COMPUTER AIDED MACRO-MODELING OF THE ELECTROMECHANICS OF THE TILTING ELASTICALLY SUPPORTED PLATE

by

Lynn Daniel Gabbay

B.S., Applied and Engineering Physics
B.S., Computer Science
Cornell University, 1993

Submitted to the

Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degree of

Master of Science

at the

Massachusetts Institute of Technology

May, 1995

© 1995 Massachusetts Institute of Technology
All rights reserved

Signature of Author ____________________
Department of Electrical Engineering and Computer Science
May 19, 1995

Certified by ____________________
Professor Stephen D. Senturia
Barton L. Weller Professor of Electrical Engineering
Thesis Supervisor

Accepted by ____________________
Professor Frederic R. Morgenthaler
Chair, Department Committee on Graduate Students

MASSACHUSETTS INSTITUTE
OF TECHNOLOGY

JUL 17 1995

ARCHIVES

COMPUTER AIDED MACRO-MODELING OF THE ELECTROMECHANICS OF THE TILTING ELASTICALLY SUPPORTED PLATE

by

Lynn D. Gabbay

Submitted to the Department of Electrical Engineering and Computer Science
on May 19, 1995 in partial fulfillment of the requirements
for the Degree of Master of Science
in Electrical Engineering and Computer Science

Abstract

Simulation of the non-linear dynamics of electrostatically actuated deformable MEMS structures involves the computationally expensive task of self-consistently determining the distribution of charges and the structure deformation that its forces cause. However, there is a class of deformable MEMS structures for which computation time can be reduced significantly by approximating the structure as a network of macro-models, which supply the potential and kinetic energies of the system as functions of a reduced set of degrees of freedom. This paper reports techniques for taking a macro-modeled electromechanical system from concept to dynamics simulation. These techniques are applied to the case of an electromechanical circular plate suspended by idealized springs above an infinite ground plane. First, software is developed that uses pre-existing tools available in the MIT MEMCAD package in order to numerically extract capacitance and stored potential energy as a function of the position and orientation of the plate. Second, an algebraic model of capacitance is constructed for the suspended circular plate using an approximate theoretical model, and a non-linear curve fitting algorithm is used to fit a parameterized version of the algebraic model to the capacitance data. Finally, a suite of tools is developed that use the parameterized model in order to simulate and characterize the non-linear dynamics of the suspended circular plate. The dynamics of the macro-modeled suspended plate, as an illustration of the results of this approach, is simulated under a variety of applied voltage waveforms.

Thesis Supervisor: Stephen D. Senturia
Title: Barton L. Weller Professor of Electrical Engineering

Acknowledgments

I would like to thank the numerous people without whom this work could not have been accomplished. First and foremost, I want to thank Professor Stephen Senturia for the endless support he provided, not only through the opportunity he gave me to work on this project, but also through his stellar guidance. His feedback always forced me to look at the task at hand with a different perspective, and invariably I would discover something new; his contribution to this work is unquestionable. Another great thanks must go to Professor Jacob White, whose Numerical Algorithms course better equipped me for this research than any other source. I want to thank Dr. John Gilbert for his support with the MEMBase library. In dealing with MEMBase, John was an encyclopedia of knowledge, a genius of programming technique, and a magician of bug fixing. Thanks must also go to Peter Osterberg, who flattened the learning curve of the I-DEAS macro language for me. Also, I need to thank Eckart Jansen and Professor Jeffrey Lang for making the design and code of PostCap publicly available. Most of all, I would like to thank ARPA for supporting my research (Contract No. J-FBI-92-196).

I would like to extend a special thanks to Scotti Fuller for her tremendous assistance with everything from graphics embedding to administration details.

Finally, I would like to thank my dearest friends and family. A warm thanks goes to Lucy Tsirulnik for proof-reading this document far more times than I think she wanted to. I also want to thank my sister, Sherri, and my mother, whom I have never referred to by any name other than Mom. They have always pushed me to excel, and for that I am eternally grateful. I dedicate this thesis to the memory of my father, Professor Edmond J. Gabbay.

Table of Contents

Abstract 3
Acknowledgments 5
Table of Contents 7
List of Figures 9
List of Tables 11
Chapter 1 Introduction 13
Chapter 2 Capacitance Extraction 17
2.1 Single State Capacitance Extraction 18
2.2 Multiple State Capacitance Extraction 19
2.2.1 Process A - Manual 19
2.2.2 Process B - AutoGen and AZ 21
2.2.3 Process C - Churn 22
2.3 Suspended Circular Plate Multiple State Capacitance Extraction 23
2.3.1 State Space Sample Points 23
2.3.2 Mesh Construction 24
2.3.3 Extracted Capacitance Information 26
Chapter 3 Numerical Model 29
3.1 Algebraic Model 29
3.1.1 Parallel Plate Approximation 30
3.1.2 Wedge Approximation 31
3.1.3 Relationship Between Parallel Plate and Wedge Approximations . 32
3.1.4 Suspended Circular Plate Algebraic Model 33
3.2 Fitting an Algebraic Model to Numerical Data 35
Chapter 4 Dynamics Simulation 43
4.1 Modularizing the Equations of Motion 44
4.2 Modular Equations of Motion for the Suspended Circular Plate 46
4.2.1 Electrostatic Contribution 47
4.2.2 Mechanical Contribution 48
4.2.3 Inertia Matrix 51
4.3 Equilibrium and Normal Modes 52
4.3.1 Equilibrium 52
4.3.2 Normal Modes 53
4.4 Suspended Circular Plate Simulation 54
Chapter 5 Conclusion 61
Appendix A Code 63
A.1 Capacitance Extraction 63
A.1.1 AutoGen 63
A.1.2 AZ 66
A.1.3 ReCap 66

recap.C 66
FCinfo.H 72
Matrix.H 74
A.1.4 Churn 80
churn.c 80
churn_fns.h 82
churn_fns.c 82
A.2 Numerical Model 85
A.2.1 Non-Linear Fitting 85
nlfit.m 85
mrqmin.m 86
mrqcof.m 86
prec.m 87
capfn.m 87
readCap.m 88
A.3 Dynamics Simulation 88
A.3.1 Initialization 88
globals.m 88
load_all.m 88
load_plate.m 88
load_springs.m 89
A.3.2 Acceleration Function 89
d2xdt2.m 89
capacitor_contrib.m 90
q_cap.m 90
capacitance.m 90
voltage.m 91
spring_contrib.m 91
q_spr.m 92
A.3.3 Equilibrium Determination 92
equilibrium.m 92
F.m 92
djfnewton.m 93
A.3.4 Normal Mode Determination 94
modes.m 94
Appendix B Simulation Results 95
B.1 Structure Specification 95
B.2 Experiments and Results 97
B.2.1 Near Equilibrium Dynamics 97
B.2.2 Voltage Ramp 119
B.2.3 Varied Voltage Ramp Rate 140
References 153

List of Figures

1. Suspended Circular Plate Structure 15
2. Parameters for the Suspended Circular Plate 15
3. Path from Structure Concept to Capacitance 19
4. Process A - Manual 20
5. Process B - AutoGen and AZ 21
6. Process C - Churn 23
7. Capacitance Extraction Sample Points in State Space
for the Suspended Circular Plate 24
8. Circular Plate Mesh 25
9. Ground Plane Mesh 25
10. Assembled Plate and Ground Plane FEM 26
11. Suspended Circular Plate Extracted Capacitance 27
12. Parallel Plate Approximation 30
13. Wedge Approximation 31
14. Suspended Circular Plate - Diagram for Determining
Algebraic Model of Capacitance 34
15. Comparison of Theoretically Based Algebraic Model
with Capacitance Data Extracted by FastCap 36
16. Comparison of Unfitted and Fitted Algebraic Models
with Capacitance Data Extracted by FastCap 40
17. Translational Beam Deflection 49
18. Rotational Beam Deformation 49
19. Simulated Suspended Plate Structure Diagram 55
20. Sample Dynamics 56
21. Sample Dynamics - Mode Projection 58
22. Simulated Suspended Circular Plate Structure Diagram 96
23. Near Equilibrium Dynamics - 1D Structure - g 99
24. Near Equilibrium Dynamics - 1D Structure - Normal Modes 100
25. Near Equilibrium Dynamics - 2D Structure - g 102
26. Near Equilibrium Dynamics - 2D Structure - φ_x 103
27. Near Equilibrium Dynamics - 2D Structure - Principle Tilt 104
28. Near Equilibrium Dynamics - 2D Structure - Normal Modes 105
29. Near Equilibrium Dynamics - 2D+ Structure - g 107
30. Near Equilibrium Dynamics - 2D+ Structure - φ_x 108
31. Near Equilibrium Dynamics - 2D+ Structure - φ_y 109
32. Near Equilibrium Dynamics - 2D+ Structure - Principle Tilt 110
33. Near Equilibrium Dynamics - 2D+ Structure - Normal Modes 111
34. Near Equilibrium Dynamics - 3D Structure - g 113
35. Near Equilibrium Dynamics - 3D Structure - φ_x 114

36. Near Equilibrium Dynamics - 3D Structure - φ_y 115
37. Near Equilibrium Dynamics - 3D Structure - Principle Tilt 116
38. Near Equilibrium Dynamics - 3D Structure - Normal Modes 117
39. Voltage Ramp Dynamics - 1D Structure - g 120
40. Voltage Ramp Dynamics - 1D Structure - Normal Modes 121
41. Voltage Ramp Dynamics - 2D Structure - g 123
42. Voltage Ramp Dynamics - 2D Structure - φ_x 124
43. Voltage Ramp Dynamics - 2D Structure - Principle Tilt 125
44. Voltage Ramp Dynamics - 2D Structure - Normal Modes 126
45. Voltage Ramp Dynamics - 2D+ Structure - g 128
46. Voltage Ramp Dynamics - 2D+ Structure - φ_x 129
47. Voltage Ramp Dynamics - 2D+ Structure - φ_y 130
48. Voltage Ramp Dynamics - 2D+ Structure - Principle Tilt 131
49. Voltage Ramp Dynamics - 2D+ Structure - Normal Modes 132
50. Voltage Ramp Dynamics - 3D Structure - g 134
51. Voltage Ramp Dynamics - 3D Structure - φ_x 135
52. Voltage Ramp Dynamics - 3D Structure - φ_y 136
53. Voltage Ramp Dynamics - 3D Structure - Principle Tilt 137
54. Voltage Ramp Dynamics - 3D Structure - Normal Modes 138
55. Ramp Time 10^{-3} - g 141
56. Ramp Time 10^{-3} - φ_x 142
57. Ramp Time 10^{-3} - φ_y 143
58. Ramp Time 10^{-3} - Principle Tilt 144
59. Ramp Time 10^{-3} - Normal Modes 145
60. Ramp Time 10^{-5} - g 147
61. Ramp Time 10^{-5} - φ_x 148
62. Ramp Time 10^{-5} - φ_y 149
63. Ramp Time 10^{-5} - Principle Tilt 150
64. Ramp Time 10^{-5} - Normal Modes 151

List of Tables

1. Non-Linear Fit Results 39
2. Equilibrium State for 1D, 2D, 2D+, and 3D Structures at 150 Volts 97
3. Mode Information for 1D Structure at 150 Volts 97
4. Mode Information for 2D Structure at 150 Volts 98
5. Mode Information for 2D+ Structure at 150 Volts 98
6. Mode Information for 3D Structure at 150 Volts 98

CHAPTER 1 Introduction

Technology benefits from computer aided design (CAD); however, the appropriate CAD tools are not always available or well developed. This is the case for the field of microelectromechanical systems (MEMS) [1-4]. MEMS structures are machined on semiconductor wafers using existing VLSI technologies. The types of structures can range from gears and motors to deformable thin membranes. The dynamics of electrostatically actuated MEMS structures involves tne tight coupling of electrostatic and mechanical forces. In the quasi-static limit, the distribution of charges and the effect of its forces upon the deformation of the structure must be determined self-consistently. In general, this solution only addresses stable states, where potential energy is minimized and kinetic energy is neglected. To extend to the non-linear dynamics case, kinetic energy must be included. Unfortunately, however, the numerical simulation of the general distributed non-linear dynamical system is computationally difficult.

In some cases, it may be practical to model a system as a network of lumped-element macro-models, each having only a small number of degrees of freedom. Each macro-model should be an analytical function that exhibits the

correct dependence upon structural dimensions and material properties while still agreeing with full three-dimensional physical simulation. The macro-model paradigm has three levels [5]. At the lowest level, there is the full three-dimensional physical simulation. In the approach taken here, the capacitance and mechanical stored energy of the system are simulated as functions of a restricted set of displacement and orientation coordinates. The resulting numerical data must then be synthesized into an algebraic form. Therefore, the next level is the development of a functional form for the system. The approach used here is to start with a simplified analytical expression derived from a physical model and enhance the functional form with additional parameters that permit a fit to the three-dimensional simulation. The highest level is full dynamical analysis of the system, which involves the construction of the dynamical equations of motion using the analytical representations of the capacitance and stored energy, and the direct integration of the non-linear equations of motion in order to simulate the dynamics of the system.

The goal of this research is to apply this macro-model paradigm to a particular case, and thus glean insight into the issues that concern the macro-modeling of MEMS structures. This research applies the paradigm to the case of a tilting, circular, capacitive plate suspended by compliant beams that are modeled as linear springs above an infinite ground plane, as depicted in Figure 1. Each beam is treated as having two modes of energy storage: bending and torsion. A voltage is applied between the plate and the ground plane and causes an electrostatic attractive force that draws the plate toward the ground plane. The plate is assumed to be rigid, and therefore the deformation of the system at any given time can be described by three parameters. The first is the distance from the ground plane to the center of the bottom face of the plate; this is referred to as the gap g. The remaining two parameters describe the tilt of the plate relative to an x-y axis projected up to the plane $z = g$; these are referred to as the tilt angles φ_x and φ_y. For any combination of φ_x and φ_y an axis can be found that remains in the plane of the untilted plate. Thus, the tilting can be characterized alternately by a principle tilt angle φ_p and a principle axis angle φ_l. All of these parameters are

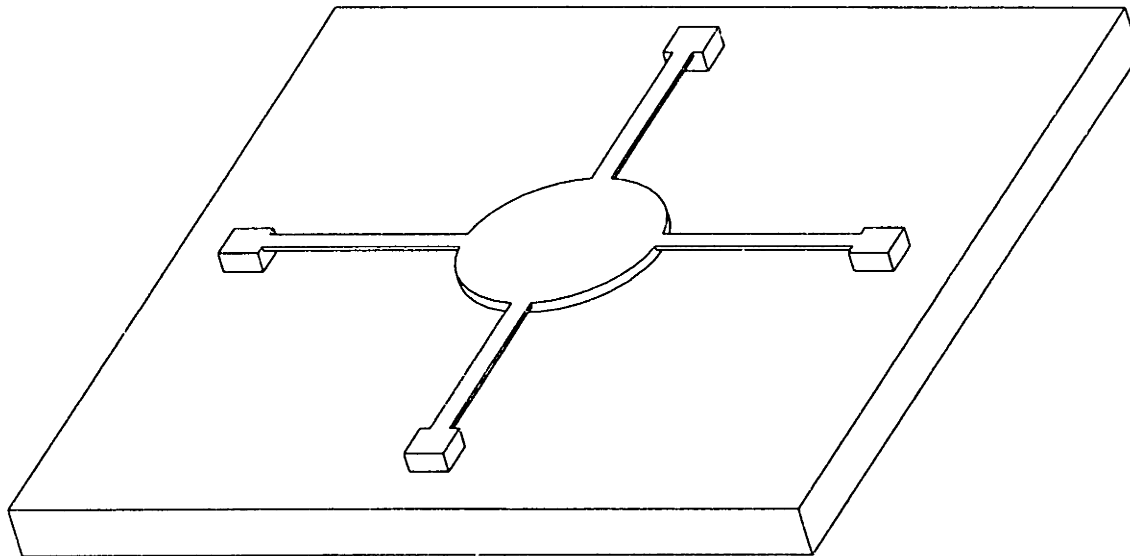


Figure 1: Suspended Circular Plate Structure

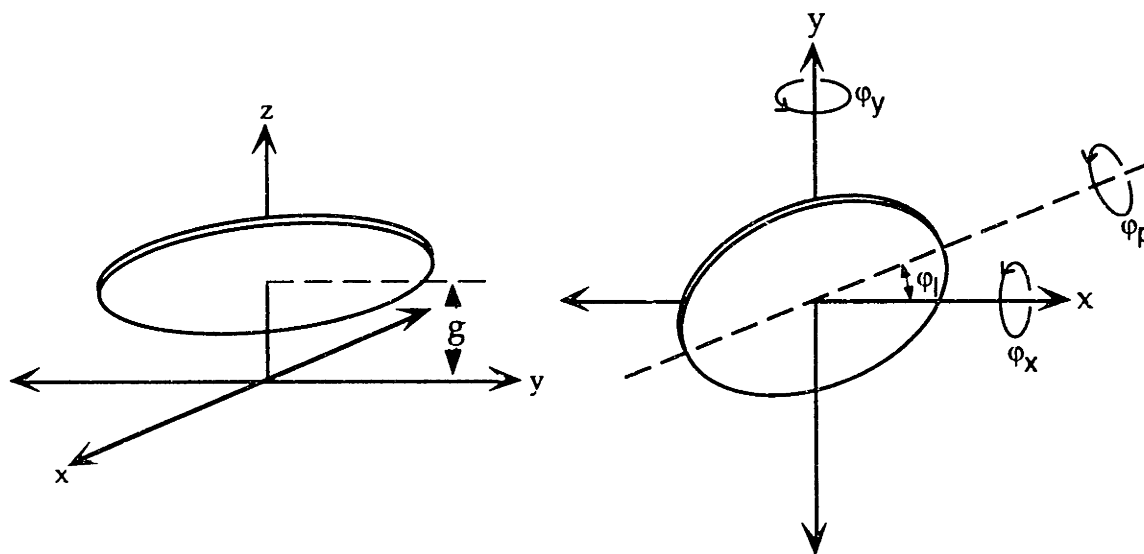


Figure 2: Parameters for the Suspended Circular Plate

depicted in Figure 2. The state of the system at any given time can be expressed by six parameters: g, φ_x, φ_y and their time derivatives. Throughout this report, the term "state" is used to refer to only the position parameters g, φ_x, and φ_y in order to illustrate the dependence of the energy domains upon position, even though there are a total of six state variables, the positive coordinates and their time derivatives, as explained below.

In the suspended circular plate system, there are five energy storage

elements, four beams and one capacitor. These elements contribute to the number of state variables; however, there are two constraints to the system that restrict the independence of the storage elements. The first is the rigidness of the suspended plate. The energy stored in each of the beams is a function of the four coordinates of attachment of the beams with the plate. Because the plate is rigid, one of these coordinates can be determined by the other three. Thus, there are only three independent energy storage elements among the four beams. The second constraint is the fact that the capacitor is attached to a voltage source. The electrostatic energy in the capacitor is determined by the charge on the plate, but the attached voltage source fixes that charge. This eliminates the independence of the capacitor storage element. Thus, the remaining independent energy storage elements are the three beams, each of which contributes one state variable for position and one for an associated inertial term. This yields six state variables for the suspended circular plate system.

This paper presents the techniques for carrying the concept of a system through the various levels of macro-modeling using the suspended circular plate as an example. In Chapter 2, an efficient process is described that is designed to perform full three-dimensional simulation of the capacitance of the structure over a range of deformations in g and φ_p. In Chapter 3, an algebraic model is constructed that represents the capacitance of the suspended circular plate as a function of g and φ_p. In Chapter 4, a simulator is implemented that uses this algebraic model to predict the dynamics of the suspended circular plate.

CHAPTER 2

Capacitance Extraction

In order to macro-model the electrostatic nature of an electromechanical structure, a numerical model must be constructed that represents the electrostatic energy of the system as a function of the deformational state of the structure. For a linear capacitor connected to a voltage source V, the electrostatic energy is equivalent to $\frac{1}{2}CV^2$, where the capacitance C is the only state dependent parameter in the function. Thus, the stored electrostatic energy can be completely described by modeling the capacitance as a function of state.

In order to model the capacitance of a structure, the capacitance is determined by three-dimensional simulation over a range of possible states. Then, a numerical model can be constructed to agree with this data. The process of extracting the capacitance of a structure at a single state can take an hour or more. This time comes not only from the actual three-dimensional simulation but also from the preparation of a model of the structure for simulation and the post-processing of the simulation output. It is therefore necessary to devise an efficient technique that will repeat the single state capacitance extraction process over multiple states.

This chapter presents the design of a multiple state capacitance extraction process using existing tools in the MIT MEMCAD package [6]. First, the single state capacitance extraction process is described. Then, three successive designs of a multiple state capacitance extraction process are presented. Finally, the last of these is used to perform multiple state capacitance extraction for the suspended circular plate structure.

2.1 Single State Capacitance Extraction

There are three tools from the MIT MEMCAD package that are used in the single state capacitance extraction process. The first is I-DEAS, a commercial CAD package, capable of constructing three-dimensional solid models and meshed models (FEMs) from these solid models [7]. The second is FastCap, a three-dimensional capacitance extraction program that, given a finite element model with a surface mesh of a set of conductors, determines the capacitance matrix for the conductors [8]. The third is MEMBase, a suite of tools and a library of C++ functions that enables the in-memory manipulation of meshed structures [9].

The process of single state capacitance extraction is outlined in Figure 3. First, a solid model of the structure is constructed in I-DEAS. This model is then surface meshed, creating a meshed model of the structure, which will be referred to here as a finite element model (FEM). This FEM is written to disk, and I-DEAS may be exited. I-DEAS can write an FEM according to several file formats. However, it is incapable of writing in a format that FastCap accepts. MEMBase includes an executable that can translate an I-DEAS Universal (UNV) FEM file format to the Patran Neutral File (PNF) FEM file format, which is accepted by FastCap. This executable is used to translate the FEM, and then the new FEM is passed to FastCap.

FastCap returns a capacitance matrix that describes charges in terms of applied voltages measured relative to a ground at infinity. This capacitance matrix must be post-processed, because the electrostatic forces on the structure are derived from the capacitance with voltages measured between conductors,

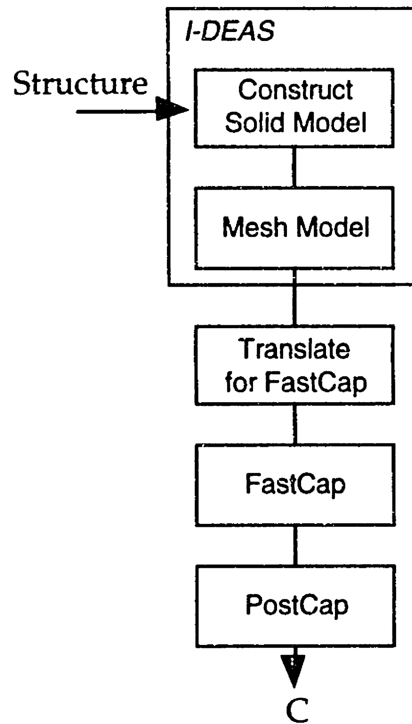


Figure 3: Path from Structure Concept to Capacitance

subject to the constraint of zero charge at infinity. Jansen and Lang presented a discussion of this issue and developed a program, called PostCap [10]. This program performs this post-processing and returns the desired capacitance matrix. Thus, PostCap is used to obtain the desired capacitance information.

2.2 Multiple State Capacitance Extraction

Three designs of a multiple state capacitance extraction processes are presented: a manually performed process, an automated version of the manual process, and an optimized automated process. The optimized automated process is used later for the multiple state capacitance extraction of the suspended circular plate.

2.2.1 Process A - Manual

The first multiple state capacitance extraction process is outlined in Figure

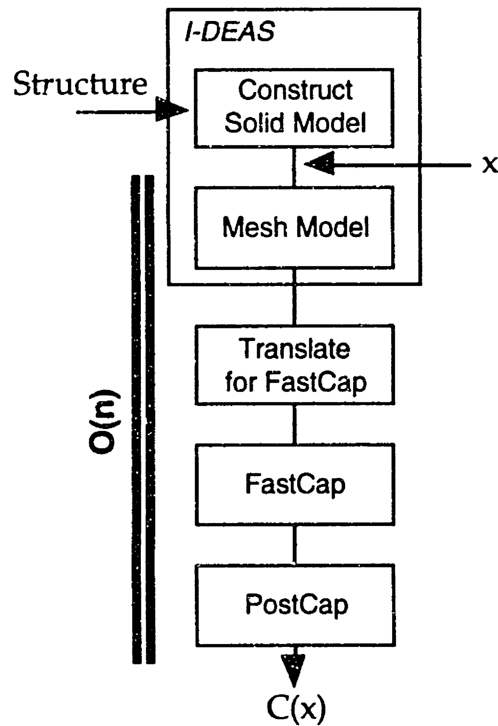


Figure 4: Process A - Manual

4. First, a solid model of the structure is constructed in an arbitrary state x. This is followed by repeating the rest of the capacitance extraction process for the desired set of states. In each pass through the loop, the solid model is deformed to a new state x, the FEM is generated, and the rest of the single state capacitance extraction process is completed, yielding the capacitance at x.

Each step in this process must be executed manually. One advantage of this is that the mesh can be refined for each state of the structure. Overall, however, the manual process is unnecessarily slow and time consuming, taking weeks to extract the capacitances at tens of state points. Therefore, it is imperative that the bulk of this process be automated. Another disadvantage of this technique is that if there are n states for which capacitance is to be extracted, then nearly all of the steps must be repeated n times. In order to optimize this process, not only must it be automated, but the amount of $O(n)$ work, i.e. the number of steps that are repeated for each extraction state, must be reduced. This can accomplished by introducing the dependence upon state at a later point in the process.

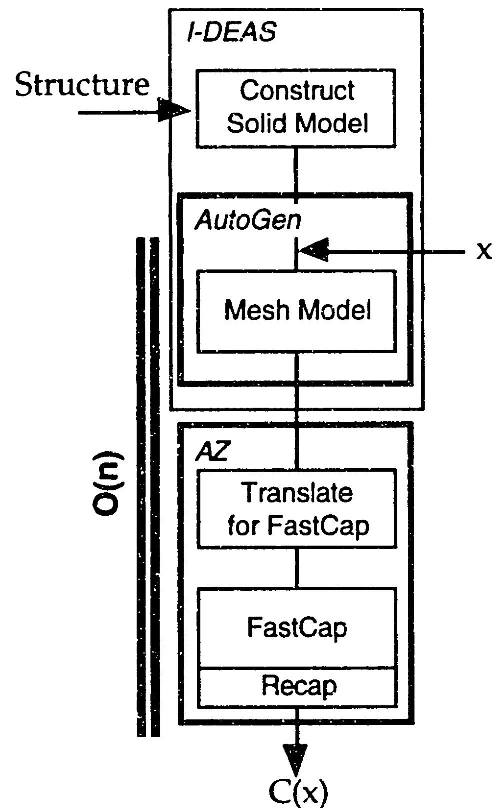


Figure 5: Process B - AutoGen and AZ

2.2.2 Process B - AutoGen and AZ

The next design is an automated version of the previous process. Three programs were written for this purpose. The first is an I-DEAS macro, called AutoGen, that for each desired state in state space deforms the given structure to that state, meshes the deformed structure, and saves the FEM to disk. The second is a UNIX shell script, called AZ, that takes the generated FEMs and passes them through the rest of the capacitance extraction process. However, because PostCap requires user input, it cannot be incorporated directly into an automated process. In order to enable automation, a clone of PostCap was written, called ReCap. ReCap is identical to PostCap in function except that it can be controlled entirely by command line options. Furthermore, it can be used as a filter to the FastCap output. Effectively, FastCap and ReCap can be used in the same command, extracting the desired capacitance in one step. The code for AutoGen, AZ, and ReCap is presented in Appendix A.

Figure 5 depicts the path of this process. In spite of the improvement over

the manual method, there are still several disadvantages to this technique. First, because of the inflexibility of the I-DEAS macro language, AutoGen must be written specifically for each class of structure; e.g. the version of AutoGen developed for this research is written specifically for the suspended circular plate model. Next, there is no communication between AutoGen and AZ; e.g. it is not possible to dynamically choose the state points for capacitance extraction based upon the previous results. Finally, although the process has been automated, the amount of $O(n)$ work has not been reduced. While the dependance upon human interaction is eliminated, the process still takes several days of computation time to generate the FEMs and then several more days to extract the corresponding capacitances. In order to optimize this automated process, the amount of $O(n)$ work is reduced by removing the dependence upon I-DEAS for model manipulation, as explained below.

2.2.3 Process C - Churn

In the third and final design of a multiple state capacitance extraction process, the dependence upon I-DEAS as a model manipulator is eliminated by transferring the task to a new MEMBase application called Churn. The code for Churn is given in Appendix A. Churn is designed to accept any number of I-DEAS Universal files of named finite element models, assemble them into one internal FEM, and deform the internal model repeatedly, writing each deformation to a Patran Neutral File and passing it through FastCap and ReCap.

The process is depicted in Figure 6. By delaying the dependence upon state, the amount of $O(n)$ work is significantly reduced. Only one model needs to be generated for each part of the structure. Churn translates the FEMs only once as the files are read. Because all structure deformation is done in memory, the models never have to be re-read. Although this version of Churn is written with the assumption that a circular plate and a ground plane base will be submitted, and although the only structure deformations that it performs are the translation and rotation of the plate, MEMBase has the capacity to apply complex

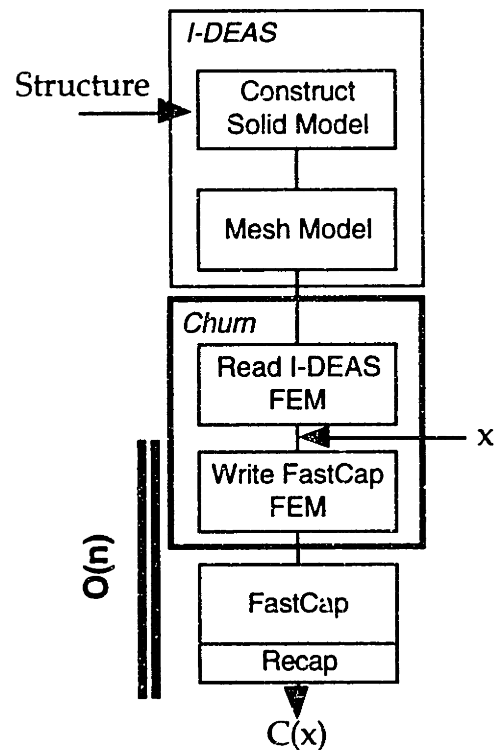


Figure 6: Process C - Churn

deformations to meshes. Thus it will be possible to construct an advanced version of Churn that will allow multiple state capacitance extraction for arbitrary structures with arbitrary state complexity.

2.3 Suspended Circular Plate Multiple State Capacitance Extraction

In order to develop the macro-model for the electrostatic component of the suspended circular plate structure, Churn is used to perform the multiple state capacitance extraction. In this section, the state space sample points, the plate and ground plane FEM meshes, and the extracted capacitance information are presented.

2.3.1 State Space Sample Points

The model of the suspended circular plate system assumes that the plate is constrained to translation in gap and rotation (tilt). Although the plate is allowed

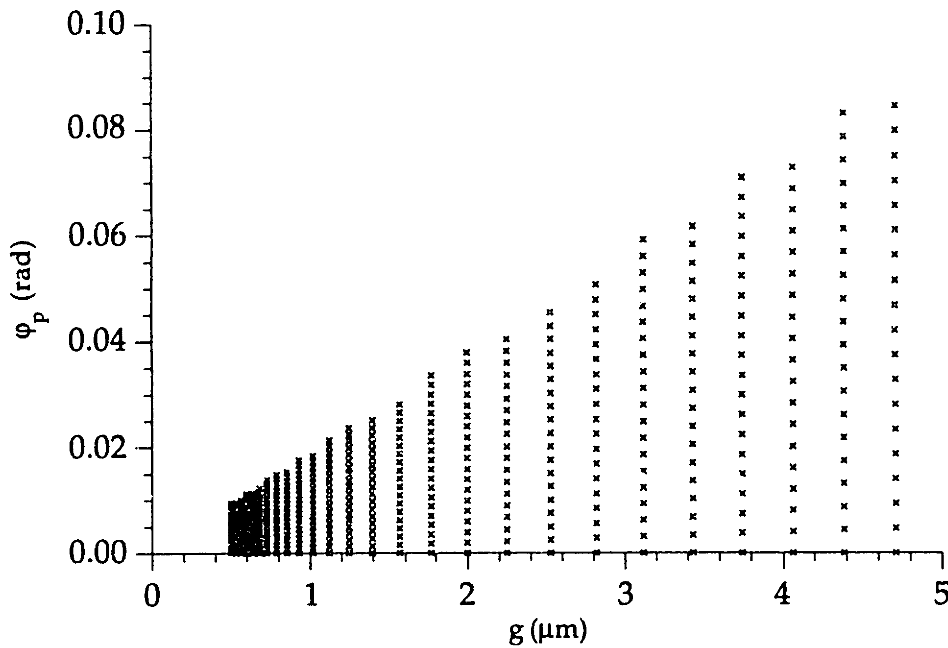


Figure 7: Capacitance Extraction Sample Points in State Space for the Suspended Circular Plate

two degrees of freedom in tilt, the capacitance sees them as being degenerate. This effect is due to the circular symmetry of the plate, which makes the capacitance independent of φ_l and dependent only upon φ_p. Thus, the extraction spans over two state parameters: the gap g and the principle tilt φ_p. The points in state space at which capacitance is to be extracted are shown in Figure 7. The plate cannot tilt beyond the point of contact with the ground plane, restricting the states by $R\sin\varphi_p < g$. Because capacitance changes more rapidly as gap decreases, the density of state sample points is increased as gap is decreased in order to insure an accurate representation of the capacitance.

2.3.2 Mesh Construction

For the case of the suspended circular plate, two meshes are created. The first is the circular plate shown in Figure 8. This surface mesh contains 2756 triangular elements whose side lengths range from 2.5μm to 5μm. The plate has dimensions of a radius of 50μm and a thickness of 5μm. For the second mesh, it is

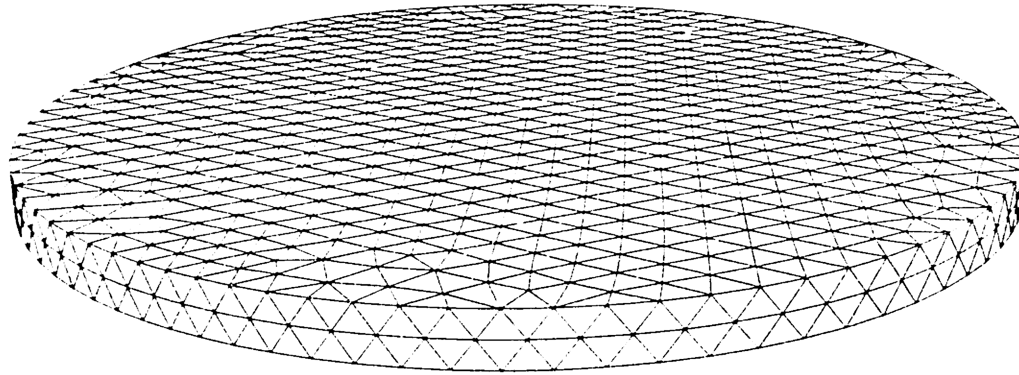


Figure 8: Circular Plate Mesh

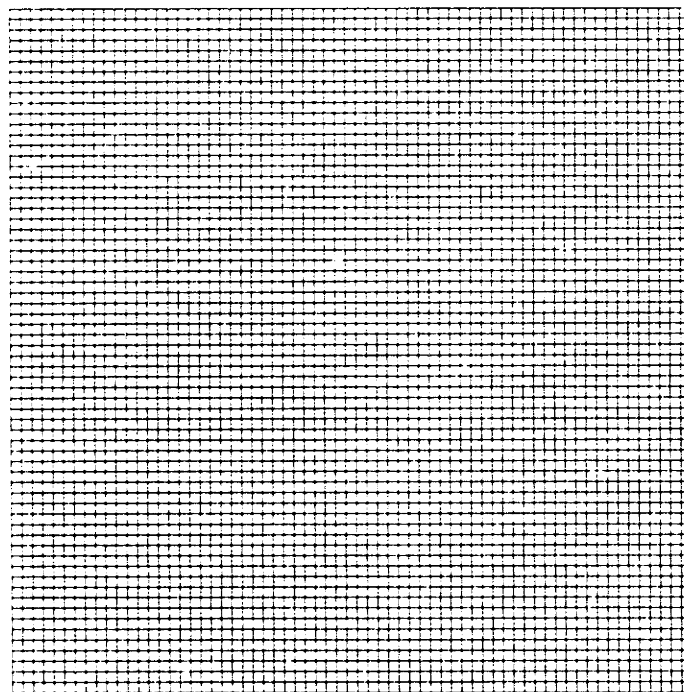


Figure 9: Ground Plane Mesh

impossible to construct an infinite ground plane, so a finite ground plane is constructed with dimensions of 200x200μm. The mesh of this structure contains 4225 (65x65) square shell elements with side lengths of about 3μm as shown in Figure 9.

When constructing finite element models for FastCap, it is important to refine the mesh sufficiently to obtain an accurate representation of the capacitance, yet it must be coarse enough to minimize computation time. The

refinements of the plate and ground plane meshes are chosen by searching for the coarsest combination of refinements such that a slight increase in either refinement does not change the extracted capacitance by more than 1%. An example of an assembled plate and ground plane FEM created by Churn is shown in Figure 10.

2.3.3 Extracted Capacitance Information

Figure 11 presents a sample of the capacitance extracted through Churn. This multiple state capacitance extraction for over 500 data points took one and a half days. In the following chapter, this data is compared to an approximate algebraic model and used to fit a parameterized form of that model, thereby creating a sufficiently accurate mathematical model of the capacitance of the suspended circular plate as a function of state.

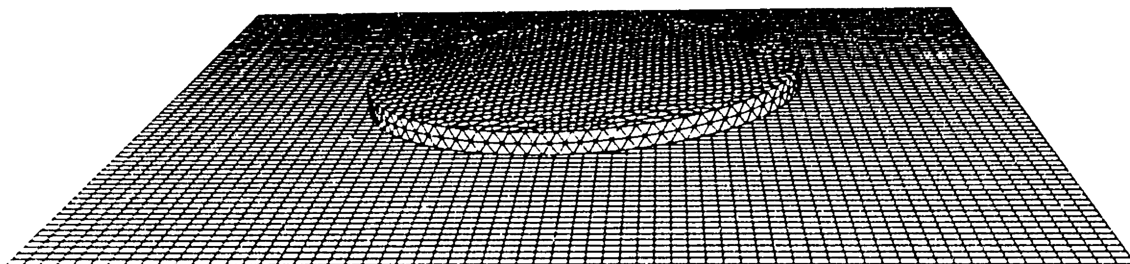


Figure 10: Assembled Plate and Ground Plane FEM

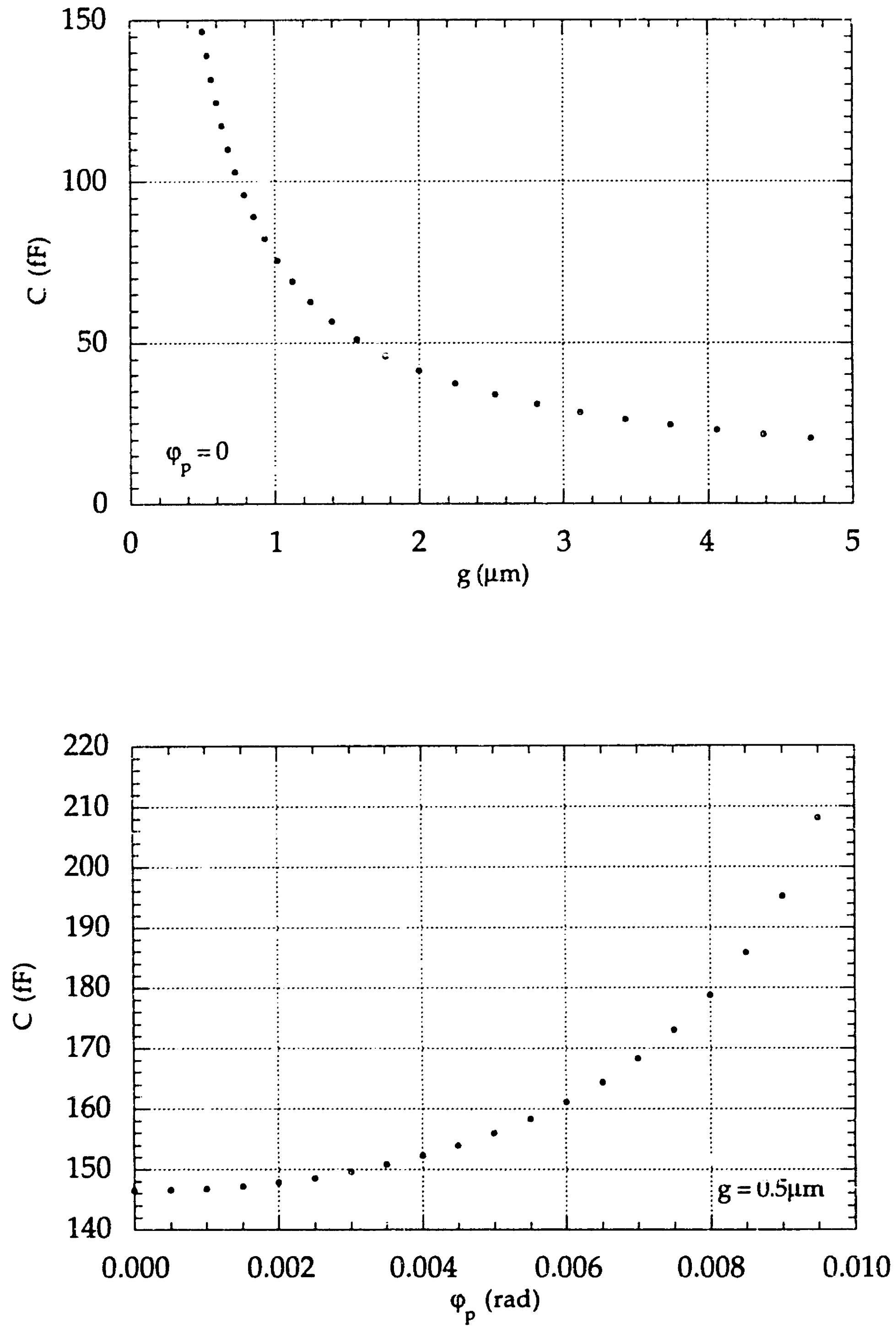
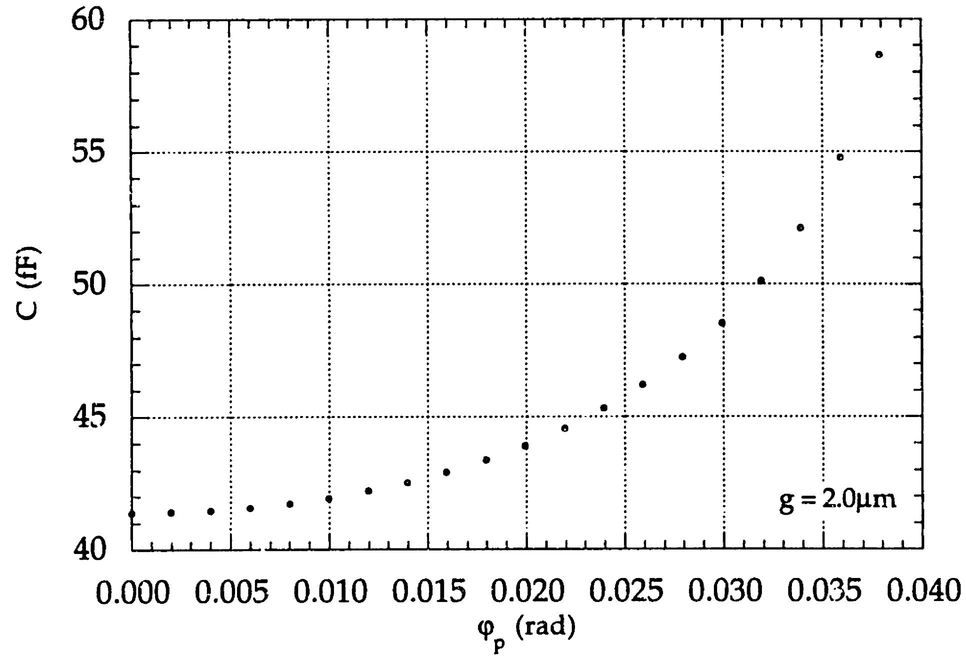


Figure 11: Suspended Circular Plate Extracted Capacitance



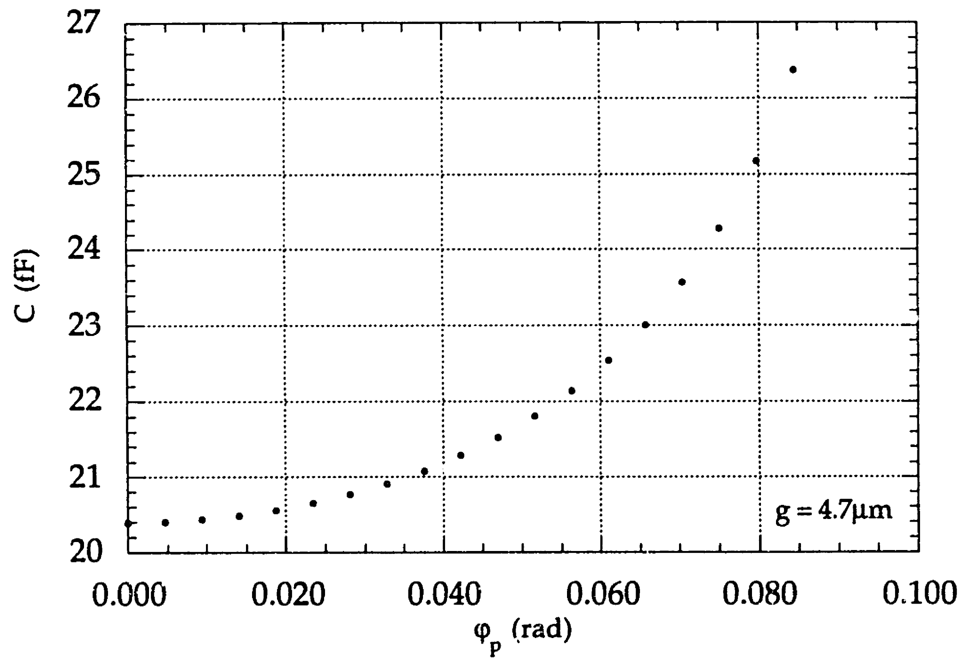


Figure 11 (cont): Suspended Circular Plate Extracted Capacitance

CHAPTER 3 Numerical Model

In order to simulate the dynamics of a macro-modeled system efficiently, each component of the system must be described by a quickly calculable function. These functions should be based upon physics and be modified to correspond with experiment or simulation. In this chapter, a numerical model of the capacitance of the suspended circular plate is constructed. First, an algebraic model is derived based upon an approximation of tilted plate capacitance. Then, this model is parameterized and fit to the data obtained in the multiple state capacitance extraction process performed in the previous chapter.

3.1 Algebraic Model

Two approximations are presented for calculating the capacitance of a flat tilted structure suspended above a ground plane; these are the parallel plate and the wedge approximations. These approximations are then shown to be related by a factor dependent only upon tilt. Finally, the wedge approximation is applied to the suspended circular plate to produce an algebraic model of its capacitance.

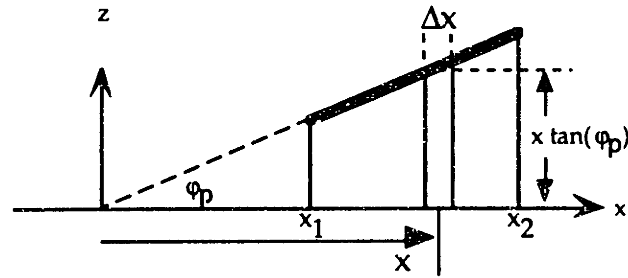


Figure 12: Parallel Plate Approximation

3.1.1 Parallel Plate Approximation

The parallel plate approximation is applied to an arbitrarily shaped flat conductor suspended above a ground plane. The *x-z* cross-section of this such system is shown in Figure 12. The coordinate system is chosen so that the plate is tilted about the *y* axis and the ground plane lies in the *x-y* plane. In this approximation, every differential element Δx of the plate is treated as a parallel plate capacitor, because all of the electric fields are perpendicular to the ground plane.

The capacitance is obtained by calculating the total stored energy in the electric fields. The energy is given by:

$$U_E = \frac{1}{2}\varepsilon_0 \iiint E^2 dv \tag{1}$$

where E is the electric field strength. Assuming that the length of the plate in the y direction is given by $L_x(x)$, where $L_x(x)$ goes to zero at the limits x_1 and x_2, the integral becomes:

$$U_E = \frac{1}{2}\varepsilon_0 \int_{x_1}^{x_2} dx \int_0^{x\tan\varphi_p} dz \int_0^{L_x(x)} E^2 dy \tag{2}$$

In the parallel plate capacitor approximation, the electric field is constant between the plates, and its magnitude is given by:

$$E = \frac{V}{d} \tag{3}$$

where d is the distance between the plates. In this approximation, $d = x\tan\varphi_p$. It

follows that

$$U_E = \frac{1}{2}\varepsilon_0 \int_{x_1}^{x_2} \int_0^{x\tan\varphi_p} \int_0^{L_x(x)} \left(\frac{V}{x\tan\varphi_p}\right)^2 dydzdx \tag{4}$$

Evaluating the two innermost integrals yields

$$U_E = \frac{1}{2}V^2\left[\frac{\varepsilon_0}{\tan\varphi_p}\int_{x_1}^{x_2}\frac{L_x(x)}{x}dx\right] \tag{5}$$

In this expression, the term in square brackets represents total capacitance between the plate and ground plane in the parallel plate approximation limit:

$$C_{pp} = \frac{\varepsilon_0}{\tan\varphi_p}\int_{x_1}^{x_2}\frac{L_x(x)}{x}dx \tag{6}$$

3.1.2 Wedge Approximation

The wedge approximation is also applied to an arbitrarily shaped flat conductor suspended above a ground plane. The wedge approximation is that all electric fields are perpendicular to the plate and curve downward to meet the ground plane perpendicularly. The *x-z* cross-section is shown in Figure 13. Again,

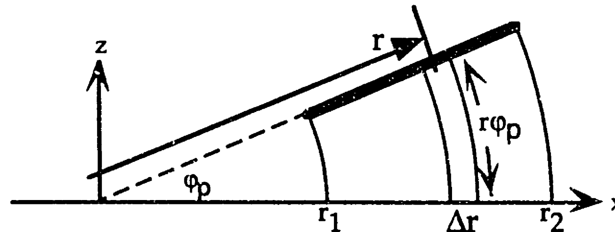


Figure 13: Wedge Approximation

the coordinate system is chosen such that the plate is tilted about the *y* axis and the ground plane lies in the *x-y* plane.

The capacitance is obtained by calculating the total stored energy in the electric fields. Assuming that the length of the plate in the *y* direction is given by

$L_r(r)$, where $L_r(r)$ goes to zero at the limits r_1 and r_2, the integral becomes

$$U_E = \frac{1}{2}\varepsilon_0 \int_{r_1}^{r_2} dr \int_0^{\varphi_p} d\theta \int_0^{L_r(r)} E^2 r dy \tag{7}$$

For the wedge approximation capacitor, the electric field is uniform along the path of the field lines. The distance that the field line travels is $d = r\varphi_p$. Thus, the integral becomes:

$$U_E = \frac{1}{2}\varepsilon_0 \int_{r_1}^{r_2} \int_0^{\varphi_p} \int_0^{L_r(r)} \left(\frac{V}{r\varphi_p}\right)^2 r dy d\theta dr \tag{8}$$

Evaluating the two innermost integrals yields:

$$U_E = \frac{1}{2}V^2 \left[\frac{\varepsilon_0}{\varphi_p} \int_{r_1}^{r_2} \frac{L_r(r)}{r} dr\right] \tag{9}$$

where the term in square brackets represents total capacitance between the plate and ground plane in the wedge approximation limit:

$$C_w = \frac{\varepsilon_0}{\varphi_p} \int_{r_1}^{r_2} \frac{L_r(r)}{r} dr \tag{10}$$

3.1.3 Relationship Between Parallel Plate and Wedge Approximations

The capacitance of a flat plate suspended above a ground plane in the parallel plate approximation has been shown to be

$$C_{pp} = \frac{\varepsilon_0}{\tan\varphi_p} \int_{x_1}^{x_2} \frac{L_x(x)}{x} dx \tag{11}$$

Similarly, the capacitance of such plate in the wedge approximation can be expressed as

$$C_w = \frac{\varepsilon_0}{\varphi_p} \int_{r_1}^{r_2} \frac{L_r(r)}{r} dr \tag{12}$$

These two approximations differ only by a constant factor. The relationship

between x and r is given by

$$x = r\cos\varphi_p \tag{13}$$

Thus, the differential elements are related by

$$dx = dr\cos\varphi_p \tag{14}$$

By definition,

$$L_x(x) = L_r(r) \tag{15}$$

and their limits of integration correspond:

$$x_1 = r_1\cos\varphi_p \text{ and } x_2 = r_2\cos\varphi_p \tag{16}$$

Therefore,

$$\int_{x_1}^{x_2} \frac{L_x(x)}{x} dx = \int_{r_1}^{r_2} \frac{L_r(r)}{r} dr \tag{17}$$

Thus, the parallel plate and wedge approximations for the capacitance of a tilted flat plate suspended above a ground plane are related by

$$(\tan\varphi_p)\, C_{pp} = (\varphi_p)\, C_w \tag{18}$$

In the examples considered here, φ_p never exceeds 0.1 radians, so these models differ by less than 0.4%.

3.1.4 Suspended Circular Plate Algebraic Model

The wedge approximation is chosen to determine the algebraic model of the capacitance of the suspended circular plate. The plate is oriented according to the wedge approximation as depicted in Figure 14. $L_r(r)$ is given by

$$L_r(r) = 2\sqrt{R^2 - (r - r_0)^2} \text{ for } r_0 - R < r < r_0 + R \tag{19}$$

and is zero elsewhere. Thus, the capacitance is expressed by

$$C_w = \frac{\varepsilon_0}{\varphi_p} \int_{r_0 - R}^{r_0 + R} \frac{2\sqrt{R^2 - (r - r_0)^2}}{r} dr \tag{20}$$

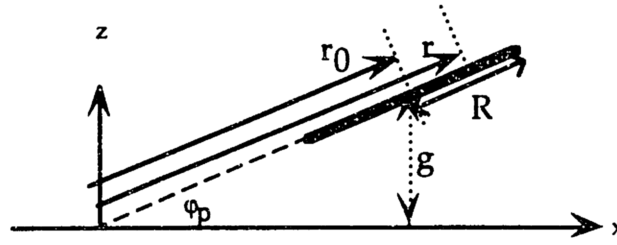


Figure 14: Suspended Circular Plate - Diagram for Determining Algebraic Model of Capacitance

The change of variables of $r \rightarrow r + r_0$ yields

$$C_w = \frac{\varepsilon_0}{\varphi_p} \int_{-R}^{R} \frac{2\sqrt{R^2 - r^2}}{r + r_0} dr \tag{21}$$

Another change of variables of $r \rightarrow Rr$ yields

$$C_w = 2\frac{\varepsilon_0 R}{\varphi_p} \int_{-1}^{1} \frac{\sqrt{1 - r^2}}{r + (r_0/R)} dr \tag{22}$$

According to Gradshteyn and Ryzhik [11]

$$\int_{-1}^{1} \frac{\sqrt{1 - x^2}}{x \pm p} dx = \pm\pi\,[p - \sqrt{p^2 - 1}] \text{ for } p \geq 1 \tag{23}$$

Using this identity, the capacitance becomes

$$C_w = 2\frac{\varepsilon_0 R}{\varphi_p}\pi\left[\left(\frac{r_0}{R}\right) - \sqrt{\left(\frac{r_0}{R}\right)^2 - 1}\right] \tag{24}$$

Substituting $r_0 = g/(\sin\varphi_p)$ into this equation yields the final expression for capacitance:

$$C = 2\pi\frac{\varepsilon_0 R}{\varphi_p \sin\varphi_p}\left[\left(\frac{g}{R}\right) - \sqrt{\left(\frac{g}{R}\right)^2 - \sin^2\varphi_p}\right] \tag{25}$$

This is used as the algebraic model of the capacitance of the suspended circular plate, which is the starting point for the macro-model.

It should be noted that this capacitance function cannot be evaluated numerically at $\varphi_p = 0$ because of the division by zero. Although this function has a limit as $\varphi_p \rightarrow 0$, the φ_p terms in the denominator cause extreme numerical error for small values of φ_p. In order to reduce this error, the first few terms of a Taylor

series expansion of the capacitance about $\varphi_p = 0$ are used to represent the capacitance for small values of φ_p. In this limit, the capacitance becomes

$$C_{\varphi_p \to 0} = \frac{\varepsilon_0 \pi R^2}{g} - \frac{1}{12}\frac{\varepsilon_0 \pi R^2 (2g^2 - 3R^2)}{g^3}\varphi_p^2 + O(\varphi_p^4) \quad (26)$$

The threshold at which the small φ_p limit for capacitance is applied is determined by trial and error to be φ_p=10^{-4} radians.

3.2 Fitting an Algebraic Model to Numerical Data

An algebraic model based upon approximations does not necessarily match the data that would be obtained by experiment or three-dimensional simulation. Figure 15 depicts the algebraic model of the suspended circular plate capacitance alongside the multiple state capacitance extraction data obtained from the three-dimensional simulation of the plate by FastCap. It can be seen that although the algebraic model of Equation 25 has the same form as the data, the model has inaccuracies.

In order to improve the accuracy of Equation 25, fitting parameters are incorporated into the algebraic model. These parameters are chosen to have physical meaning. For the suspended circular plate model, three parameters are defined. The first two parameters are used to define an effective gap $\tilde{g} = \alpha_1 g$ and an effective plate radius $\tilde{R} = \alpha_2 R$. The third parameter is used to account for the fringing of electric fields. A fringing term based upon approximations of the capacitance per unit length of a parallel plate capacitor, $\left(1 + \alpha_3\left(\frac{\tilde{g}}{\tilde{R}}\right)\right)$ is multiplied to the algebraic model [12,13]. The parameterized model becomes

$$C = 2\pi\frac{\varepsilon_0 \tilde{R}}{\varphi_p \sin\varphi_p}\left[\left(\frac{\tilde{g}}{\tilde{R}}\right) - \sqrt{\left(\frac{\tilde{g}}{\tilde{R}}\right)^2 - \sin^2\varphi_p}\right]\left(1 + \alpha_3\left(\frac{\tilde{g}}{\tilde{R}}\right)\right) \quad (27)$$

and in the small φ_p limit:

$$C_{\varphi_p \to 0} = \frac{\varepsilon_0 \pi \tilde{R}^2}{\tilde{g}}\left(1 + \alpha_3\left(\frac{\tilde{g}}{\tilde{R}}\right)\right) - \frac{1}{12}\frac{\varepsilon_0 \pi \tilde{R}^2 (2\tilde{g}^2 - 3\tilde{R}^2)}{\tilde{g}^3}\left(1 + \alpha_3\left(\frac{\tilde{g}}{\tilde{R}}\right)\right)\varphi_p^2 + O(\varphi_p^4) \quad (28)$$

The Levenberg-Marquardt non-linear fitting algorithm is used to find the

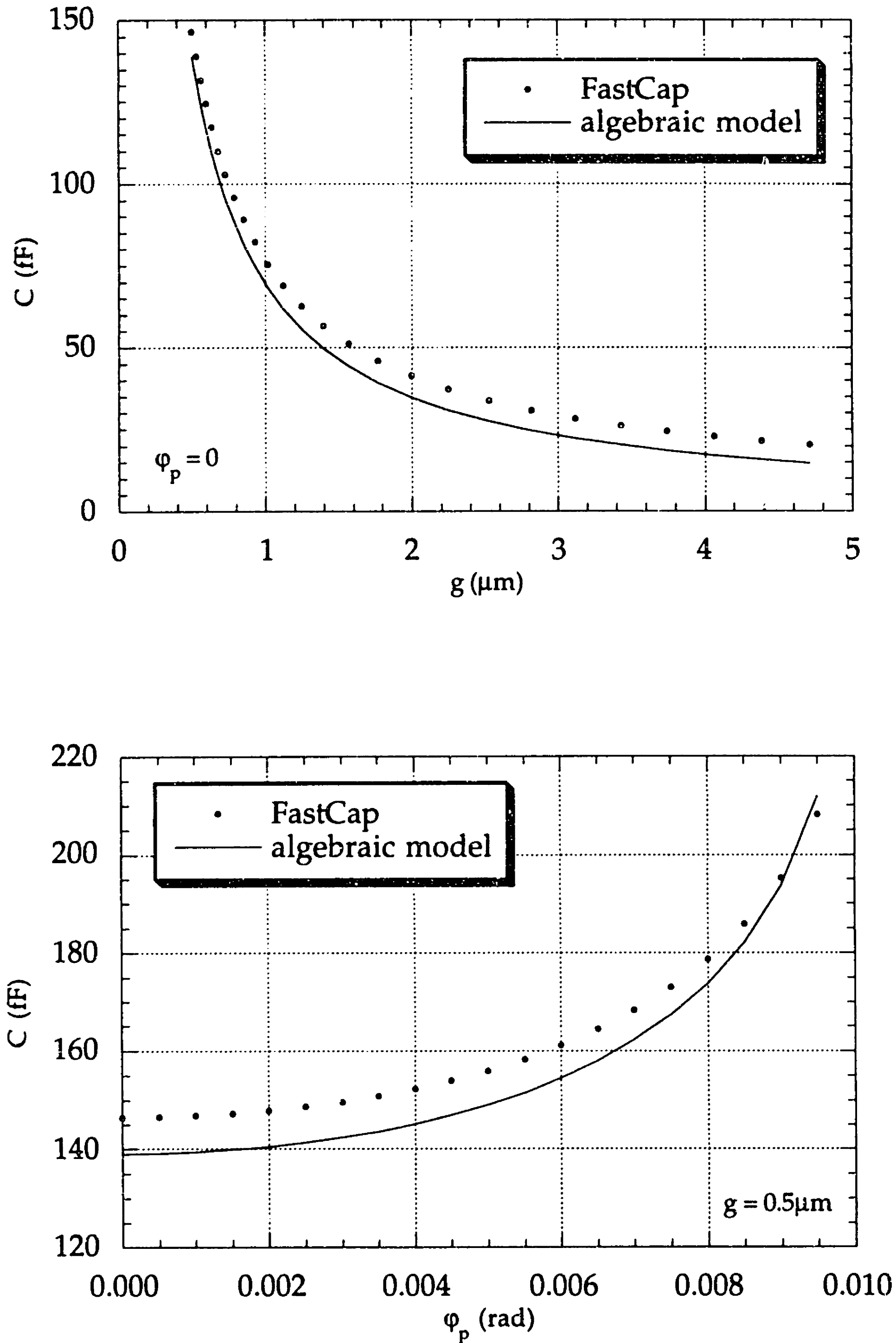
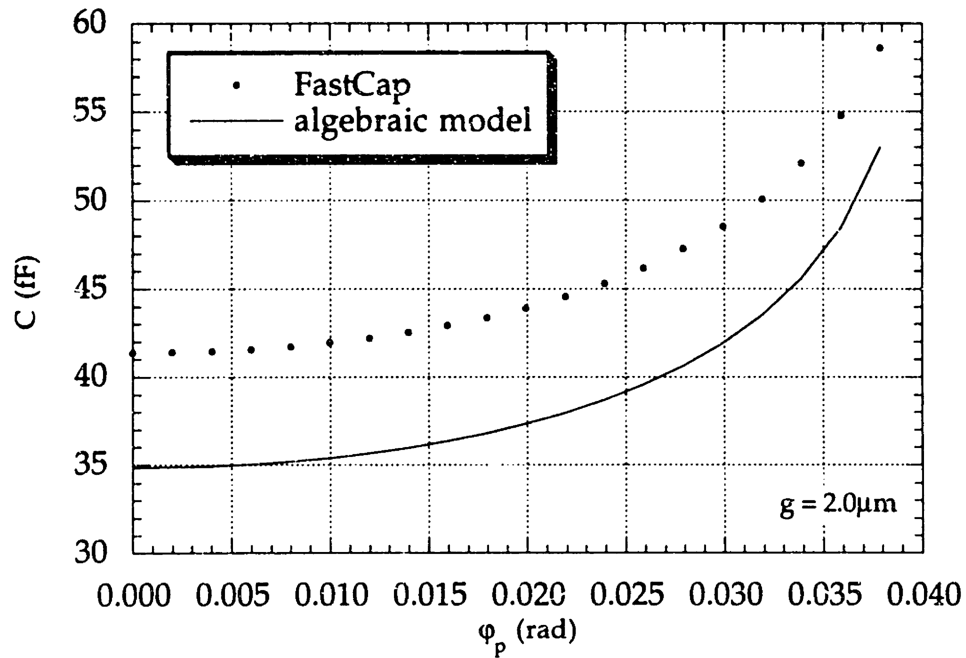


Figure 15: Comparison of Theoretically Based Algebraic Model with Capacitance Data Extracted by FastCap



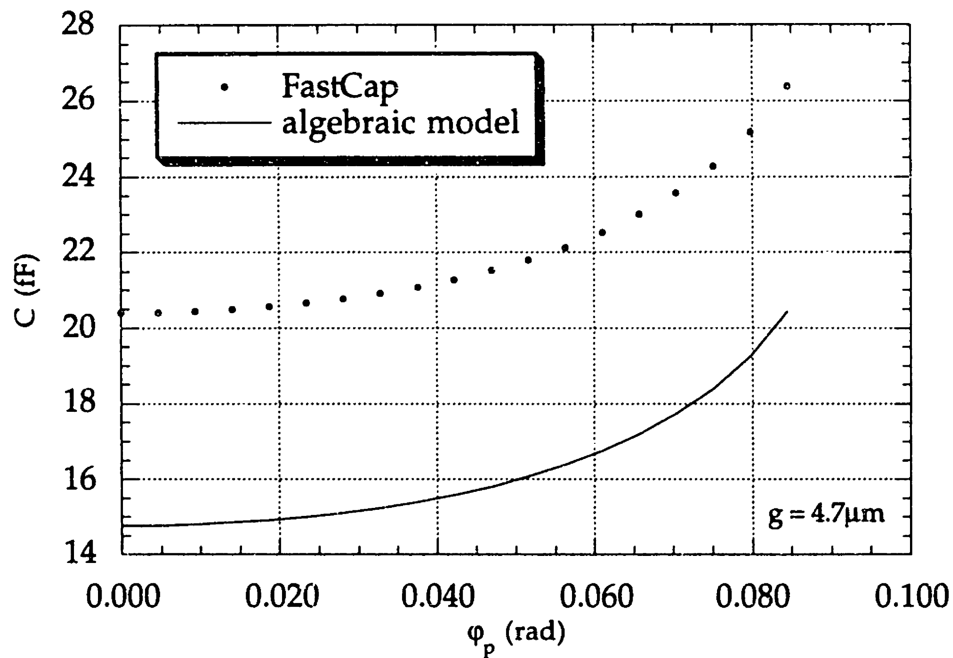


Figure 15 (cont): Comparison of Theoretically Based Algebraic Model with Capacitance Data Extracted by FastCap

optimum values for these alpha parameters. A MATLAB script of this algorithm is written by porting the code presented by Press, Flannery, Teukolsky, and Vetterling [14]. The MATLAB code for this non-linear fitting algorithm is presented in Appendix A. This program requires that derivatives of the function to be fit are taken with respect to the fitting parameters. Two variables are defined to simplify the expressions for these derivatives:

$$r \equiv \frac{\tilde{g}}{\tilde{R}} \tag{29}$$

and

$$s \equiv \sqrt{r^2 - \sin^2\varphi_p} \tag{30}$$

Differentiating the capacitance with respect to the fitting parameters yields

$$\begin{bmatrix} \frac{\partial}{\partial\alpha_1} \\ \frac{\partial}{\partial\alpha_2} \\ \frac{\partial}{\partial\alpha_3} \end{bmatrix} C = 2\pi\frac{\varepsilon_0\tilde{R}}{\varphi_p\sin\varphi_p}\begin{bmatrix} \frac{1}{\alpha_1}\left(r\left(1-\frac{r}{s}\right)(1+\alpha_3 r)+\alpha_3 r(r-s)\right) \\ \frac{-1}{\alpha_2}\left(r\left(1-\frac{r}{s}\right)(1+\alpha_3 r)+\alpha_3 r(r-s)\right) \\ r(r-s) \end{bmatrix} \tag{31}$$

and in the small φ_p limit:

$$\begin{bmatrix} \frac{\partial}{\partial\alpha_1} \\ \frac{\partial}{\partial\alpha_2} \\ \frac{\partial}{\partial\alpha_3} \end{bmatrix} C_{\varphi_p\to 0} = \varepsilon_0\pi\tilde{R}\begin{bmatrix} \frac{1}{\alpha_1}\left(-\frac{1}{r}+\frac{2r^2-6\alpha_3 r-9}{12r^3}\varphi_p^2\right) \\ \frac{1}{\alpha_2}\left(\frac{2+\alpha_3 r}{r}-\frac{2\alpha_3 r^3+4r^2-9\alpha_3 r-12}{12r^3}\varphi_p^2\right) \\ 1-\frac{2r^2-3}{12r^2}\varphi_p^2 \end{bmatrix} \tag{32}$$

This non-linear fitting algorithm also requires that each data point be assigned a standard deviation that represents its significance in the fitting. The multiple state capacitance extraction data for the suspended circular plate are assigned uniform standard deviations.

The fitting parameters are found to be:

Table 1: Non-Linear Fit Results

α_1	1.097
α_2	1.056
α_3	3.847

Figure 16 shows the parameterized model using these results plotted alongside the capacitance data extracted by FastCap and the unparameterized model. This fit corresponds sufficiently for the purposes of this demonstration, and thus it is used as the numerical model of the suspended circular plate capacitance.

In the next chapter, it will be shown that the derivatives of the capacitance with respect to g and φ_p are also needed to determine the electrostatic forces. These derivatives are

$$\begin{bmatrix} \frac{\partial}{\partial g} \\ \frac{\partial}{\partial \varphi_p} \end{bmatrix} C = 2\pi \frac{\varepsilon_0}{\varphi_p \sin\varphi_p} \begin{bmatrix} \alpha_1\left(\left(1-\frac{r}{s}\right)(1+\alpha_3 r)+\alpha_3(r-s)\right) \\ R(1+\alpha_3 r)\left(\frac{\sin\varphi_p\cos\varphi_p}{s}-(r-s)\left(\frac{1}{\varphi_p}+\frac{1}{\tan\varphi_p}\right)\right) \end{bmatrix} \tag{33}$$

and in the small φ_p limit:

$$\begin{bmatrix} \frac{\partial}{\partial g} \\ \frac{\partial}{\partial \varphi_p} \end{bmatrix} C_{\varphi_p \to 0} = \varepsilon_0 \pi \begin{bmatrix} \alpha_1\left(-\frac{1}{r^2}+\frac{2r^2-6\alpha_3 r-9}{12r^4}\varphi_p^2\right) \\ R\frac{(1+\alpha_3 r)(2r^2-3)}{6r^3}\varphi_p \end{bmatrix} \tag{34}$$

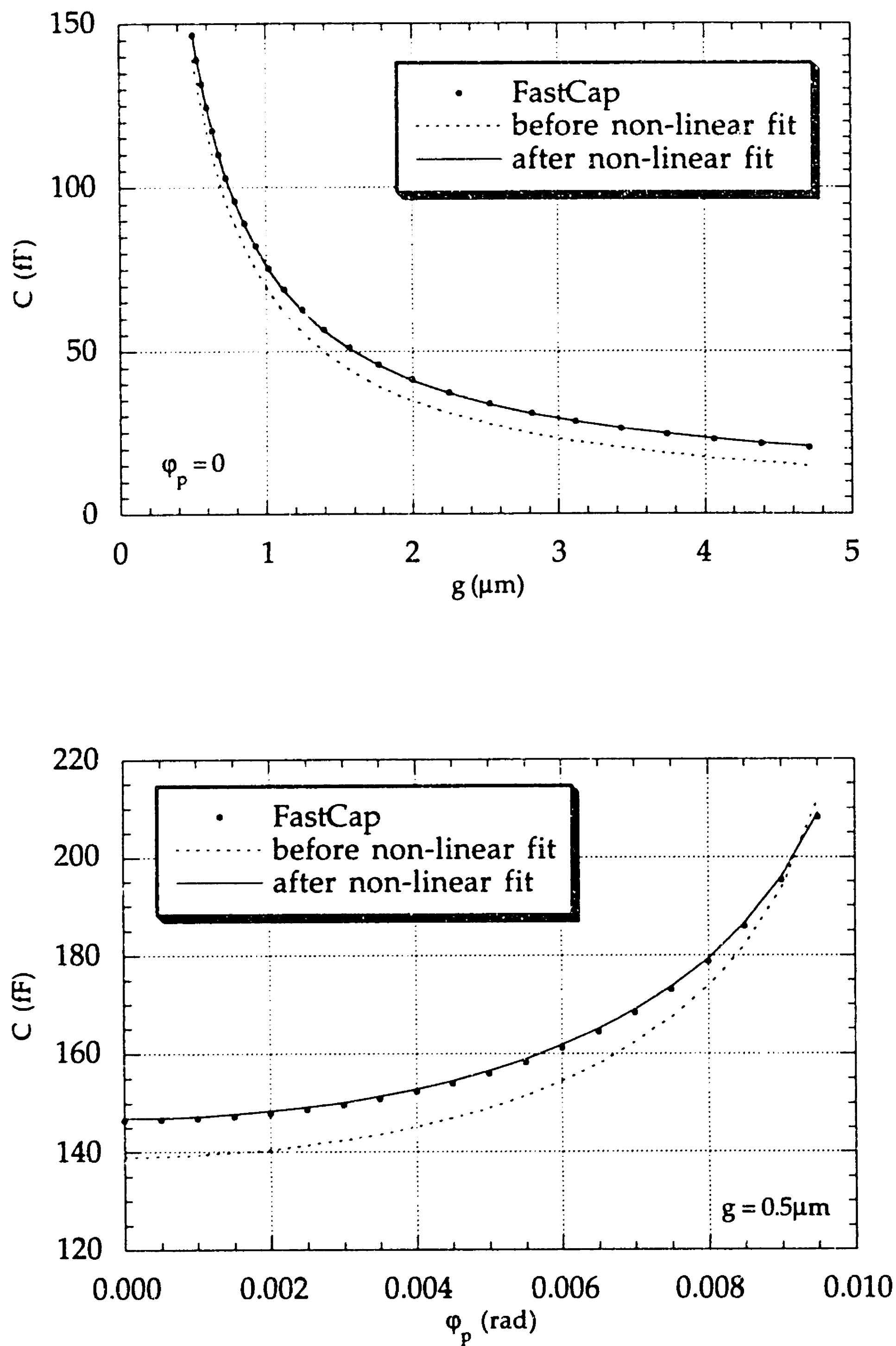
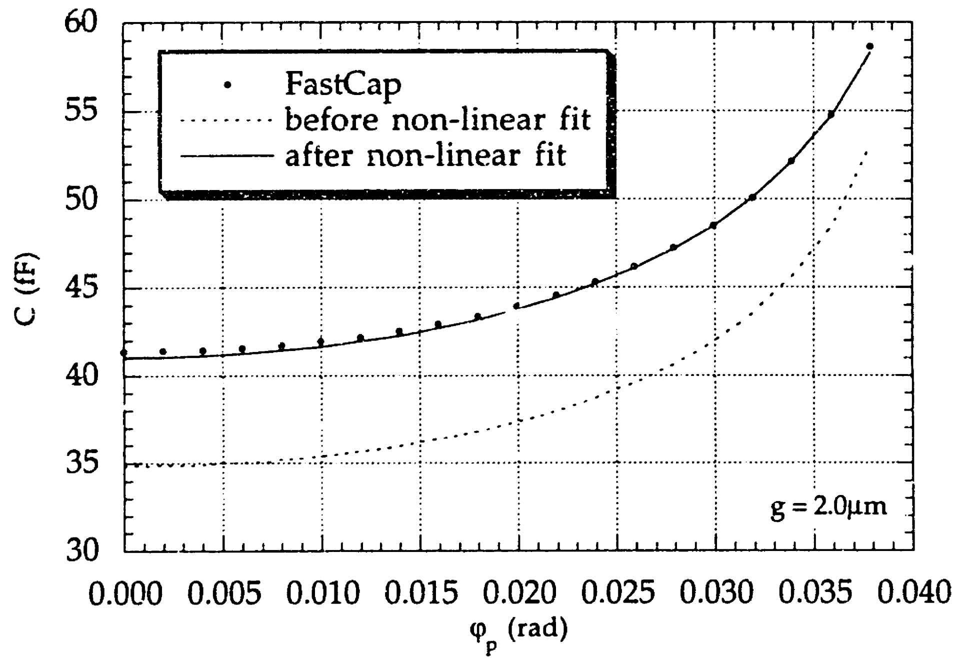


Figure 16: Comparison of Unfitted and Fitted Algebraic Models with Capacitance Data Extracted by FastCap



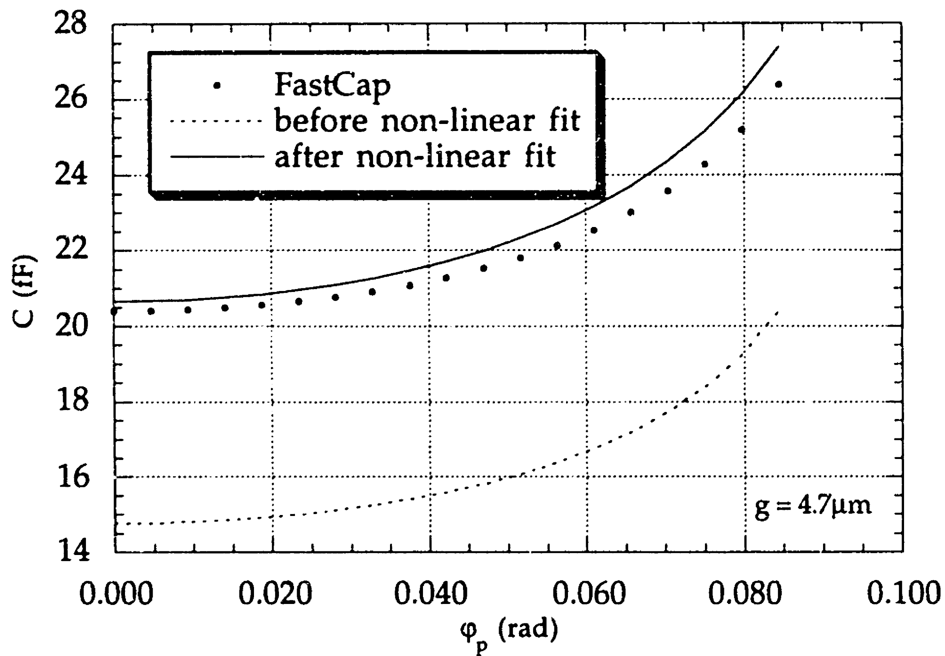


Figure 16 (cont): Comparison of Unfitted and Fitted Algebraic Models with Capacitance Data Extracted by FastCap

CHAPTER 4 Dynamics Simulation

In the preceding chapters, a numerical model of the capacitance of the suspended circular plate was developed. In this chapter, a suite of tools is developed that use this model to simulate and analyze the dynamics of suspended circular plate structures under arbitrary driving voltages. First, the theory of a modular dynamics simulator is presented. This theory is then applied to the suspended circular plate by the implementation of a simulator that depends upon modules that describe the system: the algebraic model of capacitance, a function of voltage with respect to time, and the dimensions and material parameters of the suspended plate and its beam supports. Next, a technique for determining the equilibrium state and normal modes about equilibrium of the system is presented. Finally, the dynamics simulator is used to simulate the complex dynamics of the suspended circular plate.

4.1 Modularizing the Equations of Motion

The dynamical equations of motion can be expressed as

$$\ddot{x} = f(x, \dot{x}, t) \tag{35}$$

where x is a vector that describes the displacement of the system as a function of time, and f is a vector function that returns the acceleration of the displacement as a function of position, velocity, and time. It is a misnomer to describe these as vectors, because the term implies that a common unit system exists among their components. For example, a macro-modeled system could have displacement coordinates in terms of position, angle, or bending, each of which would be defined in terms of a different dimensional unit. In actuality, these are arrays or multivalued variables, but for simplicity they are here referred to as vectors.

Solving the equations of motion on a computer is done by numerical integration. First, the second order differential equations of motion are reduced to standard state-equation form, where now both position x and velocity $\dot{x}$ appear as true state variables:

$$\frac{d}{dt}\begin{bmatrix} x \\ \dot{x} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ f(x, \dot{x}, t) \end{bmatrix} \tag{36}$$

If the acceleration function $f(x, \dot{x}, t)$ can be implemented on a computer, this equation can be integrated numerically to solve for the motion of the system.

Here, an acceleration function that contains only stored energy terms is discussed; however, the functionality for including friction and driving forces exists. The force applied by stored energy is determined by linearizing the potential energy about the state of the system. Thus, the acceleration of each state component is derived from the derivative of the potential energy function with respect to that state component. The acceleration of the state can be written as

$$\ddot{x} = \begin{bmatrix} \ddot{x}_1 \\ \dots \\ \ddot{x}_n \end{bmatrix} = -M^{-1}\begin{bmatrix} \frac{\partial}{\partial x_1} \\ \dots \\ \frac{\partial}{\partial x_n} \end{bmatrix} U(x, t) \tag{37}$$

where M is a matrix containing the inertial terms for the respective displacement coordinates, and $U(\mathbf{x},t)$ is the stored energy.

The above formulation implies that state acceleration is proportional to force by some inertial constant. This is true for the case of a force on a particle

$$F = \frac{dp}{dt} = m\frac{dv}{dt} = m\frac{d^2 x}{dt^2} \tag{38}$$

The force and state acceleration are proportional by the mass. However, torque and angular acceleration are not separated easily. For the case of a torque on an object with moment of inertia tensor $\{I\}$,

$$N = \frac{dL}{dt} = \frac{d}{dt}\{I\}\dot{\theta} \tag{39}$$

The torque N and angular velocity $\dot{\theta}$ carry associated coordinate systems that must be transformed to the coordinate system of the object. By including matrices λ_{ij} that transform the coordinate system from j to i, the torque becomes

$$N = \frac{d}{dt}\left(\lambda_{N0} I_{00} \lambda_{0\theta} \frac{d\theta}{dt}\right) = \frac{d\lambda_{N0}}{dt} I_{00} \lambda_{0\theta} \frac{d\theta}{dt} + \lambda_{N0} I_{00} \frac{d\lambda_{0\theta}}{dt} \frac{d\theta}{dt} + \lambda_{N0} I_{00} \lambda_{0\theta} \frac{d^2\theta}{dt^2} \tag{40}$$

The angular acceleration $\frac{d^2\theta}{dt^2}$ is not proportional to torque, because the transforms are functions of state. For this dynamics simulator implementation it is assumed that the tilt angles are in the small deflection limit, and torque is approximated as

$$N = I_{00} \frac{d^2\theta}{dt^2} \tag{41}$$

In the determination of forces by the differentiation of potential energy, it is not necessarily convenient to differentiate U with respect to the state variables. The chain rule is applied to alleviate this problem by separating the differentiation vector into the product of a derivative matrix and a new

differentiation vector taken with respect to a preferred coordinate basis $\tilde{x}$:

$$\begin{bmatrix} \frac{\partial}{\partial x_1} \\ \cdots \\ \frac{\partial}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial \tilde{x}_1}{\partial x_1} \cdots \frac{\partial \tilde{x}_{\tilde{n}}}{\partial x_1} \\ \cdots \ \cdots \ \cdots \\ \frac{\partial \tilde{x}_1}{\partial x_n} \cdots \frac{\partial \tilde{x}_{\tilde{n}}}{\partial x_n} \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial \tilde{x}_1} \\ \cdots \\ \frac{\partial}{\partial \tilde{x}_{\tilde{n}}} \end{bmatrix} \tag{42}$$

This is expressed more simply by introducing a shorthand notation:

$$\frac{\partial}{\partial \boldsymbol{x}} = \frac{\partial \tilde{\boldsymbol{x}}}{\partial \boldsymbol{x}} \cdot \frac{\partial}{\partial \tilde{\boldsymbol{x}}} \tag{43}$$

Thus, the acceleration function for a system of potential energy functions are expressed by

$$\ddot{\boldsymbol{x}} = -\boldsymbol{M}^{-1} \sum_i \frac{\partial \tilde{\boldsymbol{x}}_i}{\partial \boldsymbol{x}} \cdot \frac{\partial U_i}{\partial \tilde{\boldsymbol{x}}_i} \tag{44}$$

The preceding formulation describes a process for constructing a modular dynamics simulator. First, a set of parameters is chosen that fully describes the state of the system, $\boldsymbol{x}$. Second, the components of the potential energy, U_i, are defined. Third, a set of preferred coordinates, $\tilde{\boldsymbol{x}}_i$, is chosen for each potential energy function. Fourth, each potential energy function is differentiated with respect to its preferred coordinates, $\frac{\partial U_i}{\partial \tilde{\boldsymbol{x}}_i}$. Fifth, a derivative matrix of the preferred coordinates with respect to the state coordinates is implemented, $\frac{\partial \tilde{\boldsymbol{x}}_i}{\partial \boldsymbol{x}}$. Finally, the inertia matrix $\boldsymbol{M}$ is constructed and inverted. These modules combine to form a numerical model of the equations of motion, which can then be numerically integrated to yield the simulated dynamics.

4.2 Modular Equations of Motion for the Suspended Circular Plate

A dynamics simulator that solves for the motion of the macro-modeled suspended circular plate is implemented by following the process outlined above. The code developed for this simulator is presented in Appendix A.

First, the positional state of the system is defined in terms of three parameters:

$$x = \begin{bmatrix} g \\ \varphi_x \\ \varphi_y \end{bmatrix} \tag{45}$$

where g, φ_x, and φ_y are the gap and tilt angles of the suspended circular plate. Next, the potential energy of the plate is classified into two modules: an electrostatic and a mechanical energy component. These modules, in combination with the inverse of the inertia matrix, comprise the simulator.

4.2.1 Electrostatic Contribution

The potential energy stored in a linear capacitor attached to a voltage source is given by

$$U_E = \frac{1}{2}CV^2 \tag{46}$$

where C is the capacitance and V is the voltage across the capacitor. The only state dependent term in the energy equation is the capacitance; the voltage is purely time dependent. Thus, the voltage can be seen as a time dependent scaling factor to the energy, which is characterized by C.

Two preferred coordinates are inherent within the macro-model of the suspended circular plate capacitance: the gap g and the primary tilt angle φ_p. The preferred coordinates thus become

$$\tilde{x}(x) = \begin{bmatrix} g \\ \varphi_p \end{bmatrix} = \begin{bmatrix} g \\ \operatorname{atan}\sqrt{(\tan\varphi_x)^2 + (\tan\varphi_y)^2} \end{bmatrix} \tag{47}$$

and the derivative matrix becomes

$$\frac{\partial \tilde{x}}{\partial x} = \begin{bmatrix} 1 & 0 \\ 0 & \dfrac{\tan\varphi_x \sec^2\varphi_x}{\tan\varphi_p \sec^2\varphi_p} \\ 0 & \dfrac{\tan\varphi_y \sec^2\varphi_y}{\tan\varphi_p \sec^2\varphi_p} \end{bmatrix} \tag{48}$$

The electrostatic energy must be declared carefully before it is differentiated. Although $\frac{1}{2}cv^2$ is equivalent to the energy stored in the electric fields, it actually represents the co-energy [15]. The energy is written in terms of the charges on the conductors Q:

$$U_E = \frac{1}{2}\frac{Q^2}{C} \tag{49}$$

Because the electrostatic forces originate from charges and not voltage, the charge dependent formula is used for differentiation. Differentiating this with respect to the preferred coordinates yields

$$\frac{\partial U_E}{\partial \tilde{x}} = -\frac{1}{2}\frac{Q^2}{C^2}\frac{\partial C}{\partial \tilde{x}} = -\frac{1}{2}V^2\frac{\partial C}{\partial \tilde{x}} \tag{50}$$

This derivative is exactly the negative of that which would be calculated if Equation 46 were interpreted as the energy instead of the co-energy. Thus, attention to the energy versus co-energy is important, even for linear capacitors.

The numerical model developed in the previous chapters is used to supply the derivative of the capacitance. Thus, the electrostatic contribution to the force on the plate is specified completely.

4.2.2 Mechanical Contribution

Each support beam is assumed to have two modes of energy storage, bending and torsion. The spring constants for each deformation mode are determined by applying beam theory in the linear regime to the dimensions and material parameters of the beams. Beam theory is a well documented field,

having linearized the restoring forces for many beam designs under various loads. The translational spring is treated as a linearized beam in pure bending that is fixed at one end but is free and guided with a concentrated load at the other, as depicted in Figure 17. The decision to have the support beam guided by

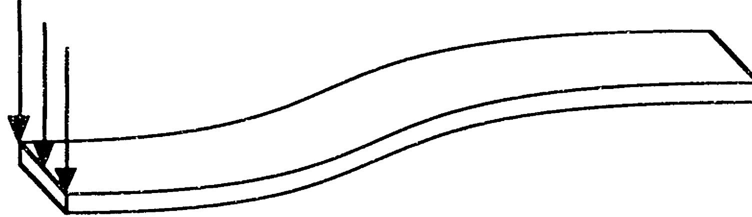


Figure 17: Translational Beam Deflection

the plate comes from the assumption that the plate is rigid, which implies that the plate will clamp the support beam. The rotational spring is treated as a linearized beam fixed at one end with concentrated torque at the other, as depicted in Figure 18. It is important to note that the rotational model does not take into account the

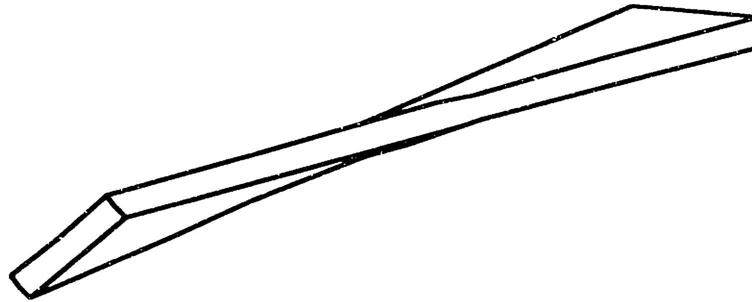


Figure 18: Rotational Beam Deformation

simultaneous deflection of the beam, and neither does the translational model take into account the twisting of the beam. Furthermore, neither model accounts for the minute stretching and shearing that occurs. To simplify this implementation, however, these issues are ignored.

The potential energy stored in the linear springs is given by

$$U = \frac{1}{2}k(\Delta z)^2 + \frac{1}{2}\kappa(\Delta\theta)^2 \tag{51}$$

where Δz is the distance by which the tip of the beam is deflected, and $\Delta\theta$ is the angle by which the tip is rotated. Clearly, Δz and $\Delta\theta$ are the preferred coordinates

of this energy function. The spring constants k and κ are defined from pre-existing models determined by beam theory [16]. For a beam of length L, width w, thickness t, Young's modulus E, and Poisson ratio ν, the spring constant k for a beam that is fixed at one end and free and guided at the other, with a concentrated load at the guided end, is given by

$$k = \frac{Ewt^3}{L^3} \tag{52}$$

The spring constant κ for a beam fixed at one end with concentrated torque at the other is given by

$$\kappa = \frac{Ewt^3}{2(1+\nu)L}\left(\frac{1}{3} - \frac{3.36}{16}\left(\frac{t}{w}\right)\left(1 - \frac{1}{12}\left(\frac{t}{w}\right)^4\right)\right) \tag{53}$$

The energy is differentiated with respect to the preferred coordinates to yield

$$\frac{\partial U}{\partial \tilde{x}} = \begin{bmatrix} \frac{\partial U}{\partial \Delta z} \\ \frac{\partial U}{\partial \Delta \theta} \end{bmatrix} = \begin{bmatrix} k\Delta z \\ \kappa \Delta \theta \end{bmatrix} \tag{54}$$

Defining Δz and $\Delta\theta$ in terms of the state coordinates can yield complicated formulas, in particular, $\Delta\theta$ for a beam whose axis is not parallel or perpendicular to the direction of principle tilt. For simplicity, only beams placed along the x or y axes are considered here. For the case of the $\pm x$ axis,

$$\tilde{x}_x(x) = \begin{bmatrix} \Delta z \\ \Delta \theta \end{bmatrix} = \begin{bmatrix} (g - g_0) \mp R\tan\varphi_y \\ \varphi_x \end{bmatrix} \tag{55}$$

and for the case of the $\pm y$ axis,

$$\tilde{x}_y(x) = \begin{bmatrix} \Delta z \\ \Delta \theta \end{bmatrix} = \begin{bmatrix} (g - g_0) \pm R\tan\varphi_x \\ \varphi_y \end{bmatrix} \tag{56}$$

where g_0 is the initial gap between the plate and the base, and R is the radius of

the plate. The derivative matrices are respectively

$$\frac{\partial \tilde{x}_x}{\partial x} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ \mp R\,(\sec\varphi_y)^2 & 0 \end{bmatrix} \tag{57}$$

and

$$\frac{\partial \tilde{x}_y}{\partial x} = \begin{bmatrix} 1 & 0 \\ \pm R\,(\sec\varphi_x)^2 & 0 \\ 0 & 1 \end{bmatrix} \tag{58}$$

Thus, the spring contribution to the force on the plate is specified completely.

4.2.3 Inertia Matrix

In order to model the inertia of the suspended circular plate system, only the mass of the rigid circular plate is considered; the mass of the beams is neglected. The circular plate is treated here as a uniform cylinder with radius R and thickness t. The mass of the plate is

$$m = \rho\pi R^2 t \tag{59}$$

where ρ is the mass density of the plate. The moment of inertia about an axis along the face of the plate through the center of the face is given by

$$I = m\left(\frac{R^2}{4} + \frac{t^2}{3}\right) \tag{60}$$

Therefore, the inertia matrix is given by

$$M = \begin{bmatrix} m & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & I \end{bmatrix} \tag{61}$$

This matrix is inverted easily and construction of the dynamics simulator is concluded.

4.3 Equilibrium and Normal Modes

It is desirable to be able to determine the equilibrium state and normal modes about equilibrium in order to characterize the behavior of macro-modeled systems. Numerical techniques are developed to determine the equilibrium and normal modes. The code that implements these techniques can be found in Appendix A.

4.3.1 Equilibrium

By definition, equilibrium occurs in a system when the accelerations for all of the state variables go to zero simultaneously. Unless the acceleration functions are readily available in an easily solvable form, finding equilibrium requires a relatively sophisticated zero searching algorithm. Of the many such algorithms available, the damped matrix-free Newton method is chosen.

Newton techniques use an initial guess and an estimated slope of the function in order to extrapolate to more exact guesses repeatedly. It follows that the initial guess passed to the Newton method determines which zero will be found. There are both stable and unstable equilibrium points in this class of electromechanical systems. The type of point the Newton method discovers depends upon the initial guess. The rest state, i.e. the undeflected state at zero voltage, is a good initial guess, because this state is closer to any stable equilibrium state than to any unstable equilibrium state. This is so because the unstable equilibrium states divide stable dynamics and pull-in. Thus, the rest state is used as the initial guess for the Newton method.

The Newton method may have difficulty converging, because the state coordinates and their corresponding accelerations have varied units and magnitudes. Newton methods applied to a function are most effective when the function output has the same order of magnitude as its input and when the individual coordinates share a common magnitude. The gap is recorded on the computer in units of meters, yielding magnitudes on the order of 10^{-6}. However, the tilt angles, recorded in radians, have magnitudes on the order of 10^{-3}.

Furthermore, the accelerations have significantly greater magnitudes, with gap acceleration on the order of 10^3 and tilt accelerations on the order of 10^8. The solution to this problematic magnitude variation is to precondition the state and acceleration function. Two diagonal matrices, P and Q, are defined to normalize the magnitudes of the state and acceleration respectively to be of order 1:

$$\hat{x} = Px \quad (62)$$

$$\hat{f}(t, x) = Qf(t, x) \quad (63)$$

A new acceleration function is defined whose inputs and outputs are of order 1:

$$g(t, \hat{x}) = Qf(t, P^{-1}\hat{x}) \quad (64)$$

This normalization enables the Newton solver to converge for $g(t, \hat{x})$ more easily. The initial guess passed to the Newton solver on $g(t, \hat{x})$ must be preconditioned by:

$$\hat{x}_0 = Px_0 \quad (65)$$

and the equilibrium state $\hat{x}_e$ that the algorithm returns must be reconditioned to its original units:

$$x_e = P^{-1}\hat{x}_e \quad (66)$$

From this theory, a preconditioned acceleration module can be written around the original acceleration function, and an equilibrium determination module can be written to use the preconditioned acceleration module with a damped Jacobian-free Newton solver to find the equilibrium state [17].

4.3.2 Normal Modes

Normal modes are calculated by first expressing the acceleration function, $\ddot{x}$, as a Taylor series expansion about some x_0:

$$\ddot{x} = f(x) = f(x_0) + \left(\frac{d}{dx}f(x)|_{x_0}\right)(x - x_0) + \ldots \quad (67)$$

At equilibrium, $\ddot{x} = 0$. Thus, by expanding about equilibrium and by neglecting

higher order terms, it is found that

$$\ddot{x} \approx \left(\frac{d}{dx}f(x)\big|_{x_e}\right)(x - x_e) \tag{68}$$

This equation is modified to match the equation for a simple harmonic oscillator:

$$\frac{d^2}{dt^2}(x - x_e) \approx \left(\frac{d}{dx}f(x)\big|_{x_e}\right)(x - x_e) \tag{69}$$

The eigenvectors v_i of the matrix $\frac{d}{dx}f(x)|_{x_e}$ determine the normal modes of the dynamical system about equilibrium, and the frequencies of those modes can be determined from the eigenvalues ζ_i by

$$\zeta_i = -\omega_i^2 \tag{70}$$

The $\frac{d}{dx}f(x)|_{x_e}$ matrix is not Hermitian because, as previously stated, the state vector components do not share a common unit system. A metric would have to be constructed and applied to the state coordinate system before it would become Hermitian. Thus, the eigenvectors are not orthogonal by dot product. In order to extract the contributions of each mode to the total state, a contravariant eigenvector basis v^i to the covariant basis v_i must be generated. By definition,

$$v^i \cdot v_j = \delta_j^i \tag{71}$$

where δ_j^i is the Kronecker delta. Thus, by placing the contravariant vectors in the rows of a matrix V^{row} and the covariant vectors in the columns of another matrix V_{col}, it follows that

$$V^{\mathrm{row}} V_{\mathrm{col}} = I \tag{72}$$

and thus the contravariant vectors are determined by

$$V^{\mathrm{row}} = V_{\mathrm{col}}^{-1} \tag{73}$$

Therefore, the projections of the state onto each mode are determined by the dot product of that state with the corresponding contravariant vector of the mode.

4.4 Suspended Circular Plate Simulation

In Appendix B, the developed simulator is used to explore the dynamics of

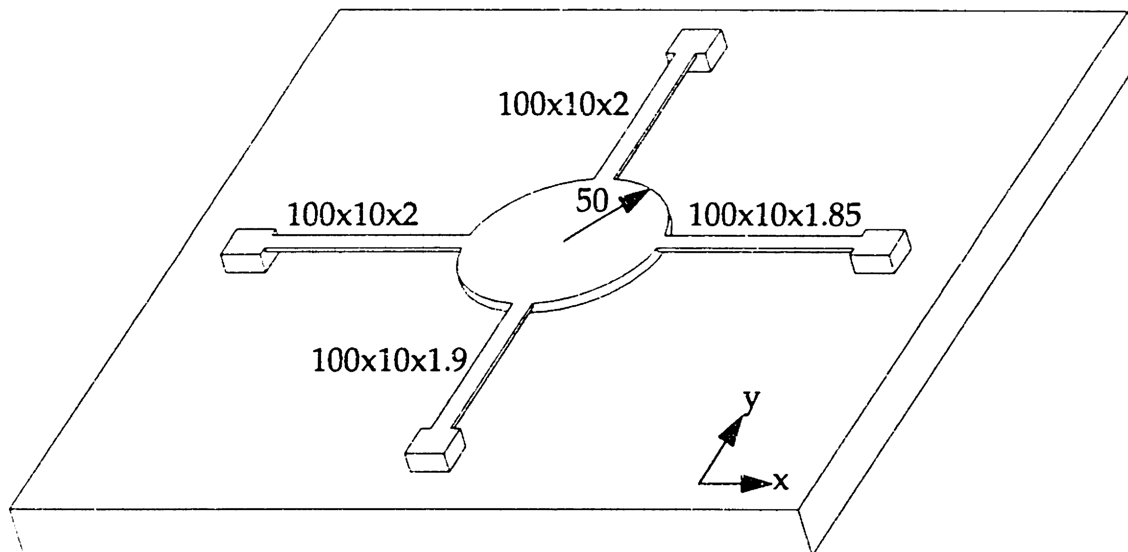


Figure 19: Simulated Suspended Plate Structure Diagram

the suspended circular plate in detail; a sample of the results is presented here. The simulated structure is depicted in Figure 19. A rigid circular plate with radius 50μm and thickness 5μm is suspended 5μm above a ground plane by four thin beams. Each beam has length 100μm and width 10μm. Two beams have thickness 2μm, one has thickness 1.85μm and the last has thickness 1.9μm. The plate starts at rest at a gap of 5μm with voltage 0. Voltage is then linearly increased to 150 volts over 10^{-4} seconds, after which voltage is held constant at 150 volts.

Figure 20 shows the motion of the plate, and Figure 21 shows this motion projected onto the calculated normal modes about equilibrium at 150 volts. The dots on each plot represent the integration steps taken, and the horizontal lines represent the calculated equilibrium value at 150 volts. The non-sinusoidal behavior in modes 2 and 3 demonstrates the energy transfer between normal modes due to large displacements about equilibrium. This is an example of the complex non-linearity of electromechanical motion.

For an extended set of dynamical simulations, see Appendix B.

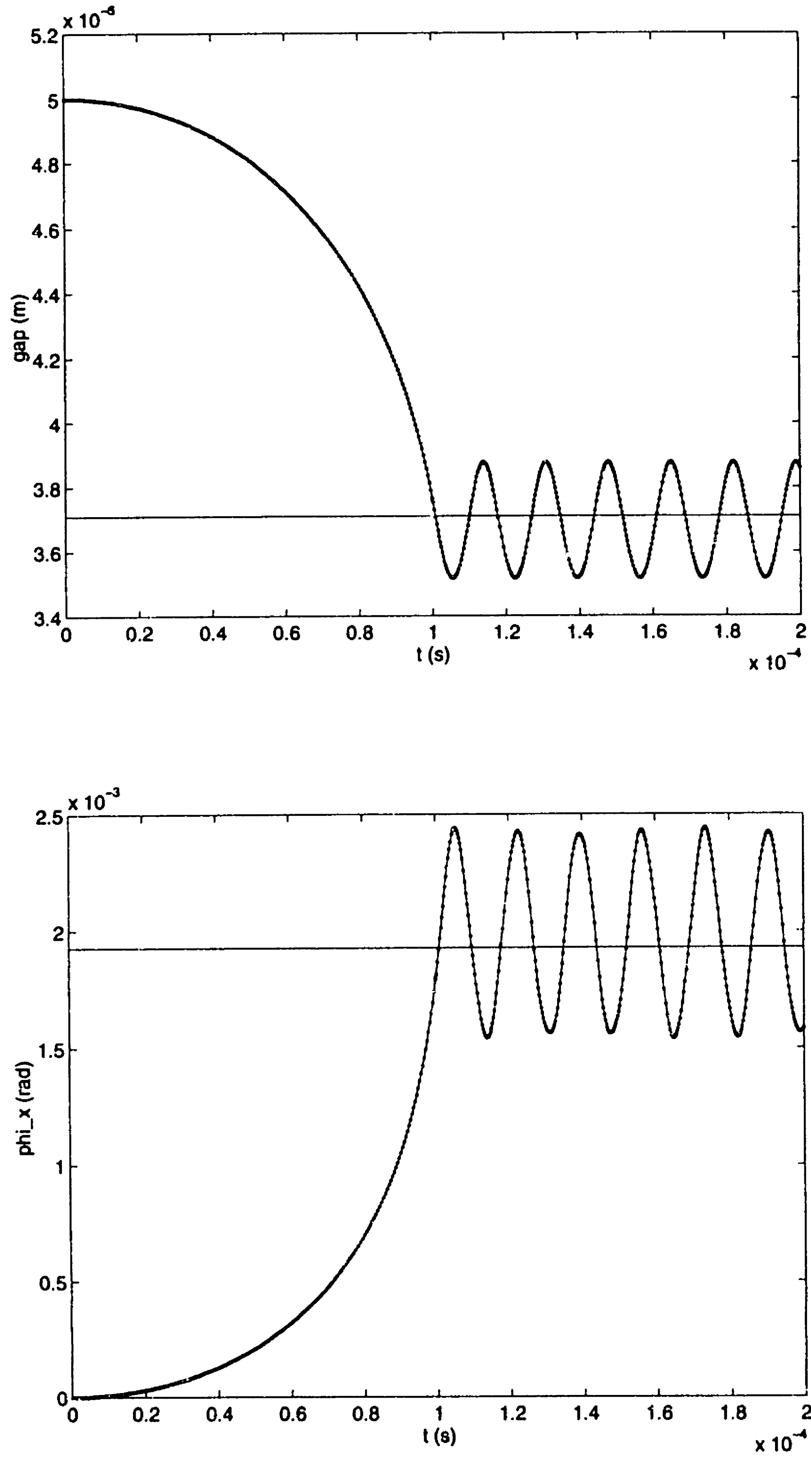


Figure 20: Sample Dynamics

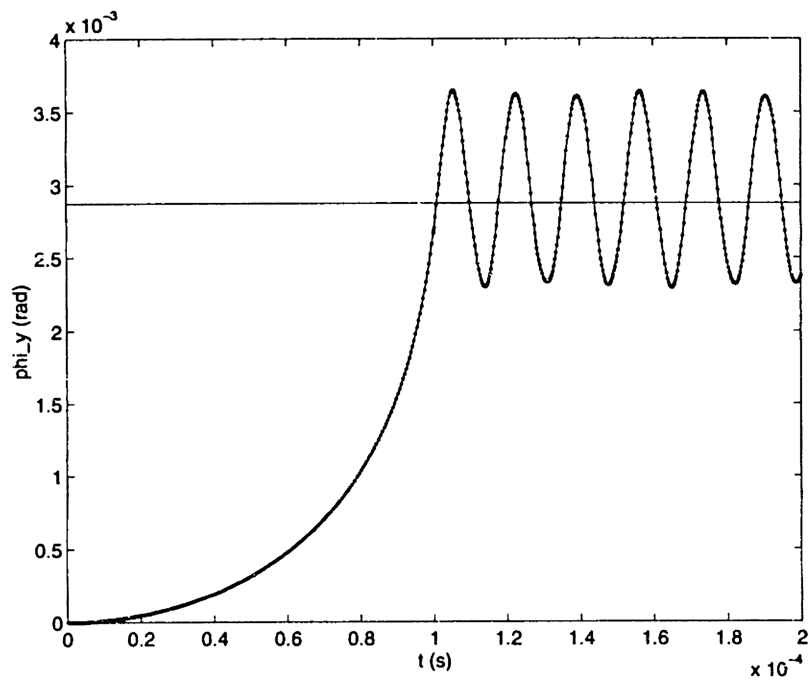
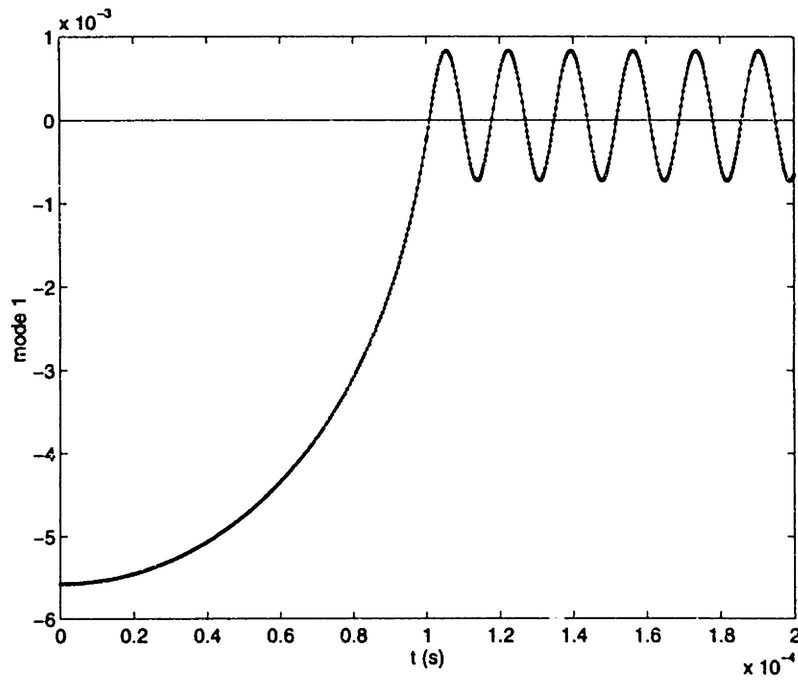


Figure 20 (cont): Sample Dynamics



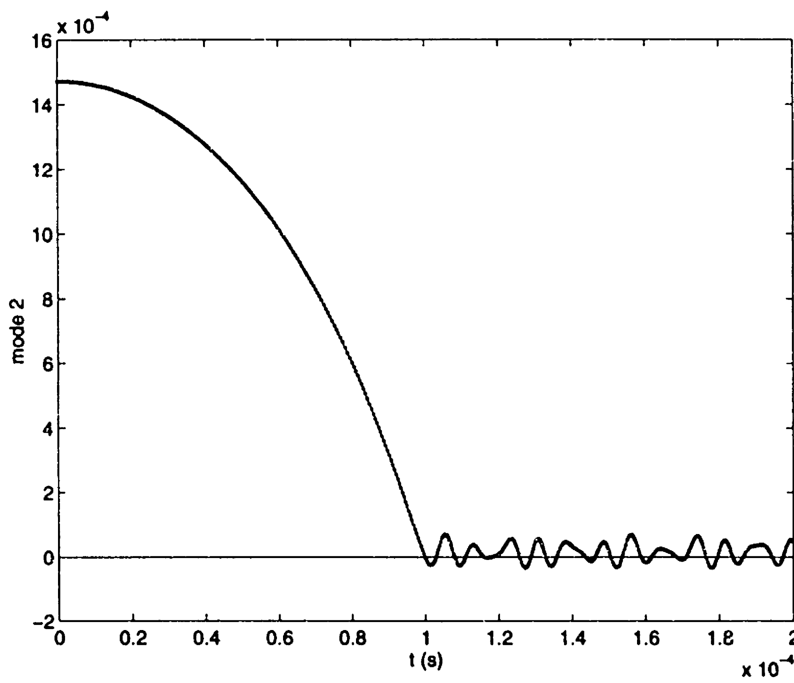


Figure 21: Sample Dynamics - Mode Projection

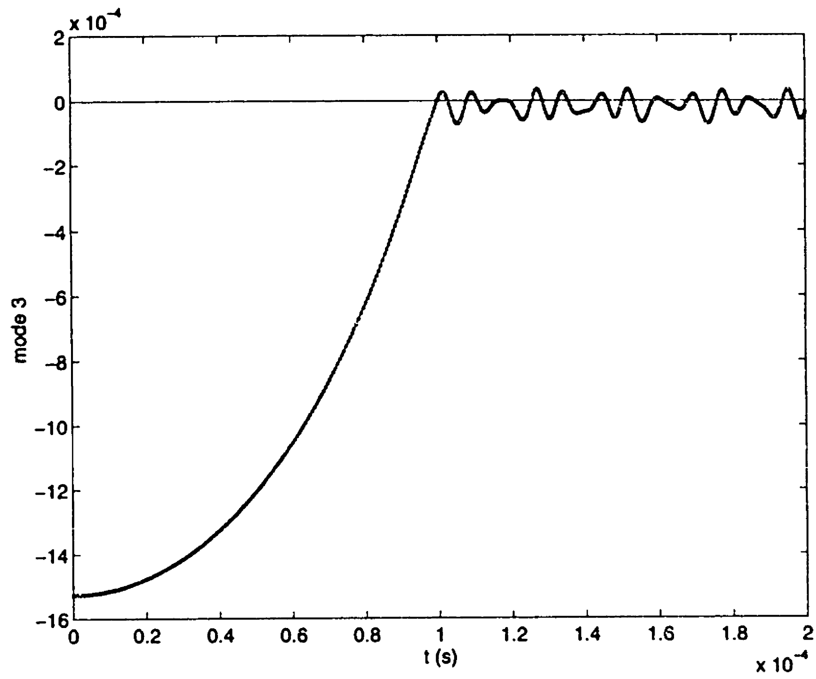


Figure 21 (cont): Sample Dynamics - Mode Projection

CHAPTER 5 Conclusion

In this research, the ability to carry a structure concept through the macro-modeling process has been demonstrated. The suspended circular plate has undergone full three-dimensional simulation via the multiple state capacitance extraction process. A physically based algebraic model has been fit to the simulation data. Finally, this model has been used to predict and characterize the dynamics of a suspended circular plate system.

Several issues regarding this macro-modeling process remain untouched. One is that the multiple state capacitance extraction process can be further optimized by incorporating MEMBase into FastCap, thereby eliminating the time taken to write out FEM files for FastCap to read. Another is that developing an approximate theoretical model of a physical system is not always possible. It might be beneficial to derive some curve fitting scheme that will construct a functional form for the system using only the obtained data without any approximate function to base it on. It might be further possible to combine the capacitance extraction tools with the numerical modeling tools, thereby permitting a numerical modeling routine to dynamically control what data is

obtained via capacitance extraction. A final issue is that the inertia of the system relative to the state coordinates is treated as being constant, although it can depend upon state. Some thought must go into deriving a general way for extracting the state dependence from the inertia of the system.

The tools developed in this paper are far from being able to construct macro-models of a given structure automatically. However, they do provide the functionality to simulate the dynamics of a suspended plate system with arbitrary plate shape. These tools also provide a framework within which specialized tools for particular structures can be constructed. In particular, the various modules which automate highly repetitive tasks reduce the total time required to construct an accurate macro-model by several orders of magnitude. Thus, a promising start has been made in the practical linking of three-dimensional numerical simulation of meshed structures to corresponding lumped-element dynamical macro-models.

APPENDIX A Code

A.1 Capacitance Extraction

A.1.1 AutoGen

This code is written in the I-DEAS program file language.

```
K : #input "Enter Gap start: " GapStart
K : #input "Enter Gap step: " GapStep
K : #input "Enter Gap stop: " GapStop
K : #input "Enter Thick start: " ThickStart
K : #input "Enter Thick step: " ThickStep
K : #input "Enter Thick stop: " ThickStop
K : #input "Enter Phi start: " PhiStart
K : #input "Enter Phi step: " PhiStep
K : #input "Enter Phi stop: " PhiStop
C : ==============================================================
C : Turning off forms...
K : /o p p 2
K :
K : fd of
K : OKAY
K : OKAY
C : --------------------------------------------------------------
C : Leave part parameters on after update...
K : /up uo ap no OKAY
C : ==============================================================
```

```
C : Loop 1 (Gap)
C : Loop 1 Init
K : #Gap = GapStart
C : -------------------------------------------------------------
C : Loop 1 Test
K : #L1TEST:
K : #if (NOT (Gap LE GapStop)) then goto L1FAIL
C : -------------------------------------------------------------
C : Loop 1 Block
C : Edit parameters...
K : /mo e
K : lab
K : Til
K : pt
K : di
K : v
C : Here is the gap (1phi, 2thick, 5gap)
K : 5
K :
K : EQ Gap
K : OKAY
K :
C : -------------------------------------------------------------
C : =============================================================
C : Loop 2 (Thick)
C : Loop 2 Init
K : #Thick = ThickStart
C : -------------------------------------------------------------
C : Loop 2 Test
K : #L2TEST:
K : #if (NOT (Thick LE ThickStop)) then goto L2FAIL
C : -------------------------------------------------------------
C : Loop 2 Block
C : Edit parameters...
K : /mo e
K : lab
K : Til
K : pt
K : di
K : v
C : Here is the thickness (1phi, 2thick, 5gap)
K : 2
K :
K : EQ Thick
K : OKAY
K :
C : -------------------------------------------------------------
C : =============================================================
C : Loop 3 (Phi)
C : Loop 3 Init
K : #Phi = PhiStart
C : -------------------------------------------------------------
C : Loop 3 Test
K : #L3TEST:
```

```
K : #if (NOT ((Phi LE PhiStop) AND (50*SIN(Phi) LT Gap))) then goto
    L3FAIL
C : -------------------------------------------------------------------
C : Loop 3 Block
C : Edit parameters...
K : /mo e
K : lab
K : Til
K : pt
K : di
K : v
C : Here is the phi (1phi, 2thick, 5gap)
K : 1
K :
K : EQ Phi
K : OKAY
K :
C : -------------------------------------------------------------------
K : #Filename = "g"+Gap+"t"+Thick+"p"+Phi+".unv"
C : Here is where you update
K : /up p
C : Here is where you save the file
K : /f exp s 1
K :
K : OKAY
K : EU 3
K :
K : FN Filename
K : Y
K : OKAY
C : -------------------------------------------------------------------
C : Loop 3 Incr
K : #Phi = Phi+PhiStep
K : #goto L3TEST
K : #L3FAIL:
C : ===================================================================
C : Loop 2 Incr
K : #Thick = Thick+ThickStep
K : #goto L2TEST
K : #L2FAIL:
C : ===================================================================
C : Loop 1 Incr
K : #Gap = Gap+GapStep
K : #goto L1TEST
K : #L1FAIL:
C : ===================================================================
C : Turning forms back on...
K : /o p p 2
K :
K : fd on
K : OKAY
K : OKAY
E : **** END OF SESSION ****
```

A.1.2 AZ

This code is written in the UNIX tcsh shell script language.

```
#!/usr/local/bin/tcsh
unalias rm cp mv

set TARGETDIR = ~/tilted_plate
source ~/bin/HP_rsh
foreach i ( $* )
  echo "Analyzing $i"
  set SOURCEDIR = $i:h
  if( "$SOURCEDIR" == "$i" ) then
    set SOURCEDIR = .
  endif
  set ROOT = $i:t:r
  if( !( -r $TARGETDIR/archive/$ROOT.tgz )) then
    rm -f $TARGETDIR/$ROOT.unv
    rm -f $TARGETDIR/$ROOT.pat
    rm -f $TARGETDIR/$ROOT.fc
    rm -f $TARGETDIR/$ROOT.cap
    cp -f $SOURCEDIR/$ROOT.unv $TARGETDIR
    echo "Unv2Pnf"
    RSH gatekeeper "unalias cp rm mv; cd /usr/local/scratch/
   ldgabbay; \
      cp $TARGETDIR/$ROOT.unv .; \
      $TARGETDIR/unv2pnf.sun $ROOT.unv $ROOT.pat; \
      rm $ROOT.unv; \
      mv -f $ROOT.pat $TARGETDIR" >& /dev/null
    echo "FastCap"
   RSH memcad3 "cd $TARGETDIR; fastcap.alpha $ROOT.pat > $ROOT.fc"
   >& /dev/null
    echo "ReCap"
    RSH blofeld "cd $TARGETDIR; (recap.hp -e -15 -r 1 -F 1 0 -i
   $ROOT.fc -o -) | tee $ROOT.cap cap/$ROOT.cap"
    echo "Spawning Archiver..."
    RSH madonna "cd $TARGETDIR; tar cvf - $ROOT.unv $ROOT.pat
   $ROOT.fc $ROOT.cap | gzip -9 > archive/$ROOT.tgz; rm -f
   $ROOT.*" >& /dev/null &
  endif
  echo "Done! ($i)"
end
```

A.1.3 ReCap

This code is written in C++.

recap.C

```
#include <stdio.h>
#include <string.h>
```

```
#include <fstream.h>

#include "Matrix.H"
#include "FCinfo.H"

FCinfo fci;
Matrix<double> C(0);
Matrix<int> I_floating(0), I_fixed(0);
int N_conductors, N_floating, N_fixed, ref, preferredExponent;

int isExponent=0;
int isInputSet=0;
int isOutputSet=0;
int isReferenceSet=0;
int isFloatSet=0;
int isFixedSet=0;

char infile[35];
char outfile[35];
istream *is;
ostream *os;

void processArgs(int argc, char *argv[])
{
  int current_arg = 1;

  while(current_arg<argc) {
    if(argv[current_arg][0] == '-')
      switch(argv[current_arg++][1]) {
          case 'e':
            isExponent=1;
            preferredExponent = atoi(argv[current_arg]);
            current_arg++;
            break;
          case 'r':
            isReferenceSet=1;
            ref = atoi(argv[current_arg]);
            current_arg++;
            break;
          case 'f': { // fixed
            isFixedSet=1;
            isFloatSet=0;
            I_floating.zero();
            N_fixed = atoi(argv[current_arg++]);
            int i;
            I_fixed.resize(1,N_fixed);
            for(i=0;i<N_fixed;i++)
              I_fixed(0,i) = atoi(argv[current_arg++]);
            break;
          }
          case 'F': { // float
            isFloatSet=1;
            isFixedSet=0;
            I_fixed.zero();
```

```
            N_floating = atoi(argv[current_arg++]);
            int i;
            I_floating.resize(1,N_floating);
            for(i=0;i<N_floating;i++)
              I_floating(0,i) = atoi(argv[current_arg++]);
            break;
          }
          case 'i':
            isInputSet=1;
            strcpy(infile,argv[current_arg]);
            current_arg++;
            break;
          case 'o':
            isOutputSet=1;
            strcpy(outfile,argv[current_arg]);
            current_arg++;
            break;
      }
  }
}

inline int isPipe(char *str)
{
  return ((str[0]=='-')&&(str[1]=='\0'));
}

void secureInput()
{
  if(!isInputSet) {
    printf("Enter fastcap output filename: ");
    gets(infile);
  }
  if(isPipe(infile))
    is = new istream(cin);
  else
    is = new ifstream(infile);
}


void secureFCinfo()
{
  (*is) >> fci;
  N_conductors = fci.getNumberOfConductors();
}

void securePreferredExponent()
{
  if(!isExponent)
    preferredExponent = fci.getUnitExponent();
}

void secureReference()
{
  if(!isReferenceSet) {
```

```
    char refs[2];
    printf("Reference conductor? ");
    gets(refs);
    ref = atoi(refs);
  }
}

void secureFloatFixed()
{
#ifdef DEBUG
  cerr << "[ secureFloatFixed()" << endl;
#endif
  int i, j, k;
  if(isFixedSet) {
    N_floating = N_conductors - N_fixed;
    I_floating.resize(1,N_floating);

    for(i=0,k=0;(i<N_conductors)&&(k<N_floating);i++) {
      for(j=0;j<N_fixed;j++)
        if(I_fixed(0,j)==i)
          break;
      if(j==N_fixed)
        I_floating(0,k++) = i;
    }
  } else if(isFloatSet) {
    N_fixed = N_conductors - N_floating;
    I_fixed.resize(1,N_fixed);

    for(i=0,k=0;(i<N_conductors)&&(k<N_fixed);i++) {
      for(j=0;j<N_floating;j++)
        if(I_floating(0,j)==i)
          break;
      if(j==N_floating)
        I_fixed(0,k++) = i;
    }
  } else {
    N_floating = 1;
    I_floating.resize(1,N_conductors);
    I_floating(0,0) = 0;

    N_fixed = 1;
    I_fixed.resize(1,N_conductors);
    I_fixed(0,0) = ref;

    char flofixs[1];
    for (i=1;(i==ref?++i:i)<N_conductors;i++) {
      do {
        printf("Is conductor %d's potential floating or controlled
   (f,c)? ",i);
         gets(flofixs);
      } while((flofixs[0]!='c')&&(flofixs[0]!='f'));
      if (flofixs[0] == 'f')
        I_floating(0,N_floating++) = i;
          /* "controlled" == "fixed" */
```

```
      if (flofixs[0] == 'c')
        I_fixed(0,N_fixed++) = i;
    }
  }
#ifdef DEBUG
  cerr << "] secureFloatFixed()" << endl;
#endif
}

void secureOutput()
{
  if(!isOutputSet) {
    printf("Enter output filename: ");
    gets(outfile);
  }
  if(isPipe(outfile))
    os = new ostream(cout);
  else
    os = new ofstream(outfile);
}

void floatConductor(int f, Matrix<double>& C)
{
#ifdef DEBUG
  cerr << "[ floatConductor()" << endl;
#endif
  int i, j;

  for(i=0;(i==f?++i:i)<C.getM();i++)
    for(j=0;(j==f?++j:j)<C.getN();j++)
      C(i,j) -= C(i,f)*C(f,j)/C(f,f);
  for(i=0;i<C.getM();i++)
    C(i,f) = C(f,i) = 0;
#ifdef DEBUG
  cerr << C << "] floatConductor()" << endl;
#endif
}

void processData()
{
#ifdef DEBUG
  cerr << "[ processData()" << endl;
#endif
  int i, j, k;

  C = fci.getCapacitanceMatrix();
  for(i=0;i<N_floating;i++)
    floatConductor(I_floating(0,i),C);

  int N_out;
  N_out = N_fixed;
  for(i=0;i<N_fixed;i++)
    if(I_fixed(0,i)==ref)
      N_out--;
```

```
  Matrix<double> S(N_conductors,N_out);
  for(i=0,j=0;(i<N_fixed)&&(j<N_out);i++)
    if(I_fixed(0,i)!=ref)
      S(I_fixed(0,i),j++) = 1;

  C = S.transpose()*C*S;

#ifdef DEBUG
  cerr << C << "] processData()" << endl;
#endif
}

double pow10(int i)
{
  if(i) {
    if(i<0)
      return 1/pow10(-i);
    else
      return 10.0*pow10(i-1);
  } else {
    return 1;
  }
}

void exportData()
{
  int i = fci.getUnitExponent() - preferredExponent;
  double a = pow10(i);
  (*os) << a*C << "* 10^" << preferredExponent << " farads" <<
   endl;
}

void initialize(int argc, char *argv[]) {
  processArgs(argc, argv);
  secureInput();
  secureFCinfo();
  securePreferredExponent();
  secureReference();
  secureFloatFixed();
  secureOutput();
}

int main(int argc, char *argv[]) {
  initialize(argc, argv);
  processData();
  exportData();
  delete is;
  delete os;
  return 0;
}
```

FCinfo.H

```
#include <iostream.h>
#include <math.h>

class FCinfo
{
  int number_of_conductors;
  int unit_exponent;
  Matrix<double> capacitance_matrix;
public:
  FCinfo() : number_of_conductors(0), unit_exponent(0),
   capacitance_matrix(1) {}
  void setNumberOfConductors(int nc)
  {
    number_of_conductors = nc+1;
    capacitance_matrix.resize(number_of_conductors);
  }
  int getNumberOfConductors()
  {
    return number_of_conductors;
  }

  void setUnitExponent(int ue)
  {
    unit_exponent = ue;
  }
  int getUnitExponent()
  {
    return unit_exponent;
  }

  Matrix<double>& getCapacitanceMatrix() { return
   capacitance_matrix; }

  friend istream& operator>>(istream&, FCinfo&);
  friend ostream& operator<<(ostream&, FCinfo&);
};

int getUnits(const char *unitString)
{
  switch(unitString[0]) {
      case 'a': // atto
        return -18;
      case 'f': // farad, femto
        switch(unitString[1]) {
             case 'a': // farad
               return 0;
             case 'e': // femto
               return -15;
        }
      case 'm': // micro, milli
        switch(unitString[2]) {
```

```
            case 'c': // micro
              return -6;
            case 'l': // milli
              return -3;
        }
      case 'n': // nano
        return -9;
      case 'p': // pico
        return -12;
      default: // i don't know
        return 0;
  }
}

#define FCLINE (30)
#define FCEOL {is.get(buf,FCLINE,'\n'); is.get(buf[0]);}
#define FCSKIP {is.get(buf,FCLINE,' ');}
istream& operator>>(istream& is, FCinfo& fci)
{
  char buf[FCLINE];
  int foo;

  FCSKIP;
  is >> foo;
  fci.setNumberOfConductors(foo);
  FCEOL;


  fci.getCapacitanceMatrix().zero();
  int i, j;
  double maxfoo = 0.0;
  for (i=1;i<fci.getNumberOfConductors();i++) {
    FCSKIP;
    for (j=1;j<fci.getNumberOfConductors();j++) {
      is >> (fci.getCapacitanceMatrix())(i,j);
      if(fabs(fci.getCapacitanceMatrix()(i,j))>maxfoo)
        maxfoo = fabs(fci.getCapacitanceMatrix()(i,j));
      (fci.getCapacitanceMatrix())(0,j)
        -= (fci.getCapacitanceMatrix())(i,j);
      (fci.getCapacitanceMatrix())(i,0)
        -= (fci.getCapacitanceMatrix())(i,j);
    }
    (fci.getCapacitanceMatrix())(0,0)
      -= (fci.getCapacitanceMatrix())(i,0);
    FCEOL;
  }

  int newfoo = (int) floor(log10(maxfoo));
  fci.setUnitExponent(-6+newfoo);
  fci.getCapacitanceMatrix() = fci.getCapacitanceMatrix()*pow(10,-
   newfoo);

#ifdef DEBUG
  cerr << fci;
```

```
#endif
  return is;
}
#undef FCLINE
#undef FCEOL

ostream& operator<<(ostream& os, FCinfo& fci)
{
  os << "Conductors: " << fci.number_of_conductors << endl
     << "UnitExponent: " << fci.unit_exponent << endl
     << "Capacitance Matrix: " << endl
     << fci.capacitance_matrix;
  return os;
}
```

Matrix.H

```
#include <iostream.h>
#include <stdlib.h>

template <class T>
class Matrix
{
  int m, n;
  T** matrix;
  T dummy_element;
  int isExist() const { return (m&&n); }

  void initialize(int, int);
  void copy(const Matrix<T>&);
  void destroy();
  void swap(int, int);
public:
  Matrix(int _m, int _n) { initialize(_m,_n); }
  Matrix(int _m) { initialize(_m,_m); }
  Matrix(Matrix& A) : m(0), n(0) { copy(A); }
  ~Matrix() { destroy(); }
  T& operator()(int, int) const;
  void zero();
  void resize(int, int);
  void resize(int _m) { resize(_m,_m); }
  friend ostream& operator<<(ostream&, const Matrix<T>&);
  Matrix<T>& operator=(const Matrix<T>& x) { copy(x); return
   *this; }
  friend Matrix<T> operator+(const Matrix<T>&, const Matrix<T>&);
  friend Matrix<T> operator-(const Matrix<T>&, const Matrix<T>&);
  friend Matrix<T> operator*(const Matrix<T>&, const Matrix<T>&);
  friend Matrix<T> operator*(T, const Matrix<T>&);
  friend Matrix<T> operator*(const Matrix<T>& A, T c) { return c*A;
   }
  Matrix<T> inverse() const;
  Matrix<T> transpose() const;
  int getM() const { return m; }
```

```
  int getN() const { return n; }
};

template <class T>
inline T& Matrix<T>::operator()(int i, int j)
{
  if((i<m)&&(j<n))
    return matrix[i][j];
  else
    return dummy_element;
}

template <class T>
inline void Matrix<T>::initialize(int _m, int _n)
{
#ifdef DEBUG
  cerr << "[ Matrix::initialize(" << _m << "," << _n << ")" <<
   endl;
#endif
  m = _m;
  n = _n;
  if(isExist()) {
    int i;
    matrix = new (T*)[m];
    for(i=0;i<m;i++)
      matrix[i] = new T[n];
    zero();
  }

#ifdef DEBUG
  cerr << "] Matrix::initialize(" << m << "," << n << ")" <<  endl;
#endif
}

template <class T>
inline void Matrix<T>::copy(const Matrix<T>& x)
{
#ifdef DEBUG
  cerr << "[ Matrix::copy" << endl;
#endif
  resize(x.m,x.n);
  if(isExist()) {
    int i,j;
    for(i=0;i<m;i++)
      for(j=0;j<n;j++)
        (*this)(i,j) = x(i,j);
  }

#ifdef DEBUG
  cerr << "] Matrix::copy" << endl;
#endif
}

template <class T>
```

```
inline void Matrix<T>::destroy()
{
#ifdef DEBUG
  cerr << "[ Matrix::destroy("  << m << "," << n << ")" << endl;
#endif
  if(isExist()) {
    int i;
    for(i=0;i<m;i++)
      delete[] matrix[i];
    delete[] matrix;
  }
  m = n = 0;
#ifdef DEBUG
  cerr << "] Matrix::destroy("  << m << "," << n << ")" <<  endl;
#endif
}

template <class T>
inline void Matrix<T>::resize(int _m, int _n)
{
  if((m!=_m)||(n!=_n)) {
    destroy();
    initialize(_m,_n);
  }
  zero();
}

template <class T>
inline void Matrix<T>::swap(int i, int j)
{
  if(isExist()) {
    T *temp = matrix[i];
    matrix[i] = matrix[j];
    matrix[j] = temp;
  }
}

template <class T>
void Matrix<T>::zero()
{
  if(isExist()) {
    int i, j;
    for(i=0;i<m;i++)
      for(j=0;j<n;j++)
        (*this)(i,j) = 0;
  }
}

template <class T>
ostream& operator<<(ostream& os, const Matrix<T>& x)
{
#ifdef DEBUG
  cerr << "[ operator<<(ostream&,Matrix<T>&)" << endl;
#endif
```

```
  if(x.isExist()) {
    int i, j;
    os.setf(ios::fixed,ios::floatfield);
    os.precision(10);
    for(i=0;i<x.m;i++) {
      for(j=0;j<x.n;j++)
        os << x(i,j) << " ";
      os << endl;
    }
  }
#ifdef DEBUG
  cerr << "] operator<<(ostream&,Matrix<T>&)" << endl;
  cerr.flush();
#endif
  return os;
}

template <class T>
Matrix<T> operator+(const Matrix<T>& a, const Matrix<T>& b)
{
  if((a.m!=b.m)||(a.n!=b.n)) {
    cerr << "Matrices cannot be added\n";
    exit(1);
  }

  Matrix<T> new_matrix(a.m,a.n);

  if(new_matrix.isExist()) {
    int i, j;
    for(i=0;i<a.m;i++)
      for(j=0;j<a.n;j++)
        new_matrix(i,j) = a(i,j) + b(i,j);
  }

  return Matrix<T>(new_matrix);
}

template <class T>
Matrix<T> operator-(const Matrix<T>& a, const Matrix<T>& b)
{
  if((a.m!=b.m)||(a.n!=b.n)) {
    cerr << "Matrices cannot be subtracted\n";
    exit(1);
  }

  Matrix<T> new_matrix(a.m,a.n);

  if(new_matrix.isExist()) {
    int i, j;
    for(i=0;i<a.m;i++)
      for(j=0;j<a.n;j++)
        new_matrix(i,j) = a(i,j) - b(i,j);
  }
```

```
  return Matrix<T>(new_matrix);
}

template <class T>
Matrix<T> operator*(const Matrix<T>& a, const Matrix<T>& b)
{
#ifdef DEBUG
  cerr << "[ Matrix::operator*(M&,M&)" << endl << a << b;
#endif
  if(a.n!=b.m) {
    cerr << "Matrices cannot be multiplied\n";
    exit(1);
  }

  Matrix<T> new_matrix(a.m,b.n);
  if(new_matrix.isExist()) {
    int i, j, k;
    for(i=0;i<a.m;i++)
      for(j=0;j<b.n;j++)
        for(k=0;k<a.n;k++)
          new_matrix(i,j) += a(i,k)*b(k,j);
  }

#ifdef DEBUG
  cerr << new_matrix << "] Matrix::operator*(M&,M&)" << endl;
#endif
  return Matrix<T>(new_matrix);
}

template <class T>
Matrix<T> operator*(T a, const Matrix<T>& b)
{
  Matrix<T> new_matrix(b.m,b.n);

  if(new_matrix.isExist()) {
    int i, j;
    for(i=0;i<b.m;i++)
      for(j=0;j<b.n;j++)
        new_matrix(i,j) = a*b(i,j);
  }

  return Matrix<T>(new_matrix);
}

template <class T>
Matrix<T> Matrix<T>::inverse()
{
  if(m!=n) {
    cerr << "Matrix cannot be inverted\n";
    exit(1);
  }
  if(!isExist()) {
#ifdef DEBUG
    cerr << "Matrix 0x0 faking inversion\n";
```

```
#endif
    return Matrix<T>(0);
  }

  int i, j, k;

  Matrix<T> inv(m);
  for(i=0;i<n;i++) {
    for(j=0;j<n;j++)
      inv(i,j) = 0;
    inv(i,i) = 1;
  }

  Matrix<T> lu(m);
  lu = *this;

  for(i=0;i<n-1;i++) {
    for(j=i;j<n;j++)
      if(lu(j,i)>1e-15)
        break;
    if(j>i) {
      lu.swap(i,j);
      inv.swap(i,j);
    }

    for(j=i+1;j<n;j++) {
      lu(j,i) /= lu(i,i);
      for(k=i+1;k<n;k++)
        lu(j,k) -= lu(j,i)*lu(i,k);
    }
  }

  for(k=0;k<n;k++) {
    for(i=0;i<n;i++)
      for(j=0;j<i;j++)
        inv(i,k) -= lu(i,j)*inv(j,k);
    for(i=n-1;i>=0;i--) {
      for(j=i+1;j<n;j++)
        inv(i,k) -= lu(i,j)*inv(j,k);
      inv(i,k) /= lu(i,i);
    }
  }

  return Matrix<T>(inv);
}

template <class T>
Matrix<T> Matrix<T>::transpose()
{
  Matrix<T> new_matrix(n,m);

  if(isExist()) {
    int i, j;
    for(i=0;i<m;i++)
```

```
      for(j=0;j<n;j++)
        new_matrix(j,i) = (*this)(i,j);
  }

  return Matrix<T>(new_matrix);
}
```

A.1.4 Churn

This code is written in C++.

churn.c

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/times.h>
#include "global_defs.h"
#include "PNF.h"
#include "UpdateNodes.h"
#include "transform.h"
#include "transforms.h"
#include "PNF_node_list.h"
#include "elementTempList.h"
#include "C_funcs.h"

#include "churn_fns.h"

struct tms timestuff;
long timecount = 0;
#define SUMTMS(t)
   ((long)(t.tms_utime+t.tms_stime+t.tms_cutime+t.tms_cstime))
#define CLK_TCK (sysconf(_SC_CLK_TCK))
#define STARTTIME {times(&timestuff); timecount -=
   SUMTMS(timestuff);}
#define STOPTIME {times(&timestuff); timecount +=
   SUMTMS(timestuff);}
#define TOTALTIME (((double)timecount)/CLK_TCK)
#define CURRENTTIME
   (times(&timestuff),((double)(timecount+SUMTMS(timestuff)))/
   CLK_TCK)
#define PRINTTIME(t) {printf("\t\t\ttime: %8.2f\n",(t));}


int main(int argc, char *argv[])
#define OUTPUT_FN "output.pat"
{
  STARTTIME;
  PRINTTIME(CURRENTTIME);

  PATRAN_neutral_file *structure;
```

```
if(!(structure = assembleNewPNF(argc-1,argv+1))) {
  showUsage(argv[0]);
  exit(1);
}

STOPTIME;
Name *plateName = chooseName(structure);
STARTTIME;

PRINTTIME(CURRENTTIME);
printf("copying old nodes starting\n");
PNF_Node_List *oldNodes = structure->getlistOfNodes()->copy();
printf("copying old nodes done\n");
PRINTTIME(CURRENTTIME);

structure->setFileName("output.pat");

FILE *outfile = fopen("testdata","a");
{
  int i;
  fprintf(outfile,"#files: ");
  for(i=1;i<argc;i++)
    fprintf(outfile," %s",argv[i]);
  fprintf(outfile,"\n");
  fflush(outfile);
}

double gap;
int gCount;
double phip;
int phiCount;
double phir;
double cap;
char capstring[80];

FILE *capfile;
PRINTTIME(CURRENTTIME);
printf("beginning to churn\n");
simulateExtractFaces1(structure);
phir=0;
for(gap=0.5;gap<=5;cap/=25*gap,gap+=0.33/sqrt(1+cap*cap))
  for(phiCount=0,phip=0;phiCount<10;phip=atan(gap/
 50)*(++phiCount)/10)
    {
      double innerlooptime = CURRENTTIME;

      structure->getlistOfNodes()->updateCoordsFrom(oldNodes);
      plateTransform(plateName,gap,phip,phir);
      simulateExtractFaces2(structure);
      structure->writeSurfaces();

      unlink("fastcap.cap");
      system("memcap output.pat");
      unlink("output.pat.qs");
```

```
        unlink("output.cap");
        system("rememcap.hp -e -15 -r 1 -F 1 0 -i fastcap.cap -o -
  | tee output.cap");
        capfile = fopen("output.cap","r");
        fscanf(capfile,"%s",capstring);
        fclose(capfile);
        if(phip==0)
          cap = atof(capstring);

  fprintf(outfile,"%13.10f\t%13.10f\t%13.10f\t%s\n",gap,phip,phi
  r,capstring);
        fflush(outfile);

        PRINTTIME(CURRENTTIME-innerlooptime);
        PRINTTIME(CURRENTTIME);
      }

  unlink("assembled.pat");
  fclose(outfile);
  return 0;
}
```

churn_fns.h

```
PATRAN_neutral_file *assembleNewPNF(int numFileNames, char
   *FileNames[]);

void showUsage(char *executable);

Name *chooseName(PATRAN_neutral_file *structure);

void setM_plate(double mat[4][4], double gap, double phip, double
   phir);

void plateTransform(Name *nameToTransform, double gap, double
   phip, double phir);

void simulateExtractFaces1(PATRAN_neutral_file *structure);

void simulateExtractFaces2(PATRAN_neutral_file *structure);
```

churn_fns.c

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/times.h>
#include "global_defs.h"
#include "PNF.h"
#include "UpdateNodes.h"
#include "transform.h"
#include "transforms.h"
#include "PNF_node_list.h"
```

```
#include "elementTempList.h"
#include "C_funcs.h"

PATRAN_neutral_file *assembleNewPNF(int numFileNames, char
   *FileNames[])
{
  if(!numFileNames)
    return NULL;

  PATRAN_neutral_file *mainPNF;

  mainPNF = new PATRAN_neutral_file;
  mainPNF->setFileName(FileNames[0]);
  mainPNF->readFile();

  if(numFileNames > 1) {
    PATRAN_neutral_file **PNFs;
    PNFs = (PATRAN_neutral_file **) malloc((numFileNames-
   1)*sizeof(PATRAN_neutral_file *));

    int count;
    for(count=0;count<numFileNames-1;count++) {
      PNFs[count] = new PATRAN_neutral_file;
      PNFs[count]->setFileName(FileNames[count+1]);
      PNFs[count]->readFile();
    }

    mainPNF->addFiles(PNFs,numFileNames-1,RENUMBER);

    for(count=0;count<numFileNames-1;count++)
      delete PNFs[count];
    free(PNFs);
  }

  mainPNF->setFileName("assembled.pat");
  mainPNF->writePNF(0);
  delete mainPNF;
  mainPNF = new PATRAN_neutral_file;
  mainPNF->setFileName("assembled.pat");
  mainPNF->readFile();

  return mainPNF;
}

void showUsage(char *executable)
{
  printf("syntax:\t%s <fem-file> [<fem-file>]*\n",executable);
}

Name *chooseName(PATRAN_neutral_file *structure)
{
  Name *aName;

  int nameCount;
```

```
  for(aName = (Name *) structure->getlistOfNames()-
   >setCurrentToBeginning(), nameCount=1;
      aName;
      aName = (Name *) structure->getlistOfNames()->Next(),
   nameCount++)
    printf("\t%2d.  \"%s\"\n",nameCount,aName->getNameString());

  int whichCount;
  whichCount=0;
  while((whichCount<1)||(whichCount>=nameCount)) {
    printf("\nSelect plate name: ");
    scanf("%d",&whichCount);
  }

  for(aName = (Name *) structure->getlistOfNames()-
   >setCurrentToBeginning(), nameCount=1;
      nameCount<whichCount;
      aName = (Name *) structure->getlistOfNames()->Next(),
   nameCount++);

  printf("%s chosen\n",aName->getNameString());
  return aName;
}

void setM_plate(double mat[4][4], double gap, double phip, double
   phir)
{
  double rotr[4][4];
  double rotp[4][4];
  double tran[4][4];
  double temp[4][4];
  double A[3];

  phip *= 180/3.1415926;
  phir *= 180/3.1415926;

  id_mat(tran);
  A[0]=0;  A[1]=0;  A[2]=gap;
  makeTranslate(tran,A);

  id_mat(rotr);
  A[0]=0;  A[1]=0;  A[2]=1;
  makeRotate(rotr,phir,A);

  id_mat(rotp);
  A[0]=0;  A[1]=1;  A[2]=0;
  makeRotate(rotp,phip,A);

  multM(temp,rotp,rotr);
  multM(mat,tran,temp);
}

void plateTransform(Name *nameToTransform, double gap, double
   phip, double phir)
```

```
{
  double mat[4][4];
  Transformation *trans = new Transformation;

  setM_plate(mat,gap,phip,phir);
  trans->setmatrix(mat);
  nameToTransform->applyNodeTransformToName(trans);
}

void simulateExtractFaces1(PATRAN_neutral_file *structure)
{
  structure->getFaces();
  structure->removeInteriorFaces();
  structure->organizeFaces();
}

void simulateExtractFaces2(PATRAN_neutral_file *structure)
{
  structure->checkFacePlanarity(1.0e-3);
  structure->tagSubPatchesbyNormal();
}
```

A.2 Numerical Model

This code is written in the MATLAB script language.

A.2.1 Non-Linear Fitting

nlfit.m

```
function []=nlfit

data = readCap('testdata.circ.55');
N = size(data,1);

x = data(:,1:2)*diag([1e-6 1]);
y = data(:,4)*1e-15;

sig = 0.01*ones(N,1);
a0 = [1;1;0];

a = mrqmin(x,a0,y,sig,'capfn');
y0 = capfn(x,a0);
yfit = capfn(x,a);

fid = fopen('output','w');
for i=1:length(a),
  fprintf(fid,'a(%d) = %15.13e\n',i,a(i));
end
fprintf(fid,'gap\tphi_p\tfastcap\tideal\tfit\n');
```

```
for i=1:N,
   fprintf(fid,'%12.8e\t%12.8e\t%12.8e\t%12.8e\t%12.8e\n',x(i,1),
   x(i,2),y(i),y0(i),yfit(i));
end
fclose(fid);
```

mrqmin.m

```
function [a] = mrqmin(x,a0,y,sig,funcs);

a = a0;
[alpha,beta,minchisq] = mrqcof(x,a,y,sig,funcs);
lambda = 0.001;

fprintf(1,'initial Chi^2 = %12.8f\n',minchisq);
for i=1:length(a),
  fprintf('a(%d) = %15.13e\n',i,a(i));
end

da = 1;
while norm(da)>1e-8,
  alphap = alpha + diag(lambda*diag(alpha));
  da = prec(alphap,beta);

  [alpha,beta,chisq] = mrqcof(x,a + da,y,sig,funcs);

  if (chisq < minchisq),
    a = a + da;
    minchisq = chisq;
    lambda = lambda * 0.1;
  else
    lambda = lambda * 10.0;
  end

  fprintf(1,'Chi^2 = %12.8e  Min(Chi^2) =
   %12.8e\n',chisq,minchisq);
  for i=1:length(a),
    fprintf('a(%d) = %15.13e\n',i,a(i));
  end;
end
```

mrqcof.m

```
function [alpha, beta, chisq] = mrqcof(x,a,y,sig,func);

%alpha: #param x #param
%beta: #param x 1
%chisq: 1 x 1

[ymod,dyda] = feval(func,x,a);
dy = y - ymod;
sig2i = 1.0./(sig.^2);
```

```
wdyda = diag(sig2i)*dyda;

alpha = wdyda'*dyda;
beta = wdyda'*dy;
chisq = sum(dy.^2.*sig2i);
```

prec.m

```
function [x] = prec(A,b);

P = diag(1./sqrt(diag(A)));
x = P*((P*A*P)\(P*b));
```

capfn.m

```
function [C,dCda] = capfn(q,a);
%capfn:
%  assumes q = [gap(m) phi_p(rad)]
%  returns [C,dCda]

N = size(q,1);
C = zeros(N,1);
dCda = zeros(N,3);

eps0 = 8.8541878162e-12;
R = a(2)*50e-6;

for i=1:N,
  g = a(1)*q(i,1);
  phi = q(i,2);
  r = g/R;

  if abs(phi)<=1e-4,
    C(i)      = eps0*pi*R * ( (1+a(3)*r)/r - (2*r^2-
   3)*(1+a(3)*r)*phi^2/(12*r^3) );
    dCda(i,1) = eps0*pi*R * ( -1/r + (2*r^2-6*a(3)*r-9)*phi^2/
   (12*r^3) ) / a(1);
    dCda(i,2) = eps0*pi*R * ( (2+a(3)*r)/r - (4*r^2+2*a(3)*r^3-
   9*a(3)*r-12)*phi^2/(12*r^3) ) / a(2);
    dCda(i,3) = eps0*pi*R * ( 1 - (2*r^2-3)*phi^2/(12*r^2) );
  else
    s = sqrt(r^2-sin(phi)^2);
    C(i)      = 2*pi*eps0*R/(phi*sin(phi)) * (r-s)*(1+a(3)*r);
    dCda(i,1) = 2*pi*eps0*R/(phi*sin(phi)) * (r*(1-r/s)*(1+a(3)*r)
   + a(3)*r*(r-s))/a(1);
    dCda(i,2) = 2*pi*eps0*R/(phi*sin(phi)) * ((r^2/s-s)*(1+a(3)*r)
   - a(3)*r*(r-s))/a(2);
    dCda(i,3) = 2*pi*eps0*R/(phi*sin(phi)) * r*(r-s);
  end
end
```

readCap.m

```
function [result] = readCap(file)

N = 0;
fid = fopen(file,'r');
fgetl(fid);
while 1
  line = fgetl(fid);
  if ~isstr(line), break, end
  if length(line) ~= 0,
    N = N + 1;
    stuff = sscanf(line, '%lf %lf %lf %lf');
    result(N,:) = stuff';
  end
end
```

A.3 Dynamics Simulation

This code is written in the MATLAB script language.

A.3.1 Initialization

globals.m

```
global PLATE_INV_M PLATE_R PLATE_GAP;
global SPRING_CONST SPRING_AXIS SPRING_NUM;
global DRIVE_MAX_V DRIVE_MAX_T;
```

load_all.m

```
function [] = load_all

globals;
load_plate;
load_springs;
DRIVE_MAX_V = 150;
DRIVE_MAX_T = 1e-4;
```

load_plate.m

```
function [] = load_plate

globals;

plate_density = 2331;   % kg/m^3
plate_radius = 50e-6;   % m
plate_thickness = 5e-6; % m
```

```
plate_mass = plate_density*pi*plate_radius^2*plate_thickness;
plate_i = plate_mass*(plate_radius^2/4+plate_thickness^2/3);

PLATE_INV_M = inv([  plate_mass   0            0
                     0            plate_i      0
                     0            0            plate_i]);
PLATE_R = plate_radius;
PLATE_GAP = 5e-6;
```

load_springs.m

```
function [] = load_springs

globals;
if (size(PLATE_R)==0)
  load_plate;
end

%              L          w          t          axis(x=1,y=2)?
springs = [    100e-6     10e-6      1.8e-6      1
               100e-6     10e-6      2e-6       -1
               100e-6     10e-6      2e-6        2
               100e-6     10e-6      1.9e-6     -2];
SPRING_NUM = size(springs,1);

L  = springs(:,1);
w  = springs(:,2);
t  = springs(:,3);

E  = 155e9; % Young's modulus
nu = 0.3;   % Poisson ratio

I = w.*t.^3/12;
k = 12*E*I./L.^3; % z axis translation spring constant

G = E/(2*(1+nu));
K = w.*t.^3.*(1/3-(3.36/16)*(t./w).*(1-((t./w).^4)/12));
kappa = G*K./L;   % beam axis rotational spring constant

SPRING_CONST(1,:) = k';
SPRING_CONST(2,:) = kappa';
SPRING_AXIS(:) = springs(:,4);
```

A.3.2 Acceleration Function

d2xdt2.m

```
function [d2xdt2] = d2xdt2(t,x);

globals;

d2xdt2 = -PLATE_INV_M * (capacitor_contrib(t,x) +
```

```
  spring_contrib(t,x));
```

capacitor_contrib.m

```
function [cc] = capacitor_contrib(t,x);

globals;

[q,dqdx] = q_cap(x);
[C,dCdq] = capacitance(q);
V = voltage(t);
dUdq = -(1/2)*dCdq*V^2;
cc = dqdx*dUdq;
```

q_cap.m

```
function [q,dqdx] = q_cap(x);
% x = [ g  phi_x  phi_y ]
% q = [ g  phi_p ]

q = zeros(2,1);
dqdx = zeros(3,2);

q(1) = x(1);
q(2) = atan(sqrt(tan(x(2))^2+tan(x(3))^2));

if (tan(q(2)) <= eps)
  dq2dx2 = 1;
  dq2dx3 = 1;
else
  dq2dx2 = tan(x(2))*sec(x(2))^2/(tan(q(2))*sec(q(2))^2);
  dq2dx3 = tan(x(3))*sec(x(3))^2/(tan(q(2))*sec(q(2))^2);
end

dqdx = [ 1      0
         0   dq2dx2
         0   dq2dx3 ];
```

capacitance.m

```
function [C,dCdq] = capacitance(q)
%capfn:
%  assumes q = [gap(m) phi_p(rad)]
%  returns [C,dCda]

globals;

N = size(q,2);
C = zeros(1,N);
dCdq = zeros(2,N);

a = [1.0974241404426e+00 1.0558692307271e+00 3.8474147762451e+00];
```

```
eps0 = 8.8541878162e-12;
R = a(2)*PLATE_R;

for i=1:N,
  g = a(1)*q(1,i);
  phi = q(2,i);
  r = g/R;

  if abs(phi)<=1e-4
    C(i)       = eps0*pi*R * ( (1+a(3)*r)/r - (2*r^2-
   3)*(1+a(3)*r)*phi^2/(12*r^3) );
    dCdq(1,i) = eps0*pi*a(1) * ( -1/r^2 + (2*r^2-6*a(3)*r-9)*phi^2/
   (12*r^4) );
    dCdq(2,i) = - eps0*pi*R * (2*r^2-3)*(1+a(3)*r)*phi/(6*r^3);
  else
    s = sqrt(r^2-sin(phi)^2);
    C(i)       = 2*pi*eps0*R/(phi*sin(phi)) * (r-s)*(1+a(3)*r);
    dCdq(1,i) = 2*pi*eps0/(phi*sin(phi)) * a(1) * ((1-r/
   s)*(1+a(3)*r)+a(3)*(r-s));
    dCdq(2,i) = 2*pi*eps0*R/(phi*sin(phi)) *
   (1+a(3)*r)*(sin(phi)*cos(phi)/s-(r-s)*(1/phi+cot(phi)));
  end
end
```

voltage.m

```
function [V] = voltage(t);

globals;

V = zeros(size(t));
for i=1:length(V),
  if t(i)<=0,
    V(i) = 0;
  elseif t(i)<=DRIVE_MAX_T,
    V(i) = DRIVE_MAX_V*t(i)/DRIVE_MAX_T;
  else
    V(i) = DRIVE_MAX_V;
  end
end
```

spring_contrib.m

```
function [sc] = spring_contrib(t,x);

globals;

sc = zeros(3,1);
for i=1:SPRING_NUM,
  [q,dqdx] = q_spr(x,SPRING_AXIS(i));
  dUdq = SPRING_CONST(:,i) .* q;
  sc = sc + dqdx * dUdq;
end;
```

q_spr.m

```
function [q,dqdx] = q_spr(x,axis);
% x = [ g  phi_x  phi_y ]
% q = [ dz  dth ]

globals;

q = zeros(2,1);
dqdx = zeros(3,2);

if (abs(axis)==1),
  r = sign(axis) * PLATE_R;
  q(1) = (x(1)-PLATE_GAP)-r*tan(x(3));
  q(2) = x(2);
  dqdx = [        1        0
                  0        1
         -r*sec(x(3))^2 0   ];
else
  r = sign(axis) * PLATE_R;
  q(1) = (x(1)-PLATE_GAP)+r*tan(x(2));
  q(2) = x(3);
  dqdx = [        1        0
          r*sec(x(2))^2 0
                  0        1   ];
end
```

A.3.3 Equilibrium Determination

equilibrium.m

```
function [x] = equilibrium(x0);

x = djfnewton(x0~=0,'F',x0).*x0;
```

F.m

```
function [a] = F(x,x0);

globals;

if any(x0~=0),
  a0 = d2xdt2(DRIVE_MAX_T,x0);
end

for i=1:length(x0),
  if x0(i)~=0,
    x(i) = x(i).*x0(i);
  end
end
```

```
a = d2xdt2(DRIVE_MAX_T,x);

for i=1:length(a0),
  if x0(i)~=0,
    a(i) = a(i)./a0(i);
  end
end
```

djfnewton.m

```
function [x] = djfnewton(x0,F_fn,fixed);

tol_abs = 1e-6;
damp_frac = 0.5;
jf_alpha=0.001;
jf_tol_rel=0.001;

N = length(x0);
k = 0;
x = x0;
dx = zeros(N,1);
p = zeros(N,N);
Ap = zeros(N,N);

fprintf('start djfnewton...\n');
while 1,
  f = feval(F_fn,x,fixed);
  norm_f = norm(f);
  fprintf('|F| = %16.12f\n',norm_f);
  if(norm_f < tol_abs),
    break;
  end;

  r = -f;
  dx = 0*dx;

  for i = 1:N,
    p(:,i) = r;
    Ar = (feval(F_fn,x+jf_alpha*r,fixed)-f)/jf_alpha;

    Ap(:,i) = Ar;
    for j=1:i-1,
      beta = Ap(:,i)' * Ap(:,j);
      p(:,i) = p(:,i) - beta * p(:,j);
      Ap(:,i) = Ap(:,i) - beta * Ap(:,j);
    end;
    norm_ap = norm(Ap(:,i),2);
    p(:,i) = p(:,i)/norm_ap;
    Ap(:,i) = Ap(:,i)/norm_ap;

    alpha = r' * Ap(:,i);
    dx = dx + alpha * p(:,i);
    r = r - alpha * Ap(:,i);
```

```
      if(norm(r) < jf_tol_rel*norm_f),
        break;
      end;
    end;
    if(norm(r) >= jf_tol_rel*norm_f),
      fprintf('\nJF Newton GCR didn''t converge!\n');
    end;

    damp_alpha = 1;
    damp_norm = norm(feval(F_fn,x+dx,fixed));
    fprintf('|damp_F| = %16.12f\n',damp_norm);
    while damp_norm >= norm_f,
      damp_norm = norm(feval(F_fn,x+damp_alpha*dx,fixed));
      fprintf('|damp_F| = %16.12f\n',damp_norm);
      damp_alpha = damp_frac*damp_alpha;
    end;

    x = x + damp_alpha*dx;
    k = k + 1;
  end;
  %fprintf('...end djfnewton\n');
```

A.3.4 Normal Mode Determination

modes.m

```
function [w,v,k] = modes(x);

globals;

J = zeros(3,3);
for i=1:3,
  h=zeros(3,1);
  h(i)=x(i)*0.001+1e-8;
  old_dx = (d2xdt2(1,x+h)-d2xdt2(1,x-h))/(2*h(i));
  h=h/2;
  dx = (d2xdt2(1,x+h)-d2xdt2(1,x-h))/(2*h(i));
  while norm(old_dx-dx)>1e-3*norm(dx),
    old_dx = dx;
    h=h/2;
    dx = (d2xdt2(1,x+h)-d2xdt2(1,x-h))/(2*h(i));
  end;
  J(:,i) = dx;
end

[v,d]=eig(J);
w = sqrt(-diag(d));
k = inv(v)';
```

APPENDIX B Simulation Results

In order to demonstrate the functionality of the dynamics simulator developed in this report, the simulator was used to determine the motion of the suspended circular plate system with a variety of support beam thicknesses and voltage inputs. The results of those simulations are presented in this appendix.

B.1 Structure Specification

The structure simulated is shown in Figure 22. The structure is a rigid circular plate suspended 5μm above a ground plane by four thin beams placed along the x and y axes. The plate and beams are composed of polysilicon, which is assigned a density of $2330 kg/m^3$, a Young's modulus of 155GPa, and a Poisson ratio of 0.3. The circular plate has a radius of 50μm and a thickness of 5μm. Each beam has a length of 100μm, a width of 10μm, and a thickness that varies between experiments but is usually 2μm.

Depending upon how the beam thicknesses are chosen, motion can occur with different numbers of degrees of freedom and within any of the three

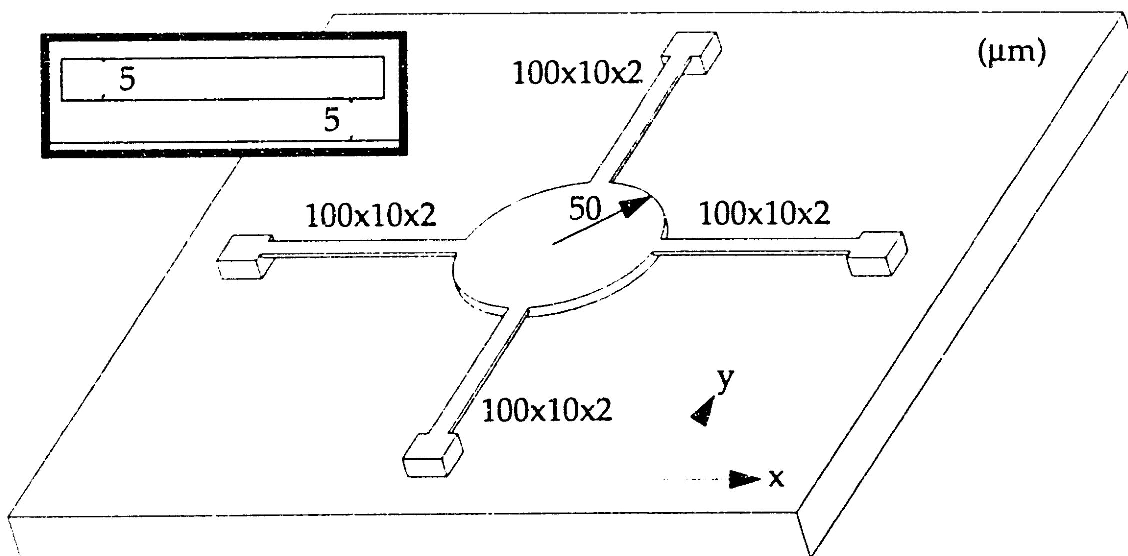


Figure 22: Simulated Suspended Circular Plate Structure Diagram

simulated parameters g, φ_x, and φ_y. Names are given to four structures with specially chosen beam thicknesses, each structure permitting more complex motion than the previous. For the simplest structure, henceforth called the 1D structure, all of the beam thicknesses are equal to 2µm. The symmetry of this structure prevents an applied voltage from causing any tilting motion. Thus, all motion is exhibited within g. For the second structure, henceforth called the 2D structure, the $-y$ axis beam is weakened by reducing its thickness to 1.8µm. Symmetry prevents tilting about the y axis, but allows it about the x axis. Thus, all motion occurs within two degrees of freedom exhibited within g and φ_x. The third structure, henceforth known as the 2D+ structure, is that for which the $-y$ and $+x$ axis beam thicknesses are reduced equally to 1.9µm. The structure is symmetric about the $y=-x$ axis; therefore an applied voltage will induce identical tilting about the x and y axes. Thus, two degrees of freedom are exhibited within all three simulated parameters, g, φ_x, and φ_y. The fourth structure, henceforth known as the 3D structure, is that for which the $-y$ and $+x$ axis beam thicknesses are reduced unequally to 1.9µm and 1.85µm respectively. This structure permits three degrees of freedom of motion exhibited within g, φ_x, and φ_y.

B.2 Experiments and Results

Three experiments are presented. The first experiment is the simulation of dynamics about equilibrium at constant voltage. The second experiment is the simulation of dynamics induced by a linear voltage ramp. The final experiment is the simulation of dynamics induced by linear voltage ramps for varying ramp rates.

B.2.1 Near Equilibrium Dynamics

In this first experiment, the 1D, 2D, 2D+, and 3D structures are held at a constant voltage of 150 volts. Each structure is displaced initially from its calculated equilibrium by 1%. The results of these simulations are given in Figures 23 through 38. The straight line across some of the plots indicates the equilibrium value as estimated by the implemented Newton method, and the dots depict the integration steps of the simulation. Table 2 presents the equilibrium states estimated for the 1D, 2D, 2D+, and 3D structures. Using this

Table 2: Equilibrium State for 1D, 2D, 2D+, and 3D Structures at 150 Volts

Equilibrium at 150v	**1D**	**2D**	**2D+**	**3D**
g **component (μm)**	3.997	3.787	3.793	3.708
φ_x **component (rad)**	0.000	$3.433\ 10^{-3}$	$1.733\ 10^{-3}$	$1.927\ 10^{-3}$
φ_y **component (rad)**	0.000	0.000	$1.733\ 10^{-3}$	$2.874\ 10^{-3}$

estimation of equilibrium, the normal modes, frequencies, and periods of small oscillation about equilibrium are calculated. The results are presented in Tables 3 through 6.

Table 3: Mode Information for 1D Structure at 150 Volts

1D Modes at 150 Volts	**mode 1**	**mode 2**	**mode 3**
g **component (m)**	1.000	0.000	0.000
φ_x **component (rad)**	0.000	1.000	0.000

Table 3: Mode Information for 1D Structure at 150 Volts

1D Modes at 150 Volts	mode 1	mode 2	mode 3
φ_y component (rad)	0.000	0.000	1.000
angular frequency (rad/sec)	$5.195\ 10^{5}$	$1.106\ 10^{6}$	$1.106\ 10^{6}$
period (sec)	$1.209\ 10^{-5}$	$5.683\ 10^{-6}$	$5.683\ 10^{-6}$

Table 4: Mode Information for 2D Structure at 150 Volts

2D Modes at 150 Volts	mode 1	mode 2	mode 3
g component (m)	$2.204\ 10^{-4}$	$2.874\ 10^{-6}$	0.000
φ_x component (rad)	-1.000	1.000	0.000
φ_y component (rad)	0.000	0.000	1.000
angular frequency (rad/sec)	$4.162\ 10^{6}$	$1.017\ 10^{6}$	$1.052\ 10^{6}$
period (sec)	$1.509\ 10^{-5}$	$6.180\ 10^{-6}$	$5.973\ 10^{-6}$

Table 5: Mode Information for 2D+ Structure at 150 Volts

2D+ Modes at 150 Volts	mode 1	mode 2	mode 3
g component (m)	$-3.114\ 10^{-4}$	$2.034\ 10^{-6}$	$9.816\ 10^{-13}$
φ_x component (rad)	$7.071\ 10^{-1}$	$7.071\ 10^{-1}$	$-7.071\ 10^{-1}$
φ_y component (rad)	$7.071\ 10^{-1}$	$7.071\ 10^{-1}$	$7.071\ 10^{-1}$
angular frequency (rad/sec)	$4.227\ 10^{5}$	$1.033\ 10^{6}$	$1.031\ 10^{6}$
period (sec)	$1.486\ 10^{-5}$	$6.082\ 10^{-6}$	$6.097\ 10^{-6}$

Table 6: Mode Information for 3D Structure at 150 Volts

3D Modes at 150 Volts	mode 1	mode 2	mode 3
g component (m)	$-2.306\ 10^{-4}$	$1.887\ 10^{-6}$	$-1.984\ 10^{-6}$
φ_x component (rad)	$5.535\ 10^{-1}$	$-2.248\ 10^{-1}$	$-9.757\ 10^{-1}$
φ_y component (rad)	$8.328\ 10^{-1}$	$9.744\ 10^{-1}$	$-2.191\ 10^{-1}$
angular frequency (rad/sec)	$3.773\ 10^{5}$	$1.004\ 10^{6}$	$1.013\ 10^{6}$
period (sec)	$1.665\ 10^{-5}$	$6.261\ 10^{-6}$	$6.204\ 10^{-6}$

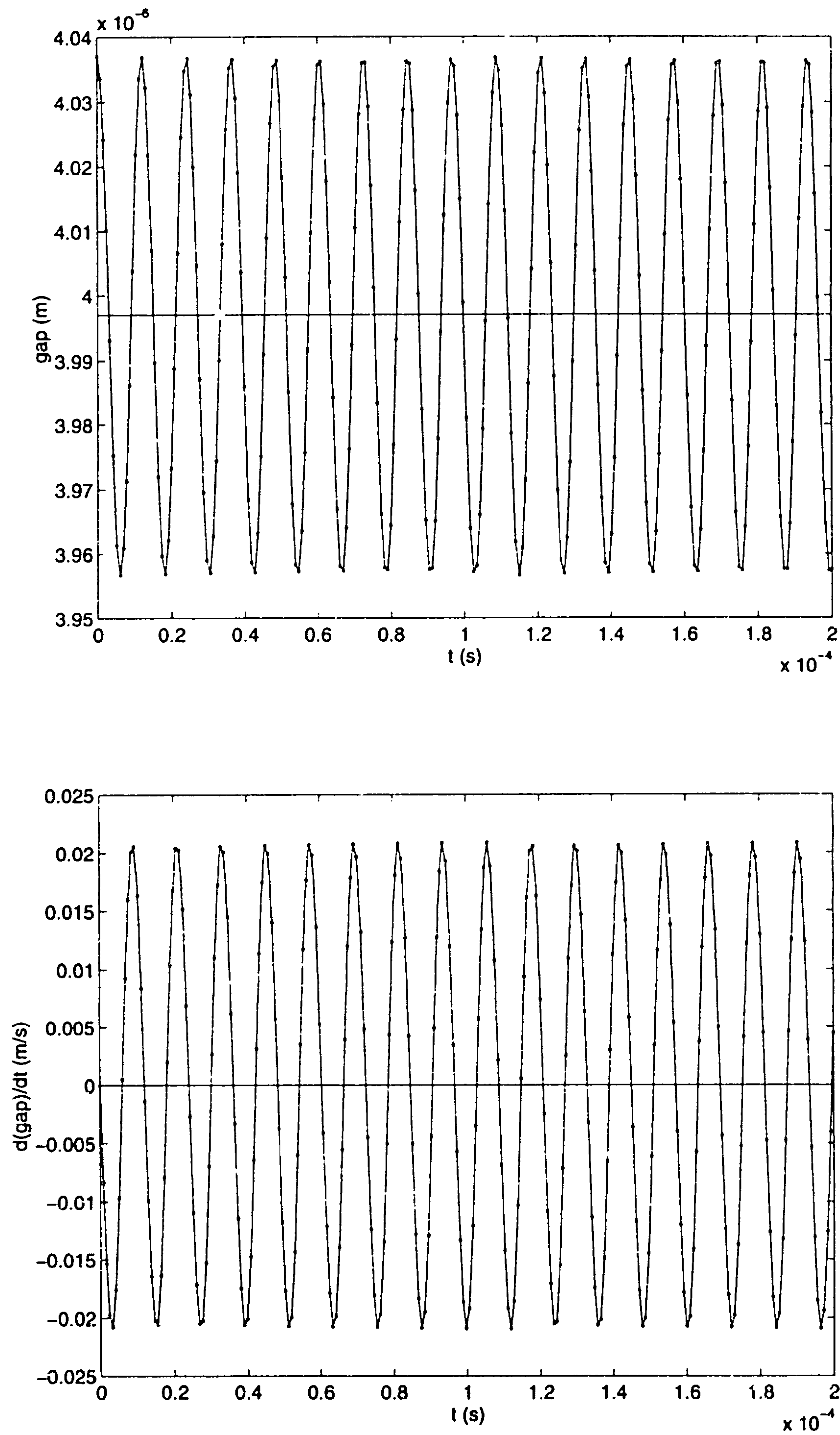


Figure 23: Near Equilibrium Dynamics - 1D Structure - *g*

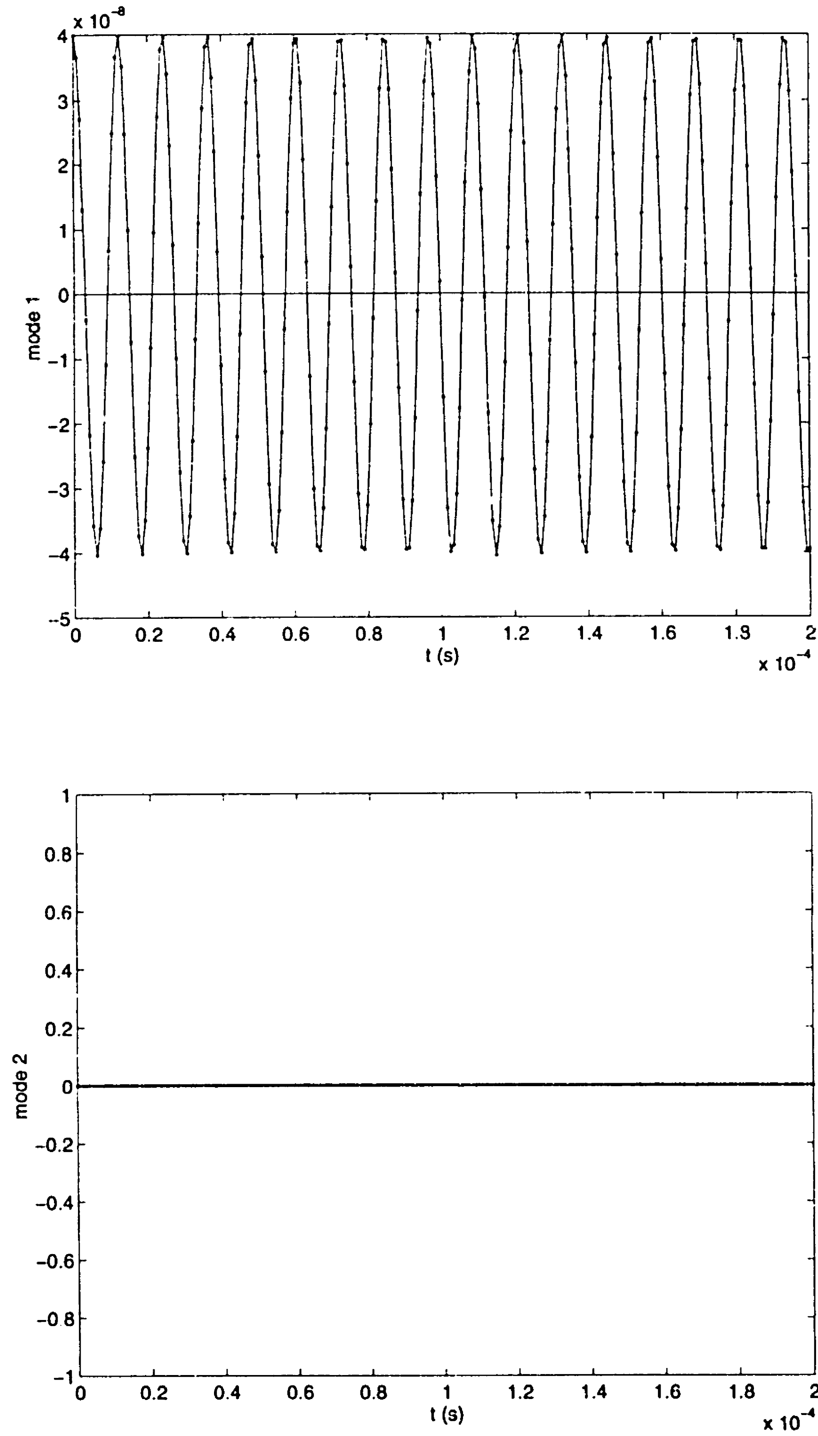


Figure 24: Near Equilibrium Dynamics - 1D Structure - Normal Modes

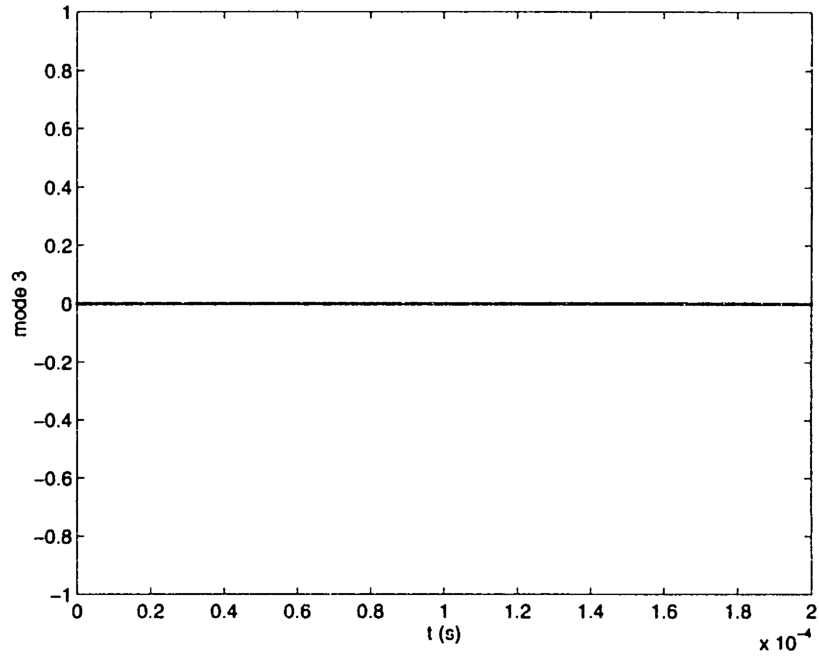


Figure 24 (cont): Near Equilibrium Dynamics - 1D Structure - Normal Modes

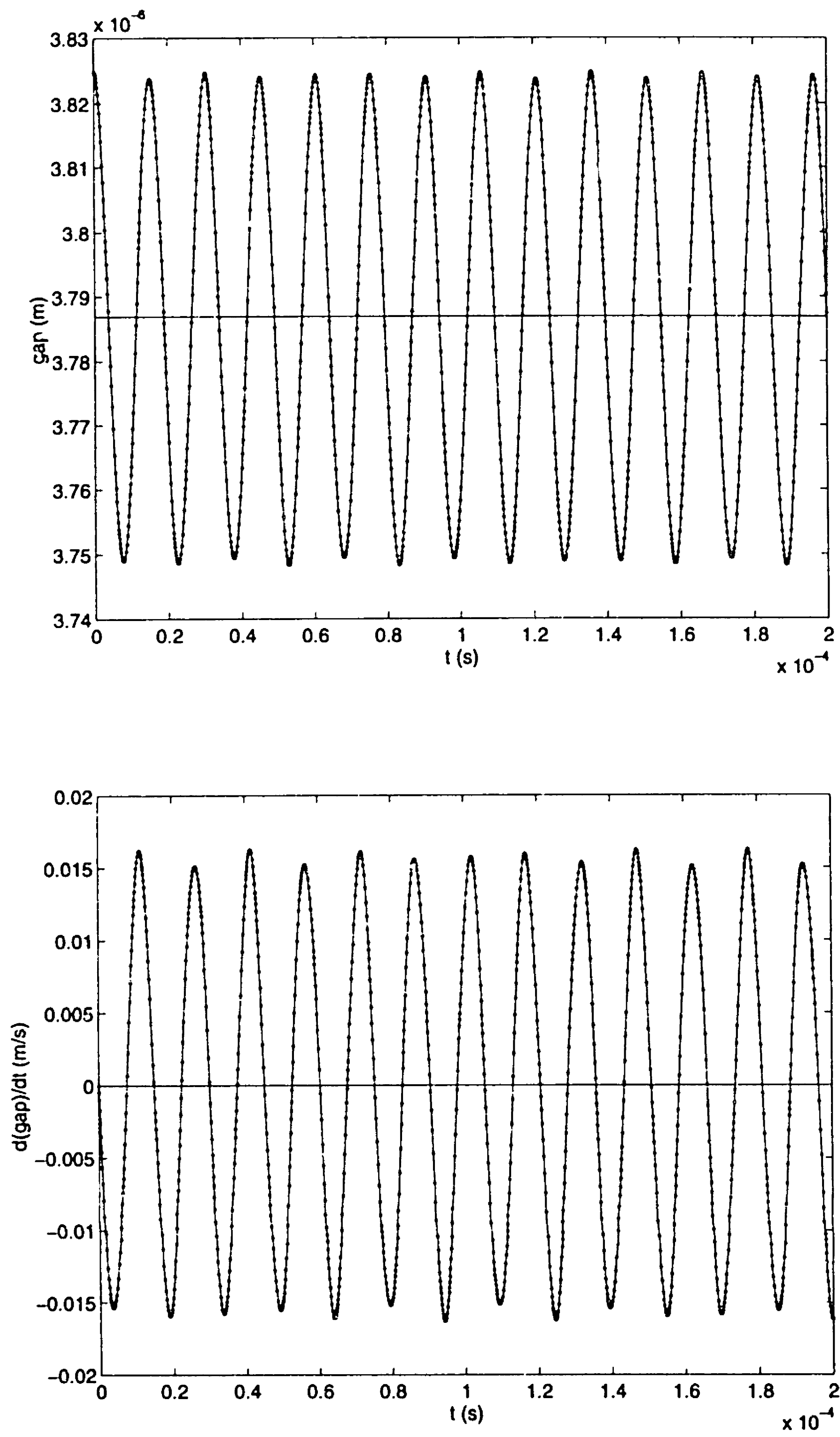


Figure 25: Near Equilibrium Dynamics - 2D Structure - *g*

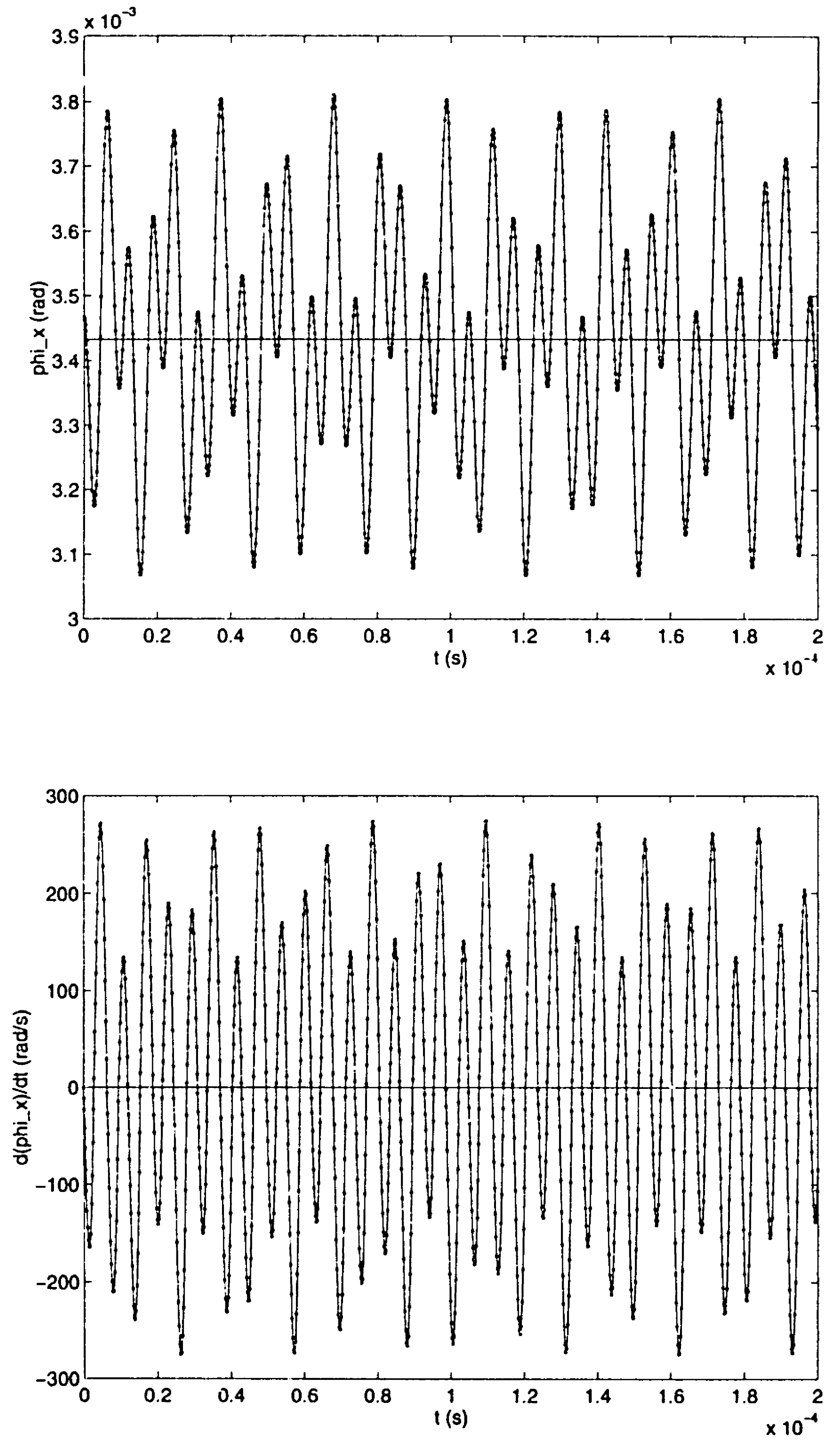
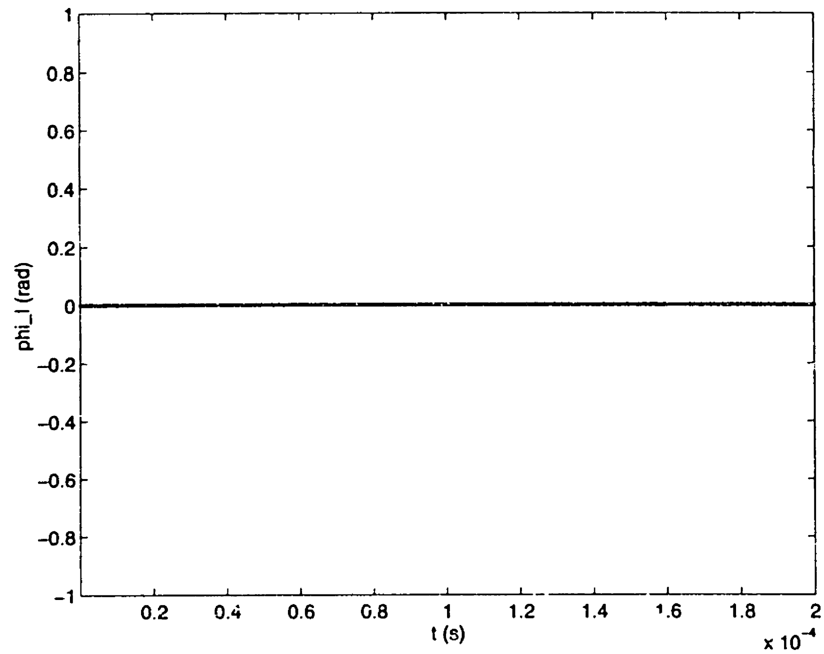


Figure 26: Near Equilibrium Dynamics - 2D Structure - φ_x



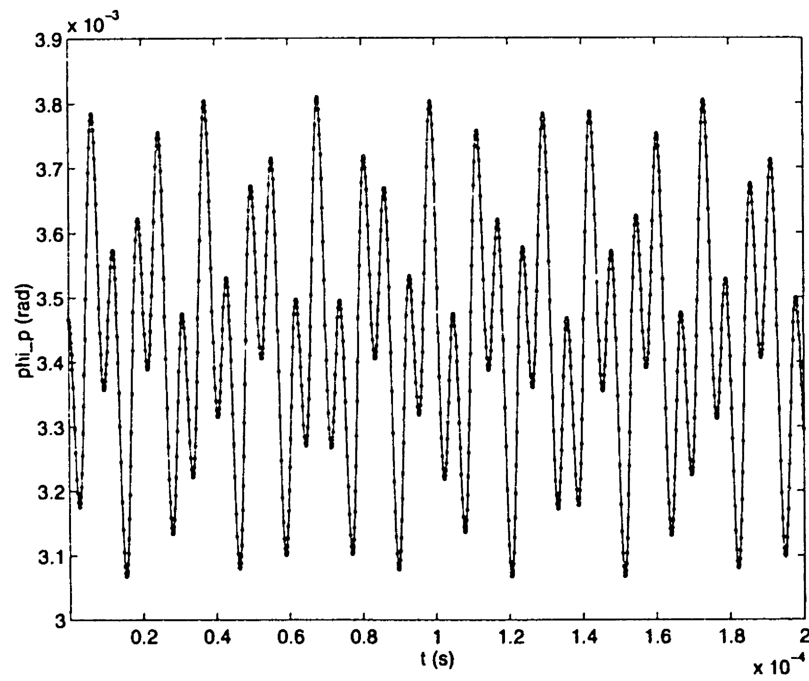
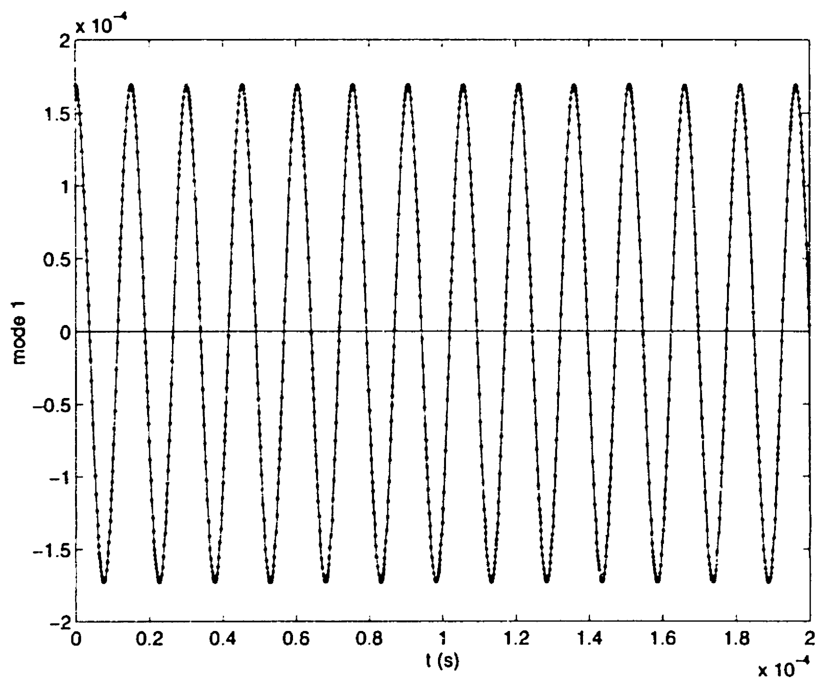


Figure 27: Near Equilibrium Dynamics - 2D Structure - Principle Tilt



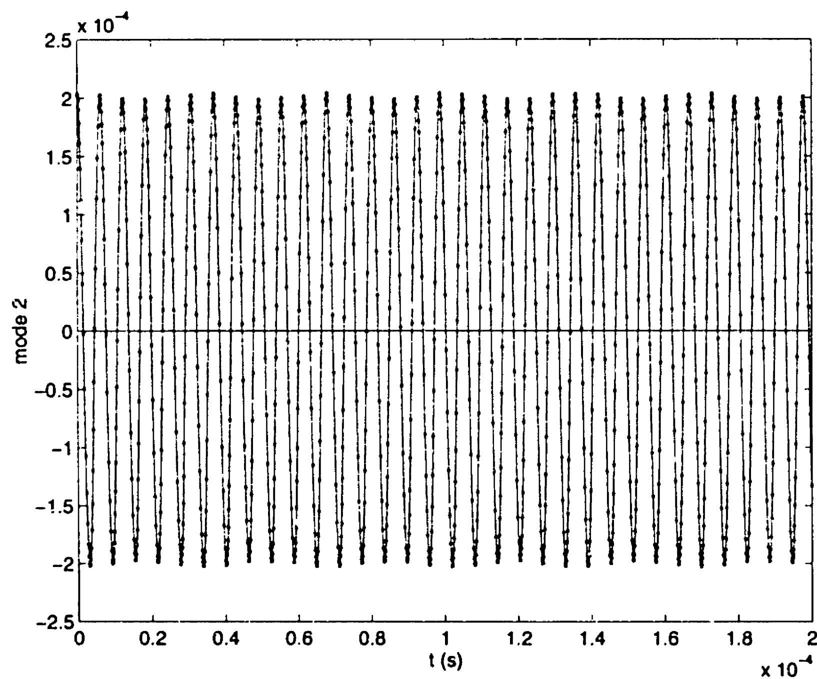


Figure 28: Near Equilibrium Dynamics - 2D Structure - Normal Modes

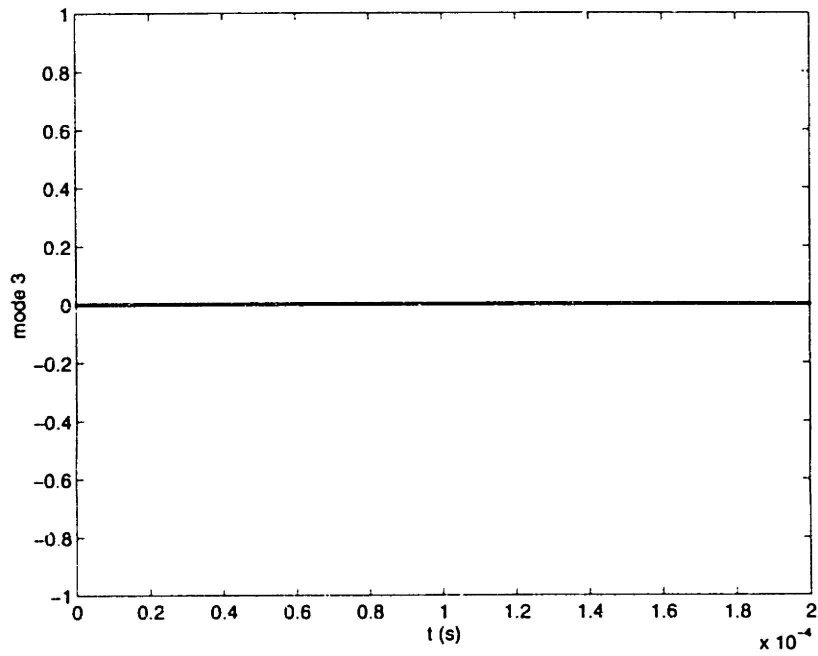


Figure 28 (cont): Near Equilibrium Dynamics - 2D Structure - Normal Modes

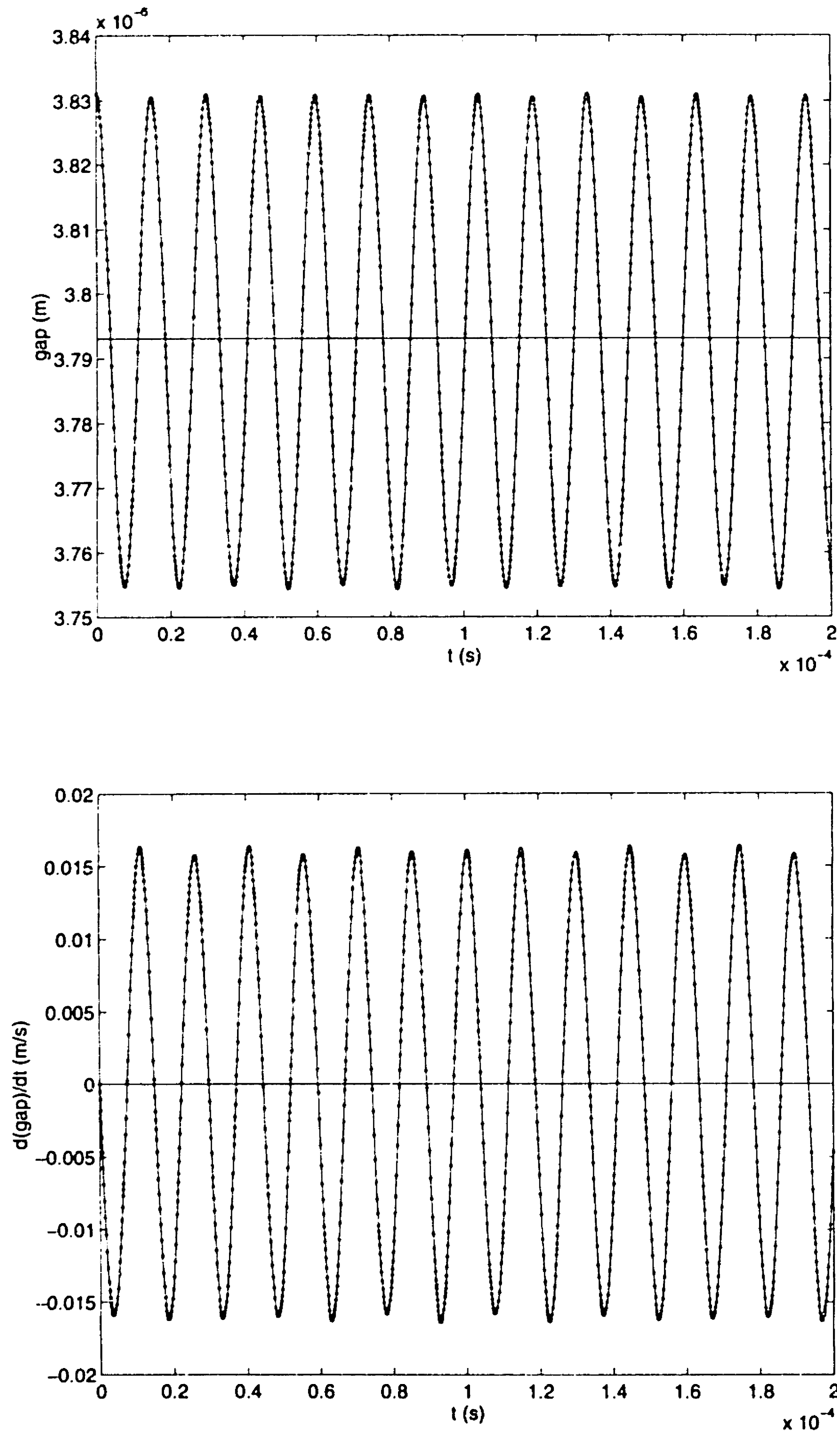


Figure 29: Near Equilibrium Dynamics - 2D+ Structure - *g*

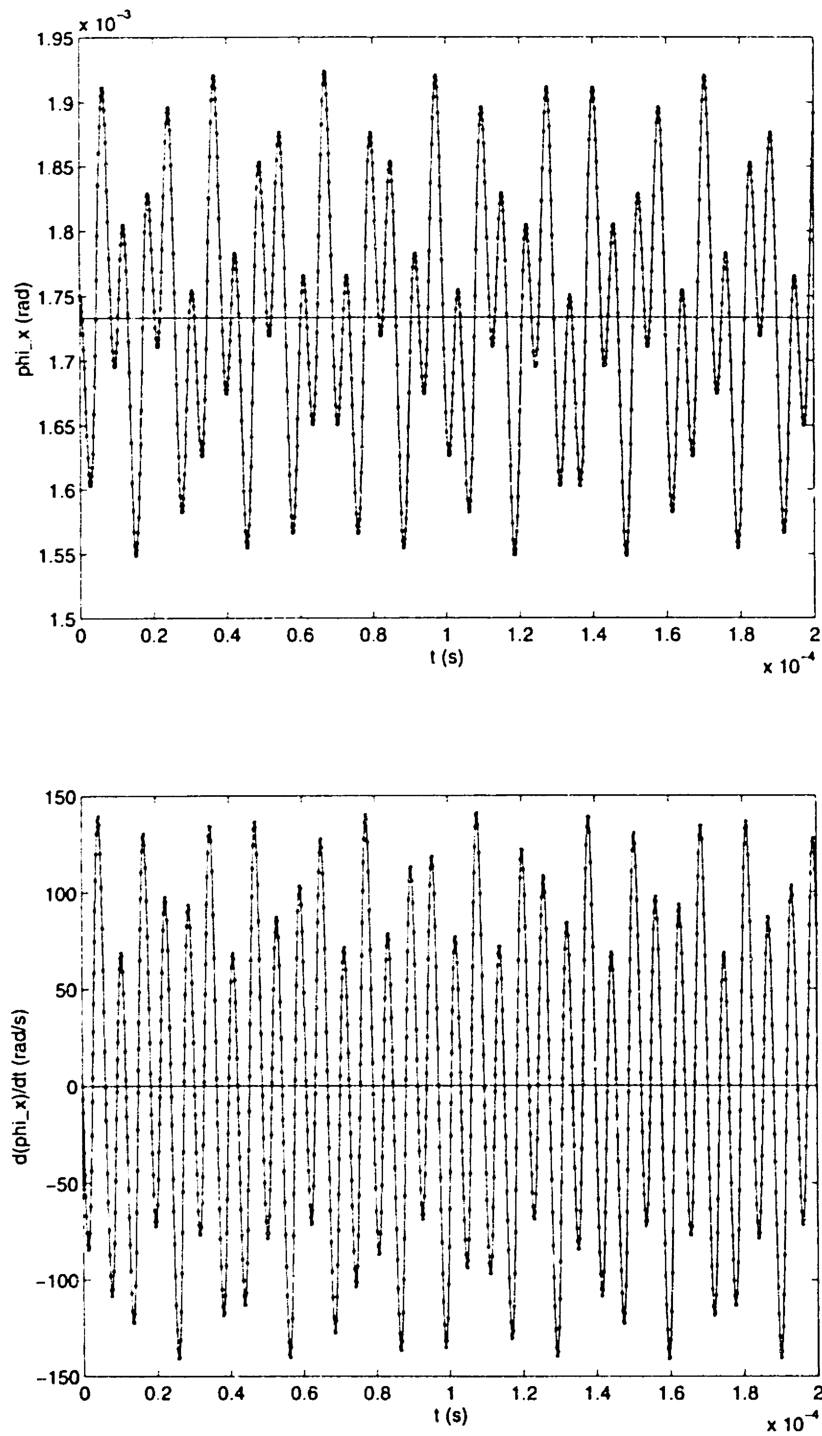
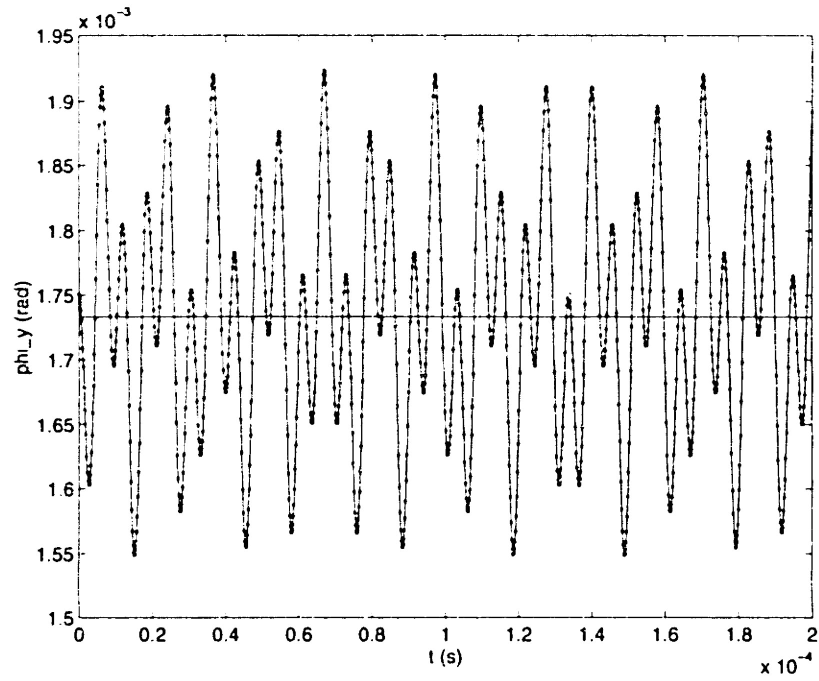


Figure 30: Near Equilibrium Dynamics - 2D+ Structure - φ_x



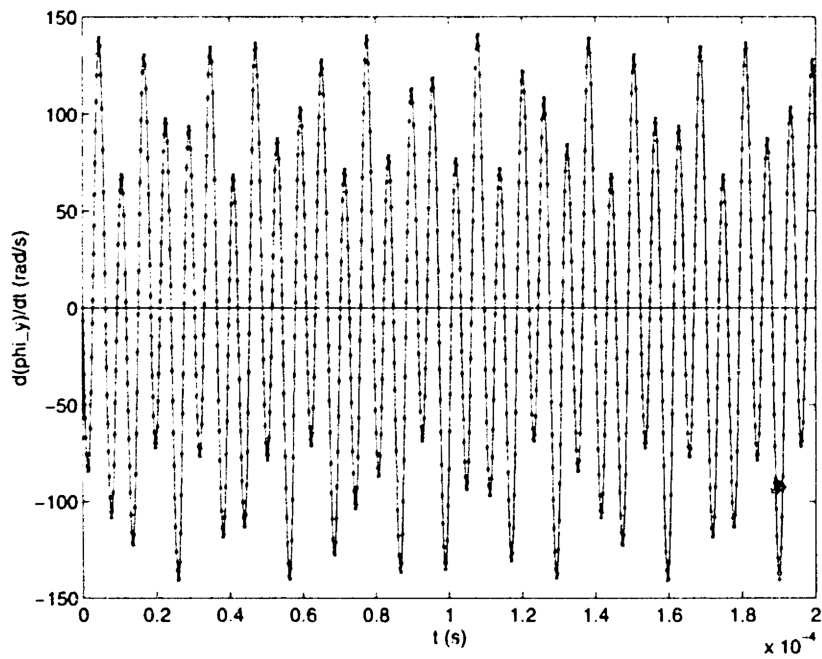


Figure 31: Near Equilibrium Dynamics - 2D+ Structure - φ_y

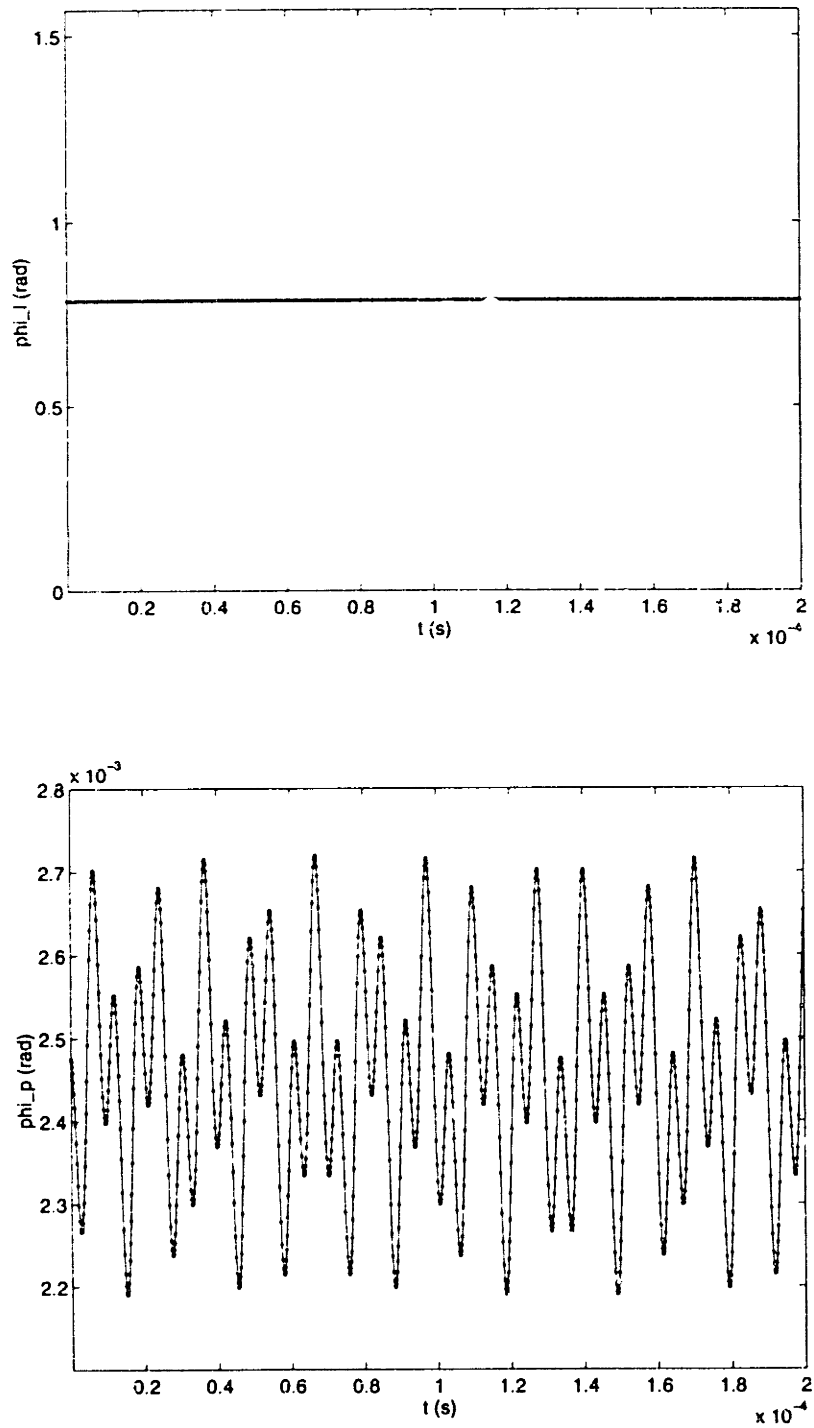
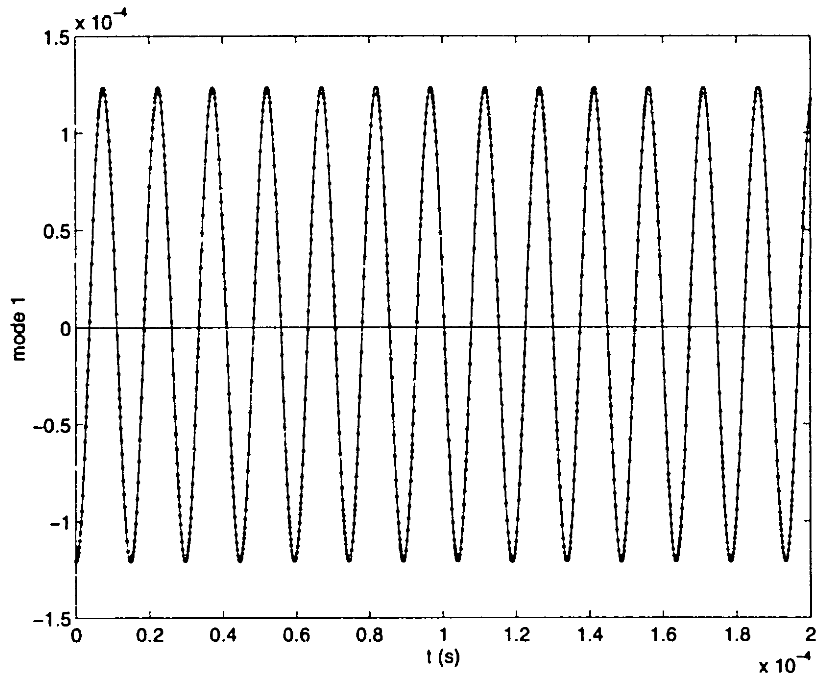


Figure 32: Near Equilibrium Dynamics - 2D+ Structure - Principle Tilt



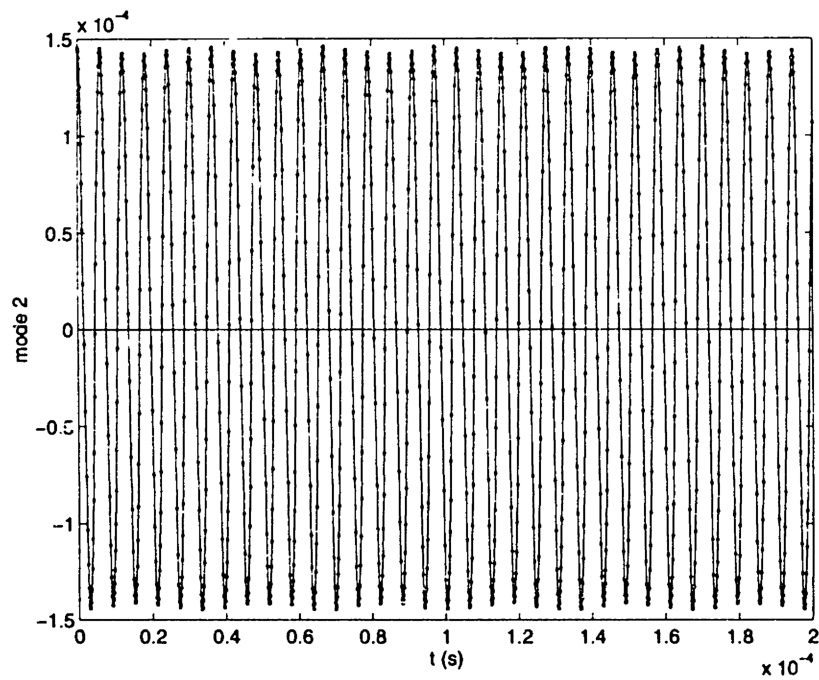


Figure 33: Near Equilibrium Dynamics - 2D+ Structure - Normal Modes

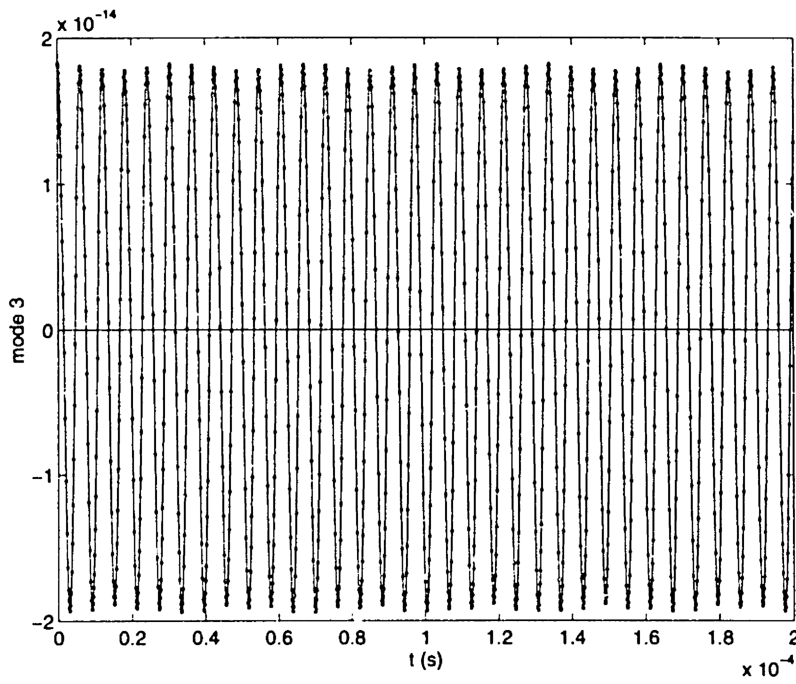


Figure 33 (cont): Near Equilibrium Dynamics - 2D+ Structure - Normal Modes

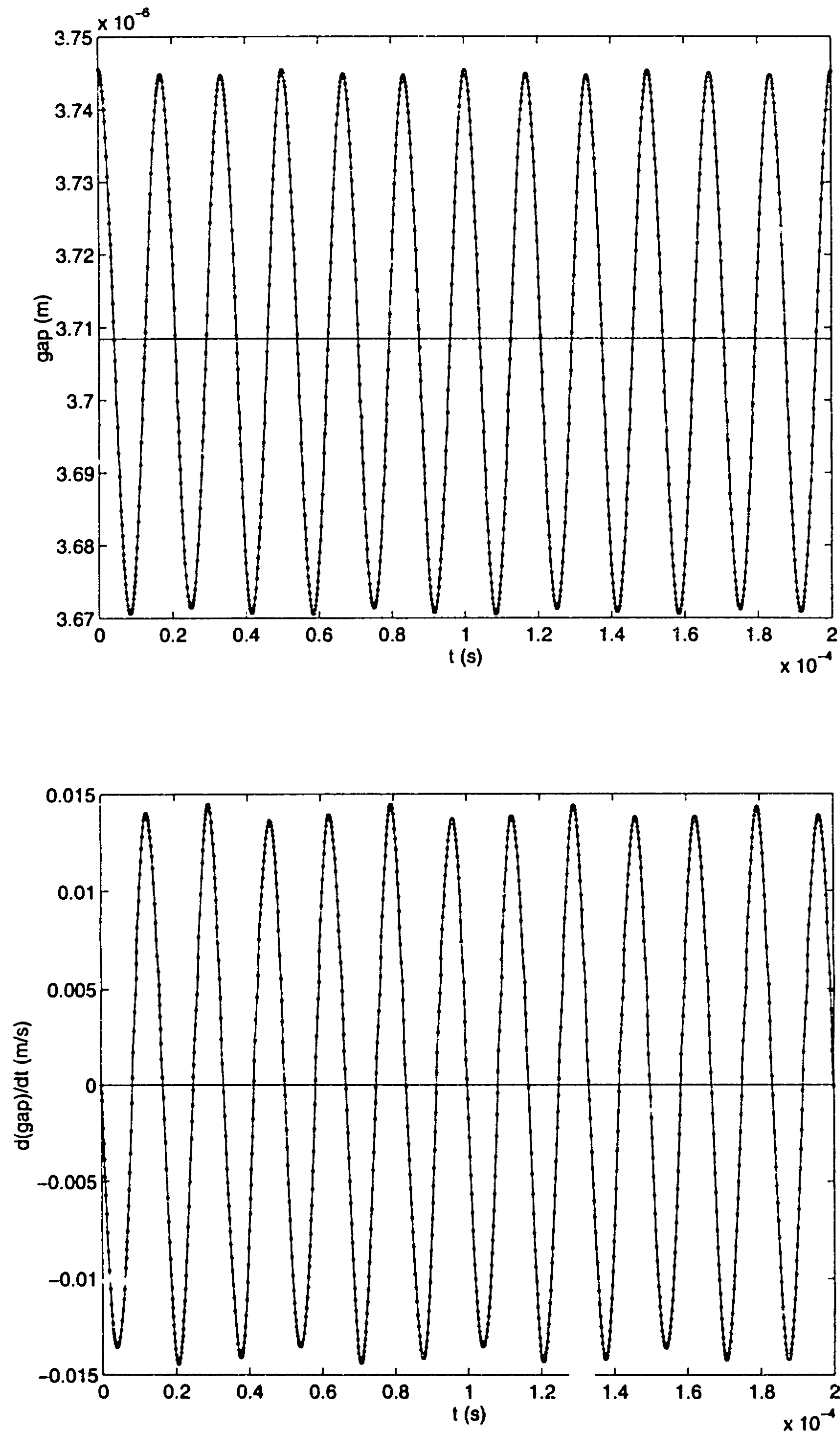


Figure 34: Near Equilibrium Dynamics - 3D Structure - g

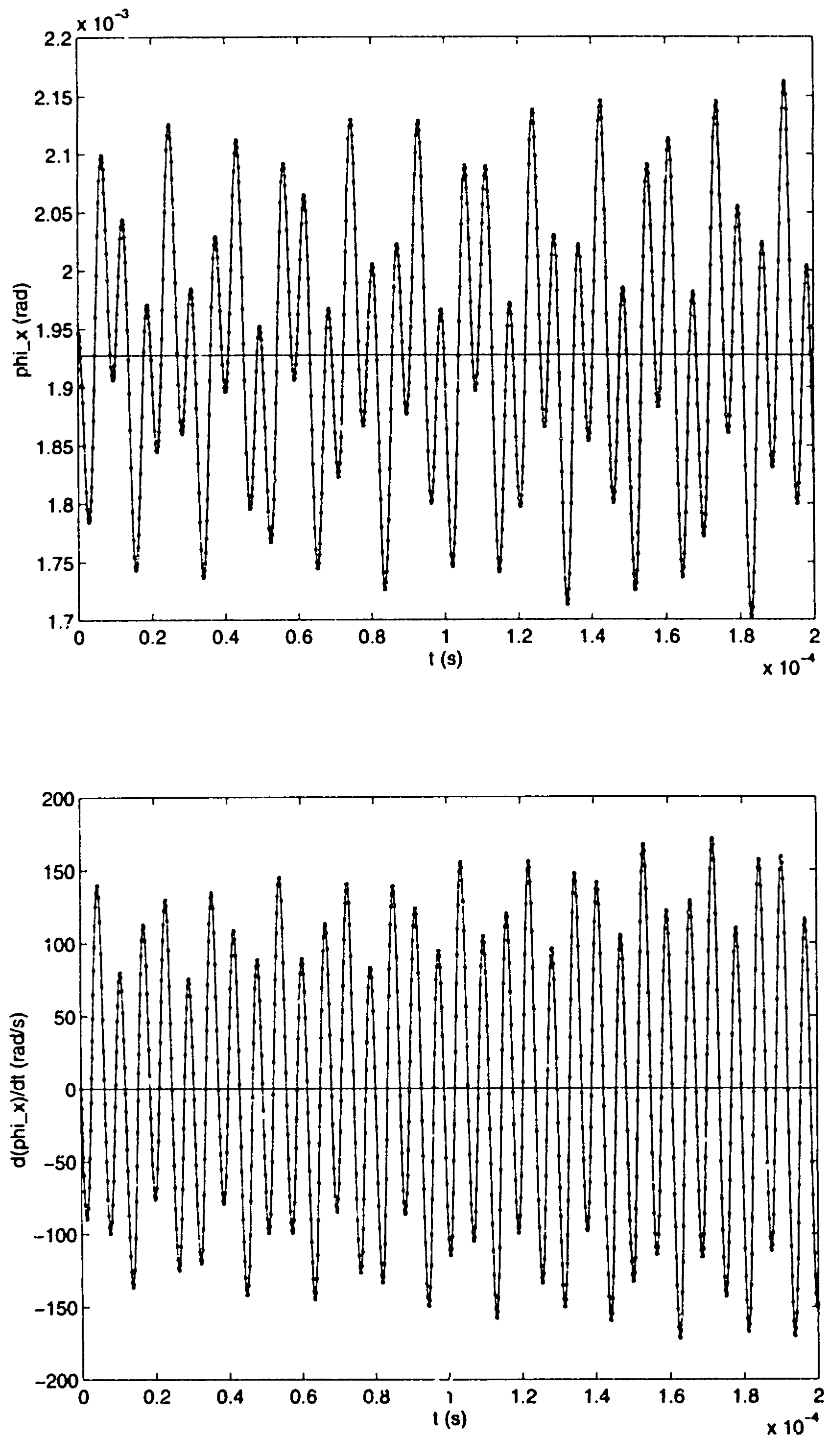
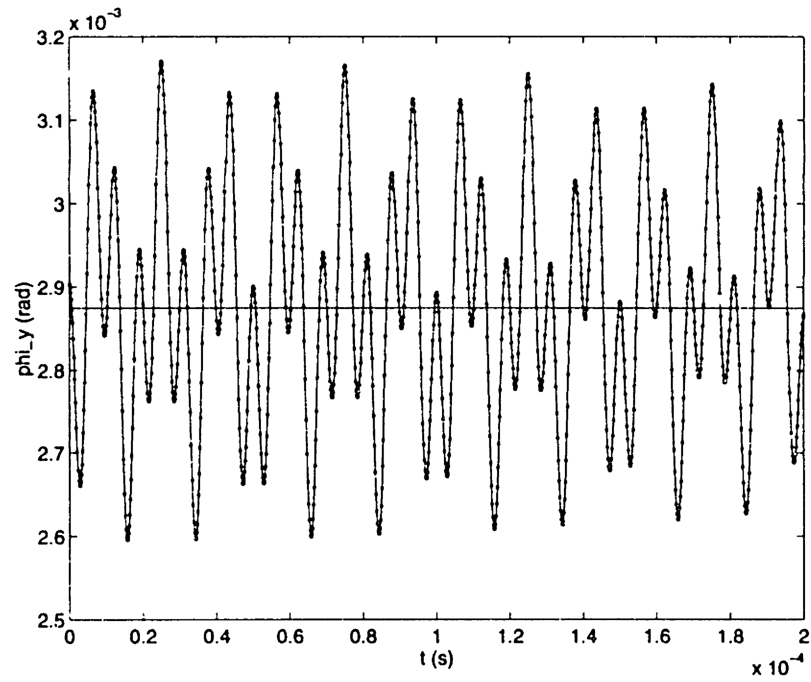


Figure 35: Near Equilibrium Dynamics - 3D Structure - φ_x



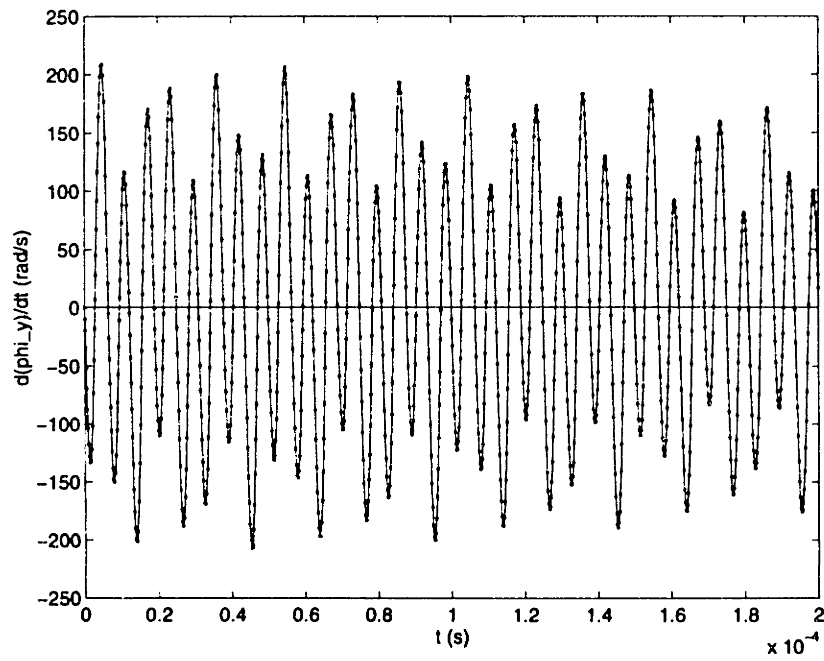
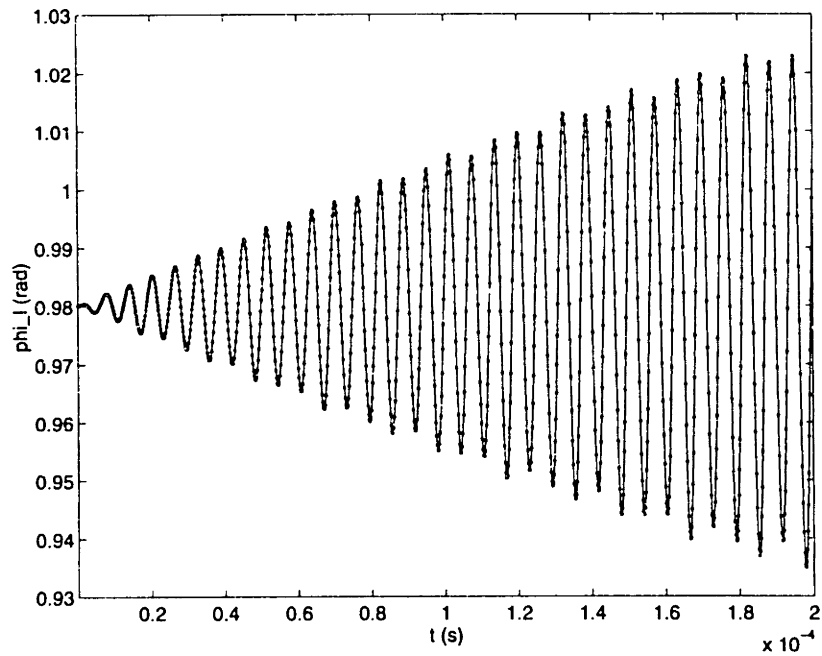


Figure 36: Near Equilibrium Dynamics - 3D Structure - φ_y



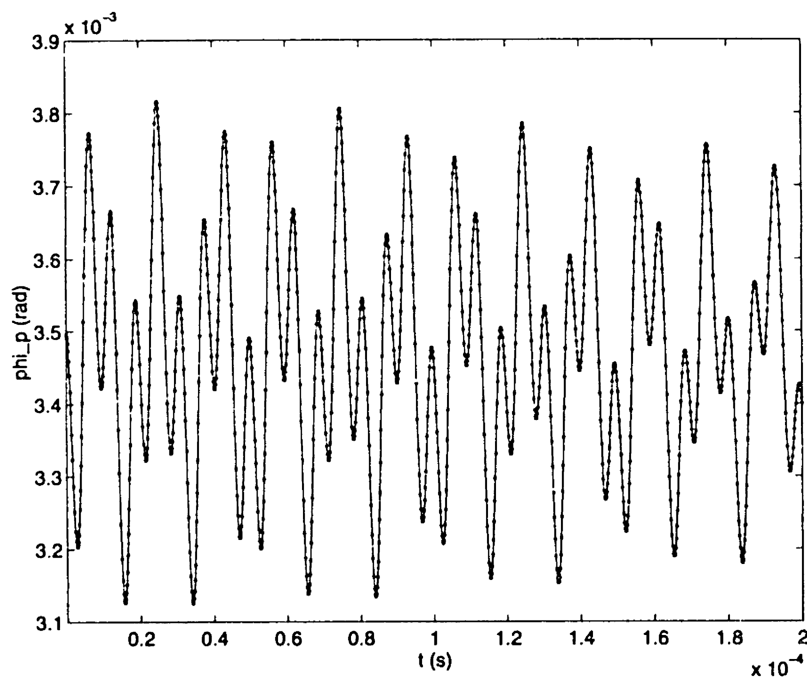
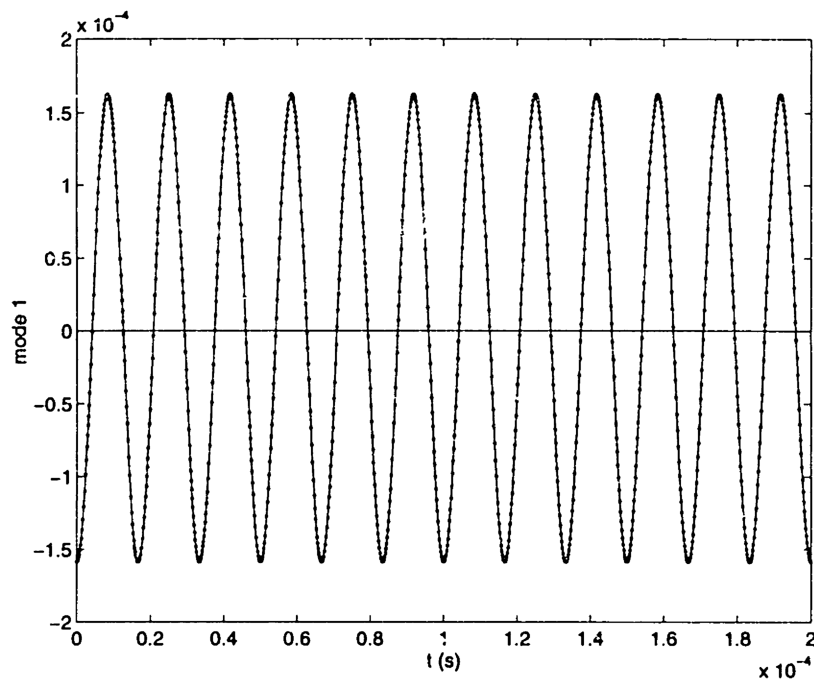


Figure 37: Near Equilibrium Dynamics - 3D Structure - Principle Tilt



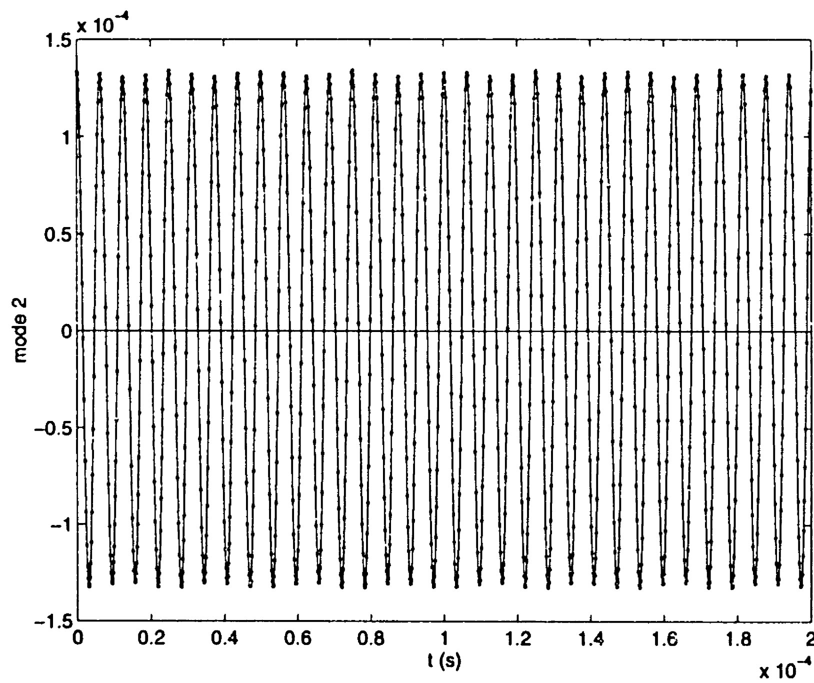


Figure 38: Near Equilibrium Dynamics - 3D Structure - Normal Modes

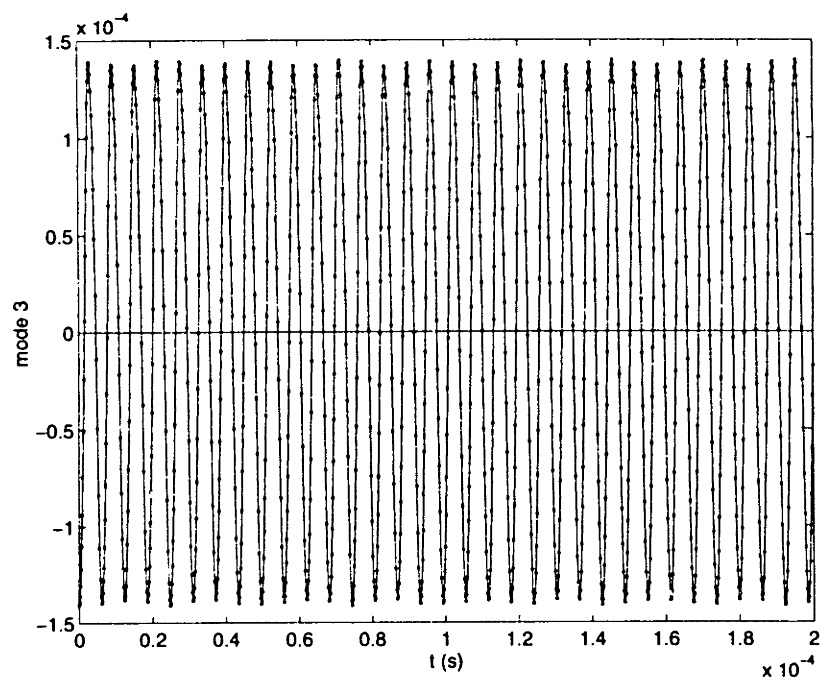


Figure 38 (cont): Near Equilibrium Dynamics - 3D Structure - Normal Modes

B.2.2 Voltage Ramp

In the second experiment, a time dependance upon voltage is introduced. At time t=0, the plate is at rest and no voltage is applied. After time t=0, the voltage is linearly increased over 10^{-4} seconds to 150 volts. The voltage is then held at 150 volts in order to observe the oscillation of the state around the new equilibrium of the system. The results of these simulations are given in Figures 39 through 54. The equilibrium and mode information given in the previous section holds for this experiment, because the final voltage level is the same.

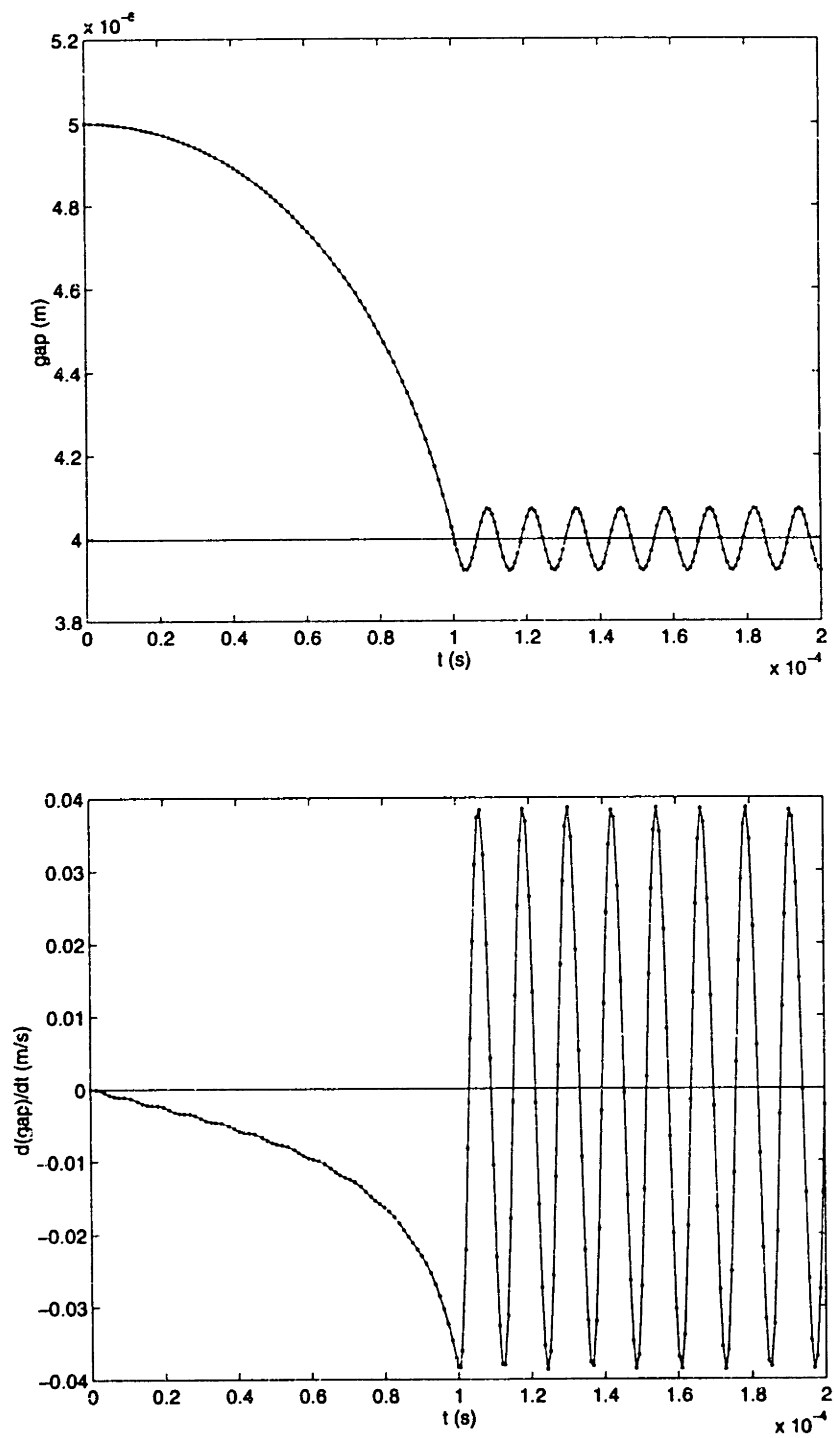


Figure 39: Voltage Ramp Dynamics - 1D Structure - *g*

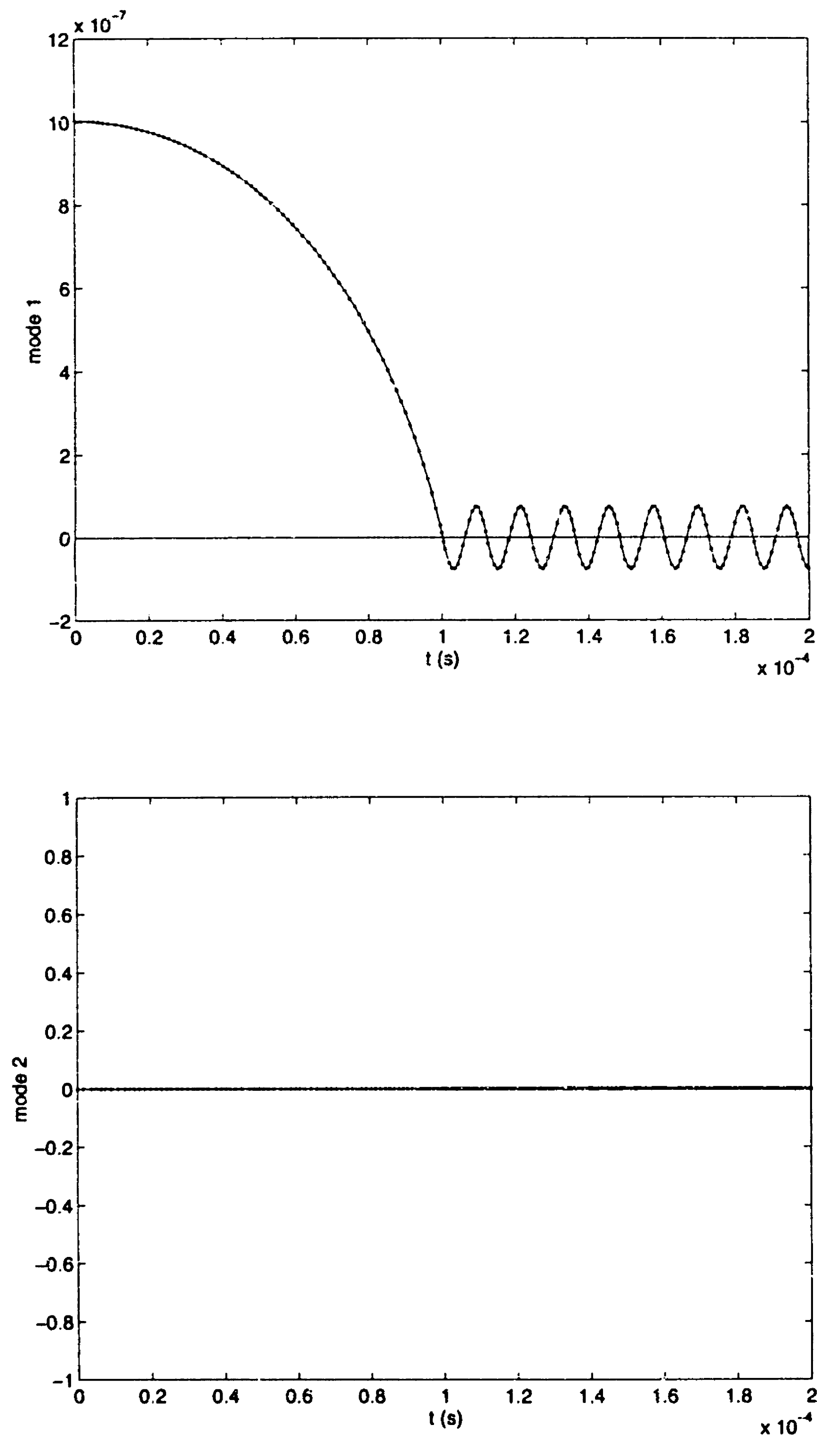


Figure 40: Voltage Ramp Dynamics - 1D Structure - Normal Modes

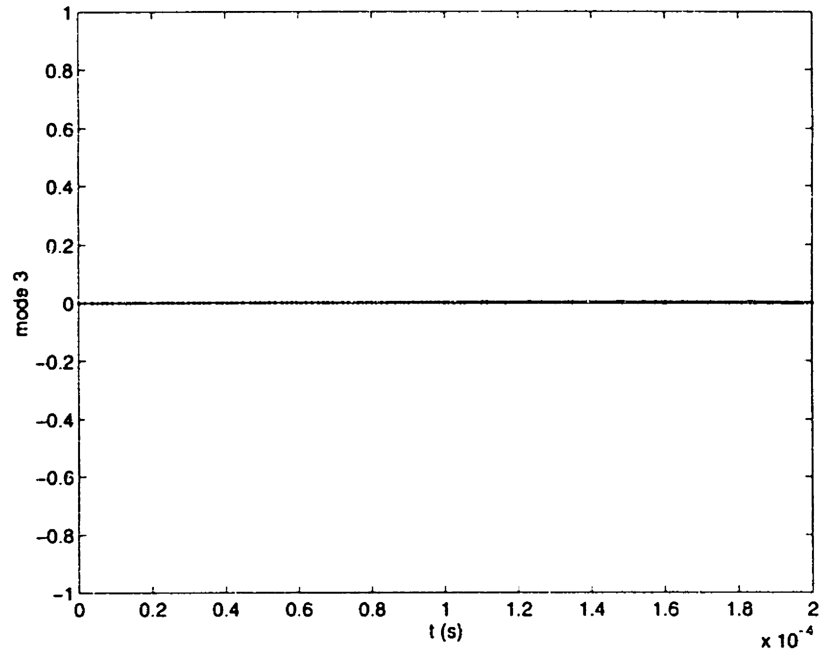


Figure 40 (cont): Voltage Ramp Dynamics - 1D Structure - Normal Modes

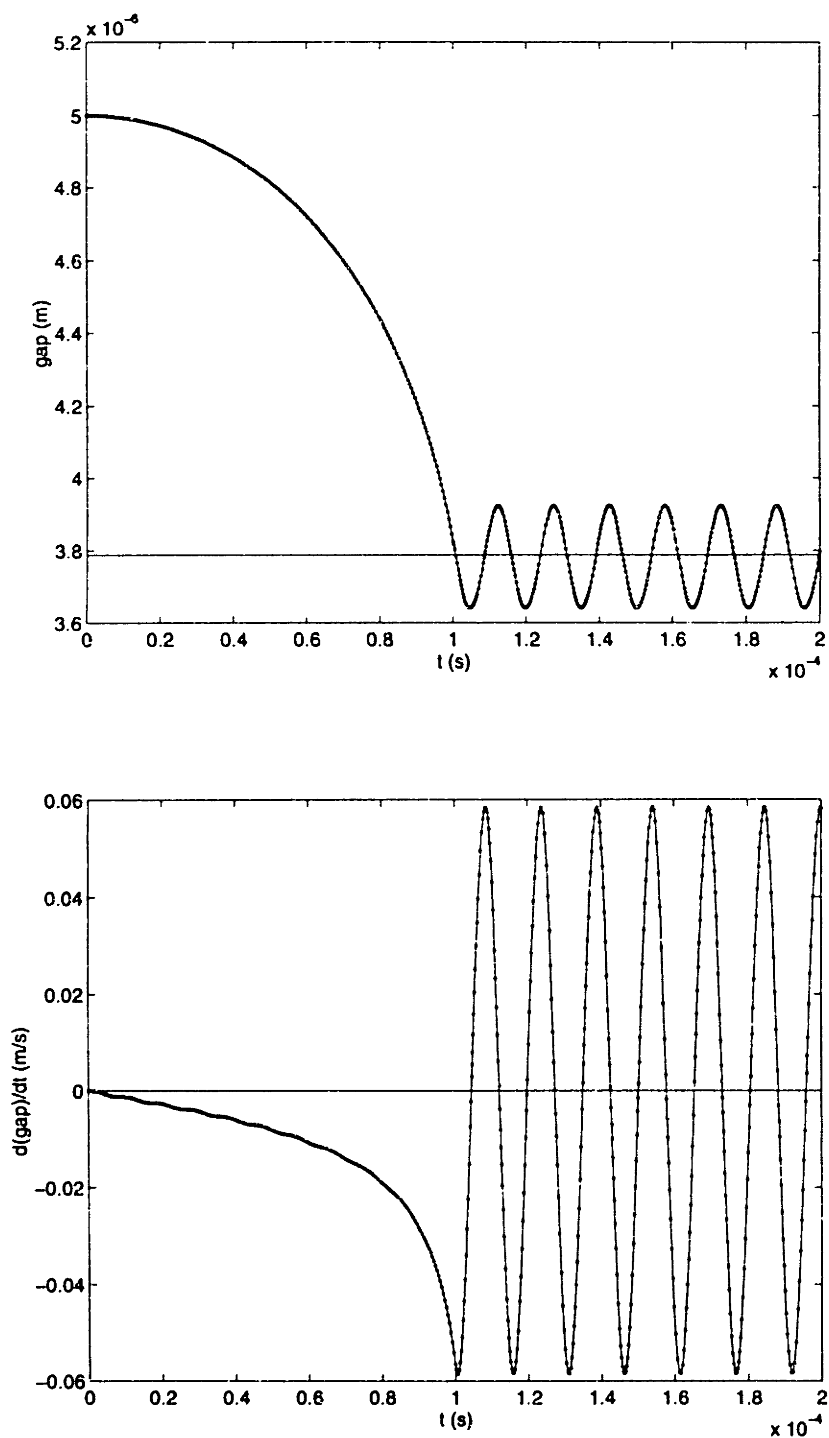


Figure 41: Voltage Ramp Dynamics - 2D Structure - *g*

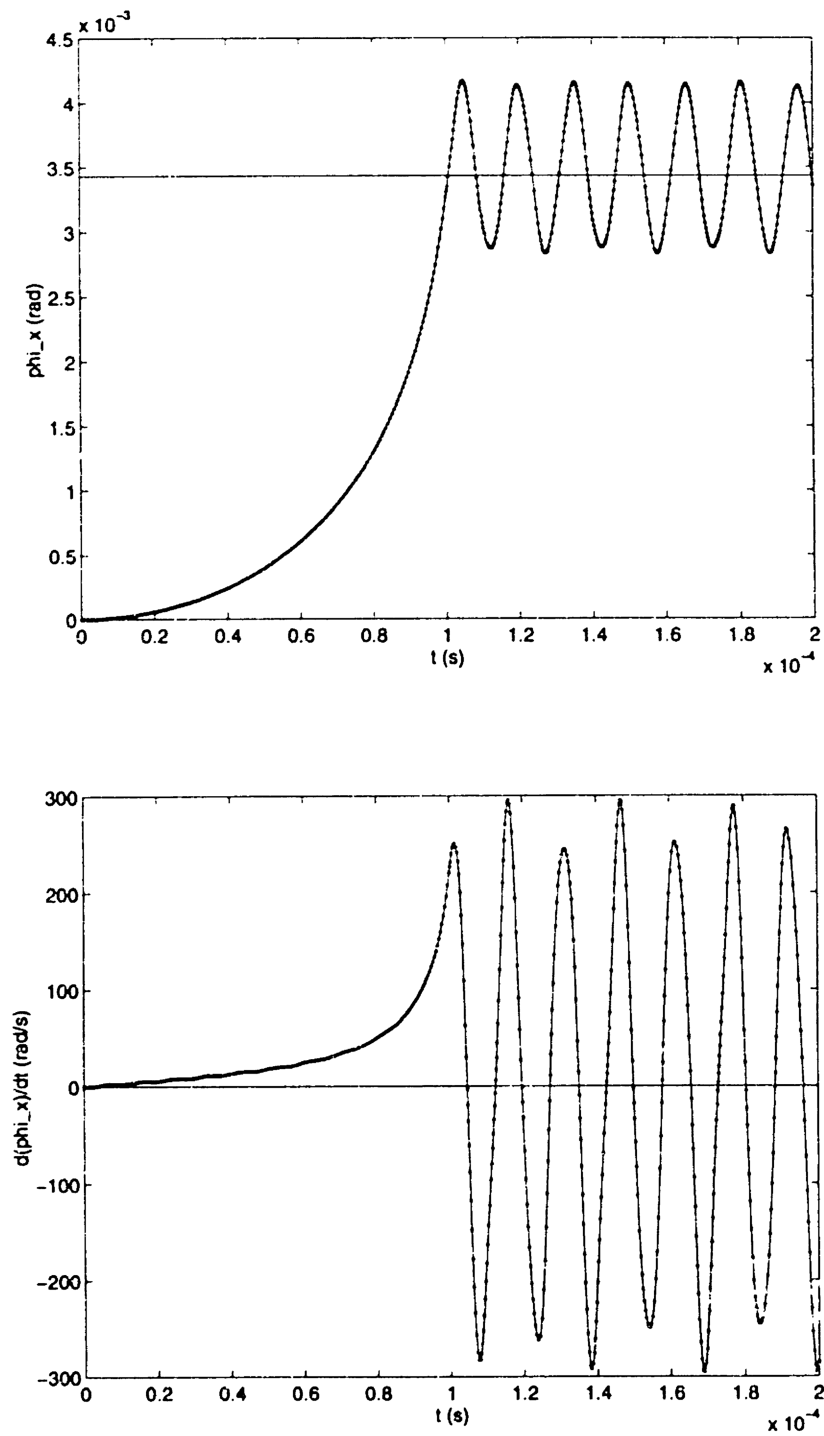
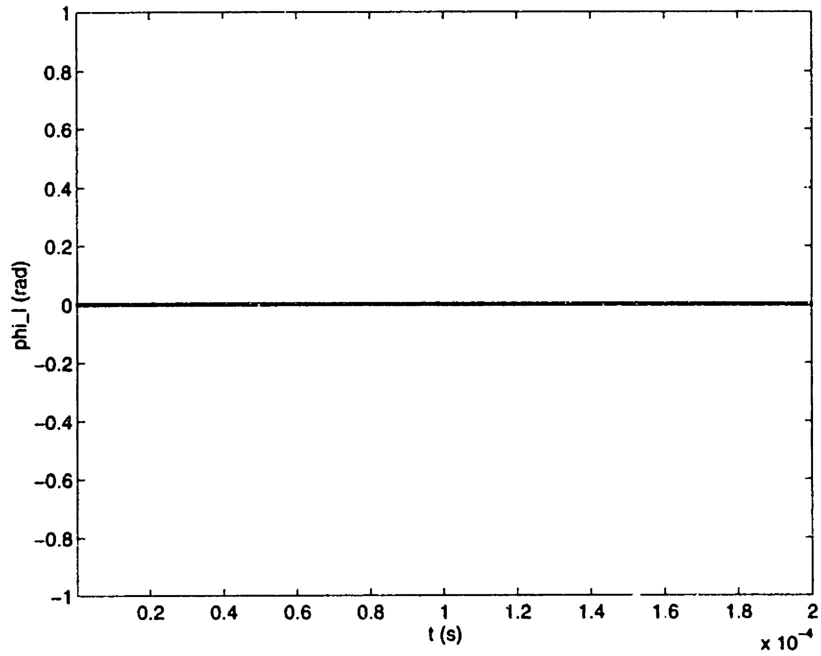


Figure 42: Voltage Ramp Dynamics - 2D Structure - φ_x



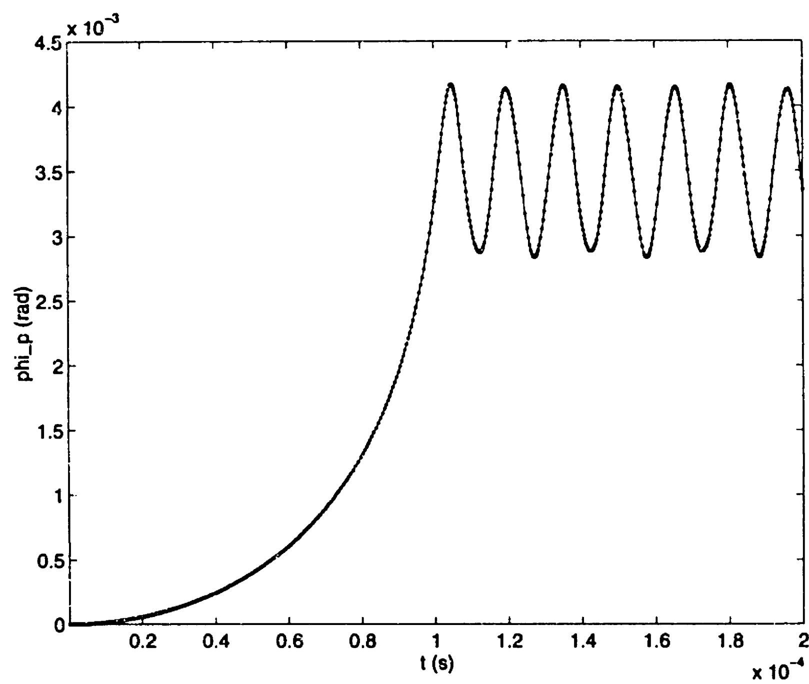
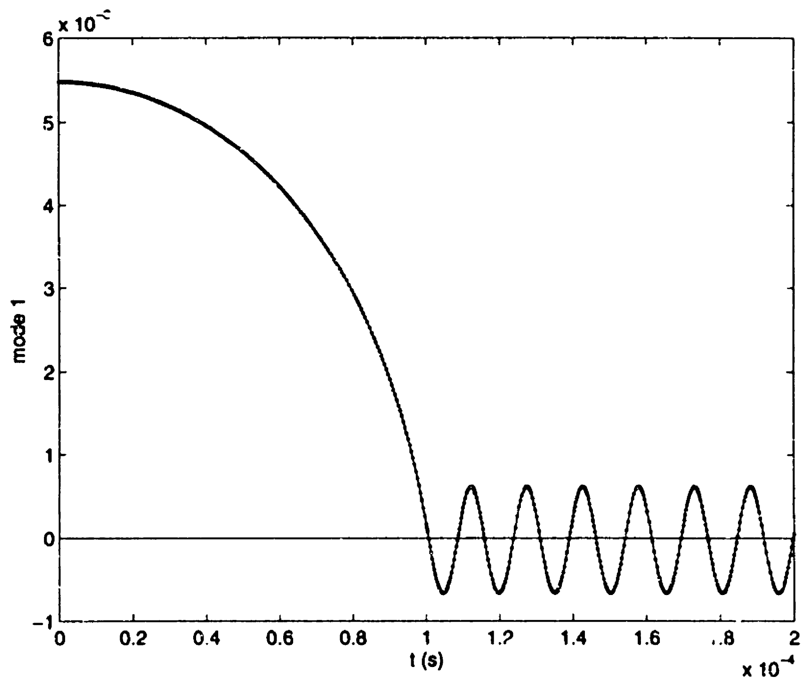


Figure 43: Voltage Ramp Dynamics - 2D Structure - Principle Tilt



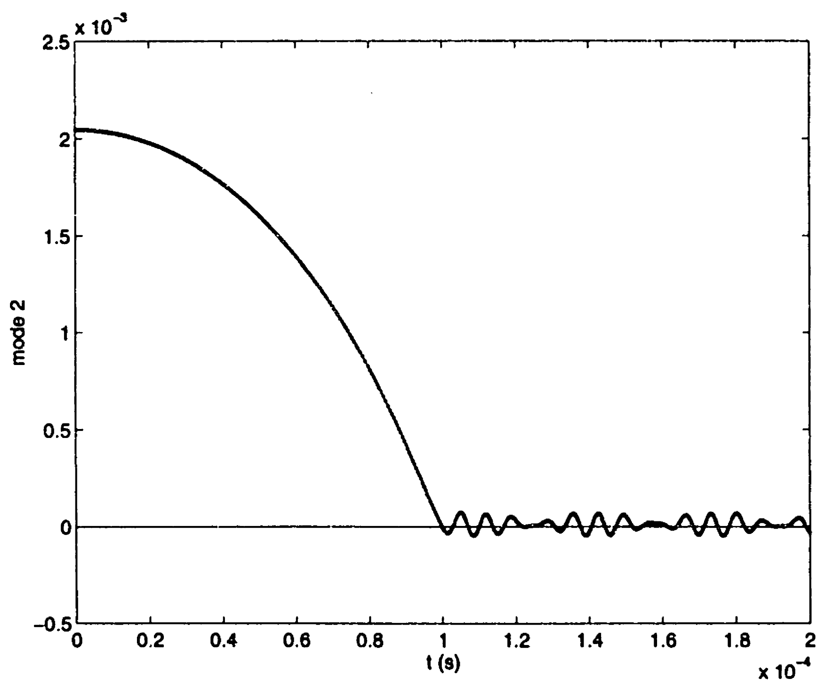


Figure 44: Voltage Ramp Dynamics - 2D Structure - Normal Modes

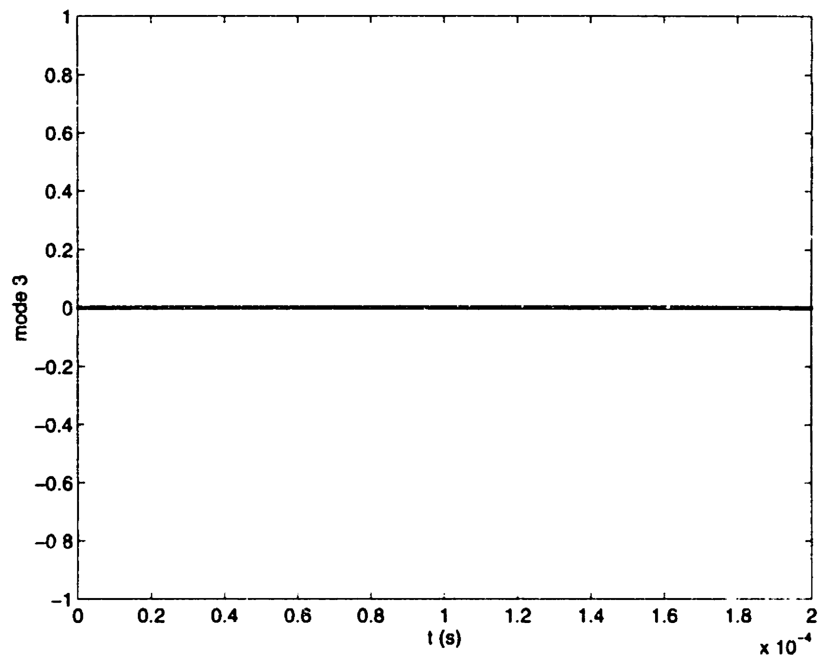


Figure 44 (cont): Voltage Ramp Dynamics - 2D Structure - Normal Modes

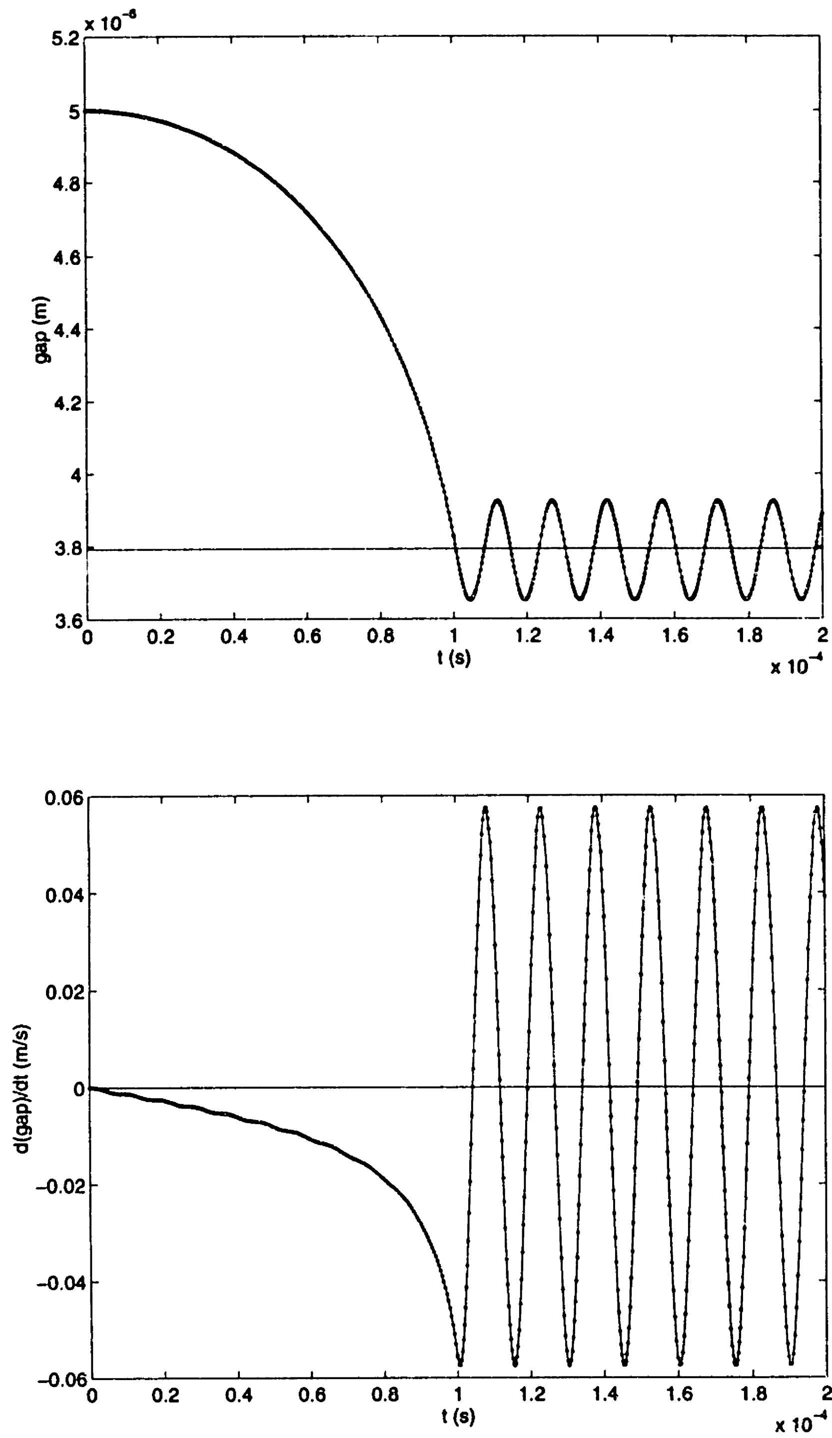


Figure 45: Voltage Ramp Dynamics - 2D+ Structure - *g*

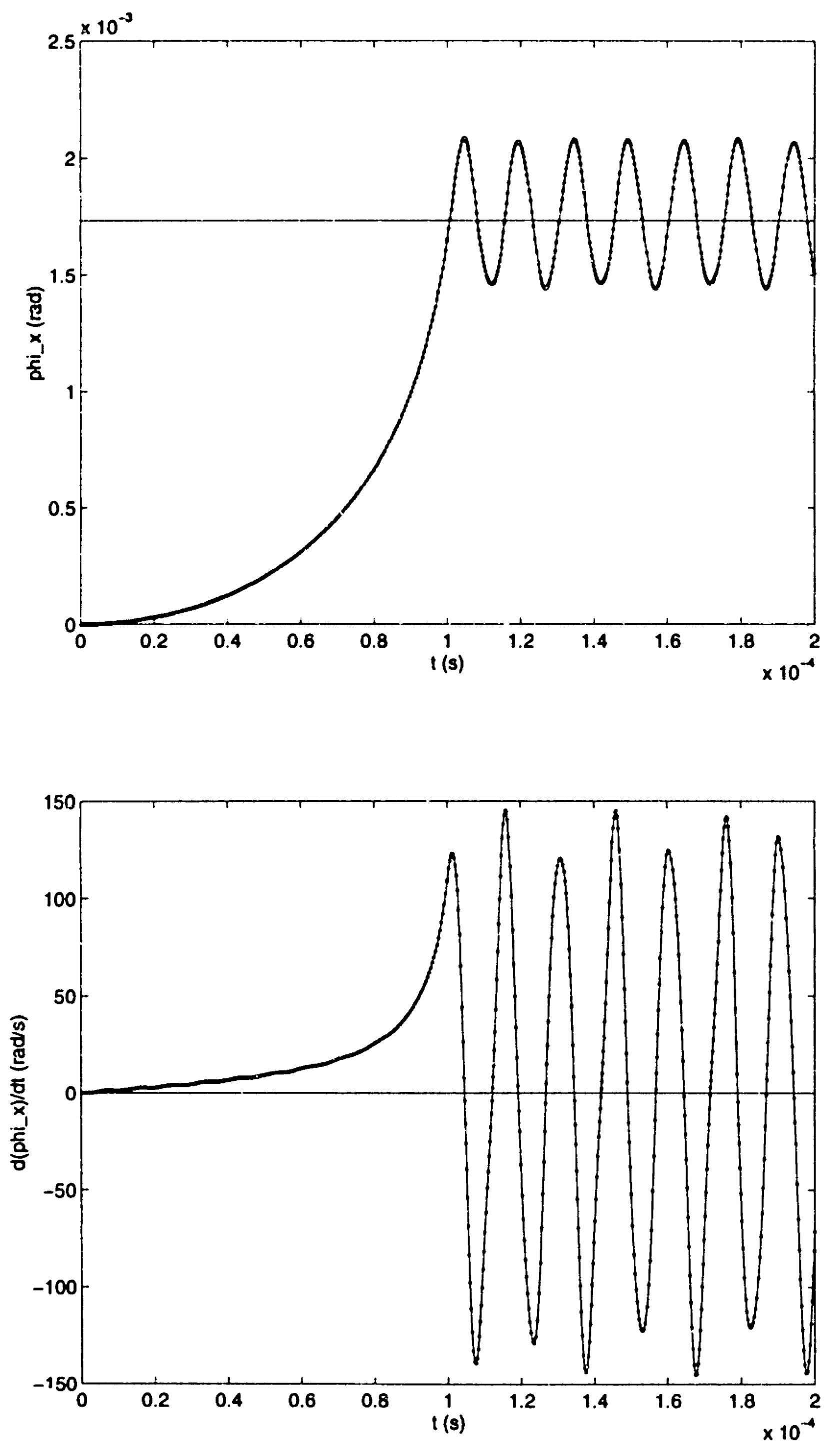


Figure 46: Voltage Ramp Dynamics - 2D+ Structure - φ_x

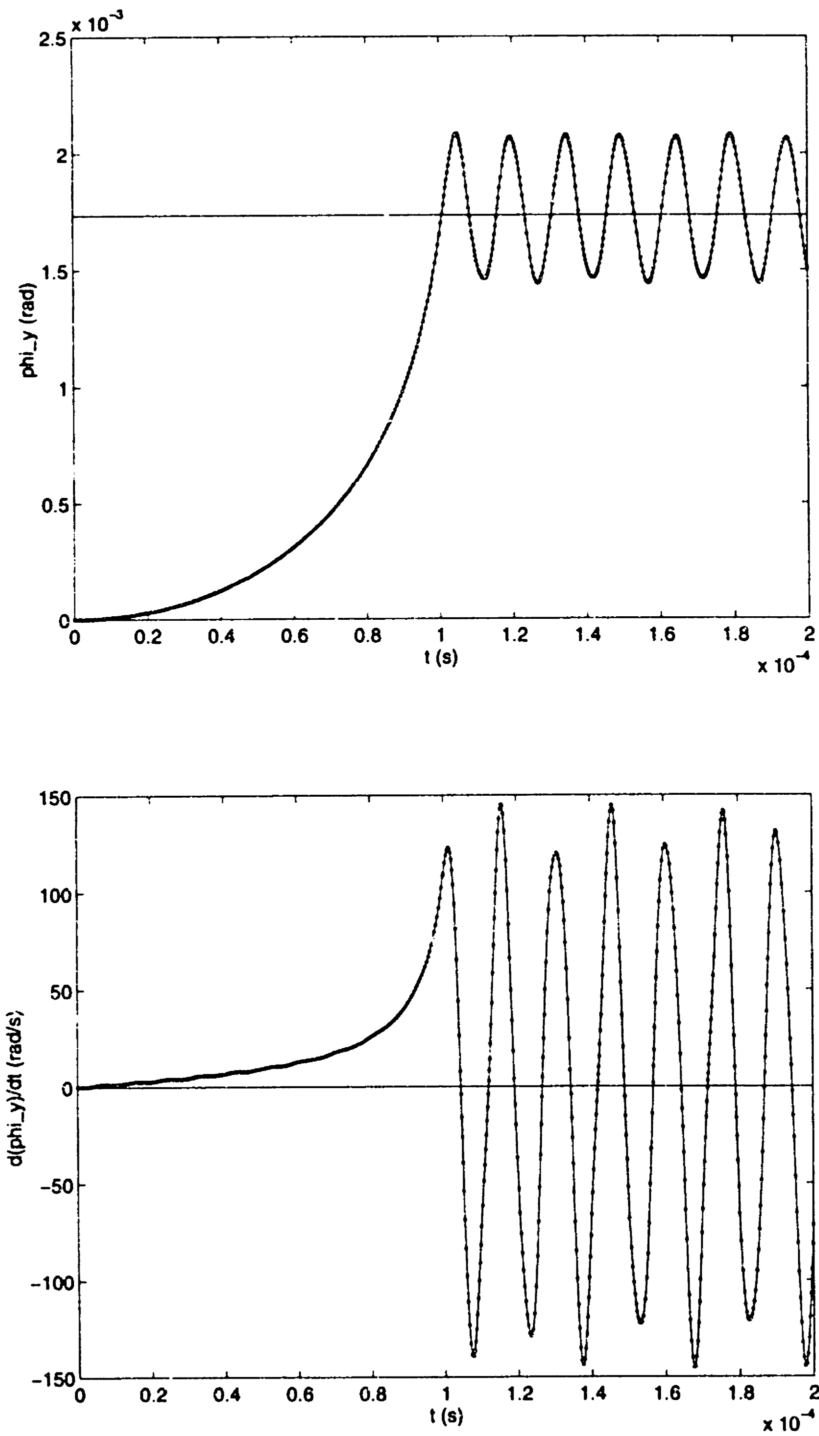


Figure 47: Voltage Ramp Dynamics - 2D+ Structure - φ_y

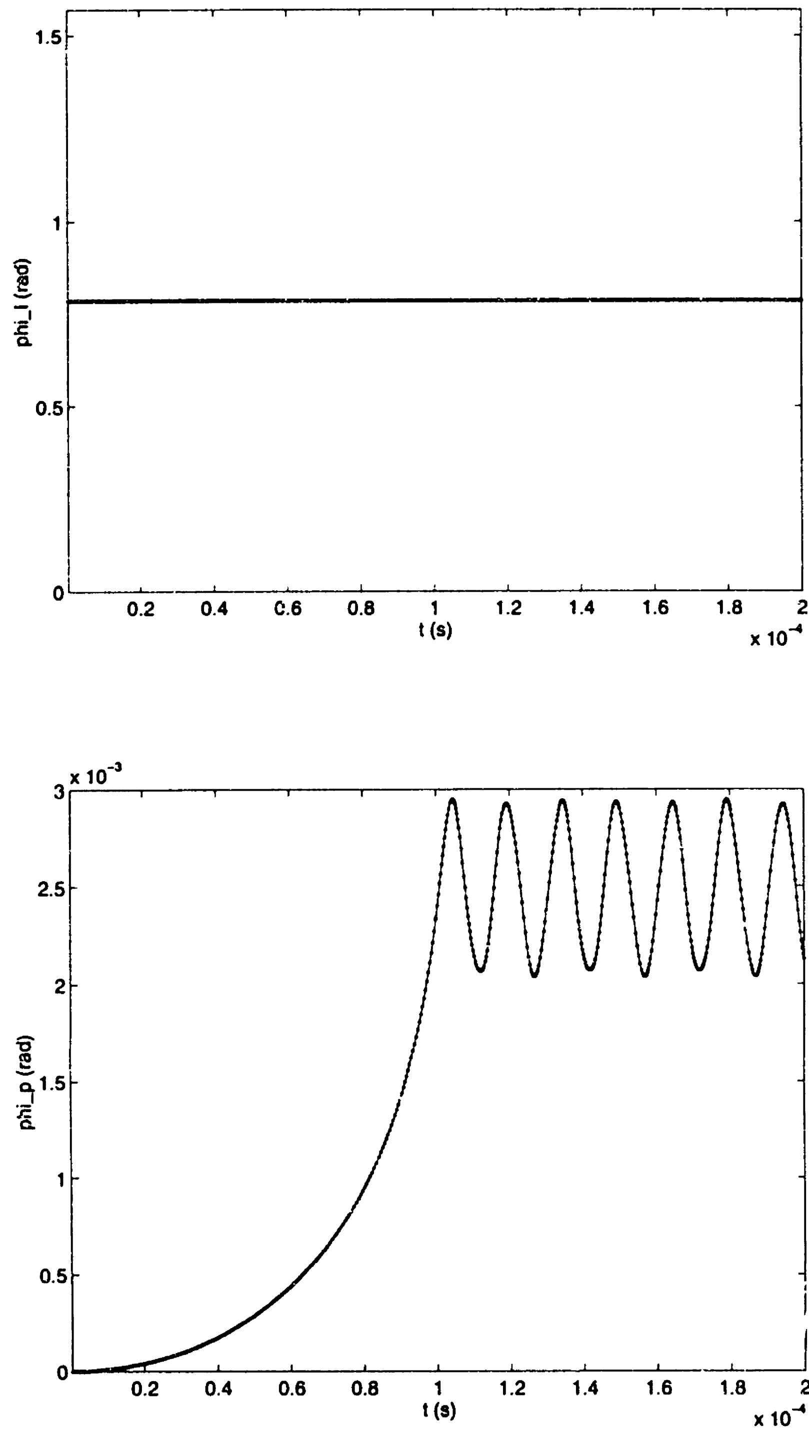


Figure 48: Voltage Ramp Dynamics - 2D+ Structure - Principle Tilt

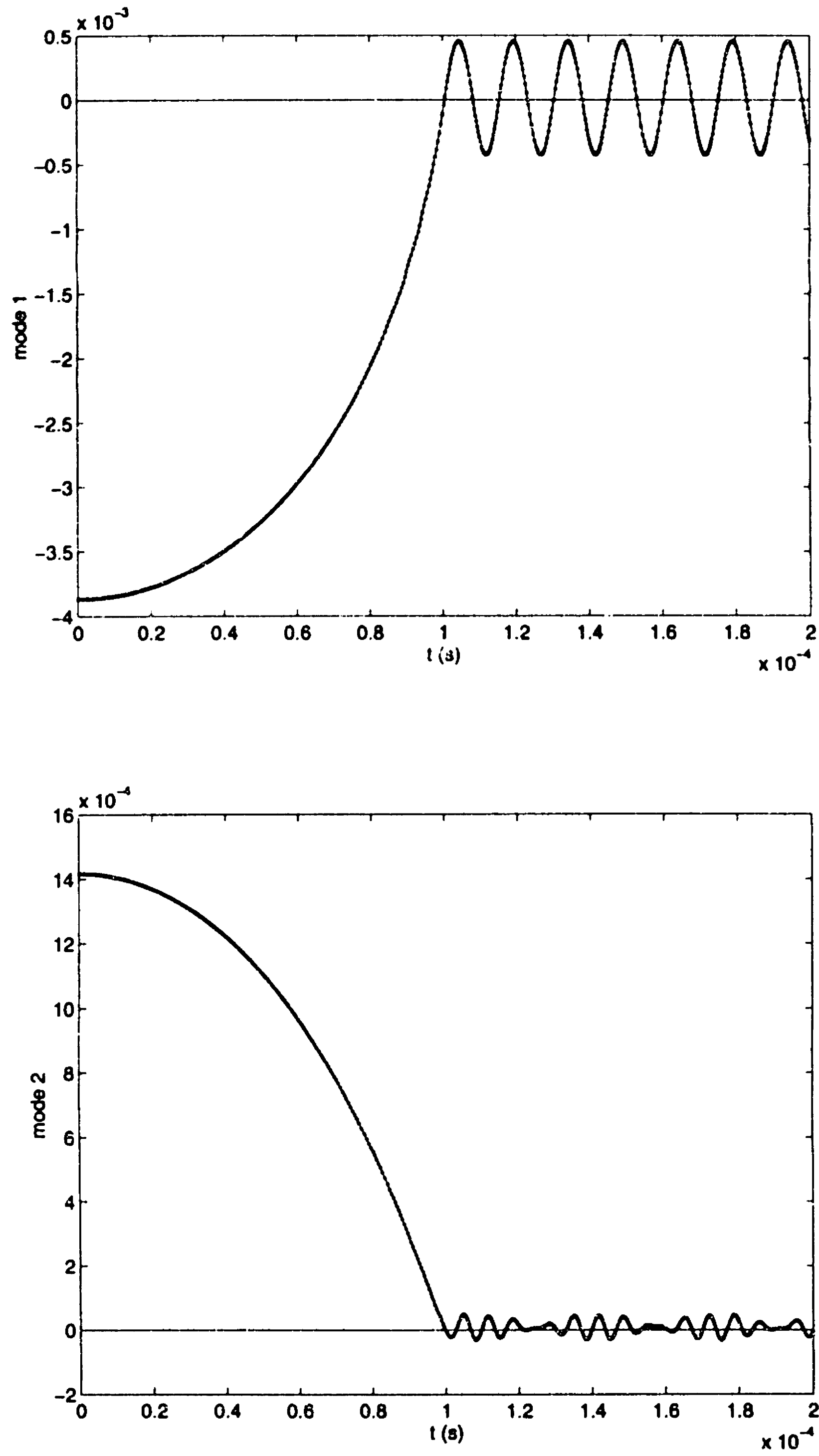


Figure 49: Voltage Ramp Dynamics - 2D+ Structure - Normal Modes

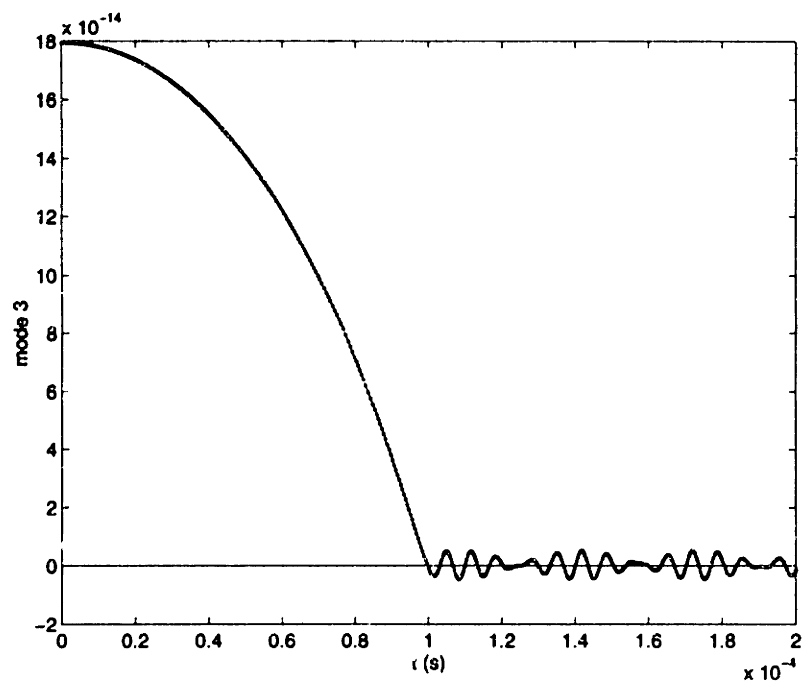


Figure 49 (cont): Voltage Ramp Dynamics - 2D+ Structure - Normal Modes

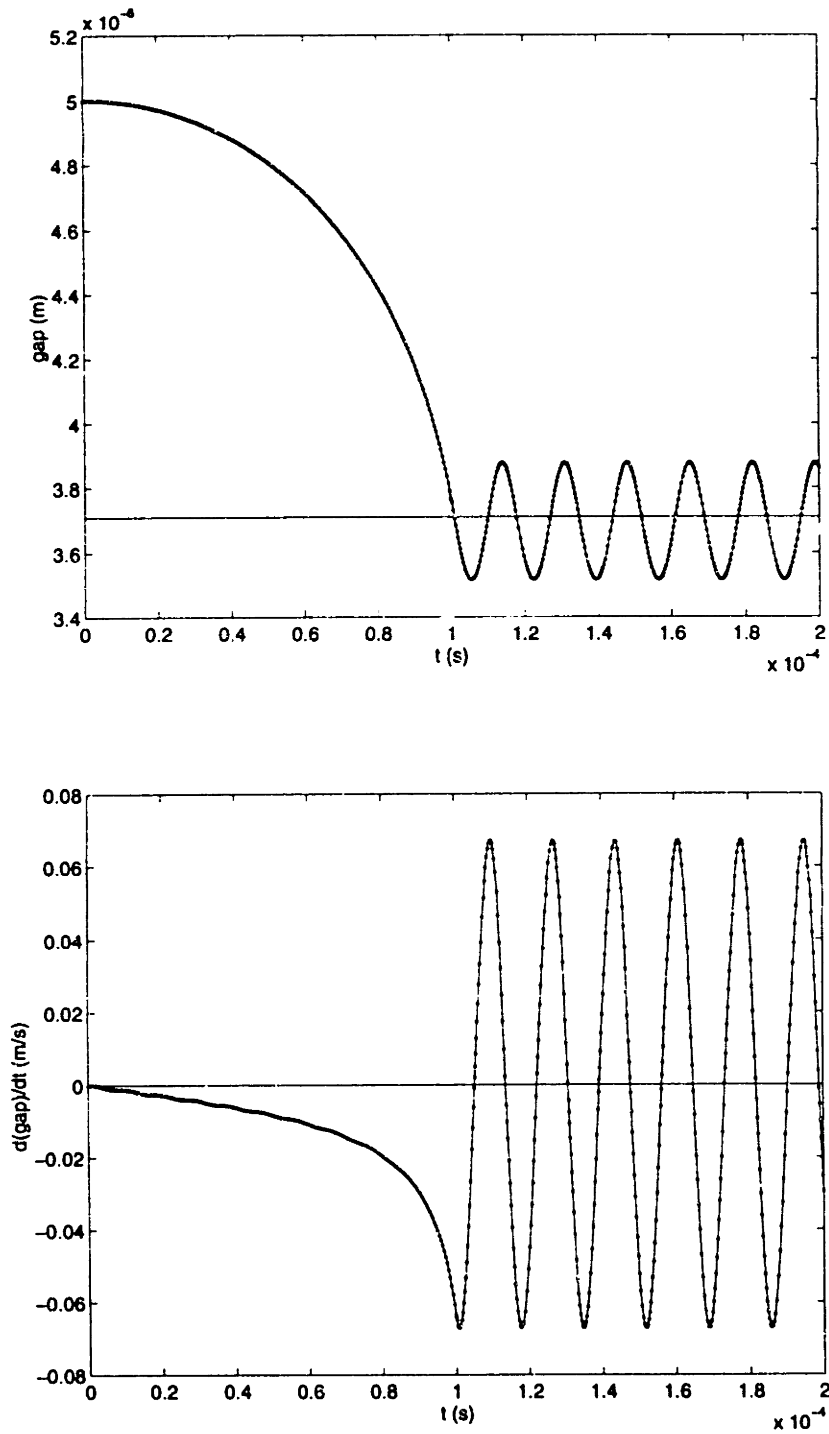


Figure 50: Voltage Ramp Dynamics - 3D Structure - *g*

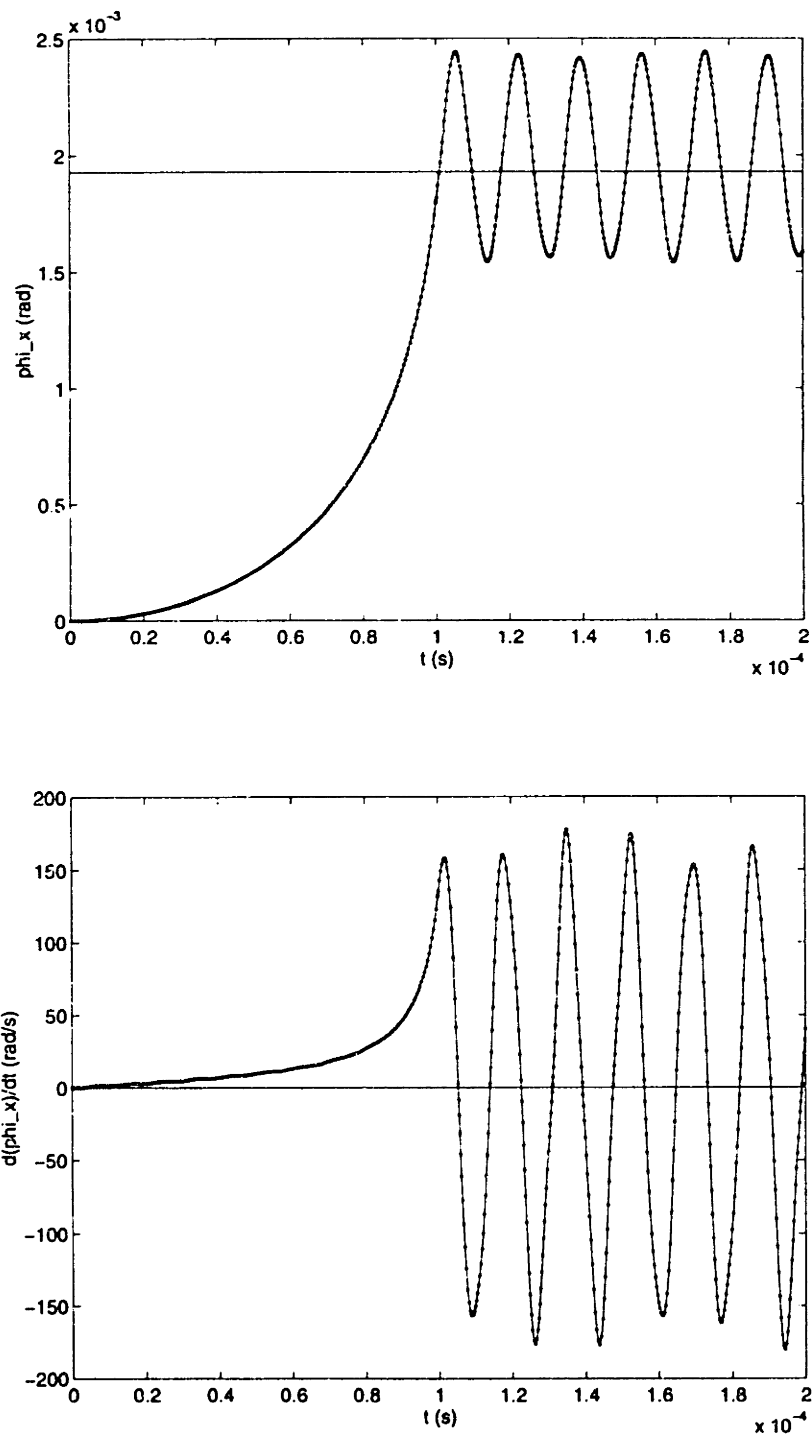


Figure 51: Voltage Ramp Dynamics - 3D Structure - φ_x

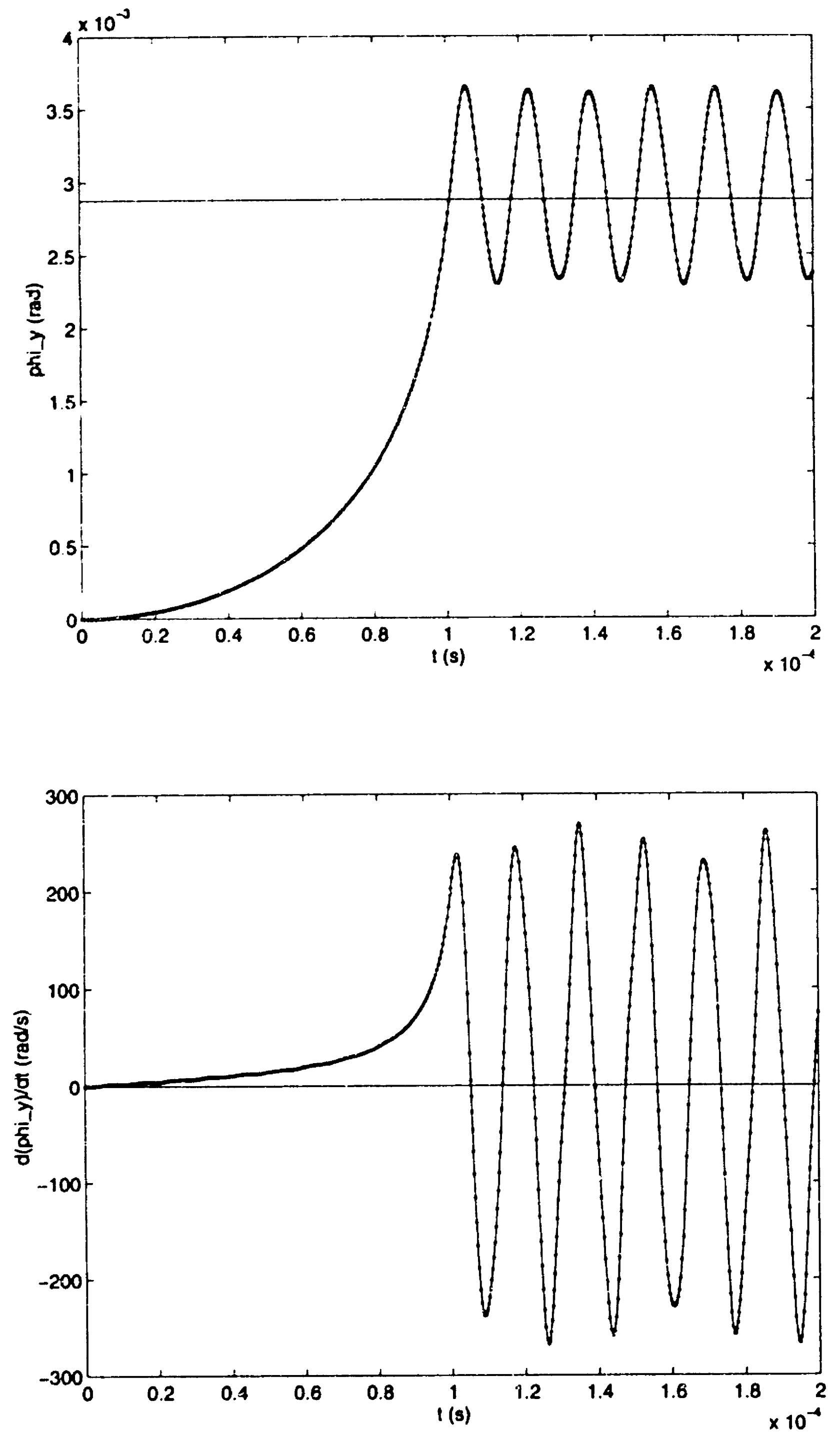
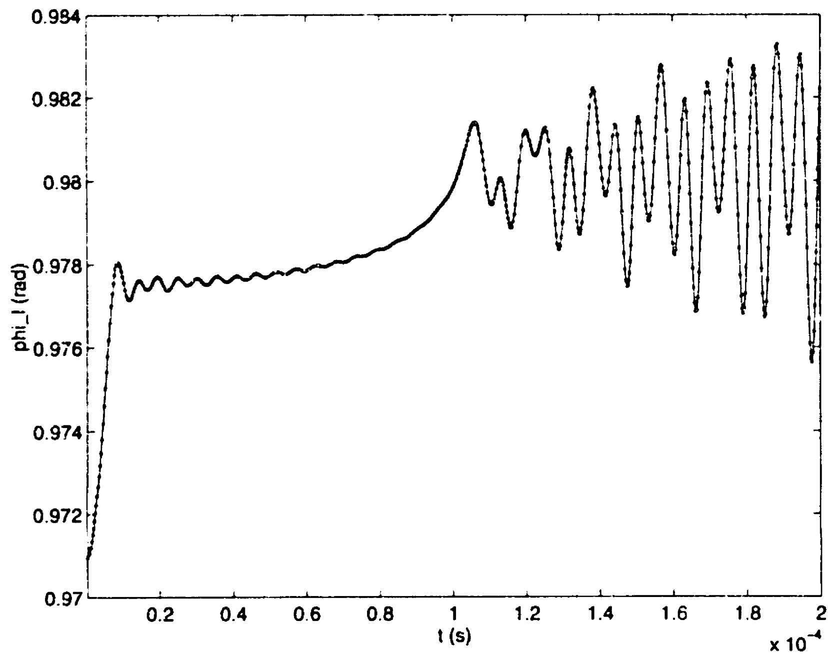


Figure 52: Voltage Ramp Dynamics - 3D Structure - φ_y



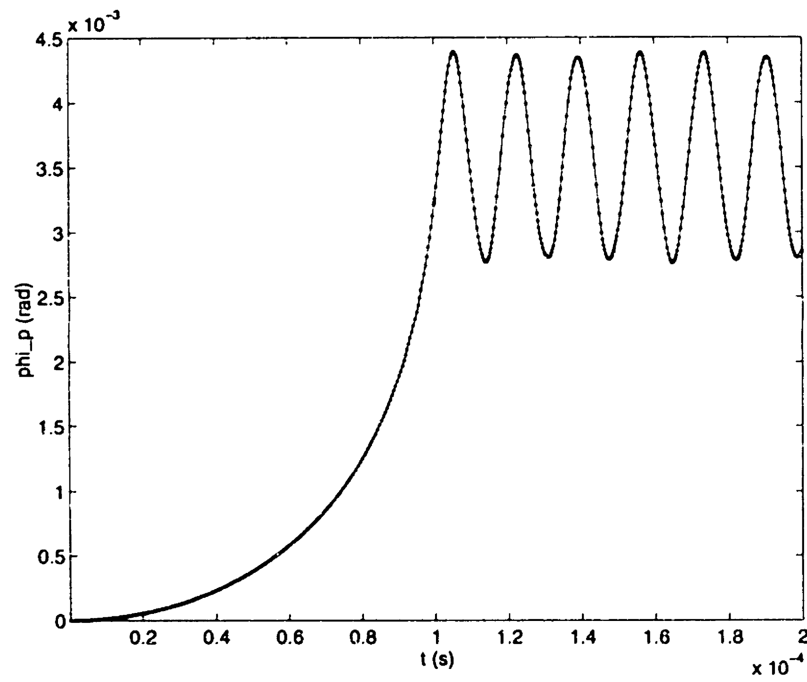


Figure 53: Voltage Ramp Dynamics - 3D Structure - Principle Tilt

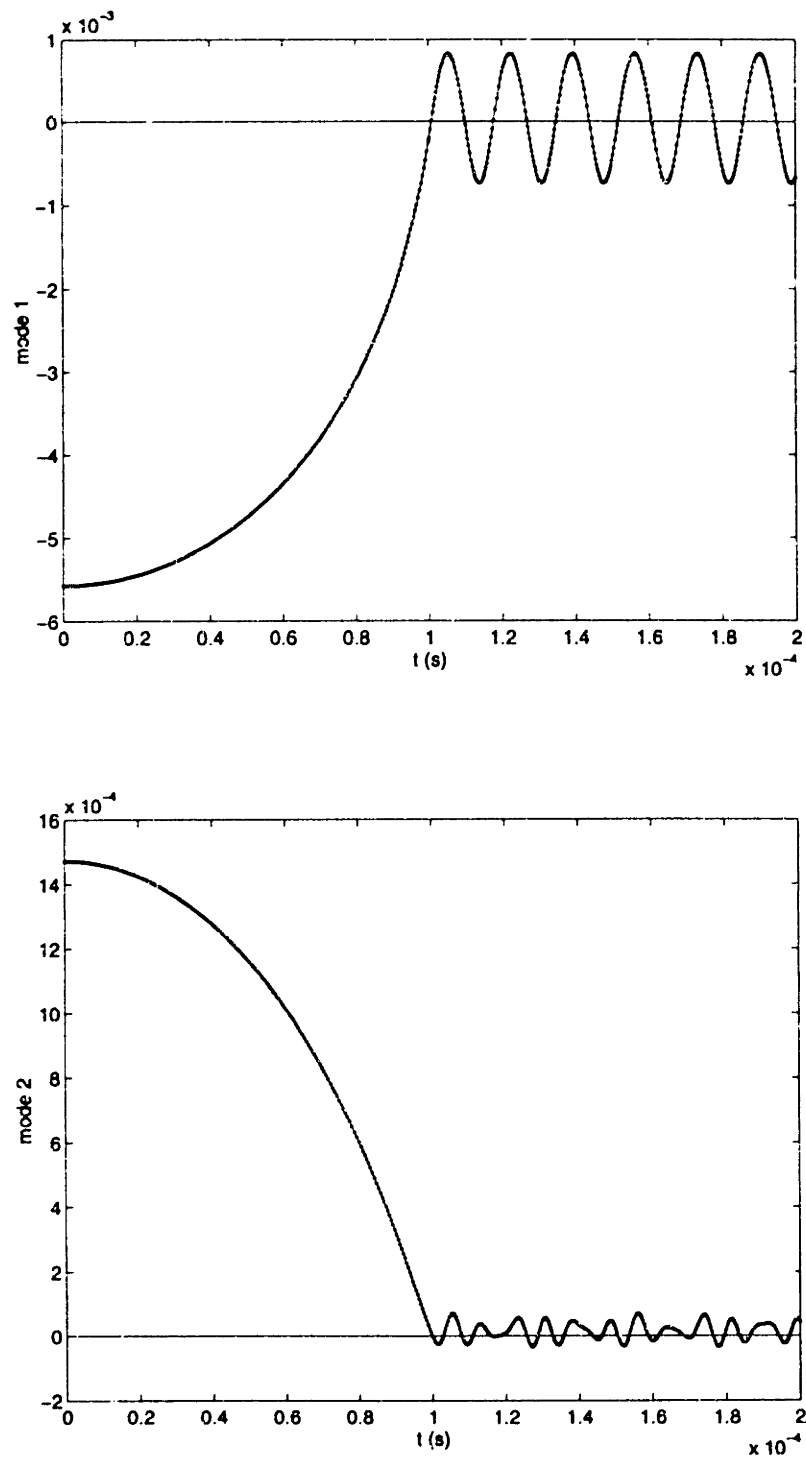


Figure 54: Voltage Ramp Dynamics - 3D Structure - Normal Modes

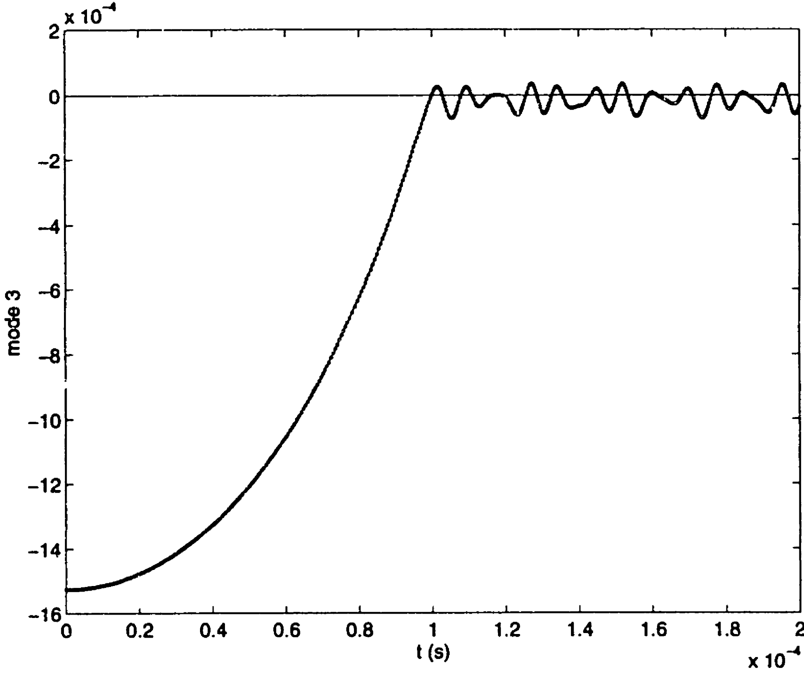


Figure 54 (cont): Voltage Ramp Dynamics - 3D Structure - Normal Modes

B.2.3 Varied Voltage Ramp Rate

The preceding voltage ramp situation demonstrates that momentum of the plate build as voltage is ramped, causing the plate to overshoot equilibrium and conceivably past pull-in. It follows that the rate at which voltage is ramped affects this overshoot. The third experiment repeats the 3D structure voltage ramp experiment, using ramp time values of 10^{-3} and 10^{-5} seconds. The results of these simulations are given in Figures 55 through 64. Notice that the 10^{-5} second ramp results in pull-in, yet the 10^{-3} second ramp brings the plate very close to equilibrium and only small oscillations occur. This implies that momentum has a significant effect upon the success of moving an electrostatically actuated suspended plate from one equilibrium position to another. The rate of voltage ramping must be chosen so as not to pull the plate past pull-in.

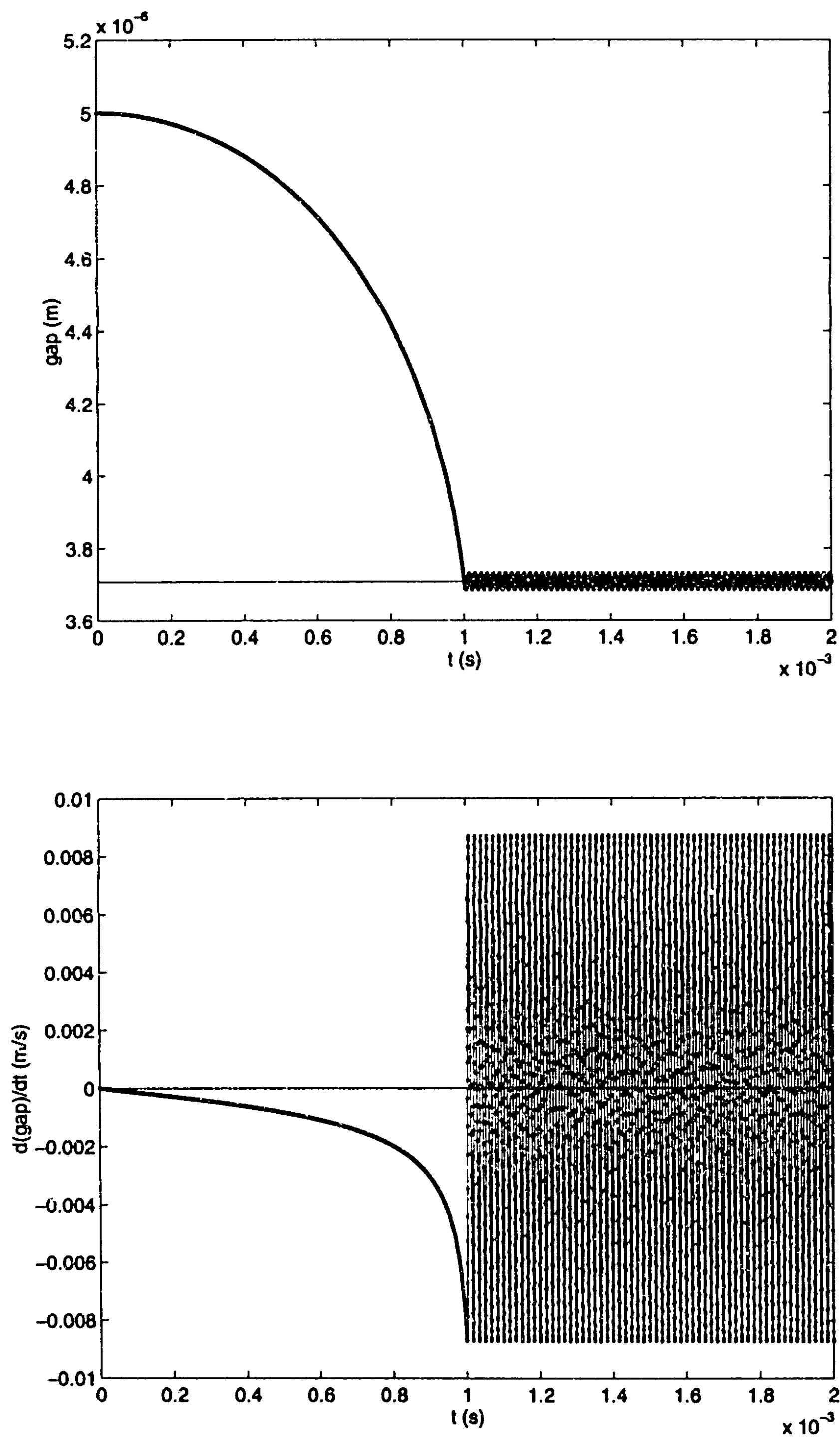
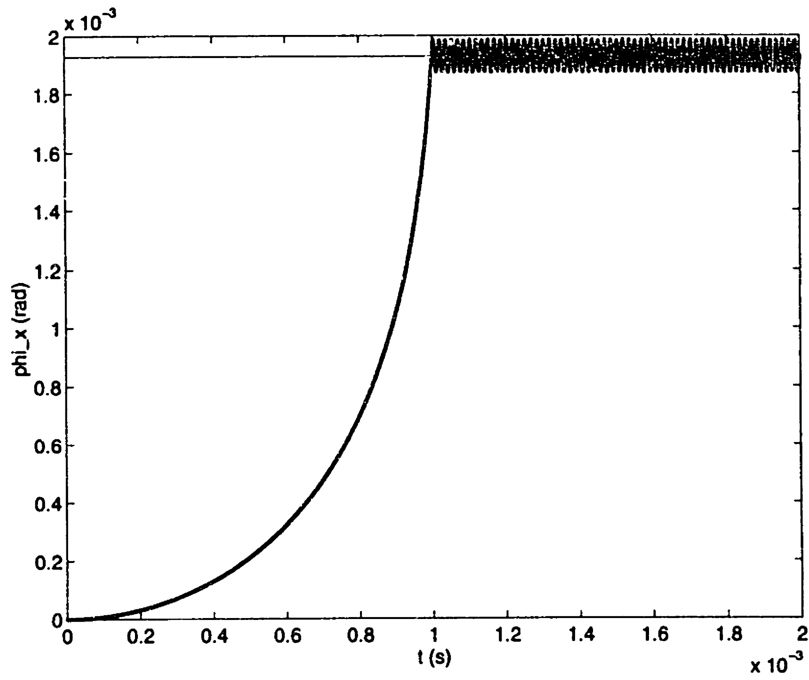


Figure 55: Ramp Time 10^{-3} - g



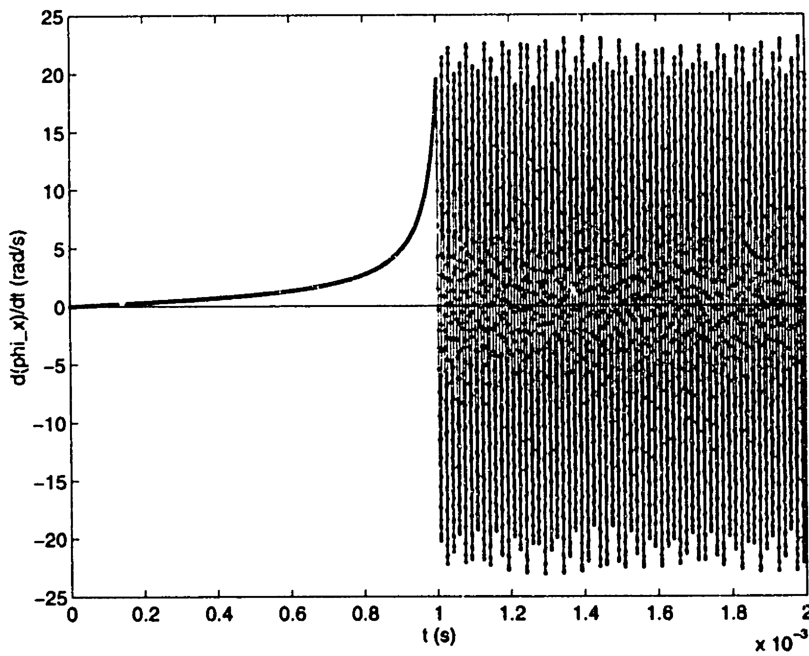
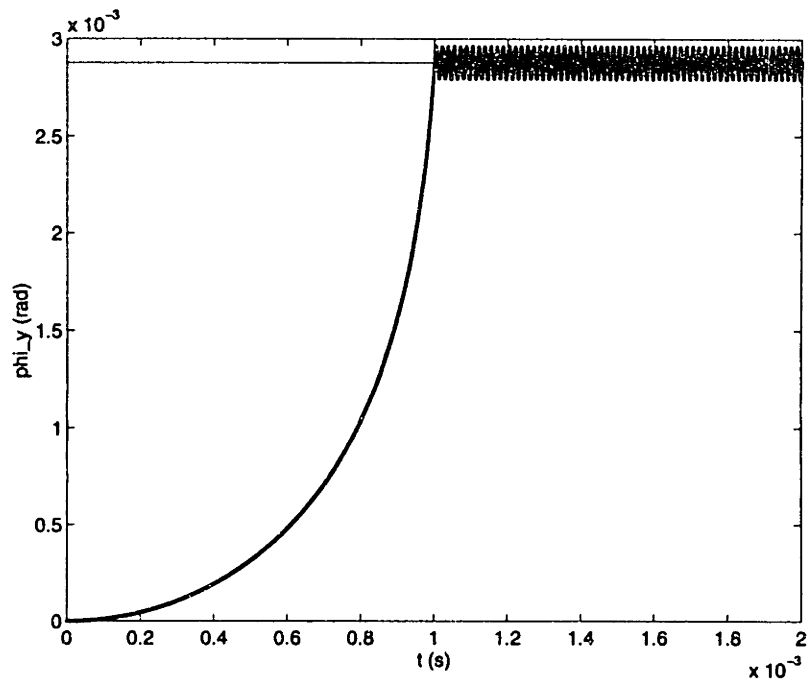


Figure 56: Ramp Time 10^{-3} - φ_x



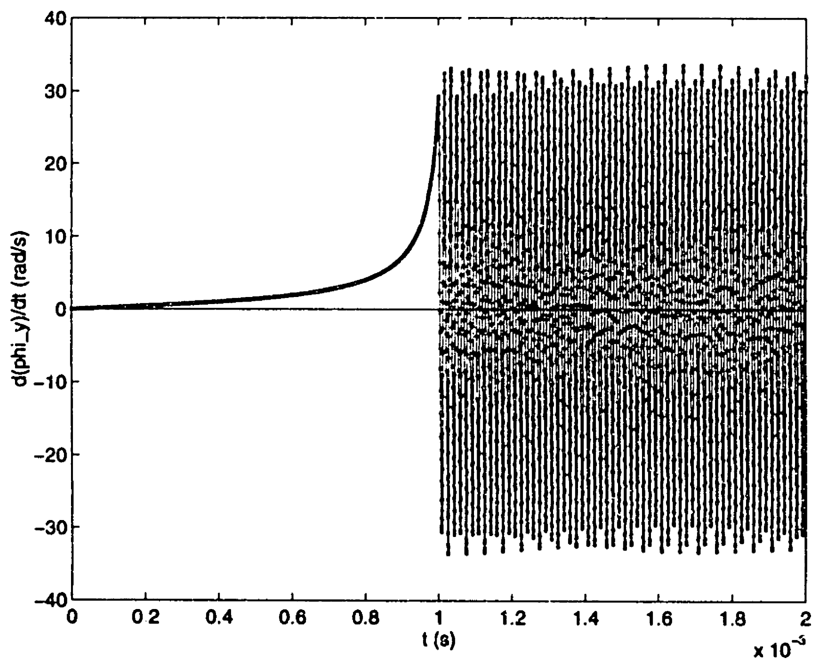
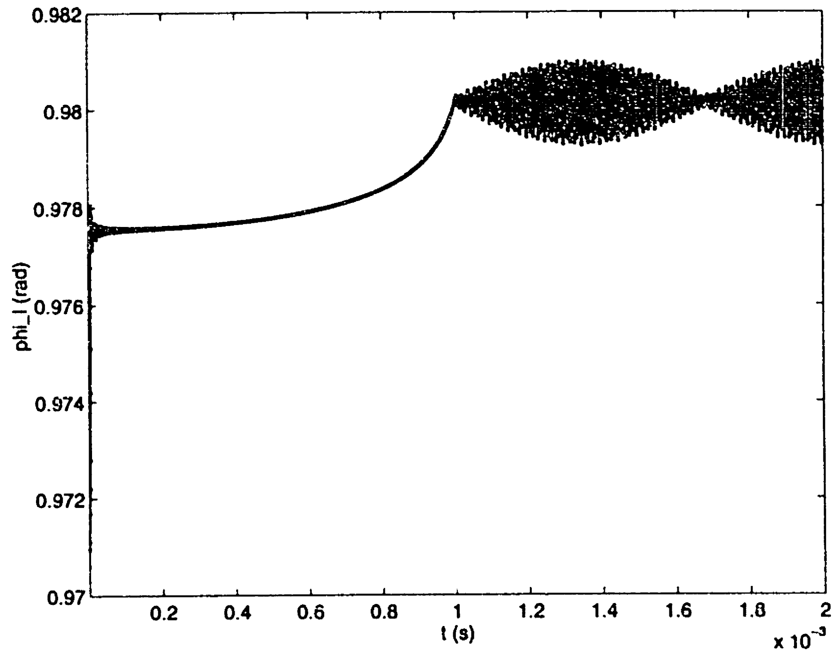


Figure 57: Ramp Time 10^{-3} - φ_y



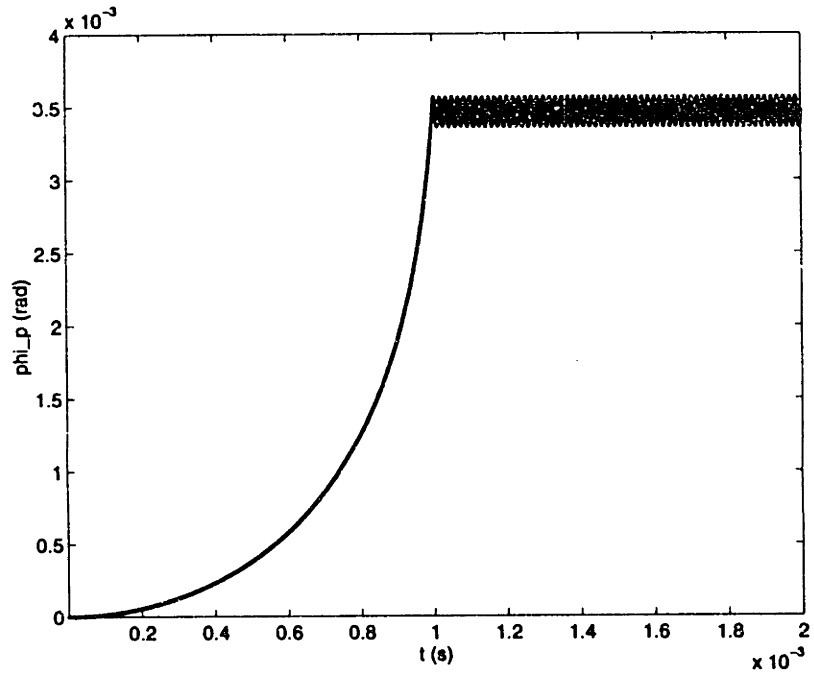
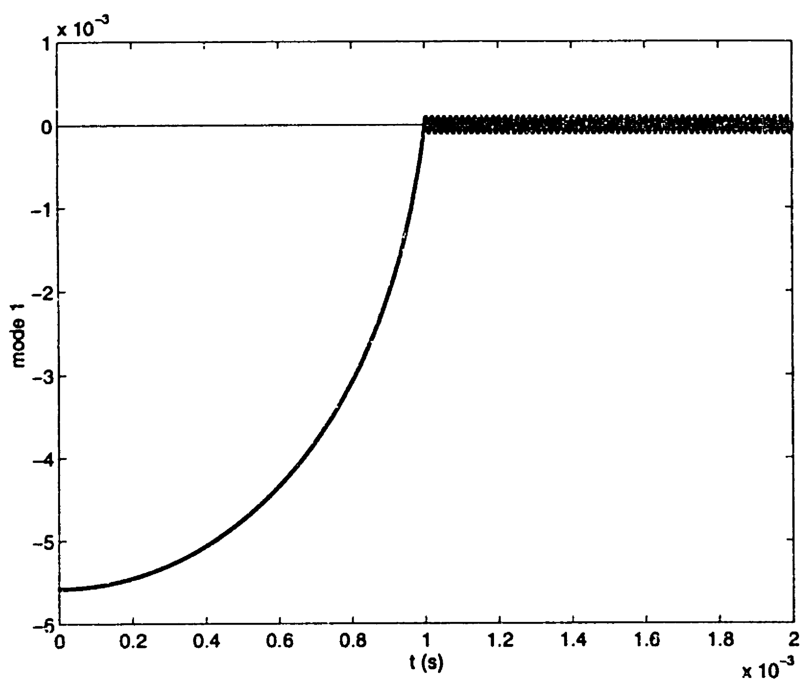


Figure 58: Ramp Time 10^{-3} - Principle Tilt



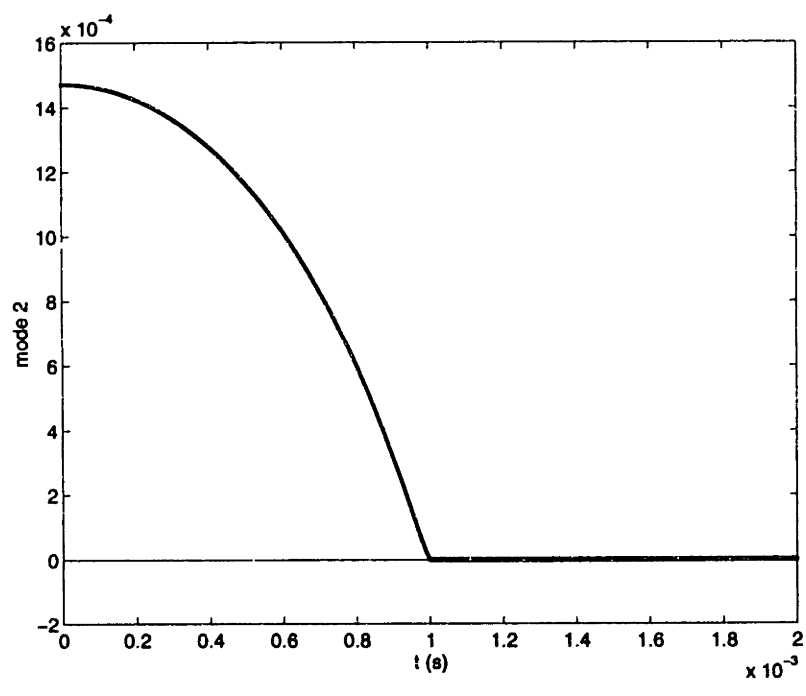


Figure 59: Ramp Time 10^{-3} - Normal Modes

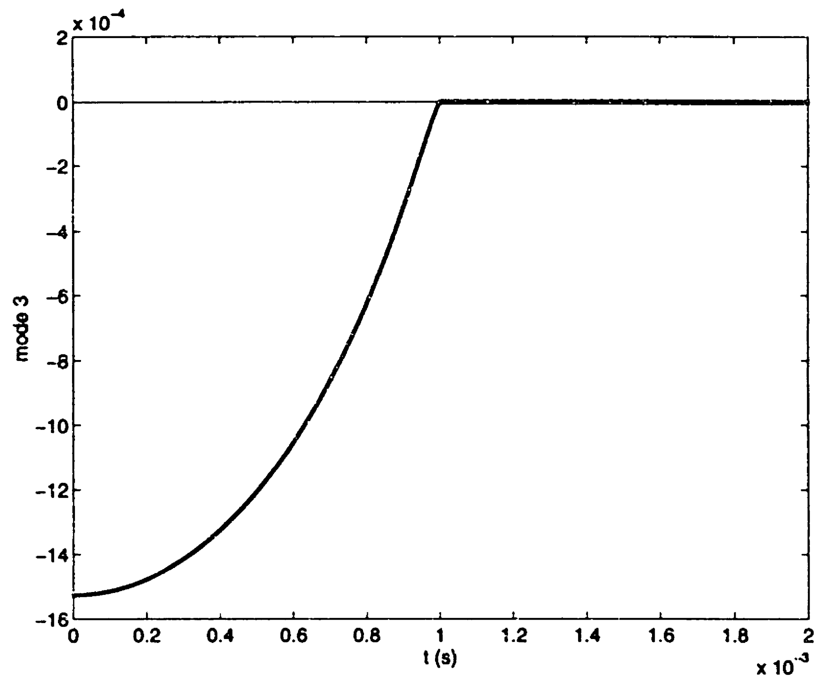


Figure 59 (cont): Ramp Time 10^{-3} - Normal Modes

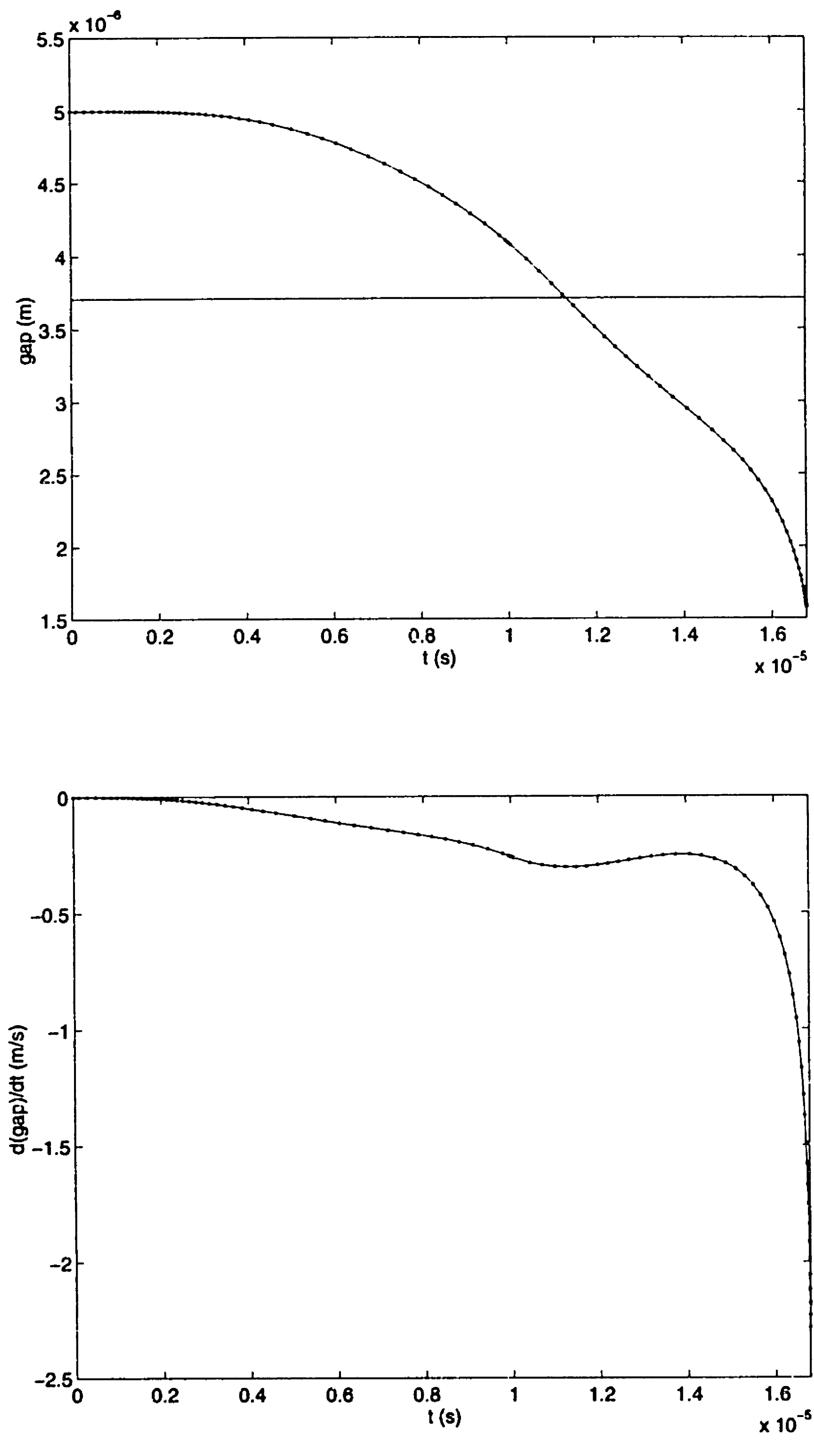
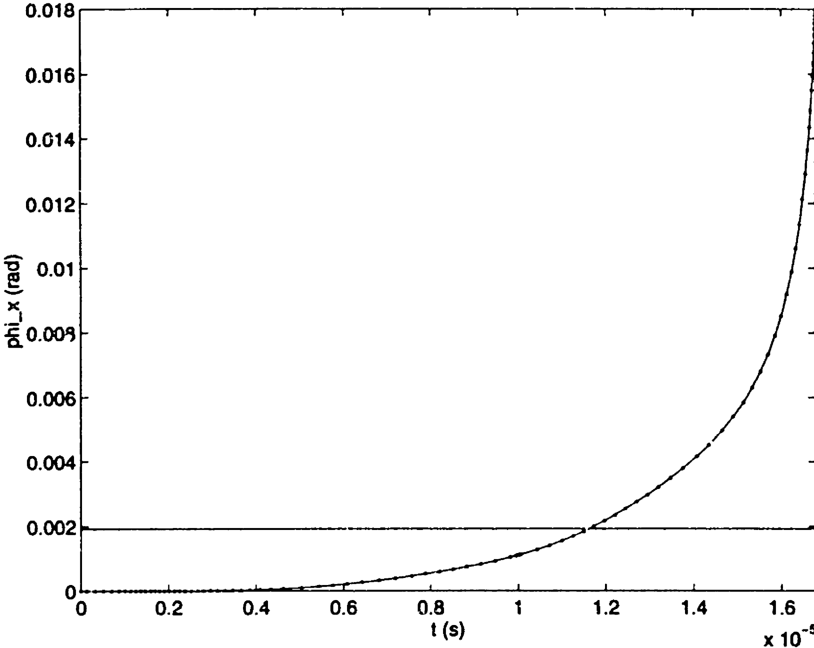


Figure 60: Ramp Time 10^{-5} - g



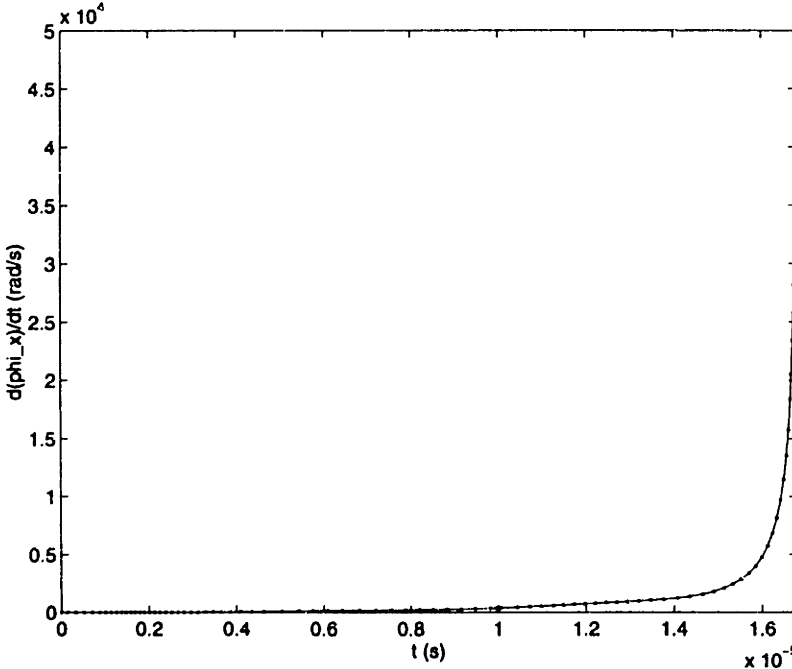
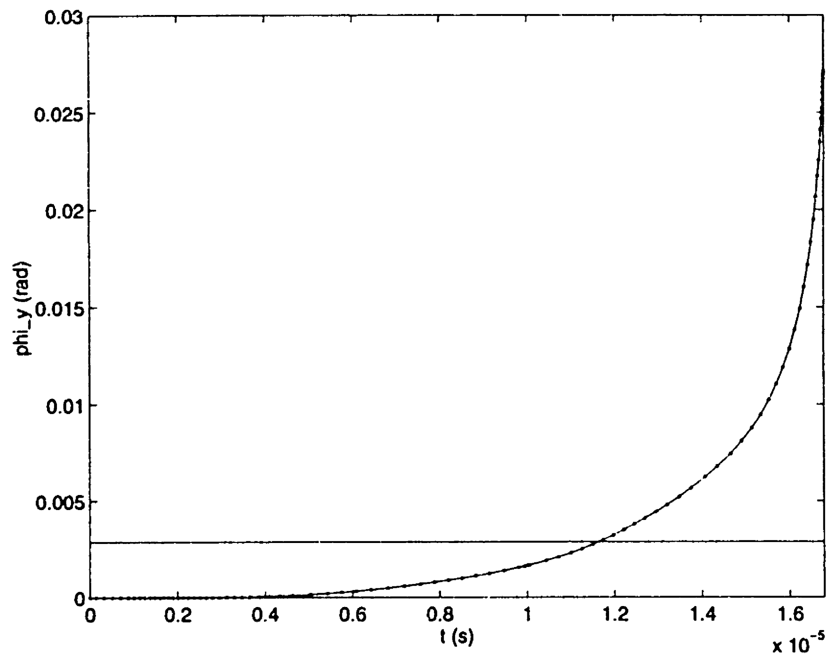


Figure 61: Ramp Time 10^{-5} - φ_x



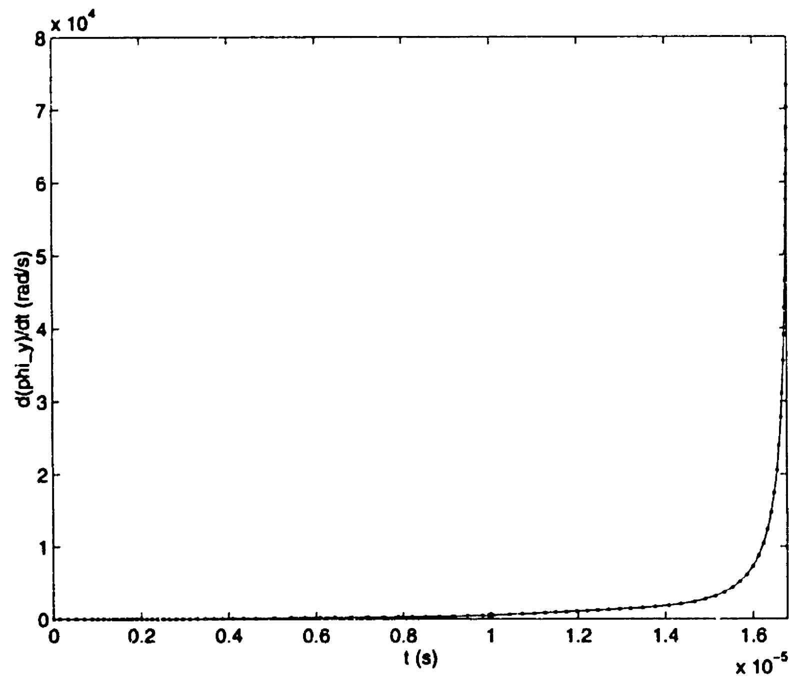
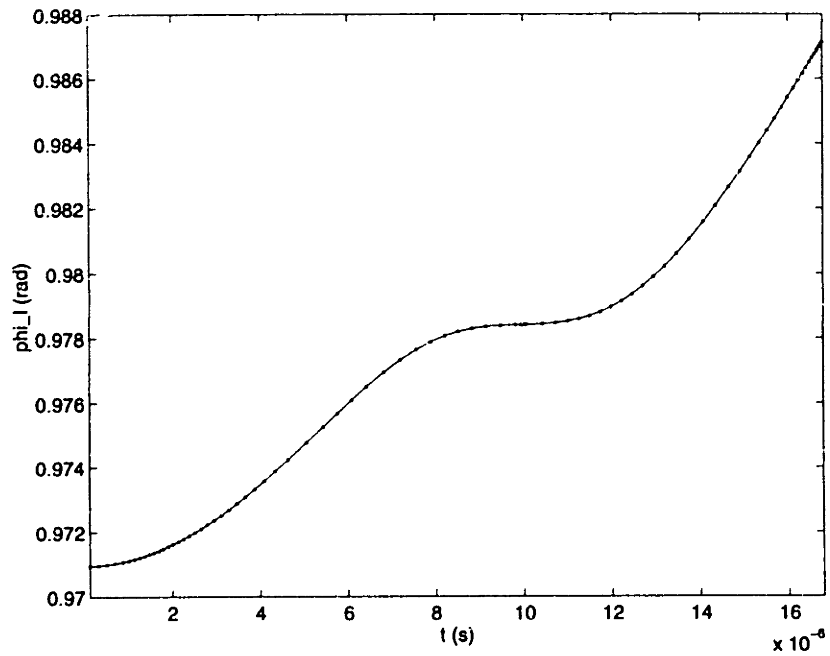


Figure 62: Ramp Time 10^{-5} - φ_y



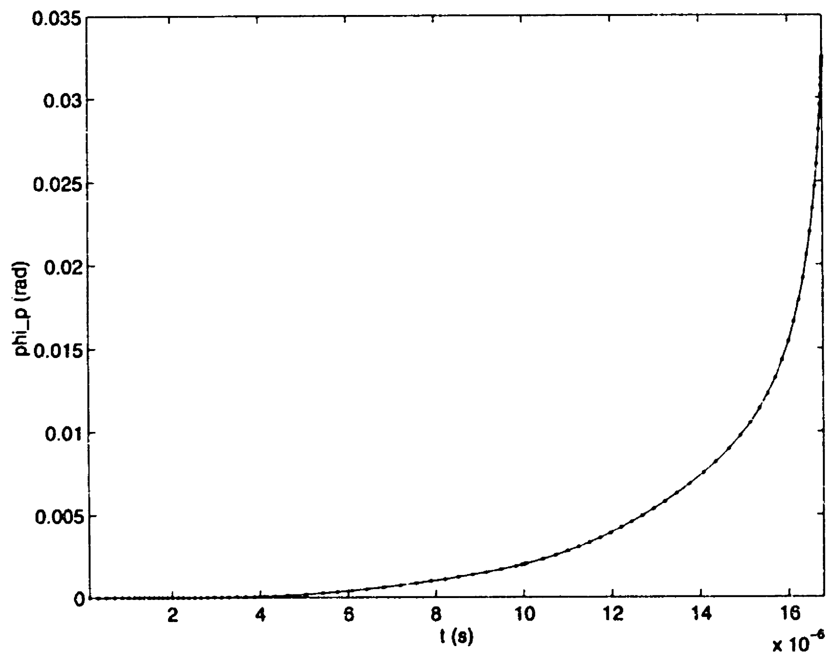
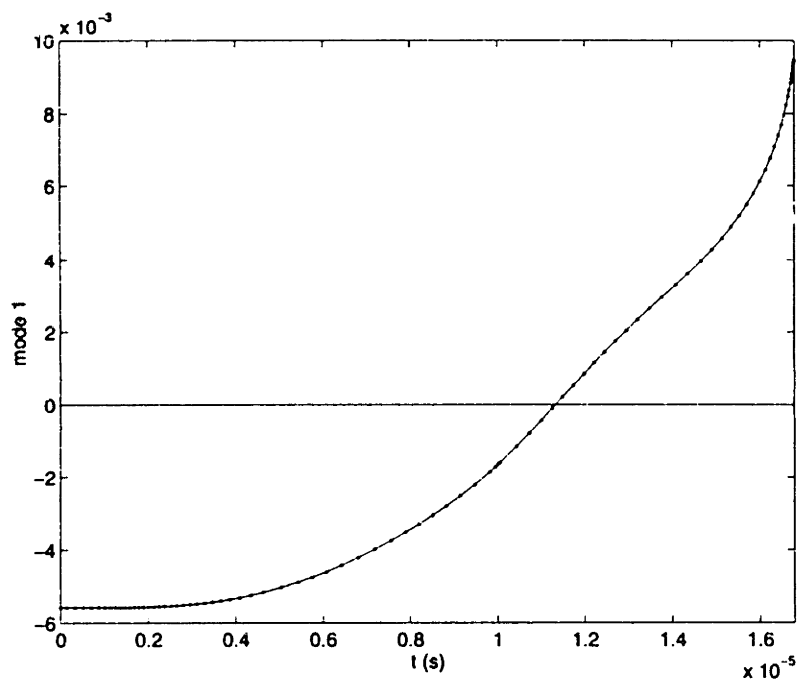


Figure 63: Ramp Time 10^{-5} - Principle Tilt



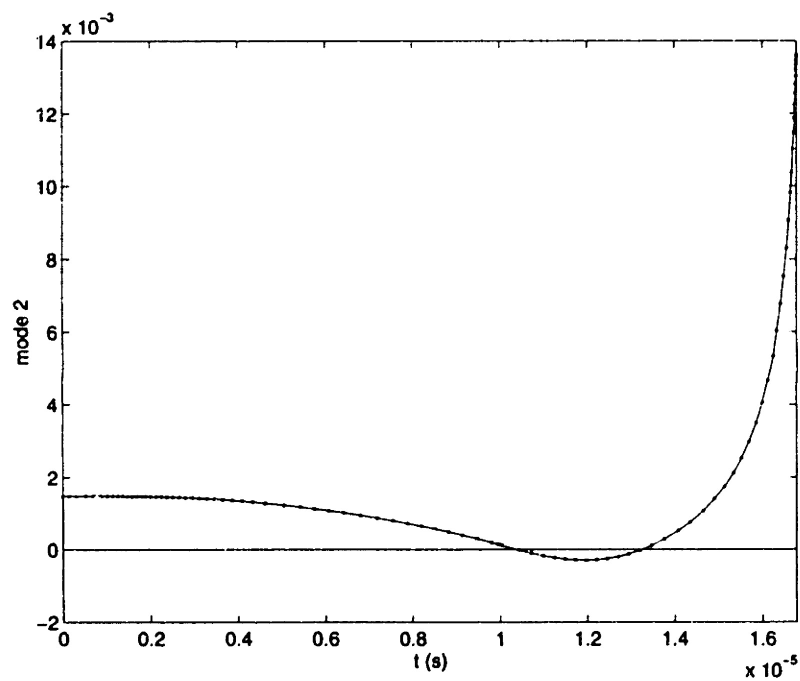


Figure 64: Ramp Time 10^{-5} - Normal Modes

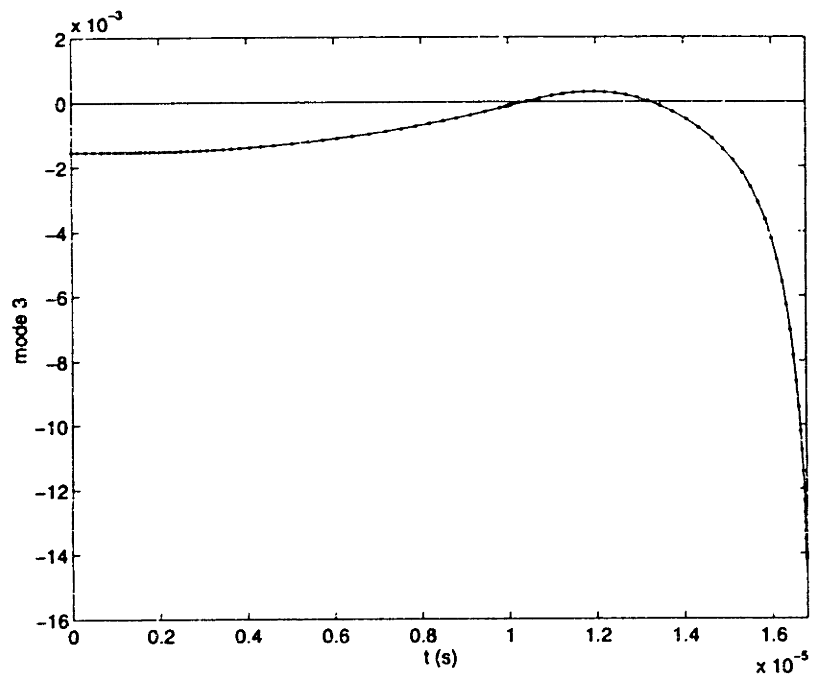


Figure 64 (cont): Ramp Time 10^{-5} - Normal Modes

References

[1] H. C. Nathanson, W. E. Newell, R. A. Wickstrom, and J. R. Davis, Jr., "The Resonant Gate Transistor", IEEE Transactions on Electron Devices, Vol. ED-14, No. 3, March 1967, pp. 117-133.

[2] K. E. Petersen, "Dynamic Micromechanics on Silicon: Techniques and Devices", IEEE Transactions on Electron Devices, Vol. ED-25, No. 10, October 1978, pp. 1241-1250.

[3] K. E. Petersen, "Silicon as a Mechanical Material", Proceedings of IEEE, Vol. 70, No. 5, May 1982, pp. 420-457.

[4] K. E. Petersen, "Silicon Sensor Technologies", Proceedings of IEDM, May 1985, pp. 2-7.

[5] S. D. Senturia, "CAD for Microelectromechanical Systems", to be presented at the International Conference on Solid-State Sensors and Actuators, Transducers '95, Stockholm, June 26-29, 1995.

[6] S. D. Senturia, R. M. Harris, B. P. Johnson, S. Kim, K. Nabors, M. A. Shulman, and J. K. White, "A Computer-Aided Design System for Microelectromechanical Systems (MEMCAD)", Journal of MEMS, Vol. 1, No. 1, March 1992, pp. 3-13.

[7] I-DEAS Master's Series Version 1.3c software, SDRC, Milford, Ohio.

[8] K. Nabors and J. White, "FastCap: A multipole-accelerated 3-D capacitance extraction program", IEEE Transactions on Computer-Aided Design, Vol. 10, No. 10, November 1991, pp. 1447-1459.

[9] J. R. Gilbert, R. Legtenberg, and S. D. Senturia, "3D Coupled Electromechanics for MEMS: Applications of CoSolve-EM", Proc. MEMS '95, Amsterdam, the Netherlands, January 29 - February 2, 1995, pp. 122-127.

[10] E. Jansen and J. Lang, private communication.

[11] I. S. Gradshteyn and I. M. Ryzhik, *Table of Integrals, Series, and Products*, 4th edition, Academic Press, Inc., 1980.

[12] F. R. Morgenthaler, "Theoretical Studies of Microstrip Antennas, Vol. 1: General Design Techniques and Analyses of Single and Coupled Elements", U. S. Department of Transportation, Federal Aviation Administration Report No. FAA-EM-79-11, Sept. 1979, pp. 32-34.

[13] H. B. Palmer, "The Capacitance of a Parallel-Plate Capacitor by the Schwartz-Christoffel Transformation", Electrical Engineering, Vol. 56, No. 3, March 1937, pp. 363-366.

[14] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press, 1988, pp. 542-547.

[15] H. A. Haus and J. R. Melcher, *Electromagnetic Fields and Energy*, Prentice Hall, 1989.

[16] J. R. Roark and W. C. Young, *Formulas for Stress and Strain*, 5th edition, McGraw Hill, 1975.

[17] J. White, lecture notes for *MIT 6.336: Introduction to Numerical Algorithms*, Fall 1994.