ME 8873 - Georgia Tech

Lecture 19: Ultrasonics Simulations in k-Wave

10 March 2019

Lecturer: Scott Schoen Jr Updated: 2020-04-28

1 Numerical Methods

1.1 Motivation

Why do we need numerical methods? After all, don't solutions to the wave equation have the form

$$p = f(x - ct) + g(x + ct)\,? \tag{1}$$

Well, yes, in the case of a linear, unbounded, unforced, homogeneous medium. For realistic environments, like in the body with different tissues, interfaces, and bone, analytical approaches aren't really an option. In these cases, we make numerical approximations to our governing equations to be able to write computer programs to solve them.

1.2 Basic Finite Differences

Finite differences are a way to approximate derivatives by using the values on a discrete computational grid, with the idea that at small grid spacings, the approximation is reasonable. Consider the Taylor expansions

$$p(x + \Delta x) = p(x) + \frac{\partial p}{\partial x}\Delta x + \frac{1}{2!}\frac{\partial^2 p}{\partial x^2}(\Delta x)^2 + \frac{1}{3!}\frac{\partial^3 p}{\partial x^3}(\Delta x)^3 + \mathcal{O}\left[(\Delta x)^4\right] \tag{2}$$

and

$$p(x - \Delta x) = p(x) - \frac{\partial p}{\partial x}\Delta x + \frac{1}{2!}\frac{\partial^2 p}{\partial x^2}(\Delta x)^2 - \frac{1}{3!}\frac{\partial^3 p}{\partial x^3}(\Delta x)^3 + \mathcal{O}\left[(\Delta x)^4\right] \tag{3}$$

Now subtract Eq. (3) from Eq. (2) so and divide by $2\Delta x$

$$\frac{p(x + \Delta x) - p(x - \Delta x)}{2\Delta x} = \frac{\partial p}{\partial x} + \frac{1}{3!}\frac{\partial^3 p}{\partial x^3}(\Delta x)^2 + \mathcal{O}\left[(\Delta x)^5\right] \tag{4}$$

Thus we can write

$$\frac{\partial p}{\partial x} \approx \frac{p(x + \Delta x) - p(x)}{\Delta x}, \tag{5}$$

i.e., that the derivative of the function may be approximated from its values at fixed points. Because the approximation in Eq. (5) has discarded terms of order $(\Delta x)^2$ and higher, we must be sure that these terms are small so that the approximation is valid. Determining the upper limit of the step size Δx is out of scope. But for our purposes, a good benchmark is that Δx must be appreciably smaller than wavelength.

Now if we were to instead add Eq. (2) to Eq. (3), and by using the notation $p(m\Delta x, n\Delta t) = p_m^n$, we have

$$\frac{\partial^2 p}{\partial t^2} \approx \frac{p_m^{n+1} - 2p_m^n + p_m^{n-1}}{(\Delta t)^2} + \mathcal{O}\left[(\Delta t)^2\right] \tag{6}$$

$$\frac{\partial^2 p}{\partial x^2} \approx \frac{p_{m+1}^n - 2p_m^n + p_{m-1}^n}{(\Delta x)^2} + \mathcal{O}\left[(\Delta x)^2\right] . \tag{7}$$

Then with our initial condition for p, and use of the governing equations[†] and boundary conditions[‡] we can use Eqs. (6) and (7) to compute the field over time.

Consider a 1D wave where the sound speed is a function of the position (i.e., the medium is heterogeneous). Then, the finite difference equation becomes

$$\begin{aligned} \frac{\partial^2 p}{\partial x^2} &= \frac{1}{c^2(x)} \frac{\partial^2 p}{\partial t^2} \\ \frac{p_{m+1}^n - 2p_m^n + p_{m-1}^n}{(\Delta x)^2} &= \frac{p_m^{n+1} - 2p_m^n + p_m^{n-1}}{(c_m \, \Delta t)^2} . \end{aligned} \tag{8}$$

Then, if we use the shorthand $\xi_m \equiv (c_m \Delta t / \Delta x)^2$, Eq. (8) can be rearranged to

$$p_n^{n+1} = -p_m^{n-1} + 2p_m^n(1 - \xi_m) + \xi_m \left(p_{m+1}^n + p_{m-1}^n\right) . \tag{9}$$

Equation (9) describes a matrix equation which must be inverted to find the pressure at subsequent time steps. Details of the process for more dimension as well as the accuracy of the method, as well as the grid spacings that must be chosen to ensure out solution doesn't blow up is a field unto itself, but some basic guidelines are as follows.

1.3 Limitations on Method

While the relations in Eqs. (6) and (7) can always be used, there are some limitations. First note the terms we're ignoring, which will lead to errors. These errors will have order

$$\epsilon \sim \left\|(\Delta t)^2 + (\Delta x)^2\right\| , \tag{10}$$

[†]The wave equation may be thought of as the governing equation for a homogeneous medium. In the heterogeneous medium case, the governing equations must be written from the more general conservation laws, since, e.g., $\nabla\rho \neq 0$. See the k-Wave documentation for full details.

[‡]While simulations with totally rigid or completely traction-free boundary conditions may be of interest, here we typically want waves that leave the domain to exit and not come back. In this case, an absorbing boundary layer or perfectly matched layer is an appropriate choice; see Sec. 3.

and accumulate over time. In order to ensure that these errors do not grow uncontrollably, the CFL condition is placed on the Courant number C:

$$C \equiv \left\| \frac{c\Delta t}{\Delta x} \right\| . \tag{11}$$

Note that the CFL condition ensures stability (i.e., that the error terms will not blow up as the simulation runs), not necessarily accuracy (i.e., that the solution we get is sufficiently close to the correct answer).

Example 1: Stable FDTD Simulation Requirements

Suppose we want to run a 3D simulation with a skull bone about $20\,\mathrm{cm}$ across. Take $c_{\max} = 2000\,\mathrm{m/s}$ and $\rho_{\max} = 1580\,\mathrm{kg/m^3}$, and want to use a frequency of $2\,\mathrm{MHz}$.

What is the shortest possible wavelength?
Ans: $\lambda = c_{\max}/f_{\max} = (2000\,\mathrm{m/s})/(2\,\mathrm{MHz}) = 1\,\mathrm{mm}$.

A rule of thumb is 10 points per wavelength. How many grid points do we need?
Ans: From above, we need $\Delta x = 1\,\mathrm{mm}/10 = 100\,\mathrm{\mu m}$. Then in each dimension, $N = (D_x/\Delta x) = [(20\,\mathrm{cm})(100\,\mathrm{\mu m})] = 2000$, so we need $N^3 = 2 \times 10^9$ grid points. Note that at double precision, this requires $64\,\mathrm{GB}$ of memory to hold a single field value.

What space step we need to have $C < 1$?
Ans: $\Delta t = C\Delta x/c_{\max} = [(1)(100\,\mathrm{\mu m})]/(2000\,\mathrm{m/s}) = 50\,\mathrm{ns}$.

1.4 Pseudospectral Methods & k-Wave

Recall that plane waves have the nice property that

$$\frac{\partial p}{\partial x} = \frac{\partial}{\partial x}\left[P_0 e^{ikx}\right] = ik\,p. \tag{12}$$

Well the (spatial) Fourier transform allows us to write the pressure as a sum of plane waves with different wavenumbers. Then, we can take the derivatives of all components separately, add them back up, and take the inverse transform! That is,

$$\frac{\partial p}{\partial x} = \mathcal{F}_k^{-1}\left\{ \mathcal{F}_k\left[\frac{\partial p}{\partial x}\right] \right\} = \mathcal{F}_k^{-1}\left\{ ik\,\mathcal{F}_k\left[p\right] \right\}. \tag{13}$$

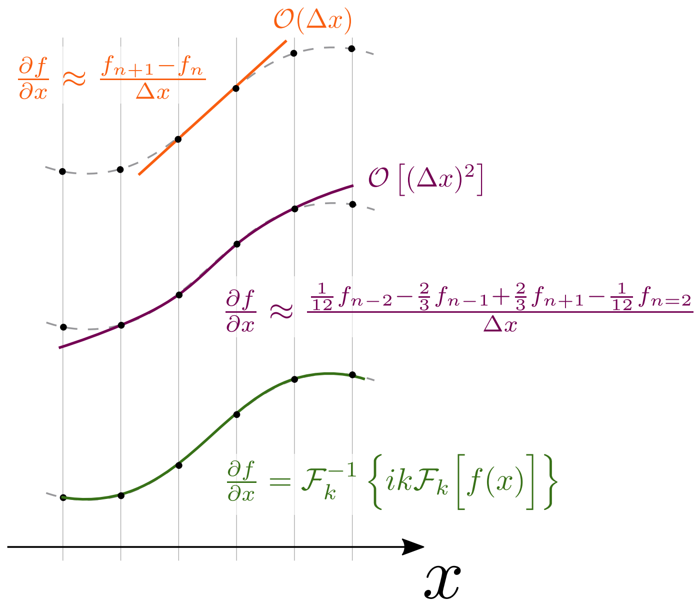

$\mathcal{O}(\Delta x)$
$\frac{\partial f}{\partial x} \approx \frac{f_{n+1}-f_n}{\Delta x}$
$\mathcal{O}\left[(\Delta x)^2\right]$
$\frac{\partial f}{\partial x} \approx \frac{\frac{1}{12}f_{n-2}-\frac{2}{3}f_{n-1}+\frac{2}{3}f_{n+1}-\frac{1}{12}f_{n=2}}{\Delta x}$
$\frac{\partial f}{\partial x} = \mathcal{F}_k^{-1}\left\{ik\mathcal{F}_k\left[f(x)\right]\right\}$
x

Figure 1: Comparison of first and second order finite difference schemes with the spectral derivative calculation.

Example 2: Prove It

Without loss of generality, consider the 1D case. By definition,

$$\mathcal{F}_k\left[\frac{\partial p}{\partial x}\right] \equiv \int_{-\infty}^{+\infty} \frac{\partial p}{\partial x}\, e^{-ik_x x}\, \mathrm{d}x\,. \tag{14}$$

Integrating by parts gives

$$\mathcal{F}_k\left[\frac{\partial p}{\partial x}\right] = p(x)\, e^{-ik_x x}\Big|_{-\infty}^{+\infty} - \int_{-\infty}^{+\infty} -ik_x\, e^{-ik_x x}\, p(x)\, \mathrm{d}x\,. \tag{15}$$

Now guaranteeing the existence of the transform requires that p is square integrable. Thus since $e^{ik_x x}$ has unity modulus, it must be true that $\lim_{x\to\pm\infty} p = 0$. Thus the first term in Eq. (15) must vanish, leaving

$$\mathcal{F}_k\left[\frac{\partial p}{\partial x}\right] = \underbrace{p(x)\, e^{-ik_x x}\Big|_{-\infty}^{+\infty}}_{=0} - \underbrace{\int_{-\infty}^{+\infty} -ik_x\, e^{-ik_x x}\, p(x)\, \mathrm{d}x}_{\mathcal{F}_k[ik_x\, p(x)]} = ik_x\mathcal{F}_k\left[p(x)\right]\,. \tag{16}$$

This has the benefit that spatial derivatives will be *exact*, provided our grid spacing is at least half the shortest wavelength; see Fig. 1. Additionally, pseudospectral methods replace, in the first-order time discretizations,

$$\Delta t \to \left(\operatorname{sinc} \frac{c_{\mathrm{ref}} k \Delta t}{2}\right) \Delta t \tag{17}$$

(termed the “k-space operator”) to enable larger time steps (note that these become equivalent as $\Delta t \to 0$). Additionally, k-Wave uses a staggered grid to improve the order of the method.

2 k-Wave Structures

The k-Wave simulation codes expect as inputs MATLAB structures describing the environment, sources, receivers, and simulation options. Each is discussed briefly in the following, in the rough order in which they should be defined. All examples are given in 2D (x, y), but extension to 3D is mostly straightforward with the addition of a z-component.

2.1 `kgrid`

The `kgrid` defines the computational grid on which the simulation will be run. It can be assembled with the built-in function `kWaveGrid` by specifying the number of points in each dimension, and the spacing in each direction. All values (except absorption, see Sec. 2.2.1) are specified in SI units.

```
% Set grid dimensions (10 cm by 10 cm square)
Nx = 100;
dx = 0.1E-3; % [m]
Ny = 100;
dy = 0.1E-3; % [m]

% Define k-Wave grid
kgrid = kWaveGrid( Nx, dx, Ny, dy );
```

A word of caution: k-Wave uses the coordinate system (x, y) such that addressing is row, column. This is the natural choice for the coding, but beware that plots will appear rotated (since x varies along the rows, it appears as the vertical dimension). Keep this in mind when defining your `medium`, `sources` (Sec. 2.3), and `sensors` (Sec. 2.4).

2.2 `medium`

The `medium` structure contains at a minimum the sound speed and density at every point of the the `kgrid`. These may be set as constants for a uniform medium.

```
% Define homogeneous properties
c0 = 1500; % [m/s]
rho0 = 1000; % [kg/m^3]
```

```

% Assign to kWaveGrid
medium.sound_speed = c0;
medium.density = rho0;
```

If we want the medium to have properties that vary in space, we will need to define them over the grid. For example, suppose we want to have $c(x, y) = c_0 (1 - |x|/x_0)$, then we would have

```
% Define sound speed field
c0 = 1500; % [m/s]
x0 = 1; % [m]
c = c0.*( 1 - abs(kgrid.x)./x0 );

% Assign to k-Wave medium
medium.sound_speed = c;
```

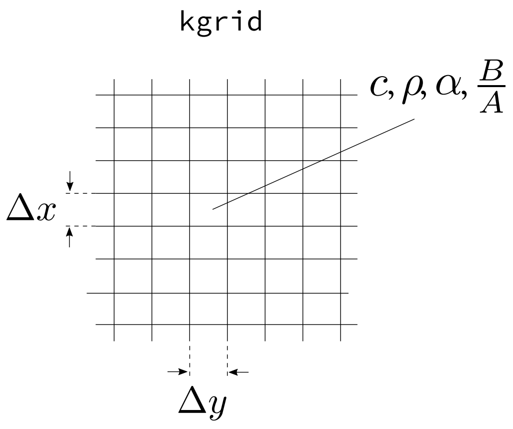


Figure 2: The `medium` contains a matrix that defines the sound speed and density (and optionally the absorption coefficient and nonlinearity parameter) at each point in `kgrid`.

2.2.1 Absorption

Absorption in k-Wave is modeled with a power-law attenuation model. The details are out of scope, but essentially the absorption is incorporated in the governing equations as

$$\alpha(x, y) = \alpha_0(x, y) \cdot f^{\beta} . \tag{18}$$

Here, α_0 is the attenuation coefficient **decibels per centimeter per megahertz**$^{\beta}$. That is to say, at 1 MHz, the wave is attenuated at α_0 dB/cm. This the most common unit, however, these values are sometimes reported in nepers per meter, so use caution when looking them up. Different materials have different power law coefficients and exponents; however, only the coefficient may be spatially dependent.

```
% Define attenuation field
alpha0 = 0.2; % Value for blood [dB/cm/MHz^beta]
beta = 1.2; % Average for whole domain

% Assign to k-Wave medium
medium.alpha_coeff = alpha0;
medium.alpha_power = beta;
```

2.2.2 Nonlinearity Parameter

In addition to the sound speed, density, and attenuation coefficient, the nonlinearity parameter may be set in the `medium.BonA` field. Note that the acoustic nonlinearity is often specified as $\beta = 1 + B/2A$, so use caution when assigning this parameter.

2.2.3 Thermal Properties

For thermal simulation in k-Wave, a few additional fields are required to solve the governing equation. They are specified the same way as the sound speed and density (i.e., they can be spatially-dependent), for instance

```
% Heat diffusion proerpties
k = 0.52; % [W/(m.K)]
Cp = 3540; % [J/(kg.K)]

% Perfusion properties
rhoBlood = 1060; % [kg/m^3]
CpBlood  = 3617; % [J/(kg.K)]
perfusion = 0.01; % [1/s]
TBlood = 37; % [deg C]

% Assign to medium
medium.thermal_conductivity = k;
medium.specific_heat  = Cp;
medium.blood_density = rhoBlood;
medium.blood_specific_heat = CpBlood;
medium.blood_perfusion_rate = perfusion;
medium.blood_ambient_temperature = TBlood;
```

2.2.4 Time Array

By default the time vector is automatically created by k-Wave when the simulation is run. However, if we want to specify a source pressure as a function of time (see Sec. 2.3), we'll need to have this before running the simulation. In this case, we can set a minimum CFL number and end time and have k-Wave create an appropriate time vector for us with the `makeTime` function.

```matlab
% Define CFL number. Should be as small as possible (though smaller numbers
% will cause longer simulation times).
minCfl = 0.3;

% Define time to end the simulation at
tMax = 1E-3; % [s]

% Create time array
kgrid.t_array = makeTime( kgrid, medium.sound_speed, minCfl, tMax );
```

2.3 source

The `source` structure allows specification of the pressure or velocity at arbitrary points on the grid. This is accomplished with two fields:[†]

1. `p_mask`: A binary mask is created indicating where the pressure will be specified.
2. `p`: A matrix containing the time series for each point in `p_mask`.

2.3.1 source.p_mask

This grid is a matrix of 1s and 0s the same size as the `kWaveGrid`. Note that we must specify an *index* in the grid to place the source. This is a little cumbersome, since it's perhaps more natural to specify its position. However, this conversion between position and index may be accomplished as below:[‡]

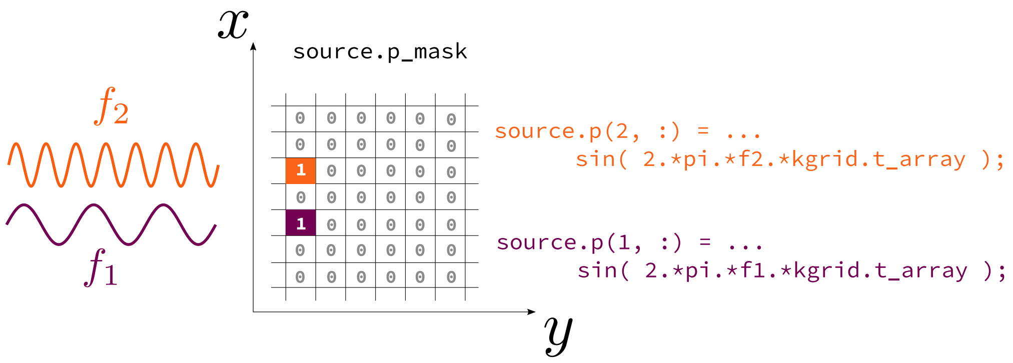
```matlab
% Define position of source
xSrc = 1E-3;
ySrc = -3E-3;

% Find indices closest to desired point
[~, xInd] = min( abs( xSrc - kgrid.x_vec ) );
[~, yInd] = min( abs( ySrc - kgrid.y_vec ) );

% Set the mask to 1 at the desired point
source.p_mask( xInd, yInd ) = 1;
```

[†]k-Wave is capable of defining velocity sources as well, but for simplicity only the pressure case is discussed here.

[‡]Note here we are using the `x_vec` and `y_vec` (rather than `x` and `y`) fields of the `kgrid` since we want to search along these 1D vectors.

2.3.2 `source.p`

For each point where `source.p_mask` is one, a pressure may be specified, either as a constant or as a time-varying value. The time series are specified as an N_s-by-N_t matrix, where N_s is the number of sources (i.e., the number of ones in `source.p_mask`) and N_t is the number of time points (i.e., the length of `kgrid.t_array`).



```
source.p(2, :) = ...
      sin( 2.*pi.*f2.*kgrid.t_array );

source.p(1, :) = ...
      sin( 2.*pi.*f1.*kgrid.t_array );
```

Figure 3: Specifying source time series.

The time series are assigned by the index on the source, rather than by the position. If we want to emit two sine waves with frequencies f_1 and f_2 as in Fig. 3, we would specify the time series in `source.p` *in the order in which the 1s appear in* `source.p_mask`.

2.3.3 Thermal Properties

For thermal simulations, two additional source properties are required: the initial temperature distribution on the grid, and the locations and strengths of any heat sources. The temperature may be specified through the `source.T0` field, and any sources of heat through the `source.Q` field.

2.4 `sensor`

The `sensor` structure specifies where field variables will be recorded. Just like the `source` structure, it has a mask field `sensor.mask` that is a binary matrix defining at which *indices* the pressure (or velocity as specified, see Sec. 2.4.1) will be recorded. The source and sensor masks are summarized in Fig. 4.

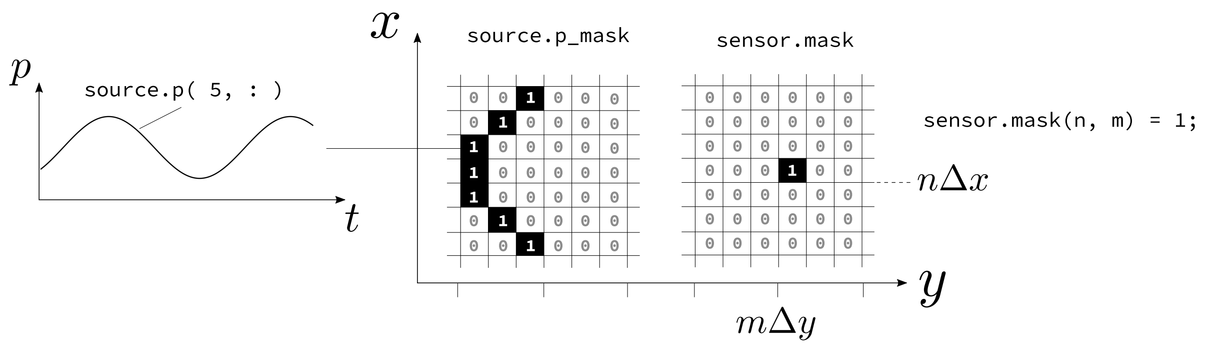


Figure 4: Source time series are defined relative to the order of the non-zero entries in `source.p_mask`.

2.4.1 `sensor.record`

The `sensor.record` field specifies which variables will be recorded at the sensor positions defined by `sensor.mask`. For instance, if we want to record the pressure and velocity at each sensor, we would use

```
% Record the pressure and velocity time series
sensor.record = { 'p', 'u' };
```

Of particular interest is the `p_max_all`, which stores the maximum pressure over the simulation at each of the entire grid. This is useful so that we don't have to specify a sensor at every grid point and consequently the whole time series for each.

2.4.2 `sensor_data`

The output of the simulation is stored in a structure `sensor_data` that contains the values of the parameters specified in the `sensor.record` field.[†] Time series values (e.g., the pressure or particle velocity) are indexed similarly to the `source.p` source conditions (see Sec. 2.3). For instance, if `sensor.mask` has 10 non-zero elements, then `sensor.p` will have 10 rows.

2.5 Helper Functions

To make specifying extended sensors or sources faster, k-Wave provides some helper functions.

2.5.1 `make_circle`

Because the grid is square and focused ultrasound transducers are usually curved, k-Wave includes the `make_circle` function to set the `source.p_mask` such that an arc is formed

[†]This variable name can be arbitrary, but the documentation and these notes will always call it `sensor_data`.

while keeping the density of sources as constant as possible. Focused ultrasound transducers are often characterized by their F-number: $F_\# \equiv$ (Focal Distance)/(Aperture). However, the `make_circle` requires the angle of the arc. To convert, call the focal distance d and the aperture a, so that $F_\# = d/a$. Then half aperture is given by $a/2 = d \sin\theta$, so that

$$\sin\theta = \frac{a}{2d} \quad \longrightarrow \quad \theta = \arcsin\frac{1}{2F_\#} \,. \tag{19}$$

Then we can create a transducer with a 3 cm focal distance and F-number of 1 as follows (assuming we have already defined the grid, see Sec. 2.1[†]).

```
% Define FUS transducer properties
xFocus = 0E-3; % [m]
yFocus = 0E-3; % [m]
fNumber = 1;
focalDistance = 3E-2; % [m]

% Find indices closest to desired focal point
[~, xInd] = min( abs( xFocus - kgrid.x_vec ) );
[~, yInd] = min( abs( yFocus - kgrid.y_vec ) );

% Convert radius to number of points
radius_inds = focalDistance./kgrid.dx; % Assumes square grid

% Calculate angle
theta = asin( 1./( 2.*fNumber ) ); % [rad]

% Get circle and assign it as the source
circle = makeCircle( kgrid.Nx, kgrid.Ny, xInd, yInd, radius_inds, theta);
source.p_mask = circle;
```

2.5.2 Interpolating the Field

Suppose we have a map of sound speed and density, but the resolution is higher than needed. We can use interpolation to adjust the size of the grid. Suppose the sound speed field $c = c(x, y)$ is stored in a variable called `soundSpeed`, with grid spacings `dx` and `dy`. Then, to decrease the resolution by 30% (for example), we would have

```
% Get size of field
[Nx, Ny] = size( soundSpeed );
resizeFactor = 0.7;
```

[†]Note that the `make_circle` function makes a circle on the grid, regardless of the dimensions. So only use this function if your computation grid is square, i.e., if $\Delta x = \Delta y$.

```

% Get interpolation vectors
[X, Y] = meshgrid( ...
     dx*[ 0 : Nx - 1], ...
     dy*[ 0 : Ny - 1] ...
     );
[XI, YI] = meshgrid( ...
     (resizeFactor.*dx).*[ 0 : resizeFactor.*(Nx - 1) ], ...
     (resizeFactor.*dy).*[ 0 : resizeFactor.*(Ny - 1) ] ...
     );

% Interpolate
soundSpeedInterp = interp2( X, Y, soundSpeed, XI, YI, 'linear');
```

2.5.3 Padding the Array

Suppose the material property matrices we have (e.g., those imported from a CT scan) don't have enough room to place our `sources` or `sensors` right where we want them in the simulation. In this case, it might be useful to pad the domain with the edge values or water values to give some more flexibility.

```
% Set how much space to add in each dimension
xExtra = 5E-3; % [m]
yExtra = 5E-3; % [m]

% Determine number of pixels to add
padNx = round( xExtra./kgrid.dx );
padNy = round( yExtra./kgrid.dy );

% Pad with water values
cWater = 1500; % [m/s]

% Add in both directions
c = medium.sound_speed;
cNew = padarray( c, [padNx, padNy], 'replicate', 'both' );
```

Arrays may be padded in the x or y direction only (e.g., set `xExtra` to 0), or only in one direction by using the `'pre'` or `'post'` options. Check the MATLAB documentation for the `padarray` function for full details.

3 Simulation

Once the grid, medium, source, and sensors are defined, running acoustic and thermal simulations is straightforward.

3.1 Acoustic Simulation

3.1.1 Options

There are a few options that can be specified as name-argument pairs. Reasonable defaults are used, so it is not necessary to specify these on your first pass. However, there are several recommended ones, of which brief description and the default (or recommended) value is given below.

- **`'DataCast' ('single')`** This specifies the precision of the values stored at each time step. By using single (rather than double) precision, the simulations are faster due to efficiencies in the FFT implementation.

- **`'DisplayMask' ('false')`** The position of the sensors may be shown during the simulation if this field is set to true.

- **`'PlotSim' ('true')`** This flag determines whether or not the simulation is plotted as it is run. Recommended for the first time, to ensure that the geometry is right and that an appropriate grid size was chosen (numerical instability will often cause obviously divergent field values). However, it causes a modest slowdown in the computation time, so recommended to set to `false` if many simulations are required.

- **`'PMLSize' (10)`** This is the size of the perfectly matched layer (PML) used in the simulation. This is an artificial layer in the computational domain designed to minimize reflections as the wave propagates to the edge of the simulated area. More points will provide better absorption at the boundary, but will in turn increase the grid size and the computational cost.

3.1.2 Running the Simulation

Once all parameters and options have been set, running the simulation is achieved with a call of `kSpaceFirstOrder2D`. Assuming the `kgrid`, `medium`, `source`, and `sensor` structures have been defined as discussed in Sec. 2, we can run the simulation as follows:

```
% Set simulation options
simOptions = { ...
  'DataCast', 'single', ...
  'DisplayMask', false, ...
  'PMLSize', 10, ...
  'PlotSim', true ...
  };

% Run simulation
sensor_data = kspaceFirstOrder2D(kgrid, medium, source, sensor, simOptions);
```

3.2 Thermal

The thermal simulation is performed in through the `kWaveDiffusion` object. Once the `kgrid`, `medium`, `source`, and `sensor` structures have been defined as appropriate (including the thermal properties), the `kdiff` object may be created and then the heat diffusion simulated for the desired number of time steps:

```
% Create kWaveDiffusion object
kdiff = kWaveDiffusion(kgrid, medium, source, sensor);

% Run simulation for 2 min
tFinal = 120; % [s]
Nt = 100; %
dt = round( tFinal./Nt );
kdiff.takeTimeStep(Nt, dt)

% Plot results
kdiff.plotTemp;
```

References

[1] Bradley E. Treeby and Benjamin T. Cox. k-Wave: MATLAB toolbox for the simulation and reconstruction of photoacoustic wave fields. *Journal of Biomedical Optics*, 15(2):021314, March 2010.

[2] Bradley E. Treeby and B. T. Cox. Modeling power law absorption and dispersion for acoustic propagation using the fractional Laplacian. *The Journal of the Acoustical Society of America*, 127(5):2741–2748, May 2010.

[3] Bradley E. Treeby, Jiri Jaros, Alistair P. Rendell, and B. T. Cox. Modeling nonlinear ultrasound propagation in heterogeneous media with power law absorption using a k-space pseudospectral method. *The Journal of the Acoustical Society of America*, 131(6):4324–4336, June 2012.

[4] Qing Huo Liu. The pseudospectral time-domain (PSTD) algorithm for acoustic waves in absorptive media. *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, 45(4):1044–1055, July 1998.