Appendix B

Source Code

B.1 Quasi-Static Ultrasound Elastography

Listing B.1: Sample code used to simulate b-mode ultrasound images through the convolution of a simulated scattering center distribution with the point spread function of an ultrasound pulse.

```
% generate normally distributed noise across the domain
rng(domainSeed);
backgroundMap = 1 + noiseMagnitude * randn([Ny, Nx]);

% transform the background map into the domain -1 -> 1
backgroundMap = (backgroundMap - min(min(backgroundMap))) / (
    max(max(backgroundMap)) - min(min(backgroundMap)));
backgroundMap = 2 * backgroundMap - 1;

% if a finite-element model of tissue compression is being used
    ,
% compress the background map
if model ~= 0
  % extract the resultant degrees of freedom from the model
  [x, y, u, v] = extractUV([transducerWidth, depth], size(
    backgroundMap), model);

  % generate the domain over which interpolation will occur
  [xx, yy] = meshgrid(x, y);

  % interpolate the data to deform it
  backgroundMap = interp2(xx, yy, backgroundMap, xx - u, yy - v
    , 'spline', mean(mean(backgroundMap)));
end
```

```
21
% use a cosine function to create the point-spread function
    shape in the axial direction
23 xpsf = linspace(-windowWidth / 2, windowWidth / 2, 1 *
    pointsPerWaveLength);
ypsf = linspace(0, 4 * waveLength, 2 * pointsPerWaveLength);
25 [xmpsf, ympsf] = meshgrid(xpsf, ypsf);
psf = cos(2 * pi * frequency * ympsf / waveSpeed);
27
% apply a lateral gaussian "filter" to the signal
29 mu = 0;
sigma = windowWidth / 4;
31 gauss = (1 / (sigma * sqrt(2 * pi))) * exp(-(xmpsf - mu) .^ 2 /
     (2 * sigma ^ 2));
psf = psf .* gauss;
33
% apply an axial gaussian "filter" to the signal
35 mu = 2 * waveLength;
sigma = waveLength * 2;
37 gauss = (1 / (sigma * sqrt(2 * pi))) * exp(-(ympsf - mu) .^ 2 /
     (2 * sigma ^ 2));
gauss = (gauss - min(min(gauss))) / (max(max(gauss)) - min(min(
    gauss)));
39 psf = psf .* gauss;

41 % normalize it to -1 -> 1
psf = (psf - min(min(psf)))/(max(max(psf)) - min(min(psf)));
43 psf = 2 * psf - 1;

45 % convolve the scattering map with the point spread function
bmode = conv2(backgroundMap, psf);
47
% crop the image to the appropriate size
49 sz = size(bmode);
bmode = bmode(int32(floor((sz(1) - pointDepth) / 2)) : int32(
    floor((sz(1) - pointDepth) / 2) + pointDepth - 1), int32(
    floor((sz(2) - numElements) / 2)) : int32(floor((sz(2) -
    numElements) / 2) + numElements - 1));
51
% apply classical ultrasound post-processing
53 % the signal is currently oscillating a high frequency
%  - extract the envelope of the signal
55 bmode = envelopeDetection(bmode')';
% apply log compression to allow to be readily viewed
57 bmode = logCompression(bmode, 3, true);

59 % normalize it to 0 -> 1
bmode = (bmode - min(min(bmode))) ./ (max(max(bmode)) - min(min
    (bmode)));
```

Listing B.2: Sample code used to generate displacement maps under quasi-static deformation

```
function model = compression(parameters)
  % import COMSOL features
  import com.comsol.model.*
  import com.comsol.model.util.*

  % create the model
  model = ModelUtil.create('Model');
  model.modelPath('compression');
  model.modelNode.create('mod1');
  model.geom.create('geom1', 2);
  model.mesh.create('mesh1', 'geom1');
  model.physics.create('solid', 'SolidMechanics', 'geom1');
  model.study.create('std1');
  model.study('std1').feature.create('stat', 'Stationary');
  model.study('std1').feature('stat').activate('solid', true);

  % set parameters in the model
  model.param.set('modelWidth', sprintf('%f[m]', parameters.
   domainWidth));
  model.param.set('modelDepth', sprintf('%f[m]', parameters.
   domainDepth));
  model.param.set('appliedStrain', sprintf('%f', parameters.
   appliedStrain));
  model.param.set('compression', 'modelDepth*appliedStrain');
  model.param.set('basalStiffness', sprintf('%f[Pa]',
   parameters.basalStiffness));
  model.param.set('stiffnessRatio', sprintf('%f', parameters.
   lesionStiffnessRatio));
  model.param.set('lesionStiffness', 'basalStiffness*
   stiffnessRatio');
  model.param.set('lesionExtraStiffness', 'lesionStiffness-
   basalStiffness');
  model.param.set('density', sprintf('%f[kg/m^3]', parameters.
   density));
  model.param.set('poissonsRatio', sprintf('%f', parameters.
   poissonsRatio));

  % deal with the human model case
  if strcmpi(cell2mat(parameters.caseCategory), 'human') == 1
    model.param.set('fatStiffness', '80[kPa]');
    model.param.set('boneStiffness', '18.6[GPa]');
    model.param.set('fatExtraStiffness', 'fatStiffness -
   basalStiffness');
    model.param.set('boneExtraStiffness', 'boneStiffness -
   basalStiffness');

    % import our stiffness image
    model.func.create('im1', 'Image');
```

```
38      model.func('im1').set('filename', sprintf('compression/
        stiffnessMap_%03d.png', caseIndex));
        model.func('im1').importData;
40      model.func('im1').set('xmin', sprintf('%f', (-parameters.
        domainWidth / 2)));
        model.func('im1').set('xmax', sprintf('%f', (parameters.
        domainWidth / 2)));
42      model.func('im1').set('ymax', sprintf('%f', parameters.
        domainDepth));
        model.func('im1').set('inplace', 'off');
44      model.func('im1').set('scaling', 'manual');
        model.func('im1').set('manualexpr', 'r');
46      model.func('im1').set('funcname', 'lesionMap');
        model.func.duplicate('im2', 'im1');
48      model.func('im2').set('funcname', 'boneMap');
        model.func('im2').set('manualexpr', 'b');
50      model.func.duplicate('im3', 'im2');
        model.func('im3').set('funcname', 'fatMap');
52      model.func('im3').set('manualexpr', 'g');
    else
54      % import our stiffness image
        model.func.create('im1', 'Image');
56      model.func('im1').set('funcname', 'stiffnessMap');
        model.func('im1').set('filename', sprintf('compression/
        stiffnessMap_%03d.png', caseIndex));
58      model.func('im1').importData;
        model.func('im1').set('xmin', sprintf('%f', (-parameters.
        domainWidth / 2)));
60      model.func('im1').set('xmax', sprintf('%f', (parameters.
        domainWidth / 2)));
        model.func('im1').set('ymax', sprintf('%f', parameters.
        domainDepth));
62  end

64  % define the model geometry
    model.geom('geom1').feature.create('r1', 'Rectangle');
66  model.geom('geom1').feature('r1').setIndex('size', '
     domainWidth', 0);
    model.geom('geom1').feature('r1').setIndex('size', '
     domainDepth', 1);
68  model.geom('geom1').feature('r1').setIndex('size', '
     modelWidth', 0);
    model.geom('geom1').feature('r1').setIndex('size', '
     modelDepth', 1);
70  model.geom('geom1').feature('r1').setIndex('pos', '-
     modelWidth/2', 0);
    model.geom('geom1').run;
72
    % set the material properties
74  model.physics('solid').feature('lemm1').set('
     NearlyIncompressible', 1, '1');
```

```
model.physics('solid').feature('lemm1').set('E_mat', 1, '
 userdef');

% deal with the human model case
if strcmpi(cell2mat(parameters.caseCategory), 'human') == 1
  model.physics('solid').feature('lemm1').set('E', 1, '
 basalStiffness + (fatMap(x, modelDepth - y) *
 fatExtraStiffness) + (lesionMap(x, modelDepth - y) *
 lesionExtraStiffness) + (boneMap(x, modelDepth - y) *
 boneExtraStiffness)');
else
  model.physics('solid').feature('lemm1').set('E', 1, '
 basalStiffness+(stiffnessMap(x,modelDepth-y)*
 lesionExtraStiffness)');
end

model.physics('solid').feature('lemm1').set('nu_mat', 1, '
 userdef');
model.physics('solid').feature('lemm1').set('nu', 1, '
 poissonsRatio');
model.physics('solid').feature('lemm1').set('rho_mat', 1, '
 userdef');
model.physics('solid').feature('lemm1').set('rho', 1, '
 density');

% set the boundary conditions
model.physics('solid').feature.create('fix1', 'Fixed', 1);
model.physics('solid').feature('fix1').selection.set([2]);
model.physics('solid').feature.create('disp1', 'Displacement1
 ', 1);
model.physics('solid').feature('disp1').selection.set([3]);
model.physics('solid').feature('disp1').set('Direction', 2, '
 1');
model.physics('solid').feature('disp1').set('U0', 2, '-
 compression');

% create the mesh
model.mesh('mesh1').feature.create('ftri1', 'FreeTri');
model.mesh('mesh1').feature('size').set('hauto', '1');
model.mesh('mesh1').run;

% setup the study and run it
model.sol.create('sol1');
model.sol('sol1').study('std1');
model.sol('sol1').feature.create('st1', 'StudyStep');
model.sol('sol1').feature('st1').set('study', 'std1');
model.sol('sol1').feature('st1').set('studystep', 'stat');
model.sol('sol1').feature.create('v1', 'Variables');
model.sol('sol1').feature('v1').set('control', 'stat');
model.sol('sol1').feature.create('s1', 'Stationary');
model.sol('sol1').feature('s1').feature.create('fc1', '
```

```
    FullyCoupled');
  model.sol('sol1').feature('s1').feature.remove('fcDef');
  model.sol('sol1').attach('std1');

  model.sol('sol1').runAll;
end
```

Listing B.3: Sample code used to estimate the lateral and axial strain along a given scanline of a pre- and post- compression b-mode image using quasi-static ultrasound elastography.

```
% calculate the number of windows along this scanline
M = maxX / deltaAx;

% these variables will return the alpha and tau coefficients
% along this scanline
alpha = [];
tau = [];

% loop through every window along this scanline
for m = 1:M
  % calculate the locations of the two windows to start the
   search from
  pr1 = m .* deltaAx;
  pr2 = sum(deltaAx ./ alpha);

  % extract the window in the uncompressed image
  x1 = linspace(pr1, pr1 + L, numPoints);
  r1 = interp1(1 : length(I1), I1, x1, 'linear', 'extrap');

  % initialize search variables
  alphas = maxAlpha:-0.001:1;
  taus = zeros(size(alphas));
  correlations = zeros(size(alphas));

  % loop through the possible values of alpha
  for i = 1 : length(alphas)
    % keep track of which scanline gives us the best
   correlation
    columnCorrelations = [];

    % loop through the adjacent scanlines
    for c = (column - columnRadius) : (column + columnRadius)
      % make sure we have a valid scanline
      if (c < 1) || (c > columns)
        continue;
      end

      % extract the window in the compressed image
      x2 = linspace(round(pr2), round(pr2) + round(L),
   numPoints);
```

```
      r2 = interp1(1:length(I2(:, c)), I2(:, c), x2, 'linear',
    'extrap');

      % get the correlation for the two windows
      columnCorrelations = [columnCorrelations, Correlate(r1,
    r2)];
    end

    % pick the scanline that had the best correlation for this
    alpha
    c = -columnRadius : column + columnRadius;
    [correlations(i), mindex] = min(columnCorrelations);
    taus(i) = c(mindex);
  end

  % employ a-priori smoothing to the data
  % in case of errant outliers
  [~, mindex] = min(correlations);
  if abs(alphas(mindex) - alpha(length(alpha))) > 0.02
    alpha = [alpha; alpha(length(alpha))];
    tau = [tau; tau(length(tau))];
  else
    alpha = [alpha; alphas(mindex)];
    tau = [tau; taus(mindex)];
  end
end
```

B.2 Acoustic Radiation Force Impulse Imaging

Listing B.4: Sample code used in conjunction with the K-Wave Toolbox to simulate acoustic radiation force body loads generated by an ultrasonic transducer.

```
% calculate the best grid size for computational efficiency
% taking into account Nyquist so that appropriate sampling is
    used
dx = medium.sound_speed / (2 * probingFrequency);
dy = dx;
Nx = bestFactor(domainDepth / dx + (2 * PML_X_SIZE)) - (2 *
    PML_X_SIZE);
Ny = bestFactor(domainWidth / dy + (2 * PML_Y_SIZE)) - (2 *
    PML_X_SIZE);
dx = domainDepth / Nx;
dy = domainWidth / Ny;

% make the grid!
kGrid = makeGrid(Nx, dx, Ny, dy);

% create the time array
```

```
kGrid.t_array = makeTime(kGrid, medium.sound_speed, [],
    pulseCycles / probingFrequency);

% beam-form the data
% create indices for our elements
numActiveElements = Ny;
elementIndices = -(numActiveElements - 1) / 2 : (
    numActiveElements - 1) / 2;

% calculate time delays for a steered and focussed beam
elementWidth = dy;
delayTimes = focalDepth / medium.sound_speed * (1 - sqrt(1 + (
    elementIndices * elementWidth ./ focalDepth) .^ 2 )); % [s]

% convert the delays to be in units of time points
delayTimes = delayTimes - min(delayTimes);
delayTimes = delayTimes ./ kGrid.dt;

% use the k-wave toolbox's function "toneBurst" to create the
    signal
inputSignal = toneBurst(1 / kGrid.dt, probingFrequency,
    pulseCycles, 'SignalOffset', delayTimes);

% scale the signal by the source pressure
source.p = sourcePressure .* inputSignal;

% truncate the input signal to the appropriate length
source.p = source.p(:, 1:length(kGrid.t_array));

% make only the nodes along the top boundary apply pressure to
    the domain
source.p_mask = zeros([Nx, Ny]);
source.p_mask(1, 1 : (numActiveElements)) = 1;

% tell the k-wave toolbox to record the pressure and intensity
    for
% the entire domain, continuously
sensor.mask = ones(Nx, Ny);
sensor.record = {'I', 'p'};

% set up simulation settings
inputArgs = {'PlotSim', false, 'PMLInside', false, 'PlotPML',
    false, 'PMLSize', [PML_X_SIZE, PML_Y_SIZE], 'DataCast', '
    single', 'DataRecast', true, 'DisplayMask', 'off'};

% run the simulation
sensorData = kspaceFirstOrder2D(kGrid, medium, source, sensor,
    inputArgs{:});

% reshape the output data to make sense
Ix = reshape(sensorData.Iy, [Nx, Ny, kGrid.Nt]);
```

```
Iy = reshape(sensorData.Ix, [Nx, Ny, kGrid.Nt]);
P = reshape(sensorData.p, [Nx, Ny, kGrid.Nt]);

% calculate body forces
alpha = (attenuationCoefficient * 100 * probingFrequency / 1e6)
    / (20 / log(10));
Fx = 2 * alpha .* Ix ./ soundSpeed;
Fy = -2 * alpha .* Iy ./ soundSpeed;
```

Listing B.5: Sample code used to simulate time-dependent displacement of viscoelastic soft tissue as a response to acoustic radiation force impulse loads applied to it.

```
function model = relaxation(parameters)
  % import COMSOL features
  import com.comsol.model.*
  import com.comsol.model.util.*

  % create the model
  model = ModelUtil.create('Model');
  model.modelPath('ARFI');
  model.name('relaxation.mph');

  % set parameters in the model
  model.param.set('domainWidth', sprintf('%f[m]', parameters.
   domainWidth));
  model.param.set('domainDepth', sprintf('%f[m]', parameters.
   domainDepth));
  model.param.set('focalDepth', sprintf('%f[mm]', parameters.
   focalDepth));
  model.param.set('timeStep', sprintf('%f[us]', parameters.
   timeStep));
  model.param.set('loadTime', sprintf('%f[us]', parameters.
   loadTime));
  model.param.set('listenTime', sprintf('%f[ms]', parameters.
   listenTime));
  model.param.set('focalY', 'domainDepth-focalDepth');
  model.param.set('dFocalY', 'focalY/1[m]');
  model.param.set('stiffnessRatio', sprintf('%f', parameters.
   stiffnessRatio));
  model.param.set('cutoffAmplitude', sprintf('%f', parameters.
   cutoffAmplitude));

  % load interpolation files for both geometry
  % and initial acoustic radiation force
  model.modelNode.create('mod1');
  model.func.create('int1', 'Interpolation');
  model.func.create('int2', 'Interpolation');
  model.func.create('step1', 'Step');
  model.func.create('im1', 'Image');
  model.func('int1').set('funcs', {'Fx' '1'});
```

```
model.func('int1').set('source', 'file');
model.func('int1').set('filename', 'ARFI/Fx.txt');
model.func('int1').set('struct', 'grid');
model.func('int1').set('defvars', 'on');
model.func('int1').set('extrap', 'value');
model.func('int1').set('argunit', 'm');
model.func('int1').set('fununit', 'N/(m^3)');
model.func('int2').set('funcs', {'Fy' '1'});
model.func('int2').set('source', 'file');
model.func('int2').set('filename', 'ARFI/Fy.txt');
model.func('int2').set('struct', 'grid');
model.func('int2').set('defvars', 'on');
model.func('int2').set('extrap', 'value');
model.func('int2').set('argunit', 'm');
model.func('int2').set('fununit', 'N/(m^3)');
model.func('step1').set('location', 'loadTime');
model.func('step1').set('from', '1');
model.func('step1').set('to', '0');
model.func('step1').set('smoothactive', false);
model.func('im1').set('funcname', 'stiffnessMap');
model.func('im1').set('filename', 'ARFI/stiffnessMap.png');
model.func('im1').set('xmin', '-domainWidth/2');
model.func('im1').set('xmax', 'domainWidth/2');
model.func('im1').set('ymax', 'domainDepth');
model.func('im1').set('extrap', 'value');

% create the geometry
model.geom.create('geom1', 2);
model.geom('geom1').feature.create('r1', 'Rectangle');
model.geom('geom1').feature.create('r2', 'Rectangle');
model.geom('geom1').feature('r1').set('pos', {'0' '0'});
model.geom('geom1').feature('r1').set('size', {'domainWidth/2
 ' 'domainDepth'});
model.geom('geom1').feature('r2').set('pos', {'domainWidth/2'
  '0'});
model.geom('geom1').feature('r2').set('size', {'domainWidth/8
 ' 'domainDepth'});
model.geom('geom1').run;

% set up material properties
model.material.create('mat1');
model.material('mat1').propertyGroup.create('KG', 'Bulk
 modulus and shear modulus');
model.material('mat1').propertyGroup('def').set('density', '
 1060');
model.material('mat1').propertyGroup('KG').set('K', '');
model.material('mat1').propertyGroup('KG').set('G', '');
model.material('mat1').propertyGroup('KG').set('K', '
 stiffnessRatio*515.656[kPa]');
model.material('mat1').propertyGroup('KG').set('G', '
 stiffnessRatio*1032[Pa]');
```

```
75
   % set up the physics
77 % including boundary and ``initial'' conditions
   % fix the bottom of the domain
79 model.physics.create('solid', 'SolidMechanics', 'geom1');
   model.physics('solid').feature.create('fix1', 'Fixed', 1);
81 model.physics('solid').feature('fix1').selection.set([2 5]);
   model.physics('solid').feature('fix1').name('bottom hold');
83
   % prevent motion in the vertical direction at the top
    boundary
85 model.physics('solid').feature.create('disp1', 'Displacement1
    ', 1);
   model.physics('solid').feature('disp1').selection.set([3 6]);
87 model.physics('solid').feature('disp1').set('Direction', {'0'
    ; '1'; '0'});
   model.physics('solid').feature('disp1').name('top hold');
89
   % the ARFI is a body load
91 model.physics('solid').feature.create('bl1', 'BodyLoad', 2);
   model.physics('solid').feature('bl1').selection.set([1]);
93 model.physics('solid').feature('bl1').set('FperVol', {'Fx(x,y
    )*step1(t[1/s])'; 'Fy(x,y)*step1(t[1/s])'; '0'});
   model.physics('solid').feature('bl1').set('Ftot', {''; 'Fy(x,
    y)*step1(t[1/s])*10^2'; '0'});
95 model.physics('solid').feature('bl1').name('ARFI load');

97 % make the model symmetric
   model.physics('solid').feature.create('sym1', 'SymmetrySolid'
    , 1);
99 model.physics('solid').feature('sym1').selection.set([1]);

101 % use a viscoelastic tissue model
   model.physics('solid').feature.create('vmm1', '
    ViscoelasticModel', 2);
103 model.physics('solid').feature('vmm1').selection.all;
   model.physics('solid').feature('vmm1').set('K_mat', 'userdef'
    );
105 model.physics('solid').feature('vmm1').set('K', '(515.656 [
    kPa]) * (1 + (stiffnessMap(x, y) * (stiffnessRatio - 1)))')
    ;
   model.physics('solid').feature('vmm1').set('G_mat', 'userdef'
    );
107 model.physics('solid').feature('vmm1').set('G', '(1032 [Pa])
    * (1 + (stiffnessMap(x, y) * (stiffnessRatio - 1)))');
   model.physics('solid').feature('vmm1').set('Branch', {'1'; '2
    '; '2'});
109 model.physics('solid').feature('vmm1').set('Gi', {'(791 [Pa])
     * (1 + (stiffnessMap(x, y) * (stiffnessRatio - 1)))'; '
    (66.5 [Pa]) * (1 + (stiffnessMap(x, y) * (stiffnessRatio -
    1)))'; '(628 [mPa]) * (1 + (stiffnessMap(x, y) * (
```

```
    stiffnessRatio - 1)))'});
model.physics('solid').feature('vmm1').set('tau', {'2 [s]'; '
    40 [s]'; '80 [s]'});
model.physics('solid').feature('vmm1').set('
    NearlyIncompressible', '1');

% create the mesh
model.mesh.create('mesh1', 'geom1');
model.mesh('mesh1').feature.create('ftri1', 'FreeTri');
model.mesh('mesh1').feature.create('map1', 'Map');
model.mesh('mesh1').feature('size').set('hauto', 3);
model.mesh('mesh1').feature('map1').active(false);
model.mesh('mesh1').feature('map1').set('adjustedgdistr',
    true);
model.mesh('mesh1').run;

% set up a probe at the focal point
model.result.table.create('tbl1', 'Table');
model.result.table.create('tbl2', 'Table');
model.probe.create('pdom1', 'DomainPoint');
model.probe('pdom1').model('mod1');
model.coordSystem.create('pml1', 'geom1', 'PML');
model.coordSystem('pml1').selection.set([2]);
model.probe('pdom1').set('coords2', {'0' 'domainDepth-
    focalDepth'});
model.probe('pdom1').feature('ppb1').set('probename', '
    focalPointDisplacement');
model.probe('pdom1').feature('ppb1').set('table', 'tbl1');
model.probe('pdom1').feature('ppb1').set('window', 'window1')
    ;
model.result.table('tbl1').name('Probe Table 1');
model.result.table('tbl2').comments('Global Evaluation 1 (t)'
    );

% create a transient study
model.study.create('std1');
model.study('std1').feature.create('time', 'Transient');

% set up the solution parameters
model.sol.create('sol1');
model.sol('sol1').study('std1');
model.sol('sol1').attach('std1');
model.sol('sol1').feature.create('st1', 'StudyStep');
model.sol('sol1').feature.create('v1', 'Variables');
model.sol('sol1').feature.create('t1', 'Time');
model.sol('sol1').feature('t1').feature.create('fc1', '
    FullyCoupled');
model.sol('sol1').feature('t1').feature.create('st1', '
    StopCondition');
model.sol('sol1').feature('t1').feature.remove('fcDef');
model.sol('sol1').attach('std1');
```

```
model.sol('sol1').feature('st1').name('Compile Equations:
 Time Dependent');
model.sol('sol1').feature('st1').set('studystep', 'time');
model.sol('sol1').feature('v1').set('control', 'time');
model.sol('sol1').feature('v1').feature('mod1_u').set('
 scalemethod', 'manual');
model.sol('sol1').feature('v1').feature('mod1_u').set('
 scaleval', '1e-2*0.0640312423743285');
model.sol('sol1').feature('t1').set('tlist', 'range(0,
 timeStep,loadTime+listenTime)');
model.sol('sol1').feature('t1').set('fieldselection', 'mod1_u
 ');
model.sol('sol1').feature('t1').set('atolmethod', {'mod1_u' '
 global' 'mod1_solid_qXX3' 'global' 'mod1_solid_qXY2' '
 global' 'mod1_solid_qXY3' 'global' 'mod1_solid_qXX1' '
 global'  ...
'mod1_solid_qXX2' 'global' 'mod1_solid_qYY3' 'global' '
 mod1_solid_qYY2' 'global' 'mod1_solid_qYY1' 'global' '
 mod1_solid_qXY1' 'global'  ...
'mod1_solid_pw' 'global'});
model.sol('sol1').feature('t1').set('atol', {'mod1_u' '1e-3'
 'mod1_solid_qXX3' '1e-3' 'mod1_solid_qXY2' '1e-3' '
 mod1_solid_qXY3' '1e-3' 'mod1_solid_qXX1' '1e-3'  ...
'mod1_solid_qXX2' '1e-3' 'mod1_solid_qYY3' '1e-3' '
 mod1_solid_qYY2' '1e-3' 'mod1_solid_qYY1' '1e-3' '
 mod1_solid_qXY1' '1e-3'  ...
'mod1_solid_pw' '1e-3'});
model.sol('sol1').feature('t1').set('atoludot', {'mod1_u' '1e
 -3' 'mod1_solid_qXX3' '1e-3' 'mod1_solid_qXY2' '1e-3' '
 mod1_solid_qXY3' '1e-3' 'mod1_solid_qXX1' '1e-3'  ...
'mod1_solid_qXX2' '1e-3' 'mod1_solid_qYY3' '1e-3' '
 mod1_solid_qYY2' '1e-3' 'mod1_solid_qYY1' '1e-3' '
 mod1_solid_qXY1' '1e-3'  ...
'mod1_solid_pw' '1e-3'});
model.sol('sol1').feature('t1').set('atoludotactive', {'
 mod1_u' 'off' 'mod1_solid_qXX3' 'off' 'mod1_solid_qXY2' '
 off' 'mod1_solid_qXY3' 'off' 'mod1_solid_qXX1' 'off'  ...
'mod1_solid_qXX2' 'off' 'mod1_solid_qYY3' 'off' '
 mod1_solid_qYY2' 'off' 'mod1_solid_qYY1' 'off' '
 mod1_solid_qXY1' 'off'  ...
'mod1_solid_pw' 'off'});
model.sol('sol1').feature('t1').set('timemethod', 'genalpha')
 ;
model.sol('sol1').feature('t1').set('tstepsgenalpha', 'manual
 ');
model.sol('sol1').feature('t1').set('timestepgenalpha', '
 timeStep');
model.sol('sol1').feature('t1').set('plot', 'on');
model.sol('sol1').feature('t1').set('plotgroup', 'pg2');
model.sol('sol1').feature('t1').feature('fc1').set('plot', '
 on');
```

```
model.sol('sol1').feature('t1').feature('fc1').set('plotgroup
 ', 'pg2');

% add a stop condition to stop the simulation when the tissue
  has relaxed
model.sol('sol1').feature('t1').feature('st1').set('stopcond'
 , 'if(t > 0.003 && mod1.focalPointDisplacement <
 cutoffAmplitude, -1, 1)');

% create data sets to evaluate later:
% * The displacement at the focal point over time
% * The displacement along an axial line going through the
 focal point
% * The displacement along a lateral line going through the
 focal point
model.result.dataset.create('cpt1', 'CutPoint2D');
model.result.dataset.create('cln1', 'CutLine2D');
model.result.dataset.create('cln2', 'CutLine2D');
model.result.dataset.create('dset2', 'Solution');
model.result.dataset('dset2').set('probetag', 'pdom1');
model.result.dataset.create('cpt2', 'CutPoint2D');
model.result.dataset('cpt2').set('probetag', 'pdom1');
model.result.dataset('cpt2').set('data', 'dset2');
model.result.numerical.create('pev1', 'EvalPoint');
model.result.numerical('pev1').set('probetag', 'ppb1');
model.result.numerical.create('gev1', 'EvalGlobal');
model.result.numerical('gev1').set('probetag', 'none');
model.result.create('pg1', 'PlotGroup1D');
model.result('pg1').set('probetag', 'none');
model.result('pg1').feature.create('ptgr1', 'PointGraph');
model.result.create('pg2', 'PlotGroup2D');
model.result('pg2').feature.create('surf1', 'Surface');
model.result.create('pg3', 'PlotGroup1D');
model.result('pg3').set('probetag', 'none');
model.result('pg3').feature.create('lngr1', 'LineGraph');
model.result.create('pg4', 'PlotGroup1D');
model.result('pg4').set('probetag', 'none');
model.result('pg4').feature.create('lngr1', 'LineGraph');
model.result.create('pg5', 'PlotGroup1D');
model.result('pg5').set('probetag', 'window1');
model.result('pg5').feature.create('tblp1', 'Table');
model.result('pg5').feature('tblp1').set('probetag', 'ppb1');
model.result.dataset('cpt1').name('Focal Point');
model.result.dataset('cpt1').set('pointx', '0');
model.result.dataset('cpt1').set('pointy', 'dFocalY');
model.result.dataset('cln1').name('Lateral Cut');
model.result.dataset('cln1').set('genpoints', {'-domainWidth
 /2' 'domainDepth - focalDepth'; 'domainWidth/2' '
 domainDepth - focalDepth'});
model.result.dataset('cln2').name('Axial Cut');
model.result.dataset('cln2').set('genpoints', {'0' '
```

```
    domainDepth'; '0' '0'});
model.result.dataset('dset2').name('Probe Solution 2');
model.result.dataset('cpt2').set('pointy', 'focalY');
model.result.numerical('gev1').set('table', 'tbl2');
model.result.numerical('gev1').set('expr', 't');
model.result.numerical('gev1').set('unit', 's');
model.result.numerical('gev1').set('descr', 'Time');
model.result.numerical('gev1').set('dataseries', 'maximum');
model.result.numerical('pev1').setResult;
model.result.numerical('gev1').setResult;
model.result('pg1').name('Focal Point Relaxation');
model.result('pg1').set('data', 'cpt1');
model.result('pg1').set('xlabel', 'Time (ms)');
model.result('pg1').set('ylabel', 'Displacement (m)');
model.result('pg1').set('xlabelactive', false);
model.result('pg1').set('ylabelactive', false);
model.result('pg1').feature('ptgr1').set('descractive', true)
    ;
model.result('pg1').feature('ptgr1').set('descr', '
    Displacement');
model.result('pg1').feature('ptgr1').set('titletype', 'manual
    ');
model.result('pg1').feature('ptgr1').set('title', 'Focal
    Point Relaxation');
model.result('pg1').feature('ptgr1').set('xdata', 'expr');
model.result('pg1').feature('ptgr1').set('xdataexpr', 't');
model.result('pg1').feature('ptgr1').set('xdataunit', 'ms');
model.result('pg1').feature('ptgr1').set('xdatadescr', 'Time'
    );
model.result('pg2').name('Surface Displacement');
model.result('pg3').name('Displacement of Axial Focal Cut');
model.result('pg3').set('data', 'cln2');
model.result('pg3').set('xlabel', 'Depth (m) (m)');
model.result('pg3').set('ylabel', 'Total displacement (m)');
model.result('pg3').set('xlabelactive', false);
model.result('pg3').set('ylabelactive', false);
model.result('pg3').feature('lngr1').set('xdata', 'expr');
model.result('pg3').feature('lngr1').set('xdataexpr', '
    domainDepth - y');
model.result('pg3').feature('lngr1').set('xdatadescractive',
    true);
model.result('pg3').feature('lngr1').set('xdatadescr', 'Depth
     (m)');
model.result('pg3').feature('lngr1').set('legend', true);
model.result('pg4').name('Displacement of Lateral Focal Cut')
    ;
model.result('pg4').set('data', 'cln1');
model.result('pg4').set('xlabel', 'x-coordinate (m)');
model.result('pg4').set('ylabel', 'abs(v) (m)');
model.result('pg4').set('legendpos', 'lowerright');
model.result('pg4').set('xlabelactive', false);
```

```
  model.result('pg4').set('ylabelactive', false);
  model.result('pg4').feature('lngr1').set('expr', 'abs(v)');
  model.result('pg4').feature('lngr1').set('descr', 'abs(v)');
  model.result('pg4').feature('lngr1').set('xdata', 'expr');
  model.result('pg4').feature('lngr1').set('xdataexpr', 'x');
  model.result('pg4').feature('lngr1').set('xdatadescr', 'x-
   coordinate');
  model.result('pg4').feature('lngr1').set('legend', true);
  model.result('pg5').name('Probe 1D Plot Group 5');
  model.result('pg5').set('xlabel', 't');
  model.result('pg5').set('ylabel', 'Total displacement, Point
   Probe Expression 1');
  model.result('pg5').set('windowtitle', 'Probe Plot 1');
  model.result('pg5').set('xlabelactive', false);
  model.result('pg5').set('ylabelactive', false);
  model.result('pg5').feature('tblp1').name('Probe Table Graph
   1');

  % initiate the domain probe we defined earlier
  model.probe('pdom1').genResult([]);

  % set the timestepping for the solution
  model.study('std1').feature('time').set('tlist', 'range(0,
   timeStep,loadTime+listenTime)');
  model.study('std1').feature('time').set('plot', 'on');
  model.study('std1').feature('time').set('plotgroup', 'pg2');

  % run the model
  model.sol('sol1').runAll;
end
```

B.3 Shear Wave Speed Quantification

Listing B.6: Sample code used to calculate the speed of a shear wave generated by an ARFI force along a lateral line traversing the focal point of the applied force.

```
% extract the displacement along a lateral line traversing the
   focal point
% throughout the complete simulation time
[x, t, lateralCut] = getLateralFocalCutDisplacement(model);

% use the mean value of the displacement through time and
   location
% as the isoline value to track
targetValue = mean(mean(lateralCut));

% loop along the x-coordinates of the dataset
points = NaN(length(x), 2);
% ignore the first 4 data points which lie within the width of
   the
```

```
% initial radiation force
for xi = 5 : length(x)
  % find the point in time where the displacement at the given
   x-coordinate
  % crosses the target value previously established
  for ti = 2 : length(t)
    if (lateralCut(ti, xi) < targetValue) && (lateralCut(ti -
   1, xi) >= targetValue)
      % if the cross-over point was found, store it and
   continue with
      % the next x-coordinate
      points(xi, :) = [t(ti), x(xi)];
      ti = length(t);
      break;
    end
  end
end

% remove NaNs from the dataset
points = points(~any(isnan(points), 2), :);

% differentiate to get shear velocity
% (use a center-weighted moving window average filter first
% otherwise the data will become unusable)
Ct = diff(smooth(points(:, 2))) ./ diff(smooth(points(:, 1)));
x = linspace(min(points(:, 2)), max(points(:, 2)), length(Ct));
```

Appendix C

Experimental Protocols

Each of the protocols detailed here were carried out on a Siemens ACUSON S2000™ portable ultrasound machine with a Siemens 9L4 transducer on a CIRS Elasticity QA Phantom model 049 as shown in Fig. C.1.

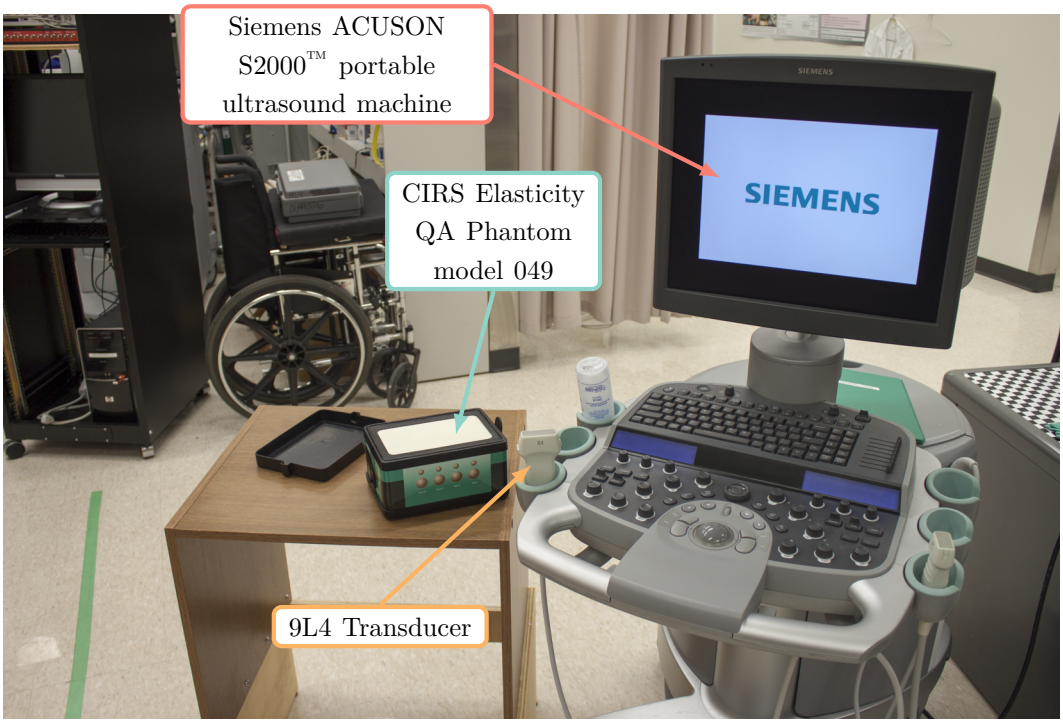

Fig. C.1: Experimental setup showing the ultrasound machine, probe, and phantom model.