10.1109/ULTSYM.2014.0037

Modelling Elastic Wave Propagation Using the k-Wave MATLAB Toolbox

Bradley E. Treeby*†, Jiri Jaros‡, Daniel Rohrbach§, and B. T. Cox*
*Department of Medical Physics and Biomedical Engineering, University College London, London, UK
‡Faculty of Information Technology, Brno University of Technology, Brno, CZ
§Lizzi Center for Biomedical Engineering, Riverside Research, New York, NY, USA
†Email: b.treeby@ucl.ac.uk

***Abstract*—A new model for simulating elastic wave propagation using the open-source k-Wave MATLAB Toolbox is described. The model is based on two coupled first-order equations describing the stress and particle velocity within an isotropic medium. For absorbing media, the Kelvin-Voigt model of viscoelasticity is used. The equations are discretised in 2D and 3D using an efficient time-stepping pseudospectral scheme. This uses the Fourier collocation spectral method to compute spatial derivatives and a leapfrog finite-difference scheme to integrate forwards in time. A multi-axial perfectly matched layer (M-PML) is implemented to allow free-field simulations using a finite-sized computational grid. Acceleration using a graphics processing unit (GPU) is supported via the MATLAB Parallel Computing Toolbox. An overview of the simulation functions and their theoretical and numerical foundations is described.**

I. Introduction

The simulation of elastic wave propagation has many applications in ultrasonics, including the classification of bone diseases and non-destructive testing [1]. In biomedical ultrasound in particular, elastic wave models have been used to investigate the propagation of ultrasound in the skull and brain, and to optimise the delivery of therapeutic ultrasound through the thoracic cage [2]. However, many existing elastic wave models are based on low-order finite difference or finite element schemes and thus require large numbers of grid points per wavelength to avoid numerical dispersion. Here, an accurate and computationally efficient elastic wave model is introduced as part of the open-source k-Wave MATLAB toolbox (http://www.k-wave.org) [3]. An overview of the numerical model is given, and the architecture of the simulation functions is described.

II. Numerical Model

A. Kelvin-Voigt Model

In an elastic medium, the propagation of compressional and shear waves can be described using Hooke's law and an expression for the conservation of momentum. For viscoelastic materials in which damping or absorption is present, Hooke's law is extended such that the stress-strain relation exhibits time dependent behaviour. For example, the classical Kelvin-Voigt model of viscoelasticity gives a time-dependent relationship that can be understood as the response of an elastic spring and viscous damper connected in parallel [4]. This model is widely used for studying the loss behaviour of viscoelastic materials. For an isotropic medium, the Kelvin-Voigt model can be written using Einstein summation notation as

$$\sigma_{ij} = \lambda\delta_{ij}\varepsilon_{kk} + 2\mu\varepsilon_{ij} + \chi\delta_{ij}\frac{\partial}{\partial t}\varepsilon_{kk} + 2\eta\frac{\partial}{\partial t}\varepsilon_{ij} \ . \quad (1)$$

Here σ is the stress tensor, ε is the dimensionless strain tensor, λ and μ are the Lamè parameters where μ is the shear modulus, and χ and η are the compressional and shear viscosity coefficients. The Lamè parameters are related to the shear and compressional sound speeds by

$$\mu = c_s^2\rho_0 \ , \qquad \lambda + 2\mu = c_p^2\rho_0 \ , \quad (2)$$

where ρ_0 is the mass density. Using the relationship between strain and particle displacement u_i for small deformations

$$\varepsilon_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \ , \quad (3)$$

Eq. (1) can be re-written as a function of the particle velocity v_i, where $v_i = \partial u_i/\partial t$

$$\begin{aligned}\frac{\partial \sigma_{ij}}{\partial t} &= \lambda\delta_{ij}\frac{\partial v_k}{\partial x_k} + \mu\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right) \\ &\quad + \chi\delta_{ij}\frac{\partial^2 v_k}{\partial x_k \partial t} + \eta\left(\frac{\partial^2 v_i}{\partial x_j \partial t} + \frac{\partial^2 v_j}{\partial x_i \partial t}\right) \ . \end{aligned} \quad (4)$$

To model the propagation of elastic waves, this is combined with an equation expressing the conservation of momentum. Written as a function of stress and particle velocity, this is given by

$$\frac{\partial v_i}{\partial t} = \frac{1}{\rho_0}\frac{\partial \sigma_{ij}}{\partial x_j} \ . \quad (5)$$

Equations (4) and (5) are coupled first-order partial differential equations that describe the propagation of linear compressional and shear waves in an isotropic viscoelastic solid. When the effect of the loss term is small, these equations account for absorption of the form $\alpha_p \approx \alpha_{0,p}\omega^2$ and $\alpha_s \approx \alpha_{0,s}\omega^2$ (for compressional and shear waves, respectively) [5]. Here ω is temporal frequency in rad/s and the power law absorption prefactors $\alpha_{0,p}$ and $\alpha_{0,s}$ in $\mathrm{Np\,(rad/s)^{-2}\,m^{-1}}$ are given by

$$\alpha_{0,p} = \frac{\chi + 2\eta}{2\rho_0 c_p^3} \ , \qquad \alpha_{0,s} = \frac{\eta}{2\rho_0 c_s^3} \ . \quad (6)$$

B. Pseudospectral Time Domain Solution

A computationally efficient model for elastic wave propagation in absorbing media can be constructed based on the explicit solution of the coupled equations given in Eqs. (4)-(5) using the Fourier pseudospectral method [6, 7]. This uses the Fourier collocation spectral method to compute spatial derivatives, and a leapfrog finite-difference scheme to integrate forwards in time. Using a temporally and spatially staggered grid, the field variables in 2D are updated in a time stepping fashion as follows (similarly for 3D):

(1) Calculate the spatial gradients of the stress field using the Fourier collocation spectral method

$$\begin{aligned}\partial_x\sigma_{xx}^- &= \mathcal{F}_x^{-1}\left\{ik_x e^{+ik_x\Delta x/2}\mathcal{F}_x\left\{\sigma_{xx}^-\right\}\right\}\\ \partial_y\sigma_{yy}^- &= \mathcal{F}_y^{-1}\left\{ik_y e^{+ik_y\Delta y/2}\mathcal{F}_y\left\{\sigma_{yy}^-\right\}\right\}\\ \partial_x\sigma_{xy}^- &= \mathcal{F}_x^{-1}\left\{ik_x e^{-ik_x\Delta x/2}\mathcal{F}_x\left\{\sigma_{xy}^-\right\}\right\}\\ \partial_y\sigma_{xy}^- &= \mathcal{F}_y^{-1}\left\{ik_y e^{-ik_y\Delta y/2}\mathcal{F}_y\left\{\sigma_{xy}^-\right\}\right\} \quad . \end{aligned} \tag{7a}$$

Here $\mathcal{F}_{x,y}\{\}$ and $\mathcal{F}_{x,y}^{-1}\{\}$ are the 1D forward and inverse Fourier transforms over the x and y dimensions, i is the imaginary unit, k_x and k_y are the discrete set of wavenumbers in each dimension, and Δx and Δy give the grid spacing assuming a uniform Cartesian mesh. The exponential terms are spatial shift operators that translate the output by half the grid point spacing (see Fig. 1). This improves the accuracy of the model.

(2) Update the particle velocity using a finite difference time step of size Δt, where the $+$ and $-$ superscripts denote the field values at the next and current time step

$$\begin{aligned} v_x^+ &= v_x^- + \frac{\Delta t}{\rho_0}\left(\partial_x\sigma_{xx}^- + \partial_y\sigma_{xy}^-\right)\\ v_y^+ &= v_y^- + \frac{\Delta t}{\rho_0}\left(\partial_x\sigma_{xy}^- + \partial_y\sigma_{yy}^-\right) \quad . \end{aligned} \tag{7b}$$

(3) Calculate the spatial gradients of the updated particle velocity using the Fourier collocation spectral method

$$\begin{aligned}\partial_x v_x^+ &= \mathcal{F}_x^{-1}\left\{ik_x e^{-ik_x\Delta x/2}\mathcal{F}_x\left\{v_x^+\right\}\right\}\\ \partial_y v_x^+ &= \mathcal{F}_y^{-1}\left\{ik_y e^{+ik_y\Delta y/2}\mathcal{F}_y\left\{v_x^+\right\}\right\}\\ \partial_x v_y^+ &= \mathcal{F}_x^{-1}\left\{ik_x e^{+ik_x\Delta x/2}\mathcal{F}_x\left\{v_y^+\right\}\right\}\\ \partial_y v_y^+ &= \mathcal{F}_y^{-1}\left\{ik_y e^{-ik_y\Delta y/2}\mathcal{F}_y\left\{v_y^+\right\}\right\} \quad . \end{aligned} \tag{7c}$$

(4) Calculate the spatial gradients of the time derivative of the particle velocity using Eq. (5)

$$\begin{aligned}\partial_x\partial_t v_x^- &= \mathcal{F}_x^{-1}\left\{ik_x e^{-ik_x\Delta x/2}\mathcal{F}_x\left\{\left(\partial_x\sigma_{xx}^- + \partial_y\sigma_{xy}^-\right)/\rho_0\right\}\right\}\\ \partial_y\partial_t v_x^- &= \mathcal{F}_y^{-1}\left\{ik_y e^{+ik_y\Delta y/2}\mathcal{F}_y\left\{\left(\partial_x\sigma_{xx}^- + \partial_y\sigma_{xy}^-\right)/\rho_0\right\}\right\}\\ \partial_x\partial_t v_y^- &= \mathcal{F}_x^{-1}\left\{ik_x e^{+ik_x\Delta x/2}\mathcal{F}_x\left\{\left(\partial_x\sigma_{xy}^- + \partial_y\sigma_{yy}^-\right)/\rho_0\right\}\right\}\\ \partial_y\partial_t v_y^- &= \mathcal{F}_y^{-1}\left\{ik_y e^{-ik_y\Delta y/2}\mathcal{F}_y\left\{\left(\partial_x\sigma_{xy}^- + \partial_y\sigma_{yy}^-\right)/\rho_0\right\}\right\} . \end{aligned} \tag{7d}$$

(5) Update the stress field using a finite difference time step

$$\begin{aligned}\sigma_{xx}^+ = \sigma_{xx}^- &+ \lambda\Delta t\left(\partial_x v_x^+ + \partial_y v_y^+\right) + \mu\Delta t\left(2\partial_x v_x^+\right)\\ &+ \chi\Delta t\left(\partial_x\partial_t v_x^- + \partial_y\partial_t v_y^-\right) + \eta\Delta t\left(2\partial_x\partial_t v_x^-\right)\\ \sigma_{yy}^+ = \sigma_{yy}^- &+ \lambda\Delta t\left(\partial_x v_x^+ + \partial_y v_y^+\right) + \mu\Delta t\left(2\partial_y v_y^+\right)\\ &+ \chi\Delta t\left(\partial_x\partial_t v_x^- + \partial_y\partial_t v_y^-\right) + \eta\Delta t\left(2\partial_y\partial_t v_y^-\right)\\ \sigma_{xy}^+ = \sigma_{xy}^- &+ \mu\Delta t\left(\partial_y v_x^+ + \partial_x v_y^+\right)\\ &+ \eta\Delta t\left(\partial_y\partial_t v_x^- + \partial_x\partial_t v_y^-\right) \quad . \end{aligned} \tag{7e}$$

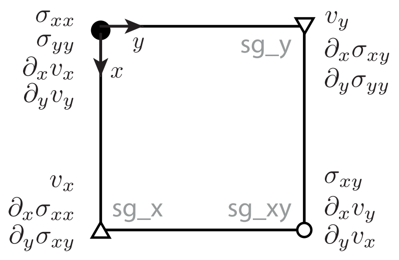

Fig. 1. Position of the field quantities and their derivatives on a spatially staggered grid in 2D. The derivatives $\partial_i\partial_t v_j$ (etc) are staggered in the same way as the $\partial_i v_j$ terms.

Here the Lamè parameters λ, μ and viscosity coefficients χ, η used in the time loop are calculated from the material properties c_p, c_s, ρ_0, $\alpha_{0,p}$, $\alpha_{0,s}$ defined by the user using Eqs. (2) and (6). For equations involving the spatially staggered grid parameters, the material properties are understood to be values defined at the staggered grid points. In addition to spatial staggering, the stress and velocity fields are also temporally staggered by $\Delta t/2$. Time varying stress and velocity sources are implemented after the update steps by adding the source terms to the relevant field values at the desired grid points within the domain. Similarly, outputs are calculated by storing the field values at the desired grid points at the end of each time step. To simulate free-field conditions, a multi-axial split-field perfectly matched layer (M-PML) is also applied to absorb the waves at the edge of computational domain [8].

III. The k-Wave Toolbox

The discrete equations given in the previous section were implemented in MATLAB as part of the open-source k-Wave toolbox (available from http://www.k-wave.org) [3]. This also contains functions for the simulation of linear and nonlinear wave fields in fluid media [9], and for the reconstruction of photoacoustic images [3]. The elastic simulation functions (named `pstdElastic2D` and `pstdElastic3D`) are called with four input structures (`kgrid`, `medium`, `source`, and `sensor`) in the same way as the other wave models in the toolbox. These structures define the properties of the computational grid, the distribution of medium properties, stress and velocity source terms, and the locations of the sensor points used to record the evolution of the wave field over time. A list of the main structure fields is given in Table I.

A simple example of a MATLAB script to perform an elastic wave simulation in a layered medium in 2D is given in Program 1. First, the computational grid is defined using the function `makeGrid`. This takes the number and spacing of the grid points in each Cartesian direction and returns an object of the `kWaveGrid` class. The time steps used in the simulation are defined by the object property `kgrid.t_array`. By default, this is set to `'auto'`, in which case the time array is automatically calculated within the simulation functions using the time taken to travel across the longest grid diagonal at the slowest sound speed, and a Courant-Friedrichs-Lewy (CFL) number of 0.1, where CFL = $c_0\Delta t/\Delta x$. The time array can also be defined by the user, either using the function `makeTime`, or explicitly in the form `kgrid.t_array = 0:dt:t_end`. The time array must be evenly spaced and monotonically increasing.

Program 1 Script for the simulation of an explosive pressure source in a layered fluid-solid half-space in 2D.

```
% create the computational grid
Nx = 128;               % [grid points]
Ny = 128;               % [grid points]
dx = 0.1e-3;            % [m]
dy = 0.1e-3;            % [m]
kgrid = makeGrid(Nx, dx, Ny, dy);

% define the compressional sound speed [m/s]
medium.sound_speed_compression = 1500*ones(Nx, Ny);
medium.sound_speed_compression(Nx/2:end, :) = 2000;

% define the shear sound speed [m/s]
medium.sound_speed_shear = zeros(Nx, Ny);
medium.sound_speed_shear(Nx/2:end, :) = 800;

% define the mass density [kg/m^3]
medium.density = 1000*ones(Nx, Ny);
medium.density(Nx/2:end, :) = 1200;

% define the absorption coefficients [dB/(MHz^2 cm)]
medium.alpha_coeff_compression = 0.1;
medium.alpha_coeff_shear = 0.5;

% define the initial pressure distribution
disc_magnitude = 5;     % [Pa]
disc_x_pos = 40;        % [grid points]
disc_y_pos = 64;        % [grid points]
disc_radius = 5;        % [grid points]
source.p0 = disc_magnitude*makeDisc(Nx, Ny,
  disc_x_pos, disc_y_pos, disc_radius);

% define a circular binary sensor mask
radius = 20;            % [grid points]
sensor.mask = makeCircle(Nx, Ny, Nx/2, Ny/2, radius);

% run the simulation
sensor_data = pstdElastic2D(kgrid, medium, source,
  sensor);
```

After the computational grid, the medium properties are defined. For a homogeneous medium, these are given as single scalar values in SI units. For a heterogeneous medium, these are defined as matrices the same size as the computational grid. There is no restriction on the distribution or values for the material properties. In this example, a heterogeneous medium is defined as a layered fluid-solid interface.

Next, any source terms are defined. Three types of source are currently supported: an initial pressure distribution (which is multiplied by -1 and assigned to the normal components of the stress), time varying velocity sources, and time varying stress sources. For time varying sources, the location of the source is specified by assigning a binary matrix (i.e., a matrix of 1's and 0's with the same dimensions as the computational grid) to `source.s_mask` or `source.u_mask`, where the 1's represent the grid points that form part of the source. The time varying input signals are then assigned to `source.sxx` (etc) or `source.ux` (etc). By default, the stress and velocity sources are added to the field variables as the injection of mass and force, respectively. The source values can also be used to replace the current values of the field variables by setting `source.s_mode` or `source.u_mode` to `'dirichlet'`. In Program 1, an initial pressure distribution within the fluid layer is defined in the shape of a small disc.

Finally, the sensor points are defined. These specify where in the computational domain the field variables are sampled at each time step. This can be defined in three ways. (1) As a binary matrix (i.e., a matrix of 1's and 0's) representing the grid points within the computational grid that will collect the data. (2) As the grid coordinates of two opposing corners of a rectangle (2D) or cuboid (3D). (3) As a series of Cartesian coordinates within the grid which specify the location of the field values stored at each time step. If the Cartesian coordinates don't exactly match the coordinates of a grid point, the output values are calculated via interpolation. In Program 1, a binary sensor mask in the shape of a circle is used. By default, only the acoustic pressure (given by the negative of the average normal stress) is stored. Other field parameters can also be returned by defining `sensor.record`.

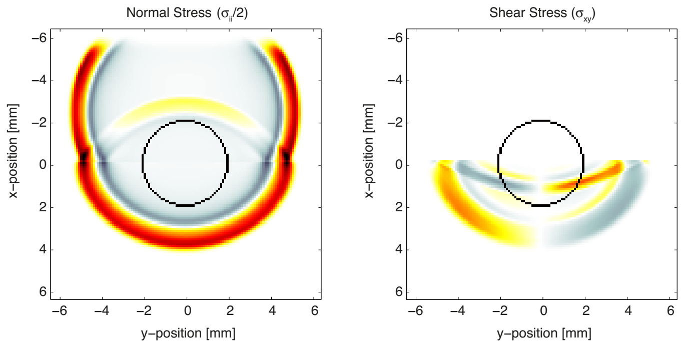

Fig. 2. Snapshot of the 2D simulation given in Program 1. The circular sensor mask is shown as a black circle. Both compression and shear waves can be seen in the lower layer. By default, the visualisation is updated every 10 time-steps. The colour map displays positive field values as yellows through reds to black, zero values as white, and negative values as light to dark blue-greys.

When the input structures have been defined, the simulation is started by passing them to `pstdElastic2D` (in 2D) or `pstdElastic3D` (in 3D). The propagation of the wave field is then computed step by step, with the field values at the sensor points stored after each iteration. By default, a visualisation of the propagating wave field and a status bar are displayed, with frame updates every ten time steps. The display is divided into two showing the normal and shear components of the stress field. In 3D, three intersecting planes through the centre of the grid are displayed. The default k-Wave colour map displays positive values as yellows through reds to black, zero values as white, and negative values as light to dark blue-greys. A snapshot of the visualisation produced by Program 1 is shown in Fig. 2. The absorption within the M-PML at the top of the domain is clearly visible.

When the time loop is complete, the function returns the field variables recorded at the sensor points defined by `sensor.mask`. If the sensor mask is given as a set of Cartesian coordinates, the computed `sensor_data` is returned in the same order. If `sensor.mask` is given as a binary matrix, `sensor_data` is returned using MATLAB's column-wise linear matrix index ordering. In both cases, the recorded data is indexed as `sensor_data(position_index, time_index)`. If `sensor.record` is defined, the output `sensor_data` is returned as a structure with the different outputs appended as structure fields. For example, if `sensor.record = {'u'}`, the output would contain the fields `sensor_data.ux` and `sensor_data.uy`.

TABLE I. SUMMARY OF THE MAIN FIELDS FOR THE FOUR INPUT STRUCTURES USED BY `pstdElastic2D` and `pstdElastic3D`.

Field	Description
`kgrid.kx`, `kgrid.Nx`, `kgrid.dx`, etc	Cartesian and wavenumber grid parameters returned by `makeGrid`
`kgrid.t_array`	Evenly spaced array of time points [s]
`medium.sound_speed_compression`	Matrix (or single value) of the compressional sound speed at each grid point within the medium [m/s]
`medium.sound_speed_shear`	Matrix (or single value) of the shear sound speed at each grid point within the domain [m/s]
`medium.density`	Matrix (or single value) of the mass density at each grid point within the domain [kg/m^3]
`medium.alpha_coeff_compression`	Matrix (or single value) of the power law absorption prefactor for compressional waves [dB/(MHz^2 cm)]
`medium.alpha_coeff_shear`	Matrix (or single value) of the power law absorption prefactor for shear waves [dB/(MHz^2 cm)]
`source.p0`	Matrix of the initial pressure distribution at each grid point within the domain [Pa]
`source.s_mask`	Binary matrix specifying the positions of the time varying stress source
`source.sxx`, `source.sxy`, etc	Matrix of time varying stress input/s at each of the source positions given by `source.s_mask` [Pa]
`source.u_mask`	Binary matrix specifying the positions of the time varying particle velocity source
`source.ux`, `source.uy`, etc	Matrix of time varying particle velocity input/s at each of the source positions given by `source.u_mask` [m/s]
`sensor.mask`	Binary matrix or a set of Cartesian points specifying the positions where the field is recorded at each time-step
`sensor.record`	Cell array listing the field variables to record, e.g., `{'p', 'u'}`
`sensor.record_start_index`	Time index at which the sensor should start recording

The behaviour of the simulation functions can be further controlled through the use of optional input parameters. These are given as `param, value` pairs following the four input structures. For example, the visualisation can be automatically recorded by setting `'RecordMovie'` to `true`, and the plot scale can be controlled by setting `'PlotScale'` in the form `[sii_min, sii_max, sij_min, sij_max]` (this defaults to `[-1, 1, -1, 1]`). Similarly, simulations can be run on an NVIDIA graphics processing unit (GPU) using the MATLAB Parallel Computing Toolbox by setting `'DataCast'` to `'gpuArray-single'`. The functions can also be used for time reversal image reconstruction in photoacoustics by assigning the recorded pressure values to `sensor.time_reversal_boundary_data`. This data is then enforced in time reversed order as a time varying Dirichlet boundary condition over the sensor surface given by `sensor.mask`. A full list and description of the different input options are given in the html help files and examples contained within the k-Wave toolbox.

IV. CONCLUSION

A new model for simulating the propagation of elastic waves using the k-Wave MATLAB toolbox is described. The model is based on two coupled first-order partial differential equations describing the variation of stress and particle velocity in an isotropic viscoelastic (Kelvin-Voigt) medium. These are discretised using an efficient pseudospectral time domain scheme. Spatial derivatives are computed using the Fourier collocation spectral method, while time integration is performed using a leapfrog finite difference. The new functions (named `pstdElastic2D` and `pstdElastic3D`) are called in the same way as the other wave models in k-Wave. The inputs are defined as fields to four input structures, with additional behaviour defined using optional input parameters. The medium parameters (shear and compressional sound speed, shear and compressional absorption coefficients, and mass density) can be heterogeneous and are defined as matrices the same size as the computational grid. The current code is implemented in MATLAB using a simple finite difference time scheme and assumes the medium is isotropic. In the future, this will be extended to account for orthotropic materials in which the planes of symmetry are aligned with the computational grid, and new models using a k-space corrected finite difference time scheme will also be introduced [10]. Versions of the code written in C++ based on OpenMP and MPI will also be developed and released at a later date [11].

ACKNOWLEDGMENT

This work was supported by the Dosimetry for Ultrasound Therapy project which is part of the European Metrology Research Programme (EMRP). The EMRP is jointly funded by the EMRP participating countries within EURAMET and the European Union. Jiri Jaros is supported by the SoMoPro II programme, REA Grant Agreement No. 291782.

REFERENCES

[1] E. Bossy, F. Padilla, F. Peyrin, and P. Laugier. Three-dimensional simulation of ultrasound propagation through trabecular bone structures measured by synchrotron microtomography. *Phys. Med. Biol.*, 50(23):5545–56, 2005.

[2] K. Okita, R. Narumi, T. Azuma, S. Takagi, and Y. Matumoto. The role of numerical simulation for the development of an advanced HIFU system. *Comput. Mech.*, 2014.

[3] B. E. Treeby and B. T. Cox. k-Wave: MATLAB toolbox for the simulation and reconstruction of photoacoustic wave fields. *J. Biomed. Opt.*, 15(2):021314, 2010.

[4] J. J. Markham, R. T. Beyer, and R. B. Lindsay. Absorption of sound in fluids. *Reviews of Modern Physics*, 23(4):353, 1951.

[5] H. F. Pollard. *Sound Waves in Solids*. Pion Ltd, London, 1977.

[6] Q. H. Liu. Large-scale simulations of electromagnetic and acoustic measurements using the pseudospectral time-domain (PSTD) algorithm. *IEEE. T. Geosci. Remote*, 37(2):917–926, 1999.

[7] M. Caputo, J. M. Carcione, and F. Cavallini. Wave simulation in biologic media based on the Kelvin-voigt fractional-derivative stress-strain relation. *Ultrasound Med. Biol.*, 37(6):996–1004, 2011.

[8] K. C. Meza-Fajardo and A. S. Papageorgiou. On the stability of a non-convolutional perfectly matched layer for isotropic elastic media. *Soil Dyn. Earthq. Eng.*, 30(3):68–81, 2010.

[9] B. E. Treeby, J. Jaros, A. P. Rendell, and B. T. Cox. Modeling nonlinear ultrasound propagation in heterogeneous media with power law absorption using a k-space pseudospectral method. *J. Acoust. Soc. Am.*, 131(6):4324–4336, 2012.

[10] K. Firouzi, B. T. Cox, B. E. Treeby, and N. Saffari. A first-order k-space model for elastic wave propagation in heterogeneous media. *J. Acoust. Soc. Am.*, 132(3):1271–1283, 2012.

[11] J. Jaros, A. P. Rendell, and B. E. Treeby. Full-wave nonlinear ultrasound simulation on distributed clusters with applications in high-intensity focused ultrasound. *arXiv:1408.4675 [physics.med-ph]*, 2014.