax) = lineXmax;
    end
    xlswrite([dir_name1 '/Result_Analyse.xls'], Result_AnalyseXM, ...
        'Maxwell X');
    xlswrite([dir_name1 '/Result_Analyse.xls'], Result_AnalyseYM, ...
        'Maxwell Y');
    xlswrite([dir_name1 '/Result_AnalyseY_remesh_Maxwell.xls'], ...
        Result_AnalyseYM, 'Result_AnalyseY');
    xlswrite([dir_name1 '/Result_AnalyseY_remesh.xls'], ...
        Result_AnalyseYM(1:3*length(x_vars(1,:))+1,:), 'MaxwellY');
    clear A
    A = genvarname(['Maxwell_' dir_identifier]);
    eval([A '=Result_AnalyseYM(1:3*length(x_vars(1,:))+1,:);']);
    filename1 = [dir_name1 '/../Results.mat'];
    if exist(filename1,'file') == 2
        save(filename1,eval('A'), '-append');
    else
        save(filename1,eval('A'));
    end
catch errormessage
    disp('Error during analysis of Maxwell tensor method');
    disp(errormessage.message);
    disp(['Error in ' errormessage.stack(1).name '.m at line ' ...
        num2str(errormessage.stack(1).line)]);
end
try
    Result_AnalyseYV = zeros(length(Result_Analyse(:))+1, ...
        length(Result_Analyse(Y_max_length_id).Y_displacement)+2);
    Result_AnalyseXV = zeros(length(Result_Analyse(:))+1, ...
        length(Result_Analyse(Y_max_length_id).Y_displacement)+2);
    Result_AnalyseYV(1,:)=[0 0 Result_Analyse(Y_max_length_id).Y_displacement];
    Result_AnalyseXV(1,:)=[0 0 Result_Analyse(Y_max_length_id).Y_displacement];
    average_counterY = zeros(length(x_vars(1,:)),...
        length(Result_Analyse(Y_max_length_id).Y_displacement));
    average_counterX = zeros(length(x_vars(1,:)),...
```

```
        length(Result_Analyse(Y_max_length_id).Y_displacement));
    average_counterY_num = zeros(length(x_vars(1,:)),...
        length(Result_Analyse(Y_max_length_id).Y_displacement));
    average_counterX_num = zeros(length(x_vars(1,:)),...
        length(Result_Analyse(Y_max_length_id).Y_displacement));
    for to_xls_counter = 1:length(Result_Analyse(:))
        if Result_Analyse(to_xls_counter).Errorcode == -1
        for mean_counter = 1:length(Result_Analyse(to_xls_counter).VWM.Es_Force_Y)
            if (Result_Analyse(to_xls_counter).VWM.Es_Force_Y(mean_counter) > 0) ...
                    && (Result_Analyse(to_xls_counter).VWM.Es_Force_Y(mean_counter) < 0.1)
                average_counterY(Result_Analyse(to_xls_counter).Sim_ID,mean_counter) ...
                    = average_counterY(Result_Analyse(to_xls_counter).Sim_ID,mean_counter) ..
                    + Result_Analyse(to_xls_counter).VWM.Es_Force_Y(mean_counter);
                average_counterY_num(Result_Analyse(to_xls_counter).Sim_ID,mean_counter) ...
                    = average_counterY_num(Result_Analyse(to_xls_counter).Sim_ID,mean_counter
                    + 1;
                average_counterX(Result_Analyse(to_xls_counter).Sim_ID,mean_counter) ...
                    = average_counterX(Result_Analyse(to_xls_counter).Sim_ID,mean_counter) ..
                    + Result_Analyse(to_xls_counter).VWM.Es_Force_X(mean_counter);
                average_counterX_num(Result_Analyse(to_xls_counter).Sim_ID,mean_counter) ...
                    = average_counterX_num(Result_Analyse(to_xls_counter).Sim_ID,mean_counter
                    + 1;
            end
        end
        clear line*
        lineY = [Result_Analyse(to_xls_counter).Mesh_Refinement ...
            Result_Analyse(to_xls_counter).Sim_ID ...
            Result_Analyse(to_xls_counter).VWM.Es_Force_Y];
        lineX = [Result_Analyse(to_xls_counter).Mesh_Refinement ...
            Result_Analyse(to_xls_counter).Sim_ID ...
            Result_Analyse(to_xls_counter).VWM.Es_Force_X];
        xmax = length(lineY);
        Result_AnalyseYV(to_xls_counter+1+3*length(x_vars(1,:)),1:xmax) ...
            = lineY;
        Result_AnalyseXV(to_xls_counter+1+3*length(x_vars(1,:)),1:xmax) ...
            = lineX;
        end
    end
    for mean_counter = 1:length(average_counterY(:,1))
        averageY = zeros(1,length(average_counterY_num(1,:)));
        averageX = zeros(1,length(average_counterY_num(1,:)));
        for average_counter = 1:length(average_counterY_num(1,:))
            averageY(1,average_counter) = average_counterY(mean_counter,...
                average_counter)/average_counterY_num(mean_counter,...
                average_counter);
            averageX(1,average_counter) = average_counterX(mean_counter,...
                average_counter)/average_counterX_num(mean_counter,...
                average_counter);
        end
        clear line*
        lineYmin = [-0.10 mean_counter averageY(1,:).*0.9];
        lineXmin = [-0.10 mean_counter averageX(1,:).*0.9];
        lineYavg = [0 mean_counter averageY(1,:)];
        lineXavg = [0 mean_counter averageX(1,:)];
        lineYmax = [0.10 mean_counter averageY(1,:).*1.1];
        lineXmax = [0.10 mean_counter averageX(1,:).*1.1];
        xmax = length(lineYmin);
```

```
        Result_AnalyseYV(1+mean_counter,1:xmax) = lineYmin;
        Result_AnalyseXV(1+mean_counter,1:xmax) = lineXmin;
        Result_AnalyseYV(1+mean_counter+...
            length(average_counterY(:,1)),1:xmax) = lineYavg;
        Result_AnalyseXV(1+mean_counter+...
            length(average_counterX(:,1)),1:xmax) = lineXavg;
        Result_AnalyseYV(1+mean_counter+...
            2*length(average_counterY(:,1)),1:xmax) = lineYmax;
        Result_AnalyseXV(1+mean_counter+...
            2*length(average_counterX(:,1)),1:xmax) = lineXmax;
    end
    xlswrite([dir_name1 '/Result_Analyse.xls'], Result_AnalyseXV, ...
        'VWM X');
    xlswrite([dir_name1 '/Result_Analyse.xls'], Result_AnalyseYV, ...
        'VWM Y');
    xlswrite([dir_name1 '/Result_AnalyseY_remesh_VWM.xls'], ...
        Result_AnalyseYV, 'Result_AnalyseY');
    xlswrite([dir_name1 '/Result_AnalyseY_remesh.xls'], ...
        Result_AnalyseYV(1:3*length(x_vars(1,:))+1,:), 'VWMY');
    clear A
    A = genvarname(['VWM_' dir_identifier]);
    eval([A '=Result_AnalyseYV(1:3*length(x_vars(1,:))+1,:);']);
    filename1 = [dir_name1 '/../Results.mat'];
    if exist(filename1,'file') == 2
        save(filename1,eval('A'), '-append');
    else
        save(filename1,eval('A'));
    end
catch errormessage
    disp('Error during analysis of Virtual Work method');
    disp(errormessage.message);
    disp(['Error in ' errormessage.stack(1).name '.m at line ' ...
        num2str(errormessage.stack(1).line)]);
end
try
    Result_Analyse_Mean_Time = zeros(length(Result_Analyse(:))+1, 3);
    for to_xls_counter = 1:length(Result_Analyse(:))
        if Result_Analyse(to_xls_counter).Errorcode == -1
            try
                check = 1;
                temp = Result_Analyse(to_xls_counter).Mesh_Refinement;
                temp = Result_Analyse(to_xls_counter).meanCPUTime;
            catch errormessage
                check = 0;
            end
            if check == 1
                clear line*
                lineY = [Result_Analyse(to_xls_counter).Sim_ID ...
                    Result_Analyse(to_xls_counter).Mesh_Refinement ...
                    Result_Analyse(to_xls_counter).meanCPUTime];
                xmax = length(lineY);
                Result_Analyse_Mean_Time(to_xls_counter+1,:) = lineY;
            end
        end
    end
    xlswrite([dir_name1 '/Result_Analyse.xls'], ...
        Result_Analyse_Mean_Time, 'Mean Simulation Time');
```

```
catch errormessage
    disp('Error during analysis of Average calculation time');
    disp(errormessage.message);
    disp(['Error in ' errormessage.stack(1).name '.m at line ' ...
        num2str(errormessage.stack(1).line)]);
end
```

A.3.4 Analyse_SimQ

```
disp('Starting: Analyse_SimQ');
load([dir_name1 '/Result_Analyse.mat']);
length_RA = length(Result_Analyse);

No_Maxwell = zeros(9,1);
Yes_Maxwell = zeros(9,1);
Q_Maxwell = zeros(9,1);
No_VWM = zeros(9,1);
Yes_VWM = zeros(9,1);
Q_VWM = zeros(9,1);
MW = linspace(1,9,9)';
length_Total = zeros(9,1);

for RA_counter = 1:length_RA
    length_Total(Result_Analyse(RA_counter).Mesh_Refinement) = ...
        length_Total(Result_Analyse(RA_counter).Mesh_Refinement) + ...
        length(Result_Analyse(RA_counter).Y_displacement);
    try
        length_Maxwell=length(Result_Analyse(RA_counter).Maxwell.Es_Force_Y);
        for Maxwell_Counter = 1:length_Maxwell
            if Result_Analyse(RA_counter).Maxwell.Es_Force_Y(Maxwell_Counter) == 0
                No_Maxwell(Result_Analyse(RA_counter).Mesh_Refinement) ...
                    = No_Maxwell(Result_Analyse(RA_counter).Mesh_Refinement) ...
                    + 1;
            else
                Yes_Maxwell(Result_Analyse(RA_counter).Mesh_Refinement) ...
                    = Yes_Maxwell(Result_Analyse(RA_counter).Mesh_Refinement) ...
                    + 1;
            end
        end
    catch errormessage
%           disp('No Maxwell');
%           disp(errormessage.message);
    end
    try
        length_VWM = length(Result_Analyse(RA_counter).VWM.Es_Force_Y);
        for VWM_Counter = 1:length_VWM
            if Result_Analyse(RA_counter).VWM.Es_Force_Y(VWM_Counter) == 0
                No_VWM(Result_Analyse(RA_counter).Mesh_Refinement) = ...
                    No_VWM(Result_Analyse(RA_counter).Mesh_Refinement) + 1;
            else
                Yes_VWM(Result_Analyse(RA_counter).Mesh_Refinement) = ...
                    Yes_VWM(Result_Analyse(RA_counter).Mesh_Refinement) + 1;
            end
        end
    catch errormessage
```

```
%           disp('No VWM');
%           disp(errormessage.message);
    end
end
for i = 1:9
    Q_Maxwell(i) = Yes_Maxwell(i)/length_Total(i);
    Q_VWM(i) = Yes_VWM(i)/length_Total(i);
end
to_xls(:,:) = [MW(:) No_Maxwell(:) Yes_Maxwell(:) Q_Maxwell(:) No_VWM(:) ...
    Yes_VWM(:) Q_VWM(:)];
xlswrite([dir_name1 '/Result_AnalyseY_remesh.xls'], to_xls, 'Quality');
```

A.3.5 Make_Overview

```
clear all
close all
disp('Starting: Make_Overview.m');
load('Temp/Analyse_All_temp.mat');

dirname = strrep(dir_name1,[dir_path '/'],'');
if exist([dir_name1 '/../Overview.mat'],'file') == 0
    %If the overview file doesn't exist, a new matrix should be made
    Overview_matrix(1,:) = {'name'};
else
    %Load the exisiting overview file
    load([dir_name1 '/../Overview.mat']);
    Overview_matrix = Overview';
end
numvars = length(Overview_matrix(:,1));
if exist([dir_name1 '/sim_const.mat'],'file') == 2
    %Load the names of the vars that should be in the overview
    available_vars = whos('-file', [dir_name1 '/sim_const.mat']);
    load([dir_name1 '/sim_const.mat']); %Load them
    clear const_overview var_loc
    for i = 1:numvars
        %Create an empty variable vector (strings, not numerical)
        const_overview(i) = {''};
    end
    const_overview(1) = {dirname};
    for const_counter = 1:length(available_vars)
        clear var_loc
        var_loc = 0;
        %Check if the variable name is already in the matrix
        for i = 1:length(Overview_matrix(:,1))
            if strcmp(Overview_matrix(i,1), ...
                    available_vars(const_counter).name)==1
                var_loc = i;
            end
        end
        %Put the variable value at the right place or add it at the bottom
        if var_loc > 0
            const_overview(var_loc) = ...
                {num2str(eval(available_vars(const_counter).name))};
        else
            for i = 1:length(Overview_matrix(1,:))
```

```
                Overview_matrix(numvars+1,i) = {''};
            end
            Overview_matrix(numvars+1,1) = ...
                {available_vars(const_counter).name};
            numvars = numvars+1;
            const_overview(numvars) = ...
                {num2str(eval(available_vars(const_counter).name))};
        end
    end

    clear var_loc
    var_loc = 0;
    %Check if the simulation run is already in the matrix
    for i = 1:length(Overview_matrix(1,:))
        if strcmp(Overview_matrix(1,i), const_overview(1))
            var_loc = i;
        end
    end
    %Put the vector at the right place or add it to the end
    if var_loc > 0
        Overview_matrix(:,var_loc) = const_overview(:);
    else
        Overview_matrix(:,length(Overview_matrix(1,:))+1) = ...
            const_overview(:);
    end
    %Save stuff
    clear Overview
    Overview = Overview_matrix';
    warning off MATLAB:xlswrite:AddSheet
    delete([dir_name1 '/../Overview.mat']);
    delete([dir_name1 '/../Overview.xls']);
    save([dir_name1 '/../Overview.mat'],'Overview');
    xlswrite([dir_name1 '/../Overview.xls'], Overview);
end
```