Sample size effect in ultrasonic testing of geomaterials - numerical and experimental study

by

Simon Bérubé

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Applied Science
in
Civil Engineering

Waterloo, Ontario, Canada, 2008

Abstract

Nondestructive evaluation of civil structures is of increasing interest to utility owners. Several methods exist to evaluate different properties of concrete, pavement, cemented sands and others. UPVM is the most commonly used ultrasonic technique in civil structures due to its simplicity and ease of use. UPVM is fast and requires minimal skill from operators. It has been used for flaw detection, study of material contents, deduction of general deterioration, determination of elastic properties , measurement of strength, and others. In such applications, accurate measurements of velocity are essential for proper parameter evaluation and thus to increase the validity of conclusions obtained from measurements. Previous research in ultrasonic pulse velocity have found that UPVM are susceptible to specimen size, attenuation and frequency but no clear conclusions have yet to be made on the fundamental reason for the differences.

This work seeks to identify the main factors responsible for velocity differences due to specimen size and measuring frequency in civil engineering materials. The effects are investigated by first performing numerical simulations of concrete specimens of varying sizes, and properties, excited by both a low (55 kHz) and high (850 kHz) frequency input source. Simulations are used to model wave propagation in cylindrical concrete specimen. Transducer sound fields are also numerically studied using known analytical solutions. An experimental program is conducted to study variations in UPVM in 12 mortar and 11 concrete cylindrical specimens of varying widths and heights caused by different measuring frequencies.

Simulations are completed for 12 specimen of different dimensions having heights of 5,10,20 and 30 cm as well as diameters of 10, 20 and 30 cm. Both a low ($f_0 = 55kHz$) and high ($f_0 = 850MHz$) frequency input source are used on each specimen. Numerical simulations using low frequencies are made for both a damped and undamped series of specimen. Results from low frequency simulations of damped models indicate that wave attenuation can lead to significant errors in first arrivals when complex wave interference is present. Conditions for wave interference at the receiver location are studied and minimum size conditions for both height and width are derived. These conditions guarantee proper pulse separation for UPVM and are dependent on source size, and source pulse width. It is argued that with proper use these conditions will lead to accuracy of measurement better than one quarter of a period of the main excitation frequency when using a full waveform and a skilled operator.

Finally, experiments are performed to assess differences in first arrivals between high and low frequency measurements. Readings are made on 11 mortar and 12 concrete specimen of different heights and widths. Experimentally significant time differences are observed between high and low frequency readings. It is found that differences in first arrivals will increase with specimen length but differences in velocity will decrease with length. Specimens 4 wavelengths in height are deemed sufficient to diminish surface effects to a minimum provided the specimens are healthy (e.g. no internal flaws). Any increase past 4 wavelengths is found to have negligible effects on measured velocity in healthy specimens.

Acknowledgements

I would like to express profound gratitude to my advisors, Dr. Giovanni Cascante and Dr. Maria Anna Polak for their support, encouragements, supervision and useful suggestions through this research work.

I am also thankful to Kinectrics Inc. and the Natural Sciences and Engineering Research Council (NSERC) for providing funding and support through the NSERC Industrial Postgraduate Scholarship. Specifically, I am grateful for the comments and guidance of Dr. Andy Z. Chen and Dr. Alex Karpelson of Kinectrics Inc. during the course of this research.

I am, as ever, indebted to my parents for their continuous support and encouragement through my entire university career. Their support and encouragement made this work possible.

Finally, I would like to extend my gratitude to the technical staff at the University of Waterloo Geotechnical Engineering Department for their help in properly conducting experiments, fabricating specimen and operating the various heavy equipments needed for this research.

Contents

1 Introduction **1**

2 Literature Review **4**

2.1 Elastic Waves 4

2.1.1 Semi-Infinite Half Space 5

2.1.2 Layered Media 6

2.2 Nondestructive Testing (NDT) Methods for Condition Assessment of Concrete 8

2.2.1 Electromagnetic Methods 8

2.2.2 Acoustic Methods 9

2.2.3 Recent Ultrasonic Applications 12

2.3 Summary 15

3 Theoretical Background **16**

3.1 Mechanical Waves 16

3.1.1 Plane Waves in a Homogeneous Medium 16

3.1.2 Elastic Waves 19

3.2 Signal Analysis 26

3.2.1 Fourier Transform 26

3.2.2 Frequency Wave-number Transform 30

3.2.3 Convolution 36

3.2.4 Hilbert Transform 39

3.3 Transducer Characterization and Design 42

3.3.1 Acoustic Field of a Piston Transducer 42

3.4 Numerical Simulations 48

3.4.1 The Finite Element Method 48

3.4.2 Model calibration and verification 55

4 Numerical and Experimental Methodology **63**

4.1 Numerical Simulations 64

4.1.1 Parameter Definition Considerations 64

4.1.2 Lamb Calibration Model 67

4.1.3 Layered Calibration Models 71

4.1.4 Model Simulations of mortar specimens 76

4.2 Laboratory testing 81

4.2.1 Calibration Methodology 81
4.2.2 First Arrival Measurments 85
4.2.3 Measurements Methodology 89
4.2.4 Specimens 90
4.2.5 Mortar specimen preparation 90
4.2.6 Concrete sample preparation 92
4.3 Applications 93
4.3.1 Data Management and Processing 93
4.3.2 Transducer Modeling 95

5 Results and Discussion 100
5.1 Numerical Simulations 100
5.1.1 Lamb's Calibration Problem 100
5.1.2 Single Layer Calibration Model 105
5.1.3 Three Layer Calibration Model 109
5.1.4 Numerical Simulations 113
5.1.5 Transducer Field 131
5.2 Experimental Velocity Measurements 134
5.3 Discussion 138
5.3.1 Calibration of Numerical Models 138
5.3.2 Numerical Models 141
5.3.3 Experimental Results 149
5.3.4 Recommendations 150

6 Conclusions 152

A LS-Dyna Input Decks 155
A.1 Lamb's Problem Deck 155
A.2 Punch Single Layer Model 157
A.3 Punch Three Layer Model 158
A.4 High Frequency Experimental Model - Sample 160
A.5 Low Frequency Experimental Model - Sample 161

B Python Classes 162
B.1 LayergenDyna 162
B.2 modelGenerator1M 171
B.3 modelGenerator50K 172
B.4 tt 173
B.5 ttlist 181

C Field Solver Classes 188
C.1 array.h 188
C.2 array.cc 189
C.3 circtd.h 194
C.4 circtd.cc 194
C.5 impsine.h 195

C.6 impsine.cc 195
C.7 impulse.h 196
C.8 impulse.cc 196
C.9 solver.h 197
C.10 solver.cc 198
C.11 transducer.h 202

D Solver User Interface **203**
D.1 main.cc 203
D.2 pistonui.h 203
D.3 pistonui.cc 206
D.4 plotsolver.h 219
D.5 plotsolver.cc 220
D.6 td.h 223
D.7 td.cc 223
D.8 tdimport.h 224
D.9 tdimport.cc 225
D.10 types.h 228
D.11 ui_pistonui.h 228
D.12 ui_td.h 237
D.13 ui_tdimport.h 240

E Numerical Data **251**
E.1 Calibration Results 251
E.2 Numerical Simulations of Experimental Models data 255
E.2.1 High Frequency 255
E.2.2 Low Frequency 286
E.2.3 Low Frequency – Damped 298
E.2.4 Damped vs Undamped Time traces 311

F Experimental Data **316**
F.1 Experimental Time Traces 316
F.1.1 High Frequency 316
F.1.2 Low Frequency 325
F.1.3 First Arrivals 333

Bibliography **337**

List of Figures

2.1 Pulse arrival according to Lapwood (1949) 5

3.1 Representation of Snell's Law. (a) Reflection/Transmission, (b) Angles 19
3.2 Surface wave coordinate system 22
3.3 Sample of seismic trace 31
3.4 Sample f-k Plot for Broad Band Input 32
3.5 Broad Band Input Source (a) Time trace, (b) Magnitude of Fourier spectrum 33
3.6 Sample f-k Plot for Narrow Band Input 34
3.7 Narrow Band Input Source (a) Time trace, (b) Magnitude of Fourier spectrum 35
3.8 Visual Representation of Convolution (a) signal wavelet , f, (b) Impulse Response, g, and (c) convolved response, a. 38
3.9 Analytical Signal of time trace shown in Figure 3.8a. (a) Amplitude, (b) Phase and (c) Frequency 41
3.10 Circular piston transducer coordinate system 45
3.11 Impulse response of a circular transducer at selected locations 46
3.12 Near-field pressure acoustic field of a piston 47
3.13 Two Four-node elements structure 49
3.14 Sample curve of frequency dependent damping ratio using $\omega_1 = 628318$, $\omega_n = 9424778$, $\alpha = 47124$ and $\beta = 7.95775 * 10^{-9}$ 53
3.15 Lamb solution for the vertical and horizontal displacement due to a point source (eq. 3.98) 58

4.1 Time step instabilities in LS-DYNA 67
4.2 Axisymmetric numerical model used for Lamb Solution Calibration (360,000 Elements) 68
4.3 Lamb problem's source for simulation 70
4.4 Load for single layer model 73
4.5 Load for three layer model 75
4.6 Low Frequency Load for Experimental Model Simulations. (a) Time trace, (b) Power spectrum 78
4.7 High Frequency Load for Experimental Model Simulations. (a) Time trace, (b) Power spectrum 79
4.8 Damping curve used for low frequency excitation 80
4.9 Calibration Rods 82
4.10 Water Calibration Data for Aluminum Rods 83

4.11 Glycerin Calibration Data for Aluminum Rods 84
4.12 Experimental Ultrasonic Pulse Velocity Setup 85
4.13 Effect of coupling on signal strength using a high frequency measurement on a 5 cm diameter aluminum rod. Amplitudes are normalized to highest peak of water coupling. 89
4.14 Gradation curve of the sand used for mortar mix. 91
4.15 Database Scheme for Experimental Time Traces 94
4.16 Database Scheme for Numerical Time Traces 95
4.17 Field Solver class hierarchy 96
4.18 Field Solver User Interface's setup screen 97
4.19 Field Solver User Interface's setup screen 98
4.20 Field Solver User Interface's setup screen 99

5.1 Lamb calibration model: numerical vs theoretical trace at 41mm 102
5.2 Lamb calibration model: numerical vs theoretical trace at 83mm 102
5.3 Lamb calibration mode: numerical vs theoretical trace at 125mm 103
5.4 Lamb calibration mode: numerical vs theoretical trace at 166mm 103
5.5 Lamb calibration model: numerical vs theoretical trace at 208mm 104
5.6 Lamb calibration model: sample numerical vs theoretical spectra at 166mm . 104
5.7 Comparison results for single layer model at 15 m 106
5.8 Comparison results for single layer model at 45 m 106
5.9 Comparison results for single layer model at 75 m 107
5.10 Comparison results for single layer model at 90 m 107
5.11 Comparison results of spectrum for single layer model at 15 m 108
5.12 Comparison results for three layer model at 15 m 110
5.13 Comparison results for three layer model at 45 m 110
5.14 Comparison results for three layer model at 75 m 111
5.15 Comparison results for three layer model at 90 m 111
5.16 Comparison results for three layer model at 15 m 112
5.17 Frequency-wavenumber plot for model "m1a" (H=9.24 λ_0, D= 18.48 λ_0) using high frequency excitation ($f_0 = 850kHz, \lambda_0 = 5.412mm$). P indicates the main p-wave, S indicates the main shear wave, RP is the reflected p-wave and RS is the reflected shear wave. 114
5.18 Time history plot of central axis of model "m1a" using high frequency excitation 115
5.19 Diagram of selected wavefronts in cylindrical specimen 116
5.20 Time history plot of vertical displacement of the top axis of model "m1a" using high frequency excitation 117
5.21 Time history plot of horizontal displacement of the top axis of model "m1a" using high frequency excitation 118
5.22 Time history plot of vertical displacement of the right axis of model "m1a" using high frequency excitation 119
5.23 Time history plot of horizontal displacement of the right axis of model "m1a" using high frequency excitation 120
5.24 Time history plot of vertical displacement of the bottom axis of model "m1a" using high frequency excitation 121

5.25 Time history plot of vertical displacement of the bottom axis of model “m1a” using high frequency excitation 122
5.26 Frequency-wavenumber plot for model “m4a” (H=55.44 λ_0, D= 18.48 λ_0) using high frequency excitation ($f_0 = 850kHz, \lambda_0 = 5.412mm$). P is the p-wave, S is the shear wave, and RP is the reflected p-wave. 123
5.27 Time history plot of vertical displacement of the bottom axis of model “m1a” using high frequency excitation 124
5.28 Frequency-wavenumber plot for model “m1a” (H=0.60 λ_0, D=1.20 λ_0) at low frequency excitation ($f_0 = 55kHz, \lambda_0 = 83.0mm$). 125
5.29 Time history plot of model “m1a” (H=0.60 λ_0, D=1.20 λ_0) at low frequency excitation ($f_0 = 55kHz, \lambda_0 = 83.0mm$). P is the p-wave arrival line and S is the computed shear wave arrival line. 125
5.30 Time history plot of damped model “m1a” 127
5.31 Time history plot of damped model “m2a” 127
5.32 Time history plot of damped model “m3a” 128
5.33 Comparison of time signal between damped ($D_{min} = 0.025, \alpha = 4435.2, \beta = 1.5 * 10^{-7}$) and undamped model “m1a” (H=0.60 λ_0, D=1.20 λ_0) at low frequency excitation ($f_0 = 55kHz, \lambda_0 = 83.0mm$) 128
5.34 Comparison of time signal between damped ($D_{min} = 0.025, \alpha = 4435.2, \beta = 1.5 * 10^{-7}$) and undamped model “m2a” (H=1.20 λ_0, D=1.20 λ_0) at low frequency excitation ($f_0 = 55kHz, \lambda_0 = 83.0mm$) 129
5.35 Comparison of time signal between damped ($D_{min} = 0.025, \alpha = 4435.2, \beta = 1.5 * 10^{-7}$) and undamped model “m3a” (H=2.41 λ_0, D=1.20 λ_0) at low frequency excitation ($f_0 = 55kHz, \lambda_0 = 83.0mm$) 129
5.36 Arrival Time Comparison for numerical models 130
5.37 Pressure Field for High Frequency Transducer 132
5.38 Pressure Field for Low Frequency Transducer 133
5.39 Wave velocity differences from high ($f_h = 850kHz$) and low ($f_l = 55kHz$) frequency measurements for different concrete specimens heights and diameters (D=0.60, 1.2 and 2.4 λ_0 for $f_0 = 55kHz, \lambda_0 = 83.01mm$). 134
5.40 Wave velocity differences from high ($f_h = 850kHz$) and low ($f_l = 55kHz$) frequency measurements for different mortar specimens heights and diameters (D=0.60, 1.2 and 2.4 λ_0 for $f_0 = 55kHz, \lambda_0 = 83.01mm$) 135
5.41 Arrival time differences from high ($f_h = 850kHz$) and low ($f_l = 55kHz$) frequency measurements for different concrete specimens heights and diameters (D=0.60, 1.2 and 2.4 λ_0 for $f_0 = 55kHz, \lambda_0 = 83.01mm$). 135
5.42 Arrival time differences from high ($f_h = 850kHz$) and low ($f_l = 55kHz$) frequency measurements for different mortar specimens heights and diameters (D=0.60, 1.2 and 2.4 λ_0 for $f_0 = 55kHz, \lambda_0 = 83.01mm$). 136
5.43 Comparison of experimental waveforms for specimen “c1a” (H=5 cm, D=10 cm) between three independent readings using high frequency excitation ($f_0 = 850kHz$). 136
5.44 Comparison of experimental waveforms for specimen “m1a” (H=5 cm, D=10 cm) between three independent readings using low frequency excitation ($f_0 = 55kHz$). 137
5.45 Normalized measurement of theoretical specimen 144

5.46 Arrival time of high amplitude fronts 145
5.47 Minimum Radius to Height ratio to ensure p-wave pulse separation in a specimen 146
5.48 Delay lag effect demonstration 148

E.1 Water Calibration Data for Aluminum Rods 251
E.2 Water Calibration Data for Steel Rods 252
E.3 Water Calibration Data for PVC Rods 252
E.4 Glycerin Calibration Data for Aluminum Rods 253
E.5 Glycerin Calibration Data for Steel Rods 253
E.6 Glycerin Calibration Data for PVC Rods 254
E.7 High frequency seismic plot for axisymmetric axis of specimen “m1a” 255
E.8 High frequency seismic plot for axisymmetric axis of specimen “m1b” 256
E.9 High frequency seismic plot for axisymmetric axis of specimen “m1c” 257
E.10 High frequency seismic plot for axisymmetric axis of specimen “m2a” 258
E.11 High frequency seismic plot for axisymmetric axis of specimen “m2b” 259
E.12 High frequency seismic plot for axisymmetric axis of specimen “m2c” 260
E.13 High frequency seismic plot for axisymmetric axis of specimen “m3a” 261
E.14 High frequency seismic plot for axisymmetric axis of specimen “m3b” 262
E.15 High frequency seismic plot for axisymmetric axis of specimen “m3c” 263
E.16 High frequency seismic plot for axisymmetric axis of specimen “m4a” 264
E.17 High frequency seismic plot for axisymmetric axis of specimen “m4b” 265
E.18 High frequency seismic plot for axisymmetric axis of specimen “m4c” 266
E.19 High frequency seismic plot for bottom axis of specimen “m1a” 267
E.20 High frequency seismic plot for top axis of specimen “m1a” 268
E.21 High frequency seismic plot for side (right) axis of specimen “m1a” 269
E.22 High frequency seismic plot for bottom axis of specimen “m2a” 270
E.23 High frequency seismic plot for top axis of specimen “m2a” 271
E.24 High frequency seismic plot for side (right) axis of specimen “m2a” 272
E.25 High frequency seismic plot for bottom axis of specimen “m2a” 273
E.26 High frequency seismic plot for top axis of specimen “m3a” 274
E.27 High frequency seismic plot for side (right) axis of specimen “m3a” 275
E.28 High frequency seismic plot for bottom axis of specimen “m3a” 276
E.29 High frequency seismic plot for top axis of specimen “m4a” 277
E.30 High frequency seismic plot for side (right) axis of specimen “m4a” 278
E.31 High frequency seismic plot for bottom axis of specimen “m4a” 279
E.32 High frequency F-K plot for axisymmetric axis of specimen “m1a” 280
E.33 High frequency F-K plot for axisymmetric axis of specimen “m1b” 280
E.34 High frequency F-K plot for axisymmetric axis of specimen “m1c” 281
E.35 High frequency F-K plot for axisymmetric axis of specimen “m2a” 281
E.36 High frequency F-K plot for axisymmetric axis of specimen “m2b” 282
E.37 High frequency F-K plot for axisymmetric axis of specimen “m2c” 282
E.38 High frequency F-K plot for axisymmetric axis of specimen “m3a” 283
E.39 High frequency F-K plot for axisymmetric axis of specimen “m3b” 283
E.40 High frequency F-K plot for axisymmetric axis of specimen “m3c” 284
E.41 High frequency F-K plot for axisymmetric axis of specimen “m4a” 284

E.42 High frequency F-K plot for axisymmetric axis of specimen “m4b” 285
E.43 High frequency F-K plot for axisymmetric axis of specimen “m4c” 285
E.44 Low frequency seismic plot for axisymmetric axis of specimen “m1a” 286
E.45 Low frequency seismic plot for axisymmetric axis of specimen “m1b” 286
E.46 Low frequency seismic plot for axisymmetric axis of specimen “m1c” 287
E.47 Low frequency seismic plot for axisymmetric axis of specimen “m2a” 287
E.48 Low frequency seismic plot for axisymmetric axis of specimen “m2b” 288
E.49 Low frequency seismic plot for axisymmetric axis of specimen “m2c” 288
E.50 Low frequency seismic plot for axisymmetric axis of specimen “m3a” 289
E.51 Low frequency seismic plot for axisymmetric axis of specimen “m3b” 289
E.52 Low frequency seismic plot for axisymmetric axis of specimen “m3c” 290
E.53 Low frequency seismic plot for axisymmetric axis of specimen “m4a” 290
E.54 Low frequency seismic plot for axisymmetric axis of specimen “m4b” 291
E.55 Low frequency seismic plot for axisymmetric axis of specimen “m4c” 291
E.56 Low frequency F-K plot for axisymmetric axis of specimen “m1a” 292
E.57 Low frequency F-K plot for axisymmetric axis of specimen “m1b” 292
E.58 Low frequency F-K plot for axisymmetric axis of specimen “m1c” 293
E.59 Low frequency F-K plot for axisymmetric axis of specimen “m2a” 293
E.60 Low frequency F-K plot for axisymmetric axis of specimen “m2b” 294
E.61 Low frequency F-K plot for axisymmetric axis of specimen “m2c” 294
E.62 Low frequency F-K plot for axisymmetric axis of specimen “m3a” 295
E.63 Low frequency F-K plot for axisymmetric axis of specimen “m3b” 295
E.64 Low frequency F-K plot for axisymmetric axis of specimen “m3c” 296
E.65 Low frequency F-K plot for axisymmetric axis of specimen “m4a” 296
E.66 Low frequency F-K plot for axisymmetric axis of specimen “m4b” 297
E.67 Low frequency F-K plot for axisymmetric axis of specimen “m4c” 297
E.68 Low frequency seismic plot for axisymmetric axis of specimen “m1a” using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$ 298
E.69 Low frequency seismic plot for axisymmetric axis of specimen “m1b” using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$ 299
E.70 Low frequency seismic plot for axisymmetric axis of specimen “m1c” using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$ 299
E.71 Low frequency seismic plot for axisymmetric axis of specimen “m2a” using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$ 300
E.72 Low frequency seismic plot for axisymmetric axis of specimen “m2b” using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$ 300
E.73 Low frequency seismic plot for axisymmetric axis of specimen “m2c” using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$ 301
E.74 Low frequency seismic plot for axisymmetric axis of specimen “m3a” using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$ 301
E.75 Low frequency seismic plot for axisymmetric axis of specimen “m3b” using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$ 302
E.76 Low frequency seismic plot for axisymmetric axis of specimen “m3c” using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$ 302
E.77 Low frequency seismic plot for axisymmetric axis of specimen “m4a” using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$ 303

E.78 Low frequency seismic plot for axisymmetric axis of specimen “m4b” using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$ 303
E.79 Low frequency seismic plot for axisymmetric axis of specimen “m4c” using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$ 304
E.80 Low frequency F-K plot for axisymmetric axis of specimen “m1a” using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$ 305
E.81 Low frequency F-K plot for axisymmetric axis of specimen “m1b” using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$ 305
E.82 Low frequency F-K plot for axisymmetric axis of specimen “m1c” using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$ 306
E.83 Low frequency F-K plot for axisymmetric axis of specimen “m2a” using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$ 306
E.84 Low frequency F-K plot for axisymmetric axis of specimen “m2b” using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$ 307
E.85 Low frequency F-K plot for axisymmetric axis of specimen “m2c” using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$ 307
E.86 Low frequency F-K plot for axisymmetric axis of specimen “m3a” using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$ 308
E.87 Low frequency F-K plot for axisymmetric axis of specimen “m3b” using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$ 308
E.88 Low frequency F-K plot for axisymmetric axis of specimen “m3c” using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$ 309
E.89 Low frequency F-K plot for axisymmetric axis of specimen “m4a” using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$ 309
E.90 Low frequency F-K plot for axisymmetric axis of specimen “m4b” using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$ 310
E.91 Low frequency F-K plot for axisymmetric axis of specimen “m4c” using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$ 310
E.92 Comparison of time signal between damped and undamped model ’m1a’ at low frequency 311
E.93 Comparison of time signal between damped and undamped model ’m1b’ at low frequency 311
E.94 Comparison of time signal between damped and undamped model ’m1b’ at low frequency 312
E.95 Comparison of time signal between damped and undamped model ’m1c’ at low frequency 312
E.96 Comparison of time signal between damped and undamped model ’m2a’ at low frequency 313
E.97 Comparison of time signal between damped and undamped model ’m2b’ at low frequency 313
E.98 Comparison of time signal between damped and undamped model ’m2c’ at low frequency 314
E.99 Comparison of time signal between damped and undamped model ’m3a’ at low frequency 314
E.100Comparison of time signal between damped and undamped model ’m3b’ at low frequency 315

E.101 Comparison of time signal between damped and undamped model 'm3c' at low frequency . . . 315

F.1 High frequency experimental time trace for 5 cm high mortar specimens . . . 316
F.2 Frequency spectrum of high frequency experimental time trace for 5 cm high mortar specimens . . . 317
F.3 High frequency experimental time trace for 5 cm high mortar specimens . . . 317
F.4 Frequency spectrum of high frequency experimental time trace for 10 cm high mortar specimens . . . 318
F.5 High frequency experimental time trace for 20 cm high mortar specimens . . . 318
F.6 Frequency spectrum of high frequency experimental time trace for 20 cm high mortar specimens . . . 319
F.7 High frequency experimental time trace for 30 cm high mortar specimens . . . 319
F.8 Frequency spectrum of high frequency experimental time trace for 30 cm high mortar specimens . . . 320
F.9 High frequency experimental time trace for 5 cm high mortar specimens . . . 321
F.10 Frequency spectrum of high frequency experimental time trace for 5 cm high mortar specimens . . . 321
F.11 High frequency experimental time trace for 5 cm high mortar specimens . . . 322
F.12 Frequency spectrum of high frequency experimental time trace for 10 cm high mortar specimens . . . 322
F.13 High frequency experimental time trace for 20 cm high mortar specimens . . . 323
F.14 Frequency spectrum of high frequency experimental time trace for 20 cm high mortar specimens . . . 323
F.15 High frequency experimental time trace for 30 cm high mortar specimens . . . 324
F.16 Frequency spectrum of high frequency experimental time trace for 30 cm high mortar specimens . . . 324
F.17 Low frequency experimental time trace for 5 cm high mortar specimens . . . 325
F.18 Frequency spectrum of low frequency experimental time trace for 5 cm high mortar specimens . . . 325
F.19 Low frequency experimental time trace for 5 cm high mortar specimens . . . 326
F.20 Frequency spectrum of low frequency experimental time trace for 10 cm high mortar specimens . . . 326
F.21 Low frequency experimental time trace for 20 cm high mortar specimens . . . 327
F.22 Frequency spectrum of low frequency experimental time trace for 20 cm high mortar specimens . . . 327
F.23 Low frequency experimental time trace for 30 cm high mortar specimens . . . 328
F.24 Frequency spectrum of low frequency experimental time trace for 30 cm high mortar specimens . . . 328
F.25 Low frequency experimental time trace for 5 cm high mortar specimens . . . 329
F.26 Frequency spectrum of low frequency experimental time trace for 5 cm high mortar specimens . . . 329
F.27 Low frequency experimental time trace for 5 cm high mortar specimens . . . 330
F.28 Frequency spectrum of low frequency experimental time trace for 10 cm high mortar specimens . . . 330
F.29 Low frequency experimental time trace for 20 cm high mortar specimens . . . 331

F.30 Frequency spectrum of low frequency experimental time trace for 20 cm high mortar specimens . . . 331
F.31 Low frequency experimental time trace for 30 cm high mortar specimens . . . 332
F.32 Frequency spectrum of low frequency experimental time trace for 30 cm high mortar specimens . . . 332

List of Tables

2.1 Formulas used to calculated longitudinal and shear wave velocity 7

4.1 Layer properties for three layer model 74
4.2 List of sizes for experimental model simulations 76
4.3 Size of calibration rods used. 81
4.4 Compiled calibration results 83
4.5 Dimensions of specimens, selected assuming $V_p = 4,800m/s$ 90
4.6 Precise measurements of mortar samples 92
4.7 Precise measurements for concrete specimens 93

5.1 Comparison of arrival time from damped and undamped numerical simulations.130

F.1 Raw data of velocity reading for 5 cm high mortar samples 333
F.2 Raw data of velocity reading for 10 cm high mortar samples 334
F.3 Raw data of velocity reading for 20 cm high mortar samples 334
F.4 Raw data of velocity reading for 30 cm high mortar samples 334
F.5 Raw data of velocity reading for 5 cm high concrete samples 335
F.6 Raw data of velocity reading for 10 cm high concrete samples 335
F.7 Raw data of velocity reading for 20 cm high concrete samples 336
F.8 Raw data of velocity reading for 30 cm high concrete samples 336

Variable	**Description**
α	Mass damping parameter
β	Stiffness damping parameter
ϵ_v	Volume expansion, defined as $\epsilon_v = \frac{\partial u_1}{\partial x_1} + \frac{\partial u_2}{\partial x_2} + \frac{\partial u_3}{\partial x_3}$
$\theta(t)$	Time-dependent phase
θ_i	Incidence angle
θ_r	Reflected angle
θ_t	Transmitted angle
$\xi^{(m)}$	Damping parameter
λ	Wavelength
λ_l	The first Lamé constant
μ_l	The second Lamé constant
ν	Poisson ratio
ρ	Material density
$\bar{\sigma}$	Fourier transform of normal stress
τ_l	Frequency modulating constant for Lamb pulse
$\bar{\tau}$	Fourier transform of shearing stress
$\phi(t)$	Shear wave function
$\bar{\psi}(\omega)$	Fourier transform of pressure on transducer's face
$\psi(t)$	Pressure wave function
ω	Angular frequency
$A(t)$	Time-dependent amplitude
a_k	Cosine fourier series constant
b_k	Sine fourier series constant
$\mathbf{C}$	Damping matrix
$D(\omega)$	Frequency dependent damping ratio
E	Young's modulus
f	Frequency
$f(x)$	Arbitrary function
$f'(x)$	Derivative of $f(x)$
f_N	Nyquist frequency
f_{kl}	Two-dimensional signal
$\bar{f}_{kl}$	Fourier transform of two-dimensional time signal
F	Amplitude factor for lamb's solution load function
G	Shear modulus
$g(t)$	Arbitrary time signal
$\bar{g}(f)$	Fourier transform of $g(t)$
$h(t)$	Arbitrary time signal
$\bar{h}(f)$	Fourier transform of $h(t)$
$h_t(t)$	Hilbert filter
$h_i(x, y, t)$	Impulse response function
$\bar{h_i}(x, y, \omega)$	Fourier transform of $h_i(x, y, t)$
$\mathbf{H}$	Displacement interpolation matrix
k	Wavenumber
$\mathbf{k}$	Wave vector
$\mathbf{K}$	Stiffness matrix

Variable	**Description**
M	Discrete number of steps in a given dimension
$\mathbf{M}$	Mass matrix
N	Discrete number of steps in a given dimension
$p(x, y, t)$	Time-dependent pressure
$\bar{p}(x, y, \omega)$	Fourier transform of pressure
$\bar{\mathbf{P}}$	Fourier transform of external load at top of a layer
Q	Source load function for Lamb's problem
R	Reflection coefficient
$\mathbf{R}$	External load vector
$\mathbf{S}$	Stress vector
$\bar{\mathbf{S}}$	Fourier transform of stress vector, $\mathbf{S}$
t	Time
T	Wave period
u	Displacement of principal axis (usually x-axis)
$\bar{u}$	Fourier transform of displacement
U	Displacement vector
$\bar{\mathbf{S}}$	Fourier transform of displacement vector
$\bar{s}(f)$	Solution from Punch, in frequency domain
$\dot{U}$	Velocity vector
$\ddot{U}$	Acceleration vector
v	Vertical displacement
V	Wave velocity
V_p	Longitudinal wave (P wave) velocity
V_s	Shear wave (S Wave) velocity
V_r	Rayleigh wave velocity
$\mathbf{x}$	Position vector
$x(t)$	One-dimensional signal
$\bar{x}(k)$	Fourier transform of one-dimensional signal
Z	Acoustic impedance

Chapter 1

Introduction

Nondestructive evaluation of civil structures is of increasing interest to utility owners. Several methods exist to evaluate different properties of concrete, pavement, cemented sands and others. Methods can be separated in two major types: electromagnetic and acoustic. Electromagnetic methods include the use of ground penetrating radar (GPR), thermal imaging, electrical resistance testing, and radiography. Acoustic methods include among others impact echo, surface waves, seismic tomography and ultrasonic pulse velocity measurements (UPVM) (Popovics, 2003; McCann and Forde, 2001).

Acoustic methods are more versatile, cheaper and more efficient for condition evaluation than traditional intrusive or destructive testing. The largest limitation of these methods is the need for complex data processing and/or skilled operators (Shickert, 2002). Unfortunately, concrete have high attenuation characteristics making accurate assessment more difficult than in metals. There is a large demand for simple, more portable acoustic techniques, the most used of which is ultrasonic pulse velocity measurement.

UPVM is fast and requires minimal skill from operators. Velocity measurements are traditionally made using commercial equipment where the output is an arrival time, or by oscilloscope where arrival time is visually deduced. The latter provides a full waveform output that is critical in accurately deducing and understanding the arrival time. UPVM has been used for flaw detection (Sutan and Jaafar, 2003), study of material contents (Abo-Qudais, 2005; Ohdaira and Masuzawa, 2000; Popovics, 2005), deducing general deterioration (Ohdaira and Masuzawa, 2000), determination of elastic properties (Washer et al., 2005; Prassianakis and Prassianakis, 2004), measuring strength (Chang et al., 2006), and others. As such, velocity is an important measurable quantity in the calculation of more significant engineering values such as strength, or other elastic properties. For these applications, precise measurement of velocity is important to increase the accuracy, and thus the validity, of conclusions obtained from the results. Several difficulties exist in measuring velocities in concrete; most notably the rough surface condition and heterogeneous material composition. These conditions lead to significant signal distortion making more precise high frequency measurements (above 250 KHz) difficult in most field applications. The use of low frequency (approx. 50 KHz) is thus prevalent in UPVM of concrete.

The use of low frequency pulses is the cause of several issues in velocity measurements. Using UPVM, these frequencies have relatively large wavelength (approx. 8 cm) in typical concrete. Thus, the ratio of sample height to wavelength is small for typical concrete specimen leading to size and boundary effects. ASTM C597-02 dictates that specimen must be at least 1 wavelength in length and width to avoid measurement error. Philippidis and Aggelis (2005), however, show that a steady increase in frequency (25 KHz to 700 KHz) leads to an increase in measured velocity and thus an increase in its estimated elastic properties. These results indicate that the one wavelength rule referred by the ASTM C597-02 standard may not fully account for effects of large wavelength UPVM.

This work seeks to identify the main factors responsible for velocity differences due to specimen size and measuring frequency in civil engineering materials. The effects are investigated by first performing numerical simulations of concrete specimens of varying sizes, and properties, excited by both a low (55 kHz) and high (850 kHz) frequency input source. Simulations are used to model wave propagation in cylindrical concrete specimen.

Transducer sound fields are also numerically studied using known analytical solutions. An experimental program is conducted to study variations in UPVM in 12 mortar and 11 concrete cylindrical specimens of varying widths and heights caused by different measuring frequencies.

In this work, size effects (boundary effects) of low frequency measurements (55 KHz) are compared to high frequency readings (500-1500 KHz broadband). Numerical simulations are conducted to study the full waveform for material characterization. Experiments are conducted on 11 mortar and 12 concrete specimen having a combination of 5, 10, 20, 30 cm height and 10, 20, 30 cm width. Finite element simulations are conducted for the corresponding specimens using the same low and high frequency source as an excitation. Experimentally significant differences are observed between the low and high frequency readings. Results are presented showing a link between signal transmission through specimen surface and variation in velocity.

The arrival of important wavefronts are studied from numerical simulations. It is found that low frequency measurements can have overlapping wavefronts for typical specimen sizes (10 cm diameter, 30 cm length) using low frequency sources; this can affect first arrival readings. A condition for minimal specimen size is presented to avoid those effects. Using full waveform acquisition it is suggested that UPVM taken using the presented minimum size conditions will lead to accuracy better than one fourth of the period of the received signal at its main frequency.

Chapter 2

Literature Review

This chapter presents a review of relevant research in nondestructive evaluation of concrete. First, a brief review of theoretical research in wave propagation is presented, followed by a review of current research in nondestructive evaluation of concrete, with a focus on ultrasonic methods.

2.1 Elastic Waves

Elastic waves represent the entire body of waves that propagate elastically in a solid. Elastic waves exist in two fundamental modes: longitudinal (P waves) and shear (S waves). In an semi-infinite half space, surface waves, or Rayleigh waves, are also created on the free surface. Longitudinal waves propagate the fastest, followed by shear waves and then surface waves. Conversely, surface waves have the largest amplitude, followed by shear waves and then longitudinal waves.

In this section a review of relevant theoretical work in elastic wave propagation is presented.

2.1.1 Semi-Infinite Half Space

The solution of for surface wave propagation in an infinite half-space was first introduced by Lamb (1904) for the case of an impulse vertical point load. Lamb's solution is valid for a point far from the source and includes separate solutions for three consecutive wave trains of longitudinal, transverse, and surface waves.

Lamb's solution was later expanded for the case of a buried line load and solved for cases close to the source (Nakano, 1925). However, Nakano's solution applies only for the two dimensional case. Lapwood (1949) revisited previous attempts by Lamb (1904) and Nakano (1925) and isolated the surface solution as an arrival of six different pulses on the surface. These pulses are different based on whether the initial impulse in the material is compressional or distortional.

For compressional excitation, the six pulses are in order: the direct P-pulse, the reflected P-pulse, the reflected S-pulse, the surface S-pulse, the secondary S-pulse and the Rayleigh-pulse. In the case of distortional excitation the pulses are: the reflected P-pulse, the surface P-pulse, the direct S-pulse, the secondary P-pulse, the reflected S-pulse and the Rayleigh-pulse. These pulses represent individual solutions to the equation presented by Lapwood (1949) and together represent the total solution.

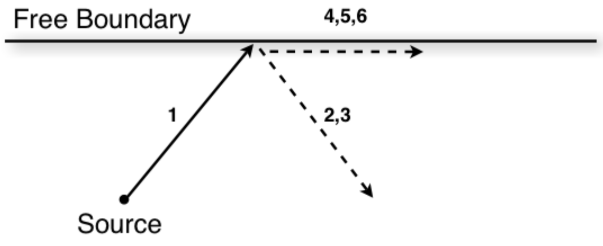


Figure 2.1: Pulse arrival according to Lapwood (1949). The numbers on the image identify the six pulses in their order of arrival at an arbitrary point

2.1.2 Layered Media

First Arrival Maps

The more complex problem of wave propagation in a layered medium has been solved differently depending on the application. The simplest approach to studying wave propagation in layered media is by first arrival maps, which is traditionally done using ray tracing algorithms (Julian and Gubbins, 1977). Ray tracing algorithms are limited because they are computationally intensive, give only approximate solutions and are only valid for wavelengths of negligible size compared to the dimensions of the medium.

Vidale (1988) suggested a different approach using finite-difference methods on two-dimensional soil profiles to compute time of flight. The method propagates the wavefronts both using linear rays, as occurs far from the source, and spherical wavefronts when values are computed close to the source. The method, however, is very sensitive to abrupt changes in velocities and is numerically unstable for discontinuous soil profiles. A three-dimensional approach was later presented (Vidale, 1990).

A solution to the discontinuity problem was presented by Podvin and Lecomte (1991). They show that the instabilities in Vidale's approach are due to the treatment of wavefronts as a single entity thus ignoring signal information. They solve this problem by separating the propagation fronts as transmission fronts, head waves, and diffracted wavefronts. Solving each of the fronts and picking the most appropriate, this method can accommodate discontinuities with impedance ratios up to 1:10.

Seismic traces

Cagniard (1939) was the first to suggest a method to solve the problem of wave propagation in a stratified media. The method, commonly known as the Cagniard-de Hoop-Pekeris method due to later contribution by de Hoop (1960) and Pekeris (1955), consists on analytically solving the inverse transform of the impulse response. The method is effective analytically on homogeneous stratified medias and numerically on more complex configurations (Chapman, 2004). However, the method is designed to solve ray refraction and reflection thus not providing a solution for surface wave propagation.

Ray tracing methods solving the propagation of different wave fronts are common (Cerveny, 2001; Um and Thurber, 1987; Asakawa and Kawanaka, 1993; Smirnova, 1995). Ray tracing methods are useful because they can be solved numerically for arbitrary configurations. They also have the added benefit of providing information about amplitude, waveform shapes and travel times (Cerveny, 2001). However, ray tracing methods are computationally intensive, and have limited accuracy since they depend on infinite summations.

Kausel (1981) presented another approach by providing an explicit, closed-form solution for the Green functions of dynamic loads acting on a layered medium. This approach is more flexible than ray tracing as it is computationally more efficient and can model arbitrary layer configurations.

Table 2.1: Formulas used to calculated longitudinal (V_p) and shear (V_s) wave velocity. Together, these relationships form the basis for ultrasonic pulse velocity measurements. E is the Young's modulus, ρ is the density, and ν is the poisson ratio.

Velocity Type	Formula
Longitudinal (P-Wave)	$V_p = \sqrt{\frac{E(1-\nu)}{\rho(1+\nu)(1-2\nu)}}$
Shear (S-Wave)	$V_s = \sqrt{\frac{E}{2(1+\nu)}}$

2.2 Nondestructive Testing (NDT) Methods for Condition Assessment of Concrete

The complexity of concrete as compared to standard metals makes nondestructive evaluation difficult. Multiple methods have been suggested and used, each one with strengths and weaknesses. The most common methods can be classified as either electromagnetic or acoustic. Electromagnetic methods include ground penetrating radar, thermal imaging, electrical resistance testing, radiography, etc. Acoustic methods include ultrasonic pulse velocity measurements (UPV), impact echo, seismic tomography, and surface waves (Popovics, 2003; McCann and Forde, 2001; Yang et al., 2005).

2.2.1 Electromagnetic Methods

Radiography provides good results but requires expensive equipment, is not portable and has limited penetration in concrete making it rarely used outside a laboratory (Popovics, 2003). Feng et al. (2000, 2002) attempted to solve this problem, to some extent, by using microwave radiography on fiber reinforced plastic (FRP) jacketed concrete beams with some success.

Conductivity and resistance testing is a less expensive, portable and efficient NDT method, which can provide information about voids, reinforcing bar location, as well as moisture and salt content (McCann and Forde, 2001). These electrical methods are used mainly for complementary information as they do not provide direct condition assessment of structures.

For example, the half cell potential method (HCP) uses the potential difference between a reinforcement bar and an electrode placed on the structure's surface. Variations in potential are used to assess corrosion of the rebar but offer limited information on the health of the concrete.

2.2.2 Acoustic Methods

Acoustic methods are economical, efficient and can be powerful methods for the condition assessment of structures. The largest limitation of these methods is the need for complex data processing and skilled operators. There is also a lack of engineering standards (Shickert, 2002).

Impact Echo

The impact-echo technique is commonly used as a simple field method but it requires significant data processing. In this method, a hammer strike or falling ball is used as an excitation source; the resulting echo from an anomaly or boundary is recorded by a transducer. Data processing can then be performed to extract information from the resonance condition of the echo (Popovics and Rose, 1994). Tawhed and Gassman (2002) successfully used the impact echo technique in the study of bridge decks, they were able to detect the onset of crack propagation before visual damage could be observed. The cracks ranged in sizes but were all larger than at least half the smallest wavelength of the source. Sutan and Jaafar (2003) used impact-echo for the detection of flaws much larger than the main wavelength generated and were able to detect crack location on concrete as young as 3 days old with 58% accuracy.

Ultrasonic Pulse Velocity

Ultrasonic pulse velocity (UPV) is the simplest and most commonly used NDT method for concrete. It consists of deducing specimen condition based on travel time information. The UPV method requires a contact transducer on each side of a specimen or a single source acting as both transmitter and receiver. Sutan and Jaafar (2003) used UPV measurements on voids and found it half as accurate as impact echo for crack location.

Abo-Qudais (2005) studied the effect of concrete mixture on velocity values and found that velocities vary widely based on type of concrete mixtures, aggregate size and specifically water content. Similarly, Ohdaira and Masuzawa (2000) studied the effect of water content on pulse velocity and found a 7% increase in water content (from dry) can lead to a 12.5% increase in velocity. They used concrete samples of 10 cm in diameter and 20 cm in length. The source impulse used had a frequency band of 20 to 100 kHz corresponding to wavelengths of 20 and 4 cm for the measured velocity of 4,000 m/s (or height to wavelength ratio of 1 and 5). They suggested that velocity readings could be related to the strength of concrete through the deduced water content; however, no tests were performed to confirm this statement. Hernandez et al. (2000) tried to evaluate concrete deterioration caused by environmental damage by deducing concrete porosity using ultrasonic pulse velocity. They concluded that while it was possible to do so, it would only be conclusive on elastic, isotropic concrete with randomly distributed pores and require previous knowledge of the concrete composition.

Popovics (2005) studied the effect of uneven moisture distribution inside concrete on UPV results. He used 10 cm and 15 cm deep slabs as specimens and used an ultrasonic source with frequency of up to 100 kHz ($\lambda = 4.5cm$, $distance/\lambda \approx 2$ and 3, respectively where λ is the wavelength of the P-wave). He concludes that the moisture content distribution makes it difficult to accurately estimate an average moisture content from UPV measurements. He also found that the effect of moisture distribution inside a crack is a significant factor when determining its length through UPV methods.

UPV was also used by Washer et al. (2005) on reactive powder concrete where they concluded that it could be used to assess the modulus of elasticity of specimens. They used both 10 cm cubes and cylinders 15 cm in length and 7.5 cm in diameter. The largest aggregate size used in the mixtures was 0.6 mm and the tests were performed with frequencies ranging from 0.5 to 1 MHz. The use of higher frequencies is possible only because of the small aggregates used in this type of materials (largest aggregate is roughly 1 order of magnitude smaller than smallest wavelength). Prassianakis and Prassianakis (2004) obtained similar conclusions when testing standard concrete and marble using UPV versus destructive tests.

They tested cylindrical specimens 15 cm in diameter and 30 cm in length as well as 15 cm cubic samples. The source used was excited at 1 MHz but no mention of the frequency bandwidth was made. Both Washer et al. (2005) and Prassianakis and Prassianakis (2004) used uniform specimens for which the composition and curing process were well known. This information was critical to properly evaluate the material properties from ultrasonic velocity.

Chang et al. (2006) claim that, for young (7-56 days) lightweight aggregate concrete, a change in strength of up to 37% leads to an increase of velocity of only 7%. This low change is directly related to the known relationship of Young's modulus and velocity: $V_p \propto \sqrt(E)$. The specimens used were cylinders 10 cm in diameter and 20 cm in length, excited by a 1 kHz to 100 kHz source with measured peak frequency at 9.07 kHz (corresponding wavelength 2.4 times larger than specimen height). The low frequency used could explain the small difference in velocity observed.

Philippidis and Aggelis (2005) studied the effect of frequency and geometry on UPV readings and found that higher frequency measurements gave higher velocities. They further noted that different geometry, and where the measurements is taken (eg. side to side or top to bottom), had an effect on velocity. Additionally, they found that the size of the specimen had an effect on the results. They compared results of 7.3 cm and 15 cm cubes at frequencies ranging from 25 kHz ($\lambda = 18cm$, $height/\lambda \approx 0.4$ for 7.3 cm specimen) to 700 kHz ($\lambda = 0.6cm$, $height/\lambda \approx 12$ and 24 respectively). The smaller specimen had marginally higher velocities, less than 3% higher, for frequencies between 25 and 100 kHz ($\lambda = 18$ and 4.5 cm, $height/\lambda \approx 0.4$ and 1.2 for 7.3 cm specimen) and the reverse was true for frequencies up to 700 kHz ($\lambda = 0.6$, $height/\lambda \approx 12$ for 7.3 cm specimen). They do not conclude on a possible cause for these effects but suggest that higher frequency measurements, above 100 kHz ($\lambda = 4.5cm$, $height/\lambda \approx 1.2$ for 7.3 cm specimen), be used for parameter estimation. The ASTM standard C597-02 states that pulse velocity measurements are independent of specimen size and shapes as long as their sizes are in the order of one wavelength or more of the source used. The standard also recommend use of frequencies between 20 and 100 kHz. The results by Philippidis and Aggelis (2005) indicate that such a requirement might not adequately eliminate variations in velocities due to frequency and size.

2.2.3 Recent Ultrasonic Applications

Several methods are emerging as effective solutions for the acoustic nondestructive evaluation of concrete. These techniques are more complex to use on the field and required more data processing but also extend the usefulness of ultrasonic methods.

Wave Attenuation Measurements

Wave attenuation is an important source of information for nondestructive testing but the difficulties in getting consistent results make it less popular. A thin crack in a concrete specimen does not produce a measurable change in wave velocity (practically no change in distance), whereas it produces a significant change in the wave amplitude because of the impedance mismatch at the crack boundary. The attenuation of a propagating front through a medium can contain information about its structural integrity. Gaydecki et al. (1992) suggested the use of frequency dependent attenuation to determine the aggregate distribution in a concrete specimen; because aggregates scatter waves differently depending on their size relative to the wavelength. They used an excitation frequency bandwidth in the range of 100 to 500 kHz ($\lambda = 4$ and 0.8 cm, $height/\lambda \approx 2.5$ and 12.5 respectively for a 10 cm specimen) on cylindrical specimens of 50 cm diameter and with lengths varying from 10 to 50 cm. Philippidis and Aggelis (2005) state that the quantity of aggregate significantly affects velocity whereas the size of the aggregate affects attenuation. They concluded this from testing on cubic specimens of 7.3 and 15 cm using a frequency range of 25 kHz to 700 kHz.

Goueygou et al. (2001, 2002) found similar results looking at degradation of concrete covers; where a small velocity change from the degradation (24%) could translate in a tenfold increase in attenuation. They used slabs 30x15 cm in area and 5 cm deep with testing centre frequencies of 1 MHz for P-waves and 0.5 MHz for S-waves ($height/\lambda \approx 11$ mm for P-waves using a velocity of 4500 m/s and $height/\lambda \approx 22$ for S-waves). Chaix et al. (2003) found that wave attenuation could be used in assessing thermal damage in concrete when measuring the response of a scattered signal. They used cylindrical specimens 11 cm in diameter and 21 to 22 cm in length. They induced varying heat damage on concrete specimens and compared the results to measurements on healthy samples using signals from 26 kHz to 2.5 MHz, corresponding to 1 and 100 wavelength per sample height respectively.

Surface Waves

Surface waves have convenient properties for NDT. They require access to only one surface, their penetration depth can be controlled by changing frequency, and they contain most of the wave energy. Popovics et al. (2000) introduced the use of surface waves for detection of surface breaking cracks. They used slabs of 41 x 41 cm in area and 10 cm in depth. Different frequencies (5 to 95 kHz) and their corresponding attenuation are used to determine the depth of a surface breaking crack. Yang et al. (2005) improved this method by introducing a new transducer configuration and the use of the wavelet transform.

Rayleigh waves were used by Goueygou et al. (2002) at high frequencies (0.5 to 1 MHz) to assess chemical damage at the top of concrete surfaces. Their method used the dispersion curve of high frequency surface waves to evaluate the condition. The tests were done on slabs 30x15 cm in area and 5 cm in depth (or 5 to 10 wavelength in depth for P-wave velocity). They find the dispersion curves of surface waves to be useable for field use in evaluating surface condition.

Digital Signal Processing

One of the greatest problem of ultrasonic methods is the difficulty to interpret complex experimental data. Digital signal analysis methods are used to extract more information.

Bilgutay et al. (2001) reviewed several signal analysis methods such as the split spectrum and self-compensating signal transmission. The split spectrum processing uses high frequency measurement over a certain bandwidth to average out the effect of particle scattering in concrete. The self-compensating signal transmission method is analogous to the transmission coefficients proposed by Popovics et al. (2000) and Yang et al. (2005). It uses a source with two receivers to obtain an excitation-independent attenuation measurement.

Kim et al. (2005) suggested the use of various artificial intelligence techniques to assess the strength of concrete samples based on algorithm training of control specimens. The technique monitor features in the ultrasonic signal such as: signal's statistical variance, number of zero crossings, mean frequency, the AR model coefficients and the linear cepstrum coefficient. The latter two parameters are based on spectral statistics. They used cylinders 15 cm in diameter and 30 cm in length along with a 54 kHz centre frequency source. They cast many specimens at varying strength levels for training of the algorithm. Under ideal conditions, they had a 92% success rate in strength categorization of their samples.

Chang and Wang (1997) used an array of transducers to effectively detect a crack in 3 dimensions. By triangulating backscatter echos from different sources at different locations the precise tip of a surface breaking crack can be imaged. They achieved high resolution measurements by using a 300 kHz source. Glushkov et al. (2006) solved a similar problem theoretically, with an improved integral equation method that solves the inverse problem of locating and sizing a crack by measuring the surface movement. The method is computationally cheaper than the traditional travel time methods.

Another detection method was suggested by Hatanaka et al. (2005), they used the wavelet transform for signal processing prior to localizing the flaws. They successfully reconstructed a flaw inside a large slab (90x50 cm in area and 25 cm deep) using a 250 kHz excitation. The signal processing was used solely for feature extraction from the time signal, making flaw reconstruction easier.

2.3 Summary

A review of significant research of wave propagation in infinite half space and layered media was presented. Methods for calculating theoretical time traces at a surface were introduced for both the infinite half space and the layered media case.

Current and upcoming NDE methods in concrete were presented in terms of electromagnetic and acoustic methods, with a strong focus on acoustic techniques. Overview of current research in impact echo, ultrasonic pulse velocity, and emerging ultrasonic methods was also presented. All of the presented research on ultrasonic pulse velocity used specimen with at least one wavelength per specimen height, in accordance with ASTM C597-02. Studies by Sutan and Jaafar (2003), Abo-Qudais (2005), Ohdaira and Masuzawa (2000), used low frequencies (below 100 kHz) to measure velocities on similarly cast and sized specimens to evaluate physical properties. Similarly, Washer et al. (2005), and Prassianakis and Prassianakis (2004) studied UPVM for estimation of Young's modulus but did so using high frequencies (0.5 to 1 MHz). In all of these studies, velocity is expected to change in accordance to a studied parameter, as such velocity differences are not fully evaluated in terms of experimental error. Philippidis and Aggelis (2005) studied the effects of specimen size, transducer location, specimen composition and frequency on velocity values and found that each of these can have an effect on velocity measured.

Chapter 3

Theoretical Background

3.1 Mechanical Waves

This section presents the fundamental theory behind elementary wave propagation in a homogeneous, linear and elastic solid. First, an introduction to body waves is presented followed by a brief description of Rayleigh waves and their properties. Finally, general properties of wave generation, near field effects, and wave attenuation are discussed.

3.1.1 Plane Waves in a Homogeneous Medium

In an infinite homogeneous media, waves are described by the well known Helmholtz equation (Chapman, 2004; Hecht, 2002; Bullen and Bolt, 1985):

$$\nabla^2\psi = V_p^2\frac{\partial^2\psi}{\partial t^2} \tag{3.1}$$

for which the general solution is:

$$\psi = Ae^{i(\mathbf{kx}-\omega t)} \tag{3.2}$$

where ψ is the waveform function, V_p is the wave velocity, A is wave amplitude, t is the time, **k** is the wave vector (direction of propagation, magnitude is wave-number), **x** is the position vector and ω is the angular frequency. Thus, waveforms traveling in a homogeneous infinite media must be a superposition of multiple variations of equation 3.2. It follows from this that each wave front is a harmonic sequence with a period T along the time axis, and wavelength λ along the direction of propagation. The velocity of propagation is given as the ratio of wavelength over period.

$$T = \frac{2\pi}{\omega}, \lambda = \frac{2\pi}{|k|}, V_p = \frac{\lambda}{T} \tag{3.3}$$

Reflection and Refraction across boundaries

Waves traveling across boundaries are reflected and transmitted in varying proportions based on the acoustic impedance on each side of the boundary. The impedance of a material (Z) is given in terms of the density (ρ) and wave velocity (V) by: (Chapman, 2004)

$$Z = \rho V \tag{3.4}$$

The wave energy after crossing the boundary is split between the reflected and transmitted wave according to the reflection and transmission coefficients: (Chapman, 2004)

$$R_{AA} = \left(\frac{Z_B - Z_A}{Z_A + Z_B}\right)^2 \tag{3.5}$$

$$R_{AB} = \left(\frac{2\sqrt{Z_A Z_B}}{Z_A + Z_B}\right)^2 \tag{3.6}$$

where R_{AA} is the reflection coefficient, R_{AB} is the transmission coefficient from material A to material B (figure 3.1a), and Z_A, Z_B are the impedances of materials A and B respectively. The reflection/transmission coefficients represent the ratio of reflected/transmitted energy to the total incident energy. Due to conservation of energy, the sum of R_{AA} and R_{AB} must always be 1.

The transmission and reflection of the wave also affects the directionality of outgoing rays (figure 3.1). The directional behavior of rays is governed by Snell's law; which states that an incoming wave vector (Chapman, 2004):

$$\frac{\mathbf{k}_i}{\omega} = \begin{pmatrix} k_{xi} \\ k_{yi} \end{pmatrix} = \frac{1}{V_A} \begin{pmatrix} \sin\theta_i \\ -\cos\theta_i \end{pmatrix} \tag{3.7}$$

must have reflected wave vector:

$$\frac{\mathbf{k}_r}{\omega} = \begin{pmatrix} k_{xr} \\ k_{yr} \end{pmatrix} = \frac{1}{V_A} \begin{pmatrix} \sin\theta_r \\ \cos\theta_r \end{pmatrix} \tag{3.8}$$

and transmitted wave vector:

$$\frac{\mathbf{k}_t}{\omega} = \begin{pmatrix} k_{xt} \\ k_{yt} \end{pmatrix} = \frac{1}{V_B} \begin{pmatrix} \sin\theta_t \\ -\cos\theta_t \end{pmatrix} \tag{3.9}$$

where $\theta_i, \theta_r, \theta_t$ are the incident, reflected, and transmitted angles from the normal (figure 3.1b). V_A, V_B are the velocities of material A and B respectively. In order to satisfy continuity, the wave must have the same frequency and wavelength along the boundary (x axis). Thus, the x component of the wave vector must be equal for all wave types. Applying this condition to equations 3.7, 3.8, and 3.9 it is obtained that:

$$\theta_i = \theta_r \tag{3.10}$$

$$V_B \sin\theta_i = V_A \sin\theta_t \tag{3.11}$$

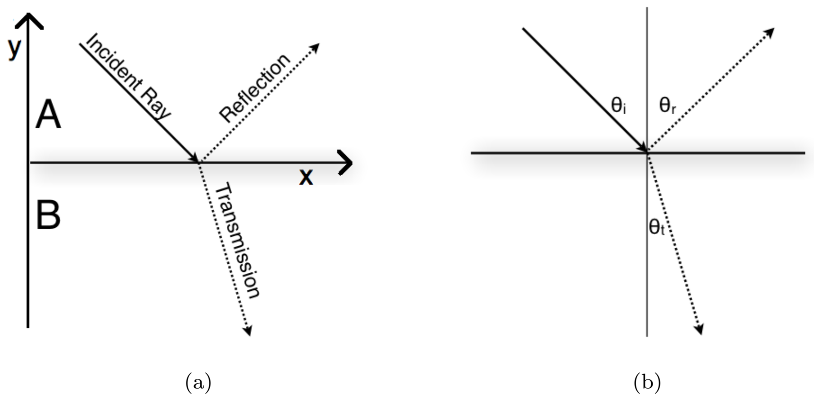


Figure 3.1: Representation of Snell's Law. (a) Reflection/Transmission, (b) Angles

3.1.2 Elastic Waves

In an elastic solid, waves propagate according to the material's elastic properties. Waves still follow the basic principles of plane wave propagation and Snell's law but are described in terms of elastic properties of the material.

Body Waves

In a homogeneous, isotropic and perfectly elastic solid with no body forces the equation of motion of the waves is described by: (Bullen and Bolt, 1985)

$$\rho \frac{\partial^2 u_i}{\partial t^2} = (\lambda_l + \mu_l) \frac{\partial \epsilon_v}{\partial x_i} + \mu \nabla^2 u_i \tag{3.12}$$

where $u_i = u_1, u_2, u_3$ is the displacement in all three principal axis (x,y and z), x_i is the coordinate of the point as compared to a fixed set of rectangular coordinates, ϵ_v is the volume expansion given by $\epsilon_v = \frac{\partial u_1}{\partial x_1} + \frac{\partial u_2}{\partial x_2} + \frac{\partial u_3}{\partial x_3}$, and λ_l, μ_l are the Lamé constants, which are related to the Young's modulus (E), and the shear modulus (G) by:

$$\lambda_l = \frac{\nu E}{(1+\nu)(1-2\nu)} \tag{3.13}$$

$$\mu_l = G = \frac{E}{2(1+\nu)} \tag{3.14}$$

where ν is the Poisson ratio.

To isolate wave propagation, two mathematical tools are used: the divergence and the curl. The divergence operation, $\nabla \cdot$, which represents the flow through an enclosing area. Waveforms expanding as described by the divergence operation propagate in the strain direction. Conversely, the curl operator, $\nabla \times$, represents the rate of rotation; the movement is perpendicular to the flow. Taking the divergence of equation 3.12, the compressional wave, or P-wave, equation is obtained:

$$\frac{\partial^2 \epsilon_v}{\partial t^2} = \frac{\lambda_l + 2\mu_l}{\rho} \nabla^2 \epsilon_v \tag{3.15}$$

where, comparing to equation 3.1, the corresponding velocity is:

$$V_p \quad = \quad \sqrt{\frac{\lambda + 2\mu}{\rho}} = \sqrt{\frac{E(1-\nu)}{\rho(1+\nu)(1-2\nu)}} \tag{3.16}$$

Similarly, using the curl operation on both sides of equation 3.12 , and simplifying, the shear wave, or S-wave, equation is obtained:

$$\frac{\partial^2}{\partial t^2}\nabla \times u_i \quad = \quad \frac{\mu}{\rho}\nabla^2 \nabla \times u_i \tag{3.17}$$

which leads to a shear wave velocity:

$$V_s \quad = \quad \sqrt{\frac{\mu}{\rho}} = \sqrt{\frac{E}{2(1+\nu)}} \tag{3.18}$$

Rayleigh Waves

Rayleigh waves are a special case of surface waves that occur at an interface. The more general case of surface waves exists for any interface between two materials. The surface wave exists on the boundary (x_1-x_2 plane) and must be continuous across it (figure 3.2). For a plane wave traveling in the x_1 direction crossing the boundary; any derivative towards x_2 vanishes. The displacement of the particles is given by (Bullen and Bolt, 1985):

$$u_1 = \frac{\partial \phi}{\partial x_1} + \frac{\partial \psi}{\partial x_3} \qquad u_3 = \frac{\partial \phi}{\partial x_3} - \frac{\partial \psi}{\partial x_1} \tag{3.19}$$

where ϕ, ψ are potential functions defined as:

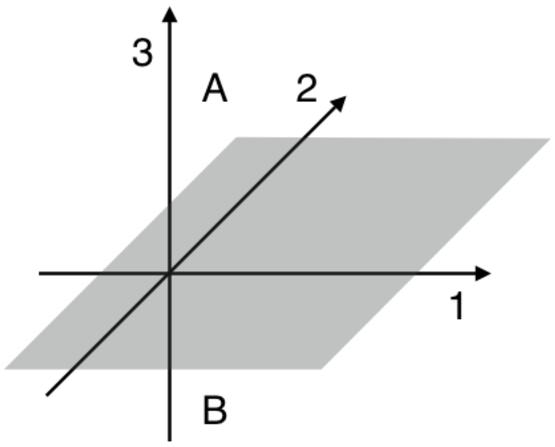


Figure 3.2: Surface wave coordinate system

$$\nabla^2\phi = \theta \tag{3.20}$$

$$\nabla^2\psi = \frac{\partial u_1}{\partial x_3} - \frac{\partial u_1}{\partial x_1} \tag{3.21}$$

The functions ϕ, ψ are related to P waves and vertical S waves respectively. The solutions of u_1, u_3 presented in equation 3.19 satisfy relation 3.12 if the relations:

$$\begin{aligned}
\frac{\partial^2\phi}{\partial t^2} &= V_p^2\nabla^2\phi \\
\frac{\partial^2\psi}{\partial t^2} &= V_s^2\nabla^2\psi \\
\frac{\partial^2 u_2}{\partial t^2} &= V_s^2\nabla^2 u_2
\end{aligned} \tag{3.22}$$

are valid for material A. Similarly, an identical set with V_p', V_s' must be valid for material B. These relations are simply wave equations and their solutions are given by:

$$\phi = f(x_3)e^{i\kappa(x_1-V_rt)} \tag{3.23}$$

$$\psi = g(x_3)e^{i\kappa(x_1-V_rt)} \tag{3.24}$$

$$u_2 = h(x_3)e^{i\kappa(x_1-V_rt)} \tag{3.25}$$

where V_r is the surface wave velocity, and f, g, h are unknown functions in material A. A similar set of relations, using f', g', h' is also valid in material B. These solutions are simple plane waves with unknown amplitude functions. A solution in material A (at boundary) is found by substituting equation 3.25 into 3.22:

$$\left[\frac{d^2}{dx_3^2} + \kappa^2\left(\frac{V_r^2}{V_s^2} - 1\right)\right]h = 0 \tag{3.26}$$

which is a second order differential equation with solution for h:

$$h(x_3) = Ce^{-i\kappa\sqrt{\frac{V_r^2}{V_s^2}-1}x_3} + Fe^{i\kappa\sqrt{\frac{V_r^2}{V_s^2}-1}x_3} \tag{3.27}$$

where C, F are integration constants. This procedure is easily repeated for functions f, g, f', g'. For the solution to be physical, ϕ, ψ, u_2 must vanish away from the boundary. This implies the exponent must be negative in x_3 for $x_3 > 0$ and positive for $x_3 < 0$. Thus, $\sqrt{\frac{V_r^2}{V_s^2} - 1}$ must be imaginary. Furthermore, the positive exponent must vanish, otherwise the solution diverges, simplifying the solutions as:

$$\phi = Ae^{i\kappa\left(-\sqrt{\frac{V_r^2}{V_p^2}-1}x_3\right)}e^{i\kappa(x_1-V_rt)} \tag{3.28}$$

$$\psi = Be^{i\kappa\left(-\sqrt{\frac{V_r^2}{V_s^2}-1}x_3\right)}e^{i\kappa(x_1-V_rt)} \tag{3.29}$$

$$u_2 = Ce^{i\kappa\left(-\sqrt{\frac{V_r^2}{V_s^2}-1}x_3\right)}e^{i\kappa(x_1-V_rt)} \tag{3.30}$$

where A, B, C are constants, and $V_r < V_s < V_p$ such that the roots are always imaginary. Similar solutions exists, with different constants and velocities, for material B. The problem of solving for the constants can then be approached using the continuity of displacements and stresses at the boundary.

The case of interest being Rayleigh waves on a half space, the constants A', B', C' are zero if material B is a void. Therefore, only the first boundary condition is required to find a solution. Thus, continuity at the boundary implies $u_1 = u_2 = u_3 = 0$ and, by using solutions for ϕ, ψ, it is obtained that:

$$A = B\sqrt{\frac{V_r^2}{V_s^2} - 1} \tag{3.31}$$

$$B = -A\sqrt{\frac{V_r^2}{V_p^2} - 1} \tag{3.32}$$

$$C = 0 \tag{3.33}$$

and solving for A, B leads to:

$$\left(2 - \frac{V_r^4}{V_s^4}\right)^2 - 4\left(1 - \frac{V_r^2}{V_p^2}\right)\left(1 - \frac{V_r^2}{V_s^2}\right) = 0 \tag{3.34}$$

further simplifying gives:

$$\frac{V_r^6}{V_s^6} - 8\frac{V_r^4}{V_s^4} + V_r^2\left(\frac{24}{V_s^2} - \frac{16}{V_p^2}\right) - 16\left(1 - \frac{V_s^2}{V_p^2}\right) \quad = \quad 0 \tag{3.35}$$

which can be used to solve for V_r. The approximate solution to this operation is given by Vinh and Ogden (2004) and Vinh and Malischewsky (2006) as:

$$\frac{V_r}{V_s} \quad \approx \quad 0.874 + 0.196\nu - 0.043\nu^2 - 0.055\nu^3 \tag{3.36}$$

for poisson ratio $\nu \in [-1, 0.5]$.

3.2 Signal Analysis

Wave signals are often difficult to interpret unless the geometry of the problem is simple (e.g. homogeneous half-space) because the effects of reflections, attenuation and scattering make a signal difficult to analyze. Several methods have been suggested over the years to extract clear signals from complex time domain traces, each with their own quirks and limitations. Four common transform methods are covered next: Fourier transforms, Frequency wave-number transform, convolution, and Hilbert transform.

3.2.1 Fourier Transform

The Fourier transform is one of the most used mathematical transforms in signal analysis and also serves as the basis of many other linear transforms. The properties of this transform are covered in the literature (Hecht, 2002; Hatton et al., 1986; Robinson and Tritel, 1980). This section briefly covers the basics of the Fourier transform and discrete Fourier transform.

Fourier Transform

The Fourier transform is based on the Fourier series, which expresses any periodic signal as an infinite series of sines and cosines of increasing frequencies. A Fourier series is defined by:

$$x(t) \quad = \quad a_0 + 2\sum_{k=1}^{\infty}\left(a_k \cos\left(\frac{2\pi kt}{T}\right) + b_k \sin\left(\frac{2\pi kt}{T}\right)\right) \tag{3.37}$$

where a_k, b_k are constants defined by:

$$a_k = \frac{1}{T}\int_0^T x(t)\cos\left(\frac{2\pi kt}{T}\right)dt, \qquad k \geq 0 \tag{3.38}$$

$$b_k = \frac{1}{T}\int_0^T x(t)\sin\left(\frac{2\pi kt}{T}\right)dt, \qquad k > 0 \tag{3.39}$$

If T becomes infinite the Fourier transform is obtained, which is a special case for a signal with an infinite period (i.e. non-periodic). Using equation 3.37-3.39 with Euler's formula ($e^{i\theta} = \cos\theta + i\sin\theta$) the Fourier transform can be expressed as:

$$\bar{x}(f) = \int_{-\infty}^{\infty} x(t)e^{-i2\pi ft}dt \tag{3.40}$$

with the inverse transform:

$$x(t) = \int_{-\infty}^{\infty} \bar{x}(f)e^{i2\pi ft}df \tag{3.41}$$

Discrete Fourier Transform

The Fourier transform itself is rarely used in signal analysis as the discrete Fourier transform (DFT) is much better suited for numerical use. The DFT is, as the name implies, a discretized version of the Fourier transform used to eliminate the integration term.

For a signal $x \in [0, T]$ sampled at interval Δt, then approximating the integral by a Riemann sum equation 3.40 gives:

$$X_k = \frac{1}{N}\sum_{r=0}^{N-1} x_r e^{-i2\pi kr/N} \tag{3.42}$$

where $N = \frac{T}{\Delta t}$, r is an integer in the range (0, ..., N-1), and x_r is the value of the signal at index r. Similarly, the inverse DFT can be found as:

$$x_r = \frac{1}{N}\sum_{k=0}^{N-1} X_k e^{i2\pi kr/N} \tag{3.43}$$

Symmetry Properties of Real Signals

The periodic nature of the Fourier series implies that X_k has period N such that $X_{N+i} = X_i$, where i is an arbitrary integer $i \in [1, N-1]$. Taking the complex conjugate of X_k, assuming x_r is real, the symmetry properties can be studied further.

$$\begin{aligned} X_k^* &= \frac{1}{N}\sum_{r=0}^{N-1} x_r e^{i2\pi kr/N} \\ &= \frac{1}{N}\sum_{r=0}^{N-1} x_r e^{-i2\pi(-k)r/N} \end{aligned} \tag{3.44}$$

This relation implies that

$$X_k^* = X_{-k} \tag{3.45}$$

and, consequently,

$$X_0^* = \bar{X_N} = X_0 \tag{3.46}$$

thus, X_0 and all its periodic equivalent are real values. Using periodicity and equation 3.45 it follows that:

$$X_{N/2}^* = X_{-N/2} = X_{N/2} \tag{3.47}$$

thus, $X_{N/2}$ and all its periodic equivalents are also real.

Discretization Effects

As the periodicity property implies, X_k has unique values only in the range $k = 0, 1, \ldots, N-1$. Furthermore, this range is reduced by the symmetry property to $k = 0, 1 \ldots, N/2$. Thus, only frequencies up to $k = N/2$ can be represented by the DFT. This correspond to angular frequencies:

$$|\omega| \leq \frac{2\pi N/2}{T} = \frac{\pi}{\Delta t} \tag{3.48}$$

or frequencies:

$$|f| \leq \frac{N/2}{T} = \frac{1}{2\Delta t} \tag{3.49}$$

This maximum upper frequency is called the Nyquist frequency. Any frequencies present in the original signals that are higher than the Nyquist frequency are not accurately represented and instead they are aliased as a lower frequency.

An aliased frequency is represented in the lower frequency range as wrapped around the $N/2$ frequency. For example, a frequency corresponding to $1.25f_N$, where f_N is the Nyquist frequency, is seen as:

$$f = f_N - (1.25f_N - f_N) = 0.75f_N \tag{3.50}$$

Thus, to have a proper representation of the signal below the Nyquist frequency, the higher frequencies must be filtered from the signal prior to transforming it. Alternatively, a higher sampling rate can be used if possible.

The Nyquist frequency range is a bandwidth limit, not an absolute range. If the frequency content is known prior to processing, and contains no frequencies outside that range it is possible to obtain an accurate spectrum of high frequencies even at low sampling rates.

3.2.2 Frequency Wave-number Transform

The frequency wave-number plot is a two dimensional Fourier transform performed over time (x-axis) and distance (y-axis) (figure 3.3). The two dimensional DFT has properties similar to the one dimensional DFT along each of its axes and it is given by:

$$\bar{f}_{pl} = \frac{1}{\sqrt{MN}} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f_{mn} e^{-j2\pi\left(p\frac{m}{M}+l\frac{n}{N}\right)} \tag{3.51}$$

with the reverse transform given by:

$$f_{mn} = \frac{1}{\sqrt{MN}} \sum_{p=0}^{M-1} \sum_{l=0}^{N-1} \bar{f}_{pl} e^{j2\pi\left(p\frac{m}{M}+l\frac{n}{N}\right)} \tag{3.52}$$

where p, l, m, n are integer indices of the discrete signal. A f-k plot is a useful tool in studying wave propagation along a surface by giving information on wave velocity, wave directionality and dispersion. A propagating wave front can be identified as a series of peaks on the f-k plot as shown in figure 3.4. This works best using signals with a broad frequency band as narrow band signals will have few peaks to use (figure 3.6).

The group wave velocity in a certain frequency range can be determined by taking the slope at a point:

$$V = \frac{\Delta\omega}{\Delta k} \tag{3.53}$$

where the slope is taken as being the linear trace along two or more consecutive peaks.

The wave directionality can be obtained from the location of the peaks. Peaks in the positive frequency domain represent waves propagating in the positive x direction whereas peaks in negative frequency domain propagate backwards. This property is useful for separating transmitted and reflected waves.

The dispersion curve is obtained from the curve formed by a series of consecutive peaks. This curve in the frequency-wavenumber domain defines the phase velocity for each frequency component ($V_{ph} = \omega/k$). A visual representation of the f-k plots is given in figure 3.3 to 3.4 for different input sources.

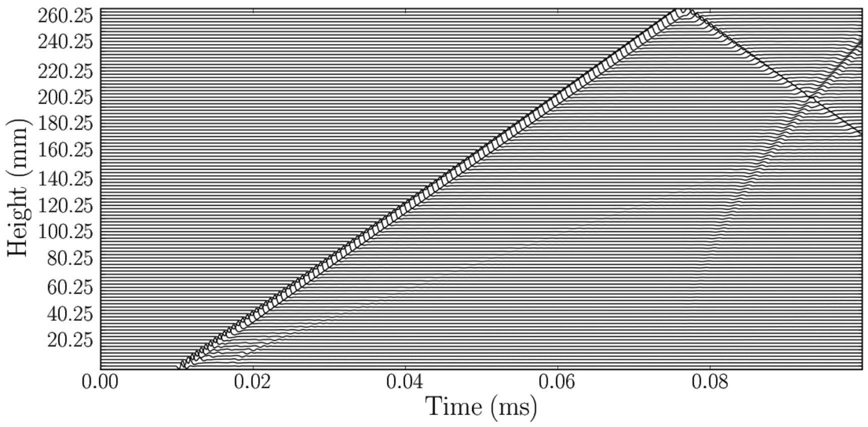


Figure 3.3: Sample of seismic trace

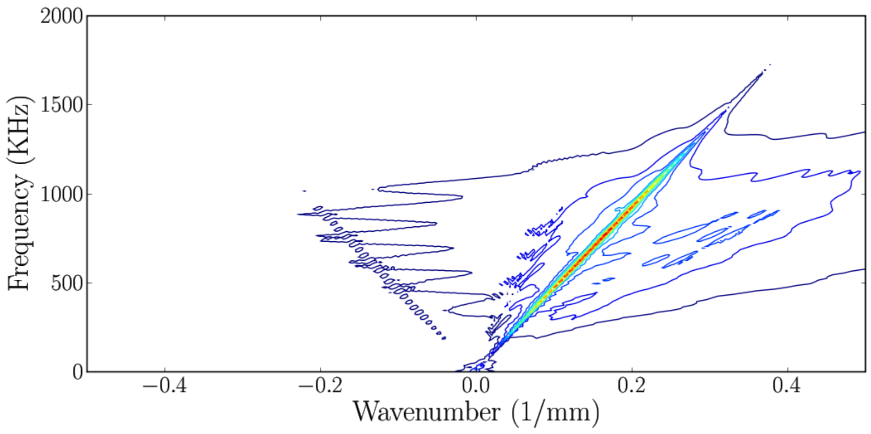
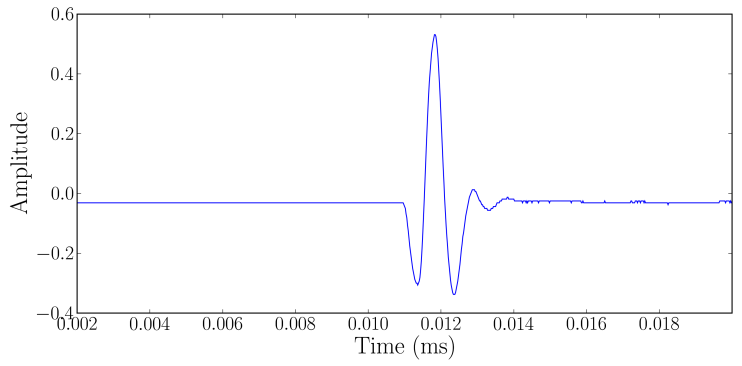
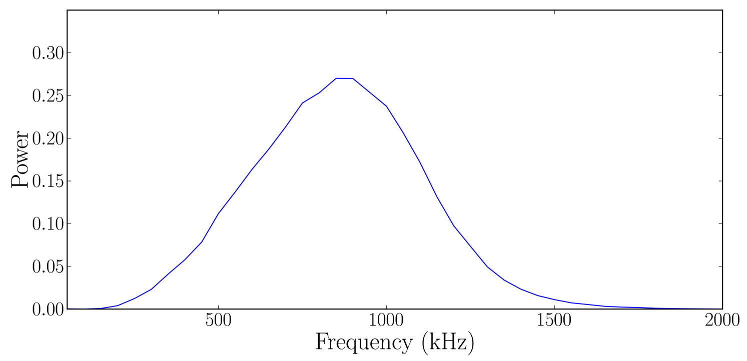


Figure 3.4: Sample f-k Plot for broad band signals shown in 3.3. The right wave (positive incline) is the longitudinal wave propagating forward while the left wave (negative incline) is the reflected wave. The velocity of a wave is given as the slope of a line on a f-k plot.



(a)



(b)

Figure 3.5: Broad Band Input Source (a) Time trace, (b) Magnitude of Fourier spectrum

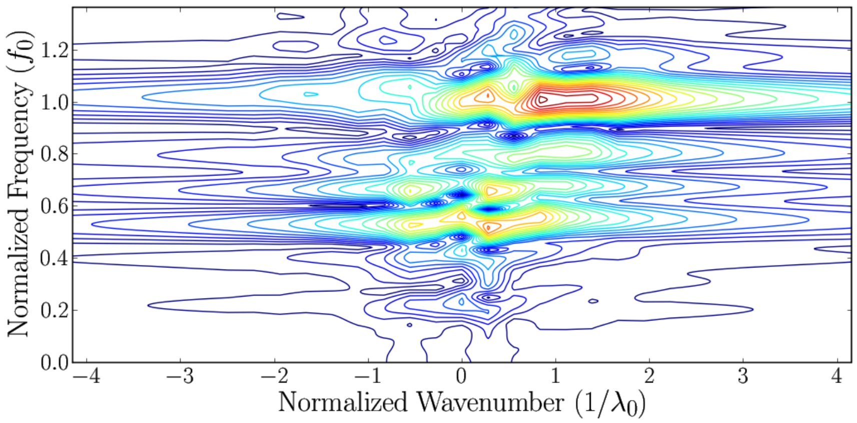
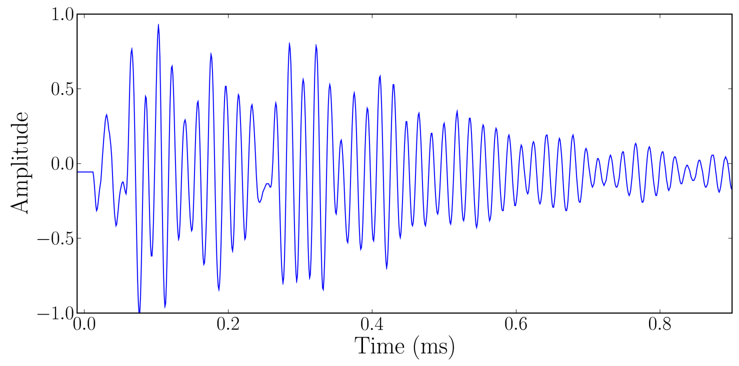
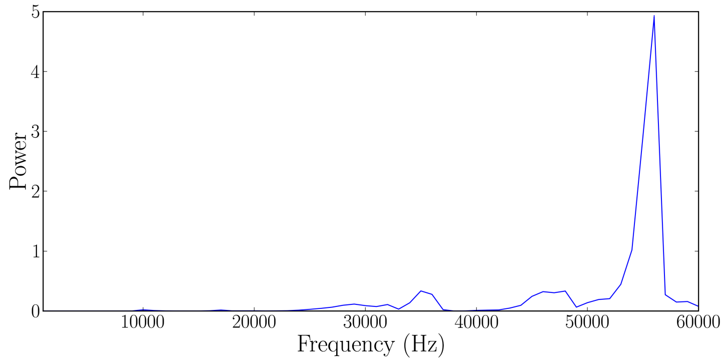


Figure 3.6: Sample f-k Plot for Narrow Band Input



(a)



(b)

Figure 3.7: Narrow Band Input Source (a) Time trace, (b) Magnitude of Fourier spectrum

3.2.3 Convolution

The convolution is a mathematical representation of the coupling of linear systems (Hatton et al., 1986). This operation couples one signal with another to produce a third convoluted signal which is a combination of both. The operation is especially useful as it is reversible if one of the two original signal is known. The convolution of two signal f(t) and g(t) is represented as:

$$a(t) = h(t) \star g(t) \tag{3.54}$$

where $\star$ is defined as the convolution operator. For continuous functions, the convolution is defined as:

$$h(t) \star g(t) = \int_{t_0}^{t_e} h(\tau)g(t-\tau)d\tau \tag{3.55}$$

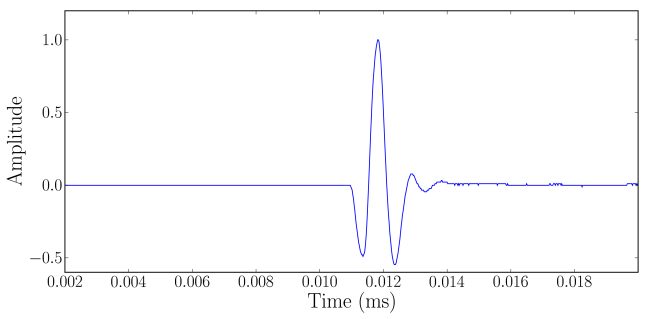
where t_0, t_e is the window in which the signals are defined. For an infinite signal $t_0 = -\infty$ and $t_e = \infty$. The discrete convolution can be directly derived from equation 3.55 to give:

$$a_k = \sum_{j=0}^{L_w} h_j g_{k-j} \tag{3.56}$$

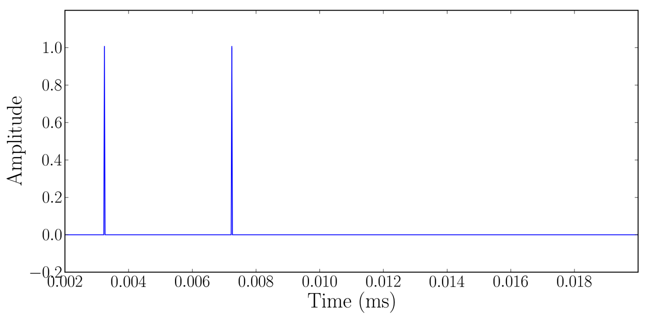
where $j = 0, 1, \ldots, L_w$, $j = 0, 1, \ldots, L_e$. This simple discretized method is computationally expensive and rarely used in practice. Instead, the fast convolution algorithm is used. Fast convolution uses the convolution theorem:

$$FT(h \star g) = FT(h)FT(g) \tag{3.57}$$

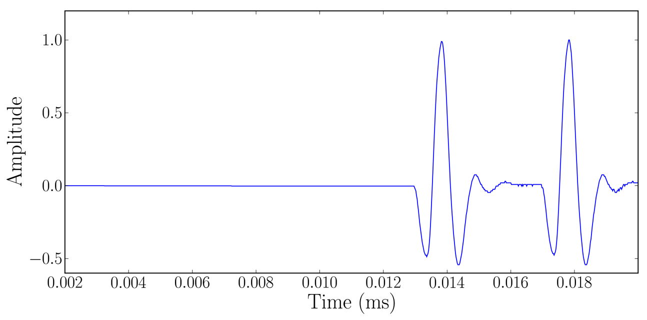
where FT denotes the Fourier transform operation. Thus, the convolution is obtained by taking the transform of f and g, multiplying them, and finally performing the inverse transform. Due to the efficiency of the fast Fourier transform algorithm, the convolution is computed more efficiently in the frequency domain.



(a)



(b)



(c)

Figure 3.8: Visual Representation of Convolution (a) signal wavelet , f, (b) Impulse Response, g, and (c) convolved response, a.

3.2.4 Hilbert Transform

The Hilbert transform is used to obtain the analytic representation of a real function. The analytic representation is a complex trace with the original signal as the real part and its Hilbert transform as the imaginary part (Taner et al., 1979).

The Hilbert transform ($\widehat{s}(t)$) can be obtained by the convolution of a signal ($s(t)$) with the impulse response of a Hilbert filter:

$$h_t(t) = \frac{1}{\pi t} \tag{3.58}$$

then

$$\widehat{s}(t) = h_t(t) \star s(t) \tag{3.59}$$

The analytical signal is mathematically represented as:

$$y(t) = s(t) + i\widehat{s}(t) \tag{3.60}$$

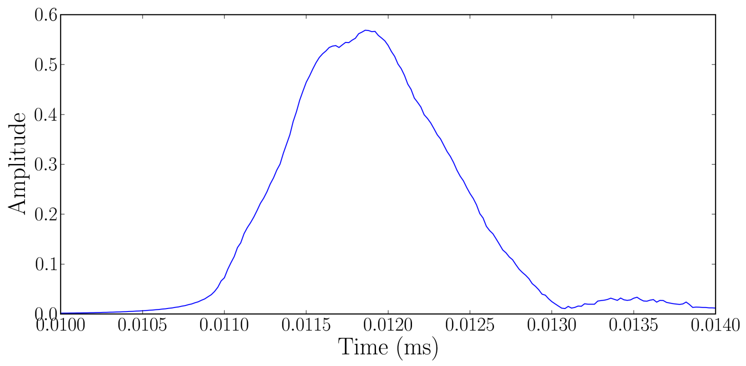
This complex trace can be used to extract time-dependent properties from the original signal. The three principal features are amplitude, phase, and angular frequency:

$$A(t) = \sqrt{s^2(t) + \widehat{s^2}(t)} \tag{3.61}$$

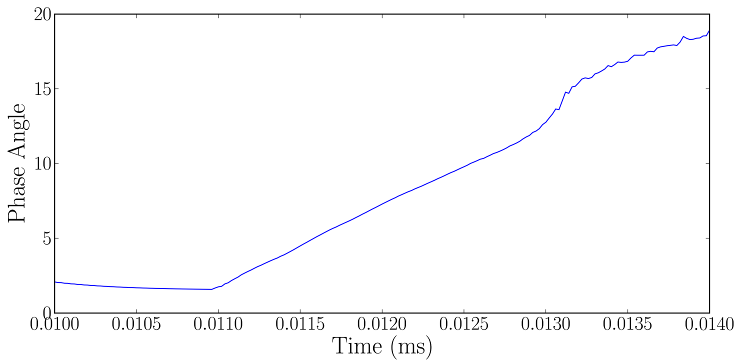
$$\theta(t) = \tan^{-1}\left(\frac{\widehat{s}(t)}{s(t)}\right) \tag{3.62}$$

$$\omega(t) = \frac{d\theta(t)}{dt} \tag{3.63}$$

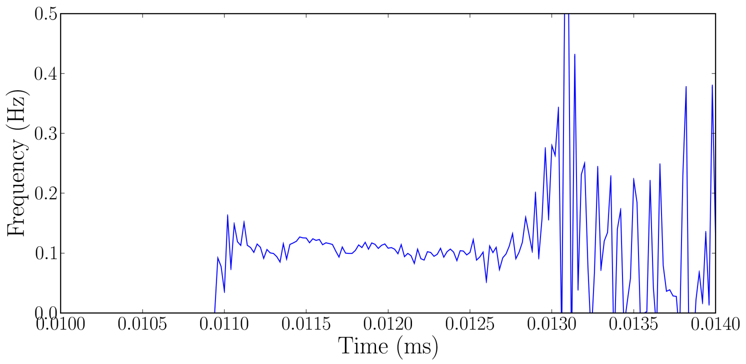
These properties are demonstrated in figure 3.9 for the time signal shown in figure 3.8a. The amplitude, A(t), represents the envelope of the function, $\theta(t)$ is the immediate phase and $\omega(t)$ is the immediate frequency of the signal.



(a)



(b)



(c)

Figure 3.9: Analytical Signal of time trace shown in Figure 3.8a. (a) Amplitude, (b) Phase and (c) Frequency

3.3 Transducer Characterization and Design

It is often overlooked in nondestructive tests of civil engineering materials that the transmitter has a particular acoustic field that depends on its mechanical, and electrical characteristics.

Transducer specific acoustic fields can have significant consequences when using pulse-echo methods, attenuation measurements, or even frequency domain comparisons (Fink and Cardoso, 1984). Depending on the type of transducer used, defects react differently because of the different pressure field being scattered. Precise descriptions of acoustic fields vary by transducer model. This research focuses on the flat uniformly excited transducer, or piston transducer, which has well known theoretical solutions.

3.3.1 Acoustic Field of a Piston Transducer

The exact solution for the acoustic field of a piston transducer can be found provided the impulse function is known. Using current methods it is possible to find the impulse function of arbitrary piston transducer shapes using discretization (Jenden and Svendsen, 1992). Closed-form solutions exist for simple shapes such as circles and rectangles and any combination of these shapes (Lockwood and Willette, 1973). Using superposition of the previous models it is also possible to simulate non-uniformly excited surfaces.

The following approach can be taken to solve for the pressure field of a transducer. Using a time domain Green's function approach, Lockwood and Willette (1973) find the frequency domain pressure distribution to be:

$$\bar{p}(x,y,\omega) = -\bar{\psi}(w)\int_s \frac{e^{-j\omega r/V_p}}{2\pi r}dS \tag{3.64}$$

$$r^2 = x^2 + y^2 \tag{3.65}$$

where ω is the angular frequency of vibration, $\bar{\psi}(w)$ is the Fourier transform of the pressure function on the transducer's face, r is the distance from the point of interest to a point on the transducer's surface, s and dS represent an integration over the entire transducer's surface and V_p is the speed of sound in the system.

Within the assumption of a linear system, the transfer function and the impulse response of the system are given by:

$$\begin{aligned} \bar{h}_i(x,y,\omega) &= -\int_s \frac{e^{-j\omega r/V_p}}{2\pi r} dS & (3.66) \\ h_i(x,y,t) &= -\int_s \frac{\delta(t - r/V_p)}{2\pi r} dS & (3.67) \end{aligned}$$

where the impulse response, $h_i(x, y, t)$ is taken to be the direct inverse Fourier transform of the transfer function $\bar{h_i}(x, y, \omega)$ and δ is the Dirac's delta function. The pressure can be given as:

$$\bar{p}(x,y,\omega) = \bar{\psi}(\omega)\bar{h_i}(x,y,\omega) \qquad (3.68)$$

or, by the convolution theorem:

$$p(x,y,t) = h_i(x,y,t) \star \psi(t) \qquad (3.69)$$

Hence, assuming that the time-varying pressure on the transducer, $\psi(t)$, is known the pressure at any point can be computed with knowledge of the impulse response. The impulse response function contains all information relating to the transducer shape and size. The circular piston transducer is of most interest as it was used for all the tests performed in this work. Lockwood and Willette (1973) cites the solution of pressure wave for the circular piston (figure 3.10) as follows:

For the case $y < a$

$$\frac{1}{V_p}h(x,y,t) = \begin{cases} 0, & t < t_1 \\ 1, & t_1 < t < t_2 \\ \frac{1}{\pi}\cos^{-1}\left(\frac{V_p^2t^2-x^2+y^2-a^2}{2y(V_p^2t^2-x^2)^{1/2}}\right), & t_2 < t < t_3 \\ 0, & t_3 < t \end{cases} \tag{3.70}$$

For the case $y \geq a$

$$\frac{1}{V_p}h(x,y,t) = \begin{cases} 0, & t < t_2 \\ \frac{1}{\pi}\cos^{-1}\left(\frac{V_p^2t^2-x^2+y^2-a^2}{2y(V_p^2t^2-x^2)^{1/2}}\right), & t_2 < t < t_3 \\ 0, & t_3 < t \end{cases} \tag{3.71}$$

For time limits:

$$t_1 = \frac{x}{V_p} \tag{3.72}$$

$$t_2 = \frac{1}{V_p}\left[x^2+(y-a)^2\right]^{1/2} \tag{3.73}$$

$$t_3 = \frac{1}{V_p}\left[x^2+(y+a)^2\right]^{1/2} \tag{3.74}$$

where t is the time of interest, t_1, t_2 and t_3 represent the shortest travel time from the transducer to the point of interest, and from the edges of the transducer to the point of interest, respectively, a is the transducer radius, x, y represent location from transducer as shown in figure 3.10 and V_p is the pressure wave velocity. A visual representation of this impulse response is shown in figure 3.11.

The pressure is computed using a computer program developed in this work to evaluate equation 3.69 numerically (Appendix C.10). A sample output can be seen in Figure 3.12.

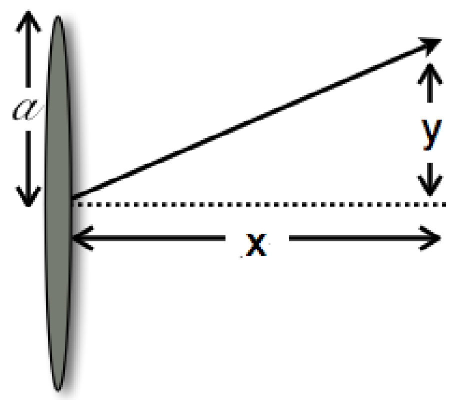


Figure 3.10: Circular piston transducer coordinate system

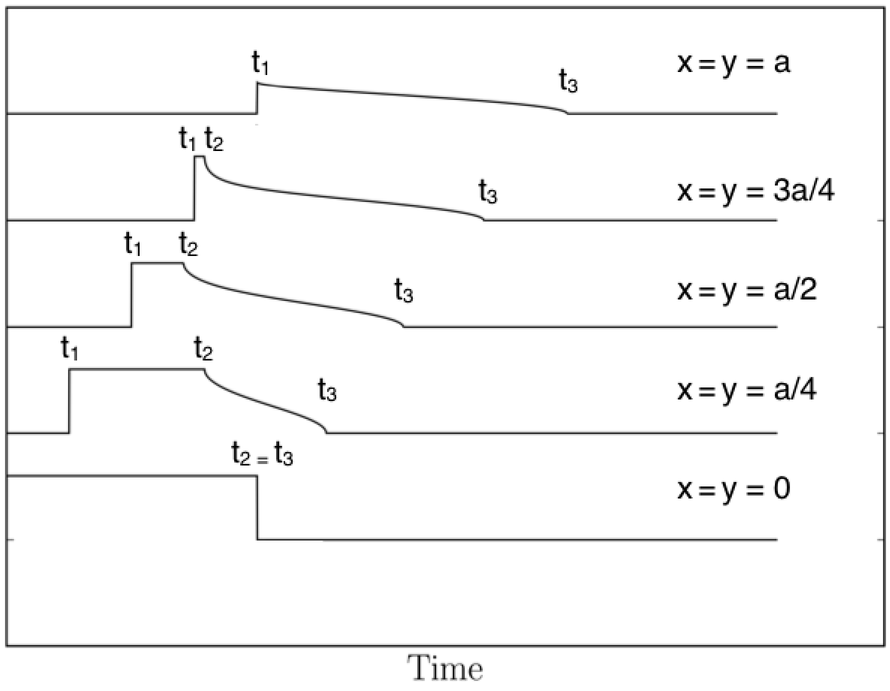
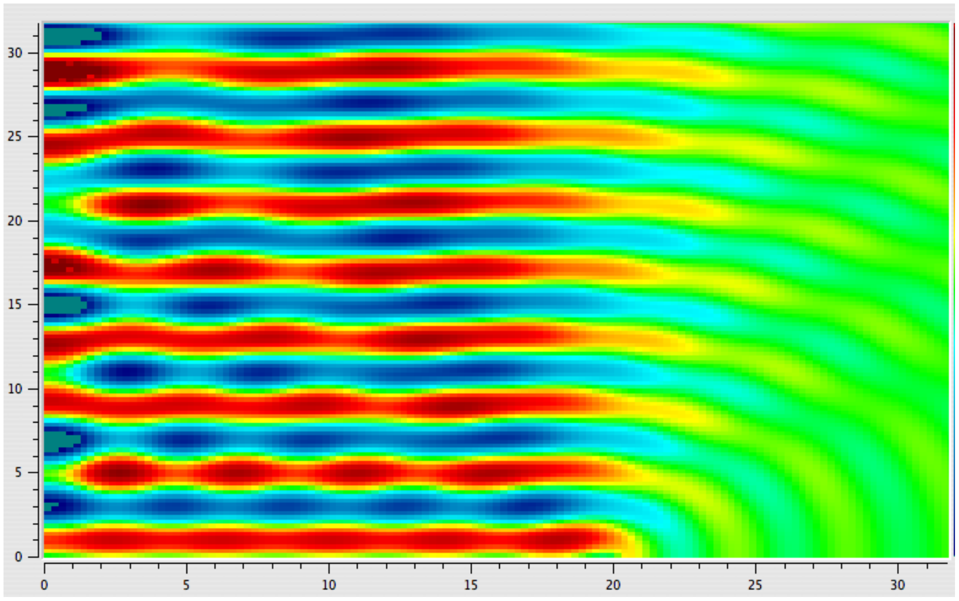
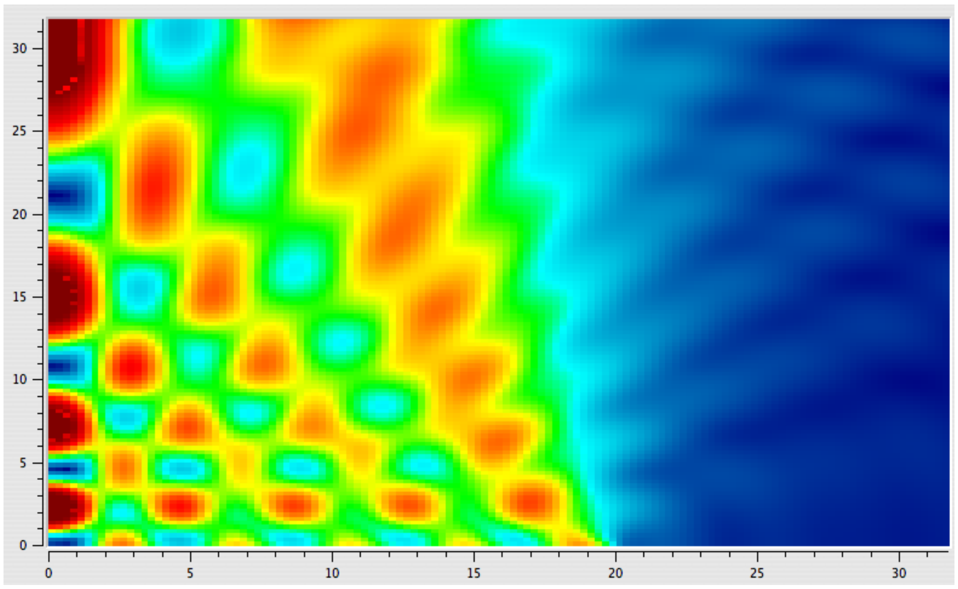


Figure 3.11: Impulse response of a circular transducer at selected locations



(a)



(b)

Figure 3.12: Snapshot of near-field acoustic field for a circular piston assuming a velocity V_p of 1 m/s and a ratio $a/\lambda = 5$ from an 81 s sine impulse. (a) Pressure, (b) Envelope of pressure

3.4 Numerical Simulations

The use of numerical methods to model wave propagation in NDT tests is very useful to obtain a better understanding of a system and to extrapolate laboratory results.

As the propagation through solid materials includes shear waves, the most appropriate methods available are finite element or finite differences analysis.

The finite difference method is a direct approximation of the equation of motion through a Taylor expansion of the displacement function that leads to a discretization of the differentials. The method is significantly faster than finite element methods but is more prone to numerical errors due to discretization effects and requires a minimum discretized step size to prevent numerical instabilities (Bathe, 1982). Discretization effects occur from a non-zero step size in the definition of a differential.

$$f'(x) = \lim_{x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x} \tag{3.75}$$

The finite element method, consists of transforming the system into a finite mesh of elements with multiple nodes each with their own characteristics and movements. A system can be represented by a series of matrices in terms of mass, stiffness and damping. The solution given in this method is not directly the differential equations but rather the estimation of equilibrium of the system given the loading conditions. Thus, finite elements rely on an estimation of the solution rather than an estimation of the differential equation. Generally, finite element analysis is slower than finite differences as each steps may require large matrix operations.

3.4.1 The Finite Element Method

The equilibrium condition of any linear, elastic solid in a static condition having surface traction, body forces, and concentrated forces can be represented as:

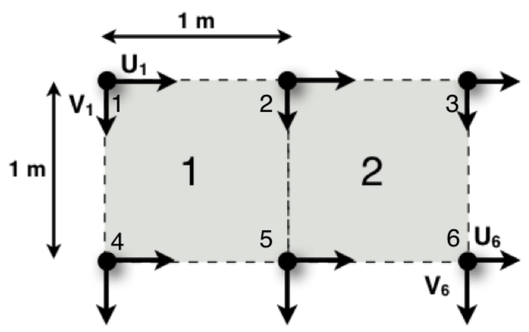


Figure 3.13: Two Four-node elements structure

$$M\ddot{U} + C\dot{U} + KU = R \tag{3.76}$$

where M is the mass matrix, C is damping matrix, K is stiffness matrix, U is displacement vector, $\ddot{U}$ is acceleration vector, $\dot{U}$ is velocity vector and R is the external load vector.

The mass and stiffness matrices are built based on the physical conditions of the problem and are constructed first. The damping matrix is not easy to determine because damping has to be evaluated experimentalfly and depends on a number of parameters such as strain, frequency, wave velocity, and boundary conditions, which are difficult to evaluate.

The damping matrix is often constructed using a combination of the stiffness and mass matrix, called Rayleigh damping, to generate a set of uncoupled equations. Other methods can also be used to approximate different damping types.

Upon constructing all matrices, equation 3.76 is then transformed into a series of uncoupled differential equations of second order. These are called the static solutions as they represent only one point in time.

Assembling Mass and Stiffness Matrices

The construction of the stiffness and mass matrices is procedural provided the geometry of the specimen is fully known. To be accurate during element integration, the discretized elements must properly account for changes between nodes. This can be achieved by building the displacement interpolation matrix, $\mathbf{H}^{(m)}(x, y, z)$, which provides a continuous variation between connected nodal points for an element m (Bathe, 1982):

$$\begin{bmatrix} u^{(m)}(x, y, z) \\ v^{(m)}(x, y, z) \\ w^{(m)}(x, y, z) \end{bmatrix} = \mathbf{H}\widehat{\mathbf{U}} \tag{3.77}$$

where $\widehat{\mathbf{U}}$ is a vector of length 3N containing the displacement of each nodal point in all directions (U,V,W). The coordinate system is chosen to be convenient and does not need to be orthogonal. In the following example, the origin is defined as the centre of an element. Using the two four-nodes elements system represented in figure 3.13, the following interpolation matrices are defined:

$$\mathbf{H}^{(1)} = \begin{bmatrix} \mathbf{H}_0^{(1)} & \mathbf{H}_1^{(1)} \end{bmatrix} \tag{3.78}$$

$$\mathbf{H}^{(2)} = \begin{bmatrix} \mathbf{H}_0^{(2)} & \mathbf{H}_1^{(2)} \end{bmatrix} \tag{3.79}$$

with:

$$\begin{aligned}
\mathbf{H}_0^{(1)} &= \frac{1}{4}\begin{bmatrix} (1-2x)(1+2y) & (1+2x)(1+2y) & 0 & (1-2x)(1-2y) & (1+2x)(1-2y) & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\
\mathbf{H}_1^{(1)} &= \frac{1}{4}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ (1-2x)(1+2y) & (1+2x)(1+2y) & 0 & (1-2x)(1-2y) & (1+2x)(1-2y) & 0 \end{bmatrix} \\
\mathbf{H}_0^{(2)} &= \frac{1}{4}\begin{bmatrix} 0 & (1-2x)(1+2y) & (1+2x)(1+2y) & 0 & (1-2x)(1-2y) & (1+2x)(1-2y) \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\
\mathbf{H}_1^{(2)} &= \frac{1}{4}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & (1-2x)(1+2y) & (1+2x)(1+2y) & 0 & (1-2x)(1-2y) & (1+2x)(1-2y) \end{bmatrix}
\end{aligned}$$

where x, y are the coordinates from the centre of the element investigated. These matrices are set for a vector:

$$\widehat{\mathbf{U}} = \begin{bmatrix} U_1 & U_2 & U_3 & U_4 & U_5 & U_6 & V_1 & V_2 & V_3 & V_4 & V_5 & V_6 \end{bmatrix} \tag{3.80}$$

The significance of the $H^{(m)}$ vector is to interpolate discrete values of displacement in $\widehat{\mathbf{U}}$ to a continuous representation within each element. The $H^{(m)}$ matrix is used as a building block for the mass and stiffness matrices:

$$\mathbf{K} = \sum_m \mathbf{K}^{(m)} = \sum_m \int_{V^{(m)}} \mathbf{B}^{(m)T}\mathbf{D}^{(m)}\mathbf{B}^{(m)} dV^{(m)} \tag{3.81}$$

$$\mathbf{M} = \sum_m \mathbf{M}^{(m)} = \sum_m \int_{V^{(m)}} \rho^{(m)}\mathbf{H}^{(m)T}\mathbf{H}^{(m)} dV^{(m)} \tag{3.82}$$

$$\mathbf{C} = \sum_m \mathbf{C}^{(m)} = \sum_m \int_{V_{(m)}} \kappa^{(m)}\mathbf{H}^{(m)T}\mathbf{H}^{(m)} dV^{(m)} \tag{3.83}$$

where $\mathbf{D}^{(m)}$ is the standard stress-strain matrix, ρ^m is the density of element m, $\kappa^{(m)}$ is the element's damping parameter, $V^{(m)}$ define integration over the volume (3D) or surface (2D) of element m, and $\mathbf{B}$ is defined as:

$$\begin{bmatrix} \frac{\partial u^{(m)}}{\partial x_1} \\ \frac{\partial v^{(m)}}{\partial x_2} \\ \frac{\partial w^{(m)}}{\partial x_3} \end{bmatrix} = \mathbf{B}\widehat{\mathbf{U}} \tag{3.84}$$

and can be easily obtained from differentiating $\mathbf{H}$ appropriately. The damping parameter, $\kappa^{(m)}$, is frequency dependent and thus difficult to obtain for finite element models. As such, the damping matrix is usually estimated as a combination of mass and stiffness matrices called Rayleigh Damping. This damping matrix simulates material damping (damping due to scattering, heat losses, friction, and others) and is defined as:

$$\mathbf{C} = \alpha\mathbf{M} + \beta\mathbf{K} \tag{3.85}$$

where α is the mass damping parameter, and β is the stiffness damping parameter. The mass damping parameter influences low frequency attenuation such as solid body motion whereas stiffness damping affects higher frequency vibrations. The values of damping simulate physical signal attenuation and can be set to zero if no damping is required. Practically, α, β can be defined as (Zerwer et al., 2002):

$$\alpha = \frac{2D(\omega)\omega_1\omega_n}{\omega_1 + \omega_n} \tag{3.86}$$

$$\beta = \frac{2D(\omega)}{\omega_1 + \omega_n} \tag{3.87}$$

where $D(\omega)$ is the frequency dependent damping ratio, ω_1 is the first natural frequency of interest, and ω_n is the highest frequency of importance. Knowing α, β the frequency dependent damping ratio is given by:

$$D(\omega) \quad = \quad \frac{\alpha}{2\omega} + \frac{\beta\omega}{2} \tag{3.88}$$

a typical variation of damping with frequency is presented in figure 3.14.

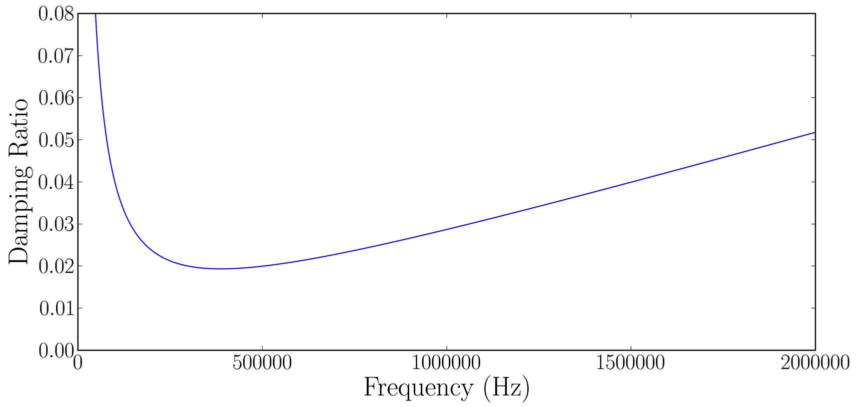


Figure 3.14: Sample curve of frequency dependent damping ratio using $\omega_1 = 628318$, $\omega_n = 9424778$, $\alpha = 47124$ and $\beta = 7.95775 * 10^{-9}$

Dynamics

The methods available to solve equation 3.76 when the external load, R, is a function of time can be separated into two group: explicit and implicit methods. In explicit methods, the solution depends only on a previous state of the system; whereas in an implicit method solution at time t depends on past and future solutions. The advantage of an explicit method is that it is much faster and requires less matrix operations but it is significantly influenced by the choice of time steps. Only explicit methods are covered in this document as they are used in all simulations presented. For simple linear elastic systems, the most common explicit integration solution used for stepping an equilibrium in time is that of central differences. From equation 3.76 above, the time-evolving solutions of $\ddot{U}$, $\dot{U}$ and U are found by Taylor expansion of U:

$$\frac{U(t+\Delta t) - U(t)}{\Delta t} = \dot{U}(t) + O(\Delta t) \qquad (3.89)$$

$$\frac{U(t-\Delta t) - U(t)}{-\Delta t} = \dot{U}(t) + O(\Delta t) \qquad (3.90)$$

$O(\Delta t)$ is an error term which represents the lowest order of unaccounted error given by this expression, in this case Δt. Adding equation 3.89 and 3.90 a relationship for $\dot{U}$ is obtained:

$$\dot{U}(t) = \frac{U(t+\Delta t) + U(t-\Delta t)}{2\Delta t} + O(\Delta t)^2 \qquad (3.91)$$

Using a similar method, but expanding series up to second derivative term, the acceleration is given by:

$$\ddot{U}(t) = \frac{U(t-\Delta t) - 2U(t) + U(t+\Delta t)}{(\Delta t)^2} + O(\Delta t)^2 \qquad (3.92)$$

Thus, with equations 3.91 and 3.92 equation 3.76 can be written as a function of displacement only.

$$\left[\frac{\mathbf{M}}{(\Delta t)^2} + \frac{\mathbf{C}}{2(\Delta t)}\right] U(t+\Delta t) = \left[\frac{-2\mathbf{M}}{(\Delta t)^2} + \mathbf{K}\right] U(t) + \left[\frac{\mathbf{M}}{(\Delta t)^2} + \frac{\mathbf{C}}{2(\Delta t)}\right] U(t-\Delta t) \qquad (3.93)$$

Given M, and C are known matrices and Δt is a known constant, it is possible to find solutions at $t + \Delta t$ provided that the initial conditions for displacement at t and $t - \Delta t$ are known. For a typical problem, initial conditions are only given at $t = 0$ and thus a special solution is needed to find a second set of solution at $t - \Delta t$. Bathe (1982) suggests that this can be easily found using a starting procedure that obtains the backwards time using:

$$U(-\Delta t) \quad = \quad U(0) - \Delta t\dot{U}(0) + \frac{(\Delta t)^2}{2}\ddot{U}(0) \tag{3.94}$$

which is the undamped equation of motion for a solid body applied at every node.

Conditions for Accuracy

The explicit method is valid only within a certain parameter range. A good solution must have convergence, accuracy and speed. An increase in accuracy is generally obtained by using a smaller time step; whereas convergence is guaranteed only if the time step is small enough compared to the period of the loading function. The solution, using the central difference method, is convergent only if the time step, Δt, is smaller than the smallest period of the applied loads. The choice of time step can greatly influence accuracy and computing time (Bathe, 1982). The actual value of Δt is selected by the software or the user, based on the wave propagation velocity of the materials.

In wave propagation problems, another important factor is the discretization of the mesh, or the distance between each node (Marfurt, 1984). Four-node elements act as low pass filters and won't be able to resolve high frequencies (i.e. wavelengths smaller than the distance between the nodes). It has been found by Zerwer et al. (2002) the frequency-wavenumber plot can be used to select the appropriate mesh size to avoid numerical dispersion; furthermore, the use of at least 10 elements for the smallest wavelength of interest is commonly accepted to obtain good time-domain results (Zerwer et al., 2002).

3.4.2 Model calibration and verification

Validation of finite element models, in any field, requires the comparison of results for theoretical data. As the models get more complex, theoretical solutions are less likely to exist leading to the use of simplified models for validation. Simplified models often offer little certainty because they are too different from the actual problem. Fortunately, the problem of Rayleigh wave propagation in an infinite half-space has long ago been solved analytically by Lamb (1904). However, the Lamb problem does not consider more complex geometries such as layered media, which are of prime interest in nondestructive tests.

Lamb's Problem

Lamb (1904) solved analytically the behavior of waves propagating through a semi-infinite half-space due to a point source for both continuous harmonic loads and time-dependent impulses. He found solutions for both horizontal and vertical displacements, separately, for each type of impulse. Lamb (1904) made no distinctions between different sources; and the full analytical solution was only provided for one case.

While Lamb's theoretical solution is available for both 2D and 3D models, only the more general three-dimensional case, which is easily replicated by an axisymmetric numerical simulation, is considered here. Lamb (1904) simplified the 3D Rayleigh wave propagation problem at the surface of an infinite half space, far from the source, to solve the radial propagation problem (eq. 3.95), and the vertical propagation problem (eq. 3.96).

$$u = \frac{H}{\pi\mu_l}\frac{\partial}{\partial r}\int_0^\infty Q\left(t-\frac{r}{V_r}\cosh u\right)du + C_0 \tag{3.95}$$

$$v = \frac{K}{V_r\pi\mu_l}\frac{\partial}{\partial t}\int_0^\infty Q'\left(t-\frac{r}{V_r}\cosh u\right)du + C_1 \tag{3.96}$$

$$Q'(t) = \frac{1}{\pi}\int_0^\infty dp\int_{-\infty}^\infty Q\left(\lambda\right)\sin p\left(t-\lambda\right)d\lambda \tag{3.97}$$

where u, v are horizontal and vertical displacements respectively, H and K are constants that depend on material properties, μ_l is the Lamé elastic constant, r is the radius from the source origin, λ, p are variables used for integration, V_r is the velocity of the surface wave, t is time, Q is the applied load function, Q' (eq. 3.97) is a transform of Q and C_0, C_1 are integration constants.

Vertical displacements are commonly measured in practice. Solving the actual equations for vertical displacement is difficult for arbitrary pulses and different geometries, however it can be achieved using numerical methods. On the other hand, for certain pulses, solutions can be found analytically as shown by Lamb (1904) and in a slightly simplified discussion by Bath and Berkhout (1984).

Using the load curve defined below:

$$Q\left(t\right) = \frac{F}{\pi}\frac{\tau}{t^2+\tau^2} \tag{3.98}$$

Lamb (1904) suggested the use of the following equation as a primed transform of equation 3.98 and it can be shown to satisfy relation 3.97.

$$Q'\left(t\right) = \frac{F}{\pi}\frac{t}{t^2+\tau^2} \tag{3.99}$$

where τ is a parameter affecting frequency content of the pulse, and F is a constant scaling the amplitude of the source. A small value of τ generates high frequencies.

Using these source values, solutions for vertical and horizontal displacement can be found using 3.96 and 3.95 respectively. The resulting displacement solutions are represented below.

$$\begin{aligned} u &= \frac{H\bar{Q}}{4\pi\mu\tau^2V_r}\sqrt{\frac{2\tau V_r}{r}}sin\left(\frac{\pi}{4}-\frac{3}{2}\nu\right)\cos^{3/2}\nu && (3.100)\\ v &= \frac{K\bar{Q}}{4\pi\mu\tau^2V_r}\sqrt{\frac{2\tau V_r}{r}}cos\left(\frac{\pi}{4}-\frac{3}{2}\nu\right)\cos^{3/2}\nu && (3.101)\\ \tan\nu &= \left(\frac{t-\frac{r}{V_r}}{\tau}\right) \end{aligned}$$

where r is the distance from the source. Displacements on the surface due to longitudinal and transverse waves are ignored given their relatively small amplitudes compared to the Rayleigh wave signal. The Lamb's solutions, however, allow for finding the P and S waves solutions if required. A sample solution for horizontal and vertical displacement is presented in figure 3.15.

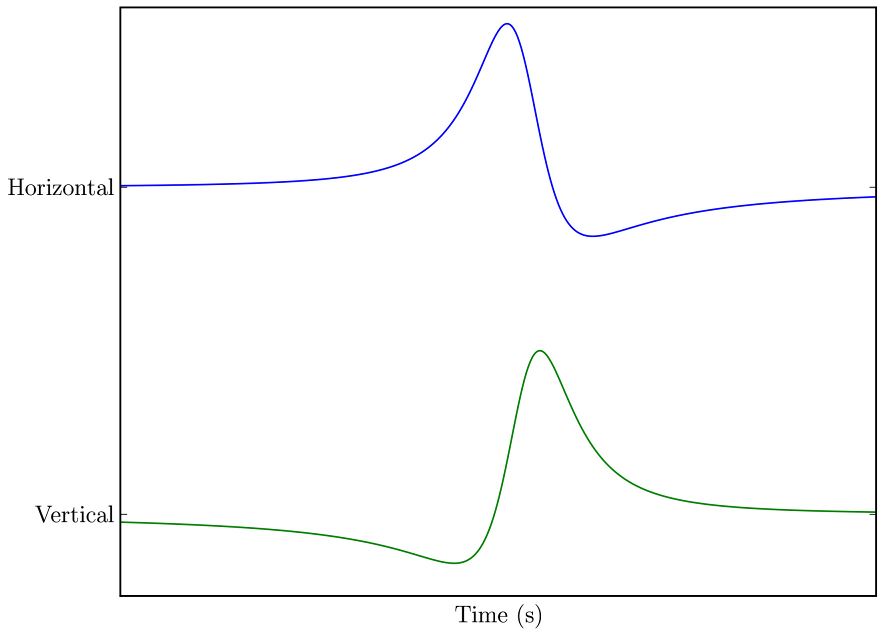


Figure 3.15: Lamb solution for the vertical and horizontal displacement due to a point source (eq. 3.98)

Layered Media

The computation of the response of a layered media can be obtained following finite element principles. However, the specific description of the problem can lead to computationally efficient solutions such as those presented by Kausel (1981). These solutions are quite involved but their principle can be succinctly explained.

Assuming a one-dimensional soil profile excited by a line source, the definition of its harmonic solutions can be simplified as (Kausel, 1981):

$$\begin{bmatrix} U \\ S \end{bmatrix} = \frac{1}{2\pi}\int_{-\infty}^{\infty} \begin{bmatrix} \bar{U} \\ \bar{S} \end{bmatrix} e^{-ikx} dk \tag{3.102}$$

with:

$$\bar{\mathbf{S}} = \begin{bmatrix} \bar{\tau}_{xz} \\ \bar{\tau}_{yz} \\ i\bar{\sigma}_z \end{bmatrix} \tag{3.103}$$

$$\bar{\mathbf{U}} = \begin{bmatrix} \bar{u}_x \\ \bar{u}_y \\ i\bar{u}_z \end{bmatrix} \tag{3.104}$$

where $\bar{u}, \bar{\tau}, \bar{\sigma}$ is the displacement, shearing stress and normal stress in the frequency-wavenumber domain. $U, S, \bar{U}, \bar{S}$ are the displacement and stress vectors in the time and frequency domain respectively.

The above system can be solved by using the stiffness matrix approach in which the soil is divided in discrete elements and the equilibrium is found for each discrete layer. Assuming each layers is composed of two nodes corresponding to the top and bottom respectively, the stiffness matrix composition of the first layer composed of node 1 and 2 is given as:

$$\bar{\mathbf{P}} = \mathbf{K}_m\bar{\mathbf{U}} \tag{3.105}$$

or, in semi-expanded form:

$$\begin{bmatrix} \bar{\mathbf{P}}_1 \\ \bar{\mathbf{P}}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{K}_{11} & \mathbf{K}_{12} \\ \mathbf{K}_{21} & \mathbf{K}_{22} \end{bmatrix} \begin{bmatrix} \bar{\mathbf{U}}_1 \\ \bar{\mathbf{U}}_2 \end{bmatrix} \tag{3.106}$$

where $\mathbf{K}_m$ is the stiffness matrix, $\bar{\mathbf{P}}_1 = \bar{\mathbf{S}}_1$ is the external load at the top, and $\bar{\mathbf{P}}_2 = -\bar{\mathbf{S}}_2$ is the external load at the bottom; the loads are so defined to preserve equilibrium within the layer. For N number of layers, N+1 elements, the stiffness matrix is a symmetric square matrix of size 3N+3 corresponding to each node's three degree of freedom.

The solution of the displacements can be found by multiplying both sides of 3.106 by the inverse stiffness matrix, $\mathbf{K}^{-1}$, such that:

$$\bar{\mathbf{U}} = \mathbf{K}^{-1}\bar{\mathbf{P}} \tag{3.107}$$

Kausel (1981) suggested the use of a thin layer formulation for the evaluation of the stiffness matrix. In this approach, each layer is divided into a number of smaller ones such that each sub-layer is about one sixth of the smallest wavelength studied. By doing so, the stiffness matrix can be estimated as:

$$\mathbf{K}_m = \mathbf{A}_m k^2 + \mathbf{B}_m k + \mathbf{G}_m - \omega^2\mathbf{M}_m \tag{3.108}$$

where k is the wave-number and matrices are defined as:

$$\mathbf{A}_m = \frac{h}{6}\begin{bmatrix} 2(\lambda+2G) & 0 & 0 & \lambda+2G & 0 & 0 \\ 0 & 2G & 0 & 0 & G & 0 \\ 0 & 0 & 2G & 0 & 0 & G \\ \lambda+2G & 0 & 0 & 2(\lambda+2G) & 0 & 0 \\ 0 & G & 0 & 0 & 2G & 0 \\ 0 & 0 & G & 0 & 0 & 2G \end{bmatrix} \tag{3.109}$$

$$\mathbf{B}_m = \frac{1}{2}\begin{bmatrix} 0 & 0 & \lambda-G & 0 & 0 & -(\lambda+G) \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \lambda-G & 0 & 0 & \lambda+G & 0 & 0 \\ 0 & 0 & \lambda+G & 0 & 0 & -(\lambda-G) \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -(\lambda+G) & 0 & 0 & -(\lambda-G) & 0 & 0 \end{bmatrix} \tag{3.110}$$

$$\mathbf{G}_m = \frac{1}{h}\begin{bmatrix} G & 0 & 0 & -G & 0 & 0 \\ 0 & G & 0 & 0 & -G & 0 \\ 0 & 0 & \lambda+2G & 0 & 0 & -(\lambda+2G) \\ -G & 0 & 0 & G & 0 & 0 \\ 0 & -G & 0 & 0 & G & 0 \\ 0 & 0 & -(\lambda+2G) & 0 & 0 & \lambda+2G \end{bmatrix} \tag{3.111}$$

$$\mathbf{M}_m = \frac{\rho h}{6}\begin{bmatrix} 2 & 0 & 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 0 & 0 & 1 \\ 1 & 0 & 0 & 2 & 0 & 0 \\ 0 & 1 & 0 & 0 & 2 & 0 \\ 0 & 0 & 1 & 0 & 0 & 2 \end{bmatrix} \tag{3.112}$$

where λ_l is the Lamé constant, G is the shear modulus, ρ is the mass density, and h is the layer thickness. From this approximation, the stiffness matrix can be built using only known material properties.

Practically, finding $\mathbf{K}^{-1}$ and solving for displacement is computationally expensive. The solutions proposed by Kausel (1981) propose the use of spectral decomposition for obtaining the inverse stiffness matrix and the use of Green's functions as impulse loads to find solutions of single frequency harmonic excitations. Kausel (1981) has a fully implemented solver in the form of an application named "Punch".

Chapter 4

Numerical and Experimental Methodology

This chapter presents the numerical and experimental methodology used for this research. Aspects pertaining to numerical simulations are presented followed by description of the laboratory experiment program.

First numerical models are validated using Lamb's solution and further verified using Punch as a trusted output in simulating wave propagation. Then, methodology for the simulation of experimental specimen is presented.

Second, experimental setup for the study on 11 mortar and 12 concrete specimen is presented.

Finally, the model used in the piston-ui application is presented. Piston-ui is used to model the pressure field of any circular transducers based on an arbitrary impulse.

4.1 Numerical Simulations

The theoretical or empirical relationships used for choosing critical parameters as well as their relevance to the final results are presented in different sections. All numerical simulations, unless otherwise noted, are performed with LS-DYNA version 970 revision 6763. LS-DYNA is an advanced explicit/implicit dynamic finite element solver (Hallquist, 2006). The numerical simulations are split in two important parts. First, the calibration section is used as confirmation of the accuracy of our finite element parameters and models. For calibration the well known Lamb problem is compared with its theoretical solution, and a layered soil profile is compared to a different, well proven code: "Punch". Second, the numerical simulations representing the laboratory tests are explained.

4.1.1 Parameter Definition Considerations

Basic finite element parameters common to all models such as element size, time step and damping govern the accuracy of the models. The conditions for accuracy are heavily dependent on the application, geometry and physics of individual models. Empirical relations shown to give good results for wave propagation in elastic solids are discussed next.

Element Size

The mesh size or size of individual elements is chosen to satisfy stability conditions for the frequency range of interest. The choice of mesh size is critical as it affects computational demand, storage space, and memory usage. Any finite element will act as a low-pass filter with a cut-off frequency inversely proportional to element size.

The choice of an element size is based on the slowest traveling wave at its highest frequency (smallest wavelength). In the case of waves in elastic solids, waves such as longitudinal, transversal and, in surface problems, the Rayleigh surface waves are of interest. Thus, a relationship for element size can be given as (Zerwer et al., 2002)

$$\Delta s \leq \xi \lambda_{min} = \xi \frac{V_{min}}{f_{max}} \tag{4.1}$$

where Δs is the element size, ξ is a constant, λ_{min} is the minimum wavelength of interest, V_{min} is the slowest wave velocity and f_{max} is the maximum frequency.

For square elements, the ξ must be, $\xi \leq 0.5$ in order to satisfy the Nyquist sampling rate (e.g. $\Delta s < \lambda_{min}/2$). Further numerical constraints require that $\xi < 0.2$ to avoid numerical dispersion in practical applications (Valliappan and Murti, 1984). Using a relatively large ξ value could give acceptable results in frequency space but poor time-domain representation of high frequencies. In this project, $\xi < 0.05$ in order to have at a minimum 20 nodes per largest studied wavelength (section 3.4.1).

The choice of frequency range is problem dependent and has to be determined either by considering the applied wave excitation or, if the excitation is not known, by doing a separate modal analysis of the problem. A modal analysis is useful in determining which frequencies are most relevant to the motion being investigated.

The accuracy of the final results can be confirmed by observing the behavior of the frequency-wavenumber plot. The results are deemed acceptable if there is no numerical dispersion (Zerwer et al., 2002).

Time Stepping

Time stepping refers to the selection of the time step to achieve a stable solution. Most modern software have algorithms to automatically determine and adjust the time steps for stability during a simulation. LS-DYNA computes the critical time step for elastic shell elements as (Hallquist, 2006):

$$\Delta t_c = \frac{L_c}{V_p} \tag{4.2}$$

where L_c is the characteristic length defined by

$$L_{c1} = \frac{(1+\beta)A_s}{\max(L_1, L_2, L_3, (1-\beta)L_4)} \tag{4.3}$$

$$L_{c2} = \frac{(1+\beta)A_s}{\max(D_1, D_2)} \tag{4.4}$$

where A_c is the area of the element, β is a parameter equal to 1 if the element is triangular and 0 otherwise and L_i are the length of each sides 1,2,3 and 4. D_1, D_2 are the length of diagonals and finally V_p is the longitudinal wave velocity (eq. 4.5).

$$V_p = \sqrt{\frac{E(1-\nu)}{\rho(1+\nu)(1-2\nu)}} \tag{4.5}$$

where E is the Young's modulus, ν is the Poisson's ratio and ρ is the density. Generally, eqn. 4.4 gives a smaller, more conservative time step and is more suitable for waves travelling diagonally as well as longitudinally through elements. Unlike element size which can be set for each individual components of a model, the time step has to be global, thus the immediate time step is:

$$\Delta t^{n+1} = a * \min\{\Delta t_1, \Delta t_2, \ldots, \Delta t_N\} \tag{4.6}$$

where N is total number of elements, n is the current time step and Δt_i is the time step calculated for each individual element. The value a is a scale factor, which is set to a maximum of 0.9 by default to assure stability. The time steps vary widely during the simulation depending on the state of deformation of the problem. An overly small time step is a common cause of instability as it implies that an element has collapsed onto itself, or crossed over an impenetrable boundary such as an adjacent cell (see Figure 4.1). The former causes the simulation to run slowly and provide wrong results while the later causes the program to stop due to the presence of a negative volume. These issues can be solved by reducing the momentum given by the load, reducing the time step, or increasing element size. Given the side effects of reducing the time step and increasing element size it is more convenient to simply reduce the load as the wave propagation is elastic.

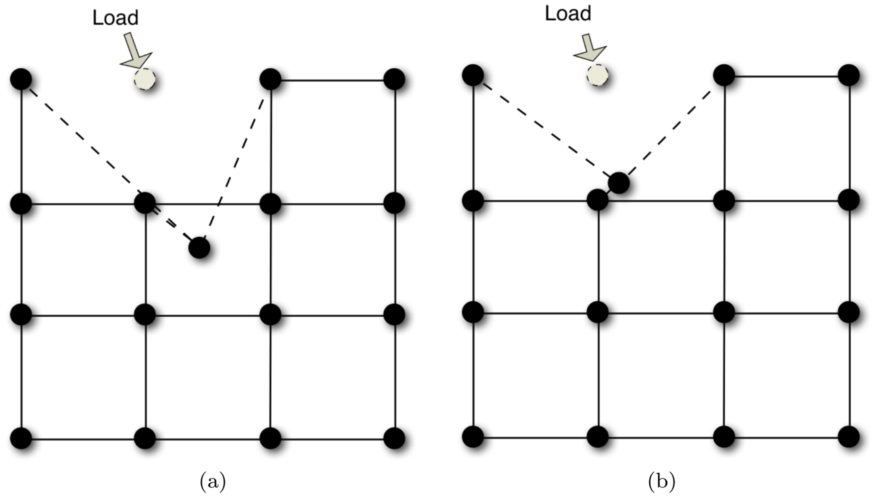


Figure 4.1: Finite elements representation of time step instabilities in LS-DYNA. (a) Negative volume error, (b) Time step instability

4.1.2 Lamb Calibration Model

The model used for the Lamb calibration is described in Figure 4.2. All elements in the model are squares with a total of 360,000 elements. The termination time was chosen as 0.1 ms in order to eliminate the effect of reflections. Quiet boundaries are used to prevent some of the reflection from p-waves but are not accurate enough to fully absorb surface waves. The dimensions are large enough to prevent surface data from including reflections from either the right side, or the bottom of the model.

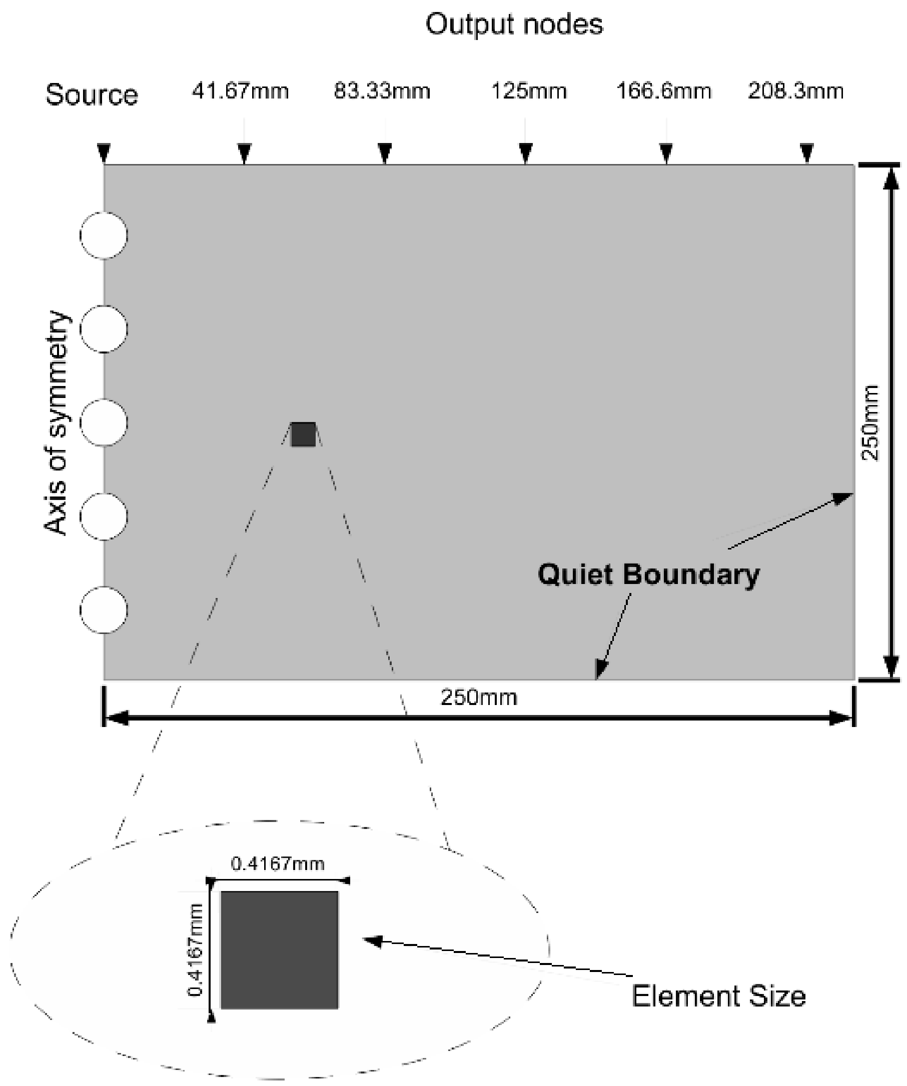


Figure 4.2: Axisymmetric numerical model used for Lamb Solution Calibration (360,000 Elements)

Material Properties

As Lamb's problem assume a linear elastic material the numerical simulation also follows a similar assumption. The material is given a Poisson's Ratio $\nu = 0.33$, Density of $\rho = 2102$ kg/m^3 and Young's Modulus of $E = 29.578\ GPa$. Using the following relations for velocity:

$$\begin{aligned} V_p &= \sqrt{\frac{E(1-\nu)}{\rho(1+\nu)(1-2\nu)}} && (4.7) \\ V_s &= \sqrt{\frac{E}{2\rho(1+\nu)}} && (4.8) \\ \frac{V_r}{V_s} &\approx 0.874 + 0.196\nu - 0.043\nu^2 - 0.055\nu^3 && (4.9) \end{aligned}$$

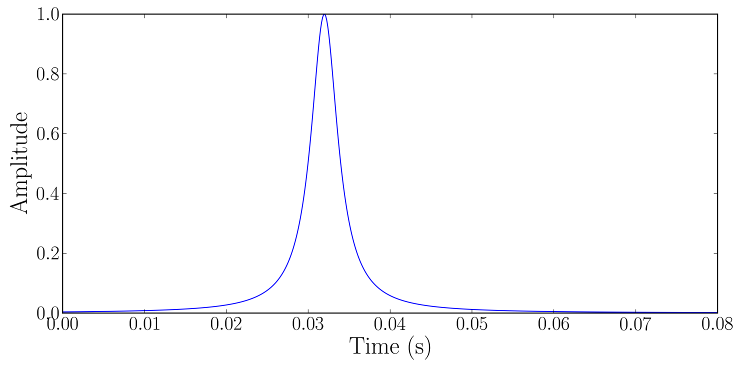
where V_p, V_s are the P and S wave velocities respectively and V_r is the approximate Rayleigh wave velocity. this leads to theoretical velocities of $V_p = 4566.0\ \frac{m}{s}$, $V_s = 2300.0\ \frac{m}{s}$ and $V_r = 2143.6\ \frac{m}{s}$. The Rayleigh wave velocity approximation is over 0.01% accurate for this material's poisson ratio (Vinh and Ogden, 2004; Vinh and Malischewsky, 2006).

Load Curve

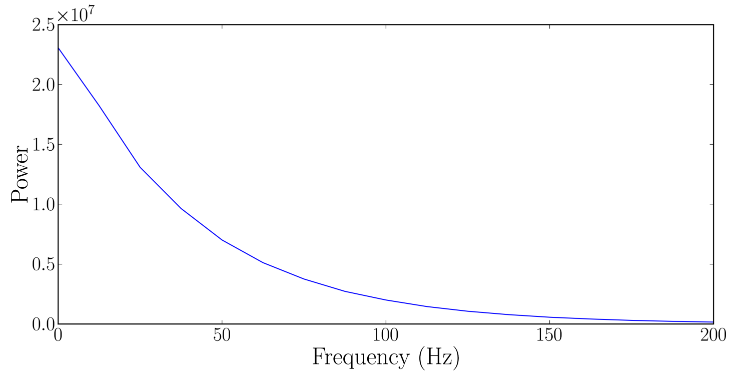
The load curve used follows equation 3.98, shifted in time. The time shift is used to prevent the numerical simulation from experiencing a sudden jerk at the start time. The time shifted load curve is represented by:

$$Q(t) = F\frac{\tau_l}{(t-t_0)^2 + \tau_l^2} \qquad (4.10)$$

where $t_0 = 0.032$ is the time shift, F is the amplitude modulation term, and $\tau_l = 0.002$ is the frequency modulation constant. Amplitude is chosen to avoid instabilities due to high amplitudes while also being large enough to avoid discretization errors.



(a)



(b)

Figure 4.3: Lamb problem's source for simulation. (a) Time trace, (b) Power spectrum

LS-Dyna Input Deck

The LS-Dyna Input file used for the simulation is presented below. Input decks for other simulations can be found in Appendix A.1. The whole node and element information is not included due to excessive length but can be reconstructed from information in figure 4.2.

```
*KEYWORD
*TITLE
Autmatically generated by cylgenDyna.py
*CONTROL_TERMINATION
$   ENDTIM    ENDCYC     DTMIN    ENDENG    ENDMAS
$#  endtim    endcyc     dtmin    endeng    endmas
  0.100000         0     0.000     0.000     0.000
*CONTROL_TIMESTEP
$   DTINIT    TSSFAC      ISDO    TSLIMT     DT2MS      LCTM     ERODE   MS1ST
$#  dtinit    tssfac      isdo    tslimt     dt2ms      lctm     erode   ms1st
     0.000  0.900000         0     0.000     0.000         0         0       0
$#  dt2msf   dt2mslc
     0.000         0
*DATABASE_GLSTAT
$#      dt    binary
  0.160000         1
*DATABASE_NODOUT
$       DT    BINARY
$#      dt    binary
  0.080000         1
*DATABASE_BINARY_D3PLOT
$#      dt      lcdt      beam     npltc
  0.160000         0         0         0
$#   ioopt
         0
*LOAD_NODE_POINT
$     NODE       DOF      LCID        SF       CID        M1        M2      M3
$#     nid       dof      lcid        sf       cid        m1        m2      m3
         1         2         1  1.000000         0         0         0       0 *PART
$# title
Part 1
$#     pid     secid       mid     eosid      hgid      grav    adpopt    tmid
         1         1         1         0         0         0         0       0
*SECTION_SHELL_TITLE
P-1
$    SECID    ELFORM      SHRF       NIP     PROPT   QR/IRID     ICOMP   SETYP
$#   secid    elform      shrf       nip     propt   qr/irid     icomp   setyp
         1        15  0.830000         4         1         0         0       1
$#      t1        t2        t3        t4      nloc     marea
     0.000     0.000     0.000     0.000         1     0.000
*MAT_ELASTIC_TITLE
Material Layer
$      MID        RO         E        PR        DA        DB
$#     mid        ro         e        pr        da        db  not used
         1  0.002100 29578.000  0.330000     0.000     0.000         0
```

4.1.3 Layered Calibration Models

Calibration using layered media is done by comparing results of similar models between LS-Dyna and Punch(Kausel, 1981). Punch is the numerical implementation of the thin layer model presented in section 3.4.2. The comparison is achieved by an automatic generation utility, LayergenDyna, that takes a Punch configuration file as an input and automatically generates a matching LS-Dyna input deck. The source code for the utility is presented in appendix B.1. Two models are used for comparison. The first model is a single layer model, which simulates an infinite half space similar to the Lamb's problem. The second model is a simple three layer soil profile.

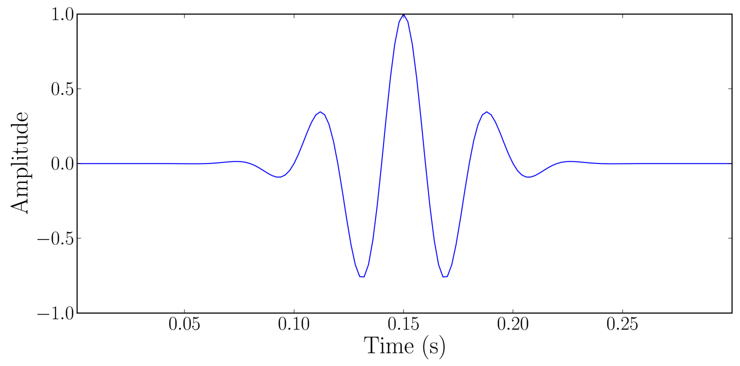
Single layer model

The single layer model is defined by a single 500m deep layer split into 300 sub-layers. It has a density $\rho = 1800kg/m^3$, a poisson ratio $\nu = 0.33$ and a shear wave velocity $V_s = 500m/s$. The element size for the LS-Dyna model is chosen to be 0.4167001 m. This element size has at least 10 elements per smallest wavelength, for shear wave velocity of the medium. The load curve for the model is presented in figure 4.4. It has a centre frequency of 25 Hz with a top frequency of 40 Hz. The LS-Dyna input file can be found in appendix A.2.

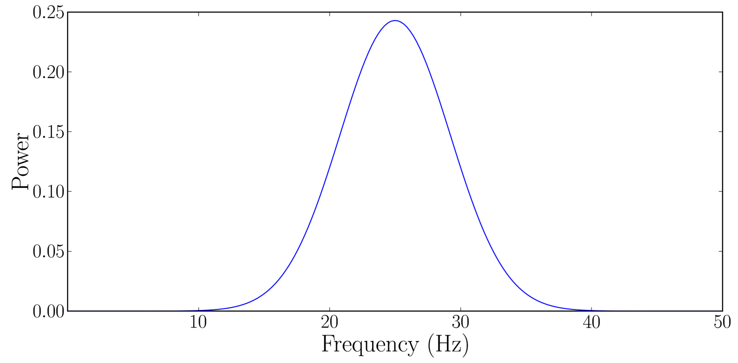
Punch considers an impulse load and provides the solution in the frequency domain. A time trace for an arbitrary excitation can be obtained by the following operation:

$$h(t) \quad = \quad FT^{-1}\left(\bar{s}(f)\bar{g}(f)\right) \tag{4.11}$$

where $h(t)$ is the resulting time domain solution, $\bar{s}(f)$ is the frequency domain solution from Punch, FT^{-1} denotes the inverse Fourier transform operation and $\bar{g}(f)$ is the Fourier transform of the applied excitation.



(a)



(b)

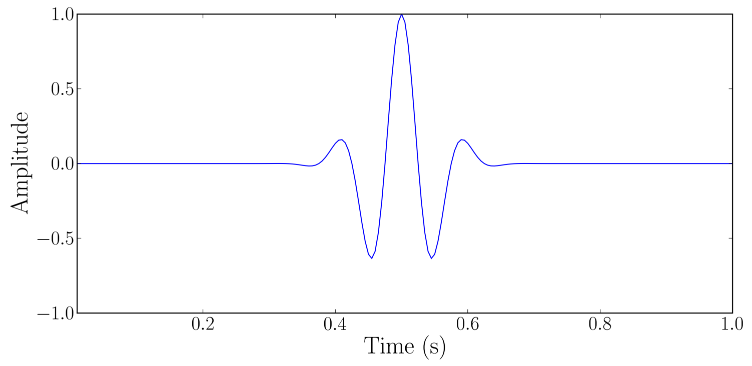
Figure 4.4: Input load for single layer calibration model. (a) Time trace, (b) Power spectrum

Three layer model

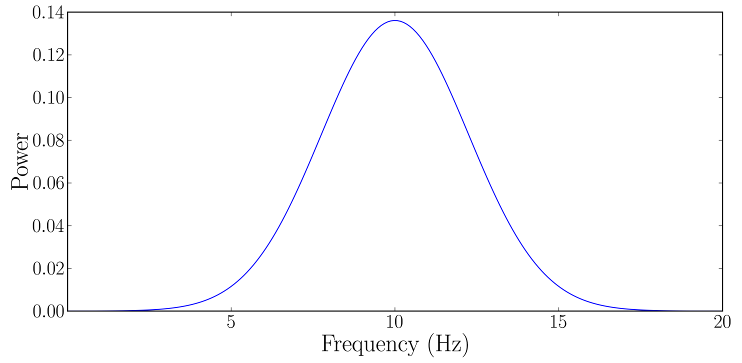
The three layers of this model have properties listed in table 4.1.3. The LS-Dyna model uses an element size of 1.0 m based on the same conditions as previous models. The load curve for this model is presented in figure 4.5. It has a centre frequency of 10 Hz and a top frequency of 17.5 Hz. The larger element size is used to compensate for the larger model (lower number of elements) and the frequency of the load is lowered to avoid numerical dispersion due to the larger element size. The LS-Dyna input deck can be found in appendix A.3.

Table 4.1: Layer properties for three layer model

Sub-layers	**Height (m)**	ρ	V_s	ν
75	100	1800	200	0.33
38	100	1800	400	0.33
155	500	1800	500	0.33



(a)



(b)

Figure 4.5: Input load for three layer calibration model. (a) Time trace, (b) Power spectrum

4.1.4 Model Simulations of mortar specimens

This set of simulations is meant as a direct representation of the experimental setup. The models consist of cylinders excited by circular transducers. These are represented as rectangular axisymmetric simulations. A total of 24 different runs are performed for 12 different sizes, as shown in 4.1.4, each being run for two transducer settings. The models are automatically created using three computer programs written in Python: cylgenDyna, modelGenerator1M, and modelGenerator50K. Source code for these utilities is presented in appendix B.1 to B.3.

Table 4.2: List of sizes for experimental model simulations

Height (mm)	Diam. (mm)
50	100
100	100
200	100
300	100
50	200
100	200
200	200
300	200
50	300
100	300
200	300
300	300

Material Properties

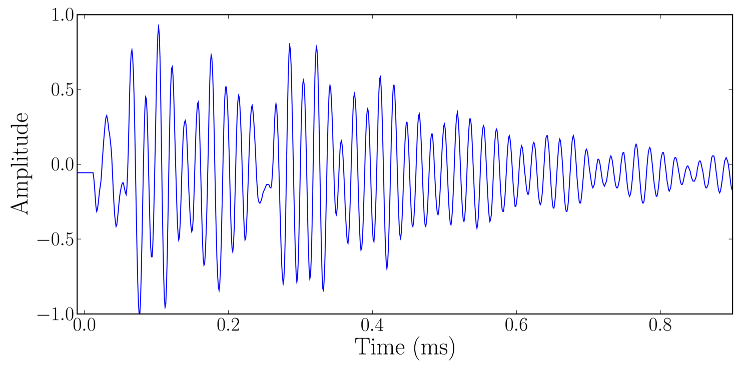
For consistency, all simulations are performed using the same material properties and element size. The material is linear elastic with a density $\rho = 2102\ Kg/m^3$, Young's modulus $E = 29.578\ GPa$ and Poisson ratio $\nu = 0.33$ giving a velocity $V_p = 4566\ m/s$. The element size is set to 0.25 mm using the same rules as previous simulations.

Load Curves

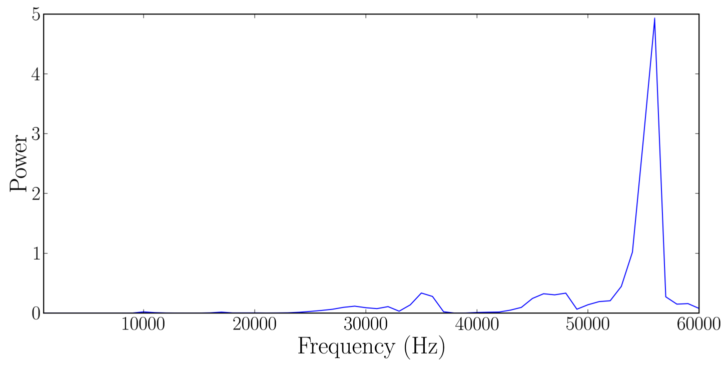
Two load curves are used; which are determined experimentally by measuring the field produced by the transmitter going through a large aluminum block, avoiding any reflections in the received pulse. The low frequency transducer excitation corresponds to a 2 inch diameter circular transducer with centre frequency of 55 KHz (figure 4.6) when excited by an impulse. The second, high frequency, transducer is a 1 inch diameter circular transducer with peak frequency at 850 KHz (figure 4.7) when excited by an impulse.

Impulse for low frequency transducers is provided by a Pundit Pulser/Receiver using no attenuation. The impulse for the high frequency transducer pair was generated by a Panametrics PR5200 Pulser/Receiver using the lowest energy setting (1) and zero attenuation. Both devices are operated in pitch-catch mode.

A sample input file for each transducer configuration is presented in appendix A.4 and A.5.

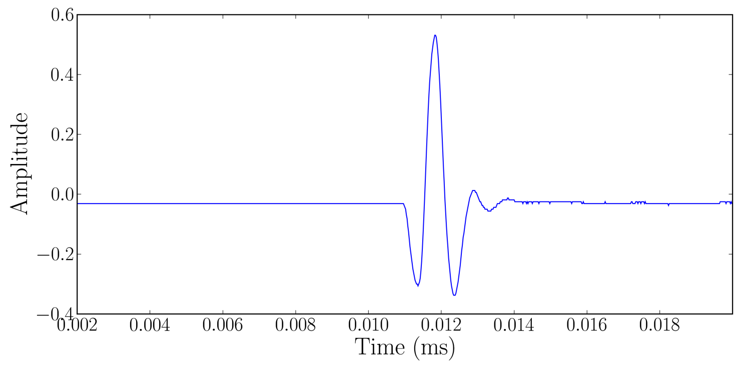


(a)

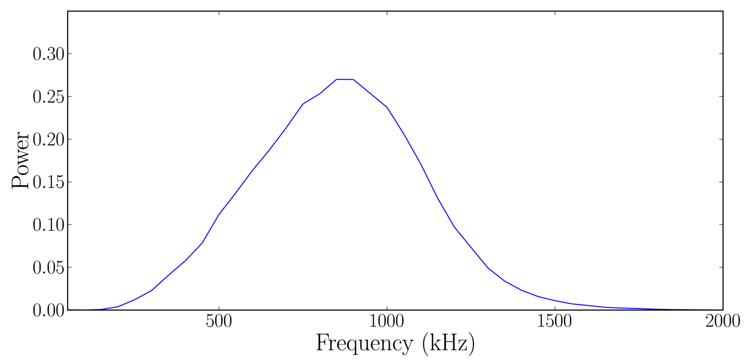


(b)

Figure 4.6: Low Frequency Load for Experimental Model Simulations. (a) Time trace, (b) Power spectrum



(a)



(b)

Figure 4.7: High Frequency Load for Experimental Model Simulations. (a) Time trace, (b) Power spectrum

Damping Curve

Damping is applied to a set of low frequency simulations for comparison with undamped results. For this purpose, damping parameters $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$ are used. The damping curve obtained from these parameters is presented in figure 4.8.

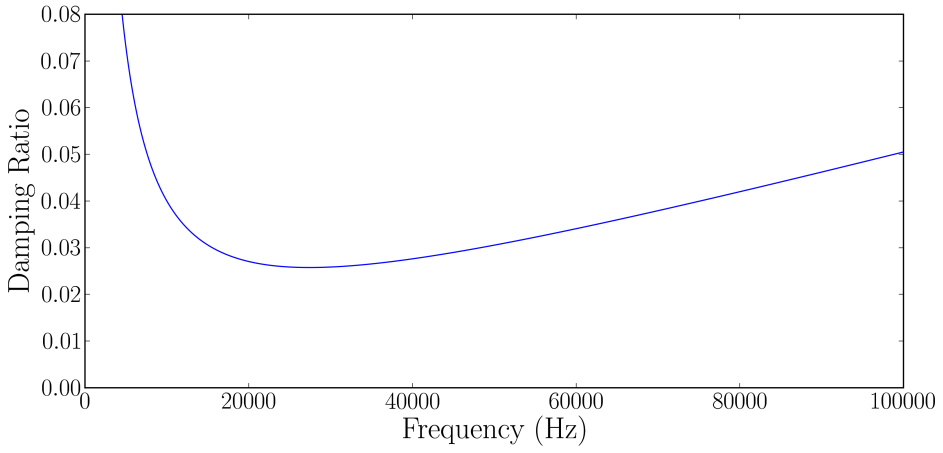


Figure 4.8: Damping curve used for low frequency excitation

4.2 Laboratory testing

4.2.1 Calibration Methodology

Calibration of the ultrasonic equipment is the initial step taken prior to the velocity readings; 16 calibration rods split into 3 different materials are used for this purpose. Table 4.3 describes the calibration rods used. The calibration ensures a good agreement between the low and high frequency transducers under similar circumstances.

Table 4.3: Size of calibration rods used.

Aluminum		Steel		PVC	
Height (mm)	Diam. (mm)	Height (mm)	Diam. (mm)	Height (mm)	Diam. (mm)
49.74	50.75	49.18	49.75	50.99	51.50
74.26	50.75	73.62	49.75		
98.28	50.75	98.90	49.75	101.18	51.50
123.22	50.75	123.90	49.75		
149.01	50.75	157.98	49.75	150.54	51.50
198.92	50.75	198.62	49.75	200.92	51.50

Using standard measurement methods, readings are taken for the high-frequency Panametrics transducers. The low-frequency transducers are not used during calibration as the equipment has an adjustable delay time. As such, it can be calibrated using the data obtained from the high frequency transducers. For calibration, water is used as a couplant as it gives better correlation on the arrival time versus height curves.

Using rods of varying sizes makes it possible to accurately determine the sum of the effects of electronics, human bias and coupling agents. All rods of a similar material originate from a single original rod ensuring they have a similar velocity. Calibration is achieved by plotting the arrival time data of each rod of a material against its height. The slope of which gives an accurate value of velocity with the intercept being the delay time of the system as is shown by equation 4.12. Sample results of the calibrations are shown in figure 4.10 to figure 4.11 with the final velocity data compiled in Table 4.4. All calibration figures can be found in Appendix E.1. Each calibration was done twice with one set using water as coupling and the other using glycerin. These two sets clearly show some of the differences between

Figure 4.9: Calibration Rods

coupling agents. The variation of travel time for the specimens is given by:

$$t = V_p h + d \tag{4.12}$$

where t is travel time, V_p is the medium velocity, h is the height of the specimen and d is the delay time of the system.

The accurate velocity of the calibration rod is used to adjust the low-frequency system. This is done by computing expected arrival time for a calibration rod (200 mm PVC used) and using this time as a target for the system. Uniformity of results can be achieved by matching the readings of the high frequency transducers with those of the low frequency setup.

Table 4.4: Compiled calibration results

Material	Coupling	V_p	d	r^2
Aluminum	Water	6329.4	0.377	0.999970
Aluminum	Glycerin	6384.1	1.74	0.999978
Steel	Water	5737.7	0.115	0.999996
Steel	Glycerin	5770.4	1.79	0.999999
PVC	Water	2309.1	-0.230	0.999999
PVC	Glycerin	2320.2	0.331	0.999992

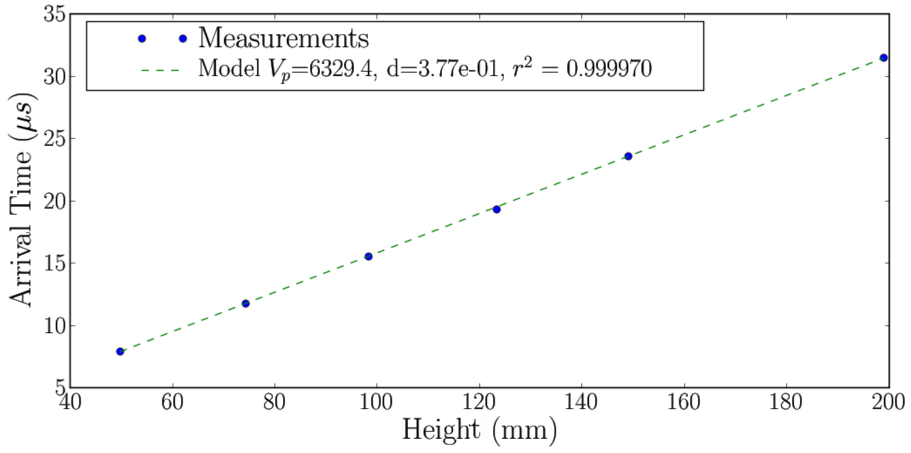


Figure 4.10: Water Calibration Data for Aluminum Rods

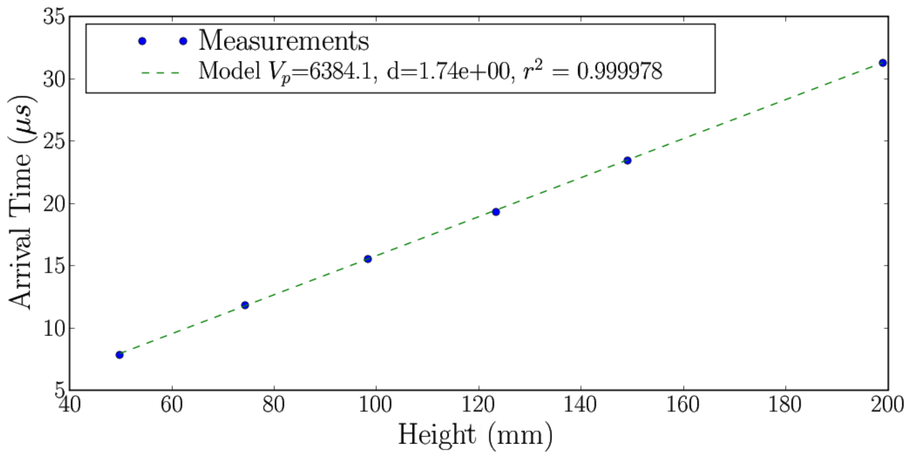


Figure 4.11: Glycerin Calibration Data for Aluminum Rods

4.2.2 First Arrival Measurments

The setup used for velocity measurements uses an oscilloscope connected to a computer for data recording and a pulser-receiver as a source. The oscilloscope has many advantages over most affordable ultrasonic pulse velocity equipment as it captures the entire waveform, provides a significantly better resolution and has the ability to average data and accurately identify points of interest. Velocity is measured using visual inspection on the oscilloscope using identification in a fully zoomed setting for increased accuracy. This reading is then confirmed using the computer data as it is saved. Best results are obtained by scaling the waveform to saturation in the oscilloscope as to have a more accurate view of the small first arrival window. The data is then averaged 256 times in order to increase signal to noise ratio which makes it possible to spot low intensity arrivals.

Visual identification of first arrival is used instead of automated identification in order to get better consistency when the signal is noisy, or very low intensity. These problems are very common in measurements of mortar and concrete specimens. On the test setup

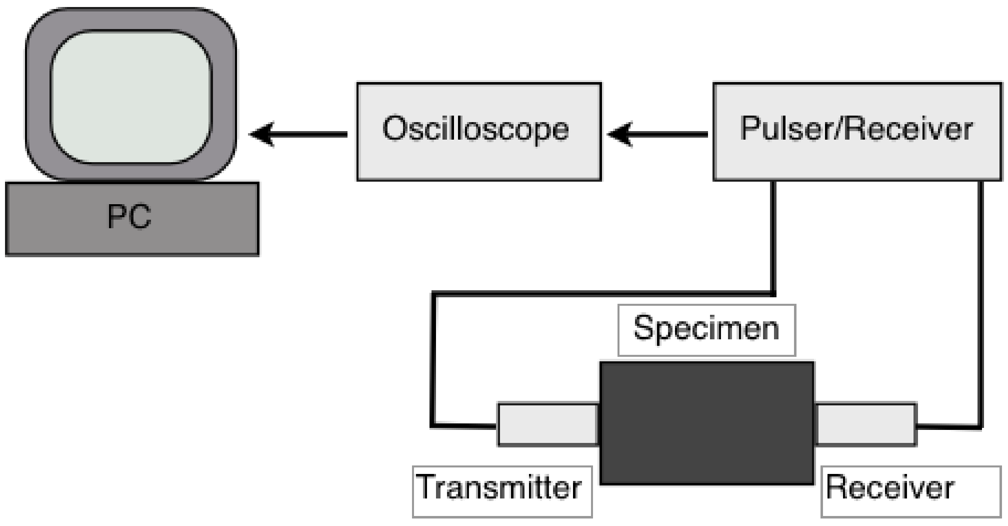


Figure 4.12: Experimental Ultrasonic Pulse Velocity Setup

used, vertical voltage resolution is 2 mV/div with a time resolution varying depending on the waveform, its amplitude and the transducer pair used.

High Frequency Transducer

The high frequency measurements are taken using a set of Panametrics V-102 transducers driven by a Panametrics 5052-PR Pulser Receiver in pitch-catch mode. The system was set to maximum amplification for better first arrival identification.

Low Frequency Transducer

Low frequency measurements are performed using a set of high power transducer with centre frequency at 55 KHz driven by a high power pundit pulser-receiver. This system is used in pitch-catch mode and requires calibration before each use.

Coupling Agent

The coupling agent used varied depending on the surface condition of the specimen. Water is an ideal coupling agent due to its high fluidity and bulk modulus, ease of cleaning and its inert nature on most materials. The high fluidity of water makes the thickness of couplant minimal in turn reducing time delay in the readings. As seen in Figure 4.13, water is also the most effective couplant on a clean, smooth surface.

Water is only of use on smooth solid surfaces such as those of calibration rods and other clean metal samples. For smoothed concrete surfaces, water is also effective provided that the readings are taken quickly as water drying out alters the waveforms over the many averages. For the cases when drying is a problem, glycerin is used. Glycerin has similar properties to water but does not dry out during averages. Glycerin has other drawbacks such as aiding bacterial and fungal growth if uncleaned and being slightly more viscous than water. This viscosity leads to a thicker couplant layer, which lowers couplant effectiveness and may add a delay to the signal.

Finally, for the case of very rough surfaces such as unsanded concrete, it is preferable to use vacuum grease as a coupling agent. Vacuum grease can easily fill holes in a porous surface and act as an excellent coupling medium. The main drawback of Silicon Gel is its relative thickness compared to water and glycerin making it unattractive for high precision readings, like calibrations. Figure 4.13 shows silicon gel as a poor couplant compared to glycerin and water. However, this only applies to clean smooth surfaces where the effectiveness of water and Glycerin is high. Rough surfaces eliminate the advantage of more liquid couplants as it provides only partial contact with the surface.

Another factor that plays an important role in coupling effectiveness is the applied pressure to the transmitter. Large variations in amplitude are thus common for silicon gel due to its relatively variable thickness. These major issues are one of the significant factors affecting amplitude-based nondestructive techniques of concrete (Warnemuende and Wu, 2004).

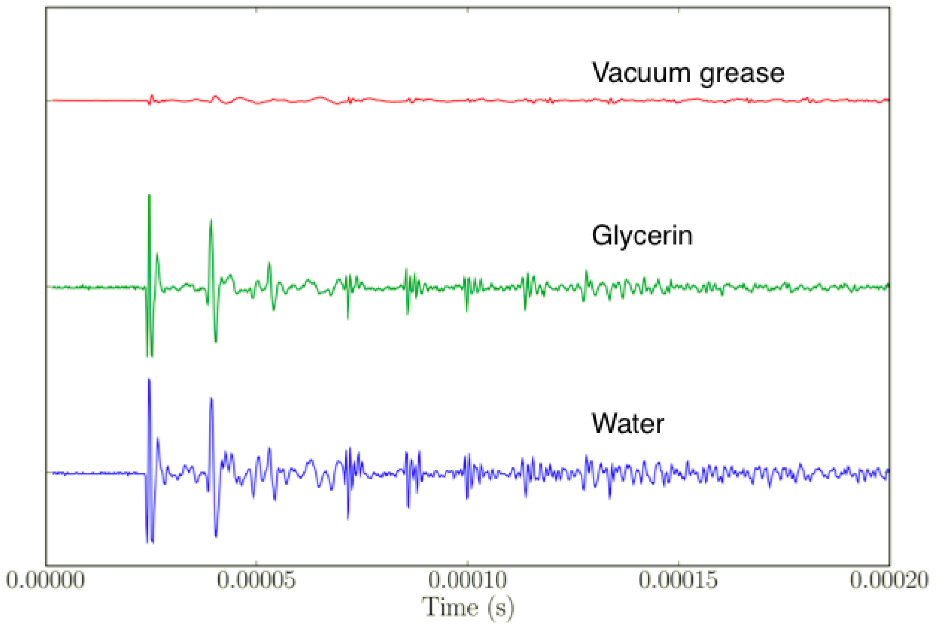


Figure 4.13: Effect of coupling on signal strength using a high frequency measurement on a 5 cm diameter aluminum rod. Amplitudes are normalized to highest peak of water coupling.

4.2.3 Measurements Methodology

Travel time readings on concrete specimen requires more attention than the calibration measurements because of the rough surface. For the mortar specimens, with sanded surface, water could be used as a coupling agent at the bottom surface, which is smooth from the casting process. The top surface required glycerin as it is a coarser surface. In the case of the concrete specimen, silicon gel is used for the top surface and water at the smooth bottom surface.

To ensure repeatability of readings, the location of the transducer is marked clearly on the concrete with the location of each transducer sets being exactly concentric. Each first arrival reading was taken 5 times per specimen, per transducer set. Each reading is taken

Table 4.5: Dimensions of specimens, selected assuming $V_p = 4,800 m/s$

Name	Height (mm)	Diam. (mm)	Height/$\lambda_{0.85MHz}$	Diam./$\lambda_{0.85MHz}$	Height/λ_{55kHz}	Diam./λ_{55KHz}
m1a	50	100	8.87	17.74	0.57	1.15
m2a	100	100	17.74	17.74	1.15	1.15
m3a	200	100	35.46	17.74	2.29	1.15
m4a	300	100	53.19	17.74	3.44	1.15
m1b	50	200	8.87	35.46	0.57	2.29
m2b	100	200	17.74	35.46	1.15	2.29
m3b	200	200	35.46	35.46	2.29	2.29
m4b	300	200	53.19	35.46	3.44	2.29
m1c	50	300	8.87	53.19	0.57	3.44
m2c	100	300	17.74	53.19	1.15	3.44
m3c	200	300	35.46	53.19	2.29	3.44
m4c	300	300	53.19	53.19	3.44	3.44

after full disassembly and reassembly of the equipment to assure repeatability of results. This procedure is used to assess the uncertainty of the results.

4.2.4 Specimens

In order to properly study size effects, multiple test specimens are built. The choice of sizes for each specimen is based on the expected wavelength of the transducers. Specimens are large enough such that the high frequency transducer (Panametrics NDT-V102), would have a centre wavelength with a height to centre wavelength ratio of at least 10. Whereas, the low frequency transducers (55KHz centre frequency) have wavelength comparable to the size of the specimen with a height to centre wavelength ratio from 0.5 to 3. These considerations also apply to specimen width.

Using 0.85 MHz as the centre frequency for the Panametrics NDT-V102 transducer, figure 4.7, and 55 KHz for the low frequency transducers, figure 4.6, the size configurations described by table 4.2.4 are chosen.

The summary of all specimen geometries is shown in Table 4.2.4.

4.2.5 Mortar specimen preparation

Mortar specimen are prepared from a fine sand, its gradation curve is shown in figure 4.14. The sand used in the mortar preparation is first sieved through a #100 (0.150 mm) mesh to remove the small particles which would absorb too much water when mixing. A sieve #16 (1.18 mm) is also used to remove large particles that would create a less homogeneous specimen.

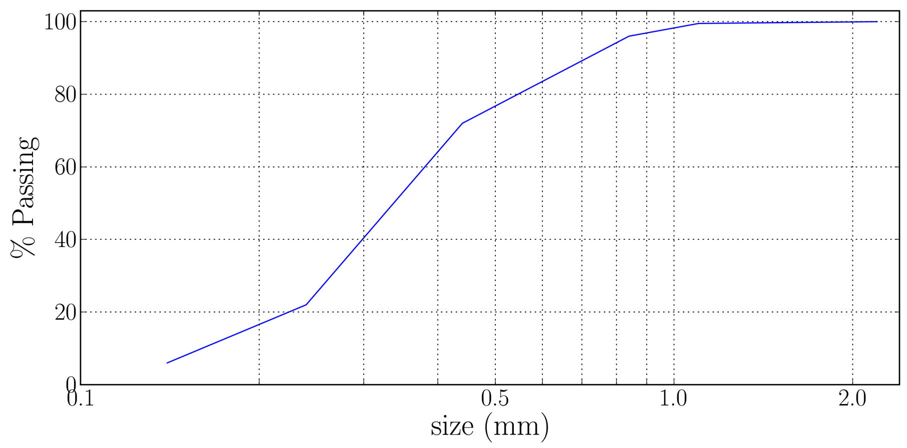


Figure 4.14: Gradation curve of the sand used for mortar mix.

The mortar mix is prepared using a 3:1 sand-cement ratio and a 0.65 water-cement ratio by weight to reduce the volume change of the specimen during curing. The relatively high water-cement ratio is necessary for the workability of the mix.

The mortar is left to solidify for 24 hours in open air, with occasional wettings, after which the moulds are removed and the specimens cured for 28 days in a humidity room. Once curing is complete, the surface of each sample is smoothed using a drill press with coarse sand paper to provide a better contact for the transducers. This smoothing method introduces some curvature to the top surface of the specimen requiring extra couplant for good contact.

All mortar specimens are cast simultaneously from the same batch to minimize differences between specimen. The 70 L pan-mixer available for mixing does not have the required capacity for casting all specimens at once leaving the 30 cm diameter by 30 cm height

Table 4.6: Precise measurements of mortar samples

Name	Height (mm)	Width (mm)
m1a	45.9	100
m1b	46.4	200
m1c	46.4	300
m2a	92.1	100
m2b	94.1	202
m2c	98.7	300
m3a	187.2	100
m3b	187.4	202
m3c	192.6	300
m4a	280.2	100
m4b	280.09	202

sample to be left out of the batch. A list of all the mortar specimens and their accurate measurements can be found in table 4.2.5.

4.2.6 Concrete sample preparation

The concrete specimen are cast from an industrial concrete batch mixed in a concrete mixing vehicle. Unlike the mortar, the concrete has several additives and plasticizers added which significantly reduced their curing time.

The concrete models were not smoothed at the surface since the solidity of their matrix and large aggregate were likely to lead to significant pore creation at the surface. This had negligible effects on coupling.

All concrete specimens were cast at the same time and left to solidify for 5 hours after which the moulds were removed and the samples put in a humidity room for 28 days. The concrete specimens are described in Table 4.7.

Table 4.7: Precise measurements for concrete specimens

Name	Height (mm)	Width (mm)
c1a	49.0	100
c1b	52.7	200
c1c	49.6	300
c2a	98.5	100
c2b	98.0	200
c2c	100.9	300
c3a	194.3	100
c3b	192.6	200
c3c	176.5	300
c4a	280.7	100
c4b	289.0	200
c4c	290.4	300

4.3 Applications

In this section, two major applications are presented. Software used for storing, analyzing, processing and retrieving of data is shown as well as an application for modeling the field of a piston transducer.

4.3.1 Data Management and Processing

As there are large amounts of numerical data in this work, as well as a significant number of experimental time traces, a system is discussed below to store, organize, process and present the data.

Time Trace Storage

To facilitate retrieval, backup and usage of gathered information, a SQL database was used for all information. Due to their different nature, experimental and numerical time traces had to be stored in independent database schemes. A representation of the each database scheme in shown in figure X and Y.

Time Trace Processing

Gathered information consisted of time traces having different properties based on applications. Tools were developed in Python 2.5 using available scientific libraries SciPy 0.6, NumPy 1.0.4 and Matplotlib 0.9. Python offers most of the signal processing facilities offered by Matlab(TM) for free and has the added benefit of having a more flexible object oriented programming language.

The underlying core routines consisted of two classes: a time trace class and a time trace list class. The time trace class(tt) is responsible for handling time traces information and provide routines for consistent plotting, transforms and manipulation of its data.

The time trace list(ttlist) class was created to handle multiple time trace(tt) classes at once. The class is populated using a database connection to the stored data and can be used to save, modify or create new information.

Source code for both these classes and some of their derivates can be found in appendix B. All scripts and processing were done using those classes as a basis.

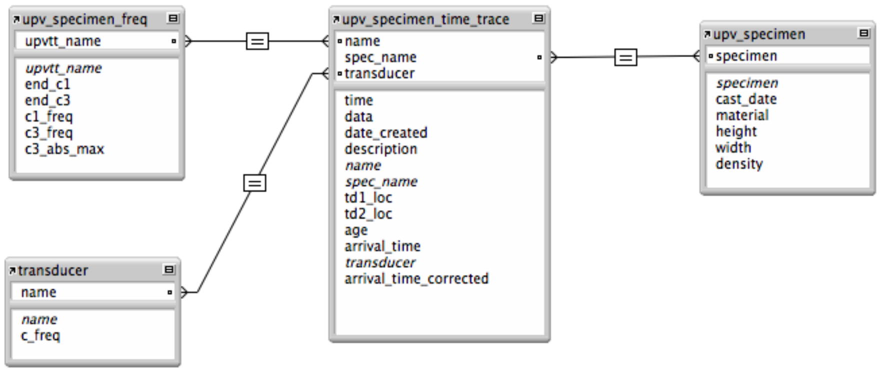


Figure 4.15: Database Scheme for Experimental Time Traces

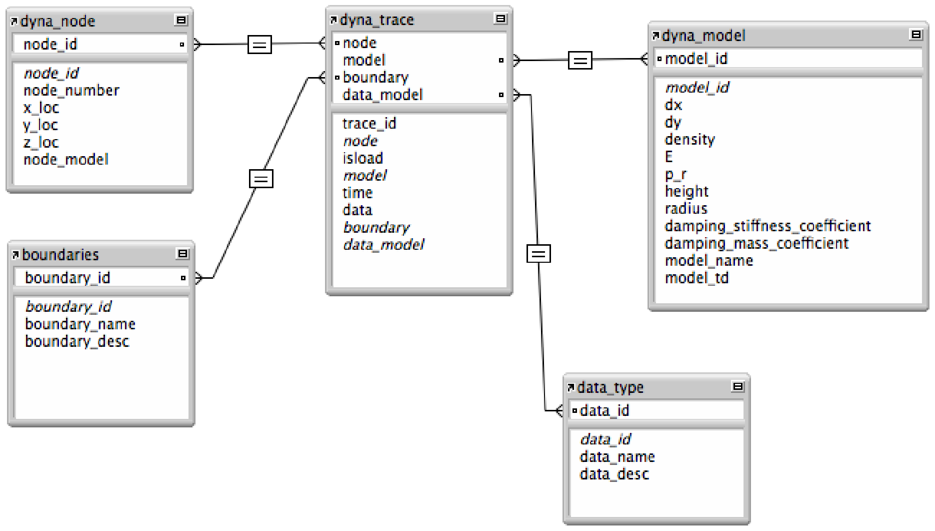


Figure 4.16: Database Scheme for Numerical Time Traces

4.3.2 Transducer Modeling

Modeling of transducer field is an important qualitative piece of information that helps in understanding energy distribution during readings. This modeling was performed by numerical implementation of the theory presented in section 3.3.

The modeling application was split into two independent components: the field solver and the user interface.

Field Solver

The field solver, or solver, is the component implementing the theory of section 3.3. The algorithm was developed in ANSI C and used FFTW3F as Fourier transform pack. FFTW3F was chosen for its performance and reliability over other free implementations.

The field solver consists of three main parts: impulses, transducers, and solver. The solver contains one or more transducers which each contain one impulse.

Impulse Class The impulse is a simple implementation of a time trace and provides procedures for easily creating and extracting needed information. Complete source code is shown in C.8.

Transducer Class The transducer class implements the impulse response of a specific transducer shape. Each one of these class contain an impulse which is used in computing the time-dependent impulse response needed for convolution. Complete source code of the class and its derivative is shown in C.11.

Solver Class The solver class contains all information about the medium and includes the specialized algorithms to compute the field. The solver also contains a list of all the transducers in the simulation as well as the final solved data.

The convolution used for in the algorithm is a direct implementation of the convolution theorem shown in equation 3.57. This class can present the data both in terms of time dependent pressure or pressure amplitude. Pressure amplitude is a more convenient way to present the information and is the result of applying a Hilbert Transform, equation 3.61, to the time dependent pressure and presenting the resulting amplitude.

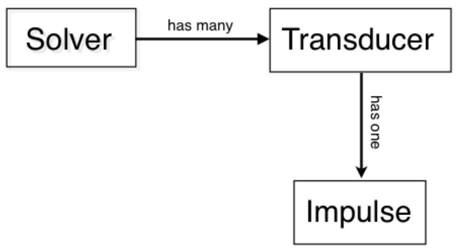
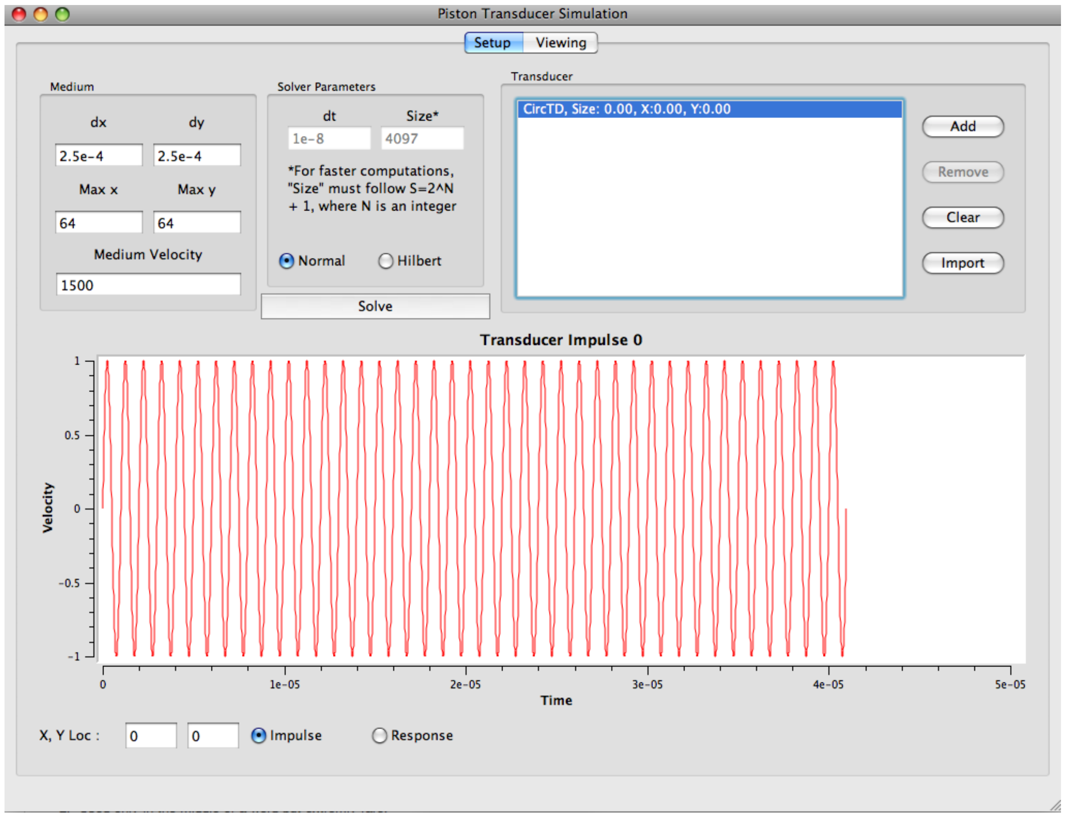


Figure 4.17: Field Solver class hierarchy

User Interface The user interface for the Field Solver was implemented in C++ using Qt 4.3.2 along with Qwt 5.0.3 as a plotting library. The user interface's only purpose was to facilitate creating configurations and impulses as well as presenting the data. The full code for the user interface is presented in D.

Software usage first consists of a setup phase where the transducers, and their impulses are set as well as the properties of the media such as width, height, velocity and element size. Furthermore, the time step and end time are also set in this phase prior to solving. The setup screen is shown in figure 4.18. The settings are unit-less so long as a consistent unit system is used. Any value are acceptable provided that the combination of number of time steps, coupled with the number of elements to solve for does not exceed the computer's memory. The maximum resolution and time steps achieved on a system with 2 gigabyte of memory is 162 by 162 medium resolution with 8193 time steps.



dx 2.5e-4 dy 2.5e-4

Max x 64 Max y 64

Medium Velocity 1500

dt 1e-8 Size* 4097

*For faster computations, "Size" must follow S=2^N + 1, where N is an integer

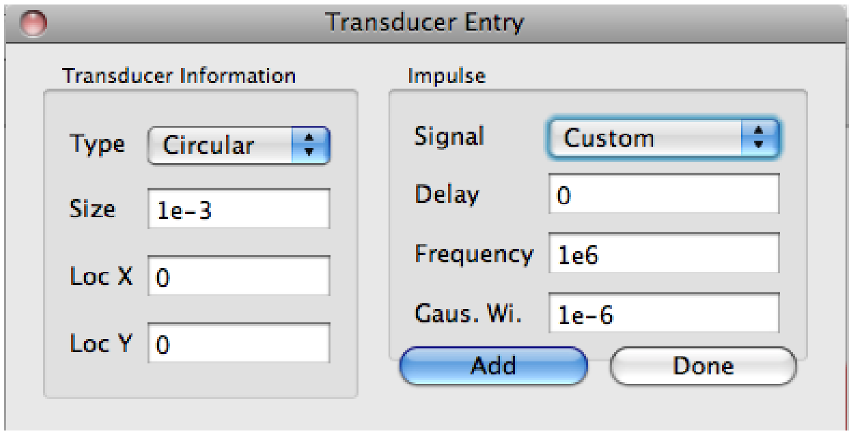
[✓ Normal] Hilbert

CircTD, Size: 0.00, X:0.00, Y:0.00

X, Y Loc : 0 0 [✓ Impulse] Response

Figure 4.18: Field Solver User Interface's setup screen

Any number of transducer is added by clicking the ”Add” button and setting the properties as shown in figure 4.19. Transducer type can be either circular or rectangular, size is the transducer’s radius and can be any value compatible with the chosen unit system while ”Loc X” and ”Loc Y” are the x and y coordinates of the transducer respectively. Impulse properties include the signal type which can be either a sine, gaussian modulated sine or a custom signal imported from Matlab(tm). Other properties include the delay before start of signal, frequency if the impulse chosen is a sine, or gaussian modulated sine and a gaussian width that is applied only if the gaussian modulated sine is chosen.

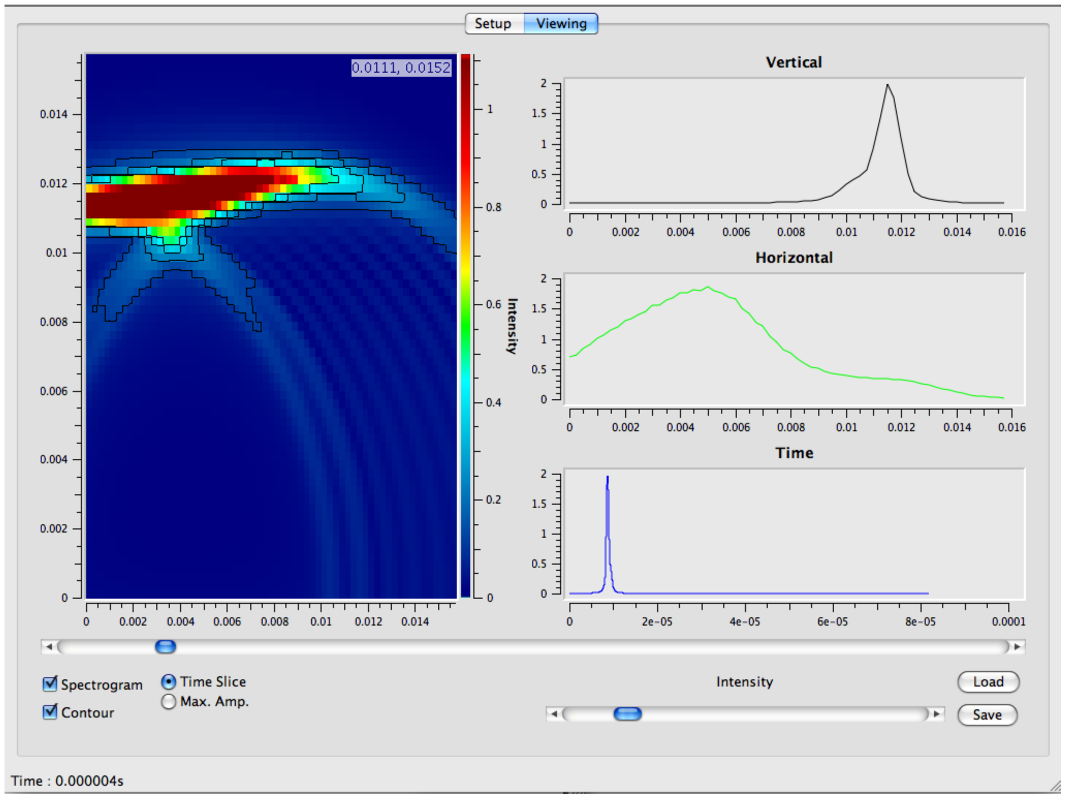


Type: Circular
Size: 1e-3
Loc X: 0
Loc Y: 0
Signal: Custom
Delay: 0
Frequency: 1e6
Gaus. Wi.: 1e-6

Figure 4.19: Field Solver User Interface’s setup screen

Upon completion of the setup, the user choses to either solve for the raw time-dependent pressure or its Hilbert transform. The amplitude of the Hilbert’s transform correspond to the pressure amplitude.

Resulting data can be viewed using the appropriate pane as shown in figure 4.20. Viewing can be controlled in the time direction by the slider positioned below the data window while the color map can be adjusted by moving the intensity slider at the bottom right. By default data is presented as a color-coded scalar plot but can optionally be shown as a contours using the appropriate checkbox at the bottom left of the screen.



Time : 0.000004s

Figure 4.20: Field Solver User Interface's setup screen

Chapter 5

Results and Discussion

This chapter presents results from the described numerical simulations and experiments. First, numerical calibrations are investigated for the Lamb problem and for the comparison with the Punch software. Second, numerical simulations of experimental models are presented for both high and low frequencies. Third, first arrival experimental results are presented and finally the results are discussed for each of the presented section.

5.1 Numerical Simulations

5.1.1 Lamb's Calibration Problem

This section compares the results from LS-Dyna lamb model (section 4.1.2) with the far field theoretical solution. Figure 5.1 through 5.5 show comparisons of measurements at distances of 41 mm to 208 mm (1.34 to 6.8 λ_R for $\lambda_R = 30.6mm$ is the Rayleigh wave wavelength at 75 kHz) away from the vertical point source. All points are taken at the top surface, where the source is located.

All results show good agreement on the sharp pulse rise and drop-off. However, the initial inverted pulse is well matched only for larger distances from the source. Arrival times coincide for every tested signal based on the computed surface wave velocity. The mismatch is also seen prior to the surface wave's arrival in the form of a small pulse in the LS-Dyna data. These are the P and S wave pulses, which are not accounted for by the theoretical model.

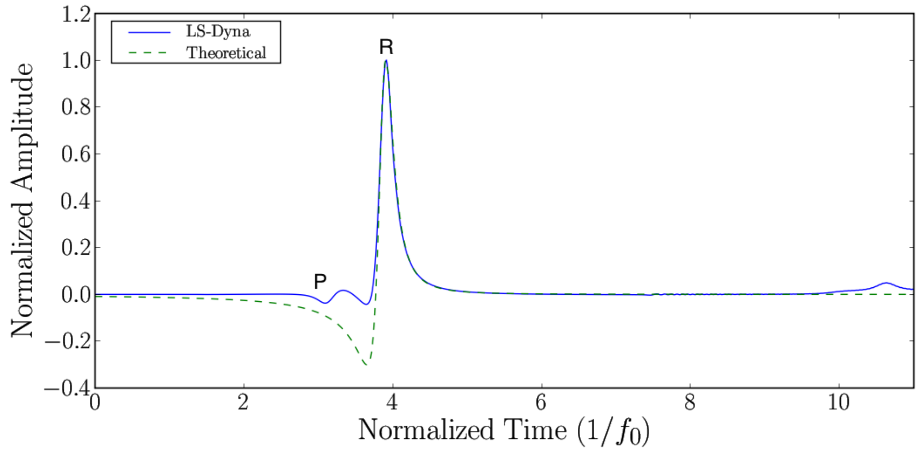


Figure 5.1: Lamb Calibration model ($f_0 = 850kHz$): numerical vs theoretical trace at 41mm (1.32 λ_R for $\lambda_R = 30.6mm$ is the Rayleigh wave wavelength at 75 kHz). P is the p-wave and R is the surface wave.

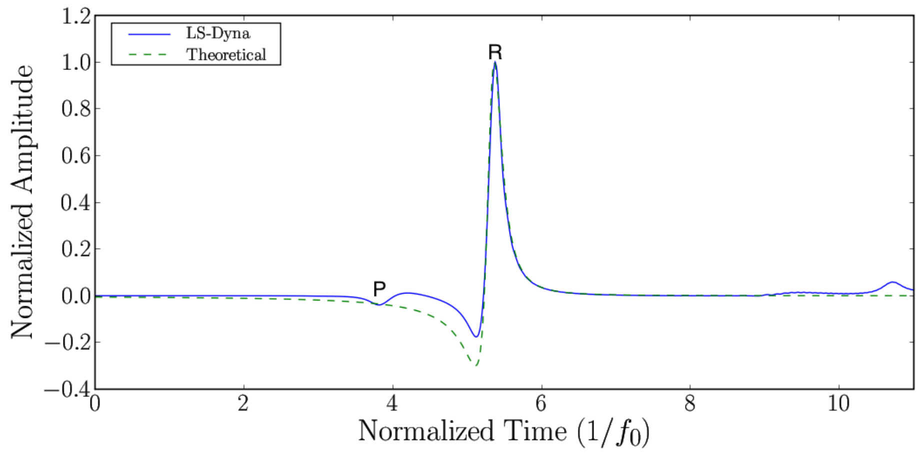


Figure 5.2: Lamb Calibration model ($f_0 = 850kHz$): numerical vs theoretical trace at 83mm (2.71 λ_R for $\lambda_R = 30.6mm$ is the Rayleigh wave wavelength at 75 kHz). P is the p-wave and R is the surface wave.

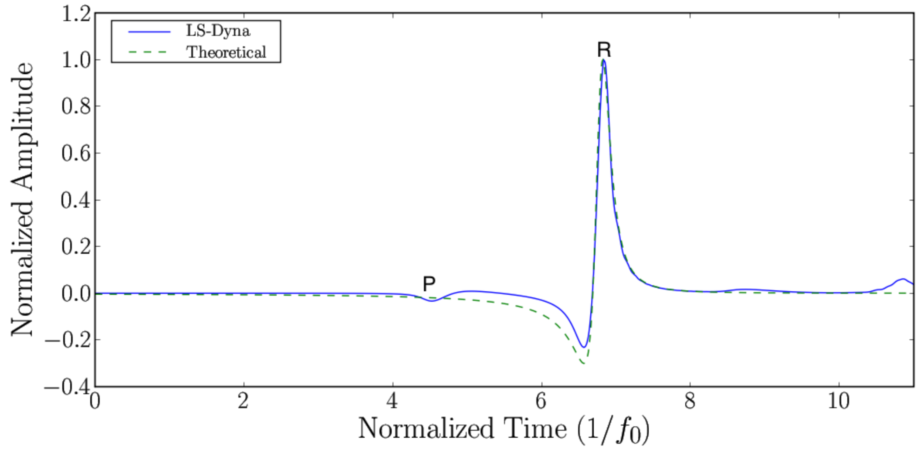


Figure 5.3: Lamb Calibration model ($f_0 = 850kHz$): numerical vs theoretical trace at 125mm (4.08 λ_R for $\lambda_R = 30.6mm$ is the Rayleigh wave wavelength at 75 kHz). P is the p-wave and R is the surface wave.

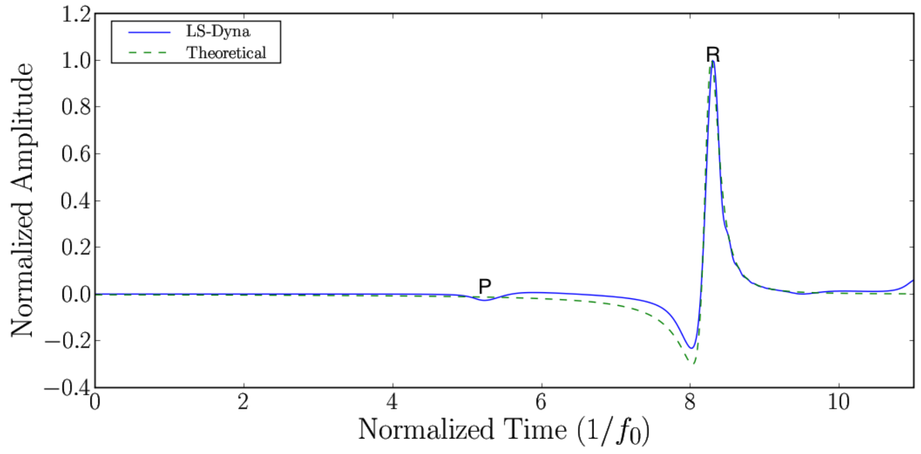


Figure 5.4: Lamb Calibration model ($f_0 = 850kHz$): numerical vs theoretical trace at 166mm (5.42 λ_R for $\lambda_R = 30.6mm$ is the Rayleigh wave wavelength at 75 kHz). P is the p-wave and R is the surface wave.

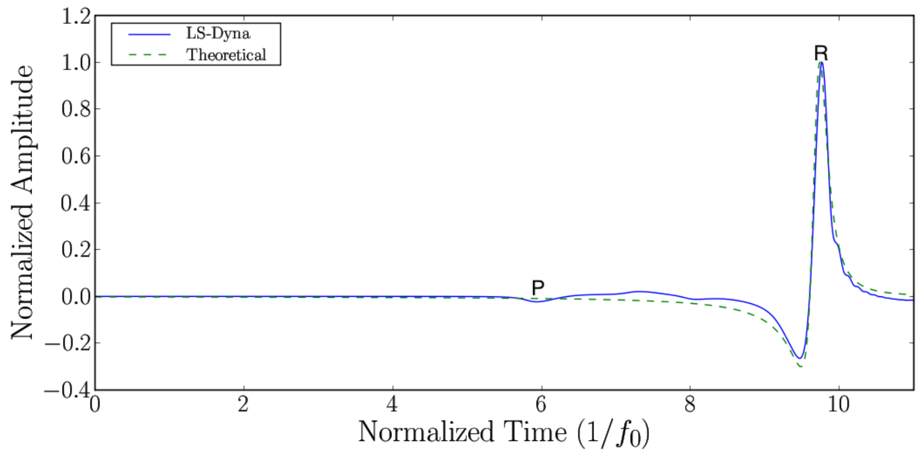


Figure 5.5: Lamb Calibration model ($f_0 = 850kHz$): numerical vs theoretical trace at 208mm (6.80 λ_R for $\lambda_R = 30.6mm$ is the Rayleigh wave wavelength at 75 kHz). P is the p-wave and R is the surface wave.

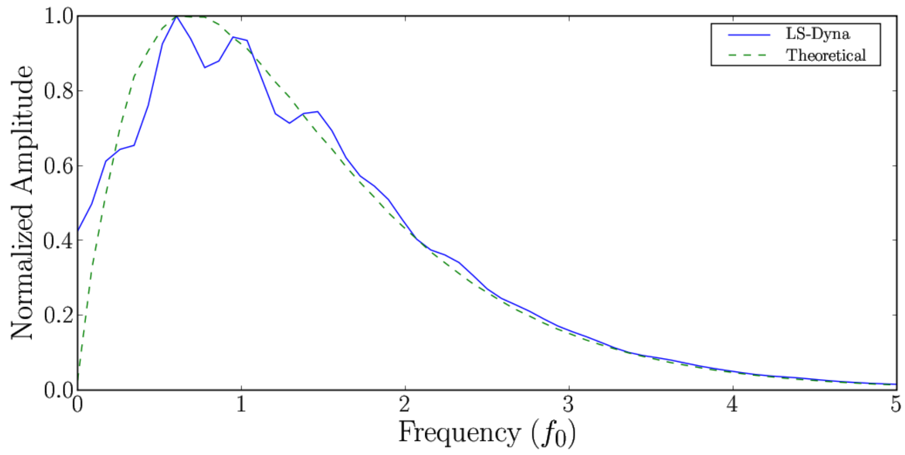


Figure 5.6: Lamb Calibration model ($f_0 = 850kHz$): sample numerical vs theoretical spectra at 166mm (5.42 λ_R for $\lambda_R = 30.6mm$ is the Rayleigh wave wavelength at 75 kHz).

5.1.2 Single Layer Calibration Model

Calibration results for a single layer model (section 4.1.3) are presented in this section. Results are shown at distances from 15 to 90 m (0.75 to 4.5 λ for $V_s = 500m/s$ and $f_0 = 25Hz$) from the source in figures 5.7 through 5.10. Figure 5.7 shows results at 15 m (0.75 λ), close to the source. The match between the Punch application and LS-Dyna is best at this distance. Figure 5.8 through 5.10 show results getting progressively worse as distance increases. Differences are noted by small differences in signal amplitude at the peaks but first arrivals and signal shape, are not affected.

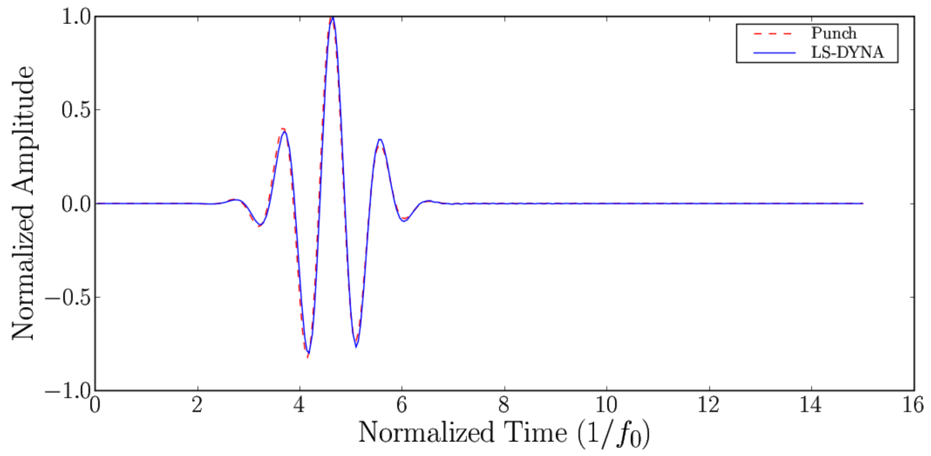


Figure 5.7: Time trace comparison between LS-Dyna and Punch output for single layer model at 15 m from source (0.75 λ for $V_s = 500m/s$ and $f_0 = 25Hz$).

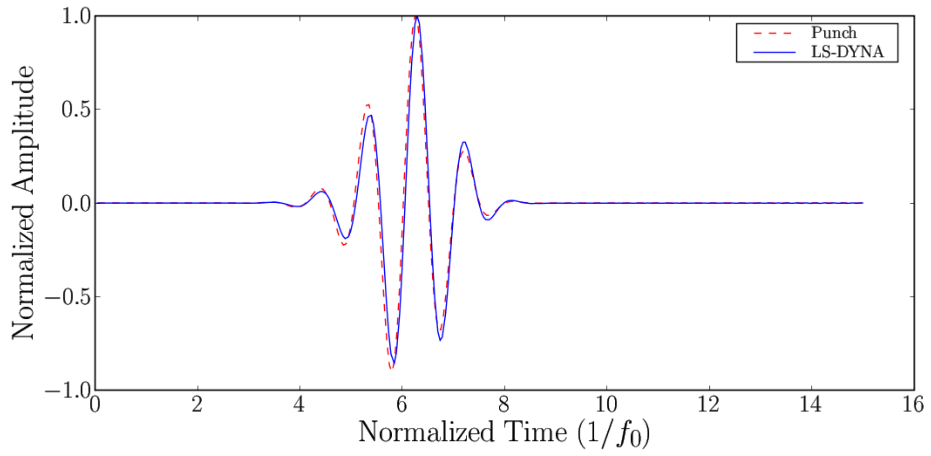


Figure 5.8: Time trace comparison between LS-Dyna and Punch output for single layer model at 45 m from source (2.25 λ for $V_s = 500m/s$ and $f_0 = 25Hz$).

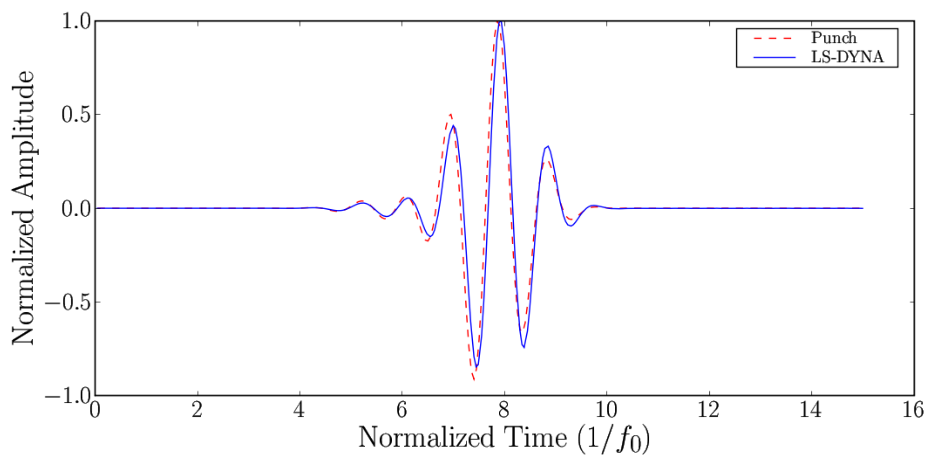


Figure 5.9: Time trace comparison between LS-Dyna and Punch output for single layer model at 75 m from source (3.75 λ for $V_s = 500 m/s$ and $f_0 = 25 Hz$).

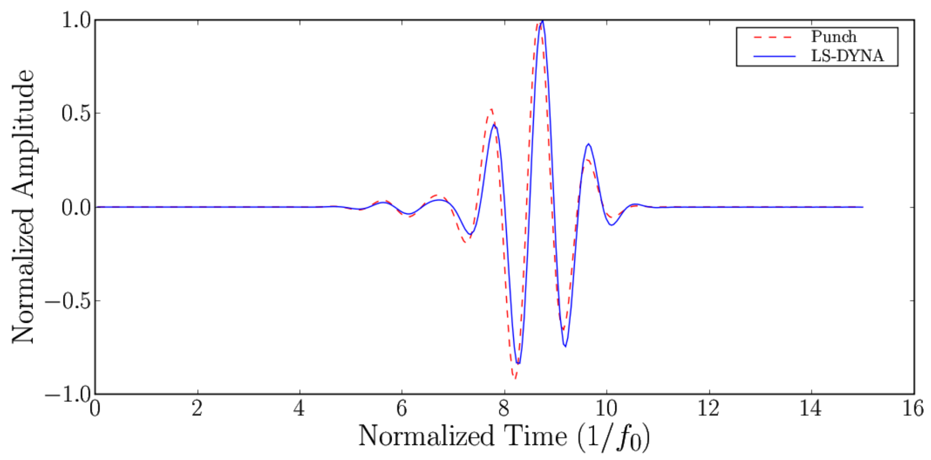


Figure 5.10: Time trace comparison between LS-Dyna and Punch output for single layer model at 90 m from source (4.50 λ for $V_s = 500 m/s$ and $f_0 = 25 Hz$).

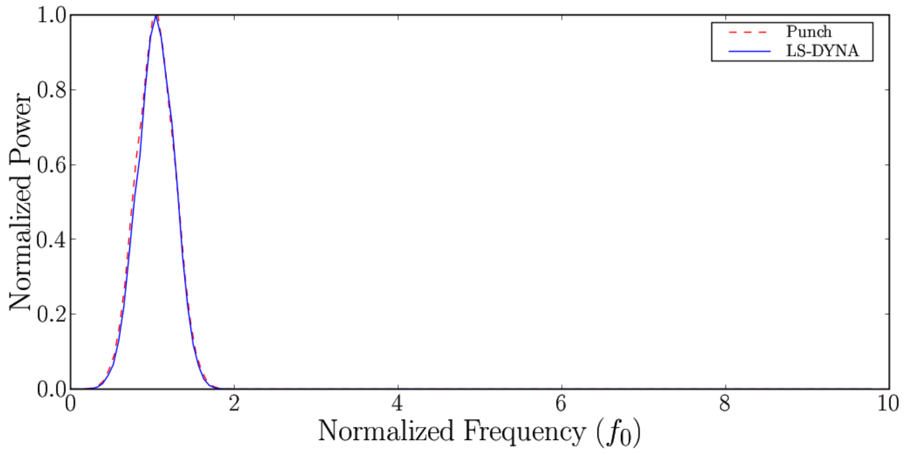


Figure 5.11: Spectrum comparison between LS-Dyna and Punch output for single layer model at 15 m from source (0.75 λ for $V_s = 500m/s$ and $f_0 = 25Hz$).

5.1.3 Three Layer Calibration Model

Calibration results for a three layer calibration model (section 4.1.3) are presented in this section. Similar to the single layer model, results are shown at distances of 15 to 90 m from the source (0.75 to 4.5 λ for $V_s = 200$, top layer shear wave velocity, and $f_0 = 10Hz$) in figure 5.12 through 5.15. It is to be noted that the thickness of the layers is of 5, 2.5 and 10 *lambda* using their respective shear wave velocities of 200 m/s, 400 m/s and 500 m/s.

Figure 5.12 shows results at 15 m (0.75λ), close to the source. The match is still best at this close distance. Figure 5.13 through 5.15 show results getting progressively worse as distance increases. Differences are more pronounced than in the single-layer model but show the same characteristics. All near-field features, and reflections, are accurately matched between both simulations. As distance increases the LS-Dyna first main oscillation, seen on figure 5.15, is marginally late compared to the Punch simulation. This phase difference is not propagated to the rest of the signal.

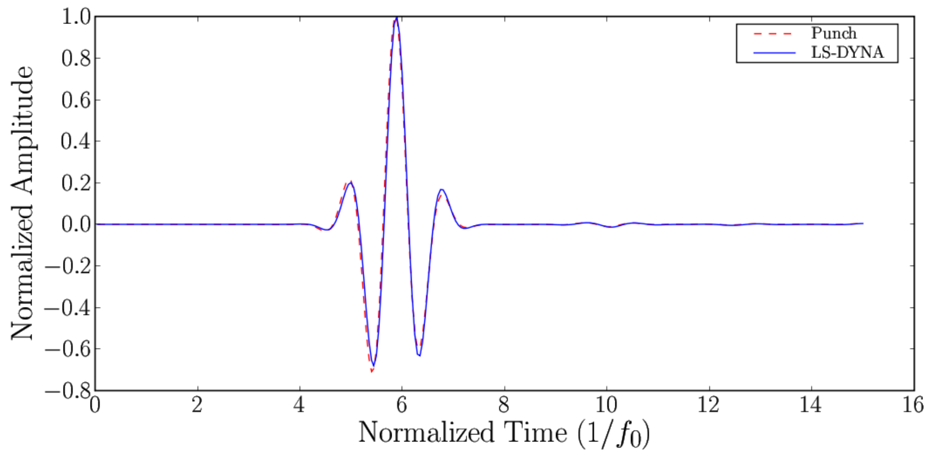


Figure 5.12: Time trace comparison between LS-Dyna and Punch output for three layer model at 15 m from source (0.75 λ for $V_s = 200m/s$ and $f_0 = 10Hz$).

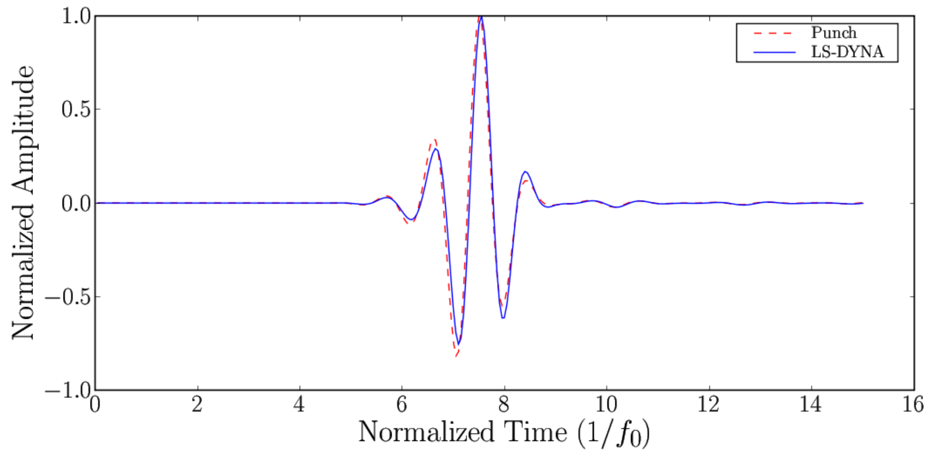


Figure 5.13: Time trace comparison between LS-Dyna and Punch output for three layer model at 45 m from source (2.25 λ for $V_s = 200m/s$ and $f_0 = 10Hz$).

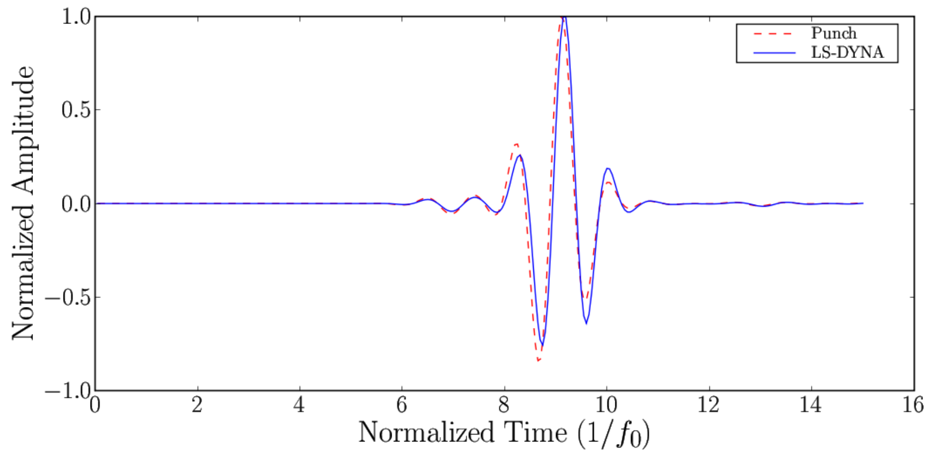


Figure 5.14: Time trace comparison between LS-Dyna and Punch output for three layer model at 75 m from source (3.75 λ for $V_s = 200m/s$ and $f_0 = 10Hz$).

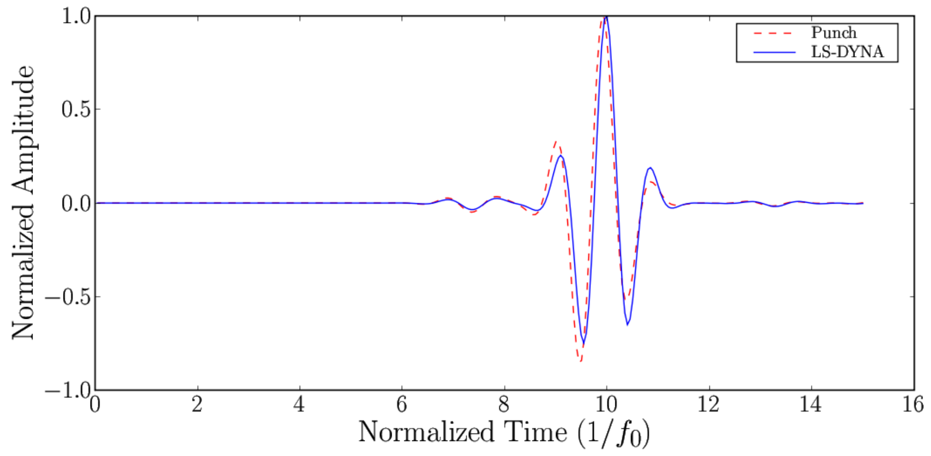


Figure 5.15: Time trace comparison between LS-Dyna and Punch output for three layer model at 90 m from source (4.5 λ for $V_s = 200m/s$ and $f_0 = 10Hz$).

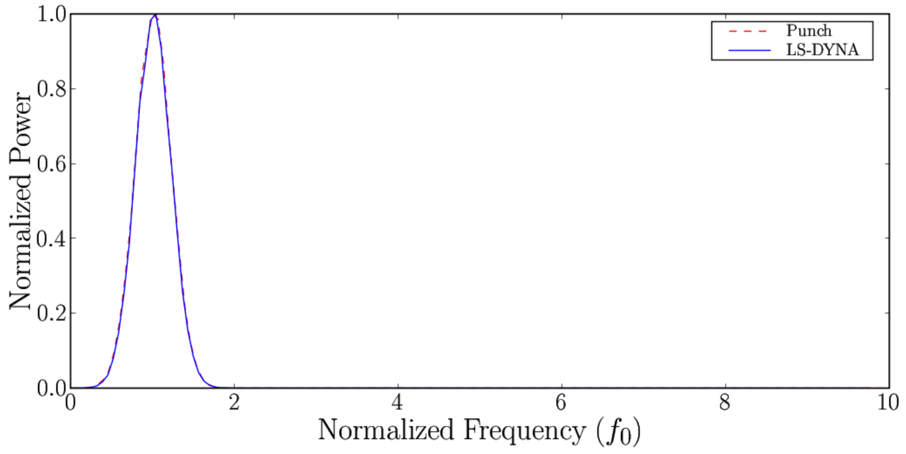


Figure 5.16: Spectrum comparison between LS-Dyna and Punch output for three layer model at 15 m from source (0.75 λ for $V_s = 200m/s$ and $f_0 = 10Hz$).

5.1.4 Numerical Simulations

Wave propagation inside the models

Data from numerical simulations discussed in section 4.1.4 are presented in this section. Signals are based on the central, axisymmetric, axis of each model unless otherwise noted. The axisymmetric axis has zero displacement in the horizontal direction at all times. Hence, only vertical displacement is shown. Figures 5.17 through 5.25 show the frequency-wavenumber (F-K) plot, and the time history plots for axisymmetric, top, right and bottom boundaries of the "m1a" (H=9.24 λ_0, D= 18.48 λ_0 using $f_0 = 850kHz, \lambda_0 = 5.412mm$) model (table 4.2.5). Top, right and bottom boundaries are presented to better understand wave patterns in the axisymmetric axis and are presented for both vertical and horizontal displacement. The "m1a" model is presented as its small size presents all of the wave features. Figure 5.26 shows the same plot, except for top, right and bottom boundaries, for the longer specimen "m4a" (H=55.44 λ_0, D= 18.48 λ_0). This long and thin specimen shows the effect of close boundaries on the central axis and has similar pattern to "m1a". Data for other models presents similar wave patterns and is provided in section E.2.1. Time history traces for all presented figures use individually normalized time traces at each height for added visual detail.

Figure 5.28 and 5.29 show central axis F-K plot vertical displacement for the low frequency source. The very long pulse makes it prohibitive to identify individual wave reflections as is done with the higher frequencies. Data for other models presents similar wave patterns and is provided in section E.2.2. The same is also true for low frequency damped models (see Appendix E.2.3).

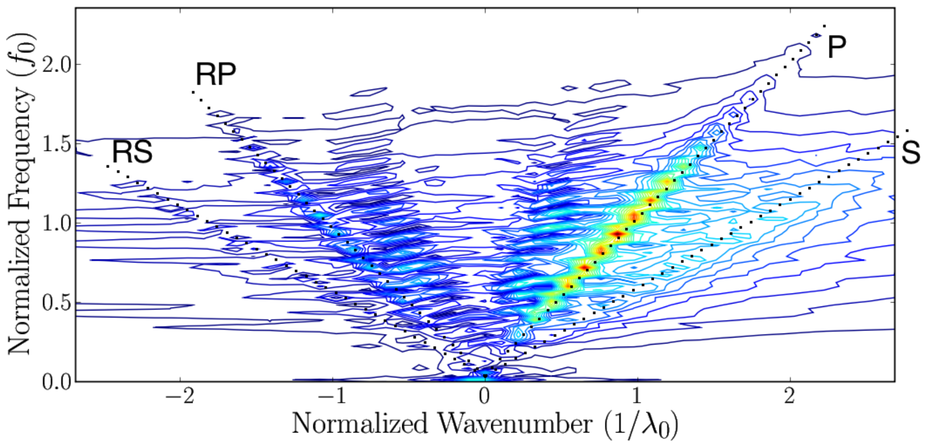


Figure 5.17: Frequency-wavenumber plot for model "m1a" (H=9.24 λ_0, D= 18.48 λ_0) using high frequency excitation ($f_0 = 850kHz, \lambda_0 = 5.412mm$). P indicates the main p-wave, S indicates the main shear wave, RP is the reflected p-wave and RS is the reflected shear wave.

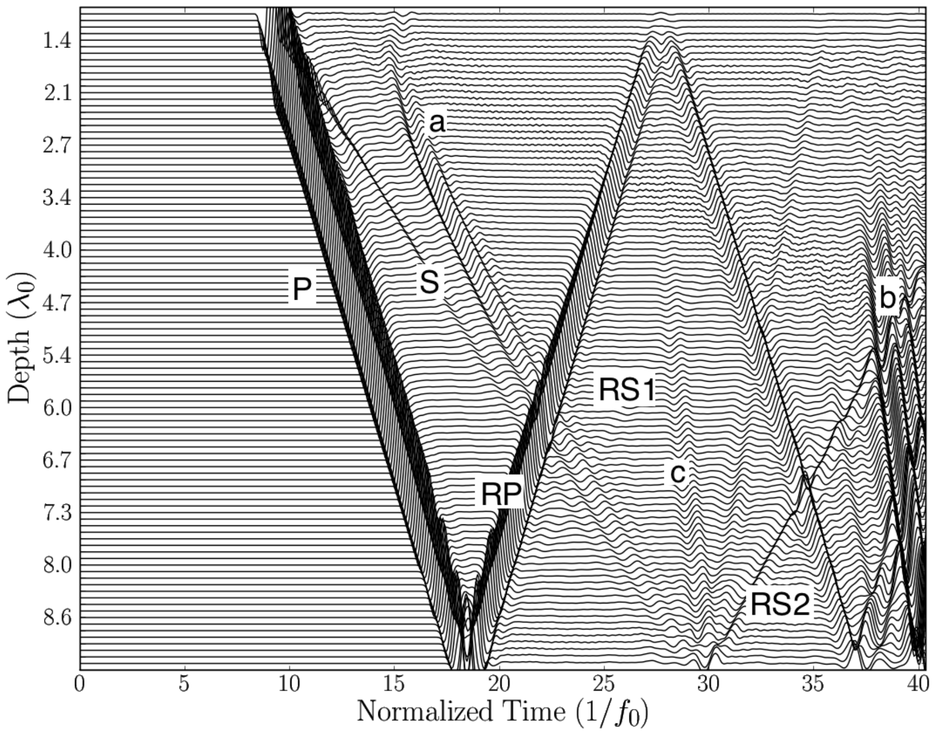


Figure 5.18: Time history plot of central axis of model "m1a" (H=9.24 λ_0, D= 18.48 λ_0) using high frequency excitation ($f_0 = 850kHz, \lambda_0 = 5.412mm$). P, RP are the main p-wave and first reflected p-wave respectively, S is the high intensity area of the shear wave originating from the transducer, RS1 is the reflected and mode converted shear wave created at the bottom boundary from P, RS2 is the reflected shear wave from S, "a" is the edge of S from the transducer's corners, "b" is the mode converted p-wave created by the surface wave impacting the side boundary and "c" is the p-wave reflection from the sides.

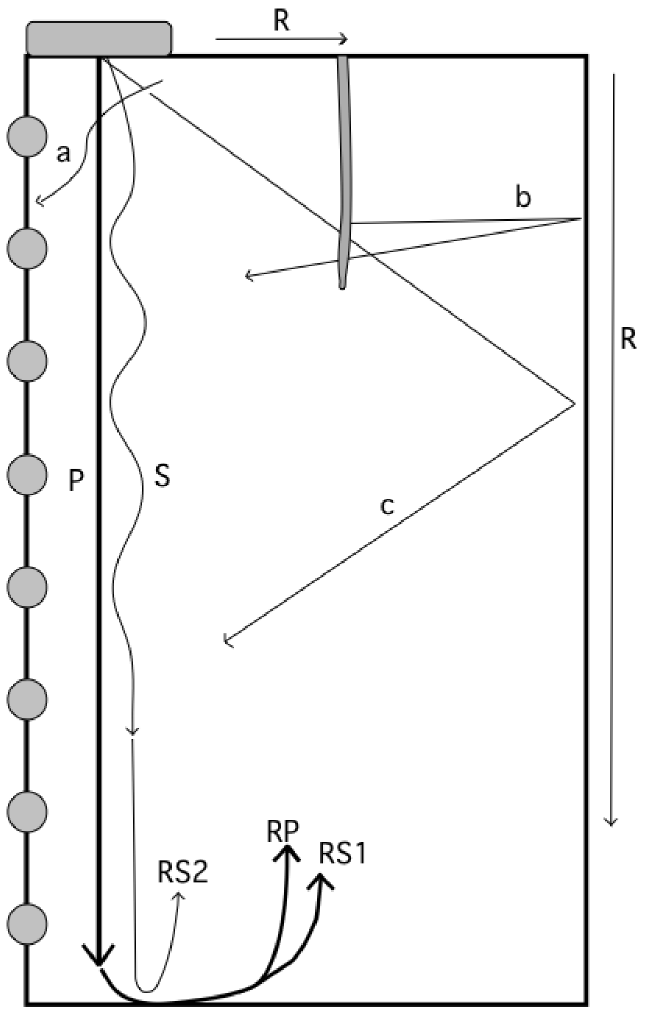


Figure 5.19: Diagram of selected wavefronts in cylindrical specimen. All presented wavefronts represent those shown in figure 5.18. P, RP are the main p-wave and first reflected p-wave respectively, S is the high intensity area of the shear wave originating from the transducer, RS1 is the reflected and mode converted shear wave created at the bottom boundary from P, RS2 is the reflected shear wave from S, "a" is the edge of S from the transducer's corners, "b" is the mode converted p-wave created by the surface wave impacting the side boundary and "c" is the p-wave reflection from the sides.

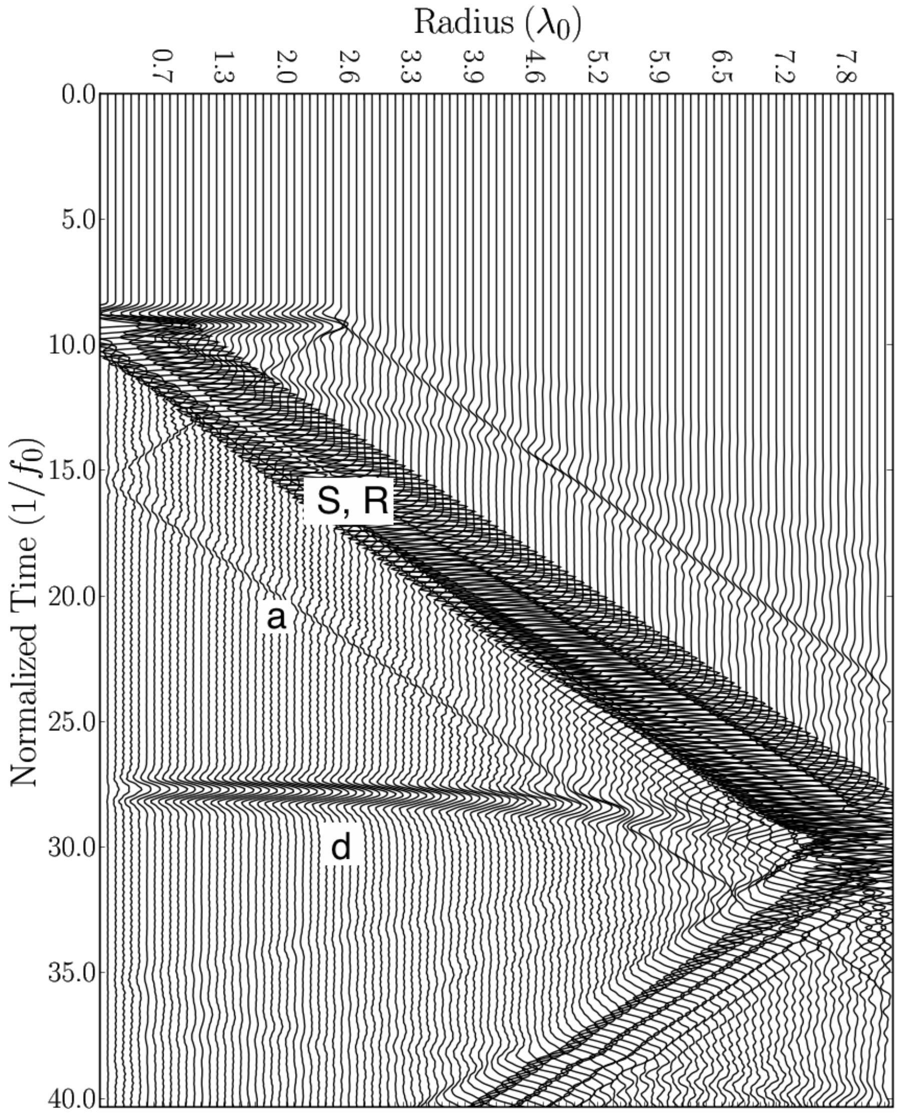


Figure 5.20: Time history plot of vertical displacement of the top axis of model "m1a" (H=9.24 λ_0, D= 18.48 λ_0) using high frequency excitation ($f_0 = 850kHz, \lambda_0 = 5.412mm$). S and "a" are the centre and edge of the shear wave as described in figure 5.18, R is the surface wave, and d is the reflected p-wave from the bottom. S and R are hard to distinguish due to the transducer's width creating a wide pulse in terms of distance on the surface.

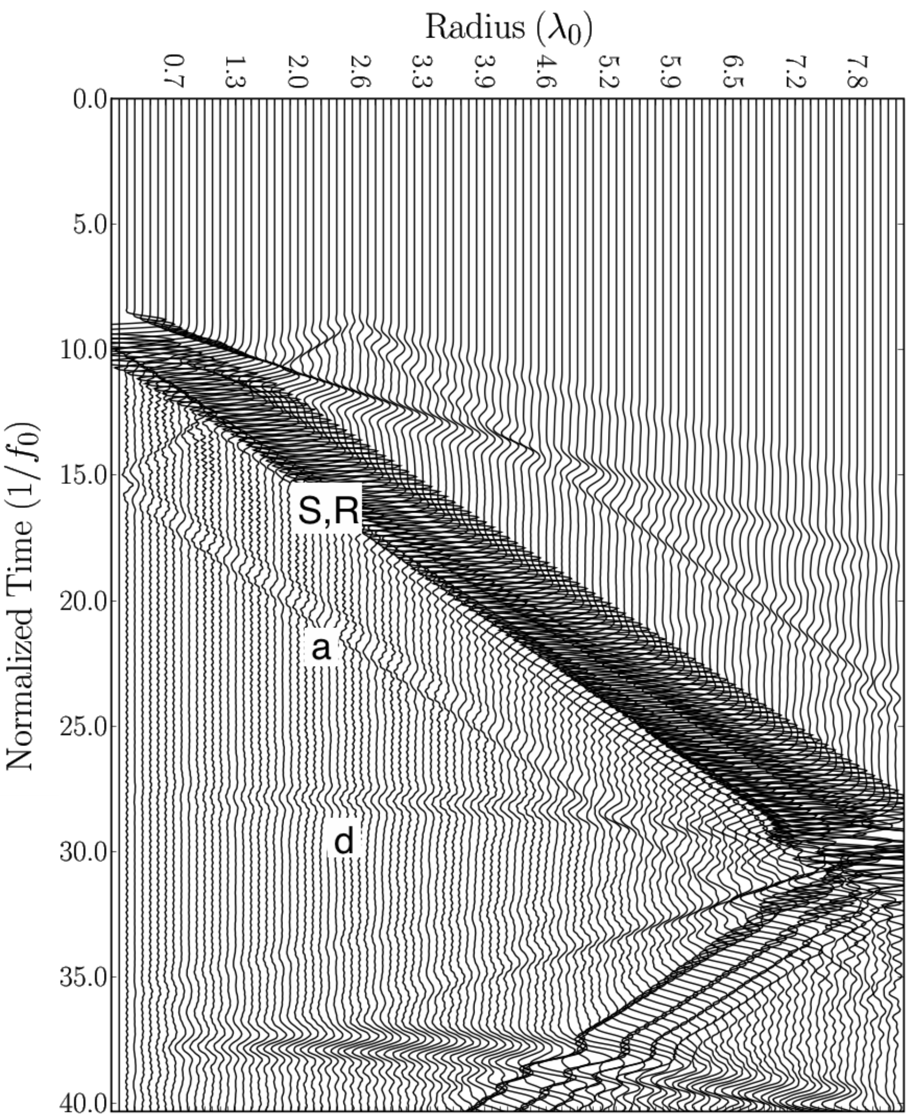


Figure 5.21: Time history plot of horizontal displacement of the top axis of model "m1a" (H=9.24 λ_0, D=18.48 λ_0) using high frequency excitation ($f_0 = 850kHz, \lambda_0 = 5.412mm$). S and "a" are the centre and edge of the shear wave as described in figure 5.18, R is the surface wave, and d is the reflected p-wave from the bottom. S and R are hard to distinguish due to the transducer's width creating a wide pulse in terms of distance on the surface.

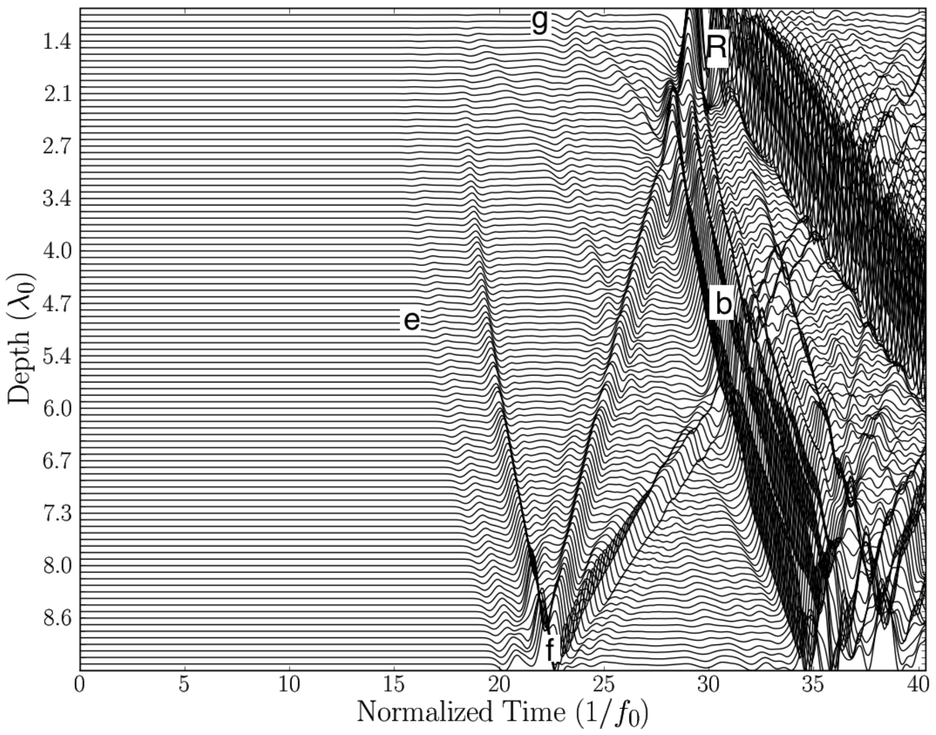


Figure 5.22: Time history plot of vertical displacement of the right axis of model “m1a” (H=9.24 λ_0, D=18.48 λ_0) using high frequency excitation ($f_0 = 850kHz, \lambda_0 = 5.412mm$). “b” is the mode converted p-wave created from the surface wave impact on the side boundary as desribed in figure 5.18, R is the surface wave originating from the top surface, g is the shear originating from the transducer’s closest corner, and e is the p-wave arrival on the sides.

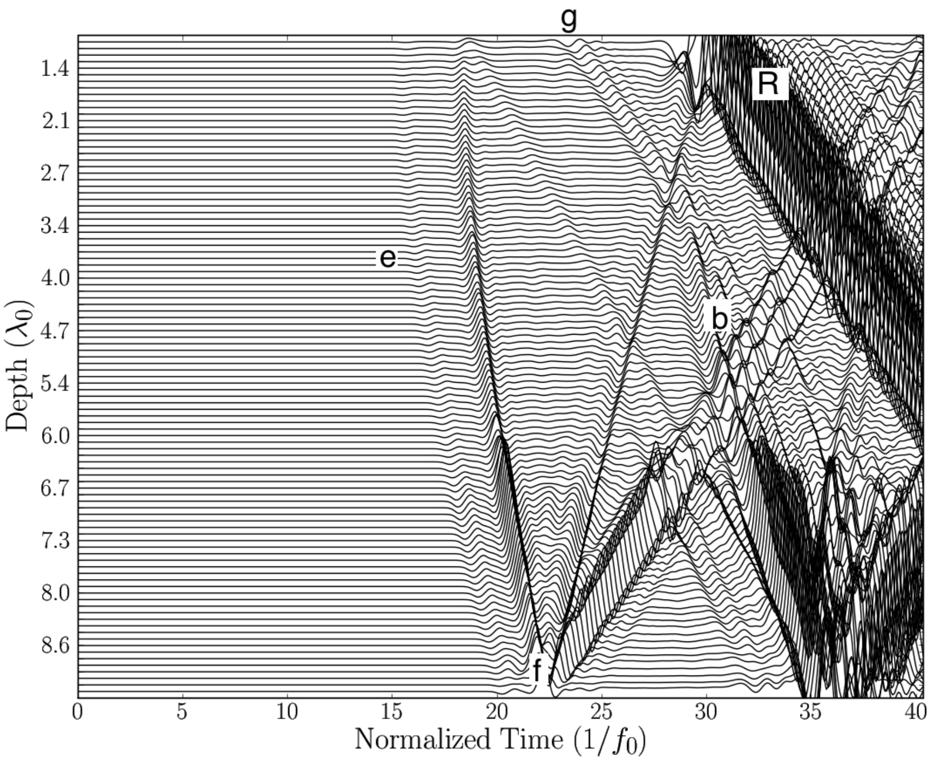


Figure 5.23: Time history plot of horizontal displacement of the right axis of model "m1a" (H=9.24 λ_0, D=18.48 λ_0) using high frequency excitation ($f_0 = 850kHz, \lambda_0 = 5.412mm$). "b" is the mode converted p-wave created from the surface wave impact on the side boundary as desribed in figure 5.18, R is the surface wave originating from the top surface, g is the shear originating from the transducer's closest corner, e is the p-wave arrival on the sides, and f is the interaction between mode converted waves created by e impacting the bottom right corner.

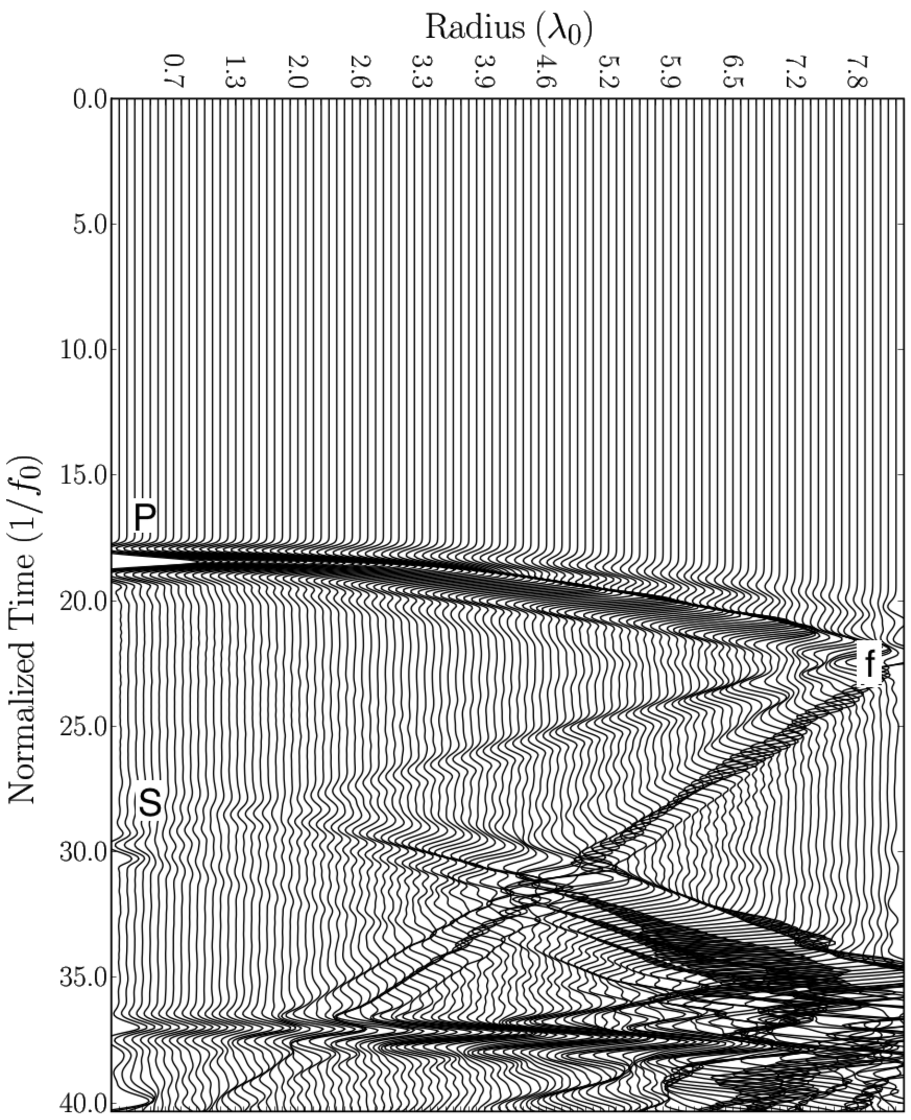


Figure 5.24: Time history plot of vertical displacement of the bottom axis of model "m1a" (H=9.24 λ_0, D=18.48 λ_0) using high frequency excitation ($f_0 = 850kHz, \lambda_0 = 5.412mm$). P is the p-wave arrival, f is the interaction between mode converted waves from P impacting the corner as well as the p-wave reflection from the right boundary, and S is the arrival of the first shear wave.

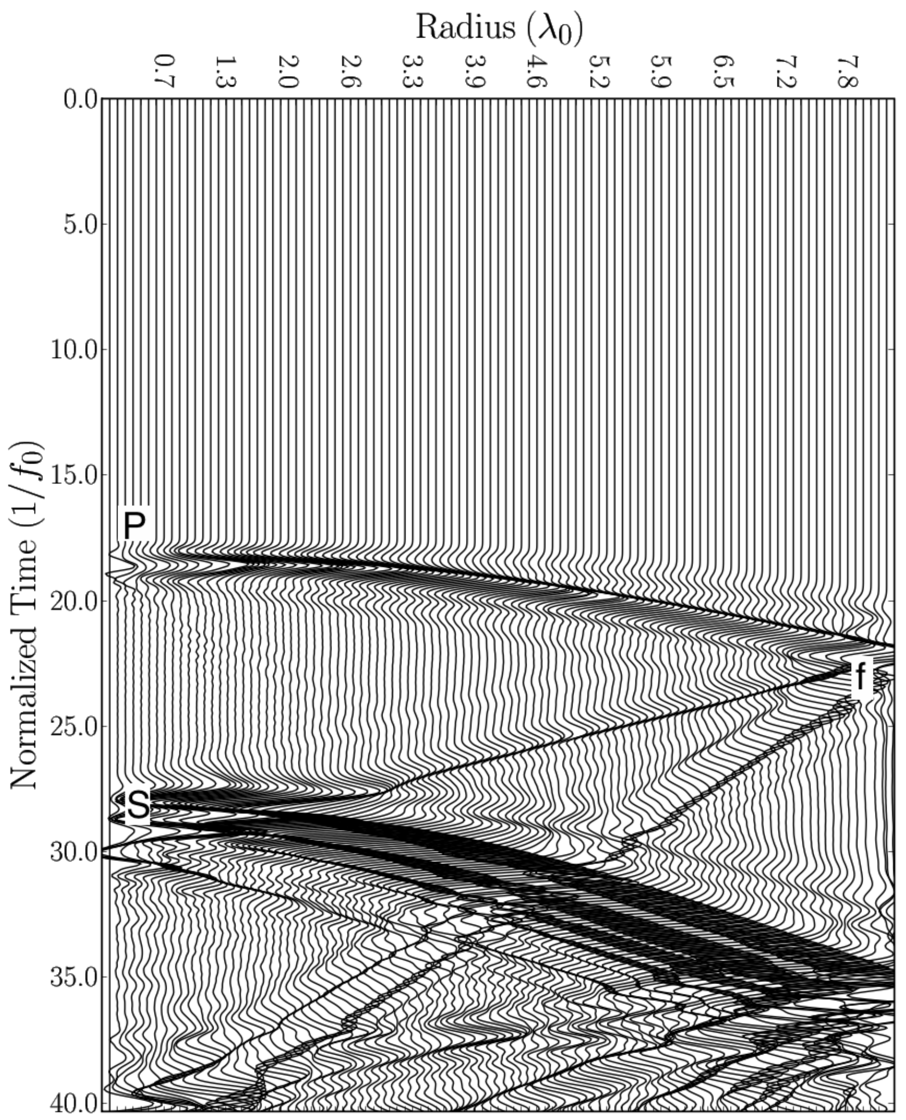


Figure 5.25: Time history plot of vertical displacement of the bottom axis of model “m1a” (H=9.24 λ_0, D=18.48 λ_0) using high frequency excitation ($f_0 = 850kHz, \lambda_0 = 5.412mm$). P is the p-wave arrival, f is the interaction between mode converted waves from P impacting the corner as well as the p-wave reflection from the right boundary, and S is the arrival of the first shear wave.

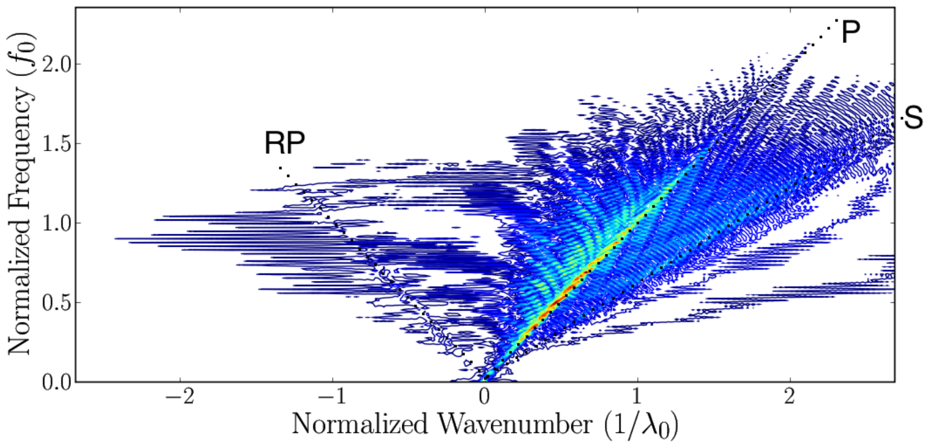


Figure 5.26: Frequency-wavenumber plot for model “m4a” (H=55.44 λ_0, D= 18.48 λ_0) using high frequency excitation ($f_0 = 850kHz, \lambda_0 = 5.412mm$). P is the p-wave, S is the shear wave, and RP is the reflected p-wave.

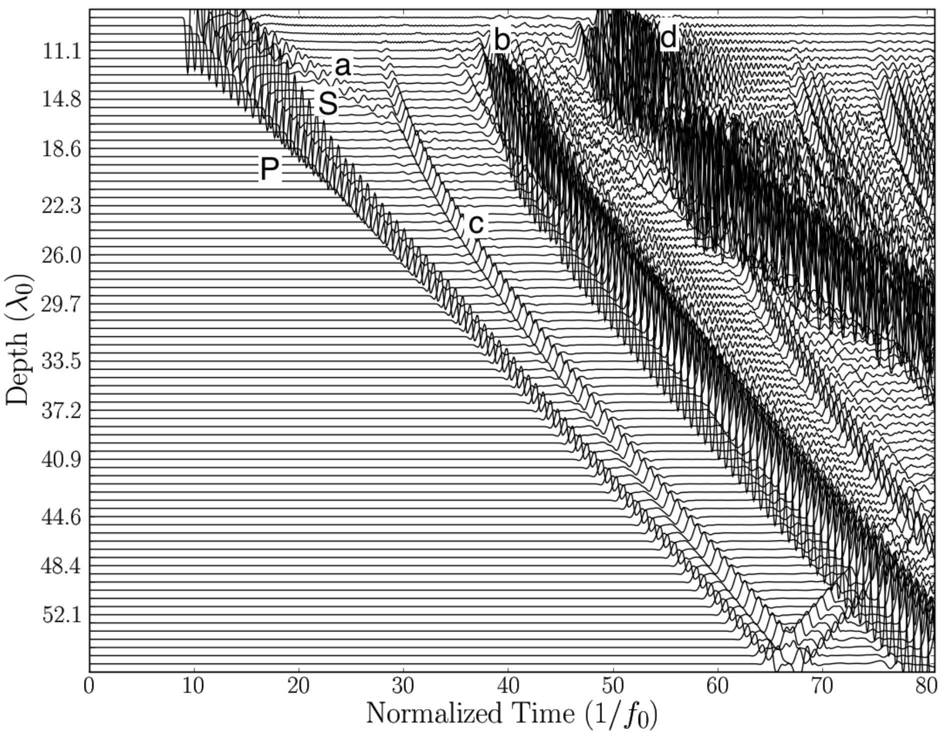


Figure 5.27: Time history plot of vertical displacement of the bottom axis of model “m1a” (H=9.24 λ_0, D=18.48 λ_0) using high frequency excitation ($f_0 = 850kHz, \lambda_0 = 5.412mm$). P is the first p-wave, S is the high intensity area of the shear wave originating from the transducer, “a” is the edge of S from the transducer’s corners, “b” is the mode converted p-wave created by the surface wave impacting the side boundary, “c” is the p-wave reflection from the side, and “d” is the mode converted shear wave created by the surface wave impacting the side.

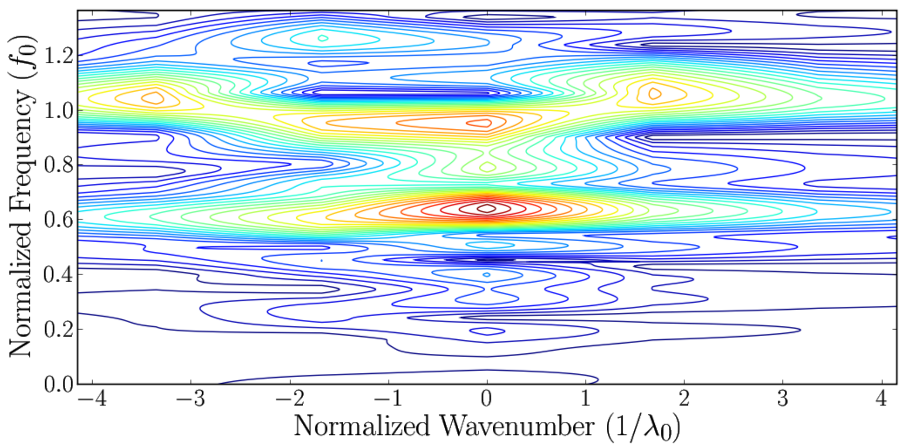


Figure 5.28: Frequency-wavenumber plot for model “m1a” (H=0.60 λ_0, D=1.20 λ_0) at low frequency excitation ($f_0 = 55kHz, \lambda_0 = 83.0mm$).

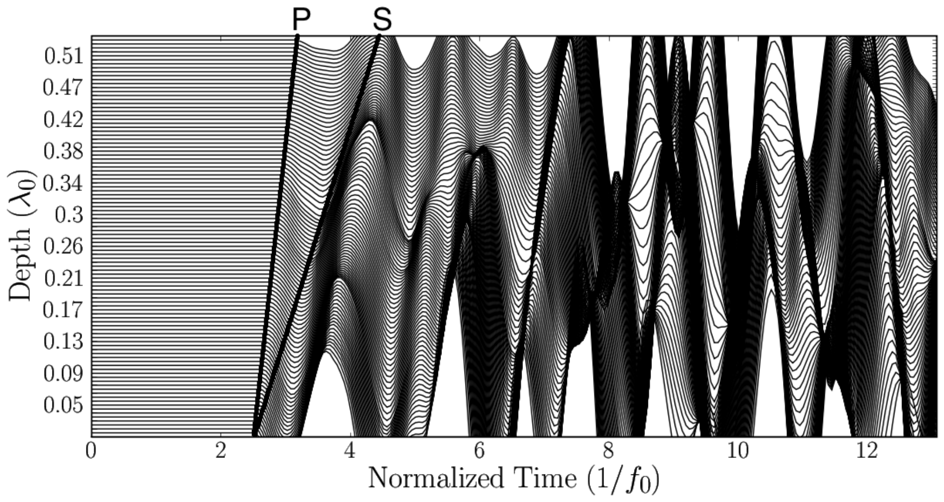


Figure 5.29: Time history plot of model “m1a” (H=0.60 λ_0, D=1.20 λ_0) at low frequency excitation ($f_0 = 55kHz, \lambda_0 = 83.0mm$). P is the p-wave arrival line and S is the computed shear wave arrival line.

First arrival in low frequency models (55 kHz transducer) – damped versus undamped

This section presents results of first arrivals between the damped an undamped models (section 4.1.4) when excited by the low frequency source (55 kHz). Undamped time history plots are similar to those shown in figure 5.29 and are presented in Appendix E.2.2. Damped seismic plots do not provide clear indication of wave front arrival as the excitation is the same large pulse as in the undamped model. The damping curve used is presented in figure 4.8 and has damping parameters $\alpha = 4435.2$, and $\beta = 1.5 * 10^{-7}$. These value were chosen to have material damping of approximately 2.5% at the main frequency (55 kHz).

First arrivals in the undamped model are, as expected, a straight line in distance (figure 5.29) with slope being the model's velocity $V_p = 4566m/s$. Damped models, on the other hand, show a non-linear P-wave first arrival slope (figure 5.30 to 5.32) with increased curvature as the model gets longer. This curvature would be interpreted as a slower overall velocity for the specimen given that UPVM setups measure velocity only from the received first arrival. This curvature is observed on all specimen as shown in Appendix E.2.3.

Figure 5.33 to 5.35 compare the damped and undamped model time trace at the receiver location (centre of bottom boundary) for specimen of different heights. Figures for all specimen are presented in Appendix E.2.4. Compiled results representing the difference in first arrival are shown in table 5.1.4 and plotted in figure 5.36. Information of first arrival for the longest damped models is not available due to the signal being damped below the numerical precision of LS-Dyna (double precision).

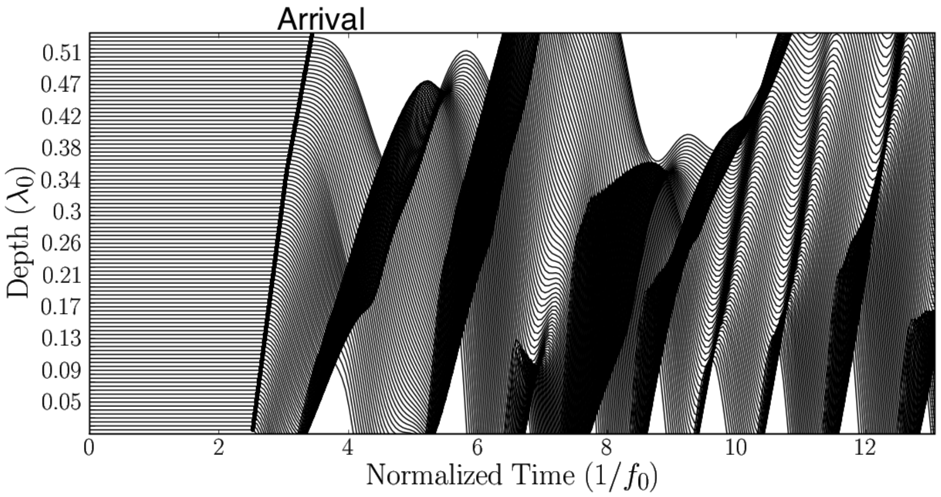


Figure 5.30: Time history plot of damped model ($D_{min} = 0.025, \alpha = 4435.2, \beta = 1.5 * 10^{-7}$) "m1a" (H=0.60 λ_0, D=1.20 λ_0) at low frequency excitation ($f_0 = 55kHz, \lambda_0 = 83.0mm$)

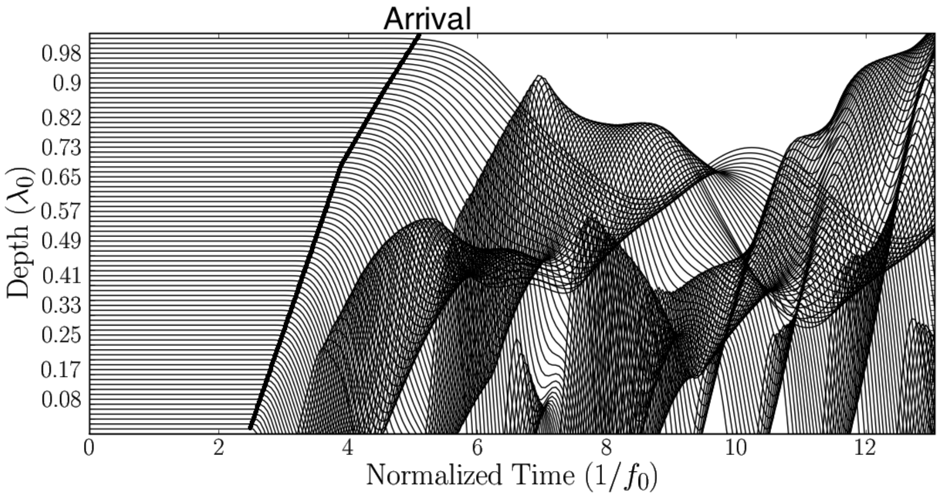


Figure 5.31: Time history plot of damped model ($\alpha = 4435.2, \beta = 1.5{*}10^{-7}$) "m2a" (H=1.20 λ_0, D=1.20 λ_0) at low frequency excitation ($f_0 = 55kHz, \lambda_0 = 83.0mm$)

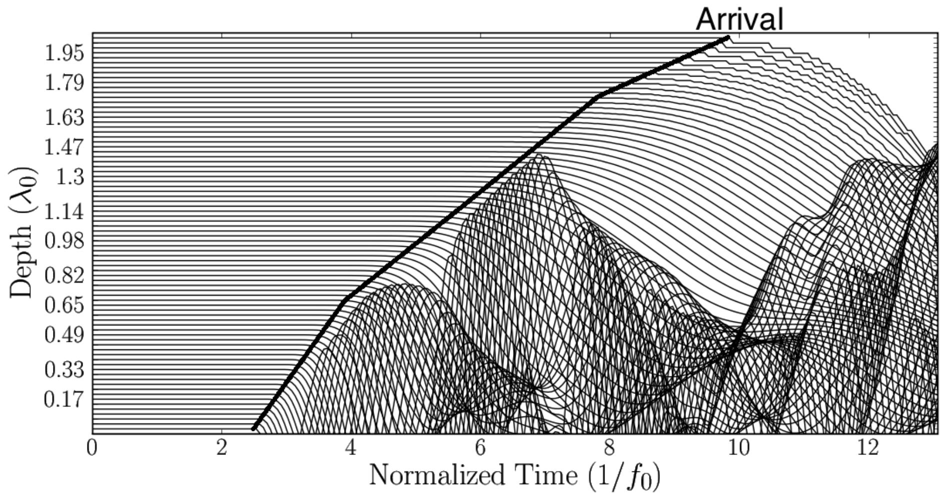


Figure 5.32: Time history plot of damped model ($D_{min} = 0.025, \alpha = 4435.2, \beta = 1.5 * 10^{-7}$) "m3a" (H=2.41 λ_0, D=1.20 λ_0) at low frequency excitation ($f_0 = 55kHz, \lambda_0 = 83.0mm$). The coarse amplitudes at high heights are caused by limitations in the numerical precision of the solver.

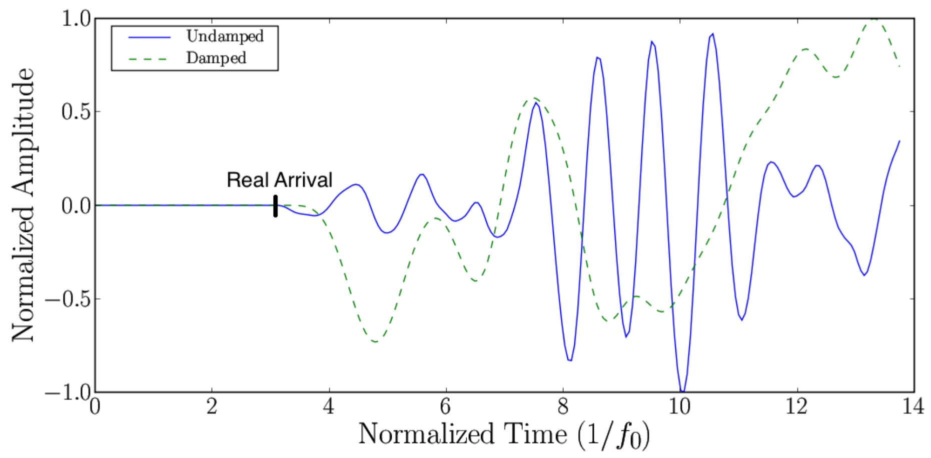


Figure 5.33: Comparison of time signal between damped ($D_{min} = 0.025, \alpha = 4435.2, \beta = 1.5*10^{-7}$) and undamped model "m1a" (H=0.60 λ_0, D=1.20 λ_0) at low frequency excitation ($f_0 = 55kHz, \lambda_0 = 83.0mm$)

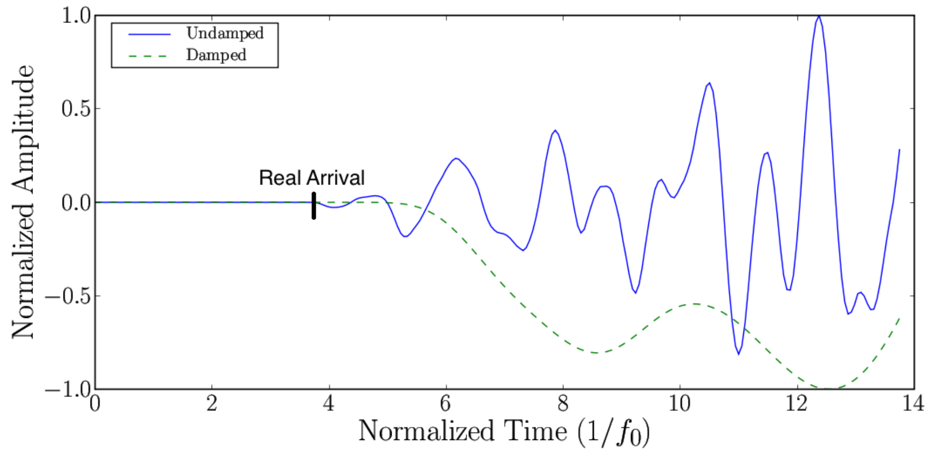


Figure 5.34: Comparison of time signal between damped ($D_{min} = 0.025, \alpha = 4435.2, \beta = 1.5*10^{-7}$) and undamped model "m2a" (H=1.20 λ_0, D=1.20 λ_0) at low frequency excitation ($f_0 = 55kHz, \lambda_0 = 83.0mm$)

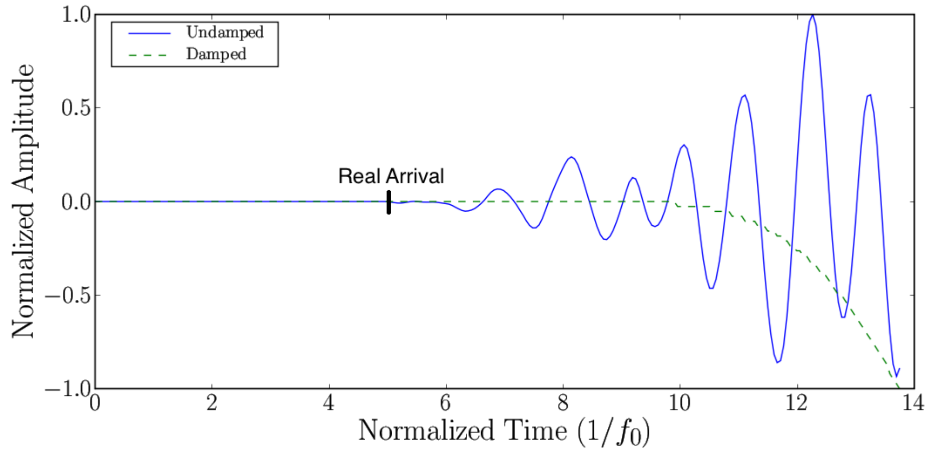


Figure 5.35: Comparison of time signal between damped ($D_{min} = 0.025, \alpha = 4435.2, \beta = 1.5*10^{-7}$) and undamped model "m3a" (H=2.41 λ_0, D=1.20 λ_0) at low frequency excitation ($f_0 = 55kHz, \lambda_0 = 83.0mm$)

Table 5.1: Comparison of arrival time from damped and undamped numerical simulations.

Model	$t_{undamped}$ (ms)	t_{damped} (ms)	$V_{undamped}$ (m/s)	V_{damped} (m/s)	% Difference
m1a	0.011	0.018	4546	2778	38.8
m1b	0.011	0.017	4546	2941	35.3
m1c	0.011	0.018	4546	2778	38.8
m2a	0.022	0.046	4546	2174	52.2
m2b	0.022	0.046	4546	2174	52.2
m2c	0.022	0.044	4546	2273	50.0
m3a	0.044	0.134	4546	1493	67.2
m3b	0.044	0.134	4546	1493	67.2
m3c	0.044	0.134	4546	1493	67.2
m4a	0.065	N/A	4615	N/A	N/A
m4b	0.065	N/A	4615	N/A	N/A
m4c	0.065	N/A	4615	N/A	N/A

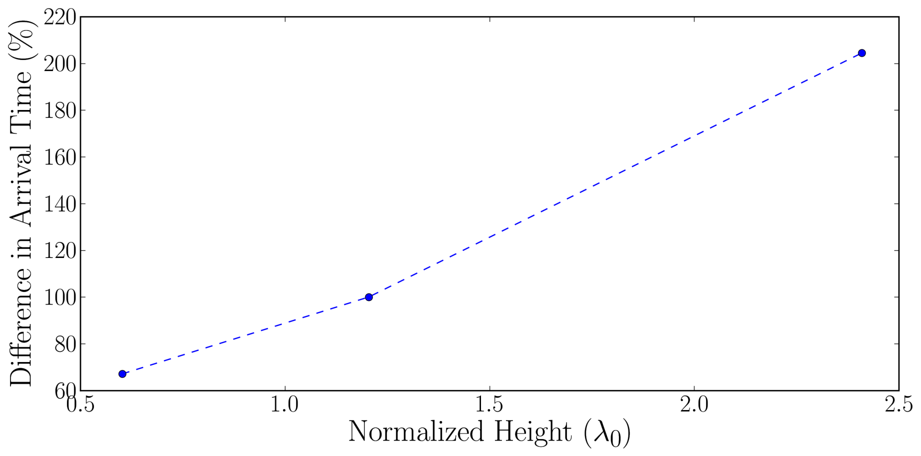
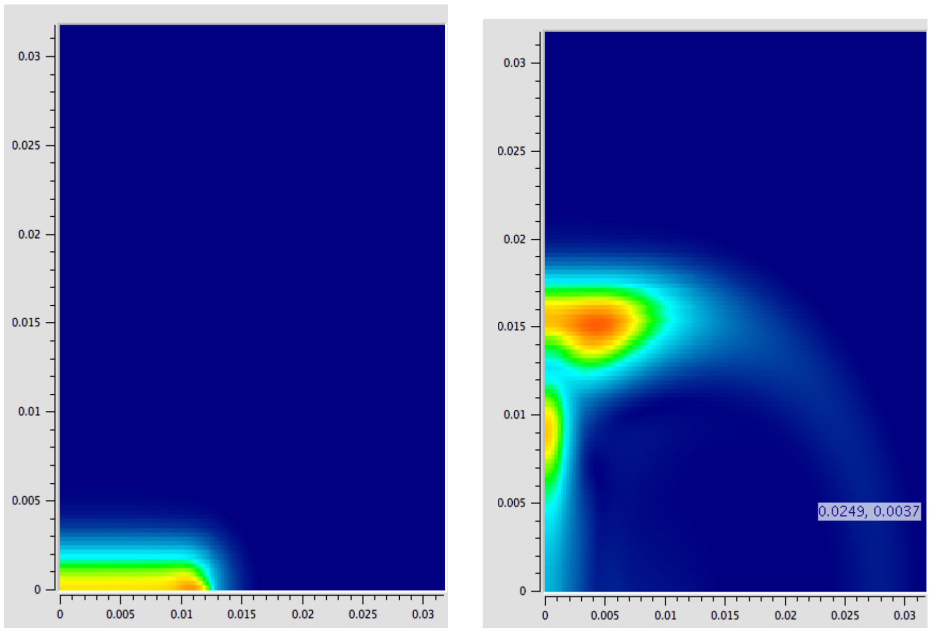


Figure 5.36: Difference in Arrival time (in ms) by height between damped and undamped numerical models

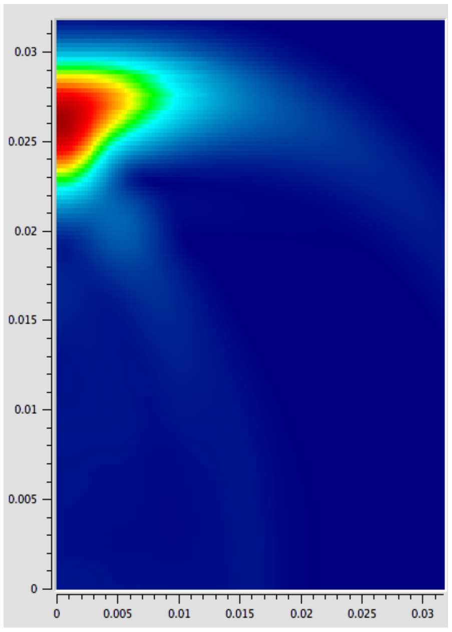
5.1.5 Transducer Field

This section presents the fields of p-waves from the experimental transducers presented in section 4.1.4. Figures 5.37 show the field of the high frequency source as it progresses in time. Similarly, figures 5.38 show the progress in time of the low frequency transducer source.

The high-frequency transducer field produces localized high pressure zones that are focused near the central axis. On the other hand, the low frequency transducer produces a field that is analogous to a point source and thus, provides no focusing. This implies that the high frequency source have a strong initial p-wave front as compared to the low frequency source, all else being equal.

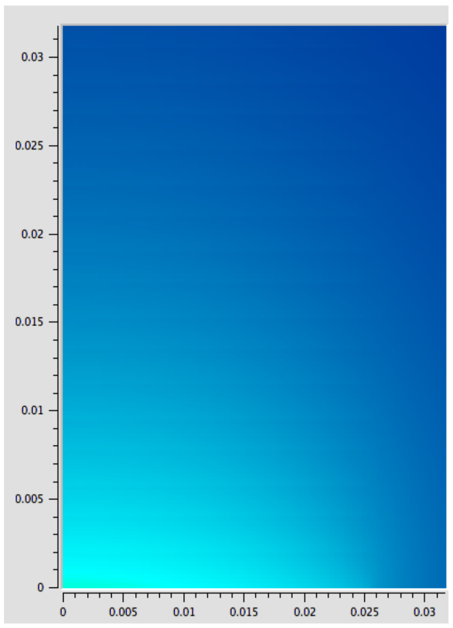


(a) (b)

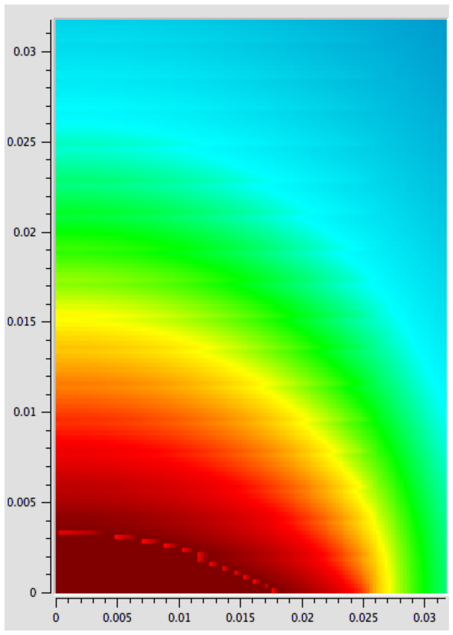


(c)

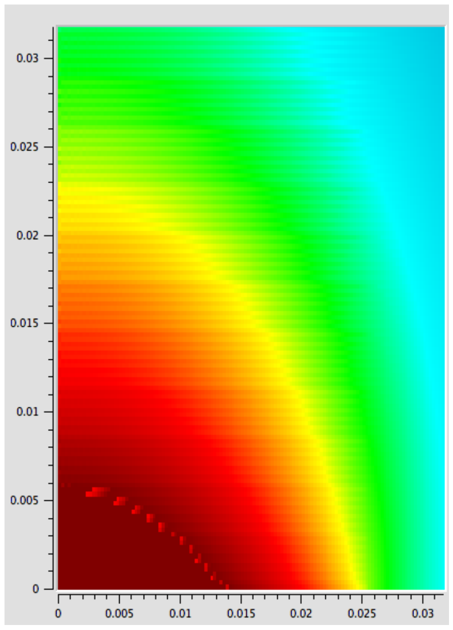
Figure 5.37: Pressure envelope of 1M transducer during excitation. (a) Near-field (less than λ), (b) mid-field (1λ to 3λ) and (c) far field (above 3λ) where $\lambda = 4.8mm$



(a)



(b)



(c)

Figure 5.38: Pressure envelope of 50K transducer during excitation. The field behaves as a point source since the source transducer is smaller (1/2 λ) than the main wavelength. (a) during excitation, and (b) mid-excitation and (c) late excitation.

5.2 Experimental Velocity Measurements

This section presents experimental results of first arrival times on all tested mortar and concrete specimens. Average results for each specimen (see Appendix F.1.3) are compiled in graphical form in figure 5.39 and 5.40 in terms of velocities while figure 5.41 and 5.42 compares first arrival times in terms of percentage difference. Captured waveforms for different readings on a single specimen are presented in figure 5.43 and 5.44 and show good waveform consistency in first pulse arrival between independent readings. Sample waveforms for each specimens are presented in Appendix F.1.

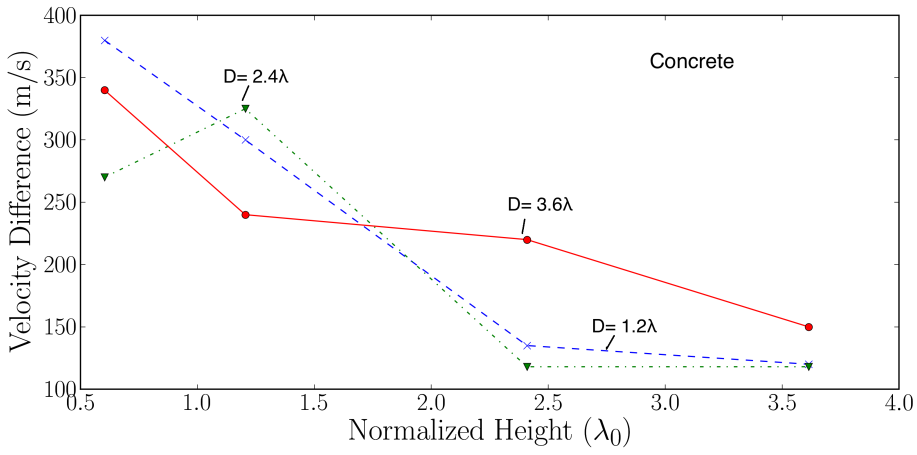


Figure 5.39: Wave velocity differences from high ($f_h = 850kHz$) and low ($f_l = 55kHz$) frequency measurements for different concrete specimens heights and diameters (D=0.60, 1.2 and 2.4 λ_0 for $f_0 = 55kHz, \lambda_0 = 83.01mm$).

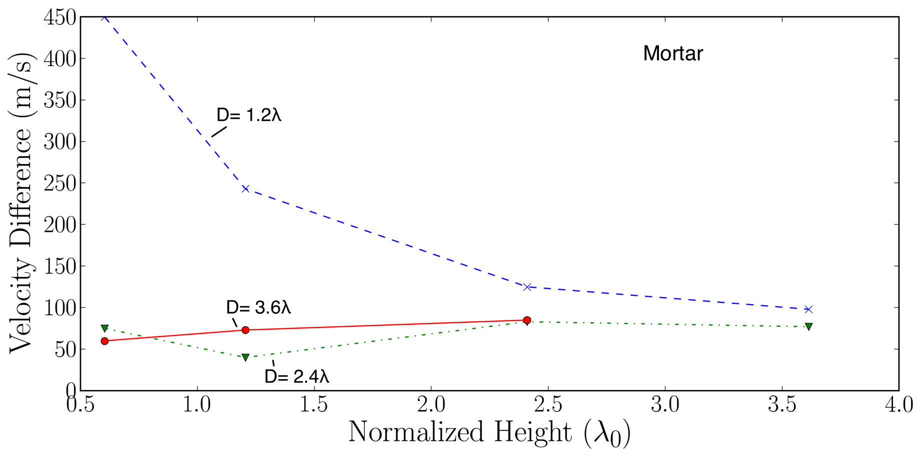


Figure 5.40: Wave velocity differences from high ($f_h = 850kHz$) and low ($f_l = 55kHz$) frequency measurements for different mortar specimens heights and diameters (D=0.60, 1.2 and 2.4 λ_0 for $f_0 = 55kHz, \lambda_0 = 83.01mm$)
.

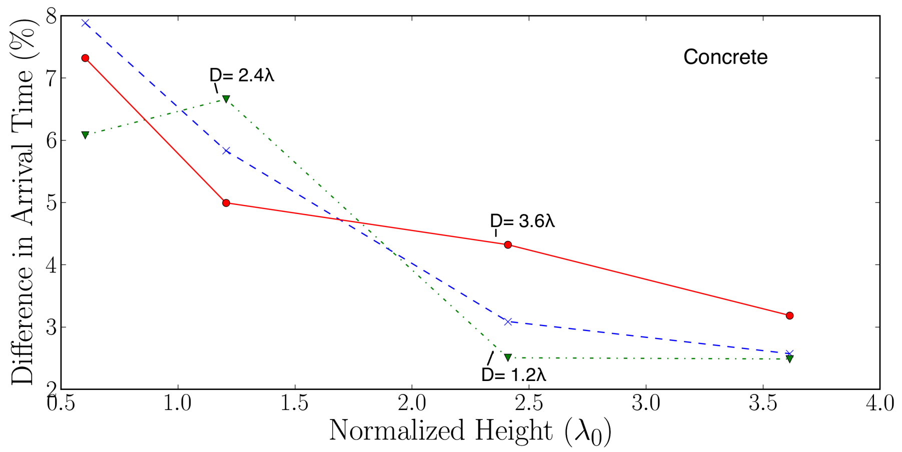


Figure 5.41: Arrival time differences from high ($f_h = 850kHz$) and low ($f_l = 55kHz$) frequency measurements for different concrete specimens heights and diameters (D=0.60, 1.2 and 2.4 λ_0 for $f_0 = 55kHz, \lambda_0 = 83.01mm$).

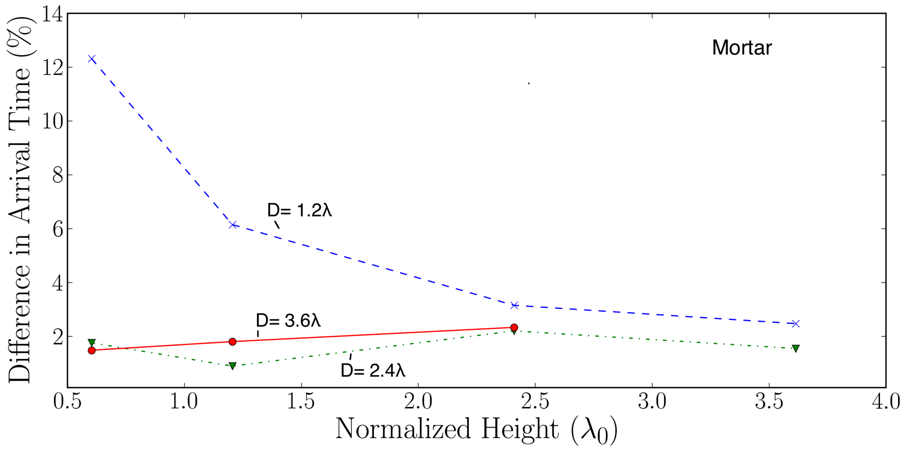


Figure 5.42: Arrival time differences from high ($f_h = 850kHz$) and low ($f_l = 55kHz$) frequency measurements for different mortar specimens heights and diameters (D=0.60, 1.2 and 2.4 λ_0 for $f_0 = 55kHz, \lambda_0 = 83.01mm$).

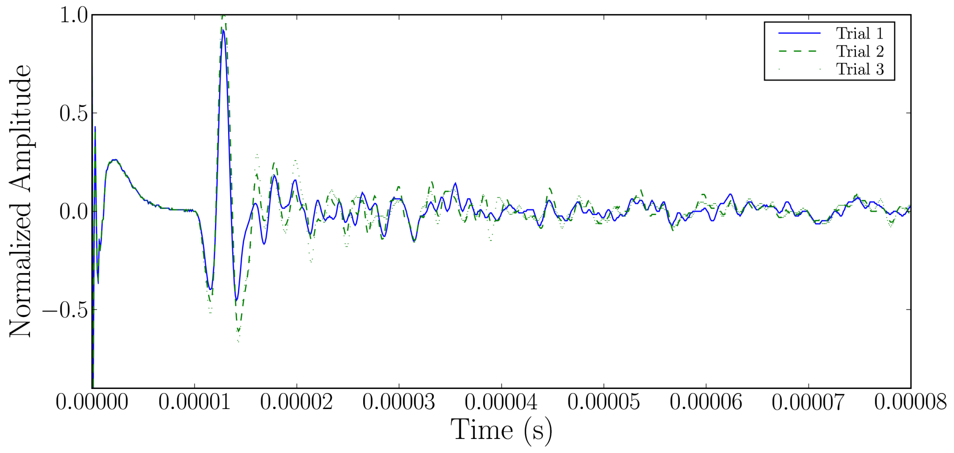


Figure 5.43: Comparison of experimental waveforms for specimen “c1a” (H=5 cm, D=10 cm) between three independent readings using high frequency excitation ($f_0 = 850kHz$).

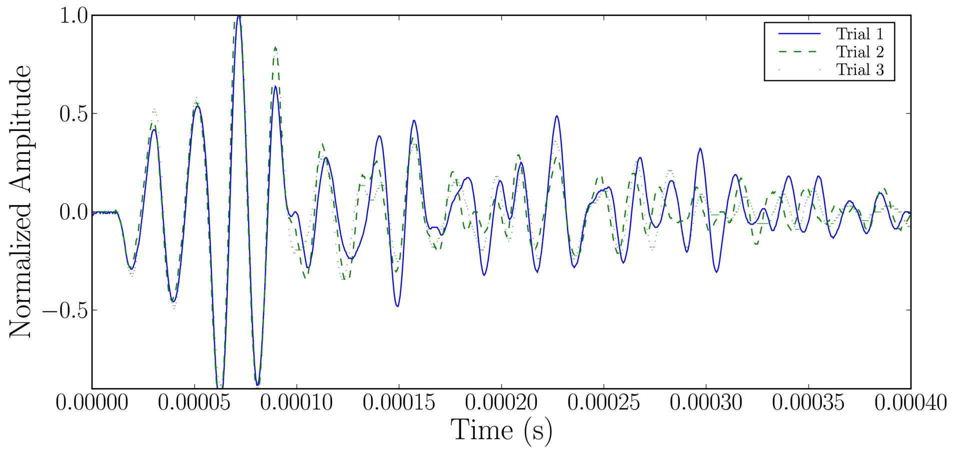


Figure 5.44: Comparison of experimental waveforms for specimen “m1a” (H=5 cm, D=10 cm) between three independent readings using low frequency excitation ($f_0 = 55kHz$).

5.3 Discussion

Results are discussed in three sections. First, numerical calibration results are evaluated to ensure that the numerical models are appropriate to simulated wave propagation. Second, findings from the numerical simulations are explained. Finally, the numerical and experimental results are compared.

5.3.1 Calibration of Numerical Models

First, the comparison between the Lamb's theoretical solution and LS-Dyna results are covered. Second, the solutions of for different layered models from Punch (Kausel (1981)) and LS-Dyna are compared. The first comparison is used to simulate wave propagation at the surface on an infinite half space whereas Punch is used as an independent confirmation for simulations on more complex layered models.

Lamb Solution

The theoretical solution to Lamb's problem assumes a point very far away from the source (section 3.4.2). This is referred to as the far field assumption. Due to size limitations in finite element simulations of wave propagation, it is prohibitive to obtain results that are far enough from the source to suit this condition. The far field is often defined, as a rule of thumb, as approximately twice the largest wavelength of the source signal using the surface wave velocity. Such a condition is not easily applied to this model as the original source has a significant zero frequency component, which has infinite wavelength. Instead, model dimensions are chosen such that most of the energy (over 90% as defined by area of power spectrum) is in the far field near the end of the model. In Lamb's theoretical solution only the surface wave component is taken into account. Hence, all P and S waves are not accounted for in the theoretical trace.

Using known material properties, it is calculated that velocities for P, S and surface waves are 4566, 2300, and 2144 m/s respectively. The main pulse has a width of approximately

0.02 ms (1.4 $(1/f_0 for f_0 = 75kHz)$); thus, clear separation of the P and surface wave pulse occurs near 91 mm ($2.97\lambda_r$ for $lambda_R = 30.6mm$) distance from the source, or closer if the tail of the surface pulse is ignored. This is observed in figure 5.2 where both pulses show a clear separation. Differences between S and surface wave velocities are small, thus, they take much longer to separate. In the model dimensions used, clear separation between S and surface wave does not occur.

Given the above considerations the discrepancies in each of figures 5.1 to 5.5 can be explained. The small pulse arriving before the main surface wave is the P wave component of the simulation. This is verified by comparing the expected P wave location with the observed pulse arrival for all figures. Second, the matching improves significantly as distance increases. Therefore, as the far field is approached the match becomes increasingly better. Finally, due to the dimensions used, it is difficult to clearly identify effects of the surface waves. However, at longer distances, the shear wave can be seen separating from the inverted pulse (figure 5.5). Finally, the presented results confirm the suggested rules for numerical simulations and good results can thus be expected for reflection free data, at a boundary.

Layered Media

Comparisons between Punch and LS-Dyna results are used to assess the ability of LS-Dyna to handle reflections, and refractions of wave propagation. Such aspects are not readily verified by previous comparisons.

Unlike the Lamb's model, data from Punch accurately models the near field and the far field. Differences exist, however, in the way both programs compute the wave field. Punch uses an iterative approach to solve the problem in frequency domain(Kausel, 1981). This leads to loss of accuracy in the higher frequencies unless the number of thin layers is increased. Conversely, LS-Dyna solves by stepping through time, which implies that inaccuracies are propagated, and compounded, in the time domain. These errors are minimized by using elements of a small enough size for the frequencies investigated and an adequately small time step.

The settings used in Punch satisfies the condition that there should be at least 4 sublayers per smallest wavelength. As such, no significant inaccuracies are expected. Similarly, LS-Dyna models have element size small enough such that there is at least 10 elements per smallest wavelength. Therefore, both models are expected to be accurately represented.

Results for both the single layers (figure 5.7 to 5.10) and the three layer model (figure 5.12 to 5.15) show good agreement. Both cases show best results for distances close to the source. Differences become more pronounced at the far distances. The only difference between the models used in both simulation is that the Punch model uses slight material damping to achieve a stable solution. Material damping is set as a percentage in Punch (0.1% in the used models). Effects of damping are frequency dependent and therefore unpredictable on the signal. Damping, however, causes changes that accentuates as distance increases from the source, which is in line with the observed discrepancies. As the signal is normalized, those changes are not easily visible in amplitude but would only show themselves in phase and pulse shape.

In conclusion, the calibration models using Punch confirm the results from LS-Dyna models.

5.3.2 Numerical Models

Numerical results are shown at the central axis of each model, where horizontal displacements are zero due to the axisymmetric condition. This axis is most important since first arrival times are measured at the centre of the specimens in practice. Results presented include sets for high frequency transducers, low frequency transducer, and low frequency transducer on damped model.

High Frequency Simulations

High frequency simulations are of most interest in understanding the arrivals of different wavefronts in the models. These wavefronts are caused entirely by the specimen's geometry and are thus applicable to all frequencies. Higher frequency sources with broadband pulses have very narrow pulse widths ensuring good separation of individual wavefronts. Frequency-wavenumber plots presented in figure 5.17 and 5.26 show the presence of forward and reflected P and S waves. No surface waves are present since this data is not at a boundary. The F-K plots do not show numerical dispersion for the wavefronts mentioned. This information is also presented in figure 5.18 and 5.27 in the form of seismic traces.

Specimen "m1a" (H=0.60 λ_0, D=1.20 λ_0 for $\lambda_0 = 5.412mm$) is used to understand the propagation of important wavefronts due to its small size. This ensures that relevant boundary effects are properly accounted for in the time-scales of the simulation. Figures 5.17 through 5.25 show the time history plots for each boundary of the specimen and are thus used to study the various wavefronts that may affect perceived first arrival of signals. Figure 5.18 shows the time history of the central axis, most relevant to first arrivals. Due to the narrow pulse in the signal, the p and s wavefronts are easily observed and clearly separated. However, both those waves are wide in time as they are created by a wide transducer. The effect of transducer shape on signal is complex and depends on transducer shape, input signal and the medium in which the wave is propagated. For the case of the high frequency input signal used in simulation, the energy concentration of the principal p-wave pulse as it forms in time is shown in figure 5.37. It is observed that the energy is strongly concentrated near the central axis of the medium with a slight front being created from each of the transducer's corners. The corner fronts will create extra p and s wavefronts, of lower amplitude, closer to the side boundaries.

From figures 5.18 to 5.25 it seen that the main wavefronts (higher amplitude wavefronts) arriving at the bottom boundary are the principal p-wave, shear wave front from centre of source, shear wave front from corners of source, P wave reflection from the side and mode-converted p front created by the surface wave impacting the sides. Other fronts also arrive at the bottom boundary before or after the mentioned fronts but their amplitudes are relatively low, or they are reflections from the top boundary which only occur in the special case of the small specimen height. The order of arrival of these fronts varies based on specimen size as can be seen by comparing to figure 5.27. In that case, the P wave fronts, mode converted or otherwise, arrive before the shear wave fronts as the long length gives them more time to overtake shear wave modes. Of all these wavefronts, the most significant in terms of amplitude length at the bottom receiver is the mode-converted P wave created by the surface wave impact on the side boundary. Simulations for larger widths confirm those results and are presented in section E.2.1 for completeness. It is clear from these results that first arrivals are well separated for all of the widths tested and would thus indicate accurate velocity values for UPVM.

Low Frequency Simulations

The use of a low frequency transducer ($f_0 = 55kHz$) for the signal coupled with the small specimen size (0.5 to 3 λ at $f_0 = 55kHz$ and $V_p = 4566m/s$) limits the resolution of the wavenumber axis in the low frequency F-K plots. Resolution depends on the number of traces and the distance between receivers. This resolution is given as:

$$\Delta k = \frac{1}{2\Delta x}\frac{2}{N} \tag{5.1}$$

where Δk is the wavenumber resolution, Δx is the distance between each point and N is the total number of points. In the smallest model the resolution is $dk = 1.69\frac{1}{\lambda_{55kHz}}$, λ_{55kHz} is the p-wave wavelength at 55 kHz, while $dk = 0.28\frac{1}{\lambda_{55kHz}}$ for the longest specimen. Thus, only 1 points per 1.5 wavelength are available for the small specimen as opposed to 3.5 per wavelength for the longest specimen at peak frequency. In addition, the high frequency source ($f_0 = 850kHz$), has 9.25 and 55.5 points per main wavelength. Longer specimens do provide improved clarity but are problematic for obtaining clear conclusions about numerical dispersion and propagating waves. As such, F-K plots are of limited utility in the study of wave propagation in the central axis of specimen less than 4 wavelength in height. F-K plots for low frequency results are presented in Appendix E.2.2 for undamped models and E.2.3 for damped models.

Time history traces for low frequency simulations show a definitive first arrival but independent wavefronts are difficult to observe. The large pulse of the 55 kHz signal makes it difficult to differentiate between P or S waves and their reflections.

Seismic plots for the damped models (figure 5.30 to 5.32) show similar results to those of the undamped model in that only first arrivals information is easily obtained. The major difference between both is the continuously slowing velocity in the model as distance from source increases. This is visible on time traces for all heights except for the longest model where signal falls below the resolving capability of LS-Dyna in double precision. Seismic traces for all seismic plots use individually normalized time trace for each height for added visual detail.

Figures 5.33 through 5.35 compare the time signal from the damped and undamped model at the end surface for the thinnest specimens. Differences between both traces show a distinct perceived lag in first arrival for the damped signal. This lag is accentuated the farther the signal travels from the source. This implies damping can leads to larger perceived arrival times. It is expected that this lag is limited by the pulse width and is thus of increasing significance for lower frequencies and/or long pulses regardless of specimen size.

For example, if a specimen x wavelengths in height and y wavelengths in width is tested using a broadband source of radius R wavelengths having a wavelet n oscillations wide at main frequency, and the time taken for the p-wave to reach the bottom boundary is defined as the normalized time unit $t_0 = xT$ (figure 5.45). Then, the main P and S waves arrive at time t_0 and $r_s t_0$ respectively where $r_s = \frac{V_p}{V_s}$ is the ratio of P and S wave velocities. Furthermore, strong p-wave reflections from the boundary are expected to arrive at t_s and t_r where (figure 5.46):

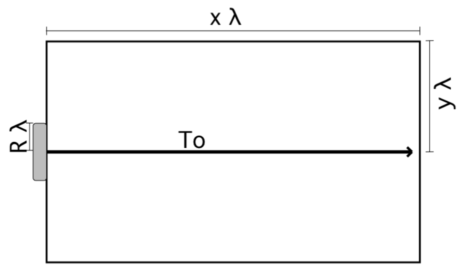


Figure 5.45: Normalized measurement of theoretical specimen

$$
\begin{aligned}
t_{shear} &= \left(r_s(y - R) + \sqrt{y^2 + x^2} \right) \frac{t_0}{x} && (5.2) \\
t_{surface} &= \left(r_r(y - R) + \sqrt{y^2 + x^2} \right) \frac{t_0}{x} && (5.3) \\
r_r &= \frac{V_p}{V_r} && (5.4)
\end{aligned}
$$

where t_{shear} is the mode converted p-wave reflection caused by the shear wave impacting the side, $t_{surface}$ is the mode converted p-wave reflection caused by the surface wave impacting the side and r_r is the ratio of shear to surface wave velocity. The time $t_{surface}$ is also representative of the arrival time of the surface wave created by the impacting p-wave at the bottom corners of the specimen. The times $t_{surface}$ and t_{shear} are derived using s-wave propagation at the top surface and p-wave propagation towards the receiver after the collision, or conversely p-wave propagation towards the corner followed by surface wave propagation towards the receiver. These values indicate first arrival of the wavefronts as created (received) by the corners of the transducer (receiver), not their highest amplitude area which are near the centre of the source (receiver). Reflection of p-waves at the side (from corner or otherwise) are also an important arrival but, under all circumstances for a uniform isotropic specimen, will be of smaller amplitude than the principal p-wave traveling at the centre. Consequently, if the main p-wave is attenuated enough to affect UPVM readings, so is its reflection from the side. Thus, the p-wave reflection from the side is assumed to have negligible effects on UPVM. These first arrivals calculations are valid regardless of frequency.

From the above it can be deduced that a minimum height, x, for initial p and s wave separation is given by:

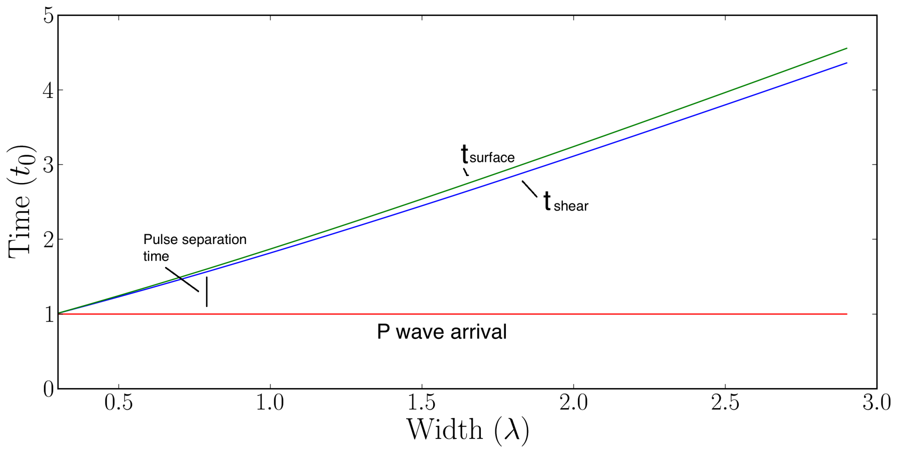


Figure 5.46: Arrival time of high amplitude fronts in terms of P wave arrival for $r_s = 2, r_r = 2.15, R = 0.3$. The value of R implies a transducer smaller than its produced wavelength, typical of lower frequency (55 kHz) readings in concrete.

$$x \geq (r_s - 1)n \tag{5.5}$$

and the minimum width for separation of p-wave from both main boundary reflections is given by (figure 5.47):

$$y \geq \frac{r_s(n + x + r_s R) - \sqrt{r_s^2(n + x + r_s R)^2 - (r_s^2 - 1)\left[(n + x)^2 + r_s R(r_s R + 2(n + x)) - x^2)\right]}}{r_s^2 - 1} \tag{5.6}$$

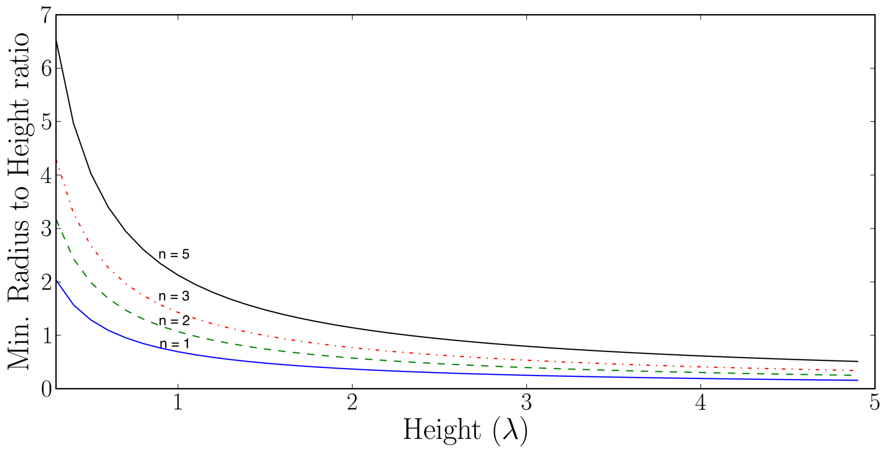


Figure 5.47: Minimum Radius to Height ratio to ensure p-wave pulse separation in a specimen for $r_s = 2, r_r = 2.15, R = 0.3$. The value of R implies a transducer smaller than its produced wavelength, typical of lower frequency (55 kHz) readings in concrete.

where the above condition guarantees $t_s \geq t_0 + nT$. These conditions ensure that the p-wave pulse itself is separated but the rest of the wave fronts may not. Separation of wavefront is useful for UPVM as an attenuated first arrival is clearly identifiable, which is not the case otherwise. For example, the first p-wave may be damped out but the constructive wave interactions from other wavefronts may still give the impression of proper first arrival. This is especially valid since, as is shown in figure 5.18, the highest amplitude waves originate from side boundary effects. Furthermore, it is shown in figure 5.38 that the first p-wave emitted by sources smaller than their main wavelength are not focused, hence producing a weaker p-wave at the receiving end. Conversely, the boundary reflections do not require focusing to achieve their high amplitudes, implying that the initial p-wave may be easily overshadowed by these stronger pulses if not separated. This result is in contrast to the high frequency source shown in figure 5.37.

Low Frequency Simulations – First arrivals

As discussed in the previous section, the delay between damped and undamped signals may occur from the observation of a first arrival of a different waveform than the first p-wave. This effect can be avoided with proper pulse separation and a full waveform inspection. In the numerical simulations of low-frequency transmitter ($f_0 = 55kHz$), the conditions for pulse separation are not met for any model. This is directly related to the very wide pulse used. Hence a “delay lag” is observed between damped and undamped results at the receiver location.

The delay lag is caused by attenuation of the rising edge of a pulse below a given noise floor, leaving higher amplitude sections of the pulse to determine pulse arrival (figure 5.48). A delay lag can be observed even with proper pulse separation. However, when the initial p-pulse is separated, the effect is limited to less than a quarter of the pulse's period (a quarter period is the distance from rise to first peak of a full sine) as measured using the main pulse's frequency at the receiver end. This value is given with the assumption that the pulse shape is known by the operator. Any delay larger than a quarter of a period would be identifiable by the operator in the form of a missing peak in the waveform. These considerations are not applicable to commercial equipment that do not provide a full waveform.

Without pulse separation, delay lag is unbounded. That is, the observed delay cannot be limited by the operator by any means other than reducing the loss of amplitude of the signal. Philippidis and Aggelis (2005) results on pulse velocity of concrete showed an increase in velocity with frequency. This is in line with previous numerical observations. Higher frequencies have reduced periods and smaller wavelengths. Hence, provided the source is excited by a broadband pulse, the conditions for pulse separation are improved. Furthermore, once the pulse is properly separated, experimental uncertainty is reduced due to a reduction of the period. It is not possible to tell wether the effects observed by Philippidis and Aggelis (2005) are caused by potential experimental error or actual physical phenomenon as no pulse width information is given.

Such lag effects are classically avoided by measuring first arrival using the first peak of the received signal. This method is accurate at higher frequencies where the pulse width is well known. However, such an approach would give unpredictable results on non-separated pulses, which have peaks change based on interference (figure 5.33, 5.34). Thus, accurate UPVM using peak arrival methods are only viable when criteria for pulse separation are met.

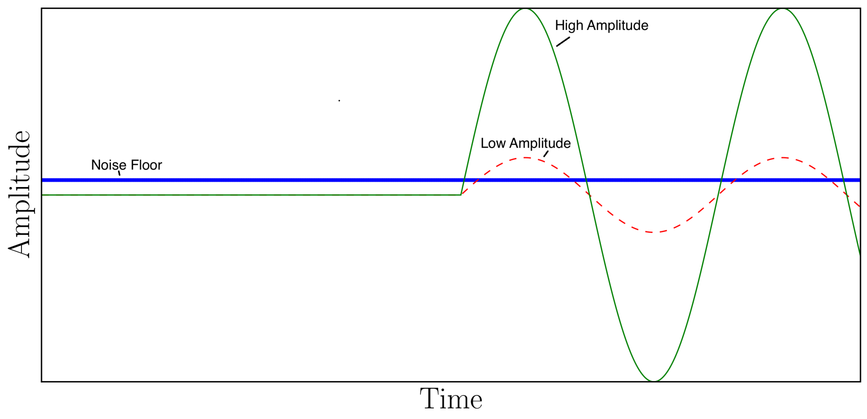


Figure 5.48: Delay lag effect demonstration

5.3.3 Experimental Results

The experiments performed are designed to validate the findings of numerical simulations: higher frequency readings have higher perceived velocity than low frequency readings. A low-frequency transducer ($f_0 = 55kHz$) with a wide pulse is used to show the case when the specimen does not meet the criteria for pulse separation. Proper pulse separation exists for high frequency readings ($f_0 = 850kHz$).

Figure 5.41 and 5.42 show that first arrivals for all specimens have a noticeable difference between high and low frequency readings. It is also observed that width has no noticeable effect on the readings other than experimental differences in the specimen. This is expected as the low frequency source does not have proper pulse separation for any of the width used.

Concrete specimens show the most consistent behavior in width in terms of velocity difference vs height (figure 5.39). Differences in velocities decrease as height increase for all three widths. Conversely, differences in first arrival increases steadily with height (figure 5.41). This is attributed to an absolute cause of error (e.g. surface contact) in arrival time that decreases as a percentage when calculating velocity over a larger distance.

Mortar specimens show consistent behavior for 200 and 300 mm wide samples but have a clear difference in the 100 mm wide specimen set. All mortar samples were cast similarly. However as their standard diameter of 100 mm permitted, these samples had their surfaces smoothed by a concrete surface grinder in order to improve signal coupling but it instead exposed a significant amount of small voids. These voids lead to very poor coupling compared to other larger width specimen. Figure 5.40 show for the 200 and 300 mm wide samples that velocity remains flat with height. The 100 mm sample shows behavior similar to that of the concrete specimen. Consistent with previous observations in concrete, all mortar specimens have an increasing first arrival time difference with height.

The increasing first arrival difference with height confirm the perceived lag effect does increase with signal loss as seen in numerical simulations. Decreasing velocity difference with height (concrete and thin mortar specimens) suggest a constant signal loss factor.

This is confirmed by figure 5.42 where 100 mm wide samples have a significantly higher perceived first arrival difference compared to other specimen with better surface coupling. The principal factor that is constant between transducers, yet differing between specimen, is the coupling with the specimen surface.

Loss in signal amplitude, either from surface coupling or from material attenuation, leads to a significant perceived late first arrival when using lower frequency ultrasound without pulse separation. Effects from the surface can be diminished by using long specimens to effectively dilute the constant error over length (reduced percentage error). Effects of signal loss caused by material attenuation (attenuation due to the material itself, not external conditions) can be limited to a quarter period of the main pulse frequency at the receiving location if conditions for pulse separation are met and the waveform is properly captured as part of the experiment.

5.3.4 Recommendations

Use of higher frequencies (above 100 Khz) is rare in the inspection of geomaterials such as concrete due to the high attenuation. The use of low frequency (below 100 kHz) transducers for velocity is thus inevitable. Conclusions from numerical and experimental results indicate that large effects on velocity measurements are preventable if specimen size, and excitation pulse are controllable and the waveform is accessible by the operator. Using the pulse separation criteria, it is possible to determine adequate specimen size to avoid these effects. If size, or source, are not controllable the difference is expected to come mainly from two attenuation sources: surface coupling and specimen length.

Experimental results show that the most significant effect (in terms of velocity difference) is due to surface coupling. These effects are most significant when the specimen is short as the difference in arrival time is a large percentage of total arrival time. As the specimen get longer, the effect becomes a lower percentage of total arrival time and the effect on velocity is much less. As such, if no physical alteration of the specimen is possible, use of longer specimen is an easy way to minimize these effects. Specimen 4 wavelength in height was sufficient to diminish surface effects to a minimum. Any increase past 20 cm has negligible effects on measured velocity.

Effects due to specimen length are best seen from 5.42 and 5.40. It is observed from figure 5.42 that delay in the smallest specimens (excluding surface ground specimen) is 0.2 ms. This delay is negligibly small given experimental uncertainty and is linked to surface effect since the specimen is short (0.5 λ_{55kHz}). These specimens show a near-flat velocity difference with length that is below 100 m/s. Thus, it can be said that lag effects due to material attenuation is negligible for specimens less than 4 wavelength in height in the absence of pulse separation.

Further studies are suggested to improve on the main conclusions of this work. Rigorous studies of surface coupling in terms of absolute signal attenuation on different surfaces is suggested. Furthermore, a more in depth study of material damping is recommended to fully understand the effects of varying damping values on both high and low frequencies. Further work is also of interest in the more subtle effects of transducer size and location during UPVM for both receivers and transmitters (e.g. off-centre, misaligned).

Chapter 6

Conclusions

Description of current and emerging nondestructive methods for civil engineering structures are presented. Distinctions are made between electromagnetic and acoustic methods with a focus on discussing the more versatile acoustic techniques. Current research in ultrasonic NDT is presented with particular attention to pulse velocity measurements (UPVM). It is mentioned that UPVM is fast and requires minimal skill from operators and has been used for flaw detection (Sutan and Jaafar, 2003), study of material contents (Abo-Qudais, 2005; Ohdaira and Masuzawa, 2000; Popovics, 2005), deducing general deterioration (Ohdaira and Masuzawa, 2000), determination of elastic properties (Washer et al., 2005; Prassianakis and Prassianakis, 2004), measuring strength (Chang et al., 2006), and others. In these applications, accurate measurements of velocity are essential for proper evaluation. It is found by Philippidis and Aggelis (2005) that UPVM are susceptible to specimen size, attenuation and frequency but no clear conclusions are made on the fundamental reason for the differences.

In this work, numerical simulations are presented to study the size effect of different specimen geometries on first arrival of wave signal. All numerical simulations are performed using the LS-Dyna 970 double precision solver. Numerical models are first calibrated to ensure accurate results using theoretical models and then using a known numerical code

"Punch". It is found that the results from theoretical models and "Punch" confirm the validity of LS-Dyna models. Calibration include the choice of element size, time step and frequencies.

Simulations are completed for 12 specimen of different dimensions having heights of 5,10,20 and 30 cm as well as diameters of 10, 20 and 30 cm. Both a low ($f_0 = 55kHz$) and high ($f_0 = 850MHz$) frequency input source are used on each specimen. Numerical simulations using low frequencies are made for both a damped and undamped series of specimen.

Arrival of different wave fronts are discussed using the high frequency numerical simulations. Low frequency simulations using undamped models are found to be of limited use in studying wave arrivals due to complex wave front interference. Results from low frequency simulations of damped models indicate that wave attenuation can lead to significant errors in first arrivals when complex wave interference is present. Conditions for wave interference at the receiver location are studied and minimum size conditions for both height and width are derived. These conditions guarantee proper pulse separation for UPVM and are dependent on source size, and source pulse width. It is argued that with proper use these conditions will lead to accuracy of measurement better than one quarter of a period of the main excitation frequency when using a full waveform and a skilled operator.

Finally, experiments are performed to assess differences in first arrivals between high and low frequency measurements. Readings are made on 11 mortar and 12 concrete specimen of different heights and widths. Experimentally significant time differences are observed between high and low frequency readings. It is found that differences in first arrivals will increase with specimen length but differences in velocity will decrease with length. Specimens 4 wavelengths in height are deemed sufficient to diminish surface effects to a minimum. Any increase past 4 wavelengths has negligible effects on measured velocity.

Effects due to specimen length are best seen from 5.42 and 5.40. It is observed from figure 5.42 that delay in the smallest specimens (excluding surface ground specimen) is 0.2

ms. This delay is negligibly small given experimental uncertainty and is linked to surface effect since the specimen is short (0.5 λ_{55kHz}). These specimens show a near-flat velocity difference with length that is below 100 m/s. Thus, it can be said that lag effects due to material attenuation is negligible for specimens less than 4 wavelength in height in the absence of pulse separation.

Further studies are suggested to improve on the main conclusions of this work. Rigorous studies of surface coupling in terms of absolute signal attenuation on different surfaces is suggested. Furthermore, a more in depth study of material damping is recommended to fully understand the effects of varying damping values on both high and low frequencies. Further work is also of interest in the more subtle effects of transducer size and location during UPVM for both receivers and transmitters (e.g. off-centre, misaligned).

Appendix A

LS-Dyna Input Decks

A.1 Lamb's Problem Deck

```
*KEYWORD
*TITLE
Autmatically generated by cylgenDyna.py
*CONTROL_TERMINATION
$   ENDTIM    ENDCYC     DTMIN    ENDENG    ENDMAS
$#  endtim    endcyc     dtmin    endeng    endmas
  0.100000         0     0.000     0.000     0.000
*CONTROL_TIMESTEP
$   DTINIT    TSSFAC      ISDO    TSLIMT     DT2MS      LCTM     ERODE     MS1ST
$#  dtinit    tssfac      isdo    tslimt     dt2ms      lctm     erode   ms1st
     0.000  0.900000         0     0.000     0.000         0         0         0
$#  dt2msf   dt2mslc
     0.000         0
*DATABASE_GLSTAT
$#      dt    binary
  0.160000         1
*DATABASE_NODOUT
$       DT    BINARY
$#      dt    binary
  0.080000         1
*DATABASE_BINARY_D3PLOT
$#      dt      lcdt      beam     npltc
  0.160000         0         0         0
$#   ioopt
         0
*LOAD_NODE_POINT
$     NODE       DOF      LCID        SF       CID        M1        M2        M3
$#     nid       dof      lcid        sf       cid        m1        m2        m3
         1         2         1  1.000000         0         0         0         0
*PART
$# title
Part 1
$#     pid     secid       mid     eosid      hgid      grav    adpopt      tmid
         1         1         1         0         0         0         0         0
*SECTION_SHELL_TITLE
P-1
$    SECID    ELFORM      SHRF       NIP     PROPT   QR/IRID     ICOMP     SETYP
$#   secid    elform      shrf       nip     propt   qr/irid     icomp     setyp
         1        15  0.830000         4         1         0         0         1
$#      t1        t2        t3        t4      nloc     marea
     0.000     0.000     0.000     0.000         1     0.000
*MAT_ELASTIC_TITLE
Material Layer
```

```
$      MID        RO         E        PR        DA        DB
$#     mid        ro         e        pr        da        db  not used
         1  0.002100 29578.000  0.330000     0.000     0.000         0
```

A.2 Punch Single Layer Model

```
*KEYWORD
*TITLE
Autmatically generated by layergenDyna.py
*CONTROL_TERMINATION
$   ENDTIM    ENDCYC     DTMIN    ENDENG    ENDMAS
$#  endtim    endcyc     dtmin    endeng    endmas
  2.500000         0     0.000     0.000     0.000
*CONTROL_TIMESTEP
$   DTINIT    TSSFAC      ISDO    TSLIMT     DT2MS      LCTM     ERODE     MS1ST
$#  dtinit    tssfac      isdo    tslimt     dt2ms      lctm     erode     ms1st
     0.000  0.900000         0     0.000     0.000         0         0         0
$#  dt2msf   dt2mslc
     0.000         0
*DATABASE_GLSTAT
$#      dt    binary
 7.5000E-4         1
*DATABASE_NODOUT
$       DT    BINARY
$#      dt    binary
  0.002000         1
*DATABASE_BINARY_D3PLOT
$#      dt      lcdt      beam     npltc
 10.000000         0         0         0
$#   ioopt
         0
*LOAD_NODE_POINT
$     NODE       DOF      LCID        SF       CID        M1        M2        M3
$#     nid       dof      lcid        sf       cid        m1        m2        m3
         1         2         1  1.000000         0         0         0         0
*PART
$# title
Layer 1
$#     pid     secid       mid     eosid      hgid      grav    adpopt      tmid
         1         1         1         0         0         0         0         0
*SECTION_SHELL_TITLE
P-1
$    SECID    ELFORM      SHRF       NIP     PROPT   QR/IRID     ICOMP     SETYP
$#   secid    elform      shrf       nip     propt   qr/irid     icomp     setyp
         1        15  0.830000         4         1         0         0         1
$#      t1        t2        t3        t4      nloc     marea
     0.000     0.000     0.000     0.000         1     0.000
*MAT_ELASTIC_TITLE
Material Layer
$      MID        RO         E        PR        DA        DB
$#     mid        ro         e        pr        da        db  not used
         1 1800.0000 1.1997E+9  0.330000     0.000     0.000         0
```

A.3 Punch Three Layer Model

```
*KEYWORD
*TITLE
Autmatically generated by layergenDyna.py
*CONTROL_TERMINATION
$   ENDTIM    ENDCYC     DTMIN    ENDENG    ENDMAS
$#  endtim    endcyc     dtmin    endeng    endmas
  3.000000         0     0.000     0.000     0.000
*CONTROL_TIMESTEP
$   DTINIT    TSSFAC      ISDO    TSLIMT     DT2MS      LCTM     ERODE     MS1ST
$#  dtinit    tssfac      isdo    tslimt     dt2ms      lctm     erode     ms1st
     0.000  0.900000         0     0.000     0.000         0         0         0
$#  dt2msf   dt2mslc
     0.000         0
*DATABASE_GLSTAT
$#      dt    binary
  0.002000         1
*DATABASE_NODOUT
$       DT    BINARY
$#      dt    binary
  0.005000         1
*DATABASE_BINARY_D3PLOT
$#      dt      lcdt      beam     npltc
 25.000000         0         0         0
$#   ioopt
         0
$#     id1
*LOAD_NODE_POINT
$     NODE       DOF      LCID        SF       CID        M1        M2        M3
$#     nid       dof      lcid        sf       cid        m1        m2        m3
         1         2         1  1.000000         0         0         0         0
*PART
$# title
Layer 1
$#     pid     secid       mid     eosid      hgid      grav    adpopt      tmid
         1         1         1         0         0         0         0         0
*SECTION_SHELL_TITLE
P-1
$    SECID    ELFORM      SHRF       NIP     PROPT   QR/IRID     ICOMP     SETYP
$#   secid    elform      shrf       nip     propt   qr/irid     icomp     setyp
         1        15  0.830000         4         1         0         0         1
$#      t1        t2        t3        t4      nloc     marea
     0.000     0.000     0.000     0.000         1     0.000
*MAT_ELASTIC_TITLE
Material Layer
$      MID        RO         E        PR        DA        DB
$#     mid        ro         e        pr        da        db  not used
         1 1800.0000 1.9195E+8  0.330000     0.000     0.000         0
*PART
$# title
Layer 2
$#     pid     secid       mid     eosid      hgid      grav    adpopt      tmid
         2         1         2         0         0         0         0         0
*MAT_ELASTIC_TITLE
Material Layer
$      MID        RO         E        PR        DA        DB
$#     mid        ro         e        pr        da        db  not used
         2 1800.0000 7.6781E+8  0.330000     0.000     0.000         0
*PART
$# title
Layer 3
$#     pid     secid       mid     eosid      hgid      grav    adpopt      tmid
         3         1         3         0         0         0         0         0
*MAT_ELASTIC_TITLE
Material Layer
$      MID        RO         E        PR        DA        DB
$#     mid        ro         e        pr        da        db  not used
```

```
3 1800.0000 1.1997E+9  0.330000     0.000     0.000         0
```

A.4 High Frequency Experimental Model - Sample

```
*KEYWORD
*TITLE
 Autmatically generated by cylgenDyna.py
*PART
$# title
                                                                      Part 1
$#     pid     secid       mid     eosid      hgid      grav    adpopt      tmid
         1         1         1         0         0         0         0         0
*MAT_ELASTIC_TITLE
Material Layer
$      MID        RO         E        PR        DA        DB
$#     mid        ro         e        pr        da        db  not used
         1   0.00210     29578      0.33         0      0.00         0
*SECTION_SHELL_TITLE
P-1
$    SECID    ELFORM      SHRF       NIP     PROPT   QR/IRID     ICOMP     SETYP
$#   secid    elform      shrf       nip     propt   qr/irid     icomp     setyp
         1        15  0.830000         4         1         0         0         1
$#      t1        t2        t3        t4      nloc     marea
  0.000000  0.000000  0.000000  0.000000         1  0.000000

*DATABASE_NODOUT
$       DT    BINARY
$#      dt    binary
  0.000012         1

*DATABASE_GLSTAT
$#      dt    binary
  0.000040         1
*DATABASE_BINARY_D3PLOT
$#      dt      lcdt      beam     npltc
  0.100000         0         0         0
$#   ioopt
         0
*CONTROL_TERMINATION
$   ENDTIM    ENDCYC     DTMIN    ENDENG    ENDMAS
$#  endtim    endcyc     dtmin    endeng    endmas
  0.050000         0  0.000000  0.000000  0.000000
*CONTROL_TIMESTEP
$   DTINIT   TSSFAC       ISDO    TSLIMT     DT2MS      LCTM     ERODE     MS1ST
$#  dtinit    tssfac      isdo    tslimt     dt2ms      lctm     erode     ms1st
  0.000000  0.900000         0  0.000000  0.000000         0         0
$#  dt2msf   dt2mslc
  0.000000         0
```

A.5 Low Frequency Experimental Model - Sample

```
*KEYWORD
*TITLE
 Autmatically generated by cylgenDyna.py
*PART
$# title
                                                                             Part 1
$#     pid     secid       mid     eosid      hgid      grav    adpopt      tmid
         1         1         1         0         0         0         0         0
  *MAT_ELASTIC_TITLE
  Material Layer
  $      MID        RO         E        PR        DA        DB
  $#     mid        ro         e        pr        da        db  not used
           1   0.00210     29578      0.33         0      0.00         0
  *SECTION_SHELL_TITLE
  P-1
  $    SECID    ELFORM      SHRF       NIP     PROPT   QR/IRID     ICOMP     SETYP
  $#   secid    elform      shrf       nip     propt   qr/irid     icomp     setyp
           1        15  0.830000         4         1         0         0         1
  $#      t1        t2        t3        t4      nloc     marea
    0.000000  0.000000  0.000000  0.000000         1  0.000000
*CONTROL_TERMINATION
    $   ENDTIM    ENDCYC     DTMIN    ENDENG    ENDMAS
    $#  endtim    endcyc     dtmin    endeng    endmas
      0.250000         0  0.000000  0.000000  0.000000
*CONTROL_TIMESTEP
    $   DTINIT    TSSFAC      ISDO    TSLIMT     DT2MS      LCTM     ERODE     MS1ST
    $#  dtinit    tssfac      isdo    tslimt     dt2ms      lctm     erode     ms1st
      0.000000  0.900000         0  0.000000  0.000000         0         0
    $#  dt2msf   dt2mslc
      0.000000         0
```

Appendix B

Python Classes

B.1 LayergenDyna

```
#! /usr/bin/python
""" Scripts to generate an elastic cylinder of fully
integrated LS-DYNA 2D Axisymmetric elements in the
form of a homogeneous cylinder of specified dimensions.
The dimensions, element size, transducer size, input
trace are all entered at the command line.

  This is a somewhat specialized script and should only
be used by people that have knowledge of the internal
workings of this scripts. It has a lot of specialized
functions used in a specific research project.
4
Author: Simon Berube, 2007
Usage: (All sizes in mm)
  cylgenDyna.py nX nY dX dY td_size impulse_file dir Output_file rho E p_r t_t n_1? n_2?
  cylgenDyna.py 101 101 0.25 0.25 30 1M_Pulse.txt DIR m1a.k 0.00210 27440 0.33 0.05
python cylgendyna.py 280 526 0.25 0.25 20 sineIn_out.txt ./ alum.k 0.002713 73000 0.33 0.25

Load Curve: tau = 0.001, skip = 0.00002*800 t=arage(0,1000)*0.0005
cylgendyna 600 600 0.4167 0.4167 1 ./lsource.txt ~/code/Lamb/ lamb.k 0.00210 29578 0.33 0.1

"""
import pickle
from numpy import array,zeros, arange, sqrt #For array management
import sys #For system arguments
import upv #For time traces, and reading from oscilloscope

# Done and Working
def getInputInfo(pfn):
    #Information needed : Max Frequency, Layer Information
    fid = open(pfn) #open punch input file
    for i in range(4): #Skip lines until Frequency information
        fid.readline() #Skip line

    f_max = fid.readline().split()
    nbf = int(f_max[0])
    f_max = (nbf-1.0)*float(f_max[1]) + float(f_max[2]) #This is max frequency

    cnt = int(fid.readline().split()[0]) #Number of layers
    fid.readline() #Skip descriptor line

    layers = []
    for i in range(cnt):
        s = fid.readline()
        vtmp = map(float,s.split()[0:7])
        vtmp[0] = int(vtmp[0])

        #Conver this to useful data V_p V_s E v rho height
        #Get E = V_s^2 2\rho(1+\nu)
        E = vtmp[3]**2 * 2 * (vtmp[2]) *(1+vtmp[4]) #Density is in 1000 kG/m^3 -> Convert to g/mm^2
        V_p = sqrt( E*(1-vtmp[4])/( (1+vtmp[4])*(1-2*vtmp[4])*(vtmp[2])))
        layers.append([V_p,vtmp[3],E, vtmp[4], vtmp[2], vtmp[1]])

    fid.readline()
    fid.readline()

    #Get number of item information
    tmpnbR = map(float, fid.readline().split()[0:3]) #Get number of radii distances
    tmpnbR[0] = int(tmpnbR[0])

    nbR = tmpnbR[0];
    dR = tmpnbR[2] - tmpnbR[1];
    if (tmpnbR[1] == 0): #If the first distance is zero, ignore it
        nbR = nbR - 1;


    fid.readline(); #Skip line
    nbD = int(fid.readline().split()[0]) #Get the number of depths

    fid.close()
    return [f_max,layers,nbR,nbD,dR,nbf]

def layeredDyna(punch_fn, dx, load_fn, out_fn, t_t):
  #Obtain and Simplify Input arguments
  td_sz = 1
  dir = ''
  n_1 = -1
  n_2 = -1
  #Read information of Input file (PUNCH file)
  pif = getInputInfo(punch_fn)
  aL = array(pif[1]) #Array of Layers
  tH = sum(aL[:,5])
  dy = dx #We only care about square (Reflections, etc..)
  max_x = int(tH/dx)
  max_y = max_x
  lHinNodes = map(int,aL[:,5].cumsum()/sum(aL[:,5])*max_y)

  HB = [] #List of Horizontal Boundaries
  HB.append(arange(max_x)+1) #Top boundary
  for i in arange(len(pif[1])):
    HB.append(arange(max_x)+ (lHinNodes[i]-1)*max_y + 1)

  HB = array(HB)



  print "Dimensions will be %i by %i" % (max_x, max_y)

  #Define the mesh, boundaries vectors
  NODES = zeros((6,max_x*max_y))
  NODES[0,:] = arange(1, max_x*max_y+1)
  LB = zeros(max_y) #Left Boundary
  RB = zeros(max_y) #Right Boundary

  """ NODES Setup:
    NODES[0,:] - NODE ID
    NODES[1,:] - X COORD
    NODES[2,:] - Y COORD
    NODES[3,:] - Z COORD
    NODES[4,:] - tc flag
    NODES[5,:] - rc flag
    """
  #Set up Nodes Vectors
  dxv = arange(max_x)*dx

  for i in range(max_y):
    NODES[1,i*max_x:(i+1)*max_x] = dxv
    NODES[2,i*max_x:(i+1)*max_x] = i*dy
    LB[i] = i*max_x + 1
    RB[i] = (i+1)*max_x

  #Set up Shells
  """ SHELL Setup:
    SHELL[0,:] - ELEMENT ID
    SHELL[1,:] - PART ID
    SHELL[2,:] - NODE 1
    SHELL[3,:] - NODE 2
    SHELL[4,:] - NODE 3
    SHELL[5,:] - NODE 4
    """

  #SEPARATE SHELLS IN PARTS BASED ON NODES LOCATION IN LAYERS.
  SHELL = zeros((6 , (max_x-1)*(max_y-1)) )
  SHELL[0,:] = arange(1, (max_x-1)*(max_y-1) +1) #Sequential Shells
  SHELL[1,:] = 1 # Only one part
  tmp = arange(1, max_x)
  part = 1;
  for i in range( (max_y-1) ):
    #Find part
    for j in range(len(lHinNodes)):
      if (i < lHinNodes[j]):
        part = j+1
        break

    #Construct Elements
    SHELL[1,i*(max_x-1):(i+1)*(max_x-1)] = part
    SHELL[2,i*(max_x-1):(i+1)*(max_x-1)] = tmp + i*max_x
    SHELL[3,i*(max_x-1):(i+1)*(max_x-1)] = tmp + 1 + i*max_x
    SHELL[4,i*(max_x-1):(i+1)*(max_x-1)] = tmp + 1 + (i+1)*max_x
    SHELL[5,i*(max_x-1):(i+1)*(max_x-1)] = tmp + (i+1)*max_x


  #Set up LOAD Curve, Normalize
  #load_tr = upv.ascii2upvtt(load_fn)
  load_tr = upv.ascii2upvtt(load_fn)
  load_tr = upv.upvtt((load_tr.time - load_tr.time[0]), (load_tr.data-load_tr.data[0])/max(load_tr.data))


  # Write the Data to LSDYNA ".k" file
  fo = open(out_fn, "w")
  fo.write('*KEYWORD\n')
  fo.write('*TITLE\n Autmatically generated by layergenDyna.py\n')

  #*parts #WRITE ALL PARTS HERE -- Done

  for i in arange(len(aL)):
    fo.write('*PART\n')
    fo.write('$# title\n')
    fo.write('Layer %i\n' % (i+1))
    fo.write('$#     pid     secid       mid     eosid      hgid      grav    adpopt      tmid\n')
    fo.write('%10i%10i%10i%10i%10i%10i%10i%10i\n' % (i+1,1,i+1,0,0,0,0,0))

  #*mat   #WRITE ALL MATS HERE
  for i in arange(len(aL)):
    fo.write('*MAT_ELASTIC_TITLE\n')
    fo.write('Material Layer\n');
    fo.write('$      MID        RO         E        PR        DA        DB\n');
    fo.write('$#     mid        ro         e        pr        da        db  not used\n')
    fo.write('%10i%10.2f%10.0f%10.2f%10.0f%10.2f%10i\n' % (i+1,aL[i][4],aL[i][2],aL[i][3],0.0, 0.0, 0))
```

```
#*section - Same section for everything
fo.write('*SECTION_SHELL_TITLE\n')
fo.write('P-1\n');
fo.write('$    SECID    ELFORM      SHRF       NIP     PROPT   QR/IRID     ICOMP     SETYP\n')
fo.write('$#   secid    elform      shrf       nip     propt   qr/irid     icomp     setyp\n')
fo.write('%10i%10i%10f%10i%10i%10i%10i%10i\n' % (1,15,0.83,4,1,0,0,1))
fo.write('$#      t1        t2        t3        t4      nloc     marea\n')
fo.write('%10f%10f%10f%10f%10i%10f\n' % (0.0, 0.0, 0.0, 0.0, 1, 0.0))

#*define_curve_Title
if (n_1 > 0):
  fo.write('*DEFINE_CURVE_TITLE\n')
  fo.write('Damping Curve\n')
  fo.write('$     LCID      SIDR       SFA       SFO      OFFA      OFFO    DATTYP\n')
  fo.write('$#    lcid      sidr       sfa       sfo      offa      offo    dattyp\n')
  fo.write('%10i%10i%10f%10f%10f%10f%10i\n'  % (2, 0, 1.0, 1.0, 0.0, 0.0, 0))
  fo.write('$#                a1                  o1\n')
  fo.write('%20f%20f\n' % (0.0, n_1))
  fo.write('%20f%20f\n' % (100.0, n_1))

  fo.write('*DAMPING_PART_MASS\n')
  fo.write('$#     pid      lcid        sf      flag\n')
  fo.write('%10i%10i%10f%10f\n' % (1,2,1.0,0))
  fo.write('*DAMPING_PART_STIFFNESS\n')
  fo.write('$#     pid      coef\n')
  fo.write('%10i%10f\n' % (1, -n_2))

fo.write('*DEFINE_CURVE_TITLE\n')
fo.write('Lamb_Source\n')
fo.write('$     LCID      SIDR       SFA       SFO      OFFA      OFFO    DATTYP\n')
fo.write('$#    lcid      sidr       sfa       sfo      offa      offo    dattyp\n')
fo.write('%10i%10i%10f%10f%10f%10f%10i\n'  % (1, 0, 1.0, 1e-8, 0.0, 0.0, 0))
fo.write('$#                a1                  o1\n')

for i in arange(len(load_tr.time)):
  fo.write('%20f%20f\n' % (load_tr.time[i], load_tr.data[i]) )
fo.write('*LOAD_NODE_POINT\n')
fo.write('$     NODE       DOF      LCID        SF       CID        M1        M2        M3\n')
fo.write('$#     nid       dof      lcid        sf       cid        m1        m2        m3\n')
for i in range(td_sz): #First td_sz nodes are loaded equally by the same curve
  fo.write('%10i%10i%10i%10.1f%10i%10i%10i%10i\n' % (i+1, 2, 1, 1.0, 0, 0, 0, 0))

# Output Database
fo.write('*DATABASE_NODOUT\n')
fo.write('$       DT    BINARY\n')
fo.write('$#      dt    binary\n')
fo.write('%10f%10i\n' % ((load_tr.time[1]-load_tr.time[0]), 1))
fo.write('*DATABASE_HISTORY_NODE_ID\n')
fo.write('$#     id1                heading\n')
outnodes = set(list(HB.flat) + list(LB) + list(RB))
for i in outnodes:
  fo.write('%10i%70s\n' % ( i,'Output_Point'))

fo.write('*DATABASE_GLSTAT\n')
fo.write('$#      dt    binary\n')
fo.write('%10f%10i\n' % ((load_tr.time[1]-load_tr.time[0])*2, 1))

fo.write('*DATABASE_BINARY_D3PLOT\n')
fo.write('$#      dt      lcdt      beam     npltc\n')
fo.write('%10f%10i%10i%10i\n'% ((load_tr.time[1]-load_tr.time[0])*5000,0,0, 0))
fo.write('$#   ioopt\n')
fo.write('%10i\n' %  (0))

#termination
fo.write('*CONTROL_TERMINATION\n')
fo.write('$   ENDTIM    ENDCYC     DTMIN    ENDENG    ENDMAS\n')
fo.write('$#  endtim    endcyc     dtmin    endeng    endmas\n')
fo.write('%10f%10i%10f%10f%10f\n' % (t_t, 0, 0.0, 0.0, 0.0))

#time steps
fo.write('*CONTROL_TIMESTEP\n')
fo.write('$   DTINIT    TSSFAC      ISDO    TSLIMT     DT2MS      LCTM     ERODE     MS1ST\n')
fo.write('$#  dtinit    tssfac      isdo    tslimt     dt2ms      lctm     erode     ms1st\n')
fo.write('%10f%10f%10i%10f%10f%10i%10i\n' % (0.0, 0.9, 0, 0.0, 0.0, 0, 0))
fo.write('$#  dt2msf   dt2mslc\n')
fo.write('%10f%10i\n' % (0.0, 0))

#left boundary
fo.write('*SET_NODE_LIST_TITLE\n')
fo.write('LB\n');
fo.write('$      SID       DA1       DA2       DA3       DA4\n')
fo.write('%10i%10f%10f%10f%10f\n' % (1, 0.0, 0.0, 0.0, 0.0))
fo.write('$     NID1      NID2      NID3      NID4      NID5      NID6      NID7      NID8\n')

for i in arange(len(LB)/8):
  fo.write('%10i%10i%10i%10i%10i%10i%10i%10i\n' % tuple(LB[i*8:(i+1)*8]))
for i in arange(len(LB)%8):
  fo.write('%10i' % LB[(len(LB)/8)*8 + i])

#right boundary
fo.write('\n*SET_NODE_LIST_TITLE\n')
fo.write('RB\n')
fo.write('$      SID       DA1       DA2       DA3       DA4\n')
fo.write('%10i%10f%10f%10f%10f\n' % (2, 0.0, 0.0, 0.0, 0.0))
fo.write('$     NID1      NID2      NID3      NID4      NID5      NID6      NID7      NID8\n')
for i in arange(len(RB)/8):
  fo.write('%10i%10i%10i%10i%10i%10i%10i%10i\n' % tuple(RB[i*8:(i+1)*8]))
for i in arange(len(RB)%8):
  fo.write('%10i' % RB[(len(LB)/8)*8 + i])

#Horizontal Boundaries
fo.write('\n*SET_NODE_LIST_TITLE\n')
fo.write('TB\n')
fo.write('$      SID       DA1       DA2       DA3       DA4\n')
fo.write('%10i%10f%10f%10f%10f\n' % (3, 0.0, 0.0, 0.0, 0.0))
fo.write('$     NID1      NID2      NID3      NID4      NID5      NID6      NID7      NID8\n')
for i in arange(len(list(HB.flat))/8):
  fo.write('%10i%10i%10i%10i%10i%10i%10i%10i\n' % tuple(HB.flat[i*8:(i+1)*8]))
for i in arange(len(list(HB.flat))%8):
  fo.write('%10i' % HB.flat[(len(list(HB.flat))/8)*8 + i])


#Shells
fo.write('\n*ELEMENT_SHELL\n');
fo.write('$    EID     PID    NID1    NID2    NID3    NID4\n')
fo.write('$#   eid     pid      n1      n2      n3      n4\n')
for i in arange(len(SHELL[0,:])):
  fo.write('%8i%8i%8i%8i%8i%8i\n' % tuple(SHELL[:,i]))


#NODES
fo.write('*NODE\n');
fo.write('$    NID               X               Y               Z      TC      RC\n');
fo.write('$#   nid               x               y               z      tc      rc\n');
for i in arange(len(NODES[0,:])):
  fo.write('%8i%16f%16f%16f%8i%8i\n' % tuple(NODES[:,i]))


#END
fo.write('*END\n')
fo.close()

#########
#Save Boundary Information Data for Database Import
#########
```

```
    print HB.tolist()
    fo = open(dir + "HBs", "w")
    pickle.dump(HB.tolist(), fo)
    fo.close()

    fo = open(dir + "RB", "w")
    pickle.dump(RB.tolist(), fo)
    fo.close()

    fo = open(dir + "LB", "w")
    pickle.dump(LB.tolist(), fo)
    fo.close()

    TD1 = range(td_sz)
    fo = open(dir + "TD1", "w")
    pickle.dump(TD1, fo)
    fo.close()

\end{lstlisting}


\section{CylgenDyna}
\label{sec:cylgendyna}

\begin{verbatim}
  #! /usr/bin/python
  """ Scripts to generate an elastic cylinder of fully
  integrated LS-DYNA 2D Axisymmetric elements in the
  form of a homogeneous cylinder of specified dimensions.
  The dimensions, element size, transducer size, input
  trace are all entered at the command line.

    This is a somewhat specialized script and should only
  be used by people that have knowledge of the internal
  workings of this scripts. It has a lot of specialized
  functions used in a specific research project.

  Author: Simon Berube, 2007
  Usage: (All sizes in mm)
    cylgenDyna.py nX nY dX dY td_size impulse_file dir Output_file rho E p_r t_t n_1? n_2?
    cylgenDyna.py 101 101 0.25 0.25 30 1M_Pulse.txt DIR m1a.k 0.00210 27440 0.33 0.05
  python cylgendyna.py 280 526 0.25 0.25 20 sineIn_out.txt ./ alum.k 0.002713 73000 0.33 0.25

  Load Curve: tau = 0.001, skip = 0.00002*800 t=arage(0,1000)*0.0005 -THANK YOU!
  cylgendyna 600 600 0.4167 0.4167 1 ./lsource.txt ~/code/Lamb/ lamb.k 0.00210 29578 0.33 0.1

  """
  import pickle
  from numpy import array,zeros, arange #For array management
  import sys #For system arguments
  import upv #For time traces, and reading from oscilloscope

  #Obtain and Simplify Input arguments
  max_x = int(sys.argv[1])
  max_y = int(sys.argv[2])
  dx = float(sys.argv[3])
  dy = float(sys.argv[4])
  td_sz = int(sys.argv[5])
  load_fn = sys.argv[6]
  dir = sys.argv[7]
  out_fn = dir + sys.argv[8]
  rho = float(sys.argv[9])
  E = float(sys.argv[10])
  p_r = float(sys.argv[11])
  t_t = float(sys.argv[12])
  try:
   n_1 = float(sys.argv[13])
```

```
 n_2 = float(sys.argv[14])
except:
 n_1 = -1
 n_2 = -1

#Define the mesh, boundaries vectors
NODES = zeros((6,max_x*max_y))
NODES[0,:] = arange(1, max_x*max_y+1)
LB = zeros(max_y) #Left Boundary
RB = zeros(max_y) #Right Boundary

""" NODES Setup:
  NODES[0,:] - NODE ID
  NODES[1,:] - X COORD
  NODES[2,:] - Y COORD
  NODES[3,:] - Z COORD
  NODES[4,:] - tc flag
  NODES[5,:] - rc flag
  """
#Set up Nodes Vectors
dxv = arange(max_x)*dx
TB = arange(max_x)+1 #First max_x nodes are top boundary
BB = arange(max_x)+(max_x*(max_y-1)) + 1 #Bottom Boundary
for i in range(max_y):
  NODES[1,i*max_x:(i+1)*max_x] = dxv
  NODES[2,i*max_x:(i+1)*max_x] = i*dy
  LB[i] = i*max_x + 1
  RB[i] = (i+1)*max_x

#Set up Shells
""" SHELL Setup:
  SHELL[0,:] - ELEMENT ID
  SHELL[1,:] - PART ID
  SHELL[2,:] - NODE 1
  SHELL[3,:] - NODE 2
  SHELL[4,:] - NODE 3
  SHELL[5,:] - NODE 4
  """
SHELL = zeros((6 , (max_x-1)*(max_y-1)) )
SHELL[0,:] = arange(1, (max_x-1)*(max_y-1) +1) #Sequential Shells
SHELL[1,:] = 1 # Only one part
tmp = arange(1, max_x)
for i in range( (max_y-1) ):
  SHELL[2,i*(max_x-1):(i+1)*(max_x-1)] = tmp + i*max_x
  SHELL[3,i*(max_x-1):(i+1)*(max_x-1)] = tmp + 1 + i*max_x
  SHELL[4,i*(max_x-1):(i+1)*(max_x-1)] = tmp + 1 + (i+1)*max_x
  SHELL[5,i*(max_x-1):(i+1)*(max_x-1)] = tmp + (i+1)*max_x


#Set up LOAD Curve, Normalize
#load_tr = upv.ascii2upvtt(load_fn)
load_tr = upv.ascii2upvtt(load_fn)
load_tr = upv.upvtt((load_tr.time - load_tr.time[0])*1000, (load_tr.data-load_tr.data[0])/max(load_tr.data))


# Write the Data to LSDYNA ".k" file
fo = open(out_fn, "w")
fo.write('*KEYWORD\n')
fo.write('*TITLE\n Autmatically generated by cylgenDyna.py\n')
#*part
fo.write('*PART\n')
fo.write('$# title\n')
fo.write('%80s\n' % ('Part 1'))
fo.write('$#     pid     secid       mid     eosid      hgid      grav    adpopt      tmid\n')
fo.write('%10i%10i%10i%10i%10i%10i%10i%10i\n' % (1,1,1,0,0,0,0,0))

#*mat
fo.write('*MAT_ELASTIC_TITLE\n')
fo.write('Material Layer\n');
fo.write('$      MID        RO         E        PR        DA        DB\n');
fo.write('$#     mid        ro         e        pr        da        db  not used\n')
fo.write('%10i%10.5f%10.0f%10.2f%10.0f%10.2f%10i\n' % (1,rho,E,p_r,0.0, 0.0, 0))

#*section
fo.write('*SECTION_SHELL_TITLE\n')
fo.write('P-1\n');
fo.write('$    SECID    ELFORM      SHRF       NIP     PROPT   QR/IRID     ICOMP     SETYP\n')
fo.write('$#   secid    elform      shrf       nip     propt   qr/irid     icomp     setyp\n')
fo.write('%10i%10i%10f%10i%10i%10i%10i%10i\n' % (1,15,0.83,4,1,0,0,1))
fo.write('$#      t1        t2        t3        t4      nloc     marea\n')
fo.write('%10f%10f%10f%10f%10i%10f\n' % (0.0, 0.0, 0.0, 0.0, 1, 0.0))

#*define_curve_Title
if (n_1 > 0):
 fo.write('*DEFINE_CURVE_TITLE\n')
 fo.write('Damping Curve\n')
 fo.write('$     LCID      SIDR       SFA       SFO      OFFA      OFFO    DATTYP\n')
 fo.write('$#    lcid      sidr       sfa       sfo      offa      offo    dattyp\n')
 fo.write('%10i%10i%10f%10f%10f%10f%10i\n'  % (2, 0, 1.0, 1.0, 0.0, 0.0, 0))
 fo.write('$#                a1                  o1\n')
 fo.write('%20f%20f\n' % (0.0, n_1))
 fo.write('%20f%20f\n' % (100.0, n_1))

 fo.write('*DAMPING_PART_MASS\n')
 fo.write('$#     pid      lcid        sf      flag\n')
 fo.write('%10i%10i%10f%10f\n' % (1,2,1.0,0))
 fo.write('*DAMPING_PART_STIFFNESS\n')
 fo.write('$#     pid      coef\n')
 fo.write('%10i%10f\n' % (1, -n_2))

fo.write('*DEFINE_CURVE_TITLE\n')
fo.write('Lamb_Source\n')
fo.write('$     LCID      SIDR       SFA       SFO      OFFA      OFFO    DATTYP\n')
fo.write('$#    lcid      sidr       sfa       sfo      offa      offo    dattyp\n')
fo.write('%10i%10i%10f%10f%10f%10f%10i\n'  % (1, 0, 1.0, 1e-8, 0.0, 0.0, 0))
fo.write('$#                a1                  o1\n')

for i in arange(len(load_tr.time)):
  fo.write('%20f%20f\n' % (load_tr.time[i], load_tr.data[i]) )
fo.write('*LOAD_NODE_POINT\n')
fo.write('$     NODE       DOF      LCID        SF       CID        M1        M2        M3\n')
fo.write('$#     nid       dof      lcid        sf       cid        m1        m2        m3\n')
for i in range(td_sz): #First td_sz nodes are loaded equally by the same curve
  fo.write('%10i%10i%10i%10.1f%10i%10i%10i%10i\n' % (i+1, 2, 1, 1.0, 0, 0, 0, 0))

# Output Database
fo.write('*DATABASE_NODOUT\n')
fo.write('$       DT    BINARY\n')
fo.write('$#      dt    binary\n')
fo.write('%10f%10i\n' % (t_t/4096., 1))
fo.write('*DATABASE_HISTORY_NODE_ID\n')
fo.write('$#     id1                heading\n')
outnodes = set(list(BB) + list(TB) + list(LB) + list(RB))
for i in outnodes:
  fo.write('%10i%70s\n' % ( i,'Output_Point'))

fo.write('*DATABASE_GLSTAT\n')
fo.write('$#      dt    binary\n')
fo.write('%10f%10i\n' % ((load_tr.time[1]-load_tr.time[0])*2, 1))

fo.write('*DATABASE_BINARY_D3PLOT\n')
fo.write('$#      dt      lcdt      beam     npltc\n')
fo.write('%10f%10i%10i%10i\n'% ((load_tr.time[1]-load_tr.time[0])*5000,0,0, 0))
fo.write('$#   ioopt\n')
fo.write('%10i\n' %  (0))
```

```
#termination
fo.write('*CONTROL_TERMINATION\n')
fo.write('$   ENDTIM    ENDCYC     DTMIN    ENDENG    ENDMAS\n')
fo.write('$#  endtim    endcyc     dtmin    endeng    endmas\n')
fo.write('%10f%10i%10f%10f%10f\n' % (t_t, 0, 0.0, 0.0, 0.0))

#time steps
fo.write('*CONTROL_TIMESTEP\n')
fo.write('$   DTINIT    TSSFAC      ISDO    TSLIMT     DT2MS      LCTM     ERODE     MS1ST\n')
fo.write('$#  dtinit    tssfac      isdo    tslimt     dt2ms      lctm     erode     ms1st\n')
fo.write('%10f%10f%10i%10f%10f%10i%10i\n' % (0.0, 0.9, 0, 0.0, 0.0, 0, 0))
fo.write('$#  dt2msf   dt2mslc\n')
fo.write('%10f%10i\n' % (0.0, 0))

#left boundary
fo.write('*SET_NODE_LIST_TITLE\n')
fo.write('LB\n');
fo.write('$      SID       DA1       DA2       DA3       DA4\n')
fo.write('%10i%10f%10f%10f%10f\n' % (1, 0.0, 0.0, 0.0, 0.0))
fo.write('$     NID1      NID2      NID3      NID4      NID5      NID6      NID7      NID8\n')

for i in arange(len(LB)/8):
  fo.write('%10i%10i%10i%10i%10i%10i%10i%10i\n' % tuple(LB[i*8:(i+1)*8]))
for i in arange(len(LB)%8):
  fo.write('%10i' % LB[(len(LB)/8)*8 + i])

#right boundary
fo.write('\n*SET_NODE_LIST_TITLE\n')
fo.write('RB\n')
fo.write('$      SID       DA1       DA2       DA3       DA4\n')
fo.write('%10i%10f%10f%10f%10f\n' % (2, 0.0, 0.0, 0.0, 0.0))
fo.write('$     NID1      NID2      NID3      NID4      NID5      NID6      NID7      NID8\n')
for i in arange(len(RB)/8):
  fo.write('%10i%10i%10i%10i%10i%10i%10i%10i\n' % tuple(RB[i*8:(i+1)*8]))
for i in arange(len(RB)%8):
  fo.write('%10i' % RB[(len(LB)/8)*8 + i])

#top boundary
fo.write('\n*SET_NODE_LIST_TITLE\n')
fo.write('TB\n')
fo.write('$      SID       DA1       DA2       DA3       DA4\n')
fo.write('%10i%10f%10f%10f%10f\n' % (3, 0.0, 0.0, 0.0, 0.0))
fo.write('$     NID1      NID2      NID3      NID4      NID5      NID6      NID7      NID8\n')
for i in arange(len(TB)/8):
  fo.write('%10i%10i%10i%10i%10i%10i%10i%10i\n' % tuple(TB[i*8:(i+1)*8]))
for i in arange(len(TB)%8):
  fo.write('%10i' % TB[(len(TB)/8)*8 + i])

#bottom boundary
fo.write('\n*SET_NODE_LIST_TITLE\n')
fo.write('BB\n')
fo.write('$      SID       DA1       DA2       DA3       DA4\n')
fo.write('%10i%10f%10f%10f%10f\n' % (4, 0.0, 0.0, 0.0, 0.0))
fo.write('$     NID1      NID2      NID3      NID4      NID5      NID6      NID7      NID8\n')
for i in arange(len(BB)/8):
  fo.write('%10i%10i%10i%10i%10i%10i%10i%10i\n' % tuple(BB[i*8:(i+1)*8]))
for i in arange(len(BB)%8):
  fo.write('%10i' % BB[(len(BB)/8)*8 + i])

#Shells
fo.write('\n*ELEMENT_SHELL\n');
fo.write('$    EID     PID    NID1    NID2    NID3    NID4\n')
fo.write('$#   eid     pid      n1      n2      n3      n4\n')
for i in arange(len(SHELL[0,:])):
  fo.write('%8i%8i%8i%8i%8i%8i\n' % tuple(SHELL[:,i]))


#NODES
```

```
  fo.write('*NODE\n');
  fo.write('$    NID               X               Y               Z      TC      RC\n');
  fo.write('$#   nid               x               y               z      tc      rc\n');
  for i in arange(len(NODES[0,:])):
    fo.write('%8i%16f%16f%16f%8i%8i\n' % tuple(NODES[:,i]))


  #END
  fo.write('*END\n')
  fo.close()

  #########
  #Save Boundary Information Data for Database Import
  #########
  fo = open(dir + "TB", "w")
  pickle.dump(TB.tolist(), fo)
  fo.close()

  fo = open(dir + "BB", "w")
  pickle.dump(BB.tolist(), fo)
  fo.close()

  fo = open(dir + "RB", "w")
  pickle.dump(RB.tolist(), fo)
  fo.close()

  fo = open(dir + "LB", "w")
  pickle.dump(LB.tolist(), fo)
  fo.close()

  TD1 = range(td_sz)
  fo = open(dir + "TD1", "w")
  pickle.dump(TD1, fo)
  fo.close()
```

B.2 modelGenerator1M

```
#! /usr/bin/python
""" Script that generates directory structure, as well as LS-DYNA files for
 each of our concrete specimen. The program then proceeds to solve the problem
  with standard settings.
"""
import os

#Configuration (mm, ms, N, g) - Assuming V_p = 4600m/s
dx = 0.25
dy = 0.25
rho = 0.002102
E = 29578
p_r = 0.33
end_time = 0.05

#damping?
#n_1 = 107711.657 #Mass Damping
#n_2 = 7.2757e-9 #Stiff Damping

n_1 = 0
n_2 = 0

folder = '1M'
MTDInput = '1M_Pulse.txt'
MTDSize  = '60' #This depends on the DX set
solver = 'C:\LSDYNA2\program\ls970_s_6763_win32.exe MEMORY=180000000 I='

# Name:(Height, Width)
spec  = {'m1a':(50,50),
        'm2a':(100,50),
 'm3a':(200,50),
 'm4a':(300,50),
 'm1b':(50,100),
 'm2b':(100,100),
 'm3b':(200,100),
 'm4b':(300,100),
 'm1c':(50,150),
 'm2c':(100,150),
 'm3c':(200,150),
 'm4c':(300,150)}


#Create files and directories
os.mkdir(folder)
for k,v in spec.iteritems():
os.mkdir(folder + "/" + k)
cmd = ('python cylgenDyna.py ' \
+ str(int(v[1]/dx+1)) + ' ' \
+ str(int(v[0]/dy+1)) + ' ' \
+ str(dx) + ' ' + str(dy) + ' ' \
+ MTDSize + ' ' + MTDInput + ' ' \
+ './' + folder + '/' + k + '/ ' + k + '.k ' \
+ str(rho) + ' ' + str(E) + ' ' \
+ str(p_r) + ' ' + str(end_time))

if (n_1 > 0):
cmd = cmd + ' ' + str(n_1) + ' ' + str(n_2)
os.system(cmd)


#Now, Solve!!!!
os.chdir(folder)
for k, v in spec.iteritems():
os.chdir(k)
os.system(solver + k + '.k')
os.chdir('..')
```

B.3 modelGenerator50K

```
  #! /usr/bin/python
  """ Script that generates directory structure, as well as LS-DYNA files for
 each of our concrete specimen. The program then proceeds to solve the problem with standard settings.
  """
  import os

  #Configuration (mm, ms, N, g) - Assuming V_p = 4600m/s
  dx = 0.25
  dy = 0.25
  rho = 0.002102
  E = 29578
  p_r = 0.33
  end_time = 0.05

  #damping?
  #n_1 = 107711.657 #Mass Damping
  #n_2 = 7.2757e-9 #Stiff Damping

  n_1 = 0
  n_2 = 0

  folder = '1M'
  MTDInput = '1M_Pulse.txt'
  MTDSize  = '60' #This depends on the DX set
  solver = 'C:\LSDYNA2\program\ls970_s_6763_win32.exe MEMORY=180000000 I='
```

```
# Name:(Height, Width)
spec  = {'m1a':(50,50),
         'm2a':(100,50),
  'm3a':(200,50),
  'm4a':(300,50),
  'm1b':(50,100),
  'm2b':(100,100),
  'm3b':(200,100),
  'm4b':(300,100),
  'm1c':(50,150),
  'm2c':(100,150),
  'm3c':(200,150),
  'm4c':(300,150)}


#Create files and directories
os.mkdir(folder)
for k,v in spec.iteritems():
 os.mkdir(folder + "/" + k)
 cmd = ('python cylgenDyna.py ' \
 + str(int(v[1]/dx+1)) + ' ' \
 + str(int(v[0]/dy+1)) + ' ' \
 + str(dx) + ' ' + str(dy) + ' ' \
 + MTDSize + ' ' + MTDInput + ' ' \
 + './' + folder + '/' + k + '/ ' + k + '.k ' \
 + str(rho) + ' ' + str(E) + ' ' \
 + str(p_r) + ' ' + str(end_time))

 if (n_1 > 0):
 cmd = cmd + ' ' + str(n_1) + ' ' + str(n_2)
 os.system(cmd)


#Now, Solve!!!!
os.chdir(folder)
for k, v in spec.iteritems():
 os.chdir(k)
 os.system(solver + k + '.k')
 os.chdir('..')
```

B.4 tt

```
"""
Database time trace utility objects. See objects for details.
Developped by: Simon Berube, 2007
"""
#Scientific Tools
from numpy import ndarray, array, arange, angle, trapz, mean, hstack,zeros, where, mean, ceil
from numpy import log, log10
from numpy.fft import rfft, fft2
from numpy.lib import unwrap
from pylab import plot, title, xlabel, ylabel, show, xlim, ylim, subplot, ion
from pylab import ioff, isinteractive, savefig, close, polyfit, legend, axes, imshow
from pylab import figure, rcParams, gca, show, setp, connect, draw, contourf, contour
import pylab
from matplotlib.transforms import Bbox, Value, Point, get_bbox_transform, unit_bbox

#PostgreSQL and Regexp
from psycopg2 import connect
import re
import datetime

""" ========================================
  Generally Useful Functions
    ======================================== """
```

```
def lst2sql(lst):
  s = "%s" % str(lst)
  s = re.sub("\[","{", s)
  s = re.sub("\]","}", s)
  s = re.sub(" ", "", s)
  return s

def sql2lst(lst):
  lst = re.sub('{','', lst)
  lst = re.sub('}','', lst)
  lst = re.sub(',',' ', lst)
  lst = map(float, lst.split())
  return lst

""" ===================================
  OBJECTS tt, upvtt, upvtt_list
    =================================== """

class tt(dict):
  """ Time trace class, contains procedures for handling,
  processing and visualizing basic time traces of any
  kind. It contains a dictionary of non-related paramters
  that can be used to identify and categorize the traces.
  This is mostly useful for database interaction whereas
  database values can be stored inside the object.

  These properties can be accessed using dictionary structures
  such as:
    tt[KEY] = Value OR Value = tt[KEY]

  #DEV_ITEM
  Has operator overloading of addition, substractions
  and multiplication. Please see the operator description
  for their behaviors.

  Also clean up PLOT function to take in **kwargs instead
  of godawfully long function descriptions
  #END_DEV_ITEM

  Accessible Class Members
  Name  Type  Desc
  ========================================================
  time  ndarray  The time vector (RO)
  data  ndarray  The data vector (RO)
  T  ndarray  Fourier Transform of the time trace (RO)
  f  ndarray  Frequency vector of the Fourier Transform (RO)
  P  ndarray  Power Spectrum corresponding to f (RO)
  phi  ndarray  Phase angle corresponding to f (RO)

  ----

  mset  set()  Contains keys of changed dictionary values
  cval  dict()  Contains dictionary values. Not to use directly!
  """

  """ ++++++++++++++++++++++++
    DATA ACCESSORS(RO)
      ++++++++++++++++++++++++ """
  def get_t(self): return self._time
  def get_d(self): return self._data
  def get_f(self):
    self._comp_fft()
    return self._f #Frequency Vector (Hz)
  def get_T(self):
    self._comp_fft()
    return self._T #Raw FFT complex data
  def get_P(self):
    self._comp_fft()
    #Power (Amplitude squared)
    return (self._T * self._T.conj())/len(self.time)
  def get_phi(self):
    self._comp_fft()
    return angle(self.T) #Phase angle

  time = property(get_t, None, None, "Time")
  data = property(get_d, None, None, "Data")
  f = property(get_f, None, None, "Frequency")
  T = property(get_T, None, None, "Transform - Complex")
  P = property(get_P, None, None, "Power")
  phi = property(get_phi, None, None, "Phase Angle")

  """ ++++++++++++++++++++++++++++++++++++++++
    SEMI-PRIVATE CLASS FUNCTIONS
      ++++++++++++++++++++++++++++++++++++++++ """
  def __init__(self,t= None, d = None, vdict = {}):
    """ Initilization of time trace:
      __init__(time, data)

      t:
        The time vector, as numpy ndarray. This
        is only mandatory if you do not define a
        trng value.

      d:
        data vector, as numpy ndarray, mandatory
        value must be given for all initializations.

      vdict:  dictionary of key/value pairs of any kind to
        keep track of time trace properties. This is
        intended to be flexible and used as the user
        sees fit. The use was originally intended
        to hold database information about a specific
        time trace.

      #DEV_ITEM: Add mode that stores P,Phi instead of r
      ecomputing
    """
    super(tt, self).__init__()

    #Check and init variables
    if (isinstance(d, ndarray) and isinstance(t,ndarray)):
      if ( len(d) == len(t)):
        self._data = d
        self._time = t
      else:
        raise ValueError('Vector Dimension Mismatch')
    else:
      raise TypeError('Wrong data type for d or t')

    #Init other class variables
    self._T = None
    self.cval = {}
    self.mset = set()

    if isinstance(vdict, dict):
      self.cval = vdict;
    else:
      raise TypeError('Wrong data type for vdict')

  def _comp_fft(self):
    """ Computes the FFT data whenever it is used.
    Does not compute again if already calculated.

    TODO: Add option to zero average signal
    """
    if (self._T == None):
      #Now compute the FFT
      f_n =self.sfreq()/2.0
      self._T = rfft(self.data)
      self._f =arange(0, len(self._T))/float(len(self._T)-1)*f_n

      if len(self.T) != len(self.f):
        raise ValueError('Stuff %i, %i' % (len(self.T),
          len(self.f)))

  def __repr__(self):
    return "<tt object at 0x%x>" % id(self)

  def __str__(self):
    return "<tt object at 0x%x>" % id(self)

  def __len__(self):
    return len(self.time) #Length is number of time elements

  def _amp_maxplot(self):
    return abs(self.data.max()) #Return maximum data

  def __getitem__(self, name):
    return self.cval[name]

  def _amp_tot_freq(self):
    # Simply use trapz(self.f,self.P) or somesuch
    return trapz(self.P, self.f)

  def __setitem__(self, name, value):
    self.mset.add(name)
    self.cval[name] = value #Update Value

  """ ++++++++++++++++++++++++++++++++++++++++
    PUBLIC CLASS FUNCTIONS
      ++++++++++++++++++++++++++++++++++++++++ """
  def plot(self,  *args, **kwargs):
    """ Plots the time trace using the matplotlib library.
    can be called with no input argument for a default full
    plot of the data.

    This works best in  iPython, if not make sure your
    matplotlibrc file is setup accurately to handle
    a back-end that works with your system.
    (Tk should have best compatibility)

    Available Options: (Same as matplotlib plot, and axes)

    All **kwargs will be redirected to axes as follows:
      axes.set(**kwargs)
      EXCEPT:
        'on_axes':? can be used to set
        an existing axe for plotting. This
        is useful when embedding in a GUI
        application.

    All *args will be redirected to plot as follows:

      If only 1 arg is given, it will be appended
      after the x, y arguments (set by default)

      If no args are given, defaults x, y, color
      will be set

      if 2 or more arguments are given, they
      are passed directly to pylab.plot(*args)

    Usage:
      plot(self, *args, **kwargs)
    Defaults:
      title  = 'Time Trace'
      xlabel ='Time'
      ylabel ='Amplitude'
      on_axes = None

    """
    defaults = {  'title':'Time Trace',
        'xlabel':'Time',
        'ylabel':'Amplitude',
        'on_axes': None,
        }
    for k, v in defaults.iteritems():
      if k not in kwargs:
        kwargs[k] = v

    if not len(args): args = (self.time, self.data, 'b')
    elif len(args) == 1: args = (self.time, self.data, args[0])

    iset = False
    do_show = False
    on_axes = kwargs.pop('on_axes')

    if (isinteractive()):
      ioff() #Disable Interactive Mode if active
      iset = True

    if (on_axes == None):
      plot(*args)
      ax = gca()
      do_show = True
    else:
      on_axes.plot(*args)
      ax = on_axes

    ax.set(**kwargs)

    if do_show: show() #If no renderers, simply create a new backend
    if iset: ion()

  def plotP(self, *args, **kwargs):
    """ Plot of the Power Spectrum of the time trace,
    See self.plot() """

    defaults = {  'title':'Power Spectrum',
        'xlabel':'Frequency (Hz)',
        'ylabel':'Power',
        'on_axes': None  }

    for k, v in defaults.iteritems():
      if k not in kwargs:
        kwargs[k] = v

    self.plot(self.f, self.P, *args, **kwargs)

  def plotT(self, *args, **kwargs):
    """ Plots the amplitude and unwrapped phase angle of
    the computed Fourier Spectrum. As two subplots.

    This function is not very customizable, unlike
    self.plot() and self.plotP() it will not react
    to the following kwargs:

    'title', 'ylabel', 'ylim'

    But has will accept a current figure
```

```
    'on_fig'
    """
    defaults = {  'xlabel':'Frequency (Hz)',
        'on_fig': None  }

    for k, v in defaults.iteritems():
      if k not in kwargs:
        kwargs[k] = v

    phi = unwrap(self.phi) #Compute once, for efficiency
    amp = abs(self.T)
    on_fig = kwargs.pop('on_fig')

    #Top Amplitude Plot
    if not on_fig:
      subplot(2,1,1)
      ax = gca()
    else:
      ax = on_fig.add_subplot(211)

    kwargs['title'] = 'Amplitude Spectrum'
    kwargs['ylabel'] = 'Amplitude'
    kwargs['on_axes'] = ax
    self.plot(self.f, amp, *args, **kwargs)

    if not on_fig:
      subplot(2,1,2)
      ax = gca()
    else:
      ax = on_fig.add_subplot(212)

    kwargs['title'] = 'Phase Spectrum'
    kwargs['ylabel'] = 'Angle'
    kwargs['on_axes'] = ax
    self.plot(self.f, phi, *args, **kwargs)

    if not on_fig:
      show()
      #Change focus, this is to fix bug with GTKAgg backend.
      #DEV_ITEM: Fix this hack solution
      subplot(2,1,2)

  def sfreq(self):
    """ Returns the sampling frequency of the time trace (dt) """
    return 1/(self.time[1] - self.time[0])

  def amp(self, method='maxplot'):
    """ Calculates the amplitude of the time trace using the
    selected method. Available methods are:

    'maxplot' - Returns the maximum value at any point in the
        time trace

    'tot_freq' - Returns the area (integration) of the frequency
        domain information of the time trace.
    """
    if method == 'maxplot':
      return self._amp_maxplot()
    elif method == 'tot_freq':
      return self._amp_tot_freq()

  def updSQL(self, items, pkeys):
    """ Builds an update SQL statement that will save every
    modified values of this class as described by mset. For
    description of items, pkeys input as well as output
    please refer to self.insSQL() """
    #Finish up by building the WHERE statement
    #Builds a simple SQL statement for every modified arguments
    #and concatenate them into a single string. This is not the
    #most efficient but significantly reduces code complexity

    SQL = {}
    for k, v in pkeys.iteritems():
      SQL[k] = ''
      if v[0] in self.mset:
        raise ValueError('Primary Key of %s was modified in object! Aborting Save.' % k)

    for item in self.mset:
      if item in items.keys():
        tbl = items[item]
        SQL[tbl] += 'UPDATE "%s" SET "%s"=' % (items[item], item)

        if isinstance(self[item], str):
          SQL[tbl] += "'%s'" % self[item]
        else:
          SQL[tbl] += str(self[item])

        SQL[tbl] += ' WHERE "%s"=\'%s\';\n' % \
          (pkeys[items[item]][0], self[pkeys[items[item]][0]])
    return SQL

  def insSQL(self, items, pkeys):
    """ Returns a dictionary of SQL INSERT Statements that saves
    all key(columns)/Value pairs of tt. The function returns
    a dictionary of format:

      {'table_name': SQL INSERT STATEMENT,
       'table_name_2': SQL..., ... }

    items dictionary of format: (items)

      'item':'table_name'
      ...

    Primary keys is another dictionary containing the table:key
    names as follow:
      'table_name':('primary_key_column', 'link1', 'link2')

      'link1' and 'link2' are not used or needed but the
      input must still be a tuple dictionary to have
      seamless compatibility with tt_list.

    If object already exists, you can use updSQL() function
    that sends an SQL statement updating only the modified
    values.

    Usage:
      toSQL(self, items, pkey)
      toSQL(self, {'name':'upvtt', 'data':'upvtt', ...},
        {'upvtt':('name',)})
    """
    if (self.keys().__len__() == 0):
      raise ValueError('tt object has no dictionary keys')

    SQL_Dict = {}
    #Init all keys (This is inefficient and repeated, but.. meh)
    for k, v in pkeys.iteritems():
      SQL_Dict[k] = 'INSERT INTO ' + k + '("'
      if v[0] in self.mset:
        raise ValueError('Primary Key of %s was modified in object! Aborting Save.' % k)

    #Build the '(name1,name2,...) ' part
    for k, v in items.iteritems():
      SQL_Dict[v] += k + '", "'
    for k in SQL_Dict.keys():
      SQL_Dict[k] = SQL_Dict[k][0:-3] + ') VALUES ('

    #Build the '(value1, value2, value3, ...)' part
    for k, v in items.iteritems():
      if k in self.keys():
        if isinstance(self[k], str):
          SQL_Dict[v] += "'%s', " % self[k]
        elif isinstance(self[k], datetime.date):
          SQL_Dict[v] += "'%s', " % str(self[k])
        elif self[k] is None:
          SQL_Dict[v] += "NULL, "
        else:
          SQL_Dict[v] += str(self[k]) + ", "
      elif k == "time":
        SQL_Dict[v] += "'%s', " % lst2sql(self.time.tolist())
      elif k == "data":
        SQL_Dict[v] += "'%s', " % lst2sql(self.data.tolist())
      else: #No value
        SQL_Dict[v] += "NULL, "

    for k in SQL_Dict.keys():
      SQL_Dict[k] = SQL_Dict[k][0:-2] + ');'

    #Return the dictionary with all tables. (Main table will have to be saved
    #First.
    return SQL_Dict

  def tslice(self, start=None, end=None, nbpts=None):
    """ Returns a tt object of the sliced trace which
    contains data from start to end as a time value.

    If no end or out of bound is given, last point assumed.
    If no start or out of bound is given, first point assumed.

    tslice(self, start=None, end=None) NpuFIFij3DdxRT
    """

    istart = 1
    if (start != None):
      if (self.time.min() < start):
        istart = (self.time < start).nonzero()[0].max()
      else:
        pass
    iend = None
    if (end != None):
      if (self.time.max() > end):
        iend = (self.time > end).nonzero()[0].min()
      else:
        pass
    if (nbpts == None):
      return tt(self.time[istart:iend], self.data[istart:iend],
        self.cval)
    else:
      #Zero pad
      dt = self.time[1] - self.time[0]
      t=arange(self.time[istart],self.time[istart]+(nbpts)*dt,dt)
      dat = self.data[istart:iend]
      dat = hstack([dat, zeros(nbpts - len(dat))])

      return tt(t,dat,  self.cval)

  def clearSet(self):
    """Clears mset """
    self.mset = set()

  def keys(self):
    return self.cval.keys()
  def values(self):
    return self.cval.values()

  def iteritems(self):
    return self.cval.iteritems()

  def iterkeys(self):
    return self.cval.iterkeys()

  def itervalues(self):
    return self.cval.itervalues()

  def get(self):
    return self.cval.get()

  def has_key(self, key):
    return self.cval.has_key(key)
```

B.5 ttlist

```
class tt_list(list):
  """ Specialized class managing tt objects taken from a
  PGSQL Database, the list is table and database independent
  so it is useful as a base class for more specialized database
  designs.

  tt items are referred as a list. This is simpler for iterations and
  ordering. Dictionary listing do not have ordering which may cause
  problems when sorting values.

  All available values, as well as the column to which they belong are
  stored as key:value pairs (column : table) inside the self.columns
  dictionary.

  IMPORTANT NOTE:
    While the class is table independent you should still have
    columns named "time" and "data" that contain a string of
    the time and data array like:
      time = "{1,2,3,4,5,6}" or 2D as "{{1,2,3},{1,2,3}}

    Otherwise, the program will fail with an appropriate error.

  IMPORTANT NOTE2:
    As of yet, this class does not support one-to-many
    relationships. Each table must be directly linked
    as a one-to-one. (This may be changed when a clear
    need arises)

  DEV_ITEM: Add plot, slice and other multi-trace functions here
  """

  """ ++++++++++++++++++++++++++++++++++++++++
    SEMI-PRIVATE CLASS FUNCTIONS
      ++++++++++++++++++++++++++++++++++++++++ """
  def __init__(self, conn, tables, ptable, *args, **kwargs):
    """
    Gets items in the database that match the given creterias.
    and store them in this list object.

    __init__(self, conn, tables, pkey, *args, **kwargs)

    conn:
      A standard API 2.0 database connection object. This
      was programmed for psycopg2 on PostgreSQL 8.2.
```

```
tables:
  Dict the secondary tables and how they aer linked
  to the main table. The following format should
  be used.

  {'main_table':('pkey')
   'table1':('pkey', 'main_table_col', 'table1_col'),
   'table2':('pkey', 'main_table_col', 'table2_col'),
   ...
   }

   The first table MUST BE THE MAIN TABLE and contain
   time and data vectors

ptable:
  The name of the primary table, which must also be in
  tables as a key. With the value of taht key being
  a tuple, not a string.

At least ONE of the following is required:

*args:
  Arguments passed as SQL WHERE conditions that will
  be used to pull objects from database.

At least ONE of the following is required:

*args:
  Arguments passed as SQL WHERE conditions that will
  be used to pull objects from database.

  e.g. __init__('date_created > SOME_DATE')
    Gives SQL:
    SELECT ... WHERE date_created > SOME_DATE
**kwargs:
  keyword:value pairs that will search for similar
  objects in the database.

  e.g.:   name='m1%' --> WHERE name LIKE 'm1%'
"""
super(tt_list, self).__init__()
#Set some class values for reference
self.conn = conn
self.tables = tables

###########################################################
#Builds the list
curs = conn.cursor()
self.columns = []
self.cdict = {}
tables = tables.keys() #All tables
self.ptable = ptable

for table in tables:
  #POSTGRE DEPENDENT SQL
  SQL = "SELECT column_name FROM " \
    + "information_schema.columns " \
    + "WHERE table_name = '" + table \
    + "' ORDER BY ordinal_position;"
  curs.execute(SQL)

  for elem in curs:
    #Do not overwrite
    if elem[0] not in self.cdict.keys():
      self.cdict[elem[0]] = str(table)
      self.columns.append(elem[0])
    else:
        print "Warning: Duplicate keys found: " + elem[0]

    #Make sure we have "time" and "data" columns
    if("time" not in self.columns) \
        or ("data" not in self.columns):
      raise ValueError("No time or data columns in table")

    #####################################################
    # Build basic SQL statement to get all columns/tables
    sql_base = 'SELECT '
    for col in self.columns: #Use for ordering
      sql_base += '"%s"."%s", '% (self.cdict[col], col)
    sql_base = sql_base[0:-2] #remove last coma
    sql_base += " FROM "

    for table in tables:
      sql_base += '"%s", ' % str(table)

    sql_base = sql_base[0:-2] #remove last coma

    sql_base += " WHERE "
    for k, v in self.tables.iteritems():
      if k != self.ptable:
        sql_base += '"%s"."%s" = "%s"."%s" AND ' % \
          (self.ptable, v[1], k, v[2])

    #######################################################
    # *args and **kwargs logic
    for arg in args:
      if not isinstance(arg, str) :
        #We are passed an existing list of (hopefully) tt
        #objects, Integrate it in the class.
        if isinstance(arg, list):
          for obj in arg:
            self.append(obj)
        else:
          self.append(arg)

      else:
        #Otherwise, treat it as an SQL WHERE request.
        curs.execute(sql_base + arg)
        for obj in curs: self._cur2upvtt(obj)

    if len(kwargs) != 0:
      s = ''
      for k, v in kwargs.iteritems():
        if isinstance(v, str):
          s += "\"%s\".\"%s\" LIKE '%s' AND " % \
            (self.cdict[k], k, v)
        else:
          s += "\"%s\".\"%s\" LIKE %s AND " % \
            (self.cdict[k], k, v)

      SQL = '%s %s;' % (sql_base, s[0:-4])
      curs.execute(SQL)
      for obj in curs: self._cur2upvtt(obj)

  def _cur2upvtt(self, curfetch):
    """Helper function that sets a new tt entry in the self[id]
    storage of this class."""
    #Due to order in which items were asked, columns should be
    #ordered
    pkcol = self.columns.index(self.tables[self.ptable][0])

    #Convert these two columns (time, data) as arrays
    d = array(sql2lst(curfetch[self.columns.index("data")]))
    t = array(sql2lst(curfetch[self.columns.index("time")]))
    #Create the dictionary of all variables except time, data
    vdict = {}
    for i in range(0,len(self.columns)):
      if self.columns[i] not in ("time", "data"):
        vdict[self.columns[i]] = curfetch[i]

    self.append(tt(t, d, vdict))

  def __str__(self):
    return str(self.keys())

  def __repr__(self):
    return "<tt_list object at 0x%x with %d items>" % \
        (id(self), len(self))

  """ ++++++++++++++++++++++++++++++++++++++++
    PUBLIC CLASS FUNCTIONS
      ++++++++++++++++++++++++++++++++++++++++ """
  def toMatrix(self, order):
    """ Returns a matrix of all the data in the list in the format
      - Data of [0]
      - Data of [1]
      ...
      - Data of [n]

      Packages the data in a list of the format
      "x-axis(time), y_axis(order), data matrix"

      The data is sorted by "order" on the vertical axis
      and simply follows "time" on the horizontal axis.
    """
    try:
      y = self.col2vec(order)[0]
      x = self[0].time
    except:
      raise ValueError('Cannot create matrix of empty list')

    if (not isinstance(y, ndarray)):
      raise ValueError('Cannot Create Matrix out of non-arrays')

    y_orig = y.copy()
    y.sort()

    #TODO: Optimize where() statements for vectors?
    # Allocate
    mat = []
    for v in y:
      idx = where(y_orig == v)[0]
      if (len(idx) != 1):
        raise ValueError('Cannot proceed. Y Axis cannot have duplicate values')

      #Since "y" is sorted, matrix can be built from this order
      mat.append(self[idx].data)

    return (x,y, array(mat))

  def col2vec(self, *args):
    """ Returns the vector data(ndarray)
    of all the columns passed as arguments
    using all the traces inside this tt_list
    object.

      col2vec(self, *args)

    Ex:  col2vec(self, "time", "data", "name")
    """
    lst = []
    for arg in args:
```

```
    if arg in self.columns:
      lst.append([])
      for k in self:
        lst[len(lst)-1].append(k[arg])
    else:
      raise ValueError('Invalid Column Name')
  lst = array(lst) #Convert to array type
  return lst

def linePlots(self, order, ylab = '', maxPlots = 50, norm = False,xmin = 0 ,xmax = 0):
  """ Creates a multiple-line seismograph-like plot of all the time
  traces in the tt_list. They will be sorted by order.

  Tested to work well with LS-DYNA Simulations.

  NOTE: This is working well only for zero-centered signals. (Not necessarily zero mean)

  This is based off the MultilinePlots example in the matplotlib cookbook.
  http://www.scipy.org/Cookbook/Matplotlib/MultilinePlots
  """

  if ylab == '':
    ylab = order

  bulk = self.toMatrix(order)

  dat = bulk[2]
  l = len(dat)
  mod = 1

  if (l > maxPlots):
    mod = ceil(l/maxPlots)

  i = 0
  dat2 = []
  new_y = []
  for sig in dat:
    if not (i % mod):
      new_y.append(bulk[1][i])
      if (norm):
        sig = sig/sig.max()
      dat2.append(sig)
    i += 1

  dat = array(dat2)

  lineprops = dict(linewidth=1, color='black', linestyle='-')
  fig = figure() #Create figure
  ax = fig.add_axes([0.1, 0.1, 0.8, 0.8])

  scale = abs(dat[0]).max() #TO be changed

  if not (scale):
    scale = 1

  boxin = Bbox(Point(ax.viewLim.ll().x(), Value(-scale)),
        Point(ax.viewLim.ur().x(), Value(scale)))

  height = ax.bbox.ur().y() - ax.bbox.ll().y()

  boxout = Bbox(Point(ax.bbox.ll().x(), Value(-0.5)*height),
         Point(ax.bbox.ur().x(), Value( 0.5)*height))

  transOffset = get_bbox_transform(unit_bbox(), Bbox(
        Point ( Value(0), ax.bbox.ll().y()),
        Point ( Value(1), ax.bbox.ur().y())))

  ticklocs = []
  for i, s in enumerate(dat):
    trans = get_bbox_transform(boxin, boxout)
    offset = (i+1.)/(len(dat)+1.)
    trans.set_offset( (0, offset), transOffset)

    ax.plot(bulk[0], s, transform=trans, **lineprops)
    ticklocs.append(offset)

  ax.set_yticks(ticklocs)
  ax.set_yticklabels(new_y)

  all = []
  labels = []
  ax.set_yticks(ticklocs)
  for tick in ax.yaxis.get_major_ticks():
    all.extend(( tick.label1, tick.label2, tick.tick1line, tick.tick2line, tick.gridline))
    labels.append(tick.label1)

  setp(all, transform=ax.transAxes)
  setp(labels, x=-0.01)

  ax.set_xlabel('time (s)')
  ax.set_ylabel(ylab)

  def set_ygain(direction):
    set_ygain.scale += (direction)*(scale/5.)
    if set_ygain.scale <= 0:
      set_ygain.scale -= (direction)*(scale/5.)
      return

    for line in ax.lines:
      trans = line.get_transform()
      box1 = trans.get_bbox1()
      box1.intervaly().set_bounds(-set_ygain.scale, set_ygain.scale)

    draw()

  set_ygain.scale = scale

  def keypress(event):
    if event.key in ('+', '='): set_ygain(-1)
    elif event.key in ('-','_'): set_ygain(1)

  pylab.connect('key_press_event', keypress)
  ax.set_title('Use + / - to change y gain')

  if (xmax == 0):
    xmax = bulk[0].max()

  xlim([xmin,xmax])
  show()

def contourf(self,order, ylab = '', *args, **kwargs):
  """ Contour plot of a tt list data matrix. Similar
  to linePlots. Order describes how the matrix is ordered by
  in the Y direction, ylab gives the y label and
  *args and **kwargs are the standard matplotlib contourf. """
  dat = self.toMatrix(order)
  contourf(dat[0], dat[1], dat[2], *args, **kwargs)
  xlabel('Time (s)')
  ylabel(ylab)

def fkPlot(self, order, ylab = 'Wavenumber (1/m)', N = 100, scale=True):
  """ Produces a FK plot of a seismic trace provided an order.
  To be useful, the order must be evenly spaced traces in any
  directions so long as they are consistant.

  Example orders are x, y and possibly R for radial, if supported.
  """

  #Prepare the data.
  t,x,dat = self.toMatrix(order)
  l = len(dat)
  x = -x

  #perform optical transform to rearrange data for plotting
  for i in arange(dat.shape[0]):
    for j in arange(dat.shape[1]):
      if ((i+j)%2): #If i+j not even, take negative
        dat[i][j] = -dat[i][j]

  tdat = fft2(dat)
  #Compute nyquist vectors
  f_n = 1/(2.0*(t[1]-t[0]))
  k_n = 1/(2.0*(x[1]-x[0]))

  f = arange(0, dat.shape[1])/float(dat.shape[1]-1)*2*f_n - f_n
  k = arange(0, dat.shape[0])/float(dat.shape[0]-1)*2*k_n - k_n

  if scale:
    tdat = log10(1+ abs(tdat))

  #Add padding in k-dir ?

  contour(k, f, abs(tdat.T), N)

def save(self, tbl_list):
  """ Saves all the changes made to every upvtt instances of
  this object back into the database. This will only save
  elements that have been modified """

  curs = self.conn.cursor()
  for tts in self:
    #For every trace, INSERT OR UPDATE - Check pkey exists
    UPD = tts.updSQL(self.cdict, self.tables)
    INS = tts.insSQL(self.cdict, self.tables)
    for table in tbl_list:
      t = self.tables[table][0]
      SQL = "SELECT \"%s\" FROM %s WHERE \"%s\"='%s';" % \
        (t, table, t, tts[t])
      curs.execute(SQL)
      if len(curs.fetchall()):
        #If primary key existss, update
        if UPD[table] != '':
          curs.execute(UPD[table])
          self.conn.commit()
      else: #Insert
        curs.execute(INS[table])
        self.conn.commit()
      tts.clearSet()

def append(self, obj, bypass=False):
  """ Adds a tt object to the list """
  #If passed a list, call self on each object
  if isinstance(obj, tt) or bypass:
    super(tt_list, self).append(obj)
  else:
    raise TypeError('Cannot add non tt object to list')
```

Appendix C

Field Solver Classes

C.1 array.h

```
/*
 * Author : Simon Berube
 */
#ifndef __ARRAYS_H__
#define __ARRAYS_H__

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <fftw3.h>


#define SUCCESS 0
#define ERROR 1

/*
 * Floats
 */
// Floating Point Array Struct
typedef struct _ARRAYF {
  unsigned int size;
  float * p_data;
} ARRAYF, *PARRAYF;

// Floating Point Matrix Struct
typedef struct _MATRIXF {
  unsigned int size_x; // Across (Columns)
  unsigned int size_y; // Down (Rows)
  unsigned int size;
  float ** p_data;
} MATRIXF, *PMATRIXF;

// Utilities
ARRAYF toArrayf(float * p_data, unsigned int size);
int matcmp(PMATRIXF p_src, PMATRIXF p_dat, float * result);
int arrcmp(PARRAYF p_src, PARRAYF p_dat, float * result);
// Matlab Compatible Output
int loadAsciiMatrixf(char * filename, PMATRIXF p_data, unsigned int sx, unsigned int sy);
int loadAsciiArrayf(char * filename, PARRAYF p_data);
int saveAsciiMatrixf(char * filename, PMATRIXF p_data);
int saveAsciiArrayf(char * filename, PARRAYF p_data);

// Smaller, Ascii Output (May add compression later)
```

```
int saveArrayf(char * filename, PARRAYF p_data);
int saveMatrixf(char * filename, PMATRIXF p_data);
int loadInitArrayf(char * filename, PARRAYF p_raw);
int loadInitMatrixf(char * filename, PMATRIXF p_raw);


// Initializes/Delete memory for various object
int initArrayf(PARRAYF array, unsigned int size);
int initMatrixf(PMATRIXF matrix, unsigned int size_x, unsigned int size_y);
int delArrayf(PARRAYF array);
int delMatrixf(PMATRIXF matrix);

/*
 * Doubles
 */
// Double Array Struct
typedef struct _ARRAYD {
  unsigned int size;
  double * p_data;
} ARRAYD, *PARRAYD;


// Floating Point Matrix Struct
typedef struct _MATRIXD {
  unsigned int size_x; // Across (Columns)
  unsigned int size_y; // Down (Rows)
  unsigned int size;
  double ** p_data;
} MATRIXD, *PMATRIXD;


// Initializes/Delete memory for various object
int init_arrayd(PARRAYF array, unsigned int size);
int init_matrixd(PMATRIXF matrix, unsigned int size_x, unsigned int size_y);
int del_arrayd(PARRAYF array);
int del_matrixd(PMATRIXF matrix);

#endif
```

C.2 array.cc

```
#include "array.h"
/*
 * Utilities
 */

/*
 * Converts a float array of a known size into an
 * ARRAYF container. Does not copy, or allocate
 * memory but simply wraps existing data.
 *
 * This function should be used with care as the
 * data has been allocated somewhere else and is
 * most likely not reliable! Use only if you know
 * what you are doing.
 *
 * This was originally intended to convert a MATRIXF
 * row into a convenient ARRAYF for some computations.
 */
ARRAYF toArrayf(float* p_data, unsigned int size) {
  ARRAYF a;
  a.p_data = p_data;
  a.size = size;
  return a;
}
```

```
/*
 * Compares two matrixf matrices and compute their
 * total error. This function simply converts the
 * matrices into two vectors and uses arrcmp() to
 * compare them.
 *
 * Unlike arrcmp(), this method is ONLY VALID for
 * matrices of EXACTLY the same size. An error will
 * be returned otherwise.
 */
int matcmp(PMATRIXF p_src, PMATRIXF p_dat, float * result) {
  ARRAYF src, dat;

  if (p_src->size_x != p_dat->size_x || p_src->size_y != p_dat->size_y)
    return ERROR;

  src = toArrayf(p_src->p_data[0], p_src->size);
  dat = toArrayf(p_dat->p_data[0], p_dat->size);
  return arrcmp(&src,&dat,result);
}

/*
 * Compares two arrayf vectors and compute their total
 * error difference. Only the first N elements of p_dat
 * are checked where N is the size of the source vector.
 *
 * Returns error if the size of p_src is greater than
 * the size of p_dat.
 *
 * This has been tested.
 */
int arrcmp(PARRAYF p_src, PARRAYF p_dat, float * result) {
  float err = 0;

  if (p_src->size > p_dat->size)
    return ERROR;

  for (unsigned int i=0; i < p_src->size; i++)
    err += p_src->p_data[i] - p_dat->p_data[i];

  *result = err;
  return 0;
}

/*
 * Loads a ASCII file Matrix as saved by the command
 * "save file matrix -ASCII"
 * in Matlab
 *
 * Will load data in the column direction (x) until sx
 * numbers have been read. And then repeat for sy lines.
 *
 * This has been tested to be working
 */
int loadAsciiMatrixf(char * filename, PMATRIXF p_data, unsigned int sx, unsigned int sy) {
  FILE * fi;

  fi = fopen(filename, "r");
  if (fi == NULL)
    return ERROR;

  for( unsigned int i =0; i<sy; i++)
    for ( unsigned int j =0; j<sx; j++)
      fscanf(fi, "%e", &p_data->p_data[i][j]);

  fclose(fi);
  return 0;
}
```

```
/*
 * Savess a ASCII file Matrix as saved by the command
 * "save file matrix -ASCII"
 * in Matlab
 *
 * This has been tested to be working
 */
int saveAsciiMatrixf(char * filename, PMATRIXF p_data) {
  FILE * fi;

  fi = fopen(filename, "w");
  if (fi == NULL)
    return ERROR;

  for( unsigned int i =0; i<p_data->size_y; i++) {
    for ( unsigned int j =0; j<p_data->size_x; j++)
      fprintf(fi, "%16e", p_data->p_data[i][j]);

    fprintf(fi, "\n");
  }

  fclose(fi);
  return 0;
}


/*
 * Loads a ASCII file vector as saved by the command
 * "save file vector -ASCII"
 * in Matlab
 *
 * Will load data until either p_data is filled, or end
 * of file is reached.
 *
 * This has been tested to be working
 */
int loadAsciiArrayf(char * filename, PARRAYF p_data) {
  FILE * fi;
  unsigned int i = 0;

  fi = fopen(filename, "r");
  if (fi == NULL)
    return ERROR;

  while ( i < p_data->size && fscanf(fi, "%e", &p_data->p_data[i]) != EOF )
    i++;

  fclose(fi);
  return 0;
}


/*
 * Save a ASCII file vector as saved by the command
 * "save file vector -ASCII"
 * in Matlab
 *
 */
int saveAsciiArrayf(char * filename, PARRAYF p_data) {
  FILE * fi;
  unsigned int i = 0;

  fi = fopen(filename, "w");
  if (fi == NULL)
    return ERROR;

  for (i =0; i < p_data->size; i++)
```

```
    fprintf(fi, "%16e", p_data->p_data[i]);

  fclose(fi);
  return 0;
}


int saveArrayf(char * filename, PARRAYF p_data) {
  FILE * fi;

  fi = fopen(filename, "w");
  if (fi == NULL)
    return ERROR;
  fwrite(&p_data->size, sizeof(int), 1,  fi);
  fwrite(p_data->p_data, sizeof(float), p_data->size, fi );

  fclose(fi);
  return 0;

}


int saveMatrixf(char * filename, PMATRIXF p_data) {
  FILE * fi;

  fi = fopen(filename, "w");
  if (fi == NULL)
    return ERROR;

  fwrite(&p_data->size, sizeof(int), 1, fi);
  fwrite(&p_data->size_x, sizeof(int), 1, fi);
  fwrite(&p_data->size_y, sizeof(int), 1, fi);
  fwrite(p_data->p_data[0], sizeof(float), p_data->size, fi );

  fclose(fi);
  return 0;
}


int loadInitArrayf(char * filename, PARRAYF p_raw) {
  FILE * fi;
  unsigned int s;

  fi = fopen(filename, "r");
  if (fi == NULL)
    return ERROR;

  if (fread(&s, sizeof(int), 1, fi ) == 0) {
    fclose(fi);
    return ERROR;
  }

  initArrayf(p_raw, s);

  if (fread(p_raw->p_data, sizeof(float), p_raw->size, fi ) != p_raw->size) {
    delArrayf(p_raw);
    fclose(fi);
    return ERROR;
  }

  fclose(fi);
  return 0;
}


int loadInitMatrixf(char * filename, PMATRIXF p_raw) {
  FILE * fi;
  unsigned int sx, sy, s;
  fi = fopen(filename, "w");
  if (fi == NULL)
    return ERROR;

  if (fread(&s, sizeof(int), 1, fi ) == 0)
    return ERROR;

  if (fread(&sx, sizeof(int), 1, fi ) == 0)
   return ERROR;

  if (fread(&sy, sizeof(int), 1, fi ) == 0)
   return ERROR;

  initMatrixf(p_raw, sx, sy);

  if (fread(p_raw->p_data[0], sizeof(float), p_raw->size, fi ) != p_raw->size) {
    delMatrixf(p_raw);
    return ERROR;
  }

  fclose(fi);
  return 0;

}


/*
 * Init and Deletes
 */
int initArrayf(PARRAYF array, unsigned int size) {
  array->size = size;
  array->p_data = (float*)fftwf_malloc(size*sizeof(float));

  if (array->p_data == NULL) {
    array->size = 0;
    return 1;
  }

  return 0;
}

int initMatrixf(PMATRIXF matrix, unsigned int size_x, unsigned int size_y) {
  unsigned int i = 0;

  // Set Sizes
  matrix->size = size_x*size_y;
  matrix->size_x = size_x;
  matrix->size_y = size_y;

  // Allocate(Matrix is continuous memory block, start at p_data[0]
  matrix->p_data = (float**)fftwf_malloc(size_y*sizeof(float*));
  if (matrix->p_data == NULL)
    return ERROR;

  matrix->p_data[0] = (float*)fftwf_malloc(matrix->size*sizeof(float));
  if (matrix->p_data[0] == NULL)
    return ERROR;

  for (i=0; i<(size_y-1); i++)
    matrix->p_data[i+1] = matrix->p_data[0] + (i+1)*size_x;

  return 0;
}

int delArrayf(PARRAYF array) {
  fftwf_free(array->p_data);
  return true;
}

int delMatrixf(PMATRIXF matrix) {
  fftwf_free(matrix->p_data[0]); // This should free entire memory block
  return true;
}
```

C.3 circtd.h

```
/*
 * Author : Simon Berube
 */
#ifndef __CIRCTD_H__
#define __CIRCTD_H__

#define SIZE_TOO_LARGE 1
#define DT_TOO_BIG  2
#define PI 3.14159265

#include <math.h>
#include "array.h"
#include "transducer.h"
#include "impulse.h"

// Circular Transducer Container
class CircTD : public Transducer {
  public:
    CircTD(double radius, double x, double y, Impulse * imp)
     { m_size = radius; m_TDloc_x = x; m_TDloc_y = y; m_p_impulse = imp;} // -- Done

    int GetIR(double vel, double x, double y, unsigned int size, PARRAYF time, PARRAYF data); // -- Done
};

#endif
```

C.4 circtd.cc

```
#include "circtd.h"

int CircTD::GetIR( double vel, double x, double y, unsigned int size, PARRAYF time, PARRAYF data ) {
 float t_1, t_2, t_3;

 // Fix Coordinates (Make sure they are relative to transducer)
 x = fabs(x - m_TDloc_x);
 y = fabs(y - m_TDloc_y);

 // Make sure data vector large enough
 if (data->size < size || time->size < size)
 return SIZE_TOO_LARGE;

 // Compute constants
 t_1 = y/vel;
 t_2 = (1.0/vel)*pow(pow(y,2.0)+ pow(x-m_size, 2.0), 0.5);
 t_3 = (1.0/vel)*pow(pow(y,2.0)+ pow(x+m_size, 2.0), 0.5);

 // Fill the data vector
 for (unsigned int i = 0; i< size; i++) {
    if( x <= m_size) {
     if (time->p_data[i] < t_1)
       data->p_data[i] = 0.0;
      else if (time->p_data[i] <= t_2)
        data->p_data[i] = vel;
      else if (time->p_data[i] < t_3)
```

```
        data->p_data[i] = (vel/PI)*
 acos( (pow(vel, 2.) * pow(time->p_data[i], 2.) - pow(y, 2.) + pow(x, 2.) - pow(m_size, 2.))
 / (2.0*x* pow( pow(vel, 2.)* pow(time->p_data[i] , 2.) - pow(y, 2.), 0.5))  );
      else
       data->p_data[i] = 0;
 }
    else {
     if (time->p_data[i] < t_1)
       data->p_data[i] = 0;
      else if (time->p_data[i] <= t_2)
        data->p_data[i] = 0;
      else if (time->p_data[i] < t_3)
        data->p_data[i] = (vel/PI)*
 acos( (pow(vel, 2.) * pow(time->p_data[i], 2.) - pow(y, 2.) + pow(x, 2.) - pow(m_size, 2.))
 / (2.0*x* pow( pow(vel, 2.)* pow(time->p_data[i] , 2.) - pow(y, 2.), 0.5))  );
      else
        data->p_data[i] = 0;
 }
 }

 return 0;
}
```

C.5 impsine.h

```
#ifndef __IMPSINE_H__
#define __IMPSINE_H__

// TODO : Add self memory management to class as option. E.G. ImpSine(float,float, PARRAYF time) constructors.
#include <math.h>
#include <string.h>
#include "array.h"
#include "impulse.h"
#define PI 3.14159265


class ImpSine : public Impulse {
  private:
   void setup(float freq, float delay);

  public:
   ImpSine(PARRAYF data, float freq, float delay, float dt, float st, float et); // -- Done
   ImpSine(PARRAYF data, float freq, float delay, PARRAYF time); // -- Done
   ~ImpSine() {};

};


#endif
```

C.6 impsine.cc

```
#include "impsine.h"

ImpSine::ImpSine(PARRAYF data, float freq, float delay, float dt, float st, float et) :  Impulse(data, dt, st, et) {
  setup(freq, delay);
}

ImpSine::ImpSine(PARRAYF data, float freq, float delay, PARRAYF time) :
  Impulse(data, time->p_data[1] - time->p_data[0], time->p_data[0], time->p_data[time->size-1]) {
  setup(freq, delay);
}
```

```
void ImpSine::setup(float freq, float delay) {

  unsigned int i_dly, i_sz;

  i_dly = (int)floor(delay/m_dt);
  i_sz =  (int)floor((m_et-m_st)/m_dt);

  if (m_data->size < i_sz) {
    m_et = m_dt*(m_data->size - 1);
    i_sz = (int)floor((m_et-m_st)/m_dt);
  }

  memset(m_data->p_data, 0, m_data->size*sizeof(float)); // Zero all data first

  for (unsigned int i=i_dly; i< i_sz; i++)
    m_data->p_data[i] = sin((i*m_dt - delay)*freq*2*PI);
}
```

C.7 impulse.h

```
/*
 * Author : Simon Berube
 */
#ifndef __IMPULSE_H__
#define __IMPULSE_H__

#define SIZE_TOO_SMALL 1

#include <stdio.h>
#include "array.h"

/* Base Impulse class. Has public function GetData which
 * will return the vector of the impulse stored. This is
 * a basic impulse which has to be fully supplied by the
 * user.
 *
 * Functionning :
 *    Returns
 */
class Impulse {
  protected:
    PARRAYF m_data;
    float m_dt, m_st, m_et;

  public:
   Impulse(PARRAYF data, float dt, float st, float et); // -- Done
   Impulse(PARRAYF data, PARRAYF time); // -- Done
   ~Impulse() {};

   int GetDataCopy(PARRAYF data);
   PARRAYF GetDataPtr() { return m_data; } // -- Done

   float Get_dt() { return m_dt; } // -- Done
   float Get_Start_Time() { return m_st; } // -- Done
   float Get_End_Time() { return m_et; } // -- Done
};

#endif
```

C.8 impulse.cc

```
#include "impulse.h"

Impulse::Impulse(PARRAYF data, float dt, float st, float et) : m_data(data), m_dt(dt), m_st(st), m_et(et) {

}

Impulse::Impulse(PARRAYF data, PARRAYF time) : m_data(data) {
 m_st = time->p_data[0];
 m_dt = time->p_data[1] - time->p_data[0];
 m_et = time->p_data[time->size-1];
}

int Impulse::GetDataCopy(PARRAYF data) {
  if (data->size < m_data->size)
   return  SIZE_TOO_SMALL;

  memcpy(data->p_data, m_data->p_data, m_data->size * sizeof(float));
  return 0;
}
```

C.9 solver.h

```
#ifndef __SOLVER_H__
#define __SOLVER_H__

#define TD_LIST_SIZE_INITIAL 128
#define MODE_NORMAL 0
#define MODE_AMP 1

#include <math.h>

#include "array.h"
#include "impulse.h"
#include "transducer.h"

class Solver {
  private:
    // Convolution of x with y, saved in result.
    // x and y must be equal length such as to satisfy : x = [x_data + zeros(length of y - 1)]
    //                                                   y = [y_data + zeros(length of x - 1)]
    // Input data for x, y WILL BE DESTROYED by this function. However, results will be copied
    // in "results" memory location.
    // The length of result should be x.size + y.size - 1 (floats)
    int convf(fftwf_plan p_r2c_x, fftwf_plan p_r2c_y, fftwf_plan p_c2r, PARRAYF p_x, PARRAYF p_y, int N);
    int hilbert(int N, float*, PARRAYF, fftwf_plan , fftwf_plan );

  protected:
    // Transducer List In the Medium
    Transducer ** m_p_td_list;
    int m_td_list_sz;
    int m_td_list_sz_alloc;

    // Solver Settings
    PARRAYF m_p_time;
    MATRIXF m_results; // Matrix, for results
    bool m_isSolved;

    int m_size_x; // Discrete Resolution in x
    int m_size_y; // Discrete Resolution in y
    double m_dx; // dx step in x direction
    double m_dy; // dy step in y direction
    double m_vel;

    int m_mode;
  public:
    Solver(PARRAYF time, int res_x, int res_y, double dx, double dy, double vel); // -- Done
    ~Solver(); // -- Done

    // Transducer Functions
    int AddTD(Transducer * td); // Add Transducer to List -- DONE
    Transducer * GetTD(int id); // Return a Transducer (by ID) for modification (change delays, etc..) -- DONE
    int RemTD(int id);  // Remove a transducer from the list. -- DONE

    int Solve();
    int SaveToFile(char* file);

    int setSolutionMode(int);
    int getSolutionMode();

};

#endif
```

C.10 solver.cc

```
#include "solver.h"

Solver::Solver(PARRAYF time, int res_x, int res_y, double dx, double dy, double vel):
  m_p_time(time), m_size_x(res_x), m_size_y(res_y), m_dx(dx), m_dy(dy), m_vel(vel)
{
  // Init Mem
  m_p_td_list = (Transducer**)malloc(TD_LIST_SIZE_INITIAL * sizeof(Transducer*) );
  m_td_list_sz = 0;
  m_td_list_sz_alloc = TD_LIST_SIZE_INITIAL;

  m_isSolved = false;
  m_results.p_data = NULL;
  m_results.size = 0;

  m_mode = MODE_AMP;
}

Solver::~Solver() {
  free(m_p_td_list);
  if (m_isSolved)
    delMatrixf(&m_results);
}

int Solver::AddTD(Transducer * td) {
  // If memory full, allocate more space and delete old one.
  // The transducer Count is incremented here
  if (++m_td_list_sz >= m_td_list_sz_alloc) {
    //Increase amount of memory for td_list
    m_td_list_sz_alloc += TD_LIST_SIZE_INITIAL;
    Transducer ** tmp = (Transducer**)malloc(m_td_list_sz_alloc * sizeof(Transducer *) );
    memcpy(tmp, m_p_td_list, m_td_list_sz * sizeof(Transducer *));
    free(m_p_td_list); // Free old memory
    m_p_td_list = tmp; // Replace by new allocation
  }

  // Add Transducer To List
  m_p_td_list[m_td_list_sz-1] = td;
  return 0;
}


Transducer * Solver::GetTD(int id) {
  if (id < m_td_list_sz)
    return m_p_td_list[id];
  return NULL;
}

int Solver::RemTD(int id) {
  if (!(id<m_td_list_sz))
    return ERROR;

  memmove(&m_p_td_list[id], &m_p_td_list[id+1], (m_td_list_sz-id-1)*sizeof(Transducer *) ); // Shift memory over
  m_td_list_sz--;

  return 0;
}


int Solver::SaveToFile(char* file) {
  if (!m_isSolved)
    return ERROR;

  return saveAsciiMatrixf(file, &m_results);
}

int Solver::getSolutionMode() {
  return m_mode;
}

int Solver::setSolutionMode(int mode) {
  m_mode = mode;
  return SUCCESS;
}


int Solver::Solve() {
  ARRAYF h, H, imp, IMP, hbt;
  fftwf_plan p_r2c_h, p_r2c_imp, p_c2r, p_hilF = NULL, p_hilB = NULL;
  int sz, N;



  if (m_td_list_sz == 0)
    return ERROR; // No Transducer to compute

  if (m_isSolved)
    delMatrixf(&m_results);

  // Init
  /*
   * Sizes here are as follows. The size of the impulse should be of
   * I = 2^K+1
   * With the size of the Impulse Response being
   * H = 2^K or H = I -1
   * Such that, the size of the convolution which is C = H + I - 1
   * is
   * C = 2*I - 2 or C = 2^(K+1)
   * This ensures that all the FFTs will be performed on powers of 2
   * for faster computations.
   *
   * NOTE: Due to data padding needed for output transforms, the actual
   * size of arrays is 2*(K/2 + 1) or K + 2, which in our case gives
   * C = 2*I - 2 + 2 = 2*I
   */
  sz = m_p_time->size;
  N = 2*sz -2;

  if( initMatrixf(&m_results, N, m_size_x * m_size_y) )
    return ERROR;
  if ( initArrayf(&h, N) || initArrayf(&imp, N) || initArrayf(&H, N+2) || initArrayf(&IMP, N+2) )
    return ERROR;
```

```
// Compute FFT of vector h->H and imp->IMP. Then, given our convf works, take the resulting
// data in IMP (which is the result of the convolution in Frequency domain, and revert back
// to time domain.

p_r2c_h = fftwf_plan_dft_r2c_1d(N, h.p_data, (fftwf_complex*)(H.p_data), FFTW_MEASURE);
p_r2c_imp = fftwf_plan_dft_r2c_1d(N, imp.p_data, (fftwf_complex*)(IMP.p_data), FFTW_MEASURE);
p_c2r = fftwf_plan_dft_c2r_1d(N, (fftwf_complex*)(IMP.p_data), H.p_data, FFTW_MEASURE);


// If signal enveloppe is needed, we need two more FFT plans.
if (m_mode == MODE_AMP) {
  if (initArrayf(&hbt, 2*N))
    return ERROR;

  p_hilF = fftwf_plan_dft_1d(N, (fftwf_complex*)(hbt.p_data),
   (fftwf_complex*)(hbt.p_data), FFTW_FORWARD, FFTW_MEASURE);
  p_hilB = fftwf_plan_dft_1d(N, (fftwf_complex*)(hbt.p_data),
   (fftwf_complex*)(hbt.p_data), FFTW_BACKWARD, FFTW_MEASURE);
}


// Compute
for (int i=0; i < m_size_y; i++)
  for (int j=0; j < m_size_x; j++) {
    for (int k=0; k < m_td_list_sz; k++) {
      // Get h -- Zero h first
      memset(h.p_data, 0, h.size*sizeof(float)); // Zero
      m_p_td_list[k]->GetIR(m_vel, (float)(j*m_dx), (float)(i*m_dy), sz - 1, m_p_time, &h); // Copy impulse

      // Get imp -- Zero imp first
      memset(imp.p_data, 0, imp.size*sizeof(float)); // Zero
      m_p_td_list[k]->GetImpulseDataCopy(&imp); // Copy impulse

      // Compute
      convf(p_r2c_h, p_r2c_imp, p_c2r, &IMP, &H, N);

      // Copy result to matrix - or add to previous results if more then 1 td.
      if (k == 0)
        memcpy(m_results.p_data[i*m_size_y + j], H.p_data, N*sizeof(float));
      else
        for (int l=0; l<N; l++)
          m_results.p_data[i*m_size_y+j][l] += H.p_data[l];
    }

    // Once done with all of the transducers, transform into analytical signal, if desired.
    if (m_mode == MODE_AMP) {

      hilbert(N, m_results.p_data[i*m_size_y+j], &hbt, p_hilF, p_hilB);
      int N2 = N*N;

      // Normalize effect of FFTs. 4 FFTs performed on data here (2 FWD, 2 BCK)
      for (int k=0; k<N; k++)
        m_results.p_data[i*m_size_y+j][k] = m_results.p_data[i*m_size_y+j][k]/(float)(N2);
    } else {
      // Normalize FFTs done to data (1 FWD, 1 BCK)
      for (int k=0; k<N; k++)
        m_results.p_data[i*m_size_y+j][k] = m_results.p_data[i*m_size_y+j][k]/(float)N;
    }

}

m_isSolved = true;

delArrayf(&h);
delArrayf(&H);
delArrayf(&imp);
delArrayf(&IMP);
```

```
  if (m_mode == MODE_AMP)
    delArrayf(&hbt);
  return 0;
}

/*
 * Private/Protected
 */

/*
 * Convolution Algorithm using FFTWF for use ONLY by Solver. It's quite specialized.
 *
 * The two plans are run first and MUST be defined such that the result of each plan saves in either p_x
 * or p_y. Then, the resulting FFT data is multiplied and saved into p_x as a fftw_complex.
 *
 * Thus, the c2r plan MUST take p_x as a source and save it somewhere else. Where c2r saves is not important as
 * long as the source is p_x, which hold sthe results of the convolution in frequency domain.
 * TODO: Check to make sure FFT is normalized
 */
int Solver::convf(fftwf_plan p_r2c_x, fftwf_plan p_r2c_y, fftwf_plan p_c2r, PARRAYF p_x, PARRAYF p_y, int N) {
  int i = 0;
  float re, im;

  // Run the FFTs Forward, this will save data in itslef at p_x->data and p_y->data respectively. (in complex form)
  fftwf_execute(p_r2c_x);
  fftwf_execute(p_r2c_y);

  // Multiply two transforms
  for(i = 0; i<(N/2 + 1); i++) {
    re = ( p_x->p_data[2*i] * p_y->p_data[2*i]  -  p_x->p_data[2*i+1]*p_y->p_data[2*i+1] )/(float)N;
    im = ( p_x->p_data[2*i] * p_y->p_data[2*i+1]  +  p_x->p_data[2*i+1]*p_y->p_data[2*i] )/(float)N;

    p_x->p_data[2*i] = re;
    p_x->p_data[2*i+1] = im;
  }

  // Take the ifft, which gives us the convolution product
  fftwf_execute(p_c2r);

  return 0;
}


// Hilbert transform algorithm. Tested to be exact against Matlab's "hilbert()" function.
int Solver::hilbert(int N, float* p_in, PARRAYF p_work, fftwf_plan p_fwd, fftwf_plan p_bck) {

  if(p_work->size % 2 != 0)
    return ERROR;

  if(N*2 != (int)p_work->size)
    return ERROR;

  // 1) Copy the float array 'p_in' into the 'p_work' vector as complex data.
  // Zero the p_work vector
  memset(p_work->p_data, 0, p_work->size*sizeof(float));
  // Copy the input vec into p_work. Keep complex values zero.
  for(int i=0; i<N; i++)
    p_work->p_data[2*i] = p_in[i];

  // 2) Perform Forward FFT on p_work data to get the complex FFT of p_work.
  fftwf_execute(p_fwd);

  // 3) Kill all the negative frequencies
  memset(&(p_work->p_data[2*(N/2 + 1)]),0, 2*(N/2 - 1)*sizeof(float));

  // 4) Double all 'inside' frequencies. That is all positive frequencies except edges
  for(int i=1; i<N/2; i++) {
    p_work->p_data[2*i] = 2*p_work->p_data[2*i];
    p_work->p_data[2*i+1] = 2*p_work->p_data[2*i+1];
  }

  // 5) Reverse the FFT
  fftwf_execute(p_bck);

  // 6) Now, only store the magnitude information
  for(int i=0; i<N; i++)
    p_in[i] = sqrt(p_work->p_data[2*i]*p_work->p_data[2*i] + p_work->p_data[2*i+1]*p_work->p_data[2*i+1]);

  return 0;
}
```

C.11 transducer.h

```
/*
 * Author : Simon Berube
 */
#ifndef __TRANSDUCER_H__
#define __TRANSDUCER_H__

#include "array.h"
#include "impulse.h"

/*
 * Transducer Class, holds Time-Impulse and creates a custom
 * impulse response depending on type of transducer.
 *
 * This should be implemented
 * for each different type of transducers. This is meant to be expandable
 * to any future transducer types
 */
class Transducer {
  protected:
    Impulse * m_p_impulse;

    double m_TDloc_x;
    double m_TDloc_y;

    double m_size;

  public:
    virtual ~Transducer() {};

    virtual int GetIR(double vel, double x, double y, unsigned int size, PARRAYF time, PARRAYF data) = 0; // -- Done
    int GetImpulseDataCopy(PARRAYF data) { return m_p_impulse->GetDataCopy(data); } // -- DONE

    // Getter and Setters
    Impulse * GetImpulse() { return m_p_impulse; } // -- DONE
    double GetTDLoc_x() { return m_TDloc_x; } // -- DONE
    double GetTDLoc_y() { return m_TDloc_y; } // -- DONE
    double GetSize()    { return m_size; }  // -- DONE

    int SetImpulse_F(Impulse * p_impulse) { m_p_impulse = p_impulse; return 0; } // -- DONE
    int SetLoc(double x, double y) { m_TDloc_x = x; m_TDloc_y = y; return 0; } // -- DONE
    int SetSize(double size) { m_size = size; return 0; } // -- DONE
};

#endif
```

Appendix D

Solver User Interface

D.1 main.cc

```
#include <QApplication>
#include "pistonui.h"

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    PistonUi *ui = new PistonUi(); // Alloc

    ui->show();
    return app.exec();
}
```

D.2 pistonui.h

```
#ifndef __PISTONUI_H__
#define __PISTONUI_H__
// C
#include <stdio.h>
// Qt
#include <QStringListModel>
#include <QtGui>
// Qwt
#include <qwt_plot.h>
#include <qwt_plot_marker.h>
#include <qwt_plot_curve.h>
#include <qwt_legend.h>
#include <qwt_data.h>
#include <qwt_text.h>
#include <qwt_plot_spectrogram.h>
#include <qwt_color_map.h>
#include <qwt_scale_widget.h>
#include <qwt_scale_draw.h>
#include <qwt_plot_zoomer.h>
#include <qwt_plot_panner.h>
#include <qwt_plot_layout.h>
#include <qwt_plot_picker.h>
// Piston-ir
#include <array.h>
#include <circtd.h>
#include <impsine.h>
#include <impgaus.h>
// Local
```

```
#include "ui_pistonui.h"
#include "td.h"
#include "tdimport.h"
#include "types.h"
#include "plotsolver.h"

// Fancy Zoomer Class extending Default QwtPlotZoomer
class MyZoomer: public QwtPlotZoomer
{
public:
    MyZoomer(QwtPlotCanvas* canvas):
        QwtPlotZoomer(canvas)
    {
        setTrackerMode(QwtPicker::AlwaysOn);
    }

protected:
    virtual QwtText trackerText( const QwtDoublePoint& p ) const
    {
        QwtText t( QwtPlotPicker::trackerText( p ));

        QColor c(Qt::white);
        c.setAlpha(180);
        t.setBackgroundBrush( QBrush(c) );
        return t;
    }
};


// Main Class
class PistonUi: public QMainWindow {

  Q_OBJECT


  public:
    PistonUi();
    ~PistonUi();

  public slots:
    // Transducer Slots
    void td_add();
    void td_added(); // Will be called when td->td_add is clicked()
    void td_selected(const QModelIndex & index );
    void td_rem();
    void td_clear();
    void td_import();
    void td_imported();

    // Radio Buttons
    void impulse();
    void response();
    void locEdited();

    void modeNormal();
    void modeAmp();


    // Viewing Slots
    void solve();
    void tslider(int value);
    void islider(int value);
    void spectrogram(bool checked);
    void contours(bool checked);
    void maxAmpMode();
    void timeMode();
    void pickerPick(const QwtDoublePoint &);
  protected:
    // Ui Components
    Ui::MainWindow ui;
    TD *tdui;
    TDIMP *tdimpui;
    QStringListModel *tdListModel;
    QStringList tdStringList;

    // Computation Components
    int m_mode;
    ARRAYF **impData; // Struct
    ARRAYF m_time; // Struct

    PlotSolver *plotSolver; // Class
    Transducer **tds; // Class
    Impulse **imps; // Class


    int m_tds_count;
    bool m_isSolved;

    // Plot Components
    QwtPlotCurve * m_p_td_curve;
    ARRAYF   m_tempIR;
    double * m_p_x_data; // on delete
    double * m_p_y_data; // on delete
    int      m_current_index;
    int      m_plot_data_size;

    // Other plot components
    QwtPlotCurve * m_p_tCurve;
    QwtPlotCurve * m_p_hCurve;
    QwtPlotCurve * m_p_vCurve;

    double * m_p_vxData;
    double * m_p_vyData;
    int      m_vDataSize;
    double * m_p_hxData;
    double * m_p_hyData;
    int      m_hDataSize;
    double * m_p_txData;
    double * m_p_tyData;
    int      m_tDataSize;

    // Raster Plot Components
    QwtPlotSpectrogram * m_p_spec;
    MyZoomer           * m_p_zoom;
    QwtPlotPanner      * m_p_panner;
    QwtPicker          * m_p_picker;
    bool m_rasterSet;

    // Slider Variables
    double m_maxIntensity;
    bool m_isNegative;

  private:
    void setupRasterPlot();
    void clearSolver();
    void addTransducer(float size, float x, float y, float dly, float f, float gwidth,
                        int imp_type, int td_type);

};


#endif
```

D.3 pistonui.cc

```
#include "pistonui.h"


PistonUi::PistonUi() {
  // TODO: Make Table View Non-Editable
  // TODO: Test Import Impulse
  // TODO: FIX ImpSine delay times
  // TODO: FIX PlotSolver::Copy() to be proper (modify solver to know if it is a copy or original)
  // TODO: Fix Density in Solver and divide by 2Pi, make sure it is real "pressure" value
  // TODO: Implement "remove"
  // TODO: Replace LocX, LocY by sliders
  // TODO: Add Cartesian to Polar display
  // TODO: Max Amplitude Display
  // TODO: Create a SOLVER_MEM class that manages memory by itself. Redo this class without memory crap. Bad Design
  // TODO: Store all parameters resx,resy, dx,dy, etc... after solving to prevent checking them again
  // TODO: Describe Unit Consistency when using .LAW file import.
  // TODO: Add Combo Box for #Time steps
  // TODO: When memSolver done, make it able to change dt, # time steps on the fly

  ui.setupUi(this);
  tdui = new TD();
  tdimpui = new TDIMP();
  tdListModel = new QStringListModel(this);

  // Set up default Values
  ui.stp_list->setModel(tdListModel);
  ui.stp_dx->setText(DEFAULT_DX);
  ui.stp_dy->setText(DEFAULT_DY);
  ui.stp_maxx->setText(DEFAULT_SX);
  ui.stp_maxy->setText(DEFAULT_SY);
  ui.stp_vel->setText(DEFAULT_VEL);
  ui.stp_dt->setText(DEFAULT_DT);
  ui.stp_size->setText(DEFAULT_LT);
  ui.stp_X->setText(DEFAULT_X);
  ui.stp_Y->setText(DEFAULT_Y);
  m_mode = MODE_NORMAL;

  // Set Button Connections
  connect(ui.stp_add, SIGNAL(clicked()), this, SLOT(td_add()));
  connect(tdui->ui.add, SIGNAL(clicked()), this, SLOT(td_added()));
  connect(tdimpui->ui.accept, SIGNAL(clicked()), this, SLOT(td_imported()));
  connect(ui.stp_rem, SIGNAL(clicked()), this, SLOT(td_rem()));
  connect(ui.stp_clear, SIGNAL(clicked()), this, SLOT(td_clear()));
  connect(ui.stp_import, SIGNAL(clicked()), this, SLOT(td_import()));
  connect(ui.stp_solve, SIGNAL(clicked()), this, SLOT(solve()));
  connect(ui.stp_list, SIGNAL(pressed(const QModelIndex&)), this, SLOT(td_selected(const QModelIndex &)));
  connect(ui.stp_impulse, SIGNAL(clicked()), this, SLOT(impulse()));
  connect(ui.stp_response, SIGNAL(clicked()), this, SLOT(response()));
  connect(ui.view_spec, SIGNAL(clicked(bool)), this, SLOT(spectrogram(bool)));
  connect(ui.view_cont, SIGNAL(clicked(bool)), this, SLOT(contours(bool)));
  connect(ui.stp_X, SIGNAL(editingFinished()), this, SLOT(locEdited()));
  connect(ui.stp_Y, SIGNAL(editingFinished()), this, SLOT(locEdited()));
  connect(ui.view_maxAmp, SIGNAL(clicked()), this, SLOT(maxAmpMode()));
  connect(ui.view_time, SIGNAL(clicked()), this, SLOT(timeMode()));
  connect(ui.stp_Normal, SIGNAL(clicked()), this, SLOT(modeNormal()));
  connect(ui.stp_Amp, SIGNAL(clicked()), this, SLOT(modeAmp()));



  // Sliders
  connect(ui.view_slider, SIGNAL(valueChanged(int)), this, SLOT(tslider(int)));
  connect(ui.view_intensity, SIGNAL(valueChanged(int)), this, SLOT(islider(int)));
  ui.view_slider->setEnabled(false);
  ui.view_intensity->setEnabled(false);
  ui.stp_rem->setEnabled(false);
  // Allocate memory for Transducers and Impulses and Initialize Values
  tds = (Transducer**)malloc(INITIAL_NB_TD_ALLOC*sizeof(Transducer*));

  impData = (ARRAYF**)malloc(INITIAL_NB_TD_ALLOC*sizeof(PARRAYF));
  for (int i = 0; i< INITIAL_NB_TD_ALLOC; i++) {
    impData[i] = (PARRAYF)malloc(sizeof(ARRAYF));
    impData[i]->size = 0;
    impData[i]->p_data = NULL;
  }

  imps = (Impulse **)malloc(INITIAL_NB_TD_ALLOC*sizeof(Impulse*));

  m_tds_count = 0;
  m_time.size = 0;
  m_time.p_data = NULL;
  plotSolver = NULL;
  m_isSolved = false;


  // Plot Stuff
  ui.stp_plot->setCanvasBackground(Qt::white);
  ui.stp_plot->setAxisTitle(ui.stp_plot->xBottom, "Time");
  ui.stp_plot->setAxisTitle(ui.stp_plot->yLeft, "Velocity");
  ui.stp_plot->setTitle("Transducer Impulse");

  m_vDataSize = 0;
  m_hDataSize = 0;
  m_tDataSize = 0;
  m_p_hxData = NULL;
  m_p_hyData = NULL;
  m_p_vxData = NULL;
  m_p_vyData = NULL;
  m_p_txData = NULL;
  m_p_tyData = NULL;

  m_tempIR.size = 0;
  m_tempIR.p_data = NULL;
  m_p_td_curve = new QwtPlotCurve("Transducer Impulse");
  m_p_tCurve = new QwtPlotCurve("Time Values");
  m_p_hCurve = new QwtPlotCurve("Horizontal Values");
  m_p_vCurve = new QwtPlotCurve("Vertical Values");
  m_p_x_data = NULL;
  m_p_y_data = NULL;
  m_plot_data_size=0;
  m_current_index=-1;
  m_rasterSet = false;
  m_p_spec = NULL;
  m_p_zoom = NULL;
  m_p_panner = NULL;
  m_p_picker = NULL;
}

// Free memory
PistonUi::~PistonUi() {
  tdui->close();
  delete tdui;
  delete tdListModel;
  if (plotSolver != NULL) delete plotSolver;
  delete m_p_td_curve;
  delete m_p_x_data;
  delete m_p_y_data;
  delete m_p_tCurve;
  delete m_p_hCurve;
  delete m_p_vCurve;

  if (m_p_hxData != NULL) delete m_p_hxData;
  if (m_p_hyData != NULL) delete m_p_hyData;
  if (m_p_vxData != NULL) delete m_p_vxData;
  if (m_p_vyData != NULL) delete m_p_vyData;
  if (m_p_txData != NULL) delete m_p_txData;
  if (m_p_tyData != NULL) delete m_p_tyData;

  // Delete Plot Data if set
  if (m_rasterSet) {
    delete m_p_spec;
    delete m_p_zoom;
    delete m_p_panner;
    delete m_p_picker;
  }

  if (m_time.size != 0)
    delArrayf(&m_time);

  for (int i=0; i<m_tds_count; i++) {
      delete tds[i];
      delArrayf(impData[i]);
      delete imps[i];
  }

  for (int i=0; i<INITIAL_NB_TD_ALLOC; i++)
    free(impData[i]);

  free(tds);
  free(impData);
  free(imps);
  delArrayf(&m_tempIR);
}

void PistonUi::modeNormal() {
  m_mode = MODE_NORMAL;
}

void PistonUi::modeAmp() {
  m_mode = MODE_AMP;
}

void PistonUi::td_add() {
  tdui->exec();
}

void PistonUi::maxAmpMode() {
  if (m_isSolved) {
    m_isNegative = false;
    ui.view_slider->setEnabled(false);
    plotSolver->setMode(PlotSolver::ModeAmplitude);
      ui.view_plot->replot();
  }
}

void PistonUi::timeMode() {
  if (m_isSolved) {
    // TODO: Reset colobar to original range
    m_isNegative = m_mode == MODE_NORMAL;
    ui.view_slider->setEnabled(true);
      plotSolver->setMode(PlotSolver::ModeNormal);
      ui.view_plot->replot();
  }

}


/*
 * This function takes information from a tdEntry
 * input dialog and converts it to a Transducer
 * class. This class is then added to the current
 * solver.
 */
void PistonUi::td_added() {
  float size, dly, x, y,  f, gw;
  bool ok;

  // Transducer Properties Loading
  size = tdui->ui.td_sz->text().toFloat(&ok);
  if(!ok) {
    QMessageBox::critical(tdui, "Error", "Invalid Size");
    return;
  }

  x = tdui->ui.td_x->text().toFloat(&ok);
  if(!ok) {
    QMessageBox::critical(tdui, "Error", "Invalid Location X");
    return;
  }

  y = tdui->ui.td_y->text().toFloat(&ok);
  if(!ok) {
    QMessageBox::critical(tdui, "Error", "Invalid Location Y");
    return;
  }

  dly = tdui->ui.sig_dly->text().toFloat(&ok);
  if(!ok) {
     QMessageBox::critical(tdui, "Error", "Invalid Delay Time");
     return;
  }

  f = tdui->ui.sig_f->text().toFloat(&ok);
  if(!ok) {
    QMessageBox::critical(tdui, "Error", "Invalid Frequency");
    return;
  }

  gw = tdui->ui.sig_gwidth->text().toFloat(&ok);
  if(!ok) {
    QMessageBox::critical(tdui, "Error", "Invalid Gaussian Width");
    return;
  }

  addTransducer(size,x,y,dly,f,gw, tdui->ui.sig_type->currentIndex(), tdui->ui.td_type->currentIndex());
}

void PistonUi::td_rem() {
}


void PistonUi::td_import() {
  tdimpui->exec();
}

void PistonUi::td_imported() {
  if (tdimpui->ready()) {
    for (int i=0; i< tdimpui->nTDs(); i++) {
      addTransducer(tdimpui->sz(), tdimpui->x(i), 0, tdimpui->del() + tdimpui->tDelay(i),
                    tdimpui->freq(), (float)tdimpui->nCycles()/(float)tdimpui->freq(),
                    tdimpui->ui.sig_type->currentIndex(), tdimpui->ui.td_type->currentIndex());
    }
  }
}


void PistonUi::addTransducer(float size, float x, float y, float dly, float f, float gwidth, int imp_type, int td_type) {
  char buf[256];
  bool ok;
```

```
QString fn;


// If the first transducer is added create time vector
if ( m_tds_count == 0) {
  if (m_time.size != 0)
    delArrayf(&m_time); // Delete if previously set

  int i_size = ui.stp_size->text().toInt(&ok, 0);
  if(!ok) {
    QMessageBox::critical(this, "Error", "Invalid Time Vector Size");
    return;
  }

  float dt = ui.stp_dt->text().toFloat(&ok);
  if(!ok) {
    QMessageBox::critical(this, "Error", "Invalid dt");
    return;
  }

  initArrayf(&m_time, i_size);

  // Set the time vector
  for (int i=0; i< i_size; i++ ) {
    m_time.p_data[i] = i*dt;
  }
}

if ( m_tds_count >= INITIAL_NB_TD_ALLOC) {
  QMessageBox::critical(tdui, "Error", "Maximum Number of Transducers Reached");
  return;
}

// Create Impulse
switch(imp_type) {
  case SIGNAL_SINE: if( initArrayf(impData[m_tds_count], m_time.size) ) {
                      QMessageBox::critical(tdui, "Error", "Could not Allocate Memory for Impulse");
                      return;
                    }

                    imps[m_tds_count] = new ImpSine(impData[m_tds_count], f, dly, &m_time);
                    break;

  case SIGNAL_GAUSSIAN_SINE: if ( initArrayf(impData[m_tds_count], m_time.size)) {
                               QMessageBox::critical(tdui, "Error", "Could not Allocate Memory for Impulse");
                               return;
                             }

                             imps[m_tds_count] = new ImpGaus(impData[m_tds_count], f, dly, gwidth, &m_time);
                             break;


  case SIGNAL_CUSTOM: fn = QFileDialog::getOpenFileName(tdui, "Open Saved Array","./", "All Files (*.*)");
                      if (fn.isEmpty()) // If choice cancelled, exit
                        return;

                      // Load and check if loading works.
                      if(loadInitArrayf(fn.toAscii().data(), impData[m_tds_count]) == ERROR) {
                        QMessageBox::critical(tdui, "Error", "Could not load impulse file");
                        return;
                      }

                      imps[m_tds_count] = new Impulse(impData[m_tds_count-1], &m_time);
                      break;
}

// Now create the transducer
switch(td_type) {
    case TYPE_CIRCULAR: tds[m_tds_count] = new CircTD(size,x,y,imps[m_tds_count]);
                        sprintf(buf, "CircTD, Size: %.2f, X:%.2f, Y:%.2f", size, x, y);
                        break;
  }

  m_tds_count++;

  // Disable the time slot
  if (m_tds_count == 1) {
    ui.stp_size->setEnabled(false);
    ui.stp_dt->setEnabled(false);
  }

  // Now add the information to the List.
  tdStringList << buf;
  tdListModel->setStringList(tdStringList);

}

void PistonUi::td_clear() {
  clearSolver();


  // If time is already set, reset it.
  if (m_time.size) {
    delArrayf(&m_time);
    // Disable the time slot
      ui.stp_size->setEnabled(true);
      ui.stp_dt->setEnabled(true);
  }

  tdStringList.clear();

  // Clear memory for transducers
    for (int i=0; i<m_tds_count; i++) {
        delete tds[i];
        delArrayf(impData[i]);
        delete imps[i];
    }

  m_tds_count = 0;
  tdListModel->setStringList(tdStringList); // Remove transducer list

  // Disable Sliders
  ui.view_slider->setEnabled(false);
  ui.view_intensity->setEnabled(false);


}

void PistonUi::td_selected(const QModelIndex & index ) {
    char buf[64];
    int row = index.row();

    if (row >= m_tds_count || row < 0 )
      return;

    if(m_plot_data_size == 0) { // If first plot
      m_p_td_curve->setRenderHint(QwtPlotItem::RenderAntialiased);
      m_p_td_curve->setPen(QPen(Qt::red));
      m_p_td_curve->attach(ui.stp_plot);
    }

    if (m_plot_data_size != (int)m_time.size) {
      if (m_plot_data_size != 0) {// If this is not first initialization, delete old first
        delete m_p_x_data;
        delete m_p_y_data;
      }
      m_p_x_data = new double[m_time.size];
      m_p_y_data = new double[m_time.size];
      m_plot_data_size = m_time.size;

    }

      // If we wanted to plot the impulse
      if (ui.stp_impulse->isChecked()) {
        sprintf(buf, "Transducer Impulse %i", index.row());
        ui.stp_plot->setTitle(buf);

        // Update the data
        for (unsigned int i=0; i<m_time.size; i++) {
          m_p_x_data[i] = m_time.p_data[i];
          m_p_y_data[i] = impData[row]->p_data[i];
        }
      }
      // Or, if we wanted to plot the response to the impulse
      else if (ui.stp_response->isChecked()) {
        double x,y,vel;
        bool ok;

        if (m_tempIR.size != m_time.size)
          initArrayf(&m_tempIR, m_time.size);

        // Read values from form

        x = ui.stp_X->text().toFloat(&ok);
        if(!ok) {
          QMessageBox::critical(tdui, "Error", "Invalid X Location for Trace");
          return;
        }

        y = ui.stp_Y->text().toFloat(&ok);
        if(!ok) {
          QMessageBox::critical(tdui, "Error", "Invalid Y Location for Trace");
          return;
        }

        vel = ui.stp_vel->text().toFloat(&ok);
        if(!ok) {
          QMessageBox::critical(tdui, "Error", "Invalid Velocity");
          return;
        }

        tds[row]->GetIR( vel, x, y, m_time.size, &m_time, &m_tempIR);

        // Update the data (Float to double, cannot memcpy)
        for (unsigned int i=0; i<m_time.size; i++) {
          m_p_x_data[i] = m_time.p_data[i];
          m_p_y_data[i] = m_tempIR.p_data[i];
        }
      }

      m_current_index = row;
      m_p_td_curve->setData(m_p_x_data, m_p_y_data, m_plot_data_size);

    ui.stp_plot->replot();
}


void PistonUi::impulse() {
  if (m_tds_count) {
    ui.stp_response->setChecked(false);
    td_selected(ui.stp_list->currentIndex());
  }
}
```

```
void PistonUi::response() {
  if (m_tds_count) {
    ui.stp_impulse->setChecked(false);
    td_selected(ui.stp_list->currentIndex());
  }
}

void PistonUi::locEdited() {
  td_selected(ui.stp_list->currentIndex());
}


void PistonUi::solve() {
  int res_x, res_y;
  bool ok;
  double dx,dy,vel;

  // Get Variables
  if (m_tds_count == 0) {
    QMessageBox::critical(this, "Error", "Nothing to solve");
    return;
  }

  res_x = ui.stp_maxx->text().toInt(&ok, 0);
  if(!ok) {
    QMessageBox::critical(tdui, "Error", "Invalid Max x");
    return;
  }

  res_y = ui.stp_maxy->text().toInt(&ok, 0);
  if(!ok) {
    QMessageBox::critical(tdui, "Error", "Invalid Max y");
  }

  dx = ui.stp_dx->text().toFloat(&ok);
  if(!ok) {
    QMessageBox::critical(tdui, "Error", "Invalid dx");
    return;
  }

  dy = ui.stp_dy->text().toFloat(&ok);
  if(!ok) {
    QMessageBox::critical(tdui, "Error", "Invalid dy");
    return;
  }

  vel = ui.stp_vel->text().toFloat(&ok);
  if(!ok) {
    QMessageBox::critical(tdui, "Error", "Invalid Medium Velocity");
    return;
  }

  if (m_isSolved)
  clearSolver();


  // Create Solver
  plotSolver = new PlotSolver(&m_time, res_x, res_y, dx, dy, vel);

  for (int i=0; i<m_tds_count; i++)
    plotSolver->AddTD(tds[i]);

  // Solve
  plotSolver->setSolutionMode(m_mode); // Set solution mode to solve
  if( plotSolver->Solve() )
    QMessageBox::critical(this, "Error", "Could not allocate memory to solve");
  // Set Data  and Sliders
  ui.view_slider->setEnabled(true);
  ui.view_intensity->setEnabled(true);

  plotSolver->setTimeIndex(0);


  ui.view_slider->setRange(0, plotSolver->getMaxTimeIndex());

  m_maxIntensity = plotSolver->maxValue()*1.05;
  m_isNegative = m_mode == MODE_NORMAL;

  // Plot
  setupRasterPlot();
  islider(ui.view_intensity->maximum()); // Activate slider max (to set scale correctly)
  m_isSolved = true;


}


void PistonUi::tslider(int value) {
 if (!m_rasterSet || !m_isSolved)  // Plot not set, nothing to do
   return;

 char buf[32];
 sprintf(buf, "Time : %fs", plotSolver->getSetTime());

 plotSolver->setTimeIndex(value);
 ui.statusbar->showMessage(buf);
 ui.view_plot->replot();
}

void PistonUi::islider(int value) {
 if (!m_rasterSet || !m_isSolved)  // Plot not set, nothing to do
   return;

 int max;
 max = ui.view_intensity->maximum();

 if (m_isNegative)
   plotSolver->setRange(-value*(m_maxIntensity/(float)max),
                        value*(m_maxIntensity/(float)max));
 else
   plotSolver->setRange(0, value*(m_maxIntensity/(float)max));


 ui.view_plot->axisWidget(QwtPlot::yRight)->setColorMap(m_p_spec->data().range(), m_p_spec->colorMap());
 ui.view_plot->setAxisScale(QwtPlot::yRight,
      m_p_spec->data().range().minValue(),
      m_p_spec->data().range().maxValue() );


 m_p_spec->setContourLevels(plotSolver->getContours(DEFAULT_NB_CONTOURS));

 ui.view_plot->replot();
}

void PistonUi::clearSolver(){
  if (m_isSolved) {
      if (m_rasterSet) {
        if(m_p_panner != NULL)  delete m_p_panner;
        if(m_p_zoom != NULL) delete m_p_zoom;
        if(m_p_spec != NULL) delete m_p_spec;
        if(m_p_picker != NULL) delete m_p_picker;

        // Line Plots
        m_vDataSize = 0;
        m_hDataSize = 0;
        m_tDataSize = 0;

        if (m_p_hxData != NULL) delete m_p_hxData;
        if (m_p_hyData != NULL) delete m_p_hyData;
        if (m_p_vxData != NULL) delete m_p_vxData;
        if (m_p_vyData != NULL) delete m_p_vyData;
        if (m_p_txData != NULL) delete m_p_txData;
        if (m_p_tyData != NULL) delete m_p_tyData;

        m_p_hxData = NULL;
        m_p_hyData = NULL;
        m_p_vxData = NULL;
        m_p_vyData = NULL;
        m_p_txData = NULL;
        m_p_tyData = NULL;
      }
      m_rasterSet = false;
      m_isSolved = false;
    }
}

void PistonUi::contours(bool checked) {
  if(m_p_spec == NULL)
    return;

  m_p_spec->setDisplayMode(QwtPlotSpectrogram::ContourMode, checked);
  ui.view_plot->replot();
}

void PistonUi::spectrogram(bool checked) {
  if (m_p_spec == NULL)
    return;

  m_p_spec->setDisplayMode(QwtPlotSpectrogram::ImageMode, checked);
  m_p_spec->setDefaultContourPen(checked ? QPen() : QPen(Qt::NoPen));
  ui.view_plot->replot();
}

void PistonUi::pickerPick(const QwtDoublePoint & pt) {

  plotSolver->getXAxis(m_hDataSize, m_p_hxData);
  plotSolver->getYAxis(m_vDataSize, m_p_vxData);
  plotSolver->getTAxis(m_tDataSize, m_p_txData);

  plotSolver->getXData(pt.y(), m_hDataSize, m_p_hyData);
  plotSolver->getYData(pt.x(), m_vDataSize, m_p_vyData);
  plotSolver->getTData(pt.x(), pt.y(), m_tDataSize, m_p_tyData);

  // Update all the data sets
  m_p_tCurve->setData(m_p_txData, m_p_tyData, m_tDataSize);
  m_p_hCurve->setData(m_p_hxData, m_p_hyData, m_hDataSize);
  m_p_vCurve->setData(m_p_vxData, m_p_vyData, m_vDataSize);

  ui.view_horizPlot->replot();
  ui.view_vertPlot->replot();
  ui.view_timePlot->replot();
}

void PistonUi::setupRasterPlot() {
  int res_x, res_y, res_t;
  bool ok;

  if (!m_rasterSet) {
    // If not initialized
    m_p_spec = new QwtPlotSpectrogram();
    m_p_zoom = new MyZoomer(ui.view_plot->canvas());
    m_p_panner = new QwtPlotPanner(ui.view_plot->canvas());
    m_p_picker = new QwtPlotPicker(ui.view_plot->canvas());

    res_x = ui.stp_maxx->text().toInt(&ok, 0);
    if(!ok) {
      QMessageBox::critical(tdui, "Error", "Invalid Max x");
      return;
    }

    res_y = ui.stp_maxy->text().toInt(&ok, 0);
    if(!ok) {
      QMessageBox::critical(tdui, "Error", "Invalid Max y");
    }

    res_t = ui.stp_maxy->text().toInt(&ok, 0);
    if(!ok) {
      QMessageBox::critical(tdui, "Error", "Invalid time");
    }

    m_p_hxData = new double[res_x];
    m_p_hyData = new double[res_x];
    m_p_vxData = new double[res_y];
    m_p_vyData = new double[res_y];
    m_p_txData = new double[plotSolver->getMaxTimeIndex()];
    m_p_tyData = new double[plotSolver->getMaxTimeIndex()];

    m_hDataSize = res_x;
    m_vDataSize = res_y;
    m_tDataSize = plotSolver->getMaxTimeIndex();

    m_p_tCurve->setRenderHint(QwtPlotItem::RenderAntialiased);
    m_p_tCurve->setPen(QPen(Qt::blue));
    m_p_tCurve->attach(ui.view_timePlot);

    m_p_hCurve->setRenderHint(QwtPlotItem::RenderAntialiased);
    m_p_hCurve->setPen(QPen(Qt::green));
    m_p_hCurve->attach(ui.view_horizPlot);

    m_p_vCurve->setRenderHint(QwtPlotItem::RenderAntialiased);
    m_p_vCurve->setPen(QPen(Qt::black));
    m_p_vCurve->attach(ui.view_vertPlot);
  }

  // Setup Picker
  m_p_picker->setTrackerMode(QwtPicker::AlwaysOff);
  m_p_picker->setSelectionFlags(QwtPicker::PointSelection);
  connect(m_p_picker, SIGNAL(selected(const QwtDoublePoint &)), this, SLOT(pickerPick(const QwtDoublePoint &)));



  // Setup Colormap
  QwtLinearColorMap colorMap(Qt::darkCyan, Qt::red);
  colorMap.addColorStop(0.0, Qt::darkBlue);
  colorMap.addColorStop(0.01, Qt::darkBlue);
  colorMap.addColorStop(0.4, Qt::cyan);
  colorMap.addColorStop(0.5, Qt::green);
  colorMap.addColorStop(0.6, Qt::yellow);
  colorMap.addColorStop(0.8, Qt::red);
  colorMap.addColorStop(0.99, Qt::darkRed);
  colorMap.addColorStop(1.0, Qt::darkRed);

  // Set color map, data, and attach to viewing area
  m_p_spec->setColorMap(colorMap);
  m_p_spec->setData(*plotSolver);
  m_p_spec->attach(ui.view_plot);


  QwtScaleWidget *rightAxis = ui.view_plot->axisWidget(QwtPlot::yRight);
```

```
  rightAxis->setTitle("Intensity");
  rightAxis->setColorBarEnabled(true);
  rightAxis->setColorMap(m_p_spec->data().range(), m_p_spec->colorMap());

  ui.view_plot->setAxisScale(QwtPlot::yRight,
      m_p_spec->data().range().minValue(),
      m_p_spec->data().range().maxValue() );

  ui.view_plot->enableAxis(QwtPlot::yRight);

  ui.view_plot->setAxisScale(QwtPlot::xBottom, 0, plotSolver->getMaxX());
  ui.view_plot->setAxisScale(QwtPlot::yLeft, 0, plotSolver->getMaxY());

  ui.view_plot->plotLayout()->setAlignCanvasToScales(true);
  ui.view_plot->setCanvasBackground(Qt::white);
  ui.view_plot->replot();

  // LeftButton for the zooming
  // MidButton for the panning
  // RightButton: zoom out by 1
  // Ctrl+RighButton: zoom out to full size


  m_p_zoom->setMousePattern(QwtEventPattern::MouseSelect2,
      Qt::RightButton, Qt::ControlModifier);
  m_p_zoom->setMousePattern(QwtEventPattern::MouseSelect3,
      Qt::RightButton);


  m_p_panner->setAxisEnabled(QwtPlot::yRight, false);
  m_p_panner->setMouseButton(Qt::MidButton);

  // Avoid jumping when labels with more/less digits
  // appear/disappear when scrolling vertically
  const QFontMetrics fm(ui.view_plot->axisWidget(QwtPlot::yLeft)->font());
  QwtScaleDraw *sd = ui.view_plot->axisScaleDraw(QwtPlot::yLeft);
  sd->setMinimumExtent( fm.width("100.00") );

  const QColor c(Qt::darkBlue);
  m_p_zoom->setRubberBandPen(c);
  m_p_zoom->setTrackerPen(c);

  m_rasterSet = true;
}

/*
  OLD TD_IMPORT
  File Format:
  N_TD(int) N_Time(int)
  Time Vector(float vector)
  Loc_x Loc_y Size Data-Vector(float vector);
  .
  .
  .

void PistonUi::td_import() {
  FILE* file_in;
  QString fn;
  int ntd = 0, nt = 0;
  float x, y, size;
  char buf[256];

  fn = QFileDialog::getOpenFileName(this, "Import Transducer Setup","./", "All Files (*)");

  if (fn.isEmpty()) // If choice cancelled, exit
       return;

    // Load and check if loading works.
  file_in = fopen(fn.toAscii().data(), "r");


    if(file_in == NULL) {
      QMessageBox::critical(tdui, "Error", "Could not load file");
        return;
    }

  fscanf(file_in, "%d", &ntd);
  fscanf(file_in, "%d\n", &nt);

  if (ntd <= 0 || nt <= 0) {
    fclose(file_in);
      QMessageBox::critical(tdui, "Error", "Invalid File Format - Invalid sizes given");
    return;
  }

    if (ntd > INITIAL_NB_TD_ALLOC) {
    QMessageBox::critical(tdui, "Error", "List contains too many transducer.");
    return;
  }
  td_clear(); // Clear current status

  // Read time
  initArrayf(&m_time, nt); // Create Time vector
  for(int i =0; i<nt; i++) {
    if (fscanf(file_in, "%f", &(m_time.p_data[i])) != 1) {
      QMessageBox::critical(tdui, "Error", "Invalid File Format - Could not read time vector");
      td_clear();
      return;
    }
  }

  fscanf(file_in, "\n");

  // Read Transducer List
  for(int i=0; i<ntd; i++) {
    // Scan line header
    if (fscanf(file_in, "%f", &x) != 1 | fscanf(file_in, "%f", &y) != 1 || fscanf(file_in, "%f", &size) != 1) {
      QMessageBox::critical(tdui, "Error", "Invalid File Format - Invalid Impulse Header");
      td_clear();
      return;
    }

    // Fill the rest with impulse data
    initArrayf(impData[i], nt); // Init data vector
    for(int j=0; j<nt; j++){
      if (fscanf(file_in, "%f", &(impData[i]->p_data[j])) != 1) {
        QMessageBox::critical(tdui, "Error", "Invalid File Format - Could not read impulses");
        td_clear();
        return;
      }
    }

    // Create the Impulse, then the Transducer
    imps[i] = new Impulse(impData[i], &m_time);
    tds[i] = new CircTD(size,x,y,imps[i]);
      sprintf(buf, "CircTD, Size: %.2f, X:%.2f, Y:%.2f", size, x, y);
    tdStringList << buf;
    fscanf(file_in, "\n");

  }

  m_tds_count = ntd;
  tdListModel->setStringList(tdStringList);
  ui.stp_size->setEnabled(false);
    ui.stp_dt->setEnabled(false);
} */
```

D.4 plotsolver.h

```
#include <solver.h>
#include <qwt_valuelist.h>
#include <qwt_raster_data.h>
#include <math.h>

typedef unsigned int uint;

class PlotSolver : public Solver, public QwtRasterData {
  private:
    double m_range[2];
    unsigned int m_i_Time;
    unsigned int m_currentMode;
 ARRAYF m_modeAmp; // Holds mode2 data when selected. This uses memory but saves a re-computing.
 bool m_ModeAmpSolved;

 void setupMode(int mode);


  public:
    // Static Constants
    static const unsigned int ModeNormal = 0;
    static const unsigned int ModeAmplitude = 1;
    static const unsigned int ModeHilbert = 2;

    PlotSolver(PARRAYF time, int res_x, int res_y, double dx, double dy, double vel);
 ~PlotSolver();


    // Custom Functions
    int setRange(double min, double max);
    int setTimeIndex(int index);
    double getSetTime();
    double getMaxTime();
    int getMaxTimeIndex();
    double getMaxX();
    double getMaxY();
    double maxValue();

    int setMode(const int mode);
    int getMode();

    int getXAxis( int size, double * data);
    int getYAxis( int size, double * data);
    int getTAxis( int size, double * data);

    int getXData(double x, int size, double * data); // Get data accross
    int getYData(double y, int size, double * data); // Get data vertically
    int getTData(double x, double y, int size, double * data); // Get data through time

    QwtValueList getContours(unsigned int nbCont);

    // QwtRasterData virtual Functions implementations
    virtual QwtRasterData *copy() const;
    virtual QwtDoubleInterval range() const;
    virtual double value(double x, double y) const;

};
```

D.5 plotsolver.cc

```
#include "plotsolver.h"

PlotSolver::PlotSolver(PARRAYF time, int res_x, int res_y, double dx, double dy, double vel)
  : Solver(time,res_x,res_y,dx,dy,vel), QwtRasterData() {

  m_currentMode = 0;
  m_range[0] = -1.0;
  m_range[1] = 1.0;
  m_i_Time = 0;

  // Init other things
  m_modeAmp.size = 0;
  m_modeAmp.p_data = NULL;
  m_ModeAmpSolved = false;

}

PlotSolver::~PlotSolver() {
 if ( m_modeAmp.p_data != NULL )
 delArrayf(&m_modeAmp);

}

int PlotSolver::setRange(double min, double max) {
  m_range[0] = min; m_range[1] = max;
  return 0;
}

int PlotSolver::setTimeIndex(int index) {
  m_i_Time = 0; // Default to zero

  if (m_isSolved)
    if( index >= 0 && index < (int)m_results.size_x )
      m_i_Time = index;

  return 0;

}

double PlotSolver::getSetTime() {
  return m_i_Time * (m_p_time->p_data[1]-m_p_time->p_data[0]);
}

double PlotSolver::getMaxTime() {
  if (m_isSolved)
    return (m_results.size_x-1) * (m_p_time->p_data[1]-m_p_time->p_data[0]);

  return 0;
}

int PlotSolver::getMaxTimeIndex() {
  if (m_isSolved)
    return m_results.size_x-1;

  return 0;
}

double PlotSolver::getMaxX() {
  if (m_isSolved)
    return (m_size_x-1)*m_dx;

  return 0;
}

double PlotSolver::getMaxY() {
  if (m_isSolved)
```

```
    return (m_size_y-1)*m_dy;

  return 0;
}

QwtValueList PlotSolver::getContours(unsigned int nbCont) {
 double step;

 // Set Contours
 step = (m_range[1] - m_range[0])/nbCont;

 QwtValueList contourLevels;
 if (m_currentMode == ModeNormal)
   for ( unsigned int i = 1; i < nbCont/2; i++ ) {
     contourLevels += i*step;
     contourLevels += i*(-step);
   }

 return contourLevels;
}

int PlotSolver::setMode(const int mode) {
  if ( mode < 3 ) {
    m_currentMode = mode;
 setupMode(mode);

  }

  return 0;
}

int PlotSolver::getMode() {
  return m_currentMode;
}

QwtRasterData * PlotSolver::copy() const {
        return (QwtRasterData* const)this;
}

QwtDoubleInterval PlotSolver::range() const {
        return QwtDoubleInterval(m_range[0], m_range[1]);
}

double PlotSolver::value(double x, double y) const {
  unsigned int i_x, i_y;

  i_x = abs((unsigned int)floor(x/m_dx));
  i_y = abs((unsigned int)floor(y/m_dy));

  if (!(m_isSolved && i_x < m_size_x && i_y < m_size_y))
    return 0.0;

  if ( m_currentMode == ModeNormal) {
    return m_results.p_data[i_y*m_size_y + i_x][m_i_Time];
  }
  else if ( m_currentMode == ModeAmplitude)
    return (double)m_modeAmp.p_data[0];//i_y*m_size_y + i_x];

  return 0; // Else return 0 (no data)
}

double PlotSolver::maxValue() {
  double max = 0.0;
  int N;

  if (!m_isSolved)
    return 0.0;
  N = getMaxTimeIndex();
  for (int j=0; j<m_size_x*m_size_y; j++)
    for (int i=0; i<= N; i++)
      if ( fabs(m_results.p_data[j][i]) > max )
        max = fabs(m_results.p_data[j][i]);

  return max;
}

int PlotSolver::getXAxis( int size, double * data) {

  if (size >= m_size_x) {
    for (int i = 0; i < m_size_x; i++)
      data[i] = i*m_dx;
    return SUCCESS;
  }
  return ERROR;

}

int PlotSolver::getYAxis( int size, double * data) {
  if (size >= m_size_y) {
    for (int i = 0; i < m_size_y; i++)
      data[i] = i*m_dy;
    return SUCCESS;
  }
  return ERROR;

}

int PlotSolver::getTAxis( int size, double * data) {
  int sz = getMaxTimeIndex();
  double dt = m_p_time->p_data[1]-m_p_time->p_data[0];

  if (size >= sz) {
    for (int i = 0; i < sz; i++)
      data[i] = i*dt;

    return SUCCESS;
  }

  return ERROR;
}

int PlotSolver::getXData(double y, int size, double * data) {

  if (size >= m_size_x) {
    for (int i = 0; i < m_size_x; i++)
      data[i] = value(i*m_dx, y);
    return SUCCESS;
  }
  return ERROR;

}

int PlotSolver::getYData(double x, int size, double * data) {

  if (size >= m_size_x) {
    for (int i = 0; i < m_size_x; i++)
      data[i] = value(x, i*m_dy);
    return SUCCESS;
  }
  return ERROR;
}

int PlotSolver::getTData(double x, double y, int size, double * data) {
  int sz = getMaxTimeIndex();
  double dt = m_p_time->p_data[1]-m_p_time->p_data[0];
  int idx = m_i_Time;

  if (size >= sz) {
    for (int i = 0; i < sz; i++) {
      setTimeIndex(i);
      data[i] = value(x, y);
    }

    setTimeIndex(idx);
    return SUCCESS;
  }
  return ERROR;
}


/* Private Functions */
// This sets up the data and transform necessary for certain display modes
void PlotSolver::setupMode(int mode) {
 // Compute maximum at each point.
 if (!m_isSolved)
 return;

 if (mode == ModeAmplitude && !m_ModeAmpSolved) {
 initArrayf(&m_modeAmp, m_size_x*m_size_y);

 for (unsigned int i=0; i<m_results.size_y; i++) {
 m_modeAmp.p_data[i] = m_results.p_data[i][0];
 for(unsigned int j=1; j<m_results.size_x; j++)
 if (m_results.p_data[i][j] > m_modeAmp.p_data[i])
 m_modeAmp.p_data[i] = m_results.p_data[i][j];
 }
 m_ModeAmpSolved = true;
 }
}
```

D.6 td.h

```
#ifndef __TD_H__
#define __TD_H__

#include <QtGui>
#include <transducer.h>
#include "ui_td.h"
#include "types.h"

class TD: public QDialog {
  public:
    TD(QWidget *parent = 0);
    Ui::tdEntry ui;

};


#endif
```

D.7 td.cc

```
#include "td.h"

TD::TD(QWidget *parent) {
  QStringList a,b;

  ui.setupUi(this);
  a << "Circular";
  ui.td_type->addItems(a);
  ui.td_sz->setText(DEFAULT_TD_SIZE);
  ui.td_x->setText(DEFAULT_TD_X);
  ui.td_y->setText(DEFAULT_TD_Y);

  b << "Sine" << "Gaussian Mod. Sine" << "Custom";
  ui.sig_type->addItems(b);
  ui.sig_dly->setText(DEFAULT_TD_DELAY);
  ui.sig_f->setText(DEFAULT_TD_F);
  ui.sig_gwidth->setText(DEFAULT_TD_GWIDTH);

  this->setParent(parent);
}
```

D.8 tdimport.h

```
#ifndef __TDIMPORT_H__
#define __TDIMPORT_H__

#include <QtGui>
#include <stdlib.h>
#include <transducer.h>
#include "ui_tdimport.h"
#include "types.h"

#define MAX_LAWS 256
#define MAX_ELEM 64

class TDIMP: public QDialog {

  Q_OBJECT

  public:
    TDIMP(QWidget *parent = 0);
    Ui::tdImport ui;
    bool m_hasImported;

    int nTDs() { return m_Ntds[m_currentLaw];}
    int freq() { return m_freq[m_currentLaw]*1000;} // in Hz
    int nCycles() { return m_cycles[m_currentLaw];}
    int velocity() { return m_velocity[m_currentLaw];}
    bool ready() {return m_hasImported;}
    float sz(); // in meters
    float del(); // in seconds

    float tDelay(int i) {return (float)m_t_delay[m_currentLaw][i]*1e-9;} // in Seconds
    float x(int i);

  private:
    int m_currentLaw;
    QStringList m_laws;
    QStringList m_elems;
    QStringListModel m_listModel;
    // //
    // Laws
    // //
    int m_NLaws;
    float m_ver;

    // Law Descriptor Variables
    int m_Ntds[MAX_LAWS], m_filter[MAX_LAWS], m_r_angle[MAX_LAWS], m_s_angle[MAX_LAWS];
    int m_t_first[MAX_LAWS], m_r_first[MAX_LAWS], m_scan_offset[MAX_LAWS], m_index_offset[MAX_LAWS];
    int m_g_delay[MAX_LAWS], m_f_depth[MAX_LAWS], m_velocity[MAX_LAWS], m_freq[MAX_LAWS], m_cycles[MAX_LAWS];
    int m_sum_gain[MAX_LAWS], m_mode[MAX_LAWS];
    // Element Variables
    int m_e_number[MAX_LAWS][MAX_ELEM], m_fl_gain[MAX_LAWS][MAX_ELEM], m_t_delay[MAX_LAWS][MAX_ELEM];
    int m_r_delay[MAX_LAWS][MAX_ELEM], m_amplitude[MAX_LAWS][MAX_ELEM], m_pulse_width[MAX_LAWS][MAX_ELEM];

  public slots:
    void lawImport();
    void lawChanged(int index);
    void elemSelected(const QModelIndex & index );

};
#endif
```

D.9 tdimport.cc

```
#include "tdimport.h"

TDIMP::TDIMP(QWidget *parent) {
  QStringList a,b;

  ui.setupUi(this);

  a << "Circular";
  ui.td_type->addItems(a);
  ui.td_sz->setText(DEFAULT_TD_SIZE);
  ui.td_dx->setText(DEFAULT_TD_DX);

  b << "Sine" << "Gaussian Sine";
  ui.sig_type->addItems(b);
  ui.sig_dly->setText(DEFAULT_TD_DELAY);

  connect(ui.law_import, SIGNAL(clicked()), this, SLOT(lawImport()));
  connect(ui.elem_list, SIGNAL(pressed(const QModelIndex&)), this, SLOT(elemSelected(const QModelIndex &)));
  connect(ui.elem_list, SIGNAL(activated(const QModelIndex&)), this, SLOT(elemSelected(const QModelIndex &)));
  connect(ui.law_num, SIGNAL(currentIndexChanged(int)), this, SLOT(lawChanged(int)));
  m_hasImported = false;

  ui.elem_list->setModel(&m_listModel);
  this->setParent(parent);
}


void TDIMP::lawImport() {
  FILE* file_in;
  char buf[16];
  QString fn;

  fn = QFileDialog::getOpenFileName(this, "Import Focal Law File","./", "Law Files (*.law)");

  if (fn.isEmpty()) // If choice cancelled, exit
       return;

  // Load and check if loading works.
  file_in = fopen(fn.toAscii().data(), "r");
  ui.law_filename->setText(fn.toAscii().data());

  if(file_in == NULL) {
    QMessageBox::critical(this, "Error", "Could not load file");
    return;
  }

  // Read the file
  fscanf(file_in, "V%f", &m_ver);
  fscanf(file_in, " %d", &m_NLaws);
  // Read Every Focal Law
  for (int i = 0; i< m_NLaws; i++) {

    sprintf(buf, "%d", i+1);
    m_laws << buf;

    // Read descriptor line
    fscanf(file_in, "%d", &m_Ntds[i]);
    fscanf(file_in, "%d", &m_freq[i]);
    fscanf(file_in, "%d", &m_cycles[i]);
    fscanf(file_in, "%d", &m_sum_gain[i]);
    fscanf(file_in, "%d", &m_mode[i]);
    fscanf(file_in, "%d", &m_filter[i]);
    fscanf(file_in, "%d", &m_r_angle[i]);
    fscanf(file_in, "%d", &m_s_angle[i]);
    fscanf(file_in, "%d", &m_r_first[i]);
    fscanf(file_in, "%d", &m_scan_offset[i]);
    fscanf(file_in, "%d", &m_index_offset[i]);
    fscanf(file_in, "%d", &m_t_first[i]);
    fscanf(file_in, "%d", &m_g_delay[i]);
    fscanf(file_in, "%d", &m_f_depth[i]);
    fscanf(file_in, "%d", &m_velocity[i]);

    for (int j = 0; j < m_Ntds[i]; j++) {
      fscanf(file_in, "%d", &m_e_number[i][j]);
      fscanf(file_in, "%d", &m_fl_gain[i][j]);
      fscanf(file_in, "%d", &m_t_delay[i][j]);
      fscanf(file_in, "%d", &m_r_delay[i][j]);
      fscanf(file_in, "%d", &m_amplitude[i][j]);
      fscanf(file_in, "%d", &m_pulse_width[i][j]);
    }
  }


  ui.law_num->addItems(m_laws);
  m_currentLaw = 0;
  lawChanged(m_currentLaw);
  m_hasImported = 1;

}


void TDIMP::lawChanged(int i) {
  char buf[256];

  m_currentLaw = i;

  // Setup the law data
  sprintf(buf, "%d", m_velocity[i]);
  ui.law_medVel->setText(buf);
  sprintf(buf, "%d", m_freq[i]);
  ui.law_freq->setText(buf);
  sprintf(buf, "%d", m_sum_gain[i]);
  ui.law_sumgain->setText(buf);
  sprintf(buf, "%d", m_mode[i]);
  ui.law_mode->setText(buf);
  sprintf(buf, "%d", m_cycles[i]);
  ui.law_cycles->setText(buf);
  sprintf(buf, "%d", m_g_delay[i]);
  ui.law_gDelay->setText(buf);
  sprintf(buf, "%d", m_t_first[i]);
  ui.law_tFirst->setText(buf);
  sprintf(buf, "%d", m_index_offset[i]);
  ui.law_indexOffset->setText(buf);
  sprintf(buf, "%d", m_scan_offset[i]);
  ui.law_scanOffset->setText(buf);
  sprintf(buf, "%d", m_r_first[i]);
  ui.law_rFirst->setText(buf);
  sprintf(buf, "%d", m_s_angle[i]);
  ui.law_sAngle->setText(buf);
  sprintf(buf, "%d", m_r_angle[i]);
  ui.law_rAngle->setText(buf);
  sprintf(buf, "%d", m_filter[i]);
  ui.law_filter->setText(buf);
  sprintf(buf, "%d", m_f_depth[i]);
  ui.law_fDepth->setText(buf);

  // Fill the element list
  m_elems.clear();
  for (int j =0; j< m_Ntds[i]; j++) {
    sprintf(buf, "%d", j+1);
    m_elems << buf;
  }

  m_listModel.setStringList(m_elems);
}

void TDIMP::elemSelected(const QModelIndex &index) {
  int row = index.row();
  int i = m_currentLaw;
  char buf[256];

  if ( row < 0 )
    return;

  sprintf(buf, "%d", m_fl_gain[i][row]);
  ui.elem_flGain->setText(buf);
  sprintf(buf, "%d", m_t_delay[i][row]);
  ui.elem_tDelay->setText(buf);
  sprintf(buf, "%d", m_r_delay[i][row]);
  ui.elem_rDelay->setText(buf);
  sprintf(buf, "%d", m_amplitude[i][row]);
  ui.elem_amplitude->setText(buf);
  sprintf(buf, "%d", m_pulse_width[i][row]);
  ui.elem_pWidth->setText(buf);
}

float TDIMP::sz(){
  bool ok;

  float size = ui.td_sz->text().toFloat(&ok);
  if(!ok) {
    QMessageBox::critical(this, "Error", "Invalid Size");
    return 0.0;
  }

  return size;
}

float TDIMP::del() {
  bool ok;

  float delay = ui.sig_dly->text().toFloat(&ok);
  if(!ok) {
    QMessageBox::critical(this, "Error", "Invalid Delay Time");
    return 0.0;
  }

  return delay;
}


float TDIMP::x(int i) {
  bool ok;

  float dx = ui.td_dx->text().toFloat(&ok);
  if(!ok) {
    QMessageBox::critical(this, "Error", "Invalid dx value");
    return 0.0;
  }

  return i*dx;
}
```

D.10 types.h

```
#ifndef __TYPE_H__
#define __TYPE_H__

#define INITIAL_NB_TD_ALLOC 128

#define TYPE_CIRCULAR  0
#define TYPE_RECTANGULAR  1

#define SIGNAL_SINE 0
#define SIGNAL_GAUSSIAN_SINE 1
#define SIGNAL_CUSTOM 2

#define DEFAULT_DX "2.5e-4"
#define DEFAULT_DY "2.5e-4"
#define DEFAULT_SX "64"
#define DEFAULT_SY "64"
#define DEFAULT_VEL "1500"
#define DEFAULT_DT "1e-8"
#define DEFAULT_LT "4097"
#define DEFAULT_TD_SIZE "1e-3"
#define DEFAULT_TD_GWIDTH "1e-6"
#define DEFAULT_TD_X    "0"
#define DEFAULT_TD_Y    "0"
#define DEFAULT_TD_DELAY   "0"
#define DEFAULT_TD_F       "1e6"
#define DEFAULT_TD_DX      "1e-3"
#define DEFAULT_X "0"
#define DEFAULT_Y "0"
#define DEFAULT_NB_CONTOURS 10


#endif
```

D.11 ui_pistonui.h

```
/********************************************************************************
** Form generated from reading ui file 'pistonui.ui'
**
** Created: Thu Nov 1 16:08:39 2007
**      by: Qt User Interface Compiler version 4.3.2
**
** WARNING! All changes made in this file will be lost when recompiling ui file!
********************************************************************************/

#ifndef UI_PISTONUI_H
#define UI_PISTONUI_H

#include <QtCore/QVariant>
#include <QtGui/QAction>
#include <QtGui/QApplication>
#include <QtGui/QButtonGroup>
#include <QtGui/QCheckBox>
#include <QtGui/QGridLayout>
```

```
#include <QtGui/QGroupBox>
#include <QtGui/QHBoxLayout>
#include <QtGui/QLabel>
#include <QtGui/QLineEdit>
#include <QtGui/QListView>
#include <QtGui/QMainWindow>
#include <QtGui/QPushButton>
#include <QtGui/QRadioButton>
#include <QtGui/QScrollBar>
#include <QtGui/QSpacerItem>
#include <QtGui/QStatusBar>
#include <QtGui/QTabWidget>
#include <QtGui/QVBoxLayout>
#include <QtGui/QWidget>
#include "qwt_plot.h"

class Ui_MainWindow
{
public:
    QWidget *centralwidget;
    QGridLayout *gridLayout;
    QGridLayout *gridLayout1;
    QTabWidget *tabWidget;
    QWidget *stp;
    QGridLayout *gridLayout2;
    QGridLayout *gridLayout3;
    QGroupBox *stp_box_medium;
    QGridLayout *gridLayout4;
    QLabel *label;
    QLabel *label_2;
    QLineEdit *stp_dx;
    QLineEdit *stp_dy;
    QLabel *label_3;
    QLabel *label_4;
    QLineEdit *stp_maxx;
    QLineEdit *stp_maxy;
    QVBoxLayout *vboxLayout;
    QLabel *label_5;
    QLineEdit *stp_vel;
    QGroupBox *stp_box_time;
    QLabel *label_33;
    QLineEdit *stp_dt;
    QLineEdit *stp_size;
    QLabel *label_32;
    QLabel *label_31;
    QWidget *layoutWidget;
    QHBoxLayout *hboxLayout;
    QRadioButton *stp_Normal;
    QRadioButton *stp_Amp;
    QPushButton *stp_solve;
    QGroupBox *stp_box_transducers;
    QGridLayout *gridLayout5;
    QListView *stp_list;
    QVBoxLayout *vboxLayout1;
    QPushButton *stp_add;
    QPushButton *stp_rem;
    QPushButton *stp_clear;
    QPushButton *stp_import;
    QwtPlot *stp_plot;
    QHBoxLayout *hboxLayout1;
    QLabel *label_7;
    QLineEdit *stp_X;
    QLineEdit *stp_Y;
    QRadioButton *stp_impulse;
    QRadioButton *stp_response;
    QSpacerItem *spacerItem;
    QWidget *view;
    QGridLayout *gridLayout6;
```

```
QwtPlot *view_plot;
QwtPlot *view_vertPlot;
QwtPlot *view_horizPlot;
QwtPlot *view_timePlot;
QScrollBar *view_slider;
QGridLayout *gridLayout7;
QVBoxLayout *vboxLayout2;
QCheckBox *view_spec;
QCheckBox *view_cont;
QVBoxLayout *vboxLayout3;
QRadioButton *view_time;
QRadioButton *view_maxAmp;
QSpacerItem *spacerItem1;
QGridLayout *gridLayout8;
QLabel *label_6;
QPushButton *view_load;
QScrollBar *view_intensity;
QPushButton *view_save;
QStatusBar *statusbar;

void setupUi(QMainWindow *MainWindow)
{
if (MainWindow->objectName().isEmpty())
    MainWindow->setObjectName(QString::fromUtf8("MainWindow"));
MainWindow->resize(1024, 760);
QSizePolicy sizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);
sizePolicy.setHorizontalStretch(0);
sizePolicy.setVerticalStretch(0);
sizePolicy.setHeightForWidth(MainWindow->sizePolicy().hasHeightForWidth());
MainWindow->setSizePolicy(sizePolicy);
MainWindow->setMinimumSize(QSize(1024, 760));
MainWindow->setMaximumSize(QSize(1024, 760));
MainWindow->setBaseSize(QSize(800, 600));
centralwidget = new QWidget(MainWindow);
centralwidget->setObjectName(QString::fromUtf8("centralwidget"));
gridLayout = new QGridLayout(centralwidget);
gridLayout->setObjectName(QString::fromUtf8("gridLayout"));
gridLayout1 = new QGridLayout();
gridLayout1->setObjectName(QString::fromUtf8("gridLayout1"));

gridLayout->addLayout(gridLayout1, 0, 0, 1, 1);

tabWidget = new QTabWidget(centralwidget);
tabWidget->setObjectName(QString::fromUtf8("tabWidget"));
QSizePolicy sizePolicy1(QSizePolicy::MinimumExpanding, QSizePolicy::MinimumExpanding);
sizePolicy1.setHorizontalStretch(0);
sizePolicy1.setVerticalStretch(0);
sizePolicy1.setHeightForWidth(tabWidget->sizePolicy().hasHeightForWidth());
tabWidget->setSizePolicy(sizePolicy1);
stp = new QWidget();
stp->setObjectName(QString::fromUtf8("stp"));
gridLayout2 = new QGridLayout(stp);
gridLayout2->setObjectName(QString::fromUtf8("gridLayout2"));
gridLayout3 = new QGridLayout();
gridLayout3->setObjectName(QString::fromUtf8("gridLayout3"));
stp_box_medium = new QGroupBox(stp);
stp_box_medium->setObjectName(QString::fromUtf8("stp_box_medium"));
QSizePolicy sizePolicy2(QSizePolicy::Minimum, QSizePolicy::Minimum);
sizePolicy2.setHorizontalStretch(0);
sizePolicy2.setVerticalStretch(0);
sizePolicy2.setHeightForWidth(stp_box_medium->sizePolicy().hasHeightForWidth());
stp_box_medium->setSizePolicy(sizePolicy2);
stp_box_medium->setMinimumSize(QSize(210, 230));
stp_box_medium->setMaximumSize(QSize(210, 230));
gridLayout4 = new QGridLayout(stp_box_medium);
gridLayout4->setObjectName(QString::fromUtf8("gridLayout4"));
label = new QLabel(stp_box_medium);
label->setObjectName(QString::fromUtf8("label"));
```

```
label->setMaximumSize(QSize(16777215, 25));
label->setAlignment(Qt::AlignCenter);

gridLayout4->addWidget(label, 0, 0, 1, 1);

label_2 = new QLabel(stp_box_medium);
label_2->setObjectName(QString::fromUtf8("label_2"));
label_2->setMaximumSize(QSize(16777215, 25));
label_2->setAlignment(Qt::AlignCenter);

gridLayout4->addWidget(label_2, 0, 1, 1, 1);

stp_dx = new QLineEdit(stp_box_medium);
stp_dx->setObjectName(QString::fromUtf8("stp_dx"));

gridLayout4->addWidget(stp_dx, 1, 0, 1, 1);

stp_dy = new QLineEdit(stp_box_medium);
stp_dy->setObjectName(QString::fromUtf8("stp_dy"));

gridLayout4->addWidget(stp_dy, 1, 1, 1, 1);

label_3 = new QLabel(stp_box_medium);
label_3->setObjectName(QString::fromUtf8("label_3"));
label_3->setMaximumSize(QSize(16777215, 25));
label_3->setAlignment(Qt::AlignCenter);

gridLayout4->addWidget(label_3, 2, 0, 1, 1);

label_4 = new QLabel(stp_box_medium);
label_4->setObjectName(QString::fromUtf8("label_4"));
label_4->setMaximumSize(QSize(16777215, 25));
label_4->setAlignment(Qt::AlignCenter);

gridLayout4->addWidget(label_4, 2, 1, 1, 1);

stp_maxx = new QLineEdit(stp_box_medium);
stp_maxx->setObjectName(QString::fromUtf8("stp_maxx"));

gridLayout4->addWidget(stp_maxx, 3, 0, 1, 1);

stp_maxy = new QLineEdit(stp_box_medium);
stp_maxy->setObjectName(QString::fromUtf8("stp_maxy"));

gridLayout4->addWidget(stp_maxy, 3, 1, 1, 1);

vboxLayout = new QVBoxLayout();
vboxLayout->setObjectName(QString::fromUtf8("vboxLayout"));
label_5 = new QLabel(stp_box_medium);
label_5->setObjectName(QString::fromUtf8("label_5"));
label_5->setMaximumSize(QSize(200, 25));
label_5->setAlignment(Qt::AlignCenter);

vboxLayout->addWidget(label_5);

stp_vel = new QLineEdit(stp_box_medium);
stp_vel->setObjectName(QString::fromUtf8("stp_vel"));

vboxLayout->addWidget(stp_vel);


gridLayout4->addLayout(vboxLayout, 4, 0, 1, 2);


gridLayout3->addWidget(stp_box_medium, 0, 0, 2, 1);

stp_box_time = new QGroupBox(stp);
stp_box_time->setObjectName(QString::fromUtf8("stp_box_time"));
```

```
QSizePolicy sizePolicy3(QSizePolicy::Fixed, QSizePolicy::Ignored);
sizePolicy3.setHorizontalStretch(0);
sizePolicy3.setVerticalStretch(0);
sizePolicy3.setHeightForWidth(stp_box_time->sizePolicy().hasHeightForWidth());
stp_box_time->setSizePolicy(sizePolicy3);
stp_box_time->setMinimumSize(QSize(210, 200));
label_33 = new QLabel(stp_box_time);
label_33->setObjectName(QString::fromUtf8("label_33"));
label_33->setGeometry(QRect(20, 80, 171, 61));
QSizePolicy sizePolicy4(QSizePolicy::Fixed, QSizePolicy::Fixed);
sizePolicy4.setHorizontalStretch(0);
sizePolicy4.setVerticalStretch(0);
sizePolicy4.setHeightForWidth(label_33->sizePolicy().hasHeightForWidth());
label_33->setSizePolicy(sizePolicy4);
label_33->setWordWrap(true);
stp_dt = new QLineEdit(stp_box_time);
stp_dt->setObjectName(QString::fromUtf8("stp_dt"));
stp_dt->setGeometry(QRect(20, 50, 81, 23));
sizePolicy4.setHeightForWidth(stp_dt->sizePolicy().hasHeightForWidth());
stp_dt->setSizePolicy(sizePolicy4);
stp_size = new QLineEdit(stp_box_time);
stp_size->setObjectName(QString::fromUtf8("stp_size"));
stp_size->setGeometry(QRect(110, 50, 81, 23));
sizePolicy4.setHeightForWidth(stp_size->sizePolicy().hasHeightForWidth());
stp_size->setSizePolicy(sizePolicy4);
label_32 = new QLabel(stp_box_time);
label_32->setObjectName(QString::fromUtf8("label_32"));
label_32->setGeometry(QRect(110, 30, 81, 20));
sizePolicy4.setHeightForWidth(label_32->sizePolicy().hasHeightForWidth());
label_32->setSizePolicy(sizePolicy4);
label_32->setAlignment(Qt::AlignCenter);
label_31 = new QLabel(stp_box_time);
label_31->setObjectName(QString::fromUtf8("label_31"));
label_31->setGeometry(QRect(20, 30, 81, 21));
sizePolicy4.setHeightForWidth(label_31->sizePolicy().hasHeightForWidth());
label_31->setSizePolicy(sizePolicy4);
label_31->setAlignment(Qt::AlignCenter);
layoutWidget = new QWidget(stp_box_time);
layoutWidget->setObjectName(QString::fromUtf8("layoutWidget"));
layoutWidget->setGeometry(QRect(10, 170, 191, 21));
hboxLayout = new QHBoxLayout(layoutWidget);
hboxLayout->setObjectName(QString::fromUtf8("hboxLayout"));
hboxLayout->setContentsMargins(0, 0, 0, 0);
stp_Normal = new QRadioButton(layoutWidget);
stp_Normal->setObjectName(QString::fromUtf8("stp_Normal"));
stp_Normal->setChecked(true);

hboxLayout->addWidget(stp_Normal);

stp_Amp = new QRadioButton(layoutWidget);
stp_Amp->setObjectName(QString::fromUtf8("stp_Amp"));

hboxLayout->addWidget(stp_Amp);


gridLayout3->addWidget(stp_box_time, 0, 1, 1, 1);

stp_solve = new QPushButton(stp);
stp_solve->setObjectName(QString::fromUtf8("stp_solve"));
sizePolicy2.setHeightForWidth(stp_solve->sizePolicy().hasHeightForWidth());
stp_solve->setSizePolicy(sizePolicy2);
stp_solve->setMinimumSize(QSize(210, 25));
stp_solve->setMaximumSize(QSize(16777215, 25));

gridLayout3->addWidget(stp_solve, 1, 1, 1, 1);


gridLayout2->addLayout(gridLayout3, 0, 0, 1, 1);
```

```
stp_box_transducers = new QGroupBox(stp);
stp_box_transducers->setObjectName(QString::fromUtf8("stp_box_transducers"));
QSizePolicy sizePolicy5(QSizePolicy::MinimumExpanding, QSizePolicy::Fixed);
sizePolicy5.setHorizontalStretch(0);
sizePolicy5.setVerticalStretch(0);
sizePolicy5.setHeightForWidth(stp_box_transducers->sizePolicy().hasHeightForWidth());
stp_box_transducers->setSizePolicy(sizePolicy5);
gridLayout5 = new QGridLayout(stp_box_transducers);
gridLayout5->setObjectName(QString::fromUtf8("gridLayout5"));
stp_list = new QListView(stp_box_transducers);
stp_list->setObjectName(QString::fromUtf8("stp_list"));
sizePolicy1.setHeightForWidth(stp_list->sizePolicy().hasHeightForWidth());
stp_list->setSizePolicy(sizePolicy1);
stp_list->setEditTriggers(QAbstractItemView::NoEditTriggers);

gridLayout5->addWidget(stp_list, 0, 0, 1, 1);

vboxLayout1 = new QVBoxLayout();
vboxLayout1->setObjectName(QString::fromUtf8("vboxLayout1"));
stp_add = new QPushButton(stp_box_transducers);
stp_add->setObjectName(QString::fromUtf8("stp_add"));

vboxLayout1->addWidget(stp_add);

stp_rem = new QPushButton(stp_box_transducers);
stp_rem->setObjectName(QString::fromUtf8("stp_rem"));

vboxLayout1->addWidget(stp_rem);

stp_clear = new QPushButton(stp_box_transducers);
stp_clear->setObjectName(QString::fromUtf8("stp_clear"));

vboxLayout1->addWidget(stp_clear);

stp_import = new QPushButton(stp_box_transducers);
stp_import->setObjectName(QString::fromUtf8("stp_import"));

vboxLayout1->addWidget(stp_import);


gridLayout5->addLayout(vboxLayout1, 0, 1, 1, 1);


gridLayout2->addWidget(stp_box_transducers, 0, 1, 1, 1);

stp_plot = new QwtPlot(stp);
stp_plot->setObjectName(QString::fromUtf8("stp_plot"));
sizePolicy1.setHeightForWidth(stp_plot->sizePolicy().hasHeightForWidth());
stp_plot->setSizePolicy(sizePolicy1);

gridLayout2->addWidget(stp_plot, 1, 0, 1, 2);

hboxLayout1 = new QHBoxLayout();
hboxLayout1->setObjectName(QString::fromUtf8("hboxLayout1"));
label_7 = new QLabel(stp);
label_7->setObjectName(QString::fromUtf8("label_7"));
label_7->setMaximumSize(QSize(75, 25));

hboxLayout1->addWidget(label_7);

stp_X = new QLineEdit(stp);
stp_X->setObjectName(QString::fromUtf8("stp_X"));
sizePolicy4.setHeightForWidth(stp_X->sizePolicy().hasHeightForWidth());
stp_X->setSizePolicy(sizePolicy4);
stp_X->setMinimumSize(QSize(50, 25));
stp_X->setMaximumSize(QSize(50, 25));
```

```
hboxLayout1->addWidget(stp_X);

stp_Y = new QLineEdit(stp);
stp_Y->setObjectName(QString::fromUtf8("stp_Y"));
sizePolicy4.setHeightForWidth(stp_Y->sizePolicy().hasHeightForWidth());
stp_Y->setSizePolicy(sizePolicy4);
stp_Y->setMinimumSize(QSize(50, 25));
stp_Y->setMaximumSize(QSize(50, 25));

hboxLayout1->addWidget(stp_Y);

stp_impulse = new QRadioButton(stp);
stp_impulse->setObjectName(QString::fromUtf8("stp_impulse"));
stp_impulse->setChecked(true);

hboxLayout1->addWidget(stp_impulse);

stp_response = new QRadioButton(stp);
stp_response->setObjectName(QString::fromUtf8("stp_response"));
stp_response->setMinimumSize(QSize(90, 0));

hboxLayout1->addWidget(stp_response);


gridLayout2->addLayout(hboxLayout1, 2, 0, 1, 1);

spacerItem = new QSpacerItem(381, 31, QSizePolicy::Expanding, QSizePolicy::Minimum);

gridLayout2->addItem(spacerItem, 2, 1, 1, 1);

tabWidget->addTab(stp, QString());
view = new QWidget();
view->setObjectName(QString::fromUtf8("view"));
gridLayout6 = new QGridLayout(view);
gridLayout6->setObjectName(QString::fromUtf8("gridLayout6"));
view_plot = new QwtPlot(view);
view_plot->setObjectName(QString::fromUtf8("view_plot"));
sizePolicy1.setHeightForWidth(view_plot->sizePolicy().hasHeightForWidth());
view_plot->setSizePolicy(sizePolicy1);

gridLayout6->addWidget(view_plot, 0, 0, 3, 1);

view_vertPlot = new QwtPlot(view);
view_vertPlot->setObjectName(QString::fromUtf8("view_vertPlot"));

gridLayout6->addWidget(view_vertPlot, 0, 1, 1, 2);

view_horizPlot = new QwtPlot(view);
view_horizPlot->setObjectName(QString::fromUtf8("view_horizPlot"));

gridLayout6->addWidget(view_horizPlot, 1, 1, 1, 2);

view_timePlot = new QwtPlot(view);
view_timePlot->setObjectName(QString::fromUtf8("view_timePlot"));

gridLayout6->addWidget(view_timePlot, 2, 1, 1, 2);

view_slider = new QScrollBar(view);
view_slider->setObjectName(QString::fromUtf8("view_slider"));
view_slider->setTracking(false);
view_slider->setOrientation(Qt::Horizontal);

gridLayout6->addWidget(view_slider, 3, 0, 1, 3);

gridLayout7 = new QGridLayout();
gridLayout7->setObjectName(QString::fromUtf8("gridLayout7"));
vboxLayout2 = new QVBoxLayout();
vboxLayout2->setObjectName(QString::fromUtf8("vboxLayout2"));
```

```
view_spec = new QCheckBox(view);
view_spec->setObjectName(QString::fromUtf8("view_spec"));
sizePolicy4.setHeightForWidth(view_spec->sizePolicy().hasHeightForWidth());
view_spec->setSizePolicy(sizePolicy4);
view_spec->setChecked(true);

vboxLayout2->addWidget(view_spec);

view_cont = new QCheckBox(view);
view_cont->setObjectName(QString::fromUtf8("view_cont"));
sizePolicy4.setHeightForWidth(view_cont->sizePolicy().hasHeightForWidth());
view_cont->setSizePolicy(sizePolicy4);

vboxLayout2->addWidget(view_cont);


gridLayout7->addLayout(vboxLayout2, 0, 0, 1, 1);

vboxLayout3 = new QVBoxLayout();
vboxLayout3->setObjectName(QString::fromUtf8("vboxLayout3"));
view_time = new QRadioButton(view);
view_time->setObjectName(QString::fromUtf8("view_time"));
view_time->setChecked(true);

vboxLayout3->addWidget(view_time);

view_maxAmp = new QRadioButton(view);
view_maxAmp->setObjectName(QString::fromUtf8("view_maxAmp"));

vboxLayout3->addWidget(view_maxAmp);

spacerItem1 = new QSpacerItem(71, 20, QSizePolicy::Expanding, QSizePolicy::Minimum);

vboxLayout3->addItem(spacerItem1);


gridLayout7->addLayout(vboxLayout3, 0, 1, 1, 1);


gridLayout6->addLayout(gridLayout7, 4, 0, 1, 2);

gridLayout8 = new QGridLayout();
gridLayout8->setObjectName(QString::fromUtf8("gridLayout8"));
label_6 = new QLabel(view);
label_6->setObjectName(QString::fromUtf8("label_6"));
QSizePolicy sizePolicy6(QSizePolicy::MinimumExpanding, QSizePolicy::Preferred);
sizePolicy6.setHorizontalStretch(0);
sizePolicy6.setVerticalStretch(0);
sizePolicy6.setHeightForWidth(label_6->sizePolicy().hasHeightForWidth());
label_6->setSizePolicy(sizePolicy6);
label_6->setAlignment(Qt::AlignCenter);

gridLayout8->addWidget(label_6, 0, 0, 1, 1);

view_load = new QPushButton(view);
view_load->setObjectName(QString::fromUtf8("view_load"));
sizePolicy4.setHeightForWidth(view_load->sizePolicy().hasHeightForWidth());
view_load->setSizePolicy(sizePolicy4);

gridLayout8->addWidget(view_load, 0, 1, 1, 1);

view_intensity = new QScrollBar(view);
view_intensity->setObjectName(QString::fromUtf8("view_intensity"));
view_intensity->setMinimum(1);
view_intensity->setMaximum(100);
view_intensity->setValue(100);
view_intensity->setTracking(true);
view_intensity->setOrientation(Qt::Horizontal);
```

```
gridLayout8->addWidget(view_intensity, 1, 0, 1, 1);

view_save = new QPushButton(view);
view_save->setObjectName(QString::fromUtf8("view_save"));
sizePolicy4.setHeightForWidth(view_save->sizePolicy().hasHeightForWidth());
view_save->setSizePolicy(sizePolicy4);

gridLayout8->addWidget(view_save, 1, 1, 1, 1);


gridLayout6->addLayout(gridLayout8, 4, 2, 1, 1);

tabWidget->addTab(view, QString());

gridLayout->addWidget(tabWidget, 1, 0, 1, 1);

MainWindow->setCentralWidget(centralwidget);
statusbar = new QStatusBar(MainWindow);
statusbar->setObjectName(QString::fromUtf8("statusbar"));
MainWindow->setStatusBar(statusbar);

retranslateUi(MainWindow);

tabWidget->setCurrentIndex(0);


QMetaObject::connectSlotsByName(MainWindow);
} // setupUi

void retranslateUi(QMainWindow *MainWindow)
{
MainWindow->setWindowTitle(QApplication::translate("MainWindow", "Piston Transducer Simulation", 0, QApplication::UnicodeUTF8));
stp_box_medium->setTitle(QApplication::translate("MainWindow", "Medium", 0, QApplication::UnicodeUTF8));
label->setText(QApplication::translate("MainWindow", "dx", 0, QApplication::UnicodeUTF8));
label_2->setText(QApplication::translate("MainWindow", "dy", 0, QApplication::UnicodeUTF8));
label_3->setText(QApplication::translate("MainWindow", "Max x", 0, QApplication::UnicodeUTF8));
label_4->setText(QApplication::translate("MainWindow", "Max y", 0, QApplication::UnicodeUTF8));
label_5->setText(QApplication::translate("MainWindow", "Medium Velocity", 0, QApplication::UnicodeUTF8));
stp_box_time->setTitle(QApplication::translate("MainWindow", "Solver Parameters", 0, QApplication::UnicodeUTF8));
label_33->setText(QApplication::translate("MainWindow",
"*For faster computations, \"Size\" must follow S=2^N + 1, where N is an integer", 0, QApplication::UnicodeUTF8));
label_32->setText(QApplication::translate("MainWindow", "Size*", 0, QApplication::UnicodeUTF8));
label_31->setText(QApplication::translate("MainWindow", "dt", 0, QApplication::UnicodeUTF8));
stp_Normal->setText(QApplication::translate("MainWindow", "Normal", 0, QApplication::UnicodeUTF8));
stp_Amp->setText(QApplication::translate("MainWindow", "Hilbert", 0, QApplication::UnicodeUTF8));
stp_solve->setText(QApplication::translate("MainWindow", "Solve", 0, QApplication::UnicodeUTF8));
stp_box_transducers->setTitle(QApplication::translate("MainWindow", "Transducer", 0, QApplication::UnicodeUTF8));
stp_add->setText(QApplication::translate("MainWindow", "Add", 0, QApplication::UnicodeUTF8));
stp_rem->setText(QApplication::translate("MainWindow", "Remove", 0, QApplication::UnicodeUTF8));
stp_clear->setToolTip(QApplication::translate("MainWindow", "Import a Transducer set", 0, QApplication::UnicodeUTF8));
stp_clear->setText(QApplication::translate("MainWindow", "Clear", 0, QApplication::UnicodeUTF8));
stp_import->setToolTip(QApplication::translate("MainWindow", "Import a Transducer set", 0, QApplication::UnicodeUTF8));
stp_import->setText(QApplication::translate("MainWindow", "Import", 0, QApplication::UnicodeUTF8));
label_7->setText(QApplication::translate("MainWindow", "X, Y Loc :", 0, QApplication::UnicodeUTF8));
stp_impulse->setText(QApplication::translate("MainWindow", "Impulse", 0, QApplication::UnicodeUTF8));
stp_response->setText(QApplication::translate("MainWindow", "Response", 0, QApplication::UnicodeUTF8));
tabWidget->setTabText(tabWidget->indexOf(stp), QApplication::translate("MainWindow", "Setup", 0, QApplication::UnicodeUTF8));
view_vertPlot->setProperty("propertiesDocument", QVariant(QApplication::translate("MainWindow", "Vertical",
0, QApplication::UnicodeUTF8)));
view_horizPlot->setProperty("propertiesDocument", QVariant(QApplication::translate("MainWindow", "Horizontal",
0, QApplication::UnicodeUTF8)));
view_timePlot->setProperty("propertiesDocument", QVariant(QApplication::translate("MainWindow", "Time", 0, QApplication::UnicodeUT
view_spec->setText(QApplication::translate("MainWindow",
"Spectrogram", 0, QApplication::UnicodeUTF8));
view_cont->setText(QApplication::translate("MainWindow", "Contour", 0, QApplication::UnicodeUTF8));
view_time->setText(QApplication::translate("MainWindow", "Time Slice", 0, QApplication::UnicodeUTF8));
view_maxAmp->setText(QApplication::translate("MainWindow", "Max. Amp.", 0, QApplication::UnicodeUTF8));
```

```
    label_6->setText(QApplication::translate("MainWindow", "Intensity", 0, QApplication::UnicodeUTF8));
    view_load->setText(QApplication::translate("MainWindow", "Load", 0, QApplication::UnicodeUTF8));
    view_save->setText(QApplication::translate("MainWindow",
    "Save", 0, QApplication::UnicodeUTF8));
    tabWidget->setTabText(tabWidget->indexOf(view), QApplication::translate("MainWindow", "Viewing", 0, QApplication::UnicodeUTF8));
    Q_UNUSED(MainWindow);
    } // retranslateUi

};

namespace Ui {
    class MainWindow: public Ui_MainWindow {};
} // namespace Ui

#endif // UI_PISTONUI_H
```

D.12 ui_td.h

```
/********************************************************************************
** Form generated from reading ui file 'td.ui'
**
** Created: Thu Nov 1 16:08:39 2007
**      by: Qt User Interface Compiler version 4.3.2
**
** WARNING! All changes made in this file will be lost when recompiling ui file!
********************************************************************************/

#ifndef UI_TD_H
#define UI_TD_H

#include <QtCore/QVariant>
#include <QtGui/QAction>
#include <QtGui/QApplication>
#include <QtGui/QButtonGroup>
#include <QtGui/QComboBox>
#include <QtGui/QDialog>
#include <QtGui/QGridLayout>
#include <QtGui/QGroupBox>
#include <QtGui/QHBoxLayout>
#include <QtGui/QLabel>
#include <QtGui/QLineEdit>
#include <QtGui/QPushButton>

class Ui_tdEntry
{
public:
    QGridLayout *gridLayout;
    QGroupBox *groupBox;
    QGridLayout *gridLayout1;
    QLabel *label_2;
    QComboBox *td_type;
    QLabel *label;
    QLineEdit *td_sz;
    QLabel *label_3;
    QLineEdit *td_x;
    QLabel *label_4;
    QLineEdit *td_y;
    QGroupBox *groupBox_2;
    QGridLayout *gridLayout2;
    QLabel *label_5;
    QComboBox *sig_type;
    QLabel *label_6;
    QLineEdit *sig_dly;
    QLabel *label_7;
    QLineEdit *sig_f;
    QLabel *label_8;
```

```
QLineEdit *sig_gwidth;
QHBoxLayout *hboxLayout;
QPushButton *add;
QPushButton *done;

void setupUi(QDialog *tdEntry)
{
if (tdEntry->objectName().isEmpty())
    tdEntry->setObjectName(QString::fromUtf8("tdEntry"));
tdEntry->setWindowModality(Qt::NonModal);
tdEntry->resize(450, 210);
QSizePolicy sizePolicy(QSizePolicy::MinimumExpanding, QSizePolicy::MinimumExpanding);
sizePolicy.setHorizontalStretch(0);
sizePolicy.setVerticalStretch(0);
sizePolicy.setHeightForWidth(tdEntry->sizePolicy().hasHeightForWidth());
tdEntry->setSizePolicy(sizePolicy);
tdEntry->setMinimumSize(QSize(450, 210));
tdEntry->setMaximumSize(QSize(450, 210));
tdEntry->setBaseSize(QSize(320, 200));
tdEntry->setModal(false);
gridLayout = new QGridLayout(tdEntry);
gridLayout->setObjectName(QString::fromUtf8("gridLayout"));
groupBox = new QGroupBox(tdEntry);
groupBox->setObjectName(QString::fromUtf8("groupBox"));
gridLayout1 = new QGridLayout(groupBox);
gridLayout1->setObjectName(QString::fromUtf8("gridLayout1"));
label_2 = new QLabel(groupBox);
label_2->setObjectName(QString::fromUtf8("label_2"));

gridLayout1->addWidget(label_2, 0, 0, 1, 1);

td_type = new QComboBox(groupBox);
td_type->setObjectName(QString::fromUtf8("td_type"));

gridLayout1->addWidget(td_type, 0, 1, 1, 1);

label = new QLabel(groupBox);
label->setObjectName(QString::fromUtf8("label"));

gridLayout1->addWidget(label, 1, 0, 1, 1);

td_sz = new QLineEdit(groupBox);
td_sz->setObjectName(QString::fromUtf8("td_sz"));

gridLayout1->addWidget(td_sz, 1, 1, 1, 1);

label_3 = new QLabel(groupBox);
label_3->setObjectName(QString::fromUtf8("label_3"));

gridLayout1->addWidget(label_3, 2, 0, 1, 1);

td_x = new QLineEdit(groupBox);
td_x->setObjectName(QString::fromUtf8("td_x"));

gridLayout1->addWidget(td_x, 2, 1, 1, 1);

label_4 = new QLabel(groupBox);
label_4->setObjectName(QString::fromUtf8("label_4"));

gridLayout1->addWidget(label_4, 3, 0, 1, 1);

td_y = new QLineEdit(groupBox);
td_y->setObjectName(QString::fromUtf8("td_y"));

gridLayout1->addWidget(td_y, 3, 1, 1, 1);


gridLayout->addWidget(groupBox, 0, 0, 2, 1);
```

```
groupBox_2 = new QGroupBox(tdEntry);
groupBox_2->setObjectName(QString::fromUtf8("groupBox_2"));
groupBox_2->setMinimumSize(QSize(0, 166));
gridLayout2 = new QGridLayout(groupBox_2);
gridLayout2->setObjectName(QString::fromUtf8("gridLayout2"));
label_5 = new QLabel(groupBox_2);
label_5->setObjectName(QString::fromUtf8("label_5"));

gridLayout2->addWidget(label_5, 0, 0, 1, 1);

sig_type = new QComboBox(groupBox_2);
sig_type->setObjectName(QString::fromUtf8("sig_type"));

gridLayout2->addWidget(sig_type, 0, 1, 1, 1);

label_6 = new QLabel(groupBox_2);
label_6->setObjectName(QString::fromUtf8("label_6"));

gridLayout2->addWidget(label_6, 1, 0, 1, 1);

sig_dly = new QLineEdit(groupBox_2);
sig_dly->setObjectName(QString::fromUtf8("sig_dly"));

gridLayout2->addWidget(sig_dly, 1, 1, 1, 1);

label_7 = new QLabel(groupBox_2);
label_7->setObjectName(QString::fromUtf8("label_7"));

gridLayout2->addWidget(label_7, 2, 0, 1, 1);

sig_f = new QLineEdit(groupBox_2);
sig_f->setObjectName(QString::fromUtf8("sig_f"));

gridLayout2->addWidget(sig_f, 2, 1, 1, 1);

label_8 = new QLabel(groupBox_2);
label_8->setObjectName(QString::fromUtf8("label_8"));

gridLayout2->addWidget(label_8, 3, 0, 1, 1);

sig_gwidth = new QLineEdit(groupBox_2);
sig_gwidth->setObjectName(QString::fromUtf8("sig_gwidth"));

gridLayout2->addWidget(sig_gwidth, 3, 1, 1, 1);


gridLayout->addWidget(groupBox_2, 0, 1, 1, 1);

hboxLayout = new QHBoxLayout();
hboxLayout->setObjectName(QString::fromUtf8("hboxLayout"));
add = new QPushButton(tdEntry);
add->setObjectName(QString::fromUtf8("add"));

hboxLayout->addWidget(add);

done = new QPushButton(tdEntry);
done->setObjectName(QString::fromUtf8("done"));

hboxLayout->addWidget(done);


gridLayout->addLayout(hboxLayout, 1, 1, 1, 1);


retranslateUi(tdEntry);
QObject::connect(done, SIGNAL(clicked()), tdEntry, SLOT(close()));
```

```
    QMetaObject::connectSlotsByName(tdEntry);
    } // setupUi

    void retranslateUi(QDialog *tdEntry)
    {
    tdEntry->setWindowTitle(QApplication::translate("tdEntry", "Transducer Entry", 0, QApplication::UnicodeUTF8));
    groupBox->setTitle(QApplication::translate("tdEntry", "Transducer Information", 0, QApplication::UnicodeUTF8));
    label_2->setText(QApplication::translate("tdEntry", "Type", 0, QApplication::UnicodeUTF8));
    label->setText(QApplication::translate("tdEntry", "Size", 0, QApplication::UnicodeUTF8));
    label_3->setText(QApplication::translate("tdEntry", "Loc X", 0, QApplication::UnicodeUTF8));
    label_4->setText(QApplication::translate("tdEntry", "Loc Y", 0, QApplication::UnicodeUTF8));
    groupBox_2->setTitle(QApplication::translate("tdEntry", "Impulse", 0, QApplication::UnicodeUTF8));
    label_5->setText(QApplication::translate("tdEntry", "Signal", 0, QApplication::UnicodeUTF8));
    label_6->setText(QApplication::translate("tdEntry", "Delay", 0, QApplication::UnicodeUTF8));
    label_7->setText(QApplication::translate("tdEntry", "Frequency", 0, QApplication::UnicodeUTF8));
    label_8->setText(QApplication::translate("tdEntry", "Gaus. Wi.", 0, QApplication::UnicodeUTF8));
    add->setText(QApplication::translate("tdEntry", "Add", 0, QApplication::UnicodeUTF8));
    done->setText(QApplication::translate("tdEntry", "Done", 0, QApplication::UnicodeUTF8));
    Q_UNUSED(tdEntry);
    } // retranslateUi

};

namespace Ui {
    class tdEntry: public Ui_tdEntry {};
} // namespace Ui

#endif // UI_TD_H
```

D.13 ui_tdimport.h

```
/********************************************************************************
** Form generated from reading ui file 'tdimport.ui'
**
** Created: Thu Nov 1 16:08:39 2007
**      by: Qt User Interface Compiler version 4.3.2
**
** WARNING! All changes made in this file will be lost when recompiling ui file!
********************************************************************************/

#ifndef UI_TDIMPORT_H
#define UI_TDIMPORT_H

#include <QtCore/QVariant>
#include <QtGui/QAction>
#include <QtGui/QApplication>
#include <QtGui/QButtonGroup>
#include <QtGui/QComboBox>
#include <QtGui/QDialog>
#include <QtGui/QGridLayout>
#include <QtGui/QGroupBox>
#include <QtGui/QHBoxLayout>
#include <QtGui/QLabel>
#include <QtGui/QLineEdit>
#include <QtGui/QListView>
#include <QtGui/QPushButton>
#include <QtGui/QSpacerItem>
#include <QtGui/QVBoxLayout>
#include <QtGui/QWidget>

class Ui_tdImport
{
public:
    QGroupBox *groupBox_5;
    QGridLayout *gridLayout;
    QVBoxLayout *vboxLayout;
```

```
QLabel *label_10;
QListView *elem_list;
QSpacerItem *spacerItem;
QHBoxLayout *hboxLayout;
QLabel *label_34;
QLineEdit *elem_flGain;
QHBoxLayout *hboxLayout1;
QLabel *label_36;
QLineEdit *elem_tDelay;
QHBoxLayout *hboxLayout2;
QLabel *label_38;
QLineEdit *elem_rDelay;
QHBoxLayout *hboxLayout3;
QLabel *label_40;
QLineEdit *elem_amplitude;
QHBoxLayout *hboxLayout4;
QLabel *label_42;
QLineEdit *elem_pWidth;
QPushButton *accept;
QWidget *layoutWidget;
QVBoxLayout *vboxLayout1;
QHBoxLayout *hboxLayout5;
QPushButton *law_import;
QLineEdit *law_filename;
QGroupBox *groupBox_4;
QGridLayout *gridLayout1;
QHBoxLayout *hboxLayout6;
QLabel *label_9;
QComboBox *law_num;
QHBoxLayout *hboxLayout7;
QLabel *label_11;
QLineEdit *law_fDepth;
QHBoxLayout *hboxLayout8;
QLabel *label_33;
QLineEdit *law_scanOffset;
QHBoxLayout *hboxLayout9;
QLabel *label_13;
QLineEdit *law_freq;
QHBoxLayout *hboxLayout10;
QLabel *label_27;
QLineEdit *law_sAngle;
QHBoxLayout *hboxLayout11;
QLabel *label_16;
QLineEdit *law_filter;
QHBoxLayout *hboxLayout12;
QLabel *label_12;
QLineEdit *law_cycles;
QHBoxLayout *hboxLayout13;
QLabel *label_35;
QLineEdit *law_tFirst;
QHBoxLayout *hboxLayout14;
QLabel *label_41;
QLineEdit *law_medVel;
QHBoxLayout *hboxLayout15;
QLabel *label_15;
QLineEdit *law_sumgain;
QHBoxLayout *hboxLayout16;
QLabel *label_21;
QLineEdit *law_rFirst;
QHBoxLayout *hboxLayout17;
QLabel *label_37;
QLineEdit *law_indexOffset;
QHBoxLayout *hboxLayout18;
QLabel *label_39;
QLineEdit *law_gDelay;
QHBoxLayout *hboxLayout19;
QLabel *label_25;
QLineEdit *law_rAngle;
```

```
QHBoxLayout *hboxLayout20;
QLabel *label_22;
QLineEdit *law_mode;
QHBoxLayout *hboxLayout21;
QGroupBox *groupBox;
QGridLayout *gridLayout2;
QLabel *label_2;
QComboBox *td_type;
QLabel *label;
QLineEdit *td_sz;
QLabel *label_3;
QLineEdit *td_dx;
QGroupBox *groupBox_2;
QGridLayout *gridLayout3;
QLabel *label_5;
QComboBox *sig_type;
QLabel *label_6;
QLineEdit *sig_dly;

void setupUi(QDialog *tdImport)
{
if (tdImport->objectName().isEmpty())
    tdImport->setObjectName(QString::fromUtf8("tdImport"));
tdImport->setWindowModality(Qt::NonModal);
tdImport->resize(820, 500);
QSizePolicy sizePolicy(QSizePolicy::MinimumExpanding, QSizePolicy::MinimumExpanding);
sizePolicy.setHorizontalStretch(0);
sizePolicy.setVerticalStretch(0);
sizePolicy.setHeightForWidth(tdImport->sizePolicy().hasHeightForWidth());
tdImport->setSizePolicy(sizePolicy);
tdImport->setMinimumSize(QSize(820, 500));
tdImport->setMaximumSize(QSize(820, 500));
tdImport->setBaseSize(QSize(320, 200));
tdImport->setModal(false);
groupBox_5 = new QGroupBox(tdImport);
groupBox_5->setObjectName(QString::fromUtf8("groupBox_5"));
groupBox_5->setGeometry(QRect(510, 72, 281, 200));
groupBox_5->setMinimumSize(QSize(0, 200));
groupBox_5->setMaximumSize(QSize(16777215, 200));
groupBox_5->setSizeIncrement(QSize(0, 200));
gridLayout = new QGridLayout(groupBox_5);
gridLayout->setObjectName(QString::fromUtf8("gridLayout"));
vboxLayout = new QVBoxLayout();
vboxLayout->setObjectName(QString::fromUtf8("vboxLayout"));
label_10 = new QLabel(groupBox_5);
label_10->setObjectName(QString::fromUtf8("label_10"));
label_10->setMaximumSize(QSize(80, 16777215));
label_10->setAlignment(Qt::AlignCenter);

vboxLayout->addWidget(label_10);

elem_list = new QListView(groupBox_5);
elem_list->setObjectName(QString::fromUtf8("elem_list"));
elem_list->setMaximumSize(QSize(80, 16777215));

vboxLayout->addWidget(elem_list);


gridLayout->addLayout(vboxLayout, 0, 0, 6, 1);

spacerItem = new QSpacerItem(141, 20, QSizePolicy::Expanding, QSizePolicy::Minimum);

gridLayout->addItem(spacerItem, 0, 1, 1, 1);

hboxLayout = new QHBoxLayout();
hboxLayout->setObjectName(QString::fromUtf8("hboxLayout"));
label_34 = new QLabel(groupBox_5);
label_34->setObjectName(QString::fromUtf8("label_34"));
```

```
label_34->setMinimumSize(QSize(80, 0));
label_34->setMaximumSize(QSize(80, 16777215));

hboxLayout->addWidget(label_34);

elem_flGain = new QLineEdit(groupBox_5);
elem_flGain->setObjectName(QString::fromUtf8("elem_flGain"));
elem_flGain->setMinimumSize(QSize(70, 0));
elem_flGain->setMaximumSize(QSize(70, 16777215));

hboxLayout->addWidget(elem_flGain);


gridLayout->addLayout(hboxLayout, 1, 1, 1, 1);

hboxLayout1 = new QHBoxLayout();
hboxLayout1->setObjectName(QString::fromUtf8("hboxLayout1"));
label_36 = new QLabel(groupBox_5);
label_36->setObjectName(QString::fromUtf8("label_36"));
label_36->setMinimumSize(QSize(80, 0));
label_36->setMaximumSize(QSize(80, 16777215));

hboxLayout1->addWidget(label_36);

elem_tDelay = new QLineEdit(groupBox_5);
elem_tDelay->setObjectName(QString::fromUtf8("elem_tDelay"));
elem_tDelay->setMinimumSize(QSize(70, 0));
elem_tDelay->setMaximumSize(QSize(70, 16777215));

hboxLayout1->addWidget(elem_tDelay);


gridLayout->addLayout(hboxLayout1, 2, 1, 1, 1);

hboxLayout2 = new QHBoxLayout();
hboxLayout2->setObjectName(QString::fromUtf8("hboxLayout2"));
label_38 = new QLabel(groupBox_5);
label_38->setObjectName(QString::fromUtf8("label_38"));
label_38->setMinimumSize(QSize(80, 0));
label_38->setMaximumSize(QSize(80, 16777215));

hboxLayout2->addWidget(label_38);

elem_rDelay = new QLineEdit(groupBox_5);
elem_rDelay->setObjectName(QString::fromUtf8("elem_rDelay"));
elem_rDelay->setMinimumSize(QSize(70, 0));
elem_rDelay->setMaximumSize(QSize(70, 16777215));

hboxLayout2->addWidget(elem_rDelay);


gridLayout->addLayout(hboxLayout2, 3, 1, 1, 1);

hboxLayout3 = new QHBoxLayout();
hboxLayout3->setObjectName(QString::fromUtf8("hboxLayout3"));
label_40 = new QLabel(groupBox_5);
label_40->setObjectName(QString::fromUtf8("label_40"));
label_40->setMinimumSize(QSize(80, 0));
label_40->setMaximumSize(QSize(80, 16777215));

hboxLayout3->addWidget(label_40);

elem_amplitude = new QLineEdit(groupBox_5);
elem_amplitude->setObjectName(QString::fromUtf8("elem_amplitude"));
elem_amplitude->setMinimumSize(QSize(70, 0));
elem_amplitude->setMaximumSize(QSize(70, 16777215));

hboxLayout3->addWidget(elem_amplitude);
```

```
gridLayout->addLayout(hboxLayout3, 4, 1, 1, 1);

hboxLayout4 = new QHBoxLayout();
hboxLayout4->setObjectName(QString::fromUtf8("hboxLayout4"));
label_42 = new QLabel(groupBox_5);
label_42->setObjectName(QString::fromUtf8("label_42"));
label_42->setMinimumSize(QSize(80, 0));
label_42->setMaximumSize(QSize(80, 16777215));

hboxLayout4->addWidget(label_42);

elem_pWidth = new QLineEdit(groupBox_5);
elem_pWidth->setObjectName(QString::fromUtf8("elem_pWidth"));
elem_pWidth->setMinimumSize(QSize(70, 0));
elem_pWidth->setMaximumSize(QSize(70, 16777215));

hboxLayout4->addWidget(elem_pWidth);


gridLayout->addLayout(hboxLayout4, 5, 1, 1, 1);

accept = new QPushButton(tdImport);
accept->setObjectName(QString::fromUtf8("accept"));
accept->setGeometry(QRect(680, 440, 111, 32));
layoutWidget = new QWidget(tdImport);
layoutWidget->setObjectName(QString::fromUtf8("layoutWidget"));
layoutWidget->setGeometry(QRect(10, 10, 498, 471));
vboxLayout1 = new QVBoxLayout(layoutWidget);
vboxLayout1->setObjectName(QString::fromUtf8("vboxLayout1"));
vboxLayout1->setContentsMargins(0, 0, 0, 0);
hboxLayout5 = new QHBoxLayout();
hboxLayout5->setObjectName(QString::fromUtf8("hboxLayout5"));
law_import = new QPushButton(layoutWidget);
law_import->setObjectName(QString::fromUtf8("law_import"));

hboxLayout5->addWidget(law_import);

law_filename = new QLineEdit(layoutWidget);
law_filename->setObjectName(QString::fromUtf8("law_filename"));

hboxLayout5->addWidget(law_filename);


vboxLayout1->addLayout(hboxLayout5);

groupBox_4 = new QGroupBox(layoutWidget);
groupBox_4->setObjectName(QString::fromUtf8("groupBox_4"));
groupBox_4->setMinimumSize(QSize(495, 200));
groupBox_4->setMaximumSize(QSize(495, 200));
gridLayout1 = new QGridLayout(groupBox_4);
gridLayout1->setObjectName(QString::fromUtf8("gridLayout1"));
hboxLayout6 = new QHBoxLayout();
hboxLayout6->setObjectName(QString::fromUtf8("hboxLayout6"));
label_9 = new QLabel(groupBox_4);
label_9->setObjectName(QString::fromUtf8("label_9"));
label_9->setMinimumSize(QSize(70, 0));
label_9->setMaximumSize(QSize(70, 16777215));
label_9->setAlignment(Qt::AlignCenter);

hboxLayout6->addWidget(label_9);

law_num = new QComboBox(groupBox_4);
law_num->setObjectName(QString::fromUtf8("law_num"));
law_num->setMinimumSize(QSize(60, 0));
law_num->setMaximumSize(QSize(60, 16777215));
```

```
hboxLayout6->addWidget(law_num);


gridLayout1->addLayout(hboxLayout6, 0, 0, 1, 1);

hboxLayout7 = new QHBoxLayout();
hboxLayout7->setObjectName(QString::fromUtf8("hboxLayout7"));
label_11 = new QLabel(groupBox_4);
label_11->setObjectName(QString::fromUtf8("label_11"));
label_11->setMinimumSize(QSize(55, 0));
label_11->setMaximumSize(QSize(55, 16777215));

hboxLayout7->addWidget(label_11);

law_fDepth = new QLineEdit(groupBox_4);
law_fDepth->setObjectName(QString::fromUtf8("law_fDepth"));
law_fDepth->setMinimumSize(QSize(70, 0));
law_fDepth->setMaximumSize(QSize(70, 16777215));

hboxLayout7->addWidget(law_fDepth);


gridLayout1->addLayout(hboxLayout7, 0, 1, 1, 1);

hboxLayout8 = new QHBoxLayout();
hboxLayout8->setObjectName(QString::fromUtf8("hboxLayout8"));
label_33 = new QLabel(groupBox_4);
label_33->setObjectName(QString::fromUtf8("label_33"));
label_33->setMinimumSize(QSize(80, 0));
label_33->setMaximumSize(QSize(80, 16777215));

hboxLayout8->addWidget(label_33);

law_scanOffset = new QLineEdit(groupBox_4);
law_scanOffset->setObjectName(QString::fromUtf8("law_scanOffset"));
law_scanOffset->setMinimumSize(QSize(70, 0));
law_scanOffset->setMaximumSize(QSize(70, 16777215));

hboxLayout8->addWidget(law_scanOffset);


gridLayout1->addLayout(hboxLayout8, 0, 2, 1, 1);

hboxLayout9 = new QHBoxLayout();
hboxLayout9->setObjectName(QString::fromUtf8("hboxLayout9"));
label_13 = new QLabel(groupBox_4);
label_13->setObjectName(QString::fromUtf8("label_13"));
label_13->setMinimumSize(QSize(55, 0));
label_13->setMaximumSize(QSize(55, 16777215));

hboxLayout9->addWidget(label_13);

law_freq = new QLineEdit(groupBox_4);
law_freq->setObjectName(QString::fromUtf8("law_freq"));
law_freq->setMinimumSize(QSize(70, 0));
law_freq->setMaximumSize(QSize(70, 16777215));

hboxLayout9->addWidget(law_freq);


gridLayout1->addLayout(hboxLayout9, 1, 0, 1, 1);

hboxLayout10 = new QHBoxLayout();
hboxLayout10->setObjectName(QString::fromUtf8("hboxLayout10"));
label_27 = new QLabel(groupBox_4);
label_27->setObjectName(QString::fromUtf8("label_27"));
label_27->setMinimumSize(QSize(55, 0));
label_27->setMaximumSize(QSize(55, 16777215));
```

```
hboxLayout10->addWidget(label_27);

law_sAngle = new QLineEdit(groupBox_4);
law_sAngle->setObjectName(QString::fromUtf8("law_sAngle"));
law_sAngle->setMinimumSize(QSize(70, 0));
law_sAngle->setMaximumSize(QSize(70, 16777215));

hboxLayout10->addWidget(law_sAngle);


gridLayout1->addLayout(hboxLayout10, 1, 1, 1, 1);

hboxLayout11 = new QHBoxLayout();
hboxLayout11->setObjectName(QString::fromUtf8("hboxLayout11"));
label_16 = new QLabel(groupBox_4);
label_16->setObjectName(QString::fromUtf8("label_16"));
label_16->setMinimumSize(QSize(80, 0));
label_16->setMaximumSize(QSize(80, 16777215));

hboxLayout11->addWidget(label_16);

law_filter = new QLineEdit(groupBox_4);
law_filter->setObjectName(QString::fromUtf8("law_filter"));
law_filter->setMinimumSize(QSize(70, 0));
law_filter->setMaximumSize(QSize(70, 16777215));

hboxLayout11->addWidget(law_filter);


gridLayout1->addLayout(hboxLayout11, 1, 2, 1, 1);

hboxLayout12 = new QHBoxLayout();
hboxLayout12->setObjectName(QString::fromUtf8("hboxLayout12"));
label_12 = new QLabel(groupBox_4);
label_12->setObjectName(QString::fromUtf8("label_12"));
label_12->setMinimumSize(QSize(55, 0));
label_12->setMaximumSize(QSize(55, 16777215));

hboxLayout12->addWidget(label_12);

law_cycles = new QLineEdit(groupBox_4);
law_cycles->setObjectName(QString::fromUtf8("law_cycles"));
law_cycles->setMinimumSize(QSize(70, 0));
law_cycles->setMaximumSize(QSize(70, 16777215));

hboxLayout12->addWidget(law_cycles);


gridLayout1->addLayout(hboxLayout12, 2, 0, 1, 1);

hboxLayout13 = new QHBoxLayout();
hboxLayout13->setObjectName(QString::fromUtf8("hboxLayout13"));
label_35 = new QLabel(groupBox_4);
label_35->setObjectName(QString::fromUtf8("label_35"));
label_35->setMinimumSize(QSize(55, 0));
label_35->setMaximumSize(QSize(55, 16777215));

hboxLayout13->addWidget(label_35);

law_tFirst = new QLineEdit(groupBox_4);
law_tFirst->setObjectName(QString::fromUtf8("law_tFirst"));
law_tFirst->setMinimumSize(QSize(70, 0));
law_tFirst->setMaximumSize(QSize(70, 16777215));

hboxLayout13->addWidget(law_tFirst);
```

```
gridLayout1->addLayout(hboxLayout13, 2, 1, 1, 1);

hboxLayout14 = new QHBoxLayout();
hboxLayout14->setObjectName(QString::fromUtf8("hboxLayout14"));
label_41 = new QLabel(groupBox_4);
label_41->setObjectName(QString::fromUtf8("label_41"));
label_41->setMinimumSize(QSize(80, 0));
label_41->setMaximumSize(QSize(80, 16777215));

hboxLayout14->addWidget(label_41);

law_medVel = new QLineEdit(groupBox_4);
law_medVel->setObjectName(QString::fromUtf8("law_medVel"));
law_medVel->setMinimumSize(QSize(70, 0));
law_medVel->setMaximumSize(QSize(70, 16777215));

hboxLayout14->addWidget(law_medVel);


gridLayout1->addLayout(hboxLayout14, 2, 2, 1, 1);

hboxLayout15 = new QHBoxLayout();
hboxLayout15->setObjectName(QString::fromUtf8("hboxLayout15"));
label_15 = new QLabel(groupBox_4);
label_15->setObjectName(QString::fromUtf8("label_15"));
label_15->setMinimumSize(QSize(55, 0));
label_15->setMaximumSize(QSize(55, 16777215));

hboxLayout15->addWidget(label_15);

law_sumgain = new QLineEdit(groupBox_4);
law_sumgain->setObjectName(QString::fromUtf8("law_sumgain"));
law_sumgain->setMinimumSize(QSize(70, 0));
law_sumgain->setMaximumSize(QSize(70, 16777215));

hboxLayout15->addWidget(law_sumgain);


gridLayout1->addLayout(hboxLayout15, 3, 0, 1, 1);

hboxLayout16 = new QHBoxLayout();
hboxLayout16->setObjectName(QString::fromUtf8("hboxLayout16"));
label_21 = new QLabel(groupBox_4);
label_21->setObjectName(QString::fromUtf8("label_21"));
label_21->setMinimumSize(QSize(55, 0));
label_21->setMaximumSize(QSize(55, 16777215));

hboxLayout16->addWidget(label_21);

law_rFirst = new QLineEdit(groupBox_4);
law_rFirst->setObjectName(QString::fromUtf8("law_rFirst"));
law_rFirst->setMinimumSize(QSize(70, 0));
law_rFirst->setMaximumSize(QSize(70, 16777215));

hboxLayout16->addWidget(law_rFirst);


gridLayout1->addLayout(hboxLayout16, 3, 1, 1, 1);

hboxLayout17 = new QHBoxLayout();
hboxLayout17->setObjectName(QString::fromUtf8("hboxLayout17"));
label_37 = new QLabel(groupBox_4);
label_37->setObjectName(QString::fromUtf8("label_37"));
label_37->setMinimumSize(QSize(80, 0));

hboxLayout17->addWidget(label_37);

law_indexOffset = new QLineEdit(groupBox_4);
```

```
law_indexOffset->setObjectName(QString::fromUtf8("law_indexOffset"));
law_indexOffset->setMinimumSize(QSize(70, 0));
law_indexOffset->setMaximumSize(QSize(70, 16777215));

hboxLayout17->addWidget(law_indexOffset);


gridLayout1->addLayout(hboxLayout17, 3, 2, 1, 1);

hboxLayout18 = new QHBoxLayout();
hboxLayout18->setObjectName(QString::fromUtf8("hboxLayout18"));
label_39 = new QLabel(groupBox_4);
label_39->setObjectName(QString::fromUtf8("label_39"));
label_39->setMinimumSize(QSize(55, 0));
label_39->setMaximumSize(QSize(55, 16777215));

hboxLayout18->addWidget(label_39);

law_gDelay = new QLineEdit(groupBox_4);
law_gDelay->setObjectName(QString::fromUtf8("law_gDelay"));
law_gDelay->setMinimumSize(QSize(70, 0));
law_gDelay->setMaximumSize(QSize(70, 16777215));

hboxLayout18->addWidget(law_gDelay);


gridLayout1->addLayout(hboxLayout18, 4, 0, 1, 1);

hboxLayout19 = new QHBoxLayout();
hboxLayout19->setObjectName(QString::fromUtf8("hboxLayout19"));
label_25 = new QLabel(groupBox_4);
label_25->setObjectName(QString::fromUtf8("label_25"));
label_25->setMinimumSize(QSize(55, 0));
label_25->setMaximumSize(QSize(55, 16777215));

hboxLayout19->addWidget(label_25);

law_rAngle = new QLineEdit(groupBox_4);
law_rAngle->setObjectName(QString::fromUtf8("law_rAngle"));
law_rAngle->setMinimumSize(QSize(70, 0));
law_rAngle->setMaximumSize(QSize(70, 16777215));

hboxLayout19->addWidget(law_rAngle);


gridLayout1->addLayout(hboxLayout19, 4, 1, 1, 1);

hboxLayout20 = new QHBoxLayout();
hboxLayout20->setObjectName(QString::fromUtf8("hboxLayout20"));
label_22 = new QLabel(groupBox_4);
label_22->setObjectName(QString::fromUtf8("label_22"));
label_22->setMinimumSize(QSize(80, 0));
label_22->setMaximumSize(QSize(80, 16777215));

hboxLayout20->addWidget(label_22);

law_mode = new QLineEdit(groupBox_4);
law_mode->setObjectName(QString::fromUtf8("law_mode"));
law_mode->setMinimumSize(QSize(70, 0));
law_mode->setMaximumSize(QSize(70, 16777215));

hboxLayout20->addWidget(law_mode);


gridLayout1->addLayout(hboxLayout20, 4, 2, 1, 1);


vboxLayout1->addWidget(groupBox_4);
```

```
hboxLayout21 = new QHBoxLayout();
hboxLayout21->setObjectName(QString::fromUtf8("hboxLayout21"));
groupBox = new QGroupBox(layoutWidget);
groupBox->setObjectName(QString::fromUtf8("groupBox"));
groupBox->setMinimumSize(QSize(232, 166));
groupBox->setMaximumSize(QSize(232, 16777215));
gridLayout2 = new QGridLayout(groupBox);
gridLayout2->setObjectName(QString::fromUtf8("gridLayout2"));
label_2 = new QLabel(groupBox);
label_2->setObjectName(QString::fromUtf8("label_2"));

gridLayout2->addWidget(label_2, 0, 0, 1, 1);

td_type = new QComboBox(groupBox);
td_type->setObjectName(QString::fromUtf8("td_type"));

gridLayout2->addWidget(td_type, 0, 1, 1, 1);

label = new QLabel(groupBox);
label->setObjectName(QString::fromUtf8("label"));

gridLayout2->addWidget(label, 1, 0, 1, 1);

td_sz = new QLineEdit(groupBox);
td_sz->setObjectName(QString::fromUtf8("td_sz"));

gridLayout2->addWidget(td_sz, 1, 1, 1, 1);

label_3 = new QLabel(groupBox);
label_3->setObjectName(QString::fromUtf8("label_3"));

gridLayout2->addWidget(label_3, 2, 0, 1, 1);

td_dx = new QLineEdit(groupBox);
td_dx->setObjectName(QString::fromUtf8("td_dx"));

gridLayout2->addWidget(td_dx, 2, 1, 1, 1);


hboxLayout21->addWidget(groupBox);

groupBox_2 = new QGroupBox(layoutWidget);
groupBox_2->setObjectName(QString::fromUtf8("groupBox_2"));
groupBox_2->setMinimumSize(QSize(245, 166));
groupBox_2->setMaximumSize(QSize(245, 166));
gridLayout3 = new QGridLayout(groupBox_2);
gridLayout3->setObjectName(QString::fromUtf8("gridLayout3"));
label_5 = new QLabel(groupBox_2);
label_5->setObjectName(QString::fromUtf8("label_5"));

gridLayout3->addWidget(label_5, 0, 0, 1, 1);

sig_type = new QComboBox(groupBox_2);
sig_type->setObjectName(QString::fromUtf8("sig_type"));
sig_type->setModelColumn(0);

gridLayout3->addWidget(sig_type, 0, 1, 1, 1);

label_6 = new QLabel(groupBox_2);
label_6->setObjectName(QString::fromUtf8("label_6"));

gridLayout3->addWidget(label_6, 1, 0, 1, 1);

sig_dly = new QLineEdit(groupBox_2);
sig_dly->setObjectName(QString::fromUtf8("sig_dly"));

gridLayout3->addWidget(sig_dly, 1, 1, 1, 1);
```

```
    hboxLayout21->addWidget(groupBox_2);


    vboxLayout1->addLayout(hboxLayout21);


    retranslateUi(tdImport);
    QObject::connect(accept, SIGNAL(clicked()), tdImport, SLOT(close()));

    QMetaObject::connectSlotsByName(tdImport);
    } // setupUi

    void retranslateUi(QDialog *tdImport)
    {
    tdImport->setWindowTitle(QApplication::translate("tdImport",
    "Focal Law Impulse and Tranducer Setup", 0, QApplication::UnicodeUTF8));
    tdImport->setProperty("", QVariant(QString()));
    groupBox_5->setTitle(QApplication::translate("tdImport", "Elements", 0, QApplication::UnicodeUTF8));
    label_10->setText(QApplication::translate("tdImport", "Elements", 0, QApplication::UnicodeUTF8));
    label_34->setText(QApplication::translate("tdImport", "FL Gain", 0, QApplication::UnicodeUTF8));
    label_36->setText(QApplication::translate("tdImport", "T Delay", 0, QApplication::UnicodeUTF8));
    label_38->setText(QApplication::translate("tdImport", "R Delay", 0, QApplication::UnicodeUTF8));
    label_40->setText(QApplication::translate("tdImport", "Amplitude", 0, QApplication::UnicodeUTF8));
    label_42->setText(QApplication::translate("tdImport", "P Width", 0, QApplication::UnicodeUTF8));
    accept->setText(QApplication::translate("tdImport", "Accept", 0, QApplication::UnicodeUTF8));
    law_import->setText(QApplication::translate("tdImport", "Import", 0, QApplication::UnicodeUTF8));
    groupBox_4->setTitle(QApplication::translate("tdImport", "Law Information", 0, QApplication::UnicodeUTF8));
    label_9->setText(QApplication::translate("tdImport", "Law Num", 0, QApplication::UnicodeUTF8));
    label_11->setText(QApplication::translate("tdImport", "F Depth", 0, QApplication::UnicodeUTF8));
    label_33->setText(QApplication::translate("tdImport", "Scan Offset", 0, QApplication::UnicodeUTF8));
    label_13->setText(QApplication::translate("tdImport", "Freq.", 0, QApplication::UnicodeUTF8));
    label_27->setText(QApplication::translate("tdImport", "S Angle", 0, QApplication::UnicodeUTF8));
    label_16->setText(QApplication::translate("tdImport", "Filter", 0, QApplication::UnicodeUTF8));
    label_12->setText(QApplication::translate("tdImport", "Cycles", 0, QApplication::UnicodeUTF8));
    label_35->setText(QApplication::translate("tdImport", "T First", 0, QApplication::UnicodeUTF8));
    label_41->setText(QApplication::translate("tdImport", "Med. Vel.", 0, QApplication::UnicodeUTF8));
    label_15->setText(QApplication::translate("tdImport", "SumGain", 0, QApplication::UnicodeUTF8));
    label_21->setText(QApplication::translate("tdImport", "R First", 0, QApplication::UnicodeUTF8));
    label_37->setText(QApplication::translate("tdImport", "Index Offset", 0, QApplication::UnicodeUTF8));
    label_39->setText(QApplication::translate("tdImport", "G Delay", 0, QApplication::UnicodeUTF8));
    label_25->setText(QApplication::translate("tdImport", "R Angle", 0, QApplication::UnicodeUTF8));
    label_22->setText(QApplication::translate("tdImport", "Mode", 0, QApplication::UnicodeUTF8));
    groupBox->setTitle(QApplication::translate("tdImport", "Transducer Information", 0, QApplication::UnicodeUTF8));
    label_2->setText(QApplication::translate("tdImport", "Type", 0, QApplication::UnicodeUTF8));
    label->setText(QApplication::translate("tdImport", "Size", 0, QApplication::UnicodeUTF8));
    label_3->setText(QApplication::translate("tdImport", "dx", 0, QApplication::UnicodeUTF8));
    groupBox_2->setTitle(QApplication::translate("tdImport", "Impulse", 0, QApplication::UnicodeUTF8));
    label_5->setText(QApplication::translate("tdImport", "Signal", 0, QApplication::UnicodeUTF8));
    label_6->setText(QApplication::translate("tdImport", "Delay", 0, QApplication::UnicodeUTF8));
    Q_UNUSED(tdImport);
    } // retranslateUi

};

namespace Ui {
    class tdImport: public Ui_tdImport {};
} // namespace Ui

#endif // UI_TDIMPORT_H
```

Appendix E

Numerical Data

E.1 Calibration Results

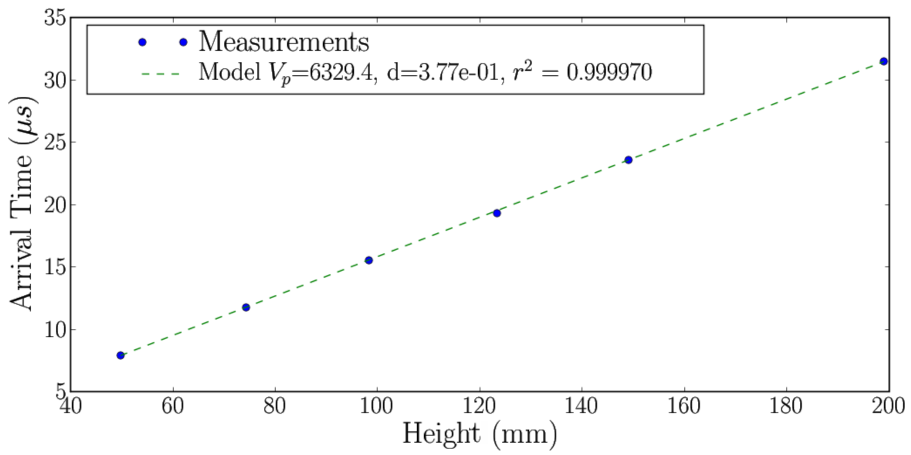


Figure E.1: Water Calibration Data for Aluminum Rods

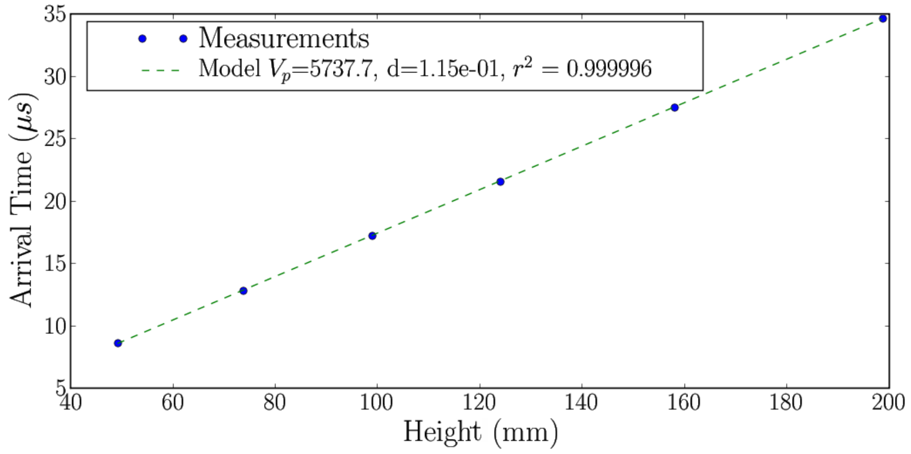


Figure E.2: Water Calibration Data for Steel Rods

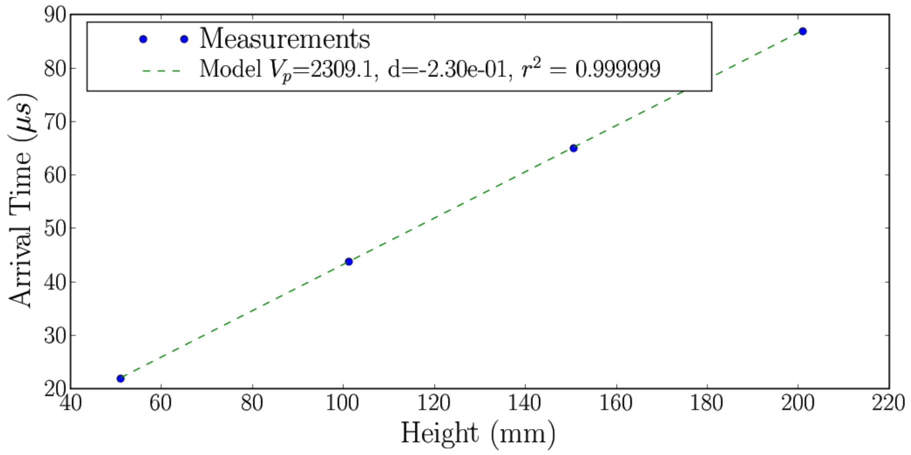


Figure E.3: Water Calibration Data for PVC Rods

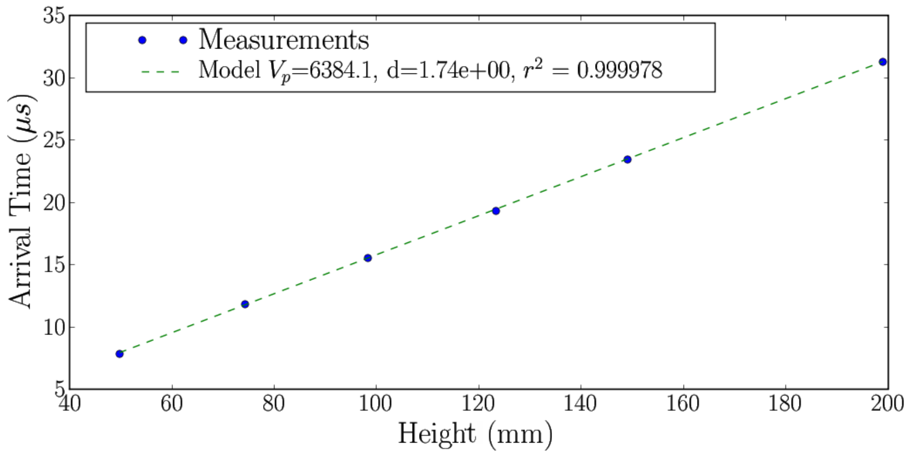


Figure E.4: Glycerin Calibration Data for Aluminum Rods

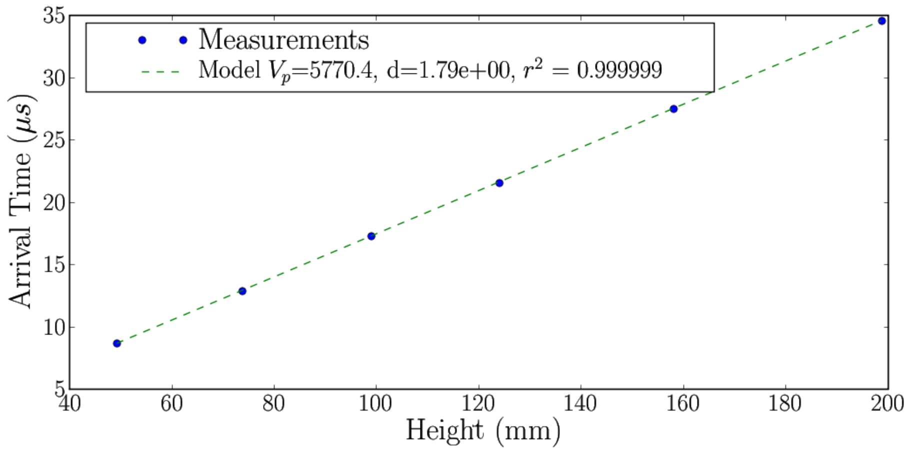


Figure E.5: Glycerin Calibration Data for Steel Rods

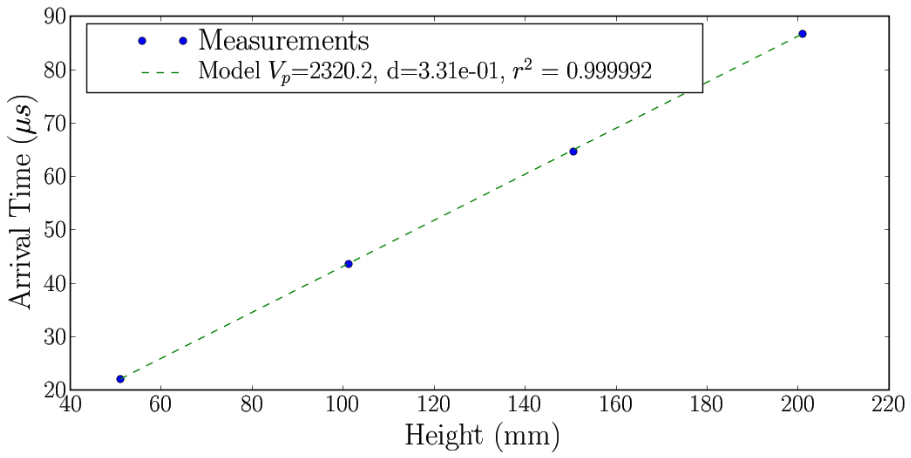


Figure E.6: Glycerin Calibration Data for PVC Rods

E.2 Numerical Simulations of Experimental Models data

E.2.1 High Frequency

Seismic Plots

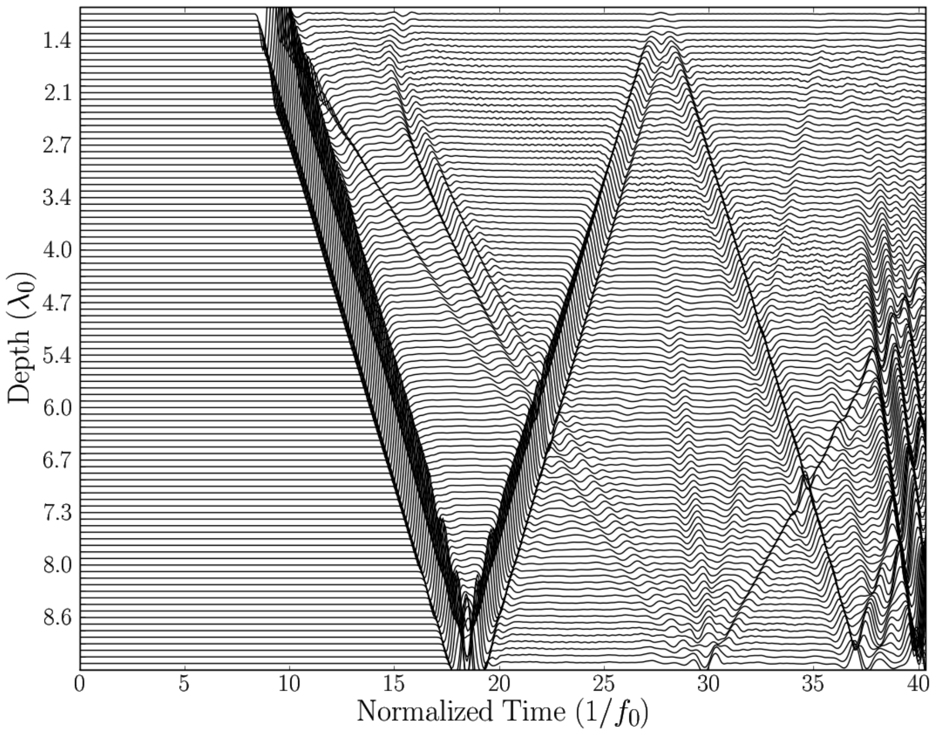


Figure E.7: High frequency seismic plot for axisymmetric axis of specimen “m1a”

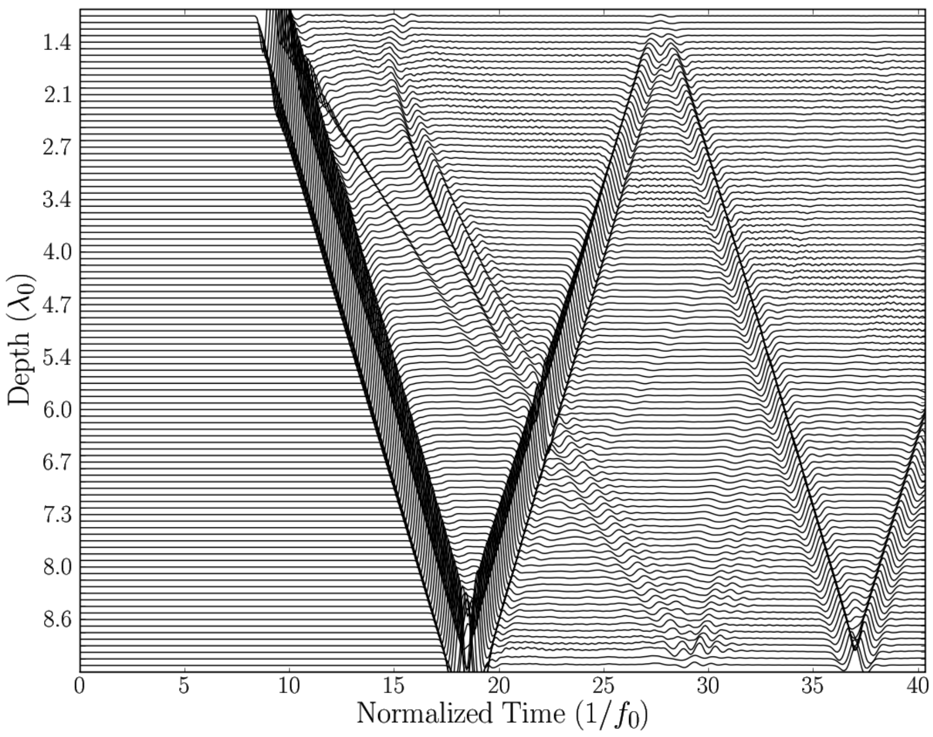


Figure E.8: High frequency seismic plot for axisymmetric axis of specimen "m1b"

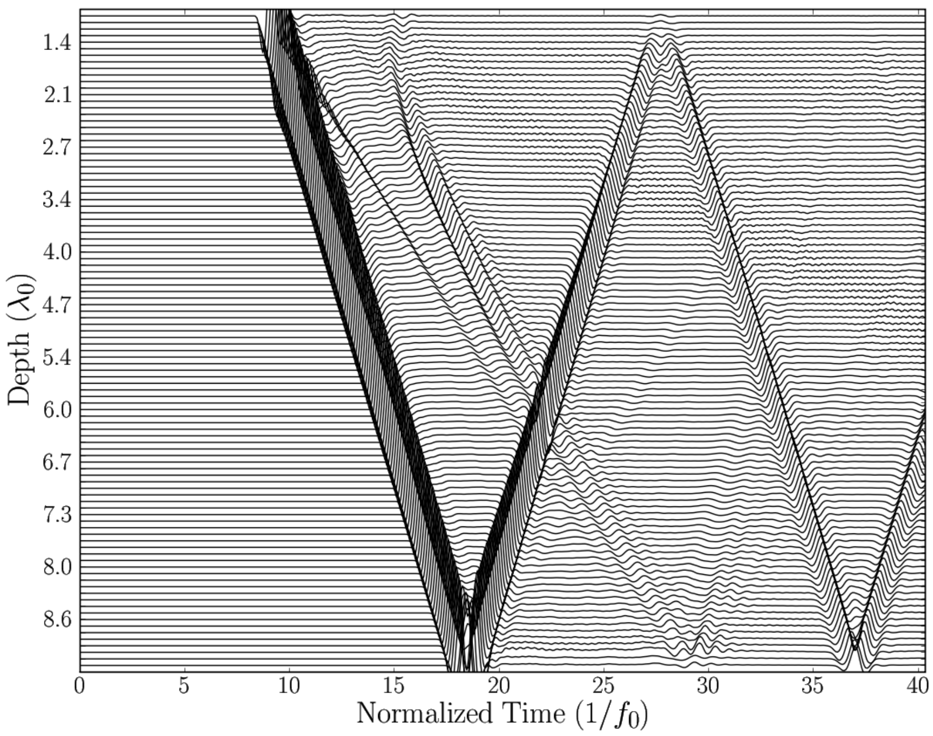


Figure E.9: High frequency seismic plot for axisymmetric axis of specimen “m1c”

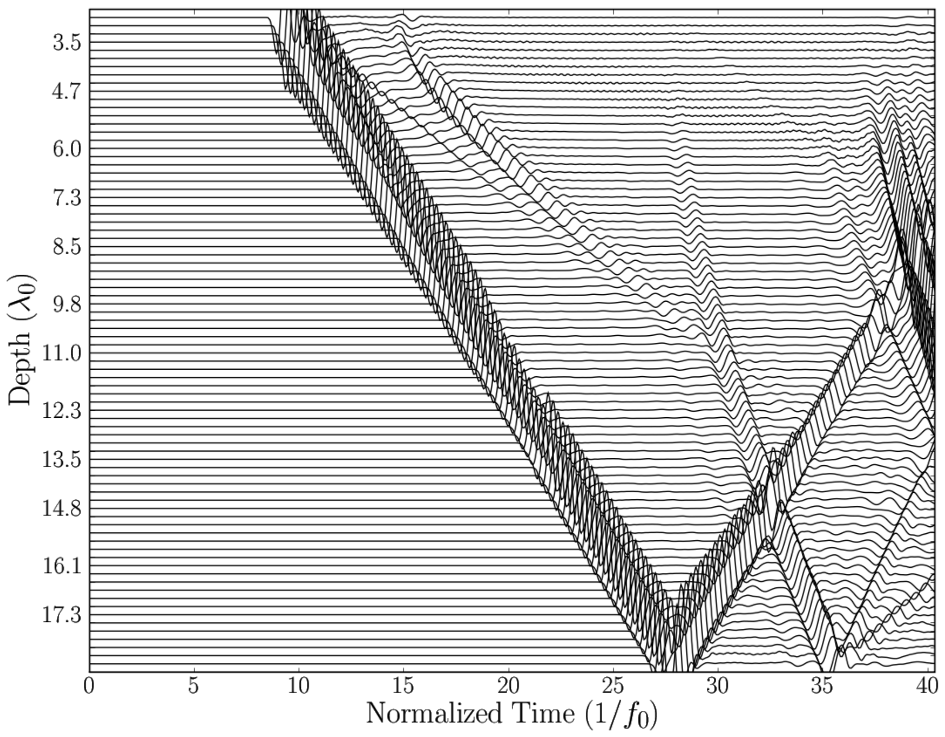


Figure E.10: High frequency seismic plot for axisymmetric axis of specimen “m2a”

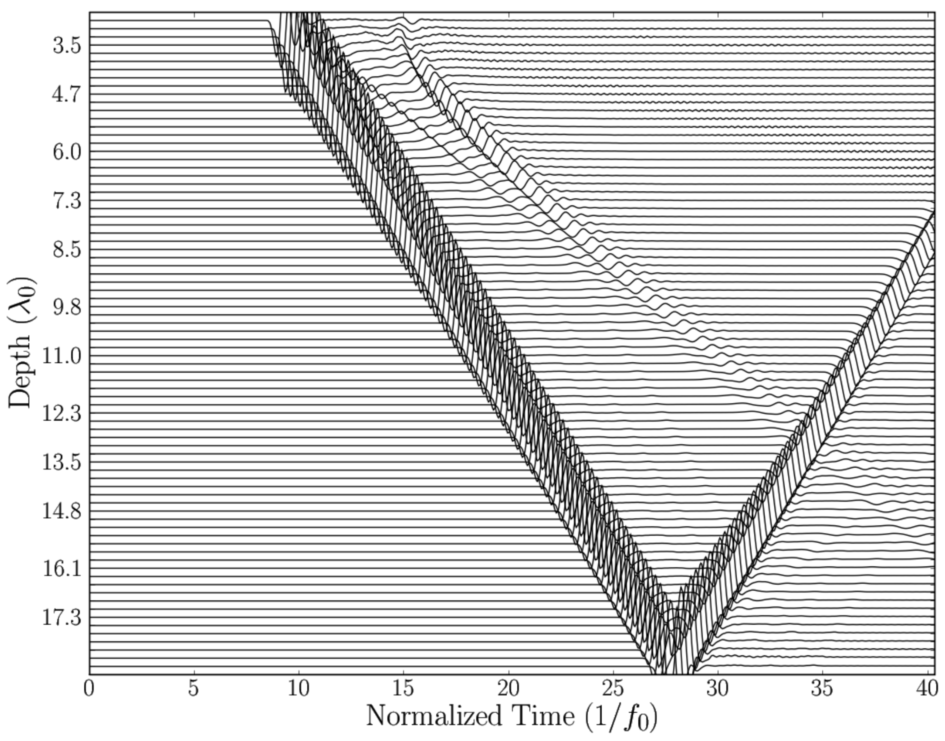


Figure E.11: High frequency seismic plot for axisymmetric axis of specimen “m2b”

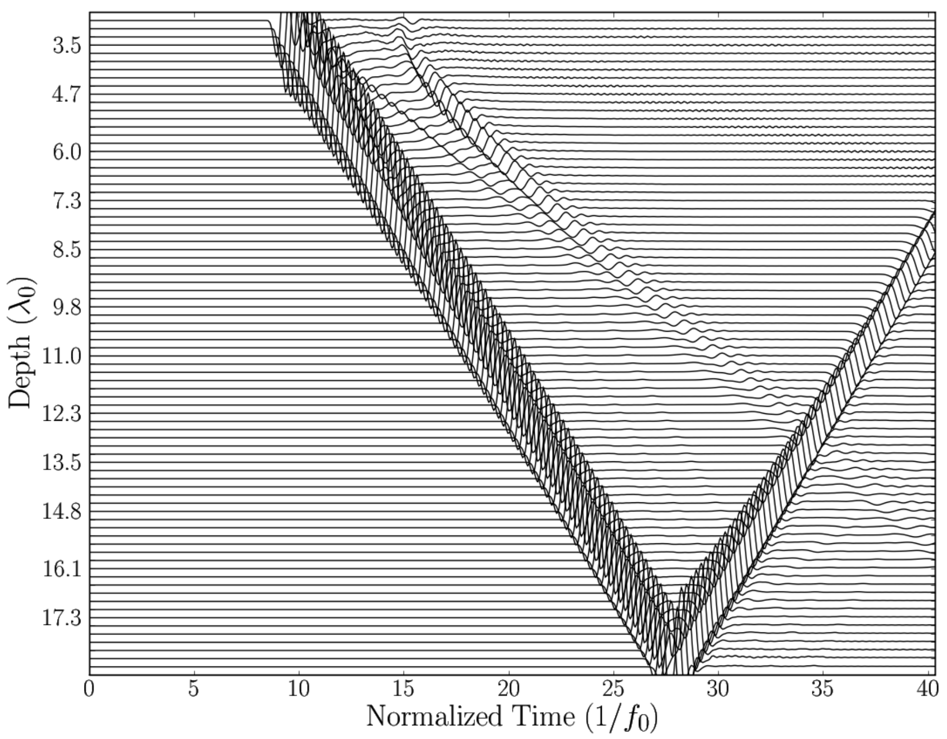


Figure E.12: High frequency seismic plot for axisymmetric axis of specimen “m2c”

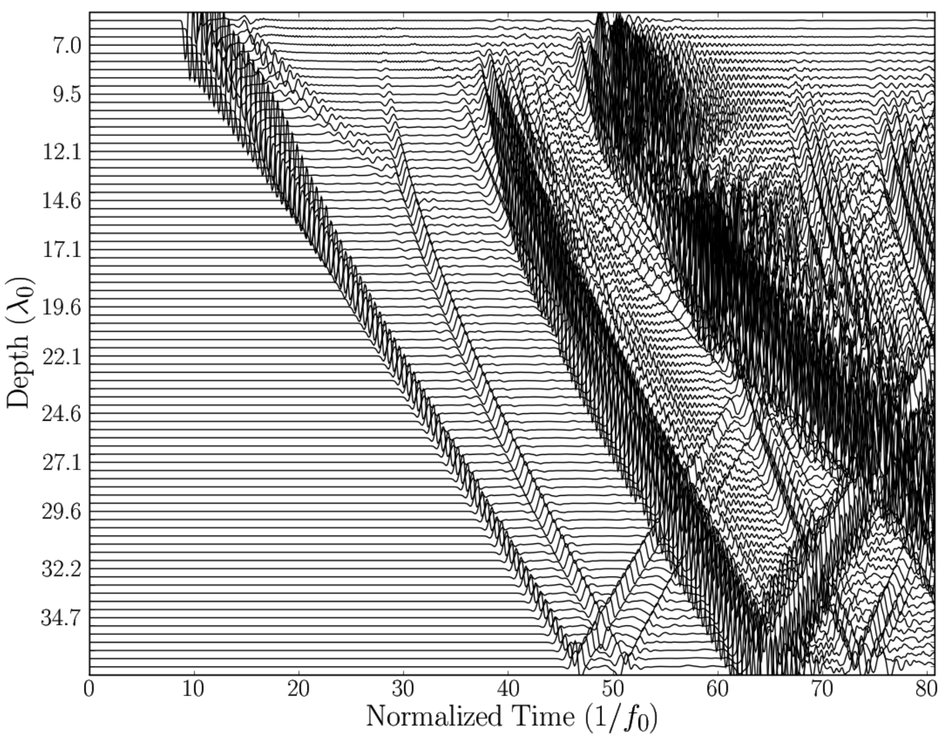


Figure E.13: High frequency seismic plot for axisymmetric axis of specimen “m3a”

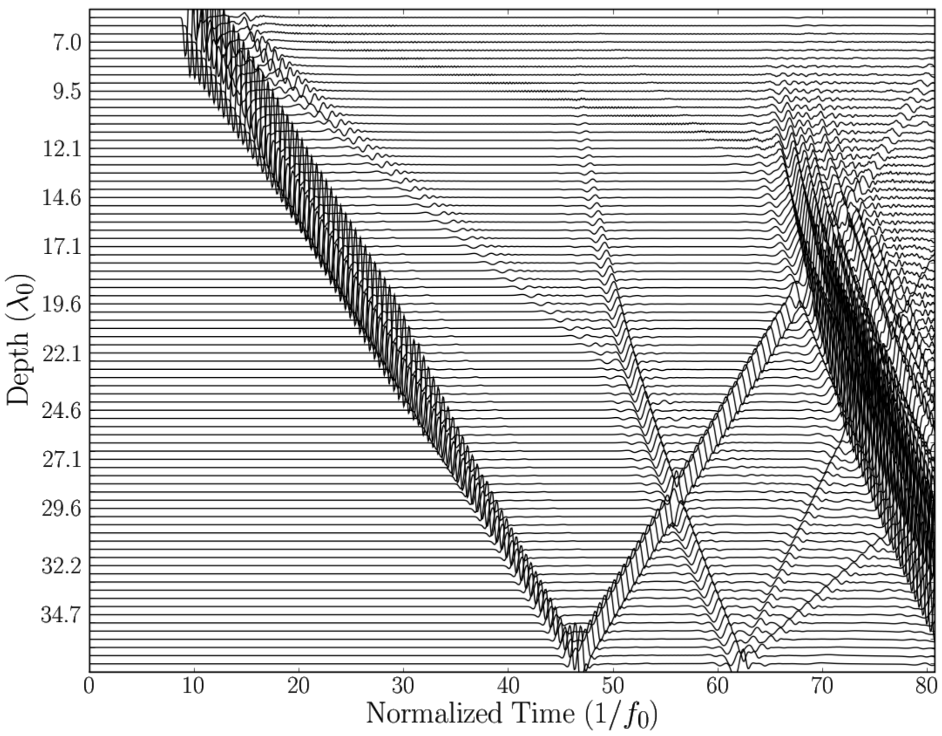


Figure E.14: High frequency seismic plot for axisymmetric axis of specimen “m3b”

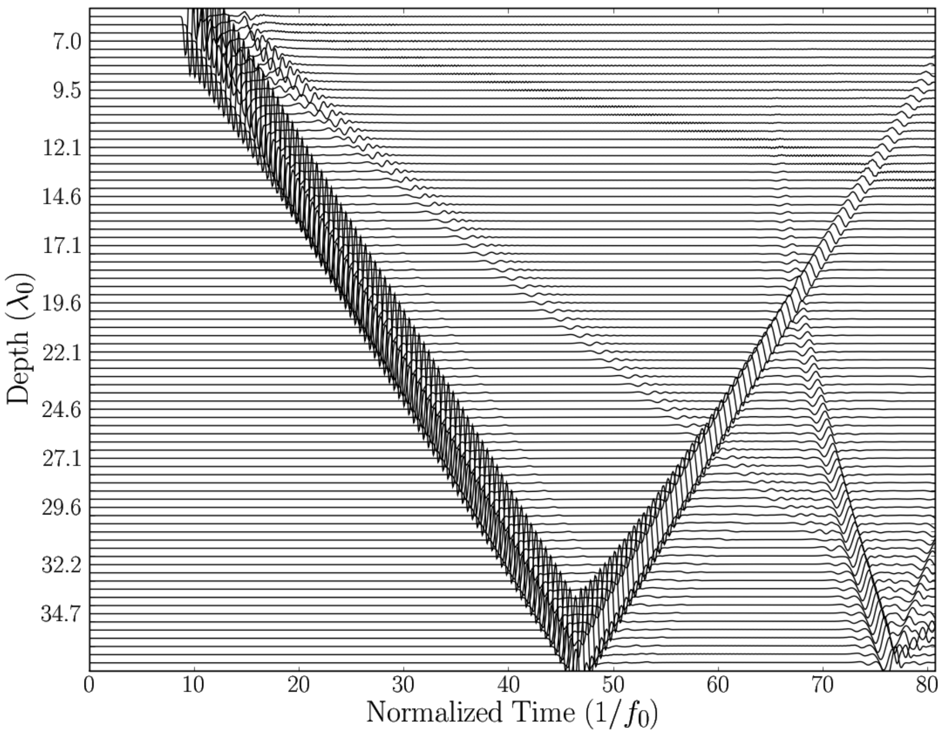


Figure E.15: High frequency seismic plot for axisymmetric axis of specimen “m3c”

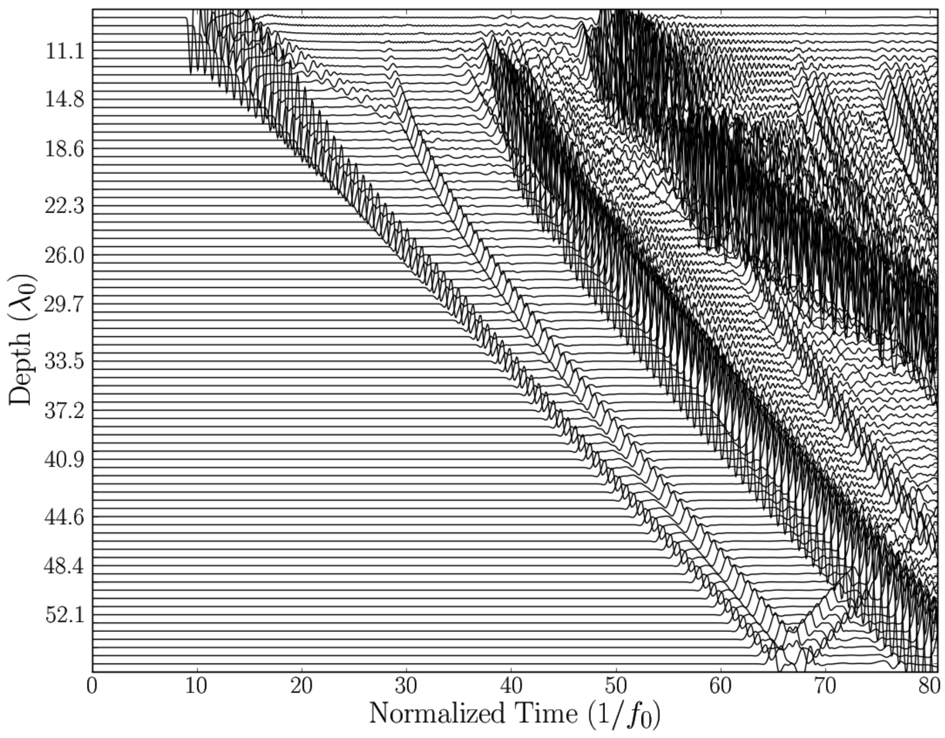


Figure E.16: High frequency seismic plot for axisymmetric axis of specimen “m4a”

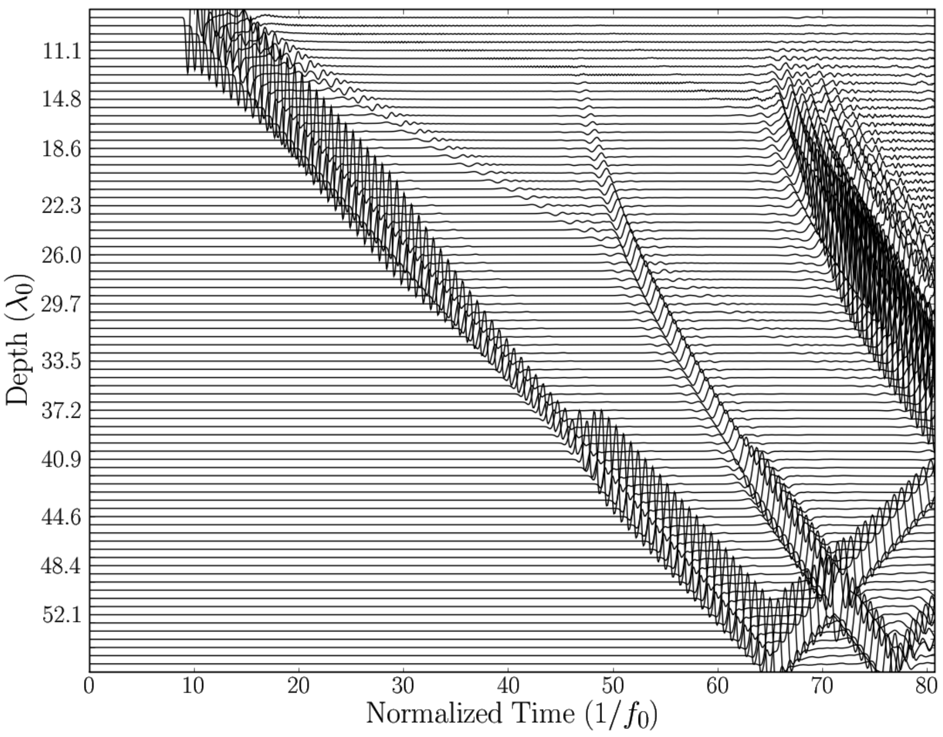


Figure E.17: High frequency seismic plot for axisymmetric axis of specimen “m4b”

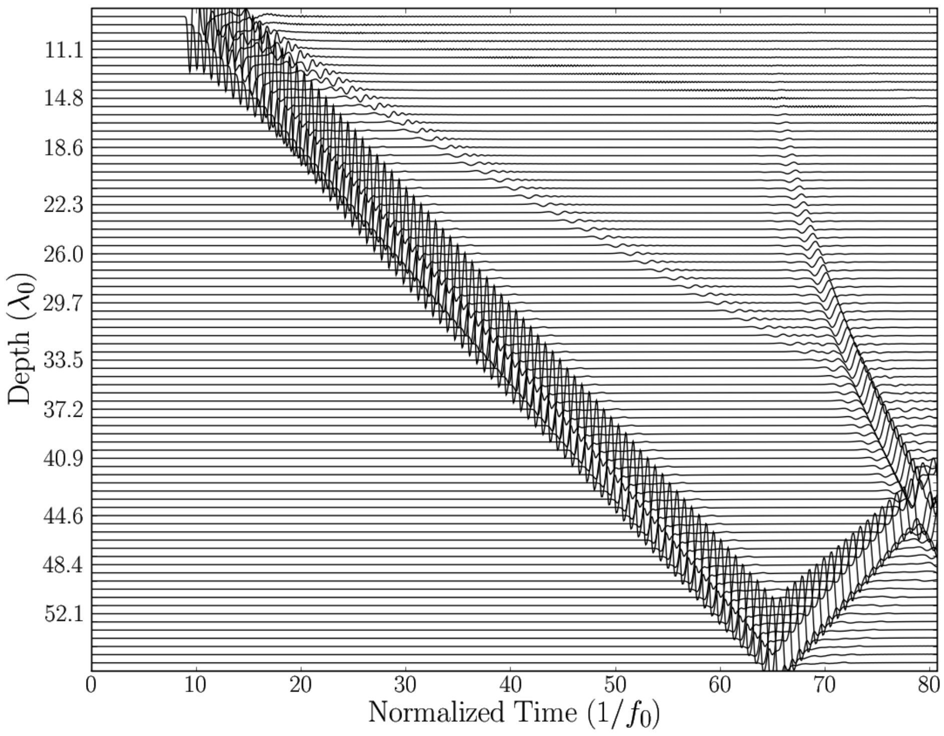


Figure E.18: High frequency seismic plot for axisymmetric axis of specimen “m4c”

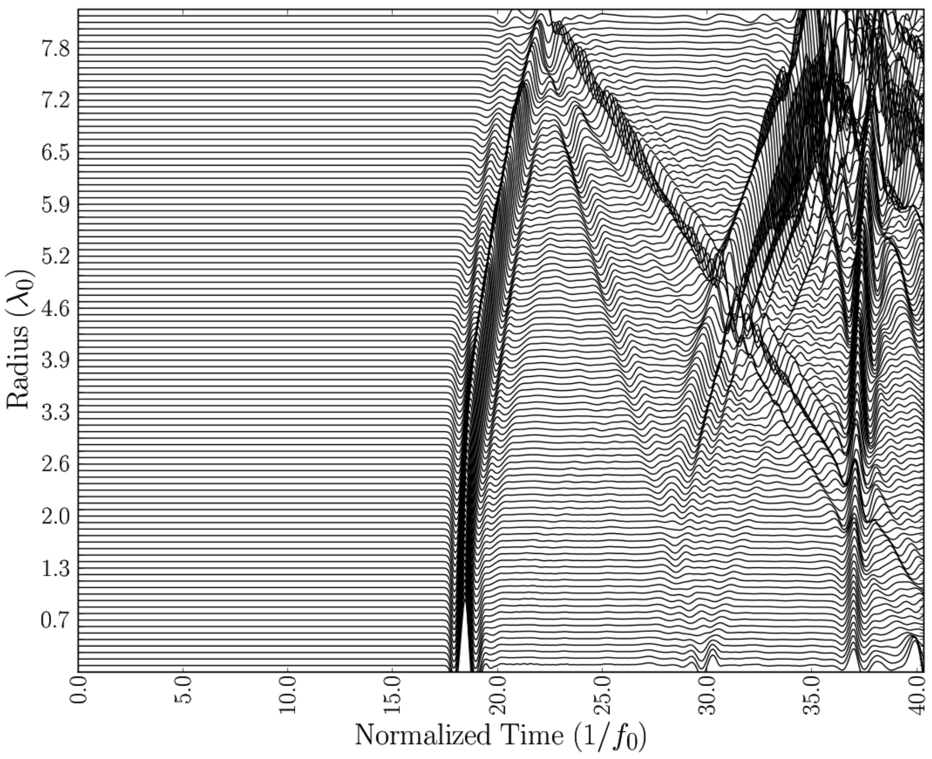


Figure E.19: High frequency seismic plot for bottom axis of specimen “m1a”

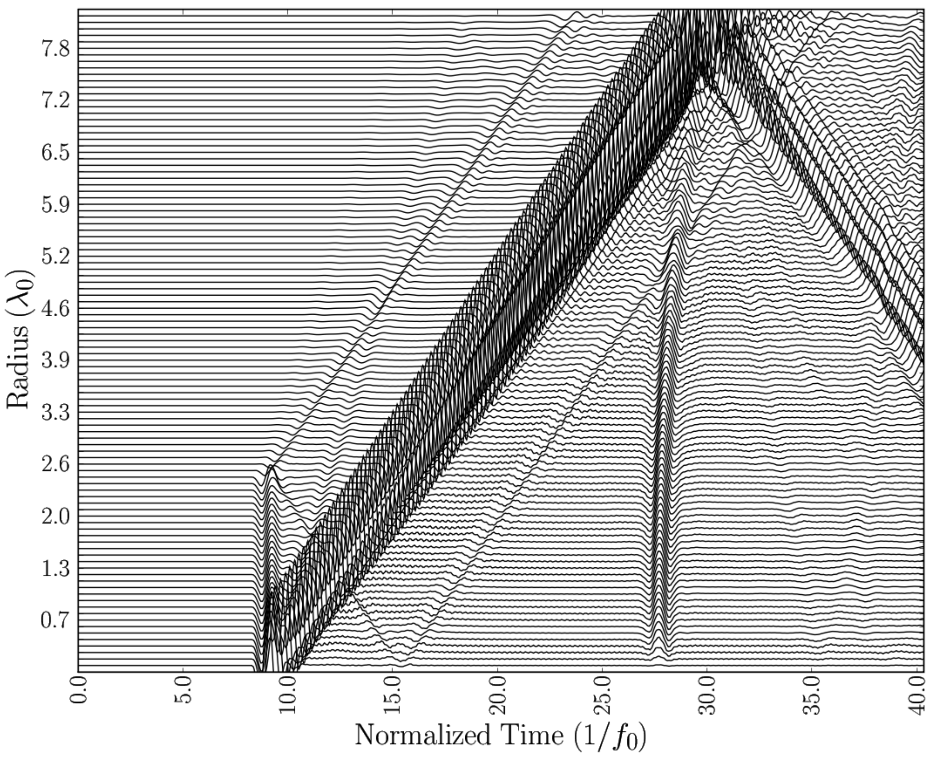


Figure E.20: High frequency seismic plot for top axis of specimen “m1a”

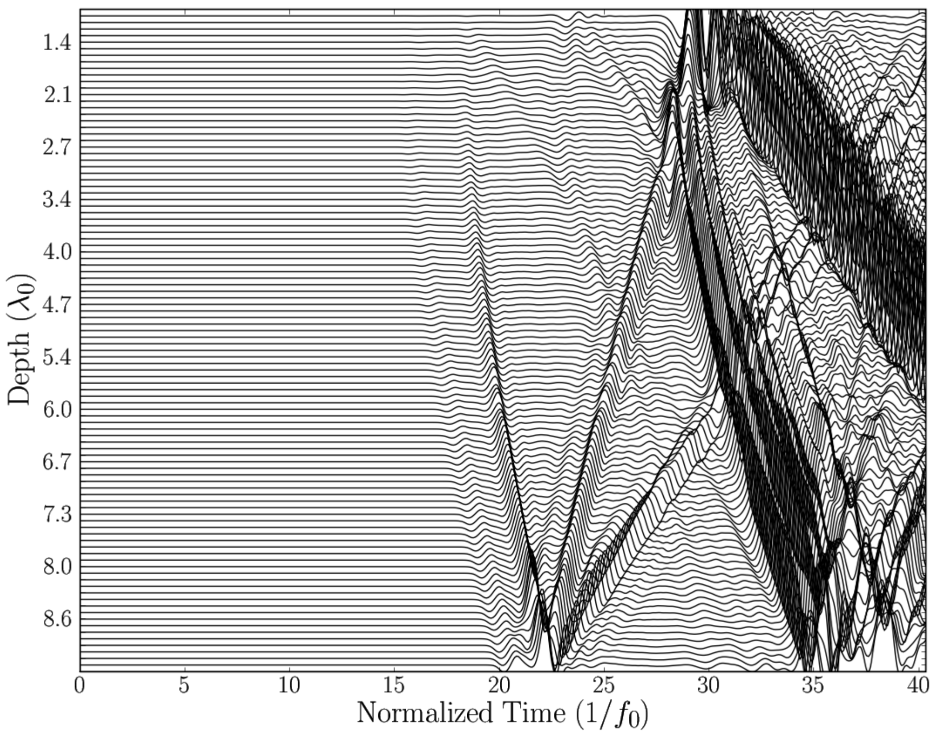


Figure E.21: High frequency seismic plot for side (right) axis of specimen “m1a”

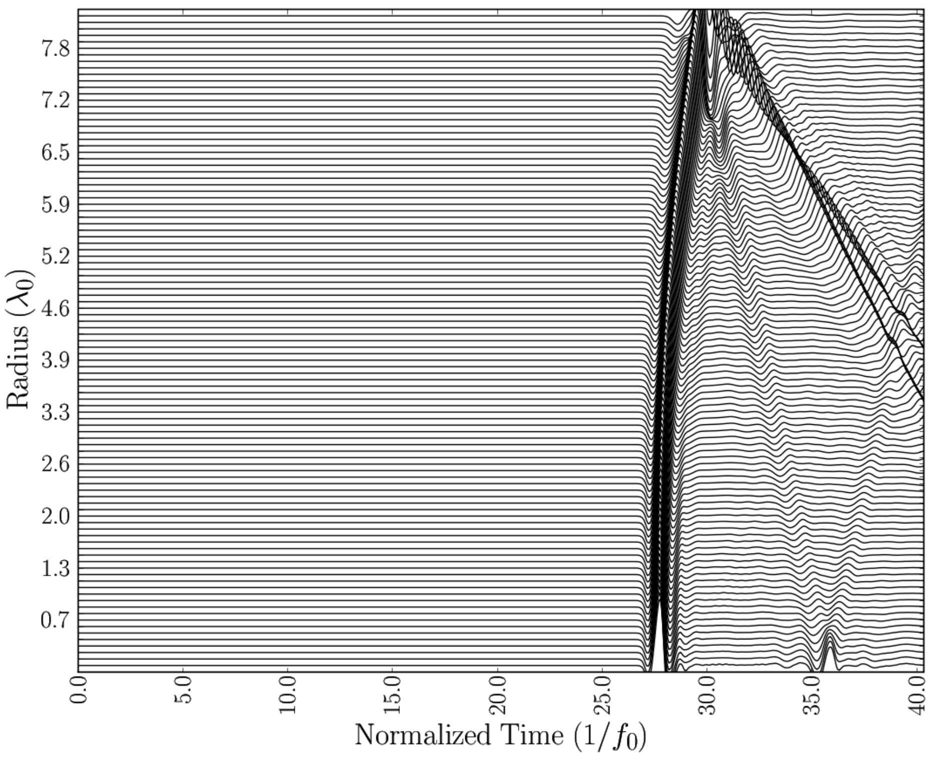


Figure E.22: High frequency seismic plot for bottom axis of specimen “m2a”

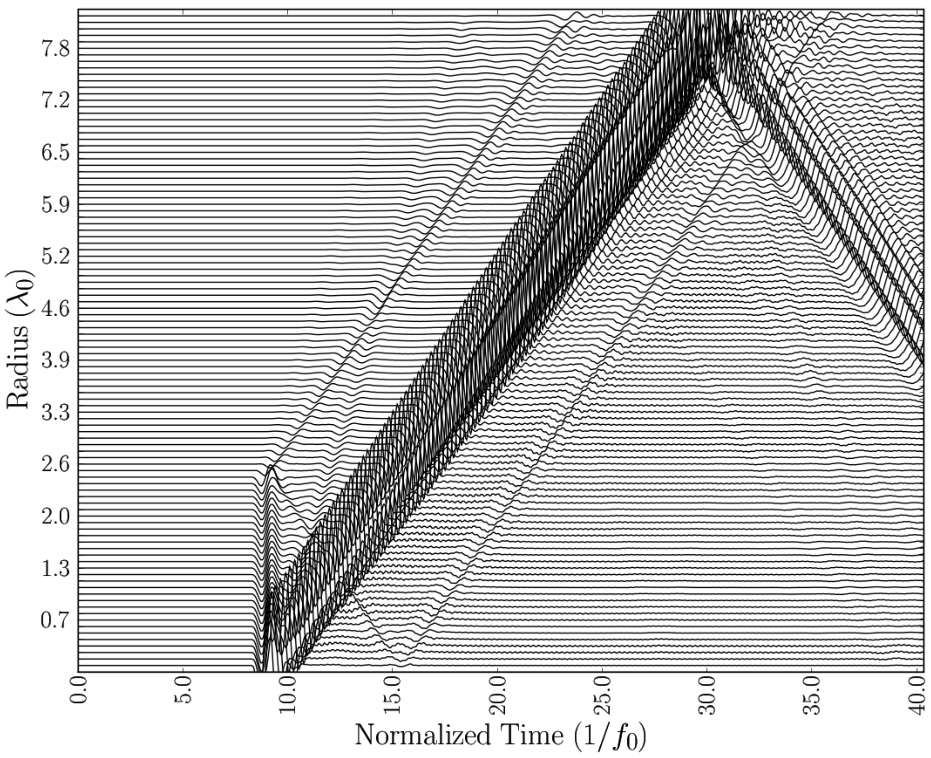


Figure E.23: High frequency seismic plot for top axis of specimen "m2a"

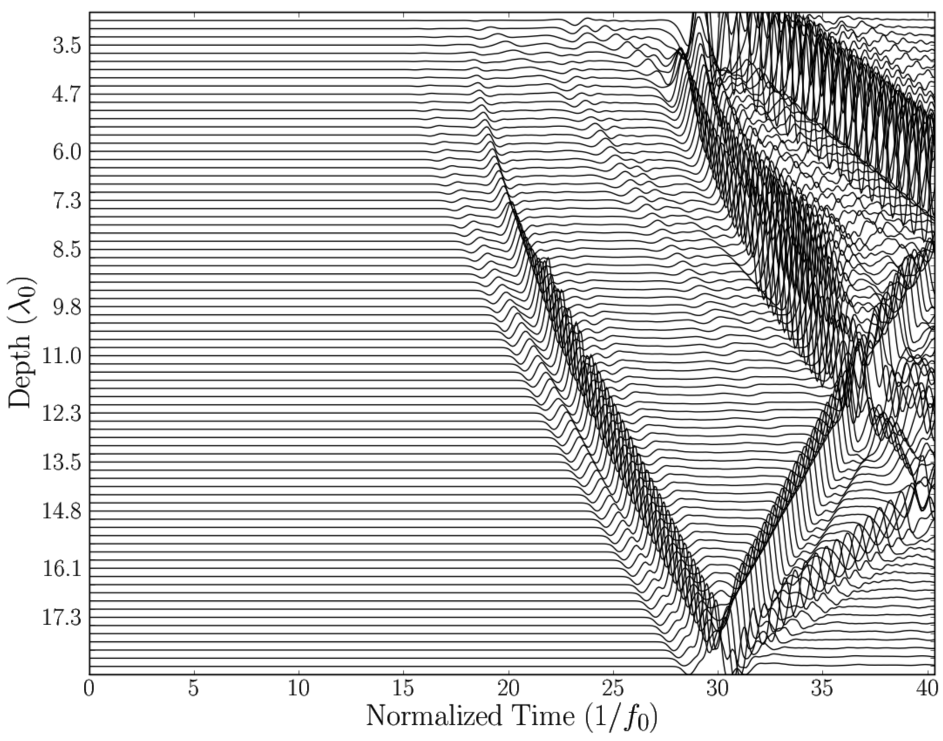


Figure E.24: High frequency seismic plot for side (right) axis of specimen “m2a”

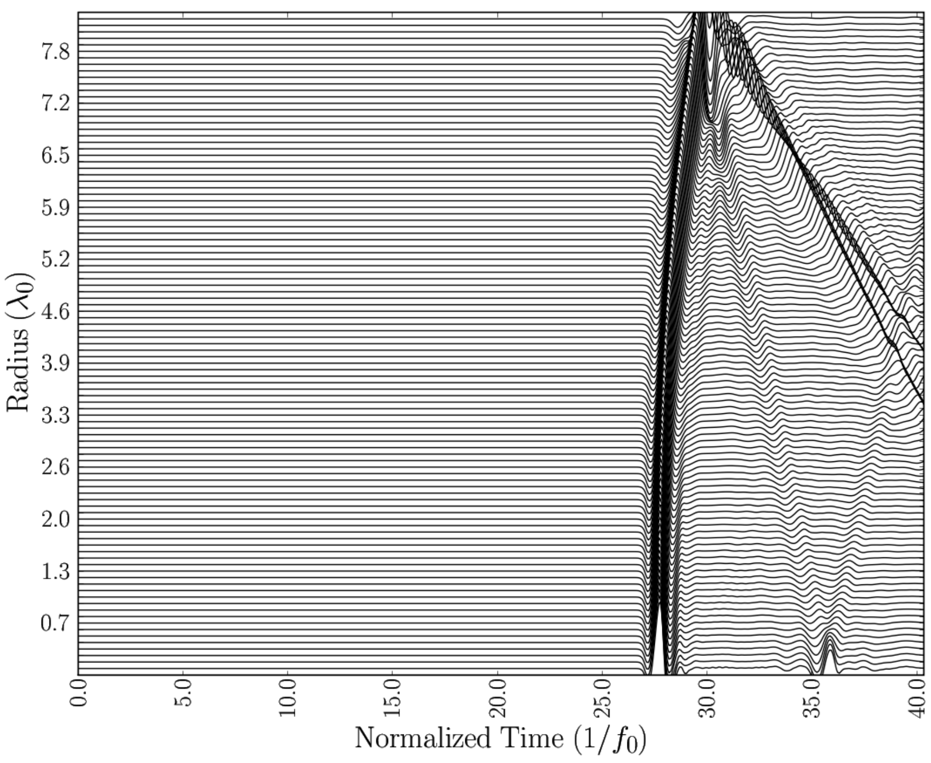


Figure E.25: High frequency seismic plot for bottom axis of specimen “m2a”

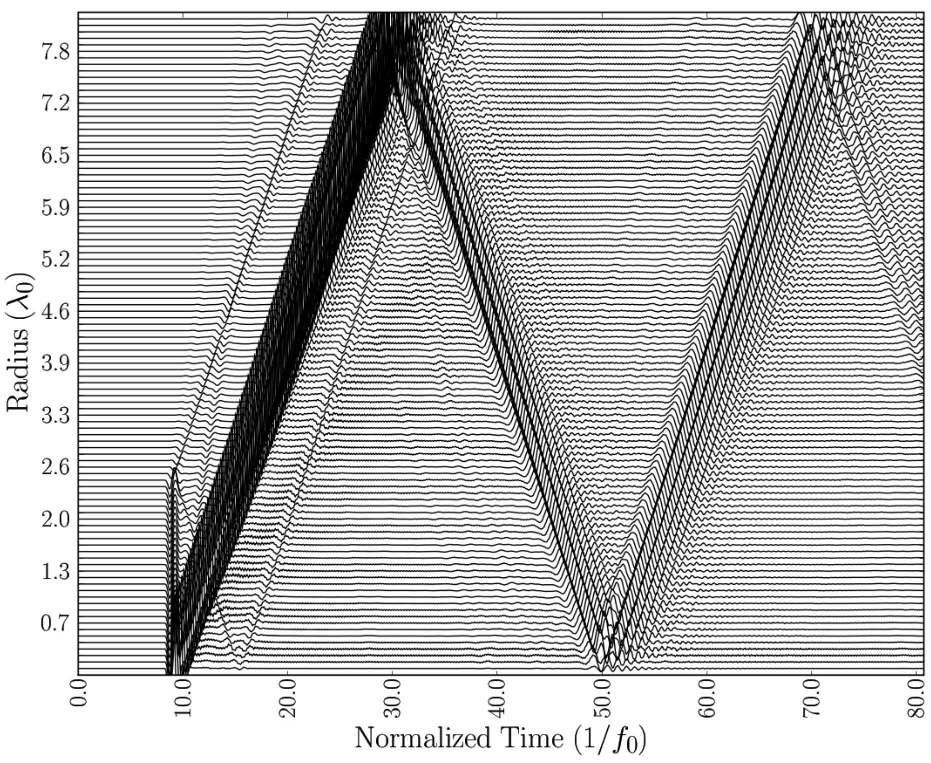


Figure E.26: High frequency seismic plot for top axis of specimen “m3a”

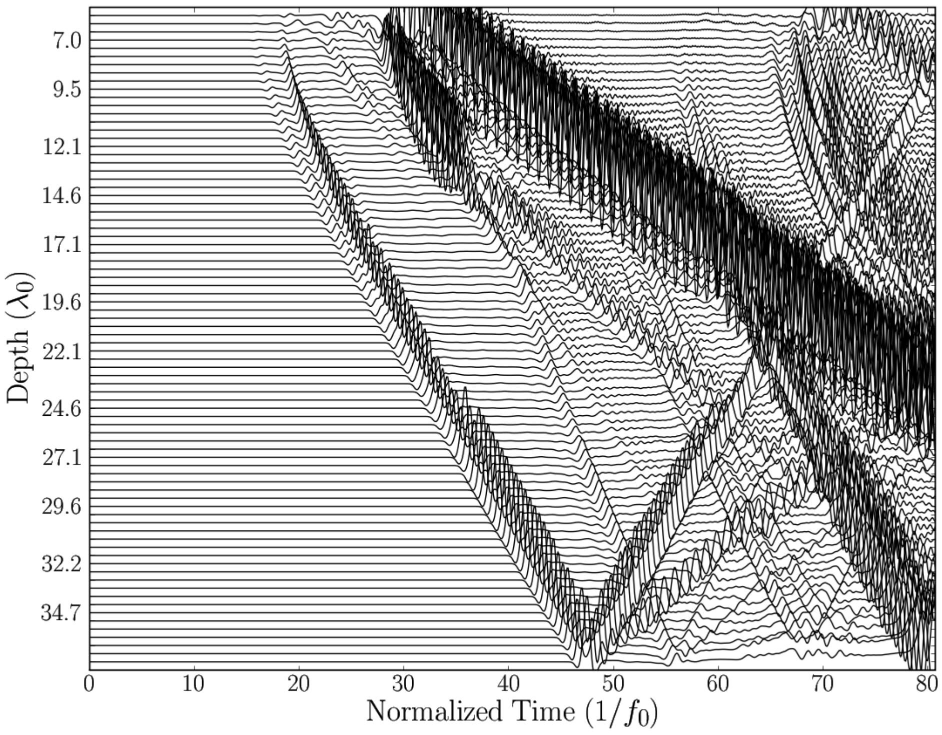


Figure E.27: High frequency seismic plot for side (right) axis of specimen “m3a”

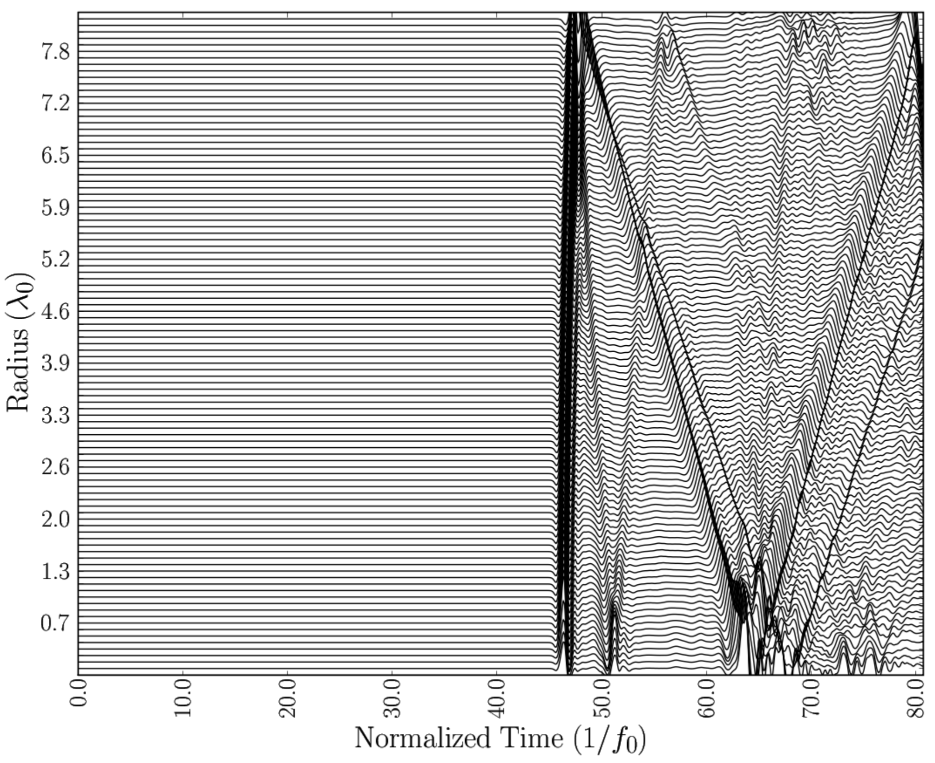


Figure E.28: High frequency seismic plot for bottom axis of specimen “m3a”

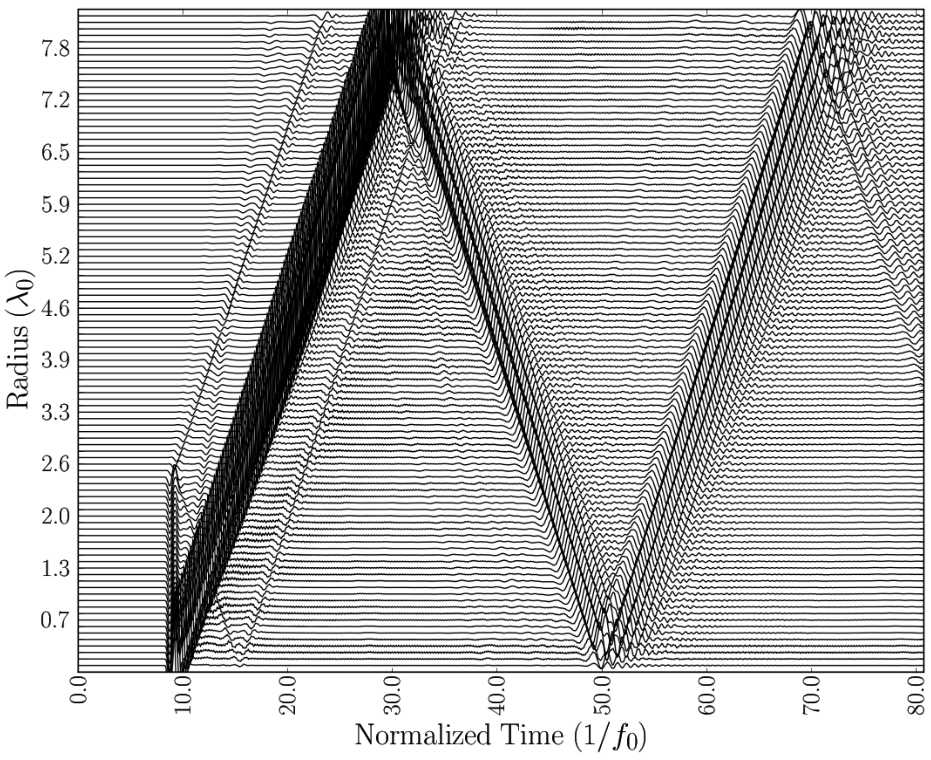


Figure E.29: High frequency seismic plot for top axis of specimen “m4a”

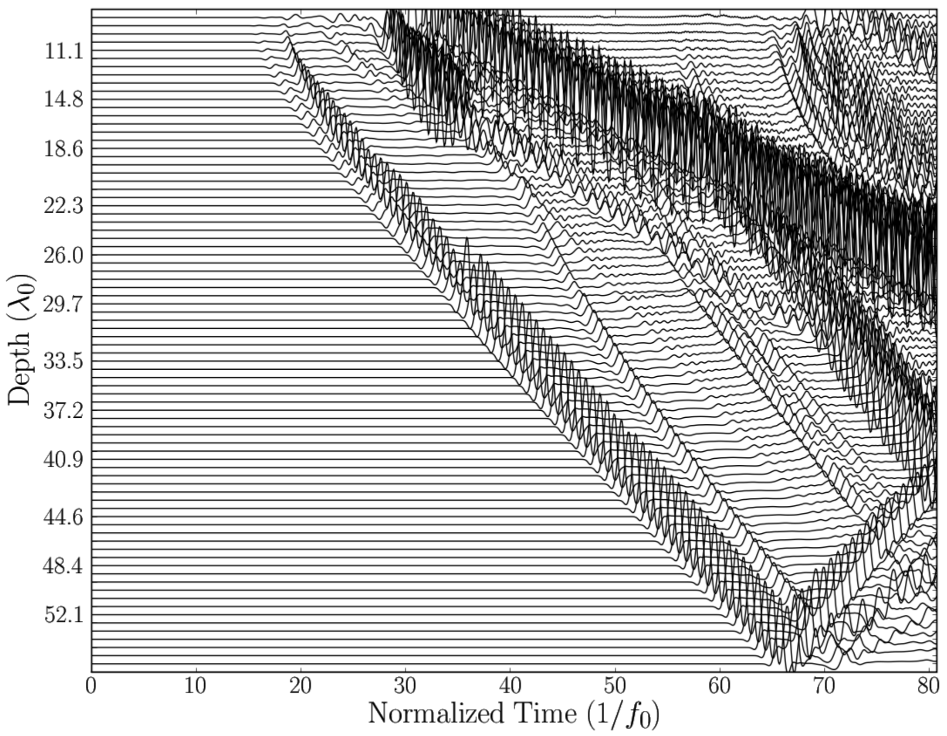


Figure E.30: High frequency seismic plot for side (right) axis of specimen “m4a”

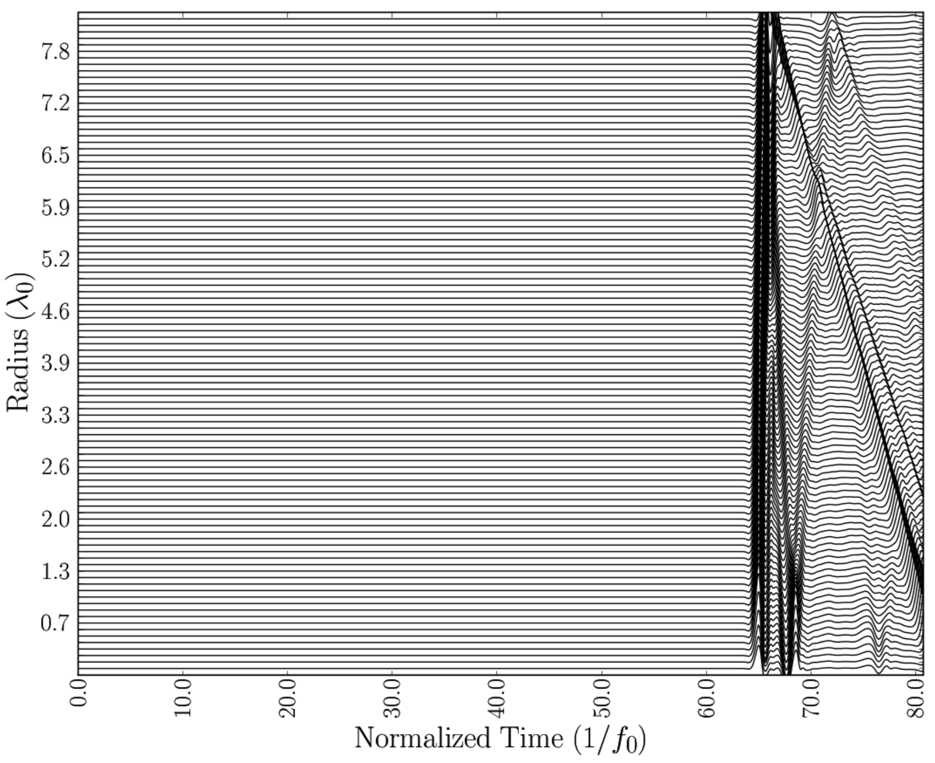


Figure E.31: High frequency seismic plot for bottom axis of specimen “m4a”

Frequency Wavenumber Plots

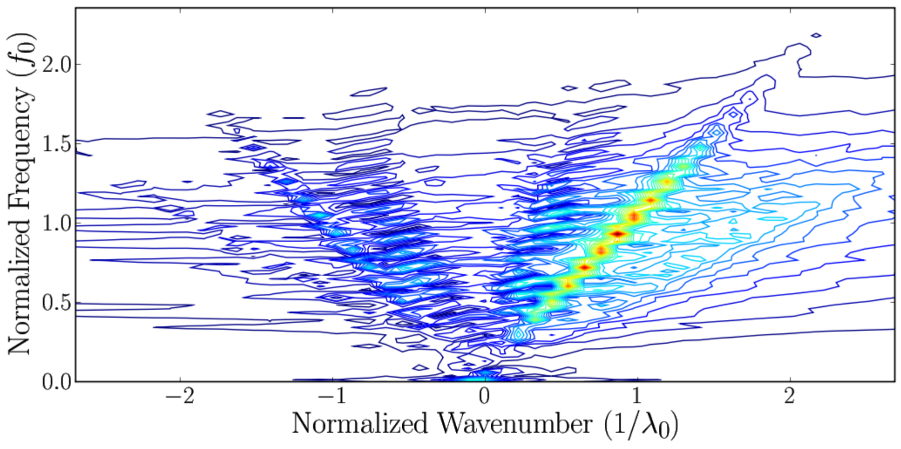


Figure E.32: High frequency F-K plot for axisymmetric axis of specimen “m1a”

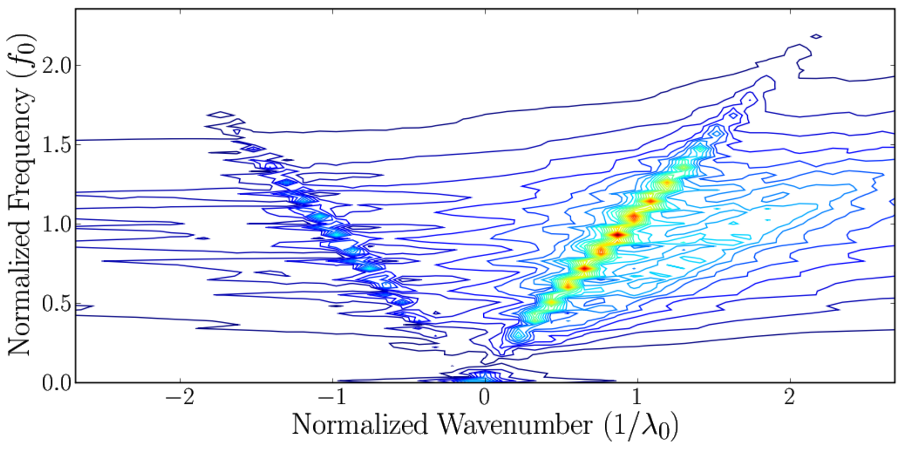


Figure E.33: High frequency F-K plot for axisymmetric axis of specimen “m1b”

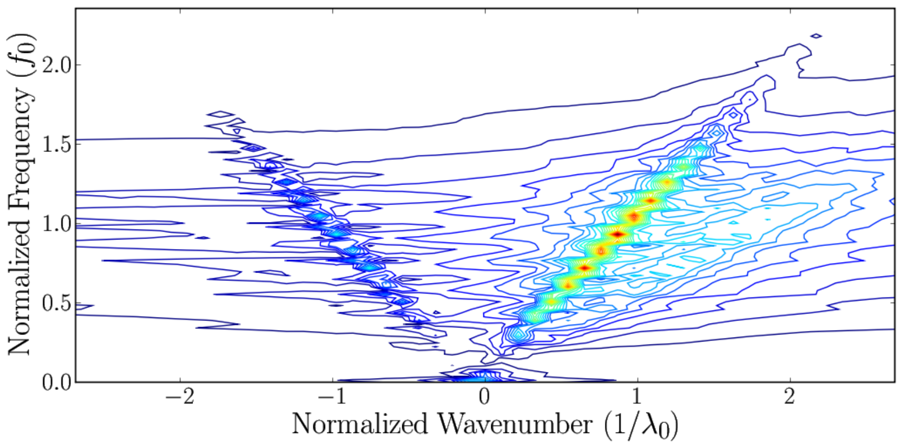


Figure E.34: High frequency F-K plot for axisymmetric axis of specimen “m1c”

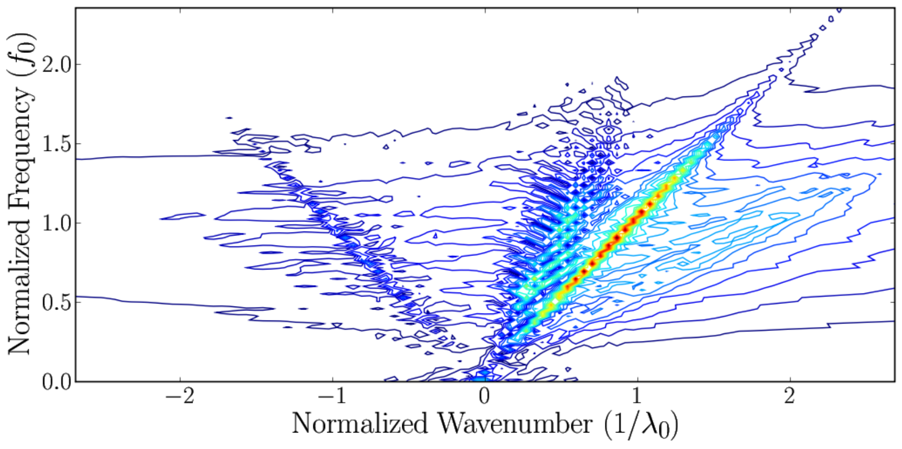


Figure E.35: High frequency F-K plot for axisymmetric axis of specimen “m2a”

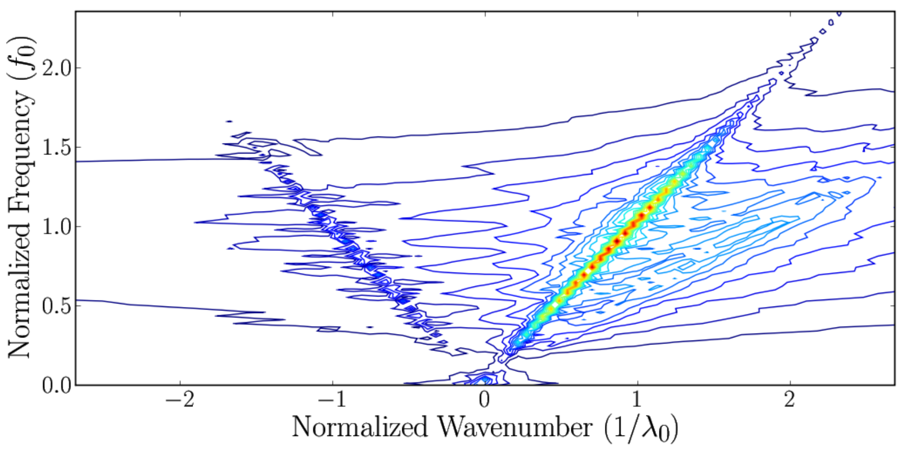


Figure E.36: High frequency F-K plot for axisymmetric axis of specimen “m2b”

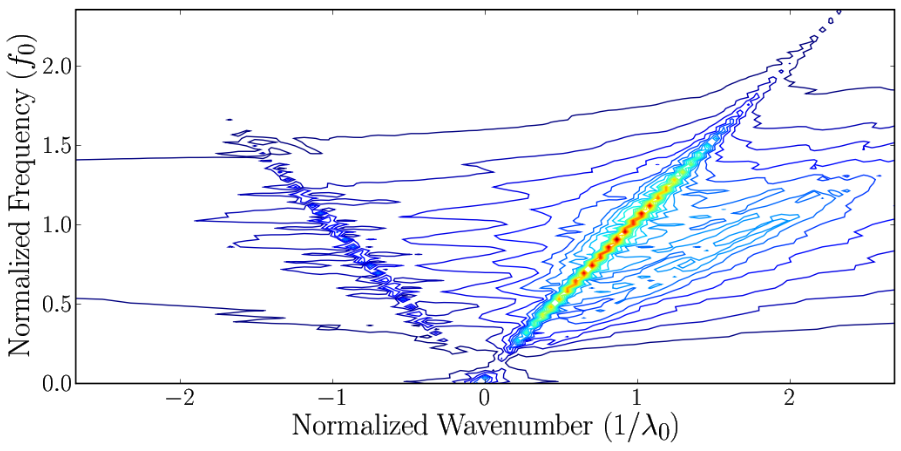


Figure E.37: High frequency F-K plot for axisymmetric axis of specimen “m2c”

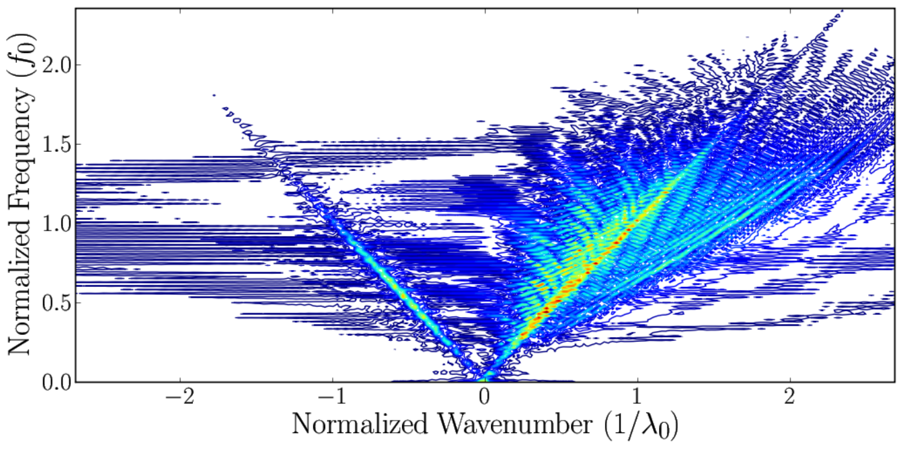


Figure E.38: High frequency F-K plot for axisymmetric axis of specimen “m3a”

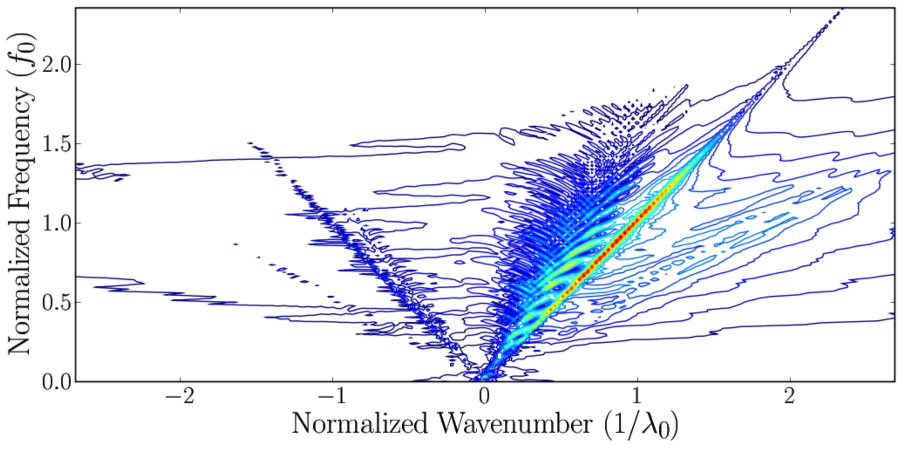


Figure E.39: High frequency F-K plot for axisymmetric axis of specimen “m3b”

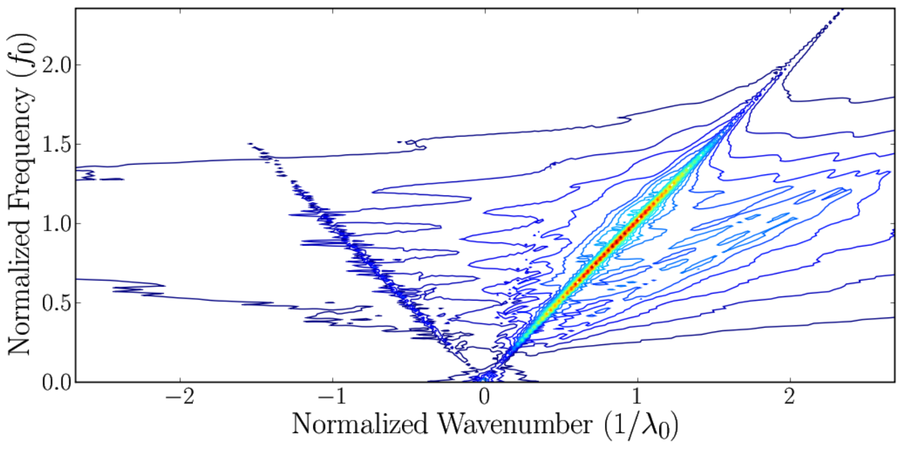


Figure E.40: High frequency F-K plot for axisymmetric axis of specimen “m3c”

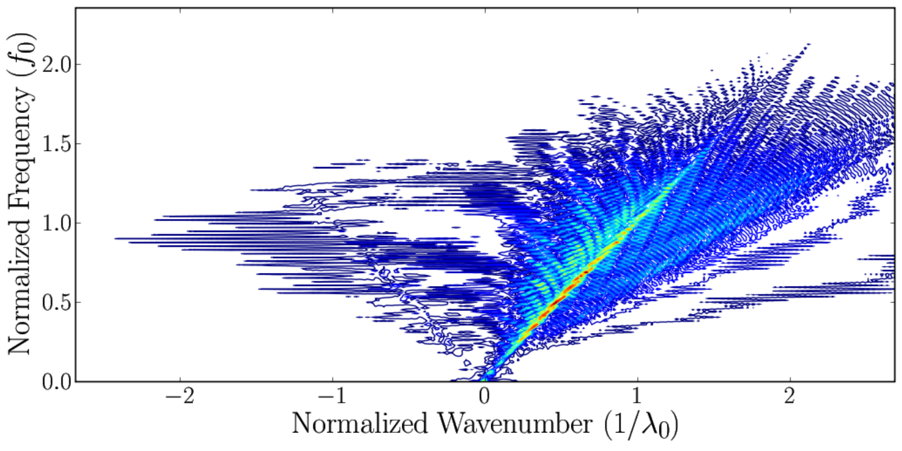


Figure E.41: High frequency F-K plot for axisymmetric axis of specimen “m4a”

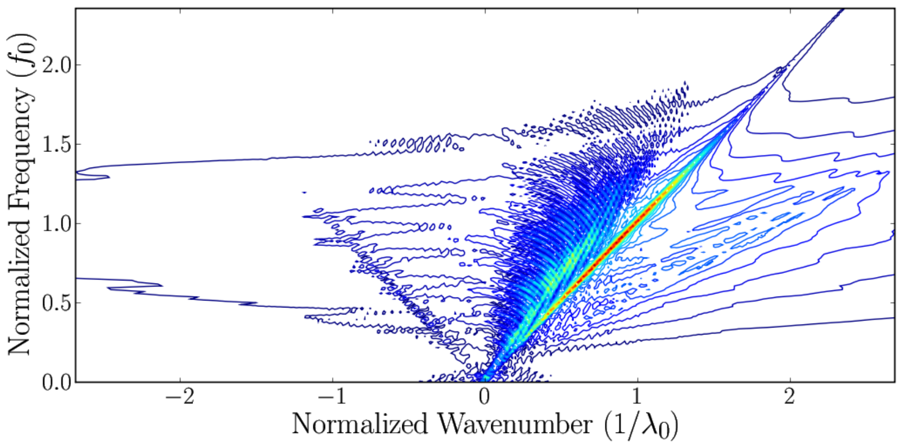


Figure E.42: High frequency F-K plot for axisymmetric axis of specimen “m4b”

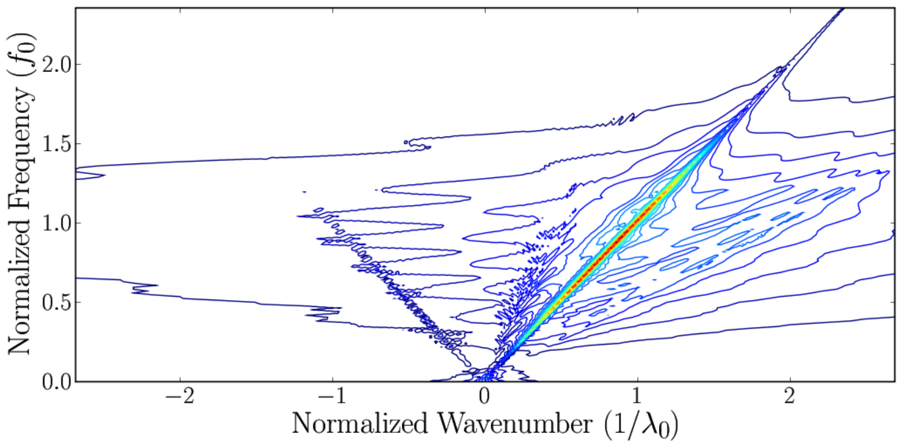


Figure E.43: High frequency F-K plot for axisymmetric axis of specimen “m4c”

E.2.2 Low Frequency

Seismic Plots

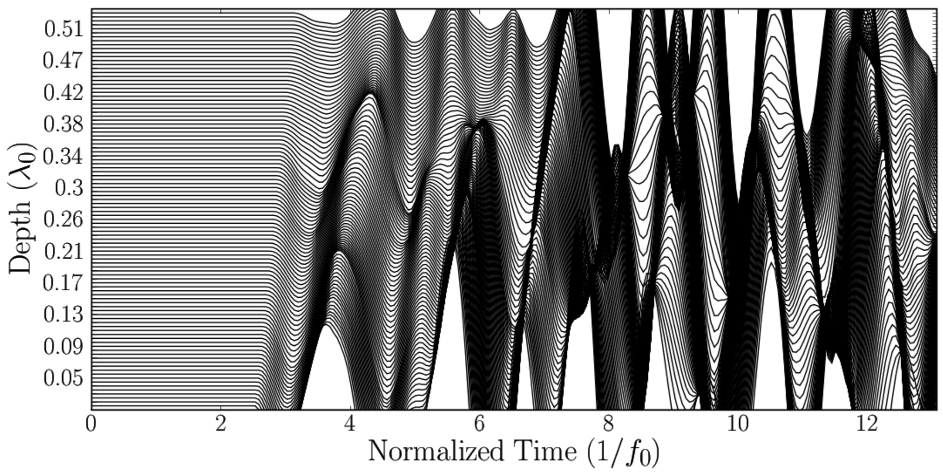


Figure E.44: Low frequency seismic plot for axisymmetric axis of specimen “m1a”

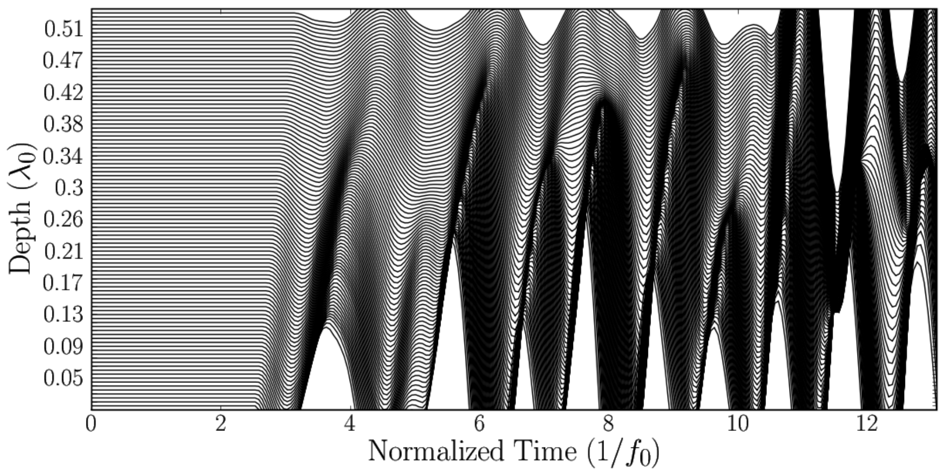


Figure E.45: Low frequency seismic plot for axisymmetric axis of specimen “m1b”

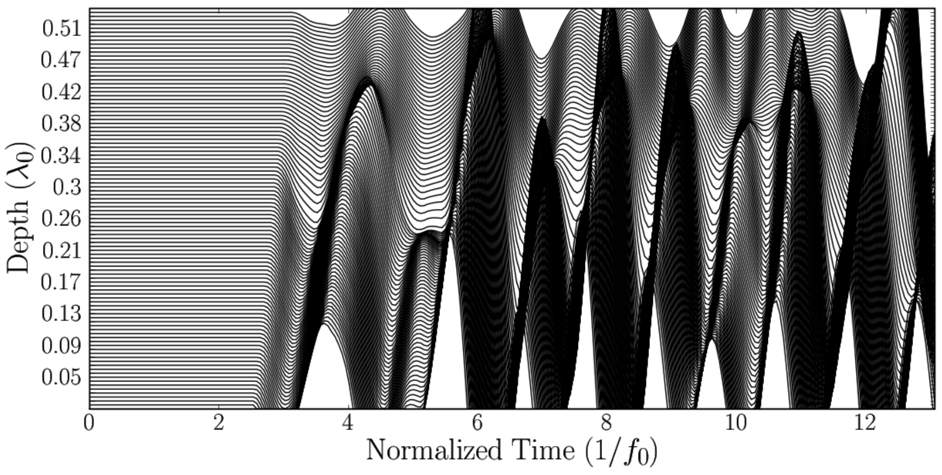


Figure E.46: Low frequency seismic plot for axisymmetric axis of specimen “m1c”

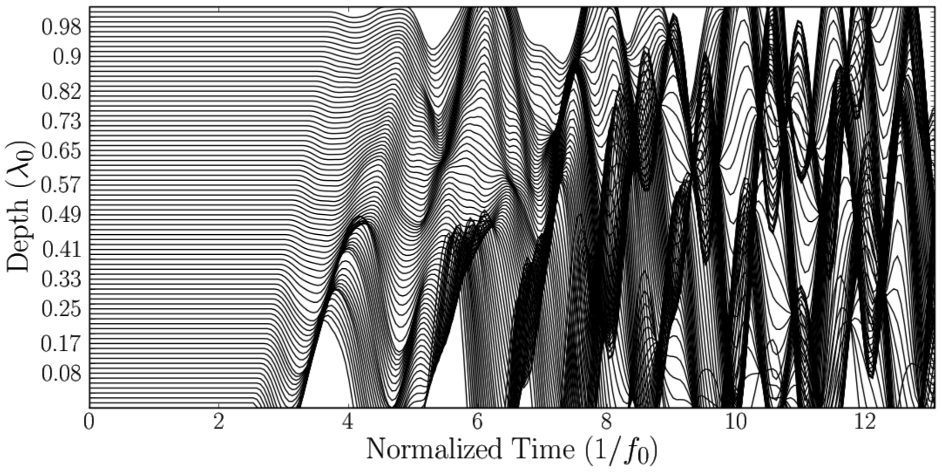


Figure E.47: Low frequency seismic plot for axisymmetric axis of specimen “m2a”

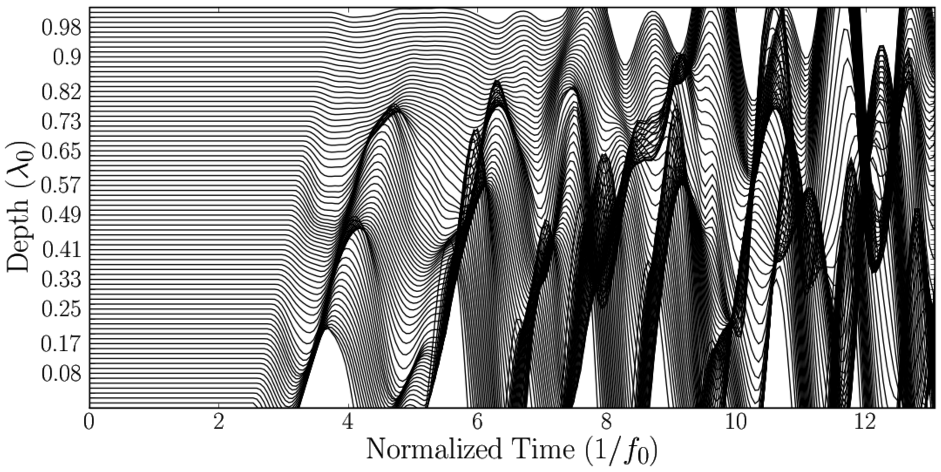


Figure E.48: Low frequency seismic plot for axisymmetric axis of specimen "m2b"

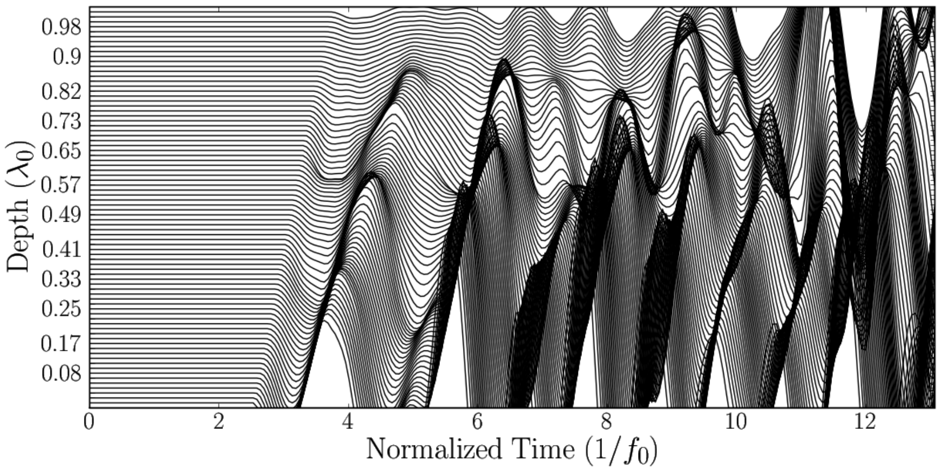


Figure E.49: Low frequency seismic plot for axisymmetric axis of specimen "m2c"

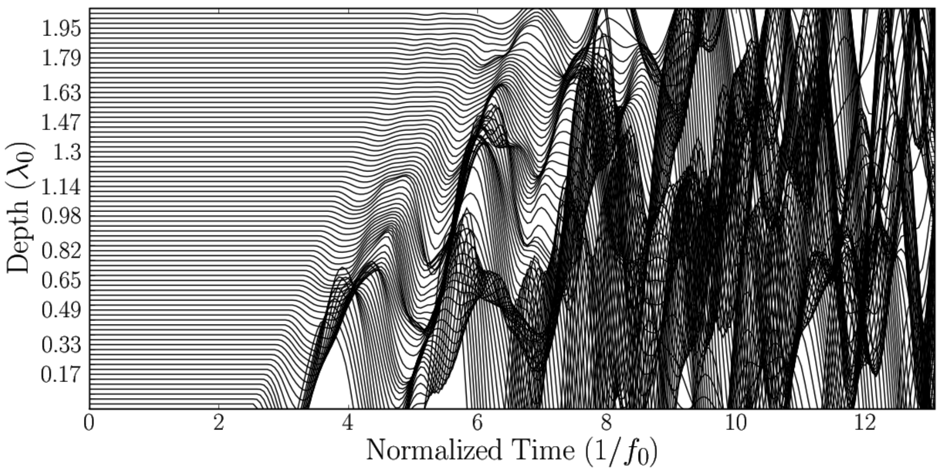


Figure E.50: Low frequency seismic plot for axisymmetric axis of specimen "m3a"

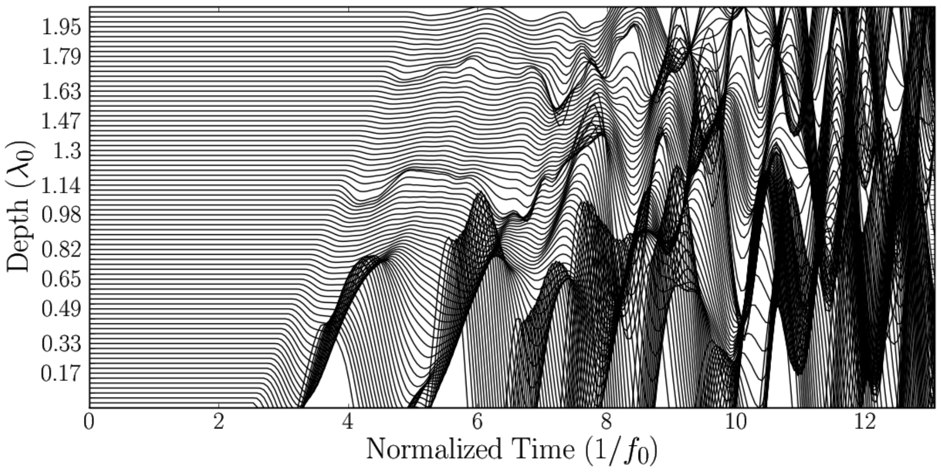


Figure E.51: Low frequency seismic plot for axisymmetric axis of specimen "m3b"

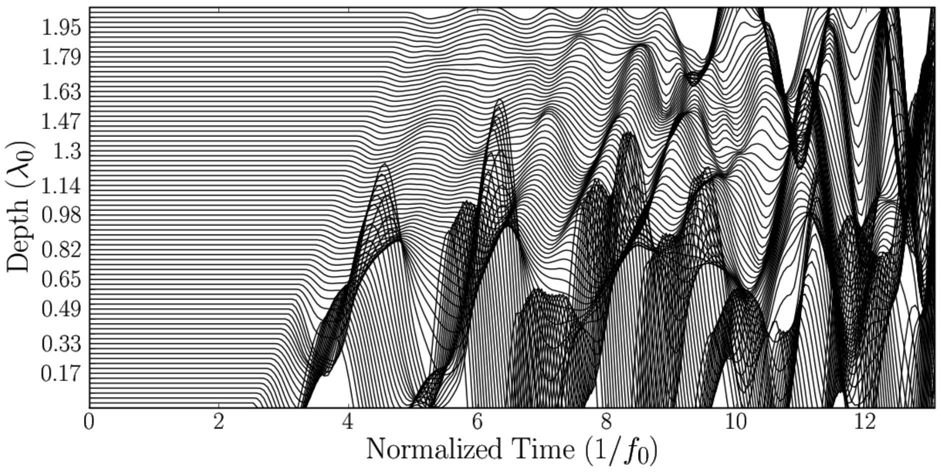


Figure E.52: Low frequency seismic plot for axisymmetric axis of specimen "m3c"

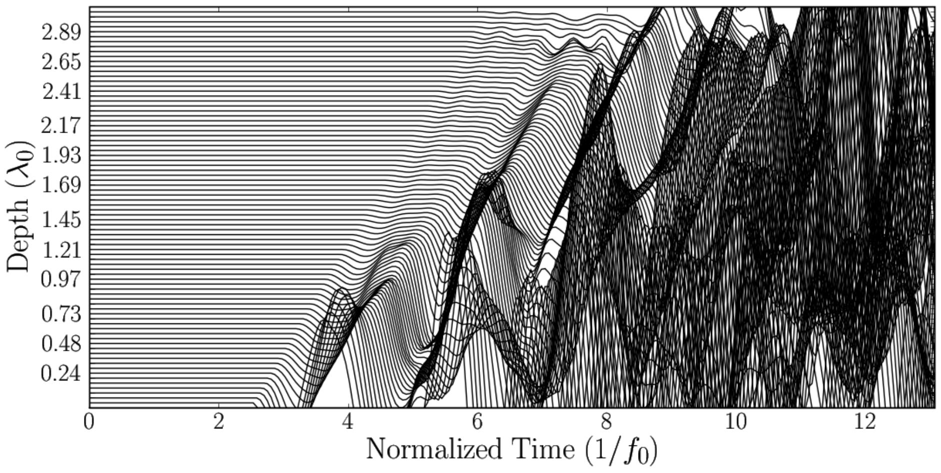


Figure E.53: Low frequency seismic plot for axisymmetric axis of specimen "m4a"

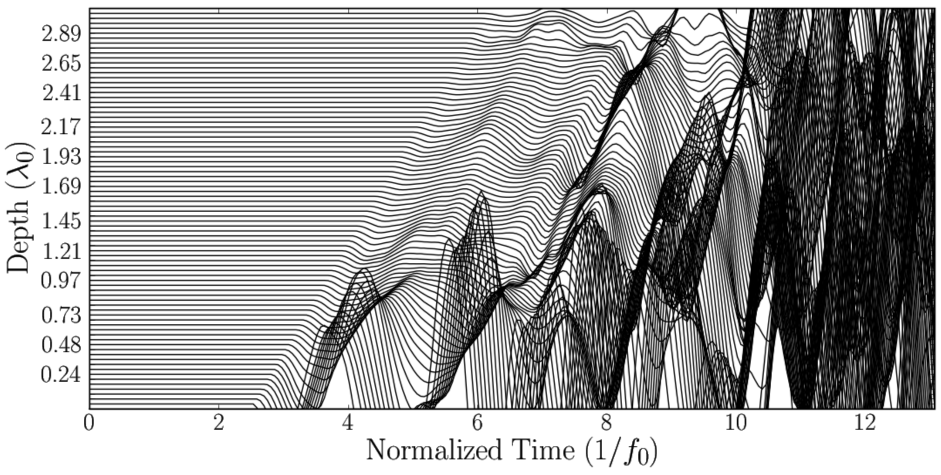


Figure E.54: Low frequency seismic plot for axisymmetric axis of specimen "m4b"

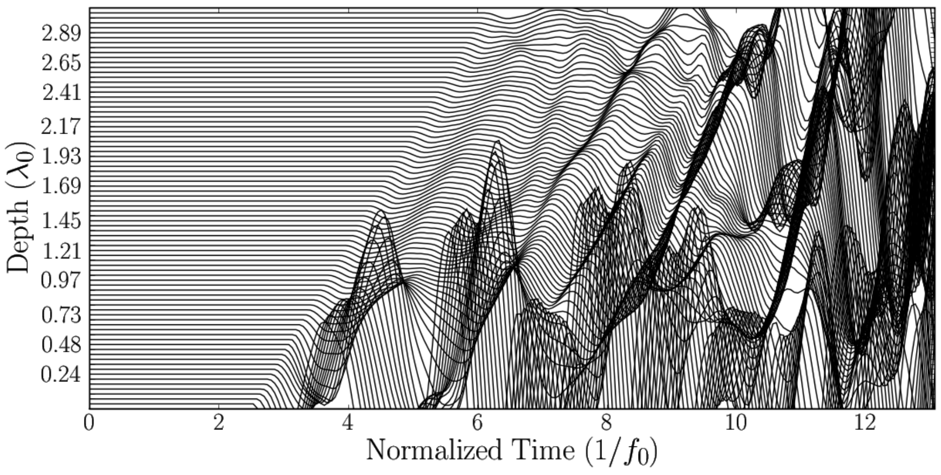


Figure E.55: Low frequency seismic plot for axisymmetric axis of specimen "m4c"

Frequency Wavenumber Plots

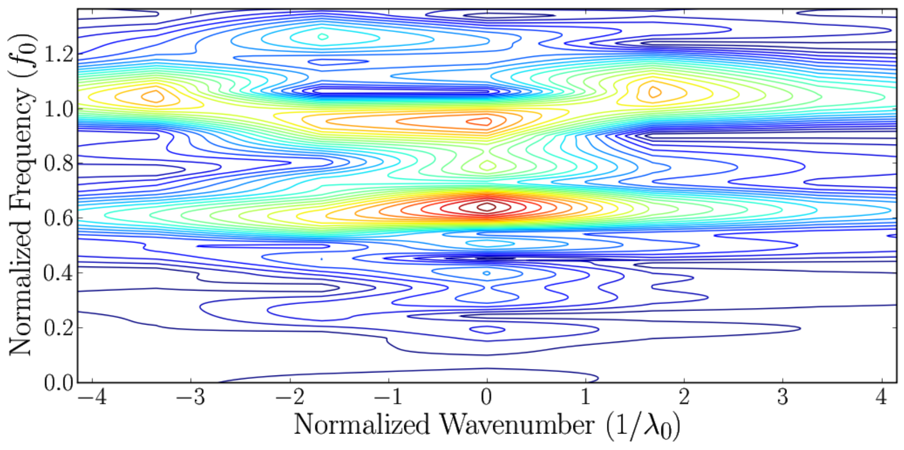


Figure E.56: Low frequency F-K plot for axisymmetric axis of specimen “m1a”

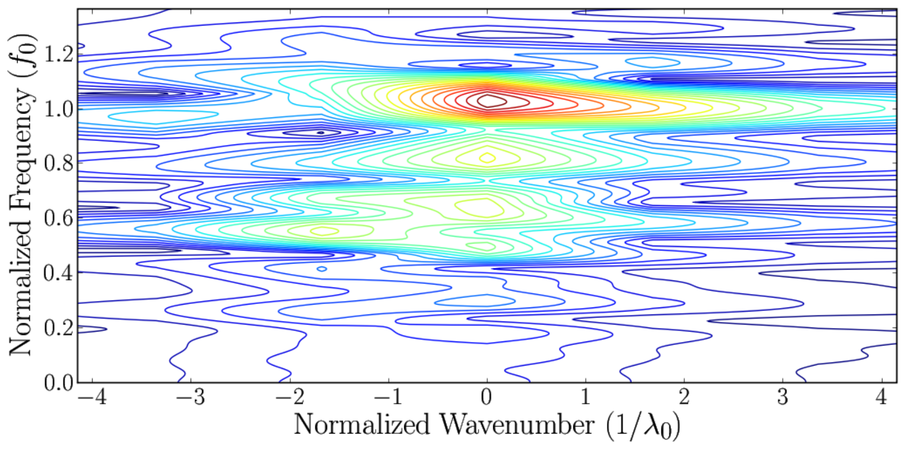


Figure E.57: Low frequency F-K plot for axisymmetric axis of specimen “m1b”

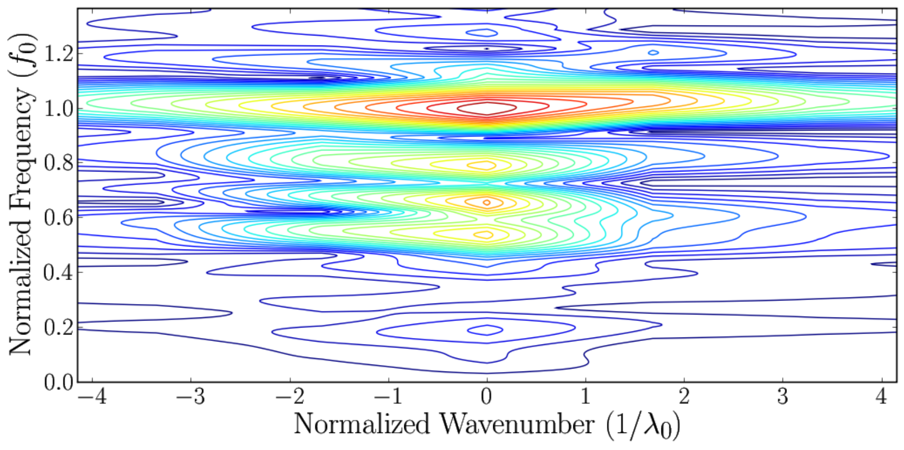


Figure E.58: Low frequency F-K plot for axisymmetric axis of specimen "m1c"

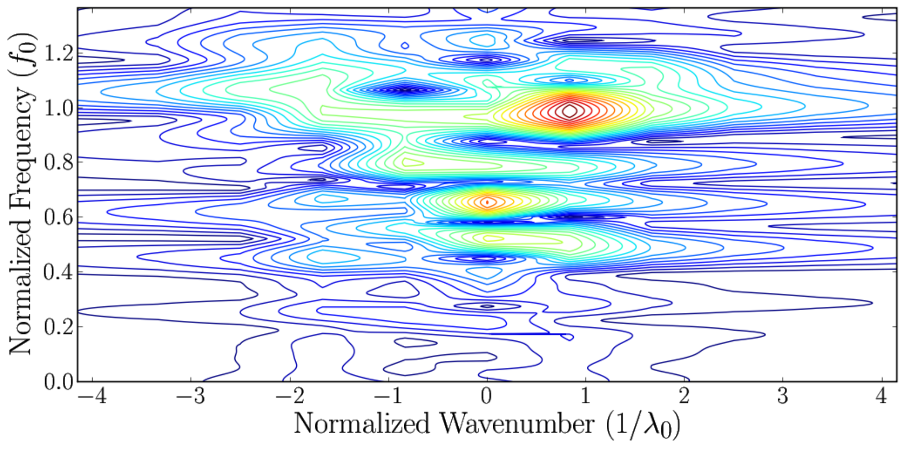


Figure E.59: Low frequency F-K plot for axisymmetric axis of specimen "m2a"

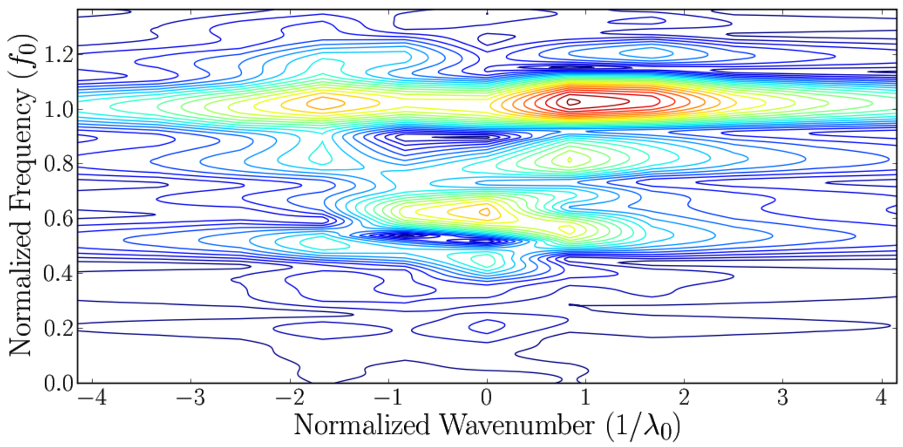


Figure E.60: Low frequency F-K plot for axisymmetric axis of specimen "m2b"

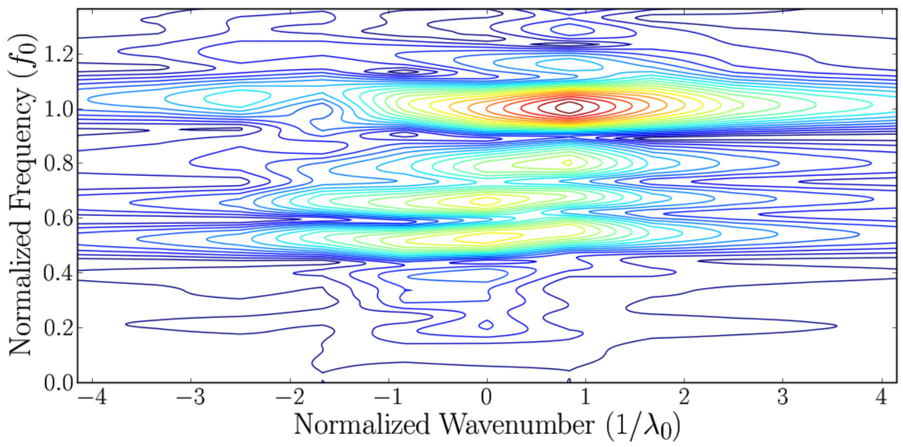


Figure E.61: Low frequency F-K plot for axisymmetric axis of specimen "m2c"

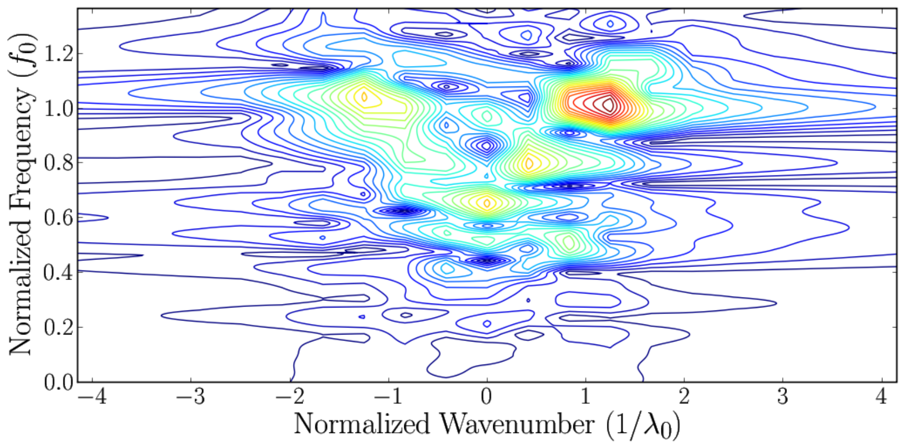


Figure E.62: Low frequency F-K plot for axisymmetric axis of specimen "m3a"

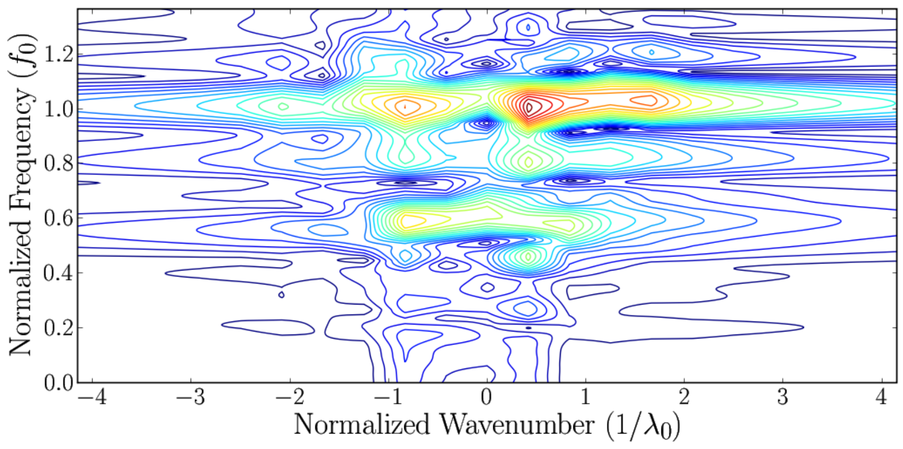


Figure E.63: Low frequency F-K plot for axisymmetric axis of specimen "m3b"

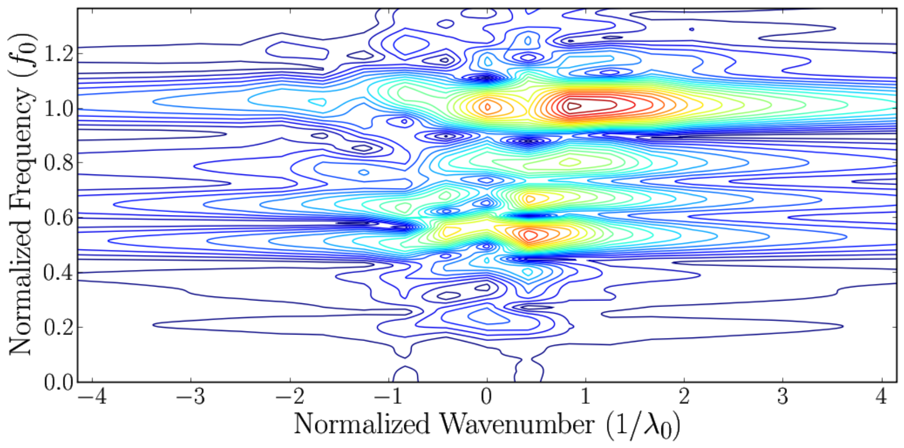


Figure E.64: Low frequency F-K plot for axisymmetric axis of specimen "m3c"

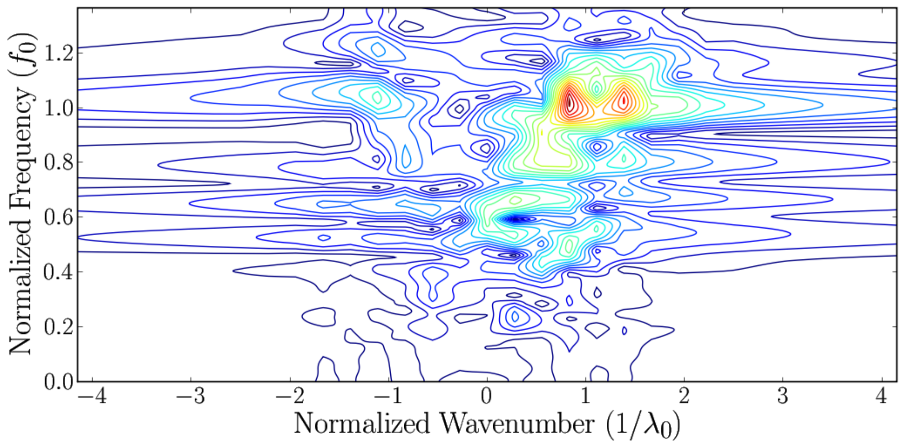


Figure E.65: Low frequency F-K plot for axisymmetric axis of specimen "m4a"

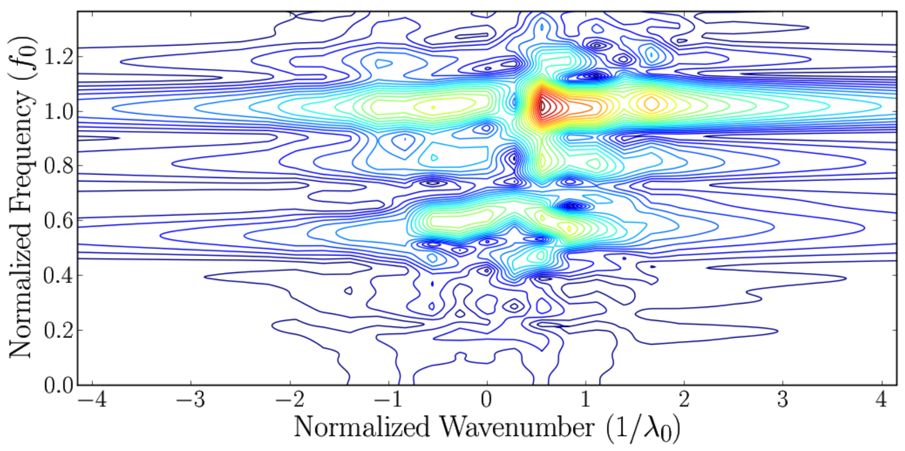


Figure E.66: Low frequency F-K plot for axisymmetric axis of specimen “m4b”

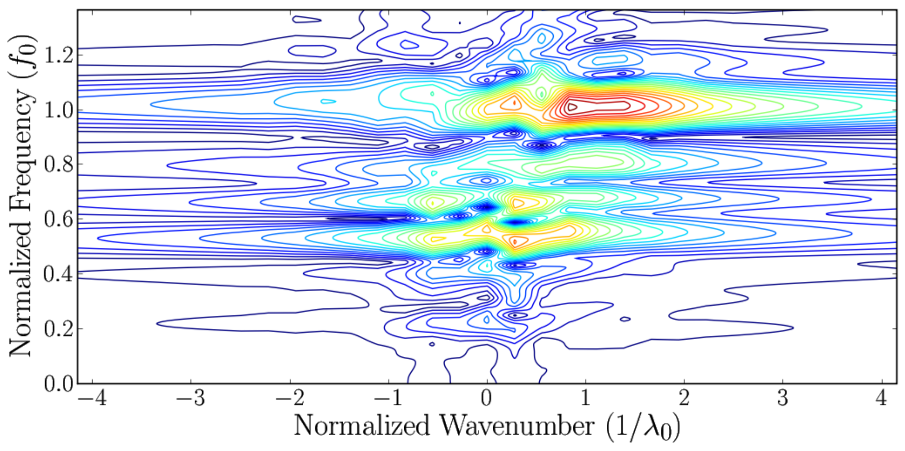


Figure E.67: Low frequency F-K plot for axisymmetric axis of specimen “m4c”

E.2.3 Low Frequency – Damped

Seismic Plot

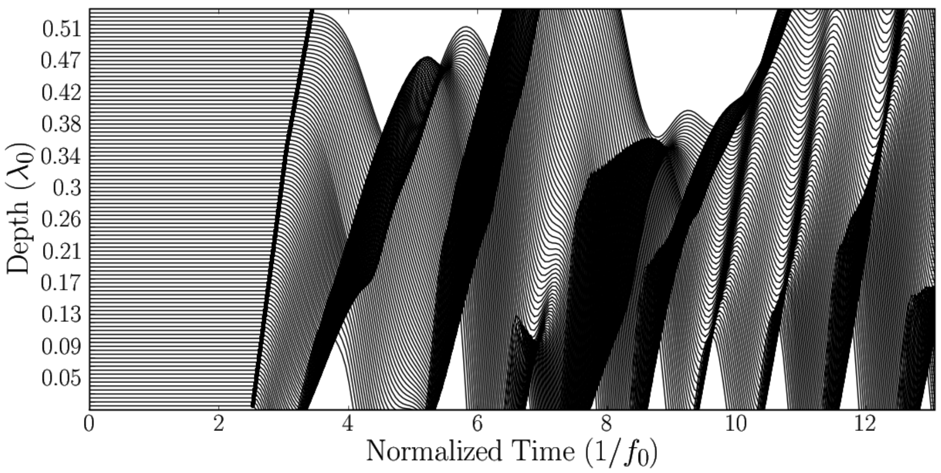


Figure E.68: Low frequency seismic plot for axisymmetric axis of specimen "m1a" using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$

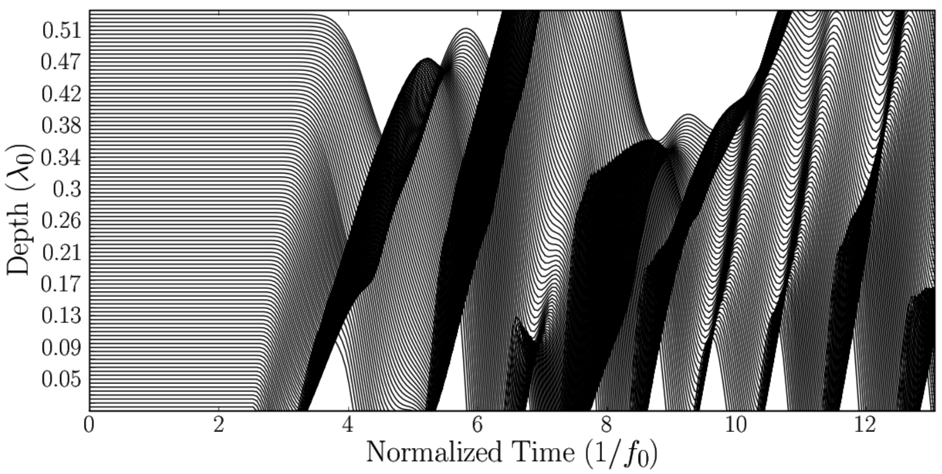


Figure E.69: Low frequency seismic plot for axisymmetric axis of specimen "m1b" using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$

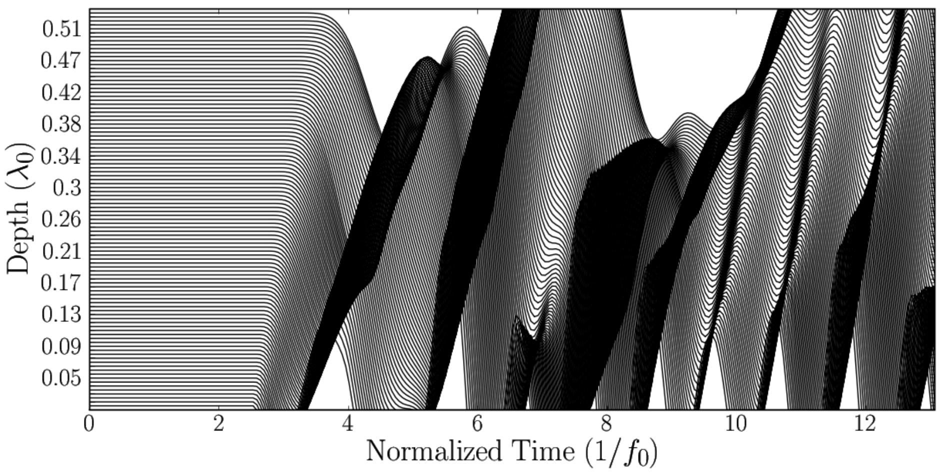


Figure E.70: Low frequency seismic plot for axisymmetric axis of specimen "m1c" using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$

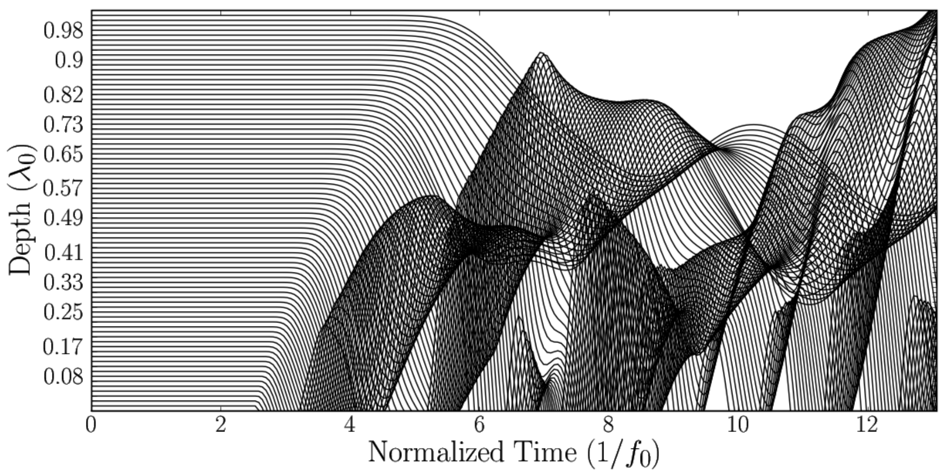


Figure E.71: Low frequency seismic plot for axisymmetric axis of specimen "m2a" using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$

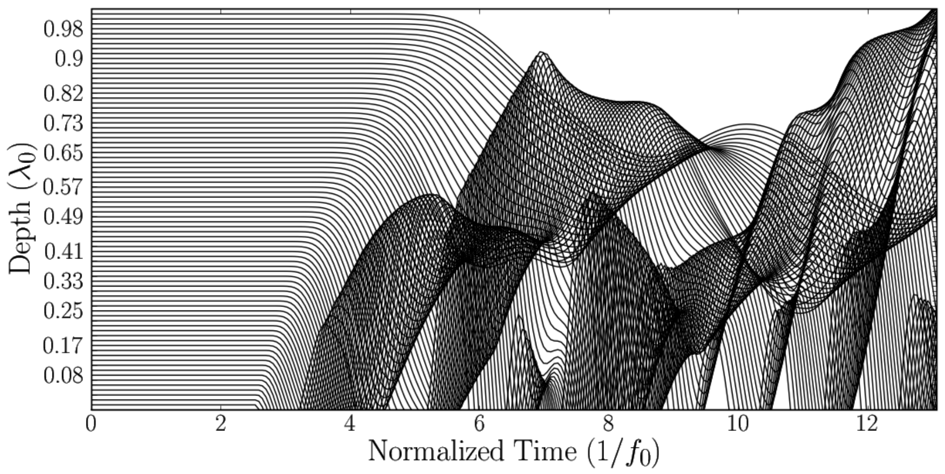


Figure E.72: Low frequency seismic plot for axisymmetric axis of specimen "m2b" using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$

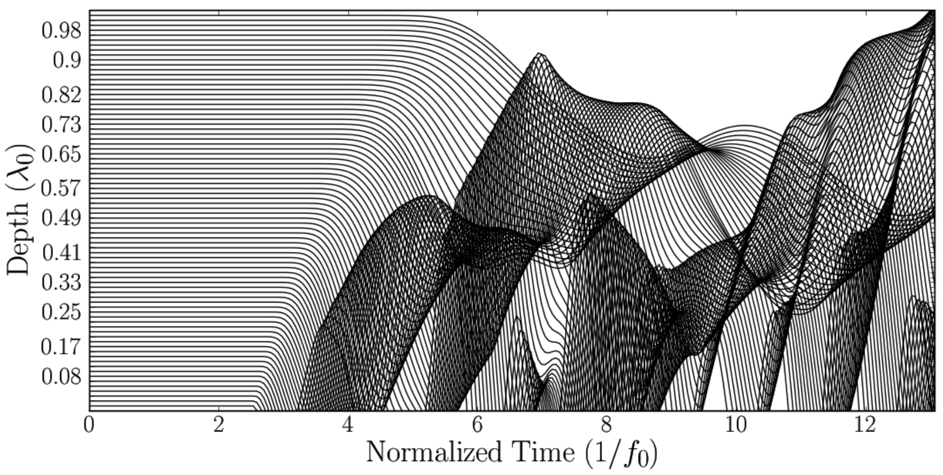


Figure E.73: Low frequency seismic plot for axisymmetric axis of specimen "m2c" using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$

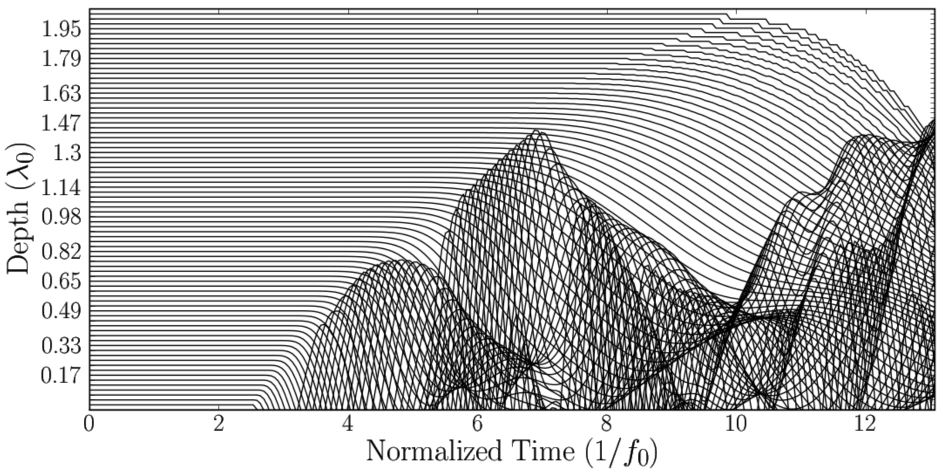


Figure E.74: Low frequency seismic plot for axisymmetric axis of specimen "m3a" using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$

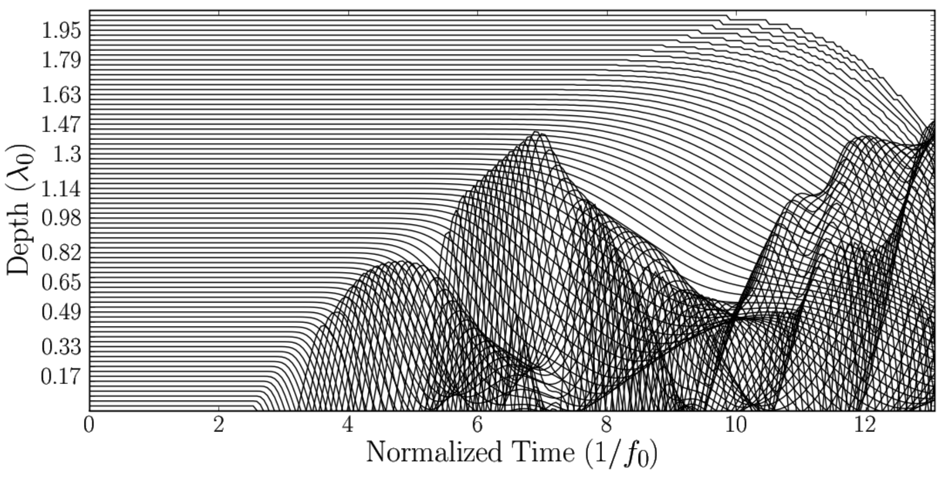


Figure E.75: Low frequency seismic plot for axisymmetric axis of specimen "m3b" using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$

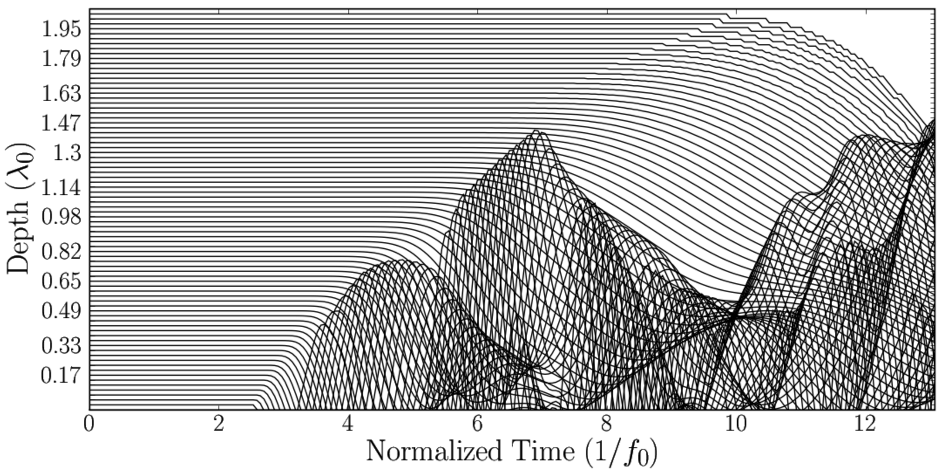


Figure E.76: Low frequency seismic plot for axisymmetric axis of specimen "m3c" using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$

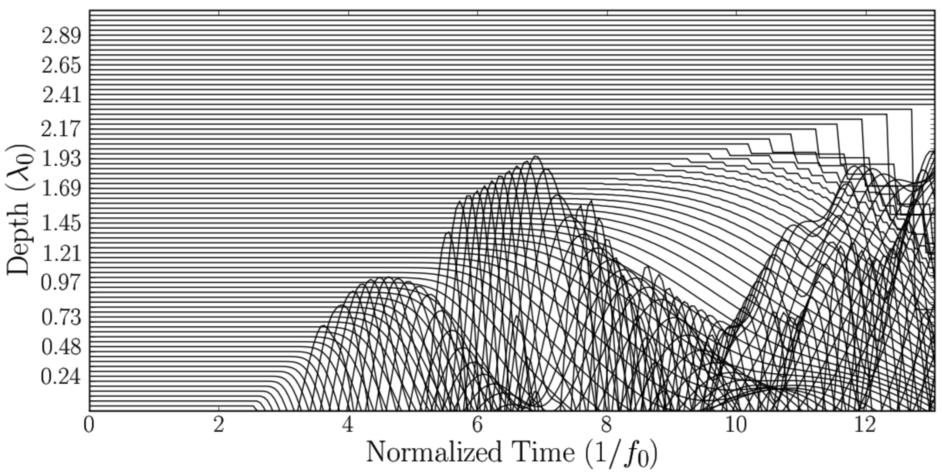


Figure E.77: Low frequency seismic plot for axisymmetric axis of specimen "m4a" using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$

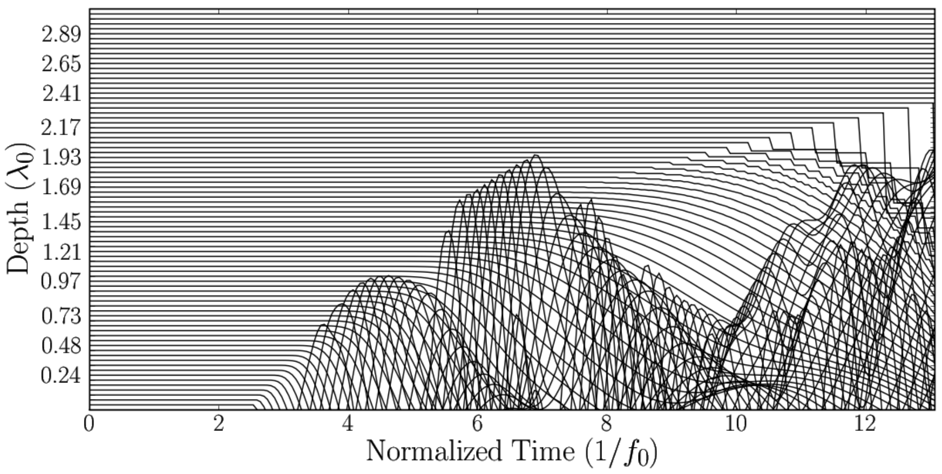


Figure E.78: Low frequency seismic plot for axisymmetric axis of specimen "m4b" using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$

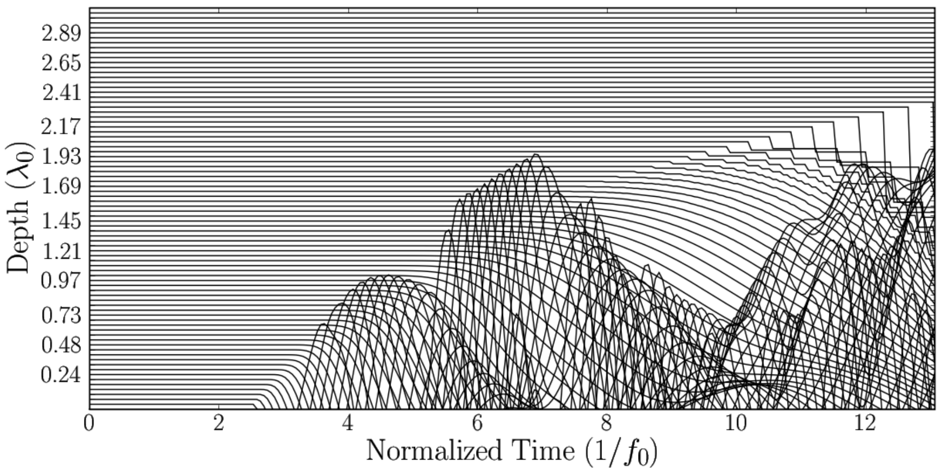


Figure E.79: Low frequency seismic plot for axisymmetric axis of specimen "m4c" using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$

Frequency Wavenumber Plots

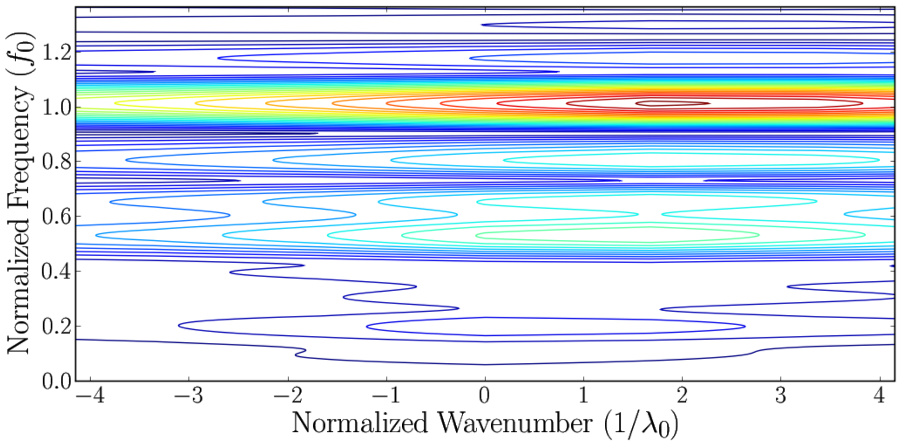


Figure E.80: Low frequency F-K plot for axisymmetric axis of specimen “m1a” using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$

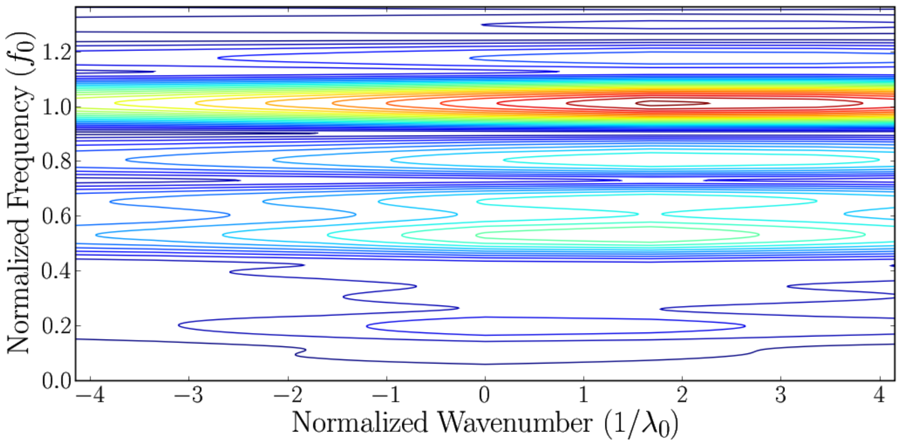


Figure E.81: Low frequency F-K plot for axisymmetric axis of specimen “m1b” using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$

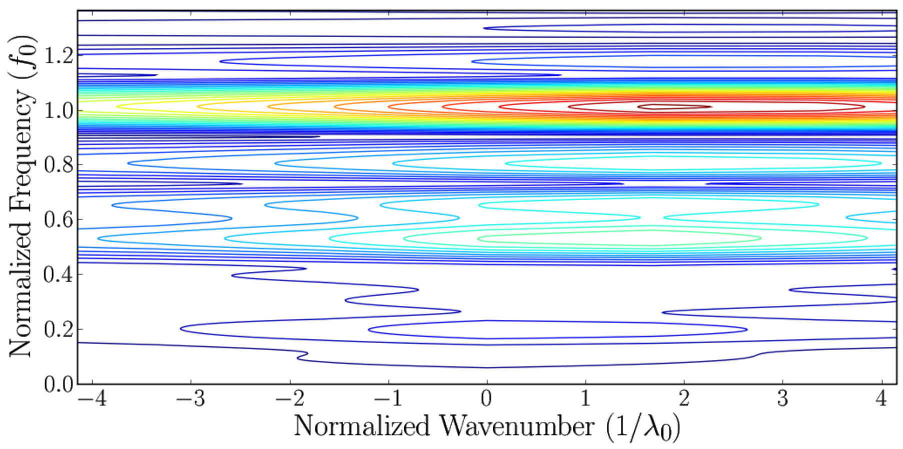


Figure E.82: Low frequency F-K plot for axisymmetric axis of specimen “m1c” using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$

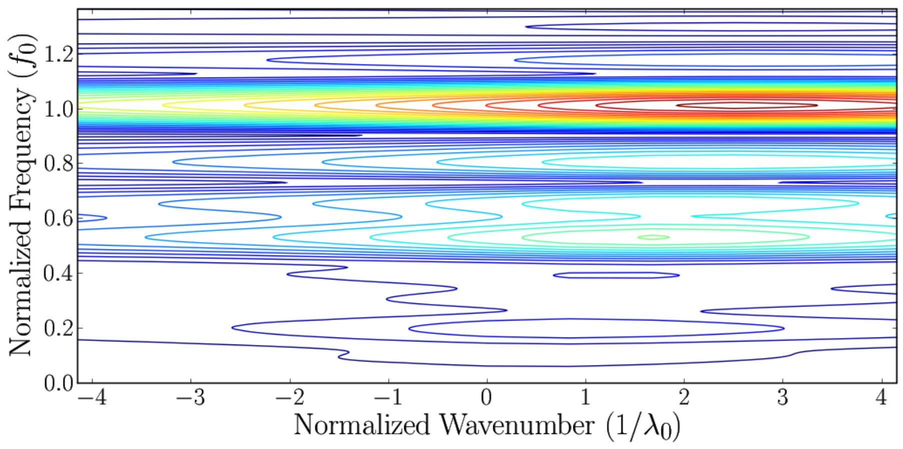


Figure E.83: Low frequency F-K plot for axisymmetric axis of specimen “m2a” using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$

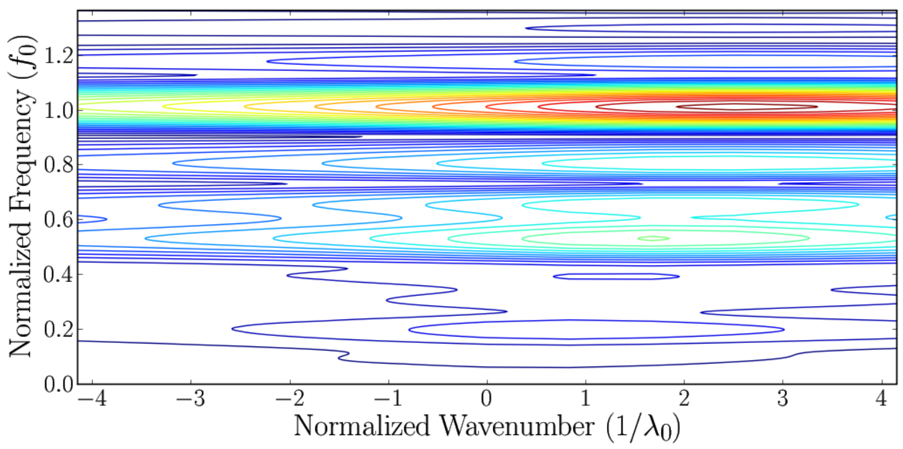


Figure E.84: Low frequency F-K plot for axisymmetric axis of specimen "m2b" using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$

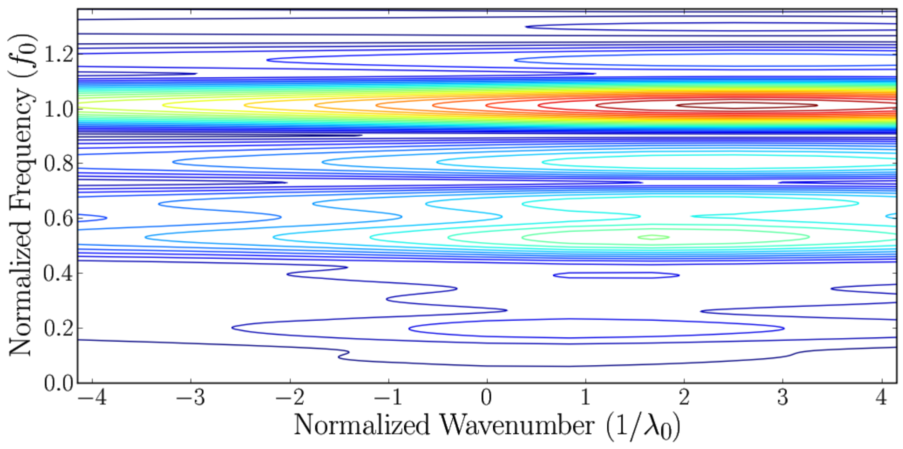


Figure E.85: Low frequency F-K plot for axisymmetric axis of specimen "m2c" using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$

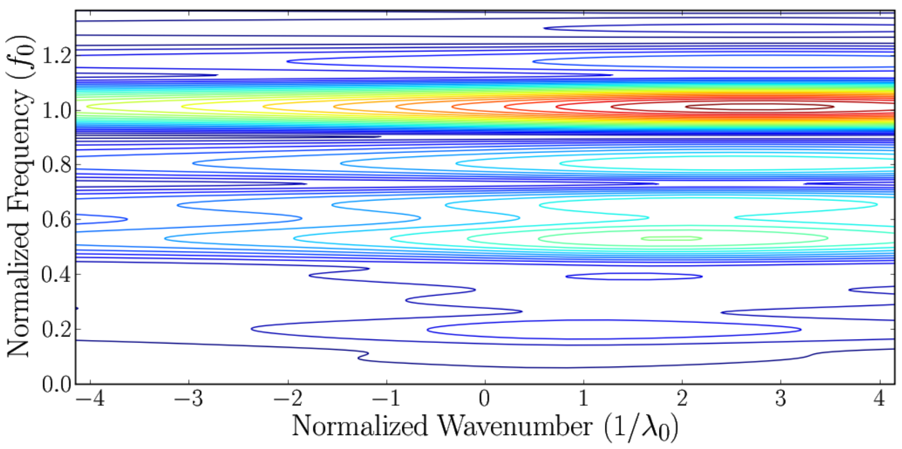


Figure E.86: Low frequency F-K plot for axisymmetric axis of specimen “m3a” using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$

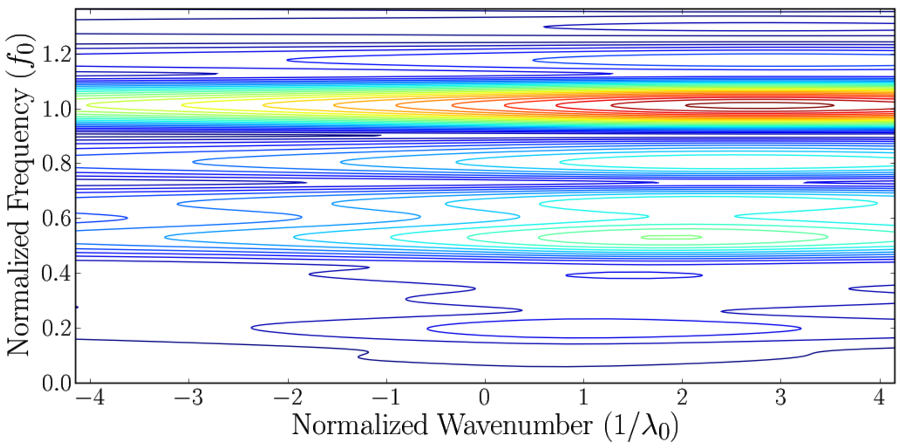


Figure E.87: Low frequency F-K plot for axisymmetric axis of specimen “m3b” using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$

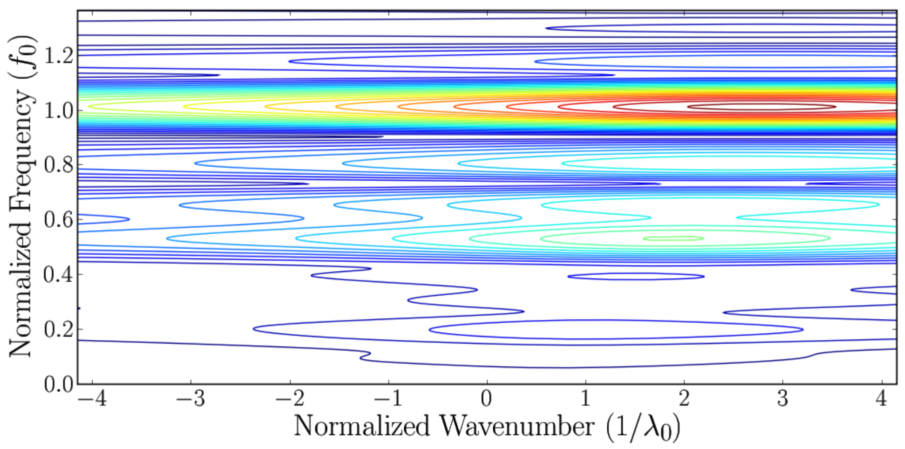


Figure E.88: Low frequency F-K plot for axisymmetric axis of specimen "m3c" using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$

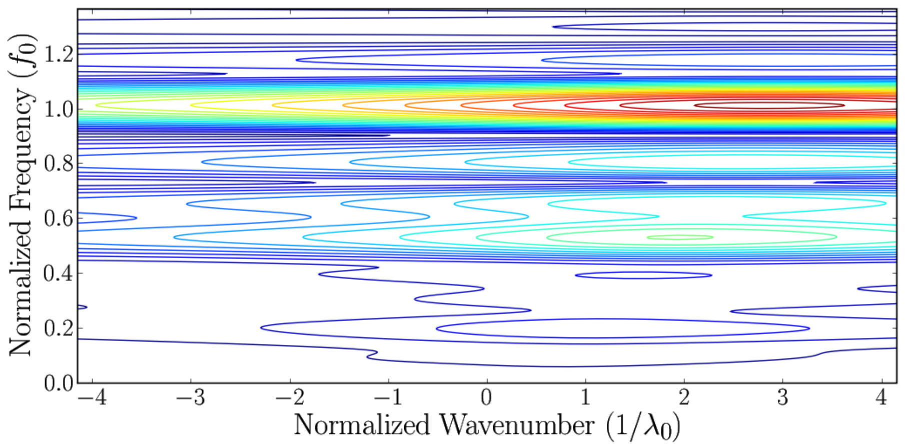


Figure E.89: Low frequency F-K plot for axisymmetric axis of specimen "m4a" using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$

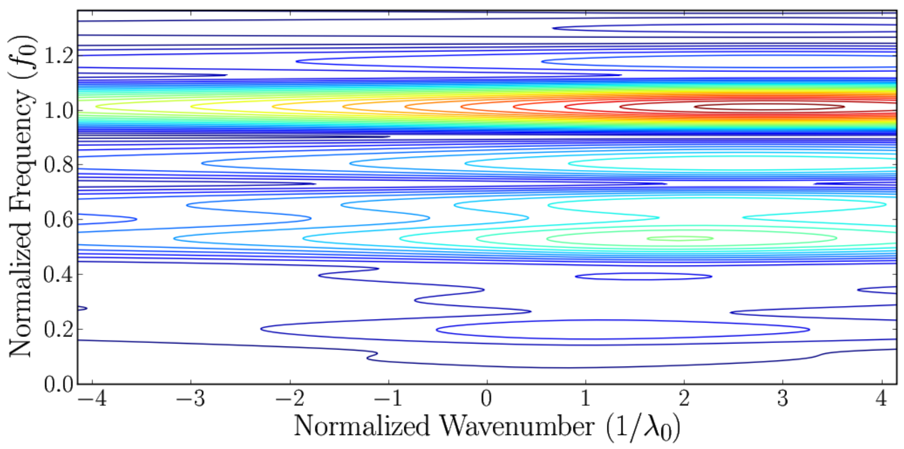


Figure E.90: Low frequency F-K plot for axisymmetric axis of specimen "m4b" using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$

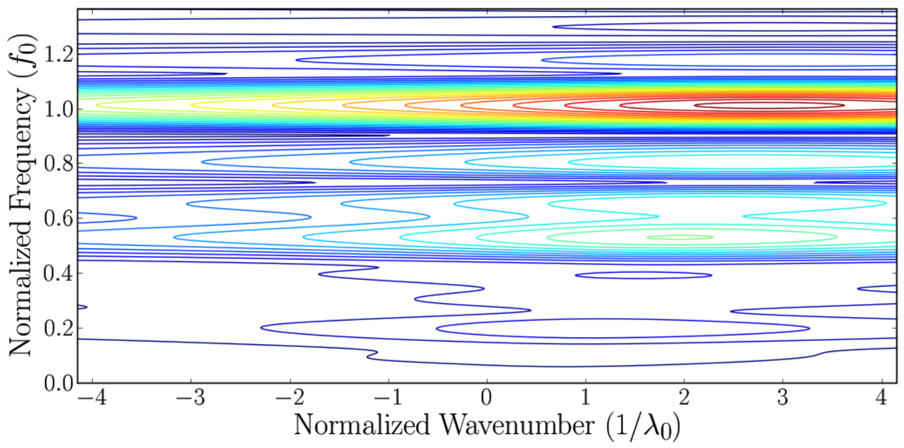


Figure E.91: Low frequency F-K plot for axisymmetric axis of specimen "m4c" using damping coefficients $\alpha = 4435.2, \beta = 1.5 * 10^{-7}$

E.2.4 Damped vs Undamped Time traces

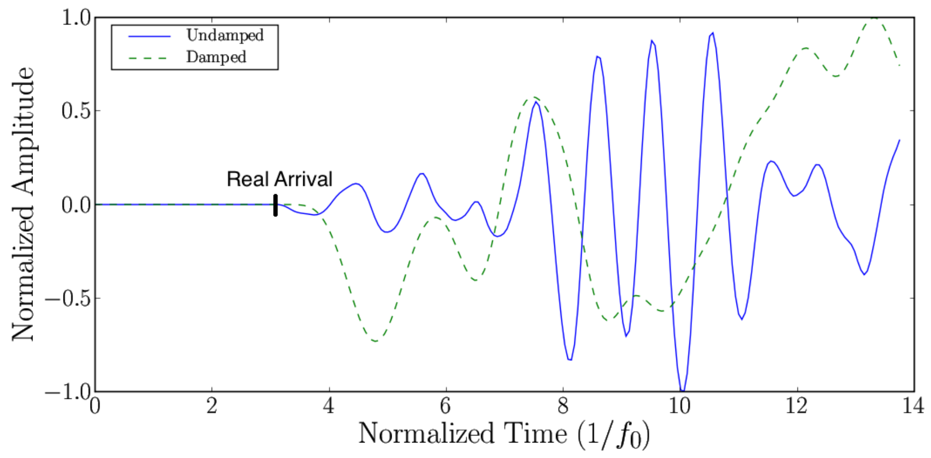


Figure E.92: Comparison of time signal between damped and undamped model 'm1a' at low frequency

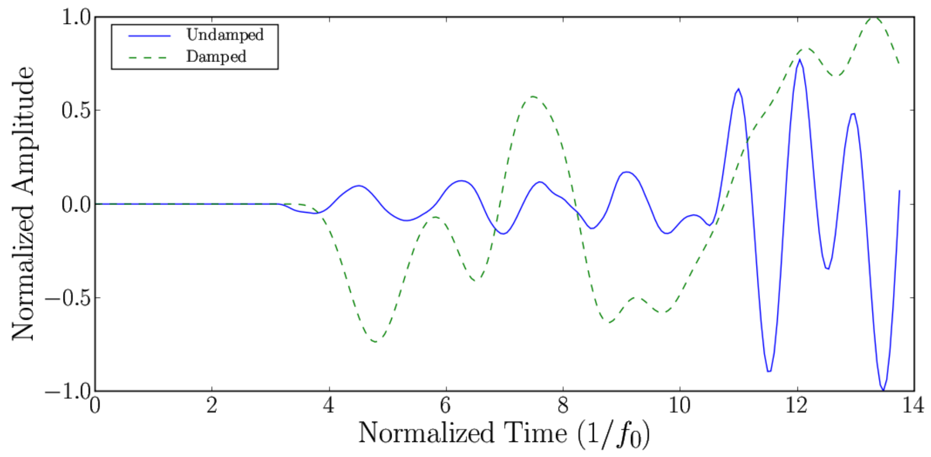


Figure E.93: Comparison of time signal between damped and undamped model 'm1b' at low frequency

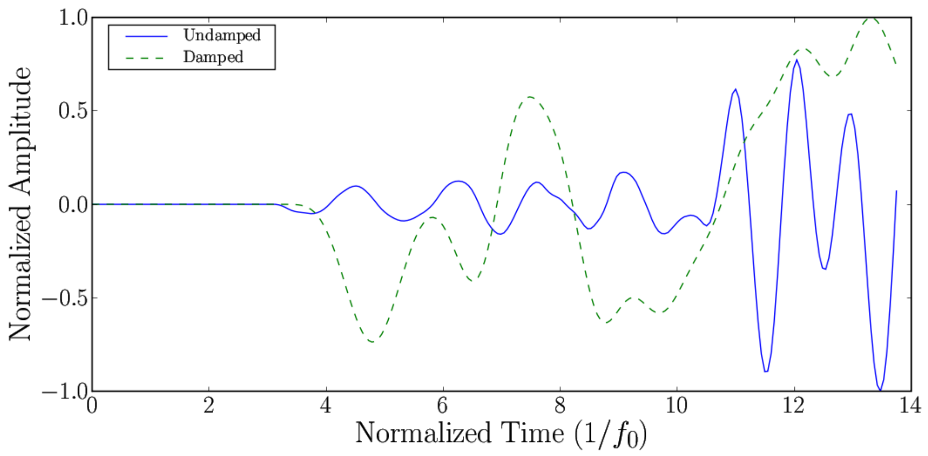


Figure E.94: Comparison of time signal between damped and undamped model 'm1b' at low frequency

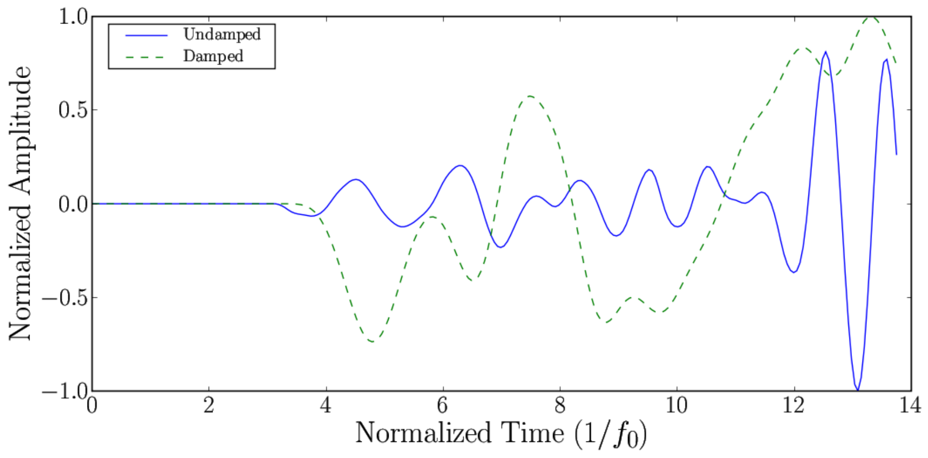


Figure E.95: Comparison of time signal between damped and undamped model 'm1c' at low frequency

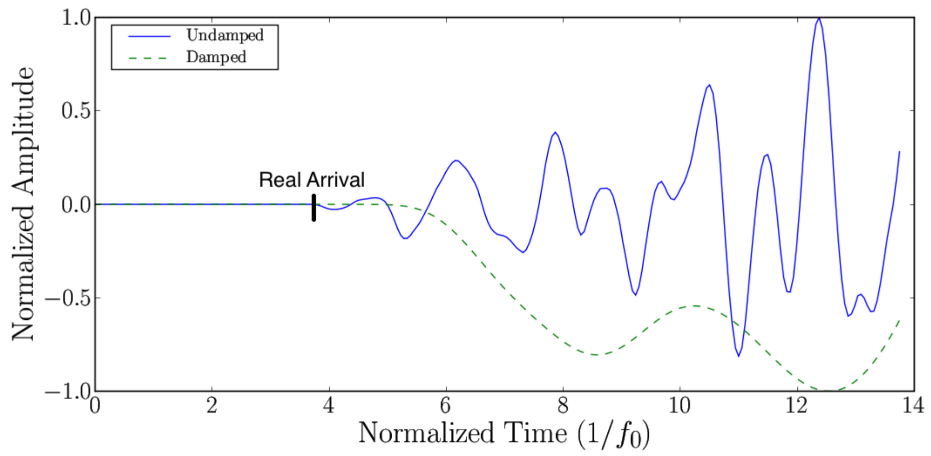


Figure E.96: Comparison of time signal between damped and undamped model 'm2a' at low frequency

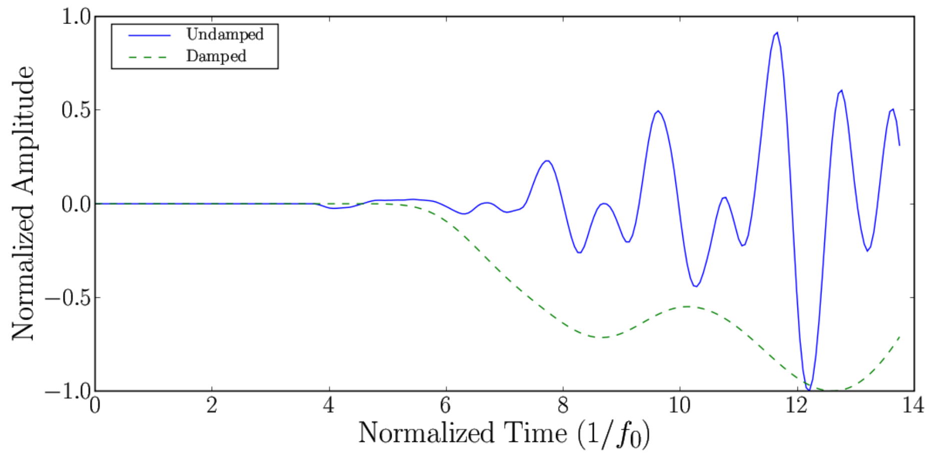


Figure E.97: Comparison of time signal between damped and undamped model 'm2b' at low frequency

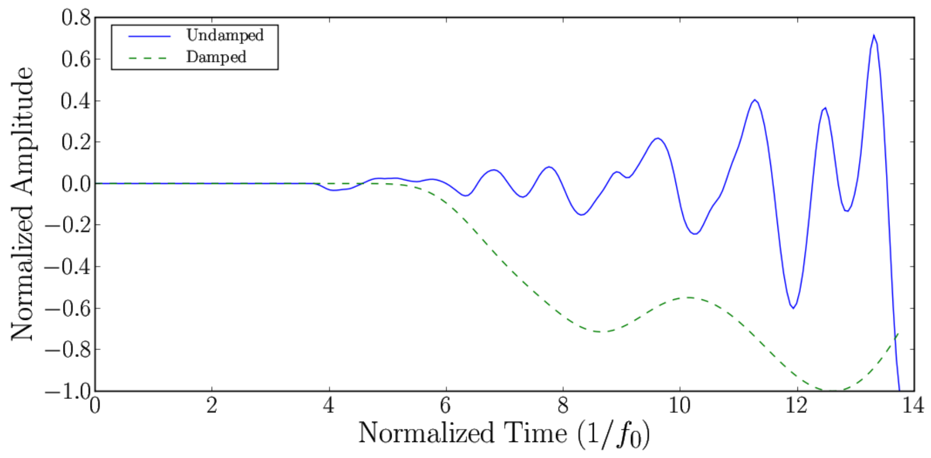


Figure E.98: Comparison of time signal between damped and undamped model 'm2c' at low frequency

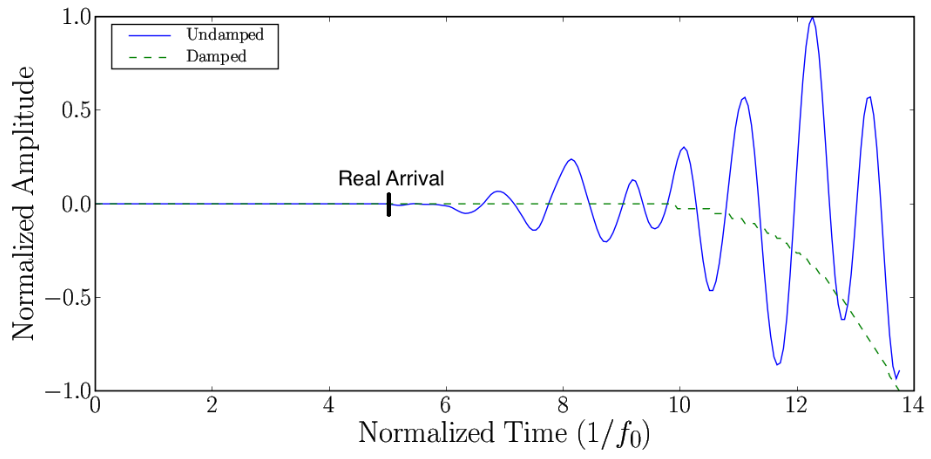


Figure E.99: Comparison of time signal between damped and undamped model 'm3a' at low frequency

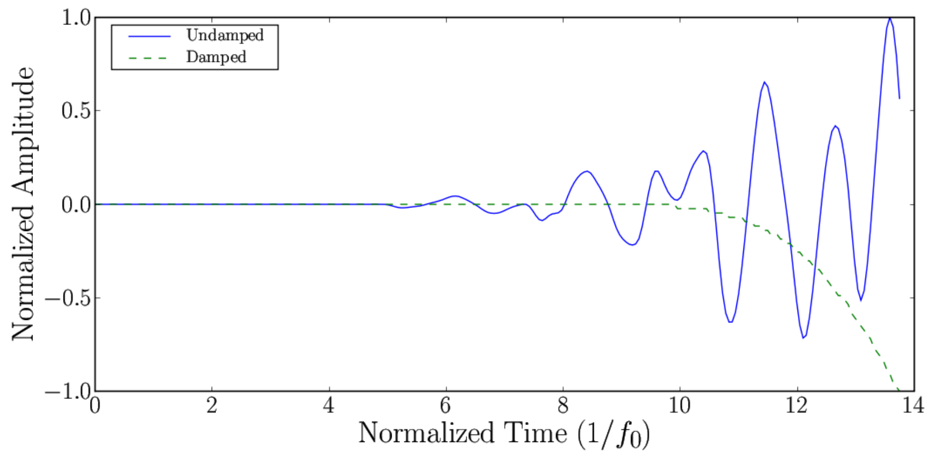


Figure E.100: Comparison of time signal between damped and undamped model 'm3b' at low frequency

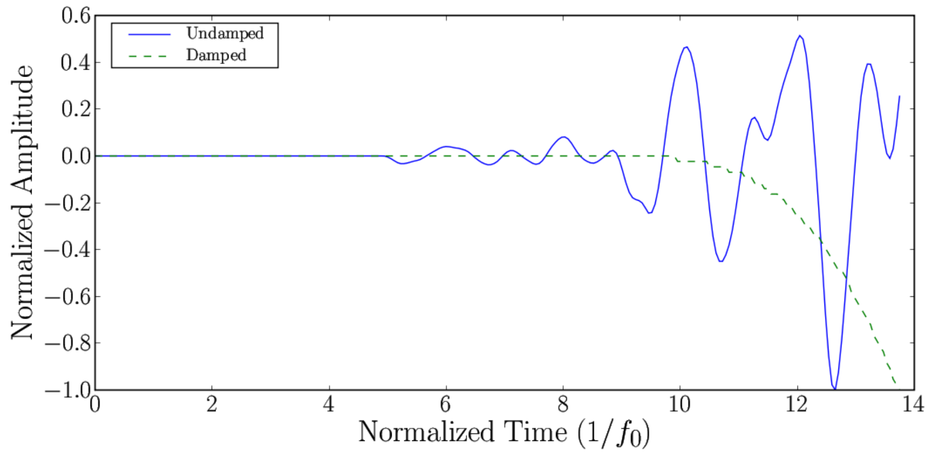


Figure E.101: Comparison of time signal between damped and undamped model 'm3c' at low frequency

Appendix F

Experimental Data

F.1 Experimental Time Traces

F.1.1 High Frequency

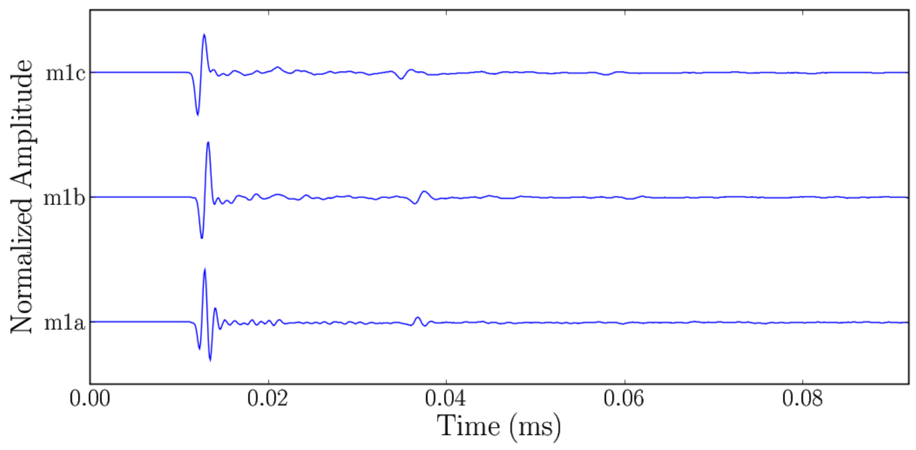


Figure F.1: High frequency experimental time trace for 5 cm high mortar specimens

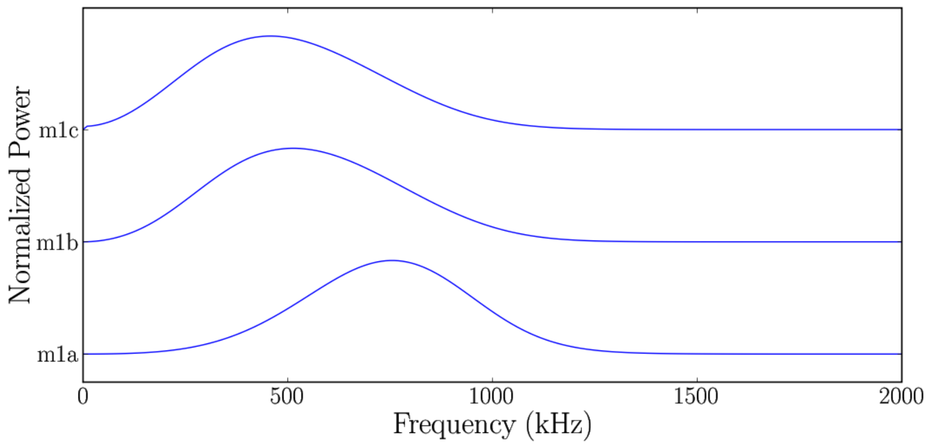


Figure F.2: Frequency spectrum of high frequency experimental time trace for 5 cm high mortar specimens

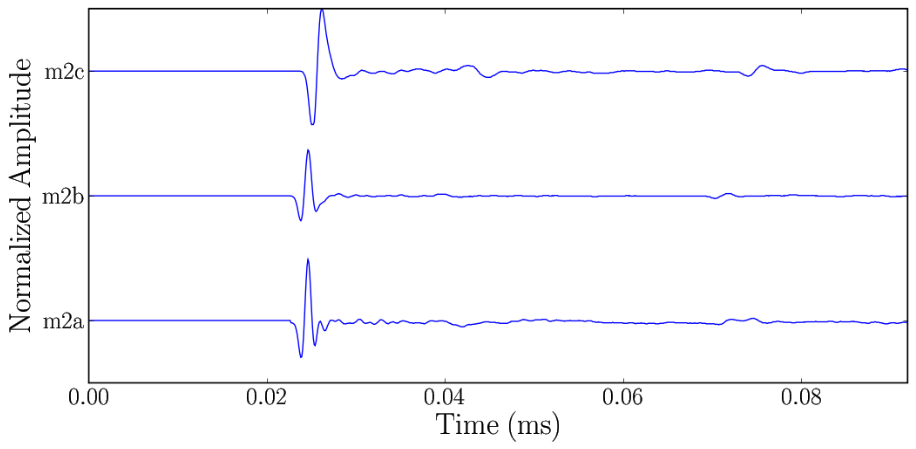


Figure F.3: High frequency experimental time trace for 5 cm high mortar specimens

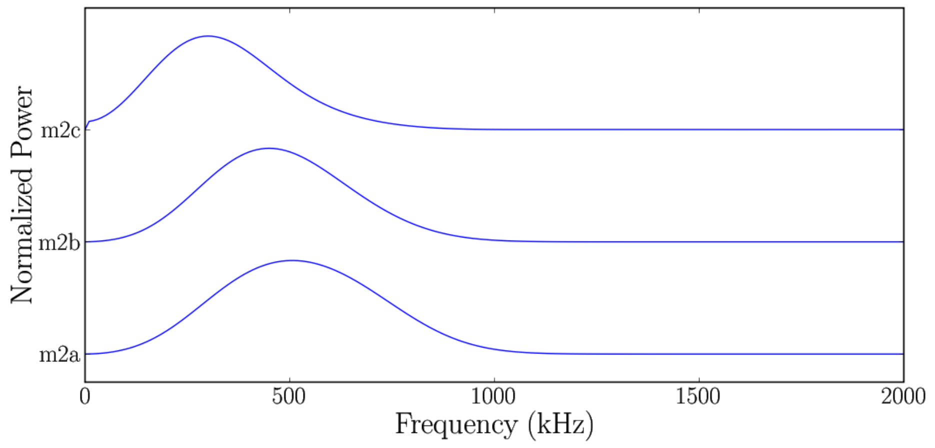


Figure F.4: Frequency spectrum of high frequency experimental time trace for 10 cm high mortar specimens

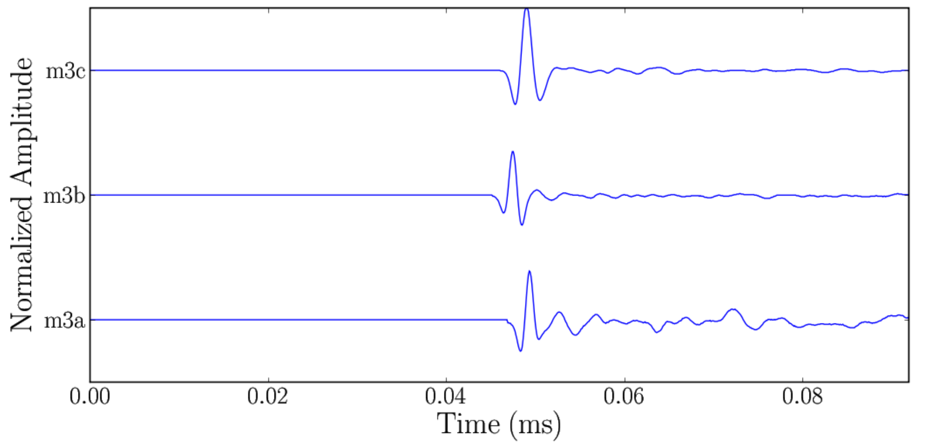


Figure F.5: High frequency experimental time trace for 20 cm high mortar specimens

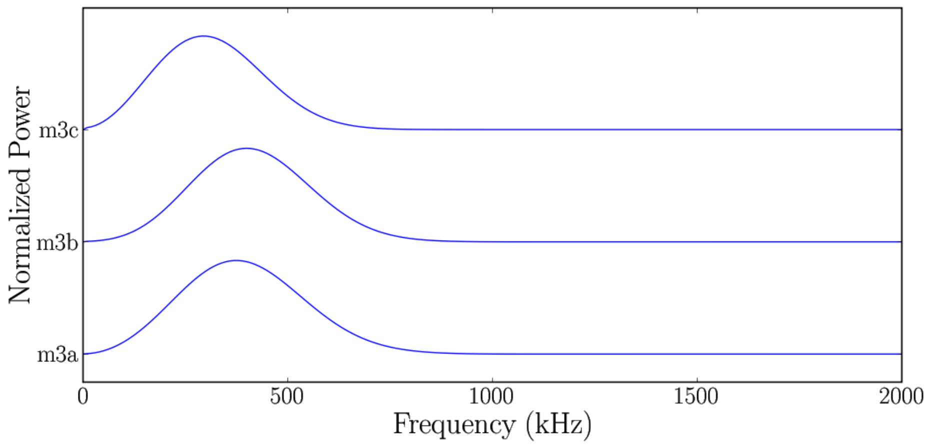


Figure F.6: Frequency spectrum of high frequency experimental time trace for 20 cm high mortar specimens

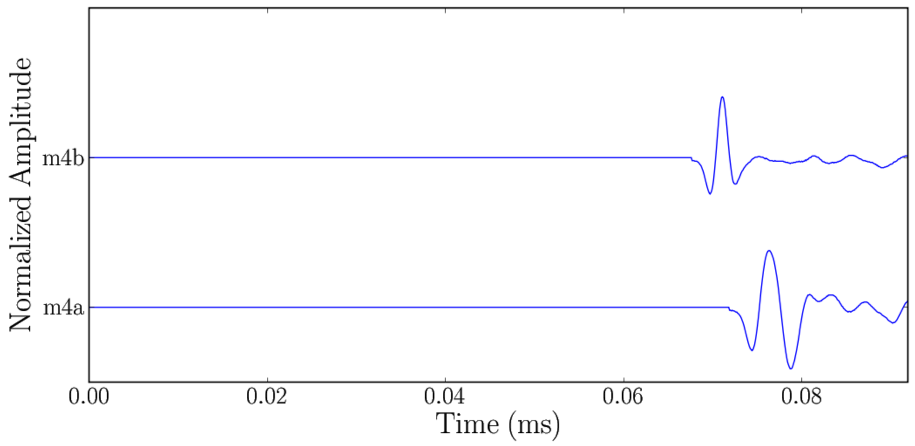


Figure F.7: High frequency experimental time trace for 30 cm high mortar specimens

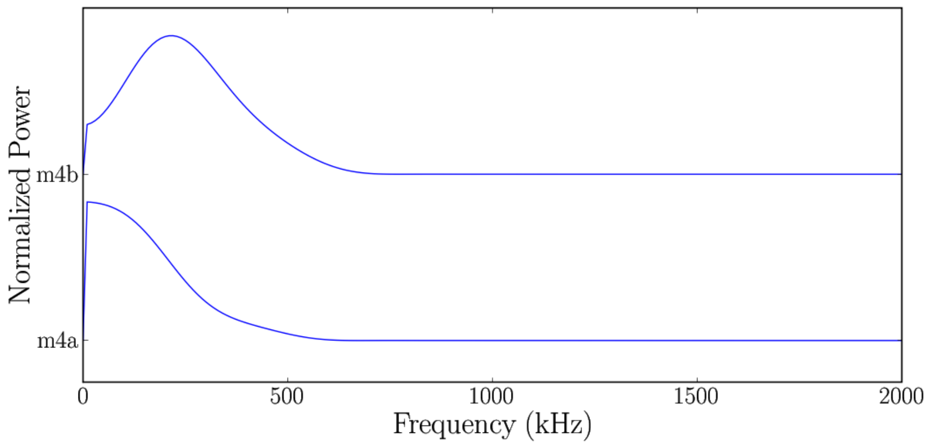


Figure F.8: Frequency spectrum of high frequency experimental time trace for 30 cm high mortar specimens

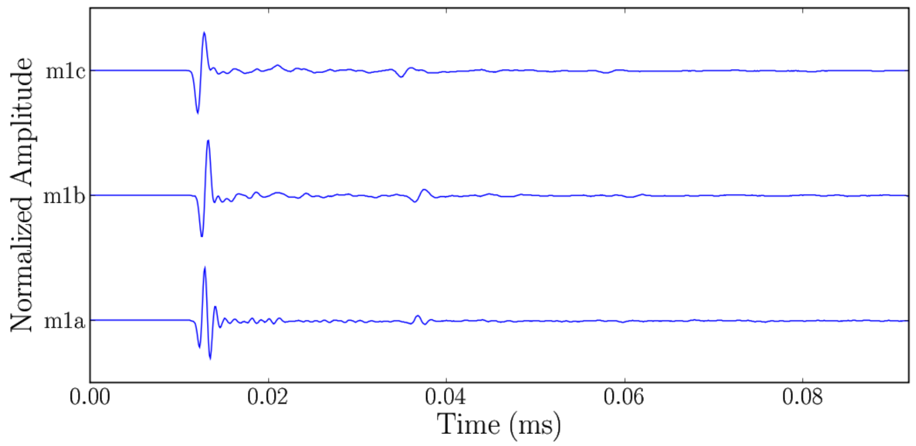


Figure F.9: High frequency experimental time trace for 5 cm high mortar specimens

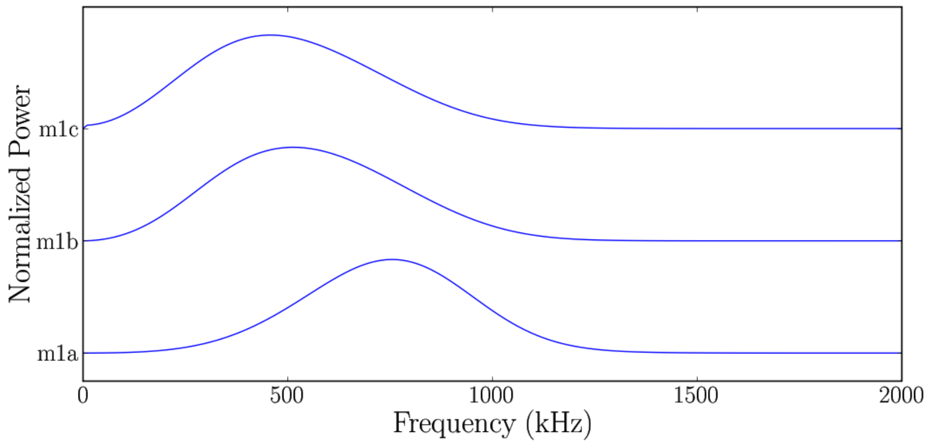


Figure F.10: Frequency spectrum of high frequency experimental time trace for 5 cm high mortar specimens

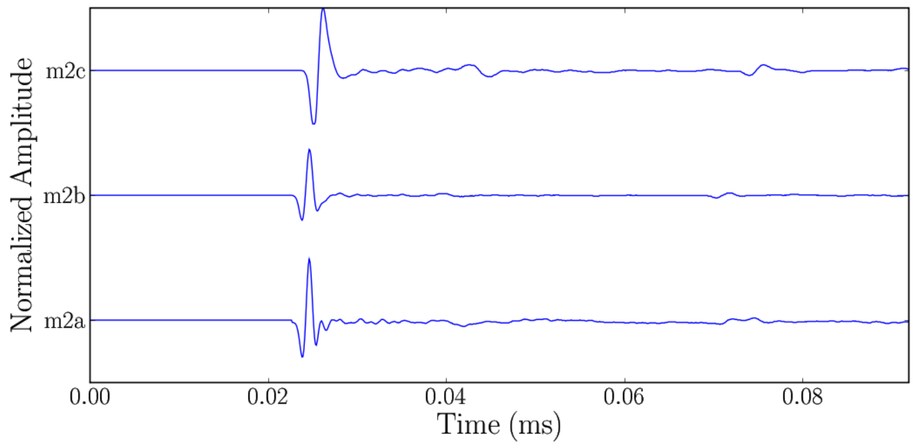


Figure F.11: High frequency experimental time trace for 5 cm high mortar specimens

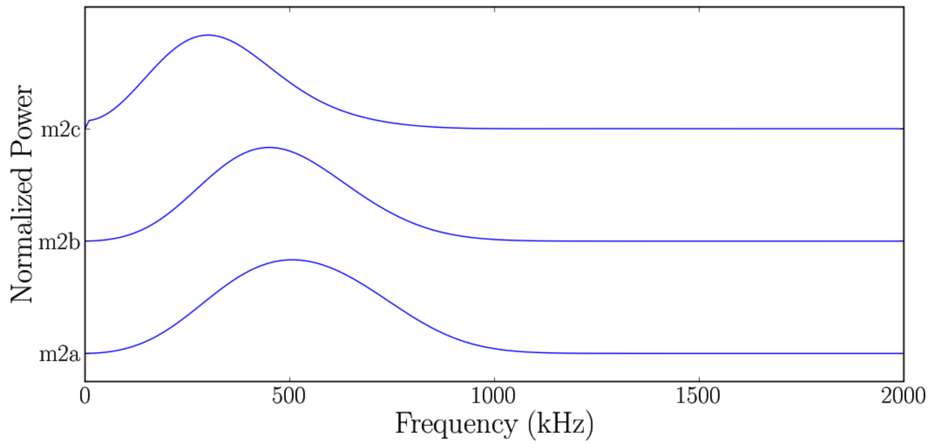


Figure F.12: Frequency spectrum of high frequency experimental time trace for 10 cm high mortar specimens

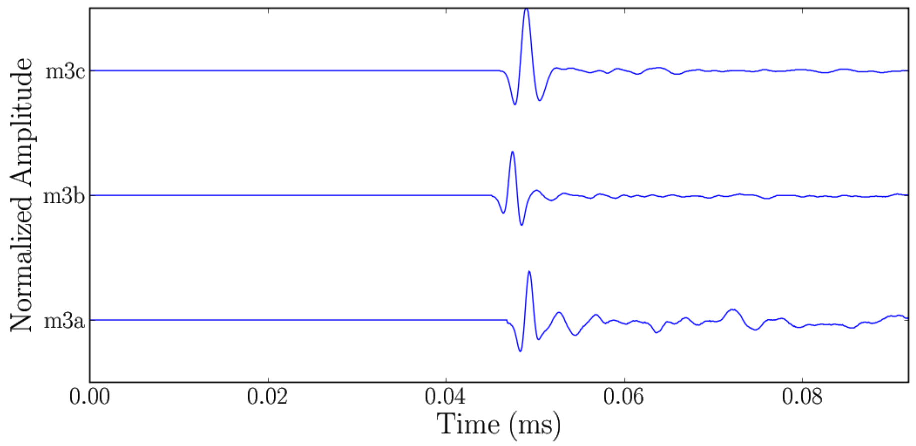


Figure F.13: High frequency experimental time trace for 20 cm high mortar specimens

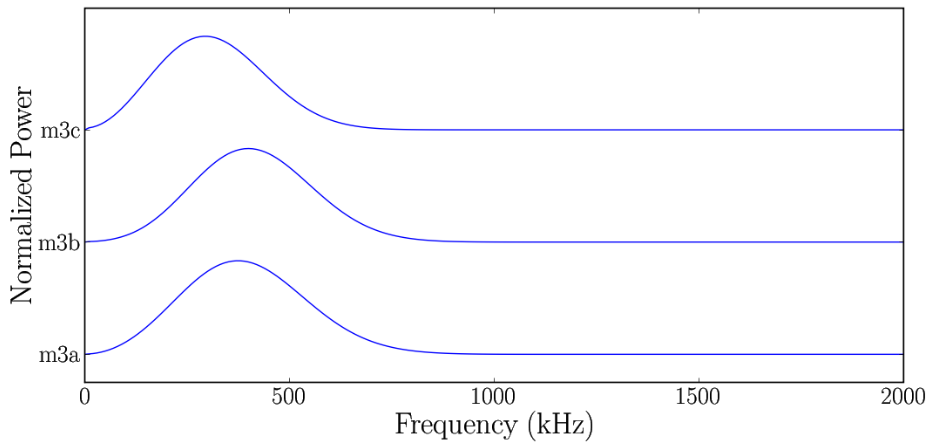


Figure F.14: Frequency spectrum of high frequency experimental time trace for 20 cm high mortar specimens

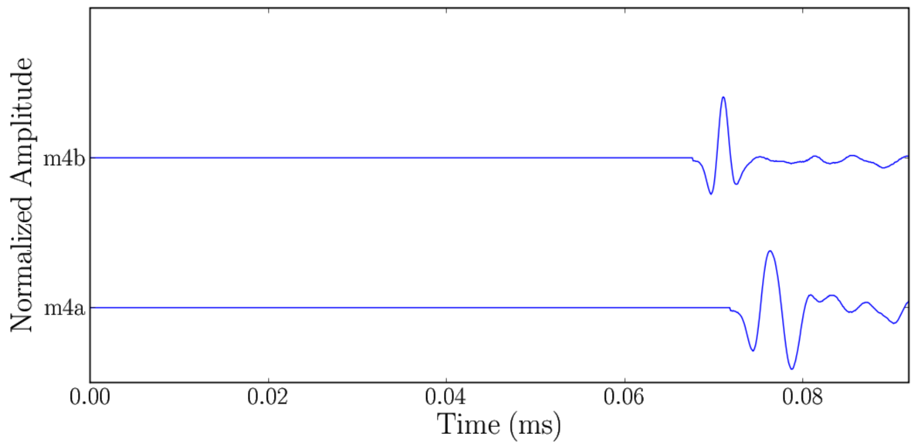


Figure F.15: High frequency experimental time trace for 30 cm high mortar specimens

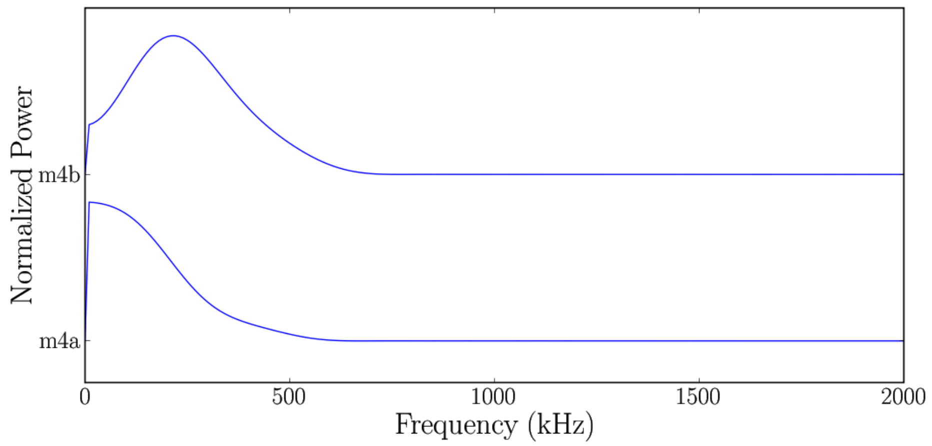


Figure F.16: Frequency spectrum of high frequency experimental time trace for 30 cm high mortar specimens

F.1.2 Low Frequency

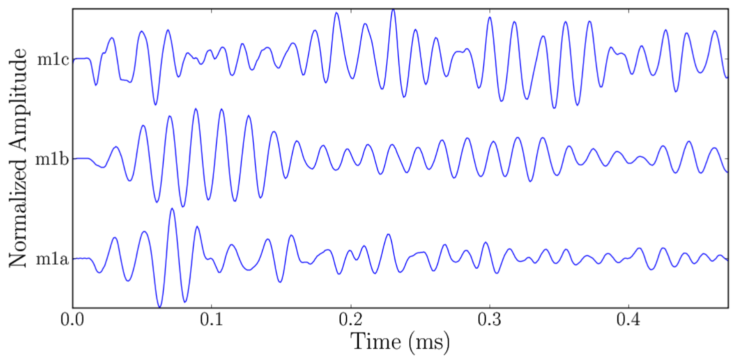


Figure F.17: Low frequency experimental time trace for 5 cm high mortar specimens

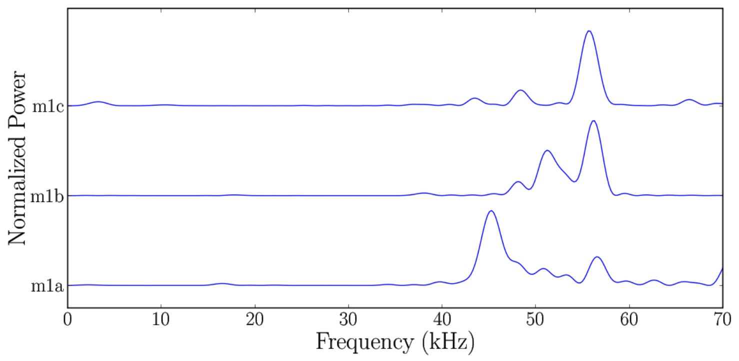


Figure F.18: Frequency spectrum of low frequency experimental time trace for 5 cm high mortar specimens

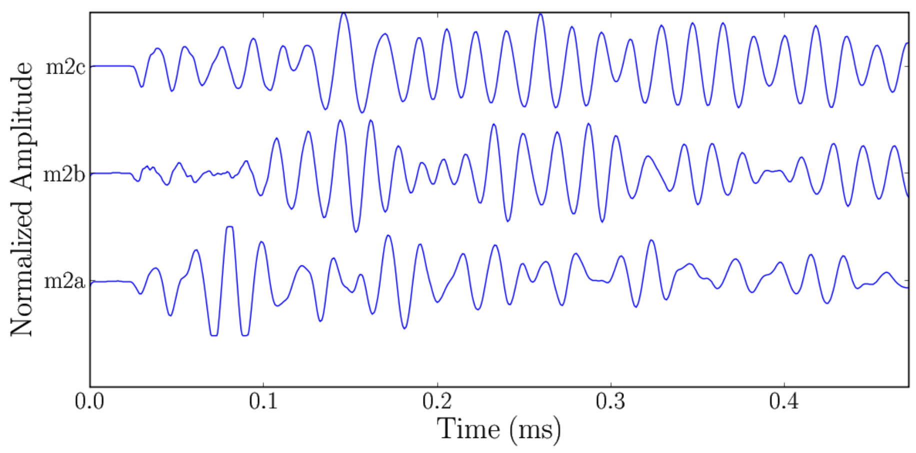


Figure F.19: Low frequency experimental time trace for 5 cm high mortar specimens

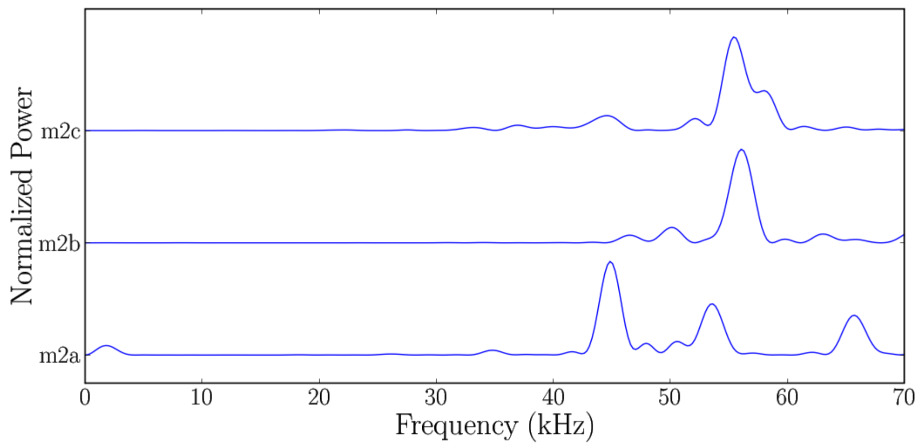


Figure F.20: Frequency spectrum of low frequency experimental time trace for 10 cm high mortar specimens

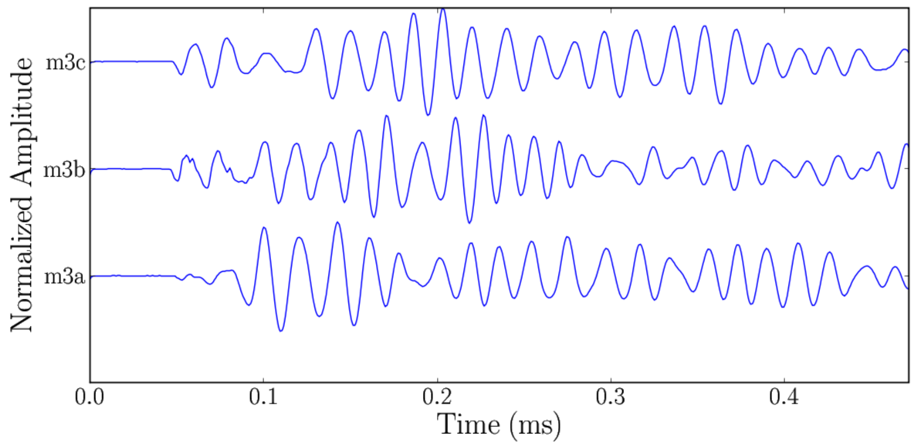


Figure F.21: Low frequency experimental time trace for 20 cm high mortar specimens

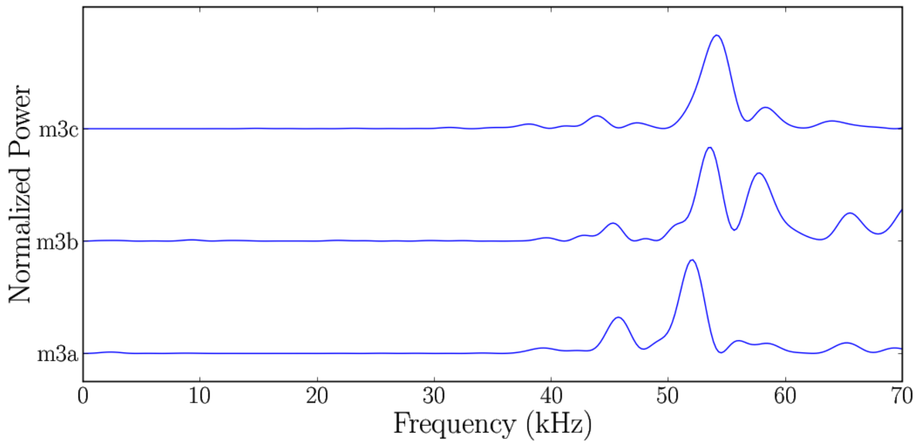


Figure F.22: Frequency spectrum of low frequency experimental time trace for 20 cm high mortar specimens

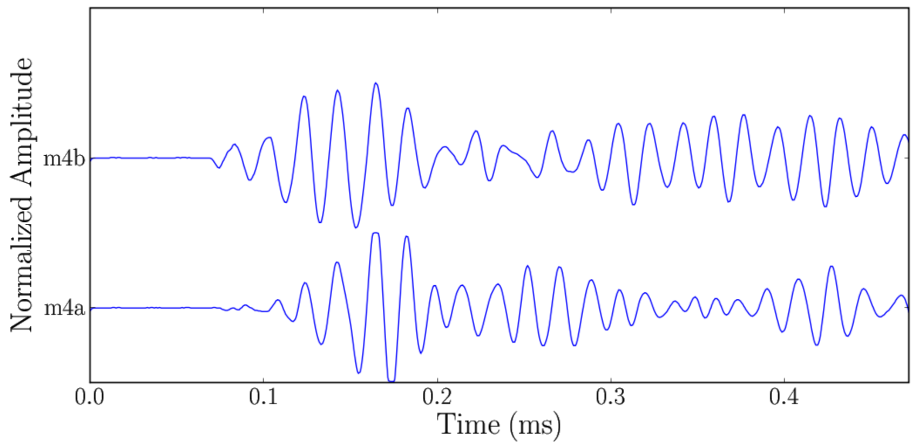


Figure F.23: Low frequency experimental time trace for 30 cm high mortar specimens

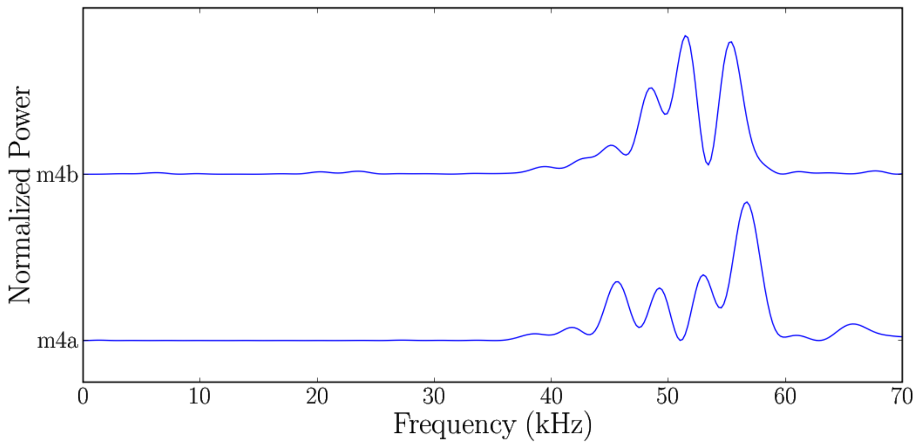


Figure F.24: Frequency spectrum of low frequency experimental time trace for 30 cm high mortar specimens

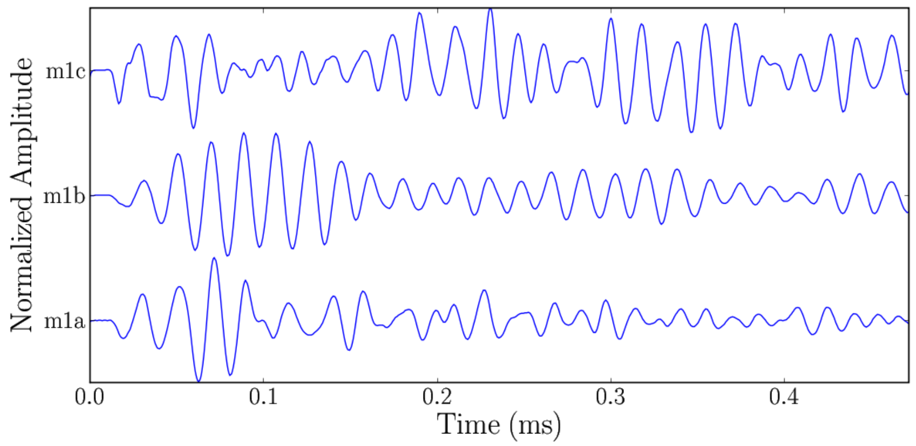


Figure F.25: Low frequency experimental time trace for 5 cm high mortar specimens

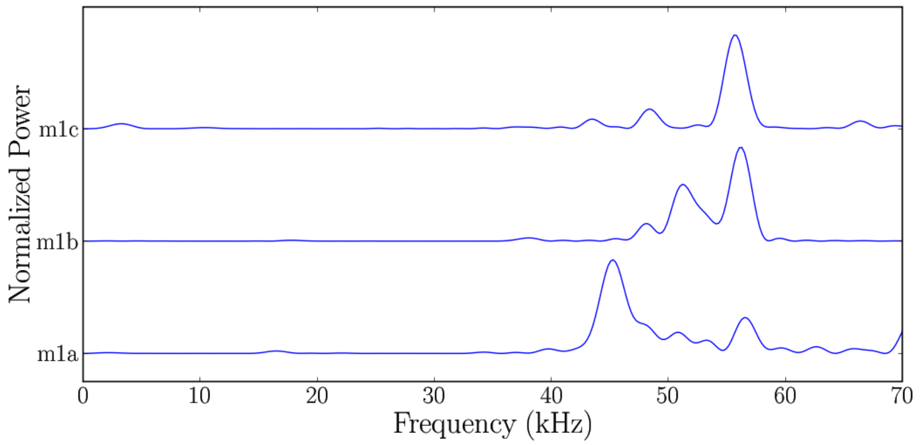


Figure F.26: Frequency spectrum of low frequency experimental time trace for 5 cm high mortar specimens

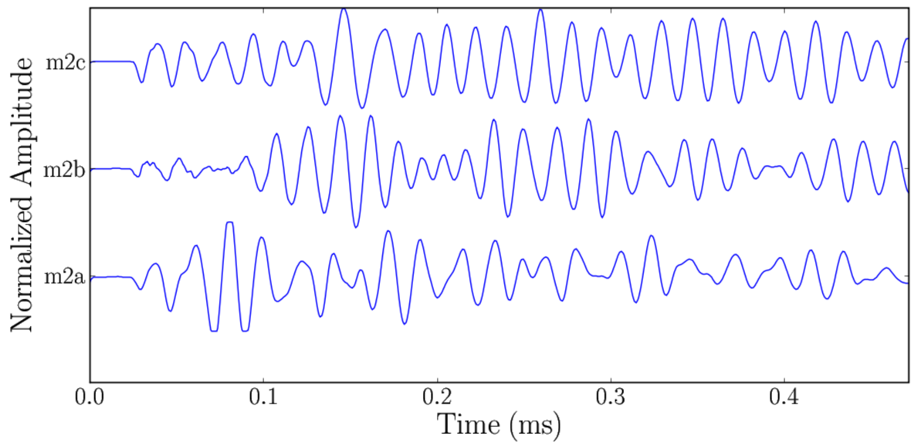


Figure F.27: Low frequency experimental time trace for 5 cm high mortar specimens

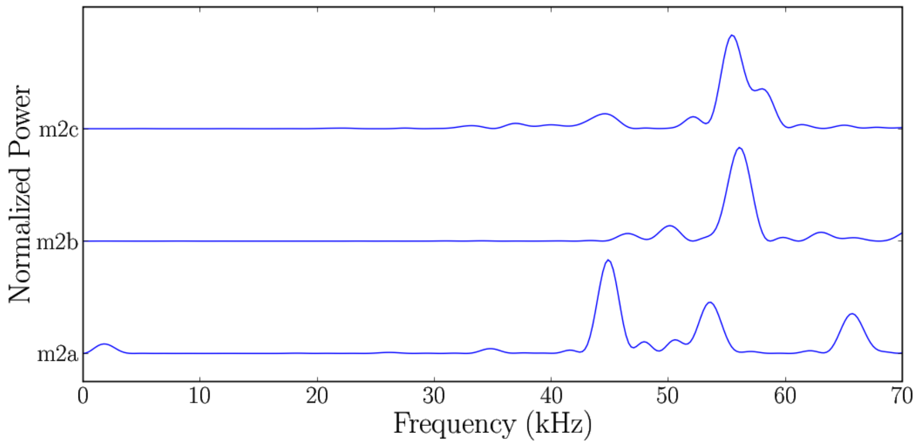


Figure F.28: Frequency spectrum of low frequency experimental time trace for 10 cm high mortar specimens

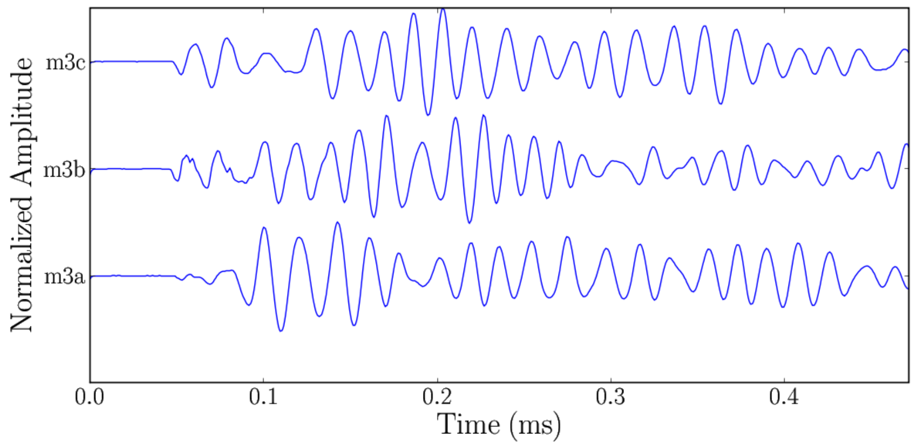


Figure F.29: Low frequency experimental time trace for 20 cm high mortar specimens

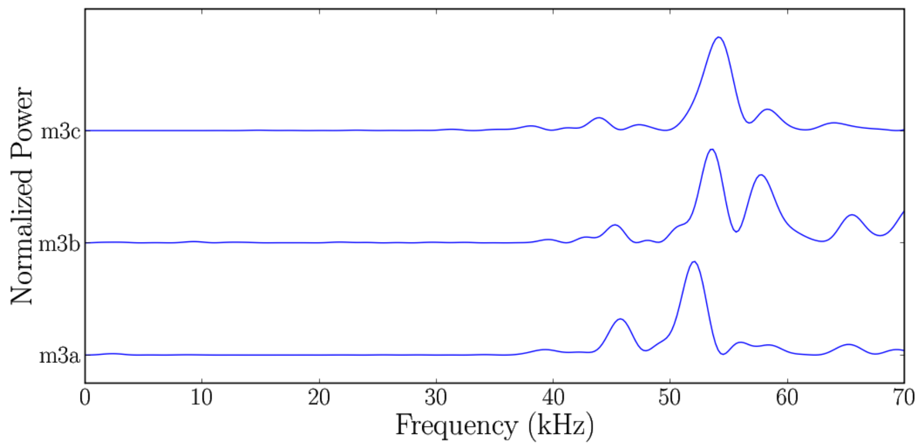


Figure F.30: Frequency spectrum of low frequency experimental time trace for 20 cm high mortar specimens

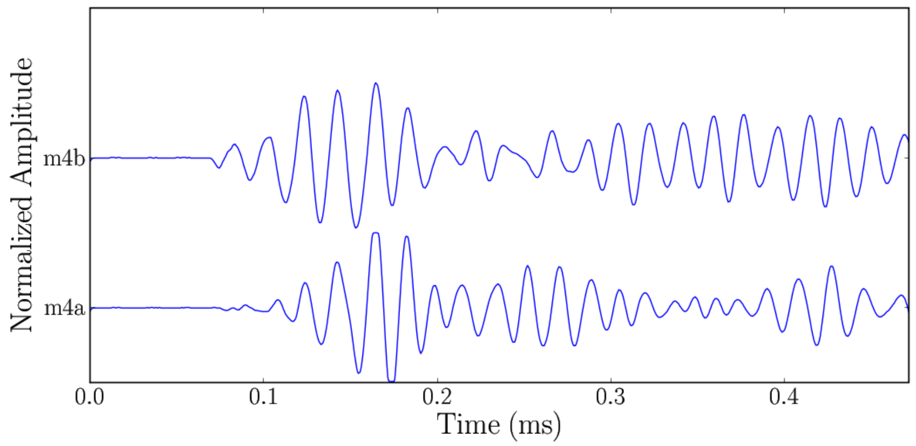


Figure F.31: Low frequency experimental time trace for 30 cm high mortar specimens

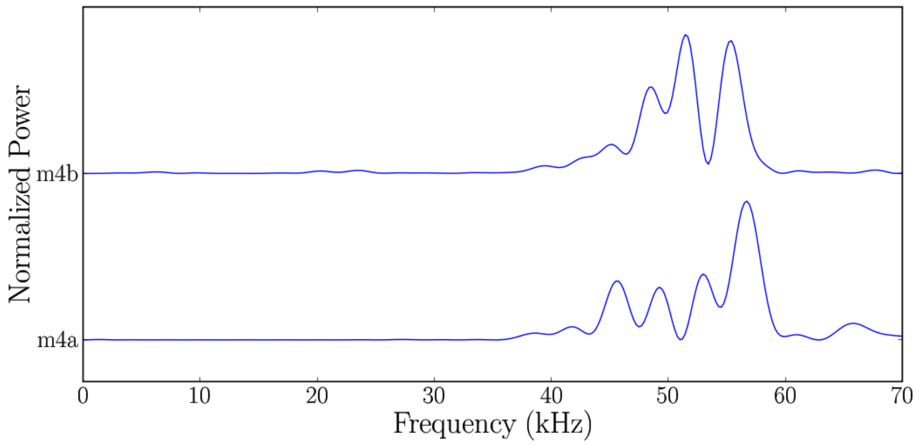


Figure F.32: Frequency spectrum of low frequency experimental time trace for 30 cm high mortar specimens

F.1.3 First Arrivals

Table F.1: Raw data of velocity reading for 5 cm high mortar samples

Model	Reading #	t_{1M} (ms)	Error (ms)	t_{50K} (ms)	Error (ms)
m1a	1	11.2	0.1	12.6	0.1
m1a	2	11.1	0.1	12.7	0.1
m1a	3	11.3	0.1	12.5	0.2
m1a	4	11.2	0.1	12.6	0.2
m1a	5	11.2	0.1	12.5	0.2
m1a	AVG	11.20	0.05	12.58	0.08
m1b	1	11.3	0.1	11.7	0.2
m1b	2	11.3	0.1	11.6	0.2
m1b	3	11.4	0.1	11.5	0.2
m1b	4	11.4	0.1	11.5	0.2
m1b	5	11.3	0.1	11.4	0.2
m1b	AVG	11.34	0.05	11.54	0.09
m1c	1	10.7	0.1	11.0	0.2
m1c	2	10.8	0.1	11.0	0.2
m1c	3	10.8	0.1	10.9	0.2
m1c	4	10.7	0.1	10.8	0.2
m1c	5	10.8	0.1	10.9	0.2
m1c	AVG	10.76	0.05	10.92	0.09

Table F.2: Raw data of velocity reading for 10 cm high mortar samples

Model	Reading #	t_{1M} (ms)	Error (ms)	t_{50K} (ms)	Error (ms)
m2a	1	22.0	0.1	23.4	0.1
m2a	2	22.1	0.1	23.6	0.2
m2a	3	22.2	0.1	23.5	0.2
m2a	4	22.1	0.1	23.4	0.2
m2a	5	22.1	0.1	23.4	0.2
m2a	AVG	22.10	0.05	23.46	0.09
m2b	1	22.3	0.1	22.6	0.2
m2b	2	22.2	0.1	22.5	0.2
m2b	3	22.3	0.1	22.4	0.2
m2b	4	22.2	0.1	22.3	0.2
m2b	5	22.3	0.1	22.5	0.2
m2b	AVG	22.26	0.05	22.46	0.09
m2c	1	23.3	0.1	23.8	0.2
m2c	2	23.2	0.1	23.6	0.2
m2c	3	23.2	0.1	23.7	0.2
m2c	4	23.2	0.1	23.5	0.2
m2c	5	23.2	0.1	23.6	0.2
m2c	AVG	23.22	0.05	23.64	0.09

Table F.3: Raw data of velocity reading for 20 cm high mortar samples

Model	Reading #	t_{1M} (ms)	Error (ms)	t_{50K} (ms)	Error (ms)
m3a	1	45.6	0.1	47.1	0.2
m3a	2	45.7	0.1	47.1	0.2
m3a	3	45.7	0.1	47.1	0.2
m3a	4	45.6	0.1	47.0	0.2
m3a	5	45.6	0.1	47.1	0.2
m3a	AVG	45.64	0.05	47.08	0.09
m3b	1	44.5	0.1	45.3	0.2
m3b	2	44.4	0.1	45.4	0.2
m3b	3	44.5	0.1	45.4	0.2
m3b	4	44.4	0.1	45.6	0.2
m3b	5	44.4	0.1	45.4	0.2
m3b	AVG	44.44	0.05	45.42	0.09
m3c	1	45.4	0.1	46.5	0.2
m3c	2	45.3	0.1	46.6	0.2
m3c	3	45.4	0.1	46.4	0.2
m3c	4	45.4	0.1	46.4	0.2
m3c	5	45.4	0.1	46.3	0.2
m3c	AVG	45.38	0.05	46.44	0.09

Table F.4: Raw data of velocity reading for 30 cm high mortar samples

Model	Reading #	t_{1M} (ms)	Error (ms)	t_{50K} (ms)	Error (ms)
m4a	1	69.4	0.2	71.2	0.2
m4a	2	69.5	0.2	71.2	0.2
m4a	3	69.4	0.2	71.2	0.2
m4a	4	69.5	0.2	71.1	0.2
m4a	5	69.5	0.2	71.2	0.2
m4a	AVG	69.46	0.09	71.18	0.09
m4b	1	67.1	0.2	68.2	0.2
m4b	2	67.0	0.2	68.0	0.2
m4b	3	67.0	0.2	68.0	0.2
m4b	4	66.9	0.2	67.9	0.2
m4b	5	67.0	0.2	68.1	0.2
m4b	AVG	67.00	0.09	68.04	0.09

Table F.5: Raw data of velocity reading for 5 cm high concrete samples

Model	Reading #	t_{1M} (ms)	Error (ms)	t_{50K} (ms)	Error (ms)
m1a	1	9.36	0.06	10.1	0.15
m1a	2	9.37	0.09	10.2	0.15
m1a	3	9.39	0.08	10.1	0.2
m1a	4	9.30	0.08	10.1	0.2
m1a	5	9.39	0.09	10.0	0.2
m1a	AVG	9.362	0.04	10.1	0.08
m1b	1	11.3	0.1	11.85	0.2
m1b	2	11.2	0.1	11.8	0.2
m1b	3	11.2	0.1	11.9	0.2
m1b	4	11.1	0.1	11.9	0.2
m1b	5	11.1	0.1	11.8	0.2
m1b	AVG	11.18	0.05	11.86	0.09
m1c	1	9.94	0.09	10.7	0.2
m1c	2	9.98	0.09	10.7	0.2
m1c	3	10.02	0.08	10.7	0.2
m1c	4	9.98	0.09	10.8	0.2
m1c	5	9.94	0.08	10.6	0.2
m1c	AVG	9.97	0.04	10.70	0.09

Table F.6: Raw data of velocity reading for 10 cm high concrete samples

Model	Reading #	t_{1M} (ms)	Error (ms)	t_{50K} (ms)	Error (ms)
m2a	1	19.03	0.06	20.2	0.2
m2a	2	18.99	0.06	20.1	0.2
m2a	3	19.05	0.06	20.2	0.2
m2a	4	19.02	0.06	20.2	0.2
m2a	5	19.00	0.06	20.0	0.2
m2a	AVG	19.03	0.03	20.14	0.09
m2b	1	19.06	0.06	20.4	0.15
m2b	2	19.04	0.06	20.3	0.15
m2b	3	19.07	0.06	20.4	0.2
m2b	4	19.09	0.06	20.3	0.15
m2b	5	19.09	0.06	20.3	0.15
m2b	AVG	19.07	0.03	20.34	0.07
m2c	1	20.45	0.06	21.5	0.15
m2c	2	20.40	0.06	21.4	0.15
m2c	3	20.44	0.06	21.4	0.15
m2c	4	20.40	0.06	21.5	0.15
m2c	5	20.43	0.06	21.4	0.15
m2c	AVG	20.42	0.03	21.44	0.07

Table F.7: Raw data of velocity reading for 20 cm high concrete samples

Model	Reading #	t_{1M} (ms)	Error (ms)	t_{50K} (ms)	Error (ms)
m3a	1	38.1	0.2	39.5	0.2
m3a	2	38.3	0.2	39.4	0.2
m3a	3	38.2	0.2	39.4	0.2
m3a	4	38.1	0.2	39.3	0.2
m3a	5	38.3	0.2	39.3	0.2
m3a	AVG	38.20	0.09	39.38	0.09
m3b	1	38.4	0.2	39.4	0.2
m3b	2	38.3	0.2	39.3	0.2
m3b	3	38.2	0.2	39.1	0.2
m3b	4	38.3	0.2	39.2	0.2
m3b	5	38.3	0.2	39.3	0.2
m3b	AVG	38.30	0.09	39.26	0.09
m3c	1	34.8	0.2	36.2	0.2
m3c	2	34.7	0.2	36.3	0.2
m3c	3	34.7	0.2	36.2	0.2
m3c	4	34.7	0.2	36.0	0.2
m3c	5	34.6	0.2	36.3	0.2
m3c	AVG	34.70	0.09	36.20	0.09

Table F.8: Raw data of velocity reading for 30 cm high concrete samples

Model	Reading #	t_{1M} (ms)	Error (ms)	t_{50K} (ms)	Error (ms)
m4a	1	56.8	0.2	58.4	0.2
m4a	2	56.7	0.2	58.1	0.2
m4a	3	56.9	0.2	58.1	0.2
m4a	4	56.7	0.2	58.2	0.2
m4a	5	56.7	0.2	58.3	0.2
m4a	AVG	56.76	0.09	58.22	0.09
m4b	1	58.0	0.2	59.2	0.2
m4b	2	57.8	0.2	59.5	0.2
m4b	3	57.9	0.2	59.4	0.2
m4b	4	57.9	0.2	59.4	0.2
m4b	5	58.0	0.2	59.3	0.2
m4b	AVG	57.92	0.09	59.36	0.09
m4c	1	58.4	0.2	60.2	0.2
m4c	2	58.4	0.2	60.3	0.2
m4c	3	58.4	0.2	60.2	0.2
m4c	4	58.3	0.2	60.3	0.2
m4c	5	58.4	0.2	60.2	0.2
m4c	AVG	58.38	0.09	60.24	0.09

Bibliography

Abo-Qudais, S. A. (2005). Effect of concrete mixing parameters on propagation of ultrasonic waves. *Construction and Building Materials*, 19:257–263.

Asakawa, E. and Kawanaka, T. (1993). Seismic ray tracing using linear traveltime interpolation. *Geophysical Prospecting*, 41(1):99–111.

Bath, M. and Berkhout, A. J. (1984). Mathematical aspects of seismology. *Klaus Helbig and Sven Trietel, eds., Geophysical, London.*

Bathe, K.-J. (1982). *Finite Element Procedures in Engineering Analysis*. Prentice-Hall.

Bilgutay, N., Popovics, J., Popovics, S., and Karaoguz, M. (2001). Recent developments in concrete nondestructive evaluation. *IEEE.*

Bullen, K. and Bolt, B. A. (1985). *An introduction to the theory of seismology*. Cambridge University Press, 4th edition.

Cagniard, L. (1939). *Réflexion et réfraction des ondes séismiques progressives*. Cauther-Villard, Cambridge, MA.

Cerveny, V. (2001). *Seismic Ray Theory*. Cambridge University Press.

Chaix, J.-F., Garnier, V., and Corneloup, G. (2003). Concrete damage evolution analysis by backscattered ultrasonic waves. *NDTE International*, 36:461–469.

Chang, T.-P., Lin, H.-C., Chang, W.-T., and Hsiao, J.-F. (2006). Engineering properties of lightweight aggregate concrete assessed by stress wave propagation methods. *Cement Concrete Composites*, 28:57–68.

Chang, Y.-F. and Wang, C.-Y. (1997). A 3-d image detection method of a surface opening crack in concrete using ultrasonic transducer arrays. *Journal of Nondestructive Evaluation*, 16(4).

Chapman, C. H. (2004). *Fundamentals of Seismic Wave Propagation*. Cambridge University Press.

Daponte, P., Maceri, F., and olivito, R. S. (1995). Ultrasonic signal-processing techniques for the measurement of damage growth in structural materials. *IEEE Transactions on Instrumentation and Measurement*, 44:1003–1008.

de Hoop, A. (1960). *A modification of Cagniard's method of solving seismic pulse problems*. Applied Scientific Research B.

Feng, M. Q., De Flaviis, F., and Kim, Y. J. (2002). Use of microwaves for damage detection of frp-wrapped oncrete structures. *ASCE Journal of Engineering Mechanics*, 128(2):172–183.

Feng, M. Q., De Flaviis, F., Kim, Y. J., and Diaz, R. (2000). Application of electromagnetic waves in damage detection of concrete structures. *Proceedings of the International Symposium on Smart Structures and Materials, SPIE*.

Fink, M. A. and Cardoso, J.-F. (1984). Diffraction effects in pulse-echo measurement. *IEEE Transactions on Sonics and Ultrasonics*, SU-31(4).

Gaydecki, P. A., Burdein, F. M., Damaj, W., John, D. G., and Payne, P. A. (1992). The propagation and attenuation of medium-frequency ultrasonic waves in concrete: a signal analytical approach. *Measurments Science and Technology*, 3:126–134.

Glushkov, E., Glushkova, N., Ekhlakov, A., and Shapar, E. (2006). An analytically based computer model for surface measurements in ultrasonic crack detection. *Wave Motion*, 43:458–473.

Goueygou, M., Naffa, S. O., Piwakowski, B., and Buyle-Bodin, F. (2001). Non destructive evaluation of degraded concrete cover using high-frequency ultrasound. *IEEE Ultrasonics Symposium*, pages 761–764.

Goueygou, M., Naffa, S. O., Piwakowski, B., Fnine, A., and Buyle-Bodin, F. (2002). Measurments of ultrasonic attenuation and rayleigh wave dispersion for testing concrete with subsurface damage. *IEEE Ultrasonics Symposium*, pages 861–863.

Hallquist, J. O. (2006). *LSDYNA Theory Manual.* Livermore Software Technology Corporation, 2006 edition.

Hatanaka, H., Kawano, Y., Ido, N., Hato, M., and Tagami, M. (2005). Ultrasonic testing with advanced signal processing for concrete structures. *Nondestructive Testing and Evaluation*, 20(2):115–124.

Hatton, L., Worthington, M. H., and Makin, J. (1986). *Seismic Data Processing.* Blackwell Scientific Publications.

Hecht, E. (2002). *Optics.* Addison Wesley, 4th edition.

Hernandez, M., Izquierdo, M., Ibanez, A., Anaya, J., and Ullate, L. (2000). Porosity estimation of concrete by ultrasonic nde. *Ultrasonics*, 38:531–533.

Jenden, J. A. and Svendsen, N. B. (1992). Calculation of pressure fields from arbitrarily shped, apodized, and excited ultrasound transducers. *IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control*, 39(2).

Julian, B. R. and Gubbins, D. (1977). Three dimensional seismic ray tracing. *Journal of Geophysics Research*, 43:95–114.

Kausel, E. (1981). An explicit solution for the green functions for dynamic loads in layered media. Research Report R81-13 699, Massachusetts Institute of Technology, Dept. of Civil Eng., MIT, Cambridge Massachusetts 02139.

Kim, S.-D., Shin, D.-H., Lim, L.-M., Lee, J., and Kim, S.-H. (2005). Designed strength identification of concrete by ultrasonic signal processing based on artificial intelligence techniques. *IEEE Transactions on Sonics and Ultrasonics*, 52(7):1145–1151.

Lamb, H. (1904). On the Propagation of Tremors over the Surface of an Elastic Solid. *Royal Society of London Philosophical Transactions Series A*, 203:1–42.

Lapwood, E. (1949). The disturbance due to a line source in a semi-infinite elastic medium. *Phil. Trans. of the Royal Society of London. Series A*, 242(841):63–100.

Lockwood, J. and Willette, J. (1973). High-speed method for computing the exact solution for the pressure variations in the nearfield of a baffled piston. *Journal of the Acoustical Society of America*, 53(3):735–741.

Marfurt, K. J. (1984). Accuracy of finite-differences and finite-element modeling of the scalar and elastic wave equations. *Geophysics*, 49(5):533–549.

McCann, D. M. and Forde, M. (2001). Review of ndt methods in the assessment of cncrete and masonry structures. *NDTE International*, 34:71–84.

Nakano, H. (1925). On rayleigh waves. *Japanese Journal of Astrophysics and Geophysics*, 2:233–326.

Ohdaira, E. and Masuzawa, N. (2000). Water content and its effect on ultrasound propagation in concrete - the possibility of nde. *Ultrasonics*, 38:546–552.

Pekeris, C. L. (1955). The seismic surface pulse. *Proceedings of the National Academy of Science*, 41:469–480.

Philippidis, T. and Aggelis, D. (2005). Experimental study of wave dispersion and attenuation in concrete. *Ultrasonics*, 43:584–595.

Podvin, P. and Lecomte, I. (1991). Finite difference computation of traveltimes in very contrasted velocity models: a massively parallel approach and its associated tools. *Geophysical Journal International*, 105:271–284.

Popovics, J., Song, W.-J., Ghandehari, M., Subramaniam, K. V., Achenbach, J. D., and Shah, S. P. (2000). Application of surface wave transmission measurements for crack depth determination in concrete. *ACI Materials Journal*, 97(2).

Popovics, J. S. (2003). Nde techniques for concrete and masonry structures. *Prog. Struct. Engng Mater.*, 5:49–59.

Popovics, J. S. and Rose, J. L. (1994). A survey of developments in ultrasonic nde of concrete. *IEEE Transactions on Sonics and Ultrasonics*, 41(1).

Popovics, S. (2005). Effects of uneven moisture distribution on the strength of and wave velocity in concrete. *Ultrasonics*, 43:429–434.

Prassianakis, I. and Prassianakis, N. (2004). Ultrasonic testing of non-metallic materials: concrete and marble. *Theoretical and Applied Fracture Mechanics*, 42:191–198.

Robinson, E. A. and Tritel, S. (1980). *Geophysical Signal Analysis*. Prentice-Hall.

Shickert, M. (2002). Ultrasonic nde of concrete. *IEEE Ultrasonics Symposium*, pages 739–747.

Smirnova, N. S. (1995). An algorithm for determining wave fields in multilayer elastic media. *Journal of Mathematical Science*, 73(3).

Sutan, N. M. and Jaafar, M. S. (2003). Evaluating efficiency of nondestructive detection of flaws in concrete. *Russian Journal of Nondestructive Testing*, 39(2):87–93.

Taner, M. T., Koehler, F., and Sheriff, R. E. (1979). Complex seismic trace analysis. *Geophysics*, 44(6):1041–1063.

Tawhed, W. F. and Gassman, S. L. (2002). Damage assessment of concrete bridge decks using impact-echo method. *ACI Materials Journal*, 99(3).

Um and Thurber (1987). A fast algorithm for two-point seismic ray tracing. *Bulletin of the Seismological Society of America*, 78(3):1190–1198.

Valliappan, H. and Murti, V. (1984). *Finite element constraints in the analysis of wave propagation problems.* UNICIV Rep. No. R-218 School of Civil Engineering, Univsersity of New South Wales, New South Wales, Australia.

Vidale, J. (1988). Finite-difference calculation of travel times. *Bulletin of the Seismological Society of America*, 78(6):2062–2076.

Vidale, J. (1990). Finite-difference calculation of travel times in 3d. *Geophysics*, 55:521–526.

Vinh, P. C. and Malischewsky, P. G. (2006). Explanation for malischewsky's approximate expression for the rayleigh wave velocity. *Ultrasonics*, 45:77–81.

Vinh, P. C. and Ogden, R. (2004). Formulas for the rayleigh wave speed. *Wave Motion*, 39:191–197.

Warnemuende, K. and Wu, H.-C. (2004). Actively modulated acoustic nondestructive evaluation of concrete. *Cement and Concrete Reearch*, 34:563–570.

Washer, G., Fuchs, P., Rezai, A., and Ghasemi, H. (2005). Ultrasonic measurement of the elastic properties of ultra high performance concrete (uhpc). *Proc. of SPIE*, 5767:416–422.

Yang, Y., Cascante, G., and Polak, M. A. (2005). Detection of depth of surface-breaking cracks in concrete pipes. In *1st Canadian Conference on Effective Design of Structures, McMaster University.*

Zerwer, A., Cascante, G., and Hutchinson, J. (2002). Parameter estimation in finite element simulations. *Journal of Geotechnical and Geoenvironmental Engineering.*, 128(3):250–261.