FOCUS Function List

Michigan State University

November 18, 2013

asa_call

Description

Notice: This function is deprecated. Please use cw_angular_spectrum instead.

This function is a gateway to the C++ function that calculates the 3D pressure using the Angular Spectrum Approach.

Usage

```
pressure = asa_call(p0, z0, z, medium, nfft, delta, type, f0);
```

Arguments

- p0: matrix of size [nx,ny], input pressure or velocity source.
- z0: scalar, location of the source pressure plane.
- z: vector, location of destination planes.
- medium: The medium in which this calculation will occur.
- nfft: FFT grid number. Must be greater than the largest dimensions of the input pressure matrix. See notes for details on choosing this value.
- delta: scalar, spatial sampling interval in meters.
- type: Selection for different choices of ASA method; default is 'Pa'.
  - 'P': Spectral propagator and pressure source without angular restriction.
  - 'Pa': Spectral propagator and pressure source with angular restriction. (This is the default)
  - 'V': Spectral propagator and velocity source without angular restriction.
  - 'Va': Spectral propagator and velocity source with angular restriction.
  - 'p': Spatial propagator and pressure source without angular restriction.
  - 'v': Spatial propagator and velocity source without angular restriction.
- f0: Excitation frequency of the source wave.

Output Parameters

- fftpress: matrix of size [nx ny nz], calculated pressure.

Notes

To choose the number of FFT terms to use, the simplest method is to zero-pad the source pressure and increase the number of terms until an acceptable error is reached. Numbers of terms that are powers of two (e.g. 512, 1024) usually result in the fastest calculations.

asa_run

Description

Notice: This function is deprecated. Please use cw_angular_spectrum instead.

Identical to asa_call but with more error checking and some sanity checking to ensure a proper calculation will occur.

Usage

```
Pressure = asa_run('source', p0, 'z0', z0, 'z', z, 'medium', medium,\
'nfft', nfft, 'delta', delta, 'type', type, 'f0', f0);
Pressure = asa_run();
```

Arguments

- p0: Matrix of size [nx,ny] describing the intial pressure or velocity field.
- z0: The location of the source pressure plane.
- z: Vector describing location of destination planes.
- medium: The medium in which this calculation will occur. Default is water; see define_media for more information.
- nfft: FFT grid number. Default is $2^{2*length(source)}$.
- delta: Spatial sampling interval in meters.
- type: Selection for different choices of ASA method; default is 'Pa'.
  - 'P': Spectral propagator and pressure source without angular restriction.
  - 'Pa': Spectral propagator and pressure source with angular restriction.
  - 'V': Spectral propagator and velocity source without angular restriction.
  - 'Va': Spectral propagator and velocity source with angular restriction.
  - 'p': Spatial propagator and pressure source without angular restriction.
  - 'v': Spatial propagator and velocity source without angular restriction.
- f0: Excitation frequency of the source wave.

Output Parameters

- fftpress: matrix of size [nx ny nz], calculated pressure.

Notes

This function is called by passing an argument name then the argument value. For example, passing argument delta with a value of 1e-5 would look like asa_run(...,'delta',1e-5,...). This function requires at least the source plane passed. Any missing arguments will cause the program to use the default values or prompt the user for the values to use.

Calling this function with no arguments will cause the program to prompt the user for each required value.

bioheat_transfer

Description

This function uses the Bioheat Transfer Equation to calculate the temperature rise at each point in a medium.

Usage

```
temperature_rise = bioheat_transfer(input_pressure, medium, dx, dy, dz,\
iterations);
```

Arguments

- input_pressure: A 3D input pressure field with the pressure at each point in Pa
- medium: A FOCUS medium struct
- dx: The distance between adjacent points in x in m
- dy: The distance between adjacent points in y in m
- dz: The distance between adjacent points in z in m
- iterations: The number of iterations to use in the calculations

Output Parameters

- temperature_rise: The temperature rise at each point in the original pressure field in degrees C.

Notes

Note that the number of iterations required for a stable temperature field will depend on the spatial step size. The number of iterations should be increased until successive results do not significantly differ to make sure that the result is correct.

compare_transient_pressures

Description

This function calculates the peak error between two transient pressure fields.

Usage

```
pressure_error = compare_transient_pressures(reference_field, comparison_field);
```

Arguments

- reference_field: The transient pressure field to use as the reference.
- comparison_field: The transient pressure field to compare to the reference field.

Output Parameters

- pressure_error: The maximum normalized error in the comparison pressure field relative to the reference.

Notes

The error between the two pressure fields is calculated by taking the difference between the two fields at each point in space and then dividing these values by the peak pressure in the reference field. The largest of these values is returned.

create_circ_csa

Description

This function creates a cylindrical section array of circular transducers.

Usage

```
transducer = create_circ_csa(nx, ny, radius, kerf_x, kerf_y, r_curv);
transducer = create_circ_csa(nx, ny, radius, kerf_x, kerf_y, r_curv, override);
```

Arguments

- nx: Number of elements in the x direction.
- ny: Number of elements in the y direction.
- radius: Radius each element in m. All elements in the array will be the same size.
- kerf_x: Kerf (edge-to-edge spacing) in the x direction.
- kerf_y: Kerf (edge-to-edge spacing) in the y direction.
- r_curv: Radius of curvature of the apparatus.
- override: Omit to allow error checking, any value to bypass error checking.

Output Parameters

- transducer: A 2-d array of transducer structs. The first element is at transducer(1,1) and the last element is at transducer(nx,ny).

Notes

The curve of cylinder is on the x axis, with elements rotating about the y axis. The center of the array is defined to be the center of the element anchoring the array. All coordinates are expressed in meters.

Transducer arrays used one-dimensional indexing prior to FOCUS version 0.332. One-dimensional indexing is still possible, though the indices may not match those used in older versions of FOCUS.

Calling this function with no arguments will cause the program to prompt the user for each required value.

create_circ_planar_array

Description

This function creates a planar array of circular transducers.

Usage

```
transducer = create_circ_planar_array(nx, ny, radius, kerf_x, kerf_y);
transducer = create_circ_planar_array(nx, ny, radius, kerf_x, kerf_y, center);
transducer = create_circ_planar_array(nx, ny, radius, kerf_x, kerf_y, center,\
override);
```

Arguments

- nx: Number of elements in the x direction.
- ny: Number of elements in the y direction.
- radius: Radius each element in m. All elements in the array will be the same size.
- kerf_x: Kerf (edge-to-edge spacing) in the x direction.
- kerf_y: Kerf (edge-to-edge spacing) in the y direction.
- center: Three element array that is the coordinate of the center of the array. This argument is optional.
- override: Omit to allow error checking, any value to bypass error checking.

Output Parameters

- transducer: An array of transducer structs.

Notes

The center of the array is defined to be the geometric center of the rectangle that bounds the array. All coordinates are expressed in meters.

Transducer arrays used one-dimensional indexing prior to FOCUS version 0.332. One-dimensional indexing is still possible, though the indices may not match those used in older versions of FOCUS.

Calling this function with no arguments will cause the program to prompt the user for each required value.

create_circ_ssa

Description

This function creates a spherical section array of circular transducers.

Usage

```
transducer = create_circ_ssa(radius, R, nrow, ang_open);
```

Arguments

- radius: Radius each element in m. All elements in the array will be the same size.
- R: Radius of curvature of the array in m.
- nrow: Number of elements in each direction.
- ang_open: Spread of the array in each direction in radians.

Output Parameters

- transducer: An array of transducer structs.

Notes

The array is defined such that the coordinate [0 0 0] corresponds to the center of the anchor element of the array.

create_concentric_ring_array

Description

Creates an array of concentric ring transducers.

Usage

```
transducer = create_concentric_ring_array(ring_count, ring_width, kerf);
transducer = create_concentric_ring_array(ring_count, inner_radii, outer_radii);
```

Arguments

- ring_count: The number of rings in the array.
- ring_width: The with of all rings in meters.
- kerf: The edge-to-edge spacing of the rings in meters.
- inner_radii: A vector of values specifying the inner radius (in meters) of each element in the array, starting with the center element and ending with the outer element.
- outer_radii: A vector of values specifying the outer radius (in meters) of each element in the array, starting with the center element and ending with the outer element.

Output Parameters

- transducer: An array of transducer structs.

Notes

Ring transducer arrays always use one-dimensional indexing starting with the innermost element.

create_rect_csa

Description

This function creates a cylindrical section array of rectangular transducers.

Usage

```
transducer = create_rect_csa(nx, ny, width, height, kerf_x, kerf_y, r_curv);
transducer = create_rect_csa(nx, ny, width, height, kerf_x, kerf_y, r_curv,\
override);
```

Arguments:

- nx: Number of elements in the x direction.
- ny: Number of elements in the y direction.
- width: Width of a single element in meters, all elements in the array are the same size.
- height: Height of a single element in meters, all elements in the array are the same size.
- kerf_x: Kerf (edge-to-edge spacing) in the x direction.
- kerf_y: Kerf (edge-to-edge spacing) in the y direction.
- r_curv: Radius of curvature of the array.
- override: Omit to allow error checking, any value to bypass error checking.

Output Parameters

- transducer: An array of transducer structs.

Notes

The curve of cylinder is on the X axis, with elements rotating about the Y axis. The center of the array is defined to be the center of the element anchoring the array. All coordinates are expressed in meters.

Transducer arrays used one-dimensional indexing prior to FOCUS version 0.332. One-dimensional indexing is still possible, though the indices may not match those used in older versions of FOCUS.

Calling this function with no arguments will cause the program to prompt the user for each required value.

create_rect_curved_strip_array

Description

This function creates a cylindrical section array of rectangular transducers.

Usage

```
transducer = create_rect_curved_strip_array(nx, ny, width, height, kerf, r_curv);
```

Arguments:

- nx: Number of elements in the x direction.
- ny: Number of subelements in the y direction.
- width: Width of a single element in meters, all elements in the array are the same size.
- height: Height of a single element in meters, all elements in the array are the same size. This is the total height of each curved element, so the height of each subelement will be height/ny.
- kerf: Kerf (edge-to-edge spacing) in the x direction.
- r_curv: Radius of curvature of the array.

Output Parameters

- transducer: An array of transducer structs.

Notes

The curve of cylinder is on the y axis, with elements rotated about the x axis. The center of the array is defined to be the center of the element anchoring the array. All coordinates are expressed in meters.

Transducer arrays used one-dimensional indexing prior to FOCUS version 0.332. One-dimensional indexing is still possible, though the indices may not match those used in older versions of FOCUS.

This function will use individual rectangular elements to model the curved array elements. More subelements will result in better curvature but will increase the calculation time.

create_rect_enclosed_csa

Description

This function creates a fully-enclosed cylindrical section array of rectangular transducers.

Usage

```
transducer = create_rect_enclosed_csa(nx, ny, width, height, kerf_y, r_curv);
transducer = create_rect_enclosed_csa(nx, ny, width, height, kerf_y, r_curv);
```

Arguments:

- nx: Number of elements in the x direction.
- ny: Number of elements in the y direction.
- width: Width of a single element in meters, all elements in the array are the same size.
- height: Height of a single element in meters, all elements in the array are the same size.
- kerf_y: Kerf (edge-to-edge spacing) in the y direction.
- r_curv: Radius of curvature of the array.
- override: Omit to allow error checking, any value to bypass error checking.

Output Parameters

- transducer: An array of transducer structs.

Notes

The array generated will be a cylinder with its center along the y-axis. The elements will be located along the edge of the cylinder in series of rings in x and z.

create_rect_planar_array

Description

This function creates a planar array of rectangular transducers.

Usage

```
transducer = create_rect_planar_array(nx, ny, width, height, kerf_x, kerf_y);
transducer = create_rect_planar_array(nx, ny, width, height, kerf_x, kerf_y,\
center);
transducer = create_rect_planar_array(nx, ny, width, height, kerf_x, kerf_y,\
center, override);
```

Arguments

- nx: Number of elements in the x direction.
- ny: Number of elements in the y direction.
- width: Width of a single element in meters, all elements in the array are the same size.
- height: Height of a single element in meters, all elements in the array are the same size.
- kerf_x: Kerf (edge-to-edge spacing) in the x direction.
- kerf_y: Kerf (edge-to-edge spacing) in the y direction.
- center: Three element array describing the coordinates of the center of the array.
- override: Omit to allow error checking, any value to bypass error checking.

Output Parameters

- transducer: An array of transducer structs.

Notes

The center of the array is defined to be the geometric center of the rectangle that bounds the array. All coordinates are expressed in meters.

Transducer arrays used one-dimensional indexing prior to FOCUS version 0.332. One-dimensional indexing is still possible, though the indices may not match those used in older versions of FOCUS.

Calling this function with no arguments will cause the program to prompt the user for each required value.

create_rect_ssa

Description

Creates a spherical section array of rectangular transducers.

Usage

```
transducer = create_rect_ssa(width, height, R, nrow, ang_open);
```

Arguments

- width: Width of a single element in meters, all elements in the array are the same size.
- height: Height of a single element in meters, all elements in the array are the same size.
- R: Radius of curvature of the array in m.
- nrow: Number of elements in each direction.
- ang_open: Spread of the array in each direction in radians.

Output Parameters

- transducer: An array of transducer structs.

Notes

The array is defined such that the coordinate [0 0 0] corresponds to the center of the anchor element of the array.

create_spherical_shell_planar_array

Description

Creates a planar array of spherical shell transducers.

Usage

```
transducer = create_spherical_shell_planar_array(nx, ny, a, r, kerf_x, kerf_y);
transducer = create_spherical_shell_planar_array(nx, ny, a, r, kerf_x, kerf_y,\
center);
transducer = create_spherical_shell_planar_array(nx, ny, a, r, kerf_x, kerf_y,\
center, override);
```

Arguments

- nx: Number of elements in the x direction.
- ny: Number of elements in the y direction.
- a: Radius of the elements in meters.
- r: Radius of curvature of the elements in meters.
- kerf_x: Kerf (edge-to-edge spacing) in the x direction.
- kerf_y: Kerf (edge-to-edge spacing) in the y direction.
- center: Three element array describing the coordinates of the center of the array.
- override: Omit to allow error checking, any value to bypass error checking.

Output Parameters

- transducer: An array of transducer structs.

Notes

The center of the array is defined to be the geometric center of the rectangle that bounds the array. All coordinates are expressed in meters.

Transducer arrays used one-dimensional indexing prior to FOCUS version 0.332. One-dimensional indexing is still possible, though the indices may not match those used in older versions of FOCUS.

Calling this function with no arguments will cause the program to prompt the user for each required value.

create_spherically_focused_ring_array

Description

Creates an array of spherically focused ring transducers.

Usage

```
transducer = create_spherically_focused_ring_array(ring_width, kerf, ring_count,\
geometric_focus);
```

Arguments

- ring_width: The with of each ring in meters.
- kerf: The spacing between the rings in meters.
- ring_count: The number of rings in the array.
- geometric_focus: The geometric focus of the array elements.

Output Parameters

- transducer: An array of transducer structs.

Notes

Transducer arrays used one-dimensional indexing prior to FOCUS version 0.332. One-dimensional indexing is still possible, though the indices may not match those used in older versions of FOCUS.

cw_angular_spectrum

Description

This function uses the Angular Spectrum Approach to quickly calculate 3D continuous-wave pressures given an intial pressure field.

Usage

```
pressure = cw_angular_spectrum(p0, coord_grid, medium, f0, nfft);
pressure = cw_angular_spectrum(p0, coord_grid, medium, f0, nfft, type);
```

Arguments

- p0: matrix of size [nx,ny], input pressure (in Pa) or velocity source (in m/s).
- coord_grid: A FOCUS coordinate grid describing the dimensions of the output pressure field.
- medium: The medium in which this calculation will occur.
- f0: Excitation frequency of the source wave in Hz.
- nfft: FFT grid number. Must be greater than the largest dimensions of the input pressure matrix. See notes for details on choosing this value.
- type: Selection for different choices of ASA method; default is 'Pa'.
  - 'P': Spectral propagator and pressure source without angular restriction.
  - 'Pa': Spectral propagator and pressure source with angular restriction. (This is the default)
  - 'V': Spectral propagator and velocity source without angular restriction.
  - 'Va': Spectral propagator and velocity source with angular restriction.
  - 'p': Spatial propagator and pressure source without angular restriction.
  - 'v': Spatial propagator and velocity source without angular restriction.

Output Parameters

- pressure: matrix of size [nx ny nz], calculated pressure.

Notes

To choose the number of FFT terms to use, the simplest method is to zero-pad the source pressure and increase the number of terms until an acceptable error is reached. Numbers of terms that are powers of two (e.g. 512, 1024) usually result in the fastest calculations.

cw_intensity

Description

This function is used to calculate the average intensity of a continuous-wave pressure field.

Usage

```
intensity = cw_intensity(pressure, medium);
intensity = cw_intensity(transducer, cg, medium, ndiv, f0);
```

Arguments

- pressure: The CW pressure at each point in space in Pa.
- transducer: A FOCUS transducer array.
- cg: A FOCUS coordinate grid.
- medium: A FOCUS medium.
- ndiv: The number of integral points to use.
- f0: Frequency of the array in Hz.

Output Parameters

- intensity: The average intensity in $\mathrm{W/m^2}$at each point in space given by the formula $I = \frac{|p|^2}{2\rho c_0}$.

cw_power

Description

This function is used to calculate the power of a continuous-wave pressure field.

Usage

```
power = cw_power(pressure, medium);
power = cw_power(transducer, cg, medium, ndiv, f0);
```

Arguments

- pressure: The CW pressure at each point in space in Pa.
- transducer: A FOCUS transducer array.
- cg: A FOCUS coordinate grid.
- medium: A FOCUS medium.
- ndiv: The number of integral points to use.
- f0: Frequency of the array in Hz.

Output Parameters

- power: The power in Wat each point in space given by the formula $P = \frac{|p|^2 \alpha}{2 \rho c_0}$.

cw_pressure

Description

This function is used to calculate continuous-wave pressures with the Fast Nearfield Method in FOCUS.

Usage

```
pressure = cw_pressure(transducer, cg, medium, ndiv, f0);
pressure = cw_pressure(transducer, cg, medium, ndiv, f0, method);
```

Arguments

- transducer: A FOCUS transducer array.
- cg: A FOCUS coordinate grid.
- medium: A FOCUS medium.
- ndiv: The number of integral points to use.
- f0: Frequency of the array in Hz.
- method: The method to use when calculating the pressure. If the string 'sse' is present, SSE instructions will be used to speed up the calculation where possible - see the documentation for fnm_cw_sse for details. Valid calculation methods are:
  - 'fnm': Use the Fast Nearfield Method (fnm_cw) to calculate the pressure - this is the default.
  - 'farfield': Use the farfield approximation (farfield_cw) to calculate the pressure.
  - 'rayleigh': Use the Rayleigh-Sommerfeld Integral (rayleigh_cw) to calculate the pressure.

Output Parameters

- pressure: A 3-d array representing the complex pressure at each point in space.

Notes

Advanced users may wish to use fnm_cw for access to more features and options.

The value of the ndiv parameter should be chosen based on the spatial location of the simulation relative to the transducer and the desired calculation accuracy. The only way to determine the optimal value for this parameter is to perform the calculation with a very large ndiv (e.g. 200) and then compare calculations performed with progressively larger ndiv values to the reference pressure until the desired accuracy is achieved. See the FNMcirc example (in /Examples/Papers) for an example of this type of calculation.

A good approximation for the ndiv value for single transducers can be calculated by the formula $n = 2 \times \frac{d}{\lambda}$, where d is the transducer width and λ is the wavelength.

SSE calculations are about four times as fast as standard calculations, but because they use single precision values rather than double precision, the maximum accuracy is on the order of 10^{-4}.

define_media

Description

Generates pre-set media usable for any calculation.

Usage

```
define_media();
```

Arguments

- None

Output Parameters

This function has no output, but adds the following medium structs to the workspace:

- lossless
  - soundspeed: 1.500×10^3 m/s
  - attenuationdBcmMHz: 0 dB/cm/MHz
  - density: 1.000×10^3 kg/m^3
  - specificheatofblood: 3.480×10^3 J/kg/K
  - bloodperfusion: 0 kg/m^3/s
  - powerlawexponent: 0
  - specificheat: 4.180×10^3 J/kg/K
  - thermalconductivity: 6.150×10^{-1} W/m/K
  - nonlinearityparameter: 0
- attenuated
  - soundspeed: 1.500×10^3 m/s
  - attenuationdBcmMHz: 1 dB/cm/MHz
  - density: 1.000×10^3 kg/m^3
  - specificheatofblood: 3.480×10^3 J/kg/K
  - bloodperfusion: 0 kg/m^3/s
  - powerlawexponent: 0
  - specificheat: 4.180×10^3 J/kg/K
  - thermalconductivity: 6.150×10^{-1} W/m/K
  - nonlinearityparameter: 0
- water
  - soundspeed: 1.500×10^3 m/s
  - attenuationdBcmMHz: 2.500×10^{-4} dB/cm/MHz
  - density: 1.000×10^3 kg/m^3

  - specificheatofblood: 3.480×10^{3} J/kg/K
  - bloodperfusion: 0 kg/m^{3}/s
  - powerlawexponent: 0
  - specificheat: 4.180×10^{3} J/kg/K
  - thermalconductivity: 6.150×10^{-1} W/m/K
  - nonlinearityparameter: 0
- skin
  - soundspeed: 1.498×10^{3} m/s
  - attenuationdBcmMHz: 1.400×10^{-1} dB/cm/MHz
  - density: 1.200×10^{3} kg/m^{3}
  - specificheatofblood: 3.480×10^{3} J/kg/K
  - bloodperfusion: 5 kg/m^{3}/s
  - powerlawexponent: 0
  - specificheat: 3.430×10^{3} J/kg/K
  - thermalconductivity: 2.660×10^{-1} W/m/K
  - nonlinearityparameter: 0
- fat
  - soundspeed: 1.445×10^{3} m/s
  - attenuationdBcmMHz: 7.000×10^{-2} dB/cm/MHz
  - density: 9.210×10^{2} kg/m^{3}
  - specificheatofblood: 3.480×10^{3} J/kg/K
  - bloodperfusion: 5 kg/m^{3}/s
  - powerlawexponent: 0
  - specificheat: 2.325×10^{3} J/kg/K
  - thermalconductivity: 2.230×10^{-1} W/m/K
  - nonlinearityparameter: 0
- muscle
  - soundspeed: 1.569×10^{3} m/s
  - attenuationdBcmMHz: 8.000×10^{-2} dB/cm/MHz
  - density: 1.138×10^{3} kg/m^{3}
  - specificheatofblood: 3.480×10^{3} J/kg/K
  - bloodperfusion: 5 kg/m^{3}/s
  - powerlawexponent: 0
  - specificheat: 3.720×10^{3} J/kg/K
  - thermalconductivity: 4.975×10^{-1} W/m/K
  - nonlinearityparameter: 0
- liver
  - soundspeed: 1.540×10^{3} m/s
  - attenuationdBcmMHz: 3.200×10^{-2} dB/cm/MHz

- density: 1.060×10^3 kg/m^3
- specificheatofblood: 3.480×10^3 J/kg/K
- bloodperfusion: 5 kg/m^3/s
- powerlawexponent: 0
- specificheat: 3.600×10^3 J/kg/K
- thermalconductivity: 5.120×10^{-1} W/m/K
- nonlinearityparameter: 0

Notes

If a variable called f0 does not exist in the MATLAB workspace, this function will automatically create it and set it to 1 MHz.

For details on the nature of the medium structures, see the documentation for set_medium.

draw_array

Description

This function creates a 3D representation of an arbitrary array of FOCUS transducers.

Usage

```
draw_array(transducer_array);
draw_array(transducer_array, input_color);
draw_array(transducer_array, input_color, new_figure);
draw_array(transducer_array, input_color, new_figure, coordinate_grid);
```

Arguments

- transducer_array: A FOCUS transducer array.
- input_color: A color matrix (of the form [R G B], where R, G, and B are between 0 and 1) or color string (e.g. 'blue') to use for the array elements. This argument can also be an array of the same size as the transducer array. In this case, one color is defined for each element.
- new_figure: Create a new MATLAB figure to draw the array in rather than drawing over the old one.
- coordinate_grid: A FOCUS coordinate grid. If this argument is present, the coordinate grid will be represented by a cube on the output plot.

Output Parameters

- None

Notes

This function returns nothing; it plots the given transducer array in a new MATLAB figure.

draw_circ

Description

This is a helper function to draw the circular transducers used by the draw_array function.

Usage

```
draw_circ(transducer, color);
```

Arguments

- transducer: The transducer object to draw.
- color: A MATLAB color struct describing the desired color of the transducer.

Notes

Users are not expected to use this function. Instead, draw_array should always be used, even for single transducers.

draw_rect

Description

This is a helper function to draw the rectangular transducers used by the draw_array function.

Usage

```
draw_rect(transducer, color);
```

Arguments

- transducer: The transducer object to draw.
- color: A MATLAB color struct describing the desired color of the transducer.

Notes

Users are not expected to use this function. Instead, draw_array should always be used, even for single transducers.

draw_ring

Description

This is a helper function to draw the planar ring transducers used by the draw_array function.

Usage

```
draw_ring(transducer, color);
```

Arguments

- transducer: The transducer object to draw.
- color: A MATLAB color struct describing the desired color of the transducer.

Notes

Users are not expected to use this function. Instead, draw_array should always be used, even for single transducers.

draw_spherically_focused_ring

Description

This is a helper function to draw the spherically focused ring transducers used by the draw_array function.

Usage

```
draw_spherically_focused_ring(transducer, color);
```

Arguments

- transducer: The transducer object to draw.
- color: A MATLAB color struct describing the desired color of the transducer.

Notes

Users are not expected to use this function. Instead, draw_array should always be used, even for single transducers.

draw_sphericalshell

Description

This is a helper function to draw a single spherical shell transducer used by the draw_array function.

Usage

```
draw_sphericalshell(transducer, color);
```

Arguments

- transducer: The transducer object to draw.
- color: A MATLAB color struct describing the desired color of the transducer.

Notes

Users are not expected to use this function. Instead, draw_array should always be used, even for single transducers.

farfield_cw

Description

Calculates the farfield continuous-wave pressure generated by a transducer array.

Usage

```
pressure = farfield_cw(transducer_array, coordinate_grid, medium, ndiv, f0);
pressure = farfield_cw(transducer_array, coordinate_grid, medium, ndiv, f0, disp_flag);
pressure = farfield_cw(transducer_array, coordinate_grid, medium, ndiv, f0, disp_flag,\
nthreads);
```

Arguments

- transducer_array: A transducer_array.
- coordinate_grid: A coordinate grid struct like the ones created by set_coordinate_grid.
- medium: A medium struct like the ones created by set_medium.
- ndiv: The number of integral points to use.
- f0: The frequency of the transducer array.
- disp_flag: Display flag, 1 = display, 0 = suppress.
- nthreads: The number of threads to use in the calculation.

Output Parameters

- pressure: A 3-d array representing the complex pressure at each point in space.

Notes

This function is only implemented for rectangular transducers.

Threading in FOCUS is implemented at the transducer level, meaning that single transducers will not benefit from this feature.

Specifying a number of threads larger than the number of CPU cores available will not result in a significant additional speed increase and may in fact result in slower speeds due to additional inter-thread communication overhead.

farfield_cw_parfor

Description

Calculates the farfield pressure field generated by a transducer array.

Usage

```
pressure = farfield_cw_parfor(transducer_array, coordinate_grid, medium, ndiv,\
f0, disp_flag);
```

Arguments

- transducer_array: A transducer_array.
- coordinate_grid: A coordinate grid struct like the ones created by set_coordinate_grid.
- medium: A medium struct like the ones created by set_medium.
- ndiv: The number of integral points to use.
- f0: The frequency of the transducer array.
- disp_flag: Display flag, 1 = display, 0 = suppress.

Output Parameters

- pressure: A 3-d array representing the complex pressure at each point in space.

Notes

This function uses the MATLAB Parallel Computing Toolkit to perform the pressure calculation using multiple threads. The speed increase realized by this function is entirely dependent on the number of processor cores present in the computer executing the code. If the Parallel Computing Toolkit is not installed, the function will fail to run.

fftw_asa

Description

This function is the C++ binary that implements the Angular Spectrum Approach. It should not be used in any scripts or used directly, as the interface can change. Use asa_call to ensure consistency. Documentation has been omitted. This is a mex function, if it doesn't run please e-mail focus@egr.msu.edu for help.

find_phase

Description

Notice: This function is deprecated. Please use find_single_focus_phase instead.

This function finds a phase adjustment for each transducer such that the phase of all transducers at a given point is the same. This function will likely undergo some significant changes in the near future, but the overall form should remain the same.

Usage

```
transducer = find_phase(transducer, x, y, z, medium, f0);
transducer = find_phase(transducer, x, y, z, medium, f0, tol);
```

Arguments

- transducer: A FOCUS transducer array.
- x: x coordinate of the focus.
- y: y coordinate of the focus.
- z: z coordinate of the focus.
- medium: A FOCUS medium.
- f0: The frequency of the array in Hz.
- tol: Optional tolerance value.

Output Parameters

- transducer: A FOCUS transducer array with the complex weights adjusted to reflect the new focus.

Notes

Currently the program finds the phase of each transducer at a given point. Once that has been established, each transducer is corrected such that the phase at the given point is 0.

find_multiple_focus_phase

Description

Calculates phase shifts to focus a transducer array at the given points.

Usage

```
transducer = find_multiple_focus_phase(transducer, x, y, z, medium, f0);
transducer = find_multiple_focus_phase(transducer, x, y, z, medium, f0, ndiv);
transducer = find_multiple_focus_phase(transducer, x, y, z, medium, f0, ndiv,\
linear_array);
```

Arguments

- transducer: A FOCUS transducer array.
- x: List of x coordinates of the foci.
- y: List of y coordinates of the foci.
- z: List of z coordinates of the foci.
- meduim: A FOCUS medium struct.
- f0: Center frequency of the array in Hz.
- ndiv: Number of abscissas used for calculations. Default value is 200.
- linear_array: set to 1 if the array is linear in x and each group of elements along the x-axis should have the same phase. If this parameter is not specified, the function will generate a warning and attempt to determine the geometry of the array.

Output Parameters

- transducer: Transducer array with complex weights set to focus it at the given points.

Notes

This function alters the transducer struct, the output transducer should be the same as the input transducer. The logic is that if all elements have a complex_weight of 0 at the target point, the intensity at that point is the highest. Each focus is defined such that focus(i) = [x(i) y(i) z(i)].

find_ndiv

Description

This function calculates the number of Gaussian quadratures needed to reach an arbitrary tolerance.

Usage

```
ndiv = find_ndiv(transducer, coordinate_grid, medium, f0);
ndiv = find_ndiv(transducer, coordinate_grid, medium, f0, tol);
```

Arguments

- transducer: A FOCUS transducer array.
- coordinate_grid: A FOCUS coordinate grid.
- medium: A FOCUS medium.
- f0: The frequency of the array in Hz.
- tol: Maximum difference in the result between two ndiv calculations. This parameter is optional; its default value is 1×10^{-10}.

Output Parameters

- ndiv: The number of divisions.

Notes

This function is mainly used by fnm_run when users do not enter the ndiv value. If the desired tolerance cannot be reached with 100 quadratures, the number will be set to 20.

find_single_focus_phase

Description

Calculates phase shifts to focus a transducer array at a given point.

Usage

```
transducer = find_single_focus_phase(transducer, x, y, z, medium, f0);
transducer = find_single_focus_phase(transducer, x, y, z, medium, f0, ndiv);
transducer = find_single_focus_phase(transducer, x, y, z, medium, f0, ndiv, lin-
ear_array);
```

Arguments

- transducer: A FOCUS transducer array.
- x: The x coordinate of the focus.
- y: The y coordinate of the focus.
- z: The z coordinate of the focus.
- meduim: A FOCUS medium struct.
- f0: Center frequency of the array in Hz.
- ndiv: Number of abscissas used for calculations. The default value is 200.
- linear_array: set to 1 if the array is linear in x and each group of elements along the x-axis should have the same phase. If this parameter is not specified, the function will generate a warning and attempt to determine the geometry of the array.

Output Parameters

- transducer: Transducer array with complex weights set to focus it at the given point.

Notes

This function alters the transducer struct, the output transducer should be the same as the input transducer. The logic is that if all elements have a complex_weight of 0 at the target point, the intensity at the point is the highest.

fnm_call

Description

This function is the gateway between the Matlab and C++ binary. It does minimal error checking to ensure enough arguments are being passed.

Usage

```
pressure = fnm_call(transducer, cg, medium, ndiv, f0);
pressure = fnm_call(transducer, cg, medium, ndiv, f0, dflag);
pressure = fnm_call(transducer, cg, medium, ndiv, f0, dflag, nthreads);
```

Arguments

- transducer: A FOCUS transducer array.
- cg: A FOCUS coordinate grid.
- medium: A FOCUS medium.
- ndiv: The number of integral points to use.
- f0: Frequency of the array in Hz.
- dflag: Display flag, 1 = display, 0 = suppress.
- nthreads: The number of threads to use in the calculation.

Output Parameters

- pressure: A 3-d array representing the complex pressure at each point in space.

Notes

Threading in FOCUS is implemented at the transducer level, meaning that single transducers will not benefit from this feature.

Specifying a number of threads larger than the number of CPU cores available will not result in a significant additional speed increase and may in fact result in slower speeds due to additional inter-thread communication overhead.

fnm_cw

Description

This function is the C++ MEX file that performs continuous-wave calculations using the Fast Nearfield Method.

Usage

```
pressure = fnm_cw(transducer, cg, medium, ndiv, f0);
pressure = fnm_cw(transducer, cg, medium, ndiv, f0, dflag);
pressure = fnm_cw(transducer, cg, medium, ndiv, f0, dflag, nthreads);
```

Arguments

- transducer: A FOCUS transducer array.
- cg: A FOCUS coordinate grid.
- medium: A FOCUS medium.
- ndiv: The number of integral points to use.
- f0: Frequency of the array in Hz.
- dflag: Display flag, 1 = display, 0 = suppress.
- nthreads: The number of threads to use in the calculation.

Output Parameters

- pressure: A 3-d array representing the complex pressure at each point in space.

Notes

Threading in FOCUS is implemented at the transducer level, meaning that single transducers will not benefit from this feature.

Specifying a number of threads larger than the number of CPU cores available will not result in a significant additional speed increase and may in fact result in slower speeds due to additional inter-thread communication overhead.

fnm_cw_apodized

Description

Computes the pressure from pistons or arrays under continuous wave excitation in homogeneous media.

Usage

```
pressure = fnm_cw_apodized(transducer_array, coordinate_grid, medium, ndiv, f0,\
disp_flag);
pressure = fnm_cw_apodized(transducer_array, coordinate_grid, medium, ndiv, f0,\
disp_flag, nthreads);
```

Arguments

- transducer_array: A transducer array.
- coordinate_grid: A coordinate grid struct like the ones created by set_coordinate_grid.
- medium: A medium struct like the ones created by set_medium.
- ndiv: The number of integral points to use.
- f0: The frequency of the array.
- disp_flag: Display flag, 1 = display, 0 = suppress.
- nthreads: The number of threads to use in the calculation.

Output Parameters

- pressure: A 3-d array representing the complex pressure at each point in space.

Notes

Piston apodization is currently only defined for circular transducers. For details, see the documentation for get_circ().

Threading in FOCUS is implemented at the transducer level, meaning that single transducers will not benefit from this feature.

Specifying a number of threads larger than the number of CPU cores available will not result in a significant additional speed increase and may in fact result in slower speeds due to additional inter-thread communication overhead.

fnm_cw_parfor

Description

Calculate a pressure field using the Fast Nearfield Method and multple threads.

Usage

```
pressure = fnm_cw_parfor(transducer, cg, medium, ndiv, f0, dflag);
```

Arguments

- transducer: A FOCUS transducer array.
- cg: A FOCUS coordinate grid.
- medium: A FOCUS medium.
- ndiv: The number of integral points to use.
- f0: Frequency of the array in Hz.
- dflag: Display flag, 1 = display, 0 = suppress.

Output Parameters

- pressure: A 3-d array representing the complex pressure at each point in space.

Notes

This function uses the MATLAB Parallel Computing Toolkit to perform the pressure calculation using multiple threads. The speed increase realized by this function is entirely dependent on the number of processor cores present in the computer executing the code and on the number of transducers in the array. No gain will be observed for small arrays, but large arrays should see significant speed increases if enough CPU cores are present. If the Parallel Computing Toolkit is not installed, the function will fail to run.

fnm_cw_single

Description

Computes the pressure from pistons under continuous wave excitation in homogeneous media using single precision floating point numbers.

Usage

```
pressure = fnm_cw_single(transducer_array, coordinate_grid, medium, ndiv, f0,\
disp_flag);
```

Arguments

- transducer_array: A transducer.
- coordinate_grid: A coordinate grid struct like the ones created by set_coordinate_grid.
- medium: A medium struct like the ones created by set_medium.
- ndiv: The number of integral points to use.
- f0: The frequency of the array.
- disp_flag: Display flag, 1 = display, 0 = suppress.

Output Parameters

- pressure: A 3-d array representing the complex pressure at each point in space.

Notes

This function is currently only defined for rectangular transducers.

fnm_cw_sse

Description

This function is the C++ MEX file that performs continuous-wave calculations using the Fast Nearfield Method and Streaming SIMD Extensions (SSE).

Usage

```
pressure = fnm_cw_sse(transducer, cg, medium, ndiv, f0);
pressure = fnm_cw_sse(transducer, cg, medium, ndiv, f0, dflag);
pressure = fnm_cw_sse(transducer, cg, medium, ndiv, f0, dflag, nthreads);
```

Arguments

- transducer: A FOCUS transducer array.
- cg: A FOCUS coordinate grid.
- medium: A FOCUS medium.
- ndiv: The number of integral points to use.
- f0: Frequency of the array in Hz.
- dflag: Display flag, 1 = display, 0 = suppress.
- nthreads: The number of threads to use in the calculation.

Output Parameters

- pressure: A 3-d array representing the complex pressure at each point in space.

Notes

SSE calculations are about 4x as fast as standard calculations, but they are less accurate (on the order of 10^{-4}) because single precision values are used.

Threading in FOCUS is implemented at the transducer level, meaning that single transducers will not benefit from this feature.

Specifying a number of threads larger than the number of CPU cores available will not result in a significant additional speed increase and may in fact result in slower speeds due to additional inter-thread communication overhead.

fnm_run

Description

This function is a guided function that does error checking and sanity checks on the input before feeding the information to the C++ binary to get the answer.

Usage

```
pressure = fnm_run('transducer', transducer, 'cg', cg, 'medium', medium,\
'f0', f0, 'tol', tol, 'ndiv', ndiv, 'dflag', dflag);
```

Arguments

- transducer: A FOCUS transducer array.
- cg: A FOCUS coordinate grid. NOTE: If the provided delta is more than half the wavelength, this function will reset it to that value.
- medium: A FOCUS medium.
- tol: Maximum difference between one ndiv value and the next. Default is 10^{-10}.
- ndiv: The number of integral points to use. Default value is calculated based on the provided tolerance.
- f0: The frequency of the array in Hz. Default value is 1 MHz.
- dflag: Display flag, 1 = display, 0 = suppress.

Output Parameters

- pressure: A 3-d array representing the complex pressure at each point in space.

Notes

All arguments to this function must be preceeded by their name (e.g. 'f0', f0, 'medium', medium). The arguments can be in any order and they are all optional, however any critical arguments not passed to the function will cause the program to prompt the user to input them.

fnm_transient

Description

Calculates the transient pressure field from an arbitrary transducer array using the Fast Nearfield Method.

Usage

```
pressure = fnm_transient(transducer_array, coordinate_grid, medium, time_samples,\
           ndiv, excitation_function);
pressure = fnm_transient(transducer_array, coordinate_grid, medium, time_samples,\
           ndiv, excitation_function, disp_flag);
pressure = fnm_transient(transducer_array, coordinate_grid, medium, time_samples,\
           ndiv, excitation_function, disp_flag, nthreads);
pressure, start_time = fnm_transient(transducer_array, coordinate_grid, medium,\
           time_samples, ndiv, excitation_function);
pressure, start_time = fnm_transient(transducer_array, coordinate_grid, medium,\
           time_samples, ndiv, excitation_function, disp_flag);
pressure, start_time = fnm_transient(transducer_array, coordinate_grid, medium,\
           time_samples, ndiv, excitation_function, disp_flag, nthreads);
```

Arguments

- transducer_array: A FOCUS transducer array.
- coordinate_grid: A FOCUS coordinate grid.
- medium: A FOCUS medium.
- time_samples: A time samples struct created by set_time_samples.
- f0: The frequency of the array in Hz.
- ndiv: The number of integral points to use.
- excitation_function: The excitation function to use.
- disp_flag: Display flag, 1 = display, 0 = suppress.
- nthreads: The number of threads to use in the calculation.

Output Parameters

- pressure: A 4-d array representing the pressure at each point in spacetime.
- start_time: A vector containing the start time for the impulse response at each observation point.

Notes

This function only calculates pressures in lossless media, so any attenuation parameter set by the user will be ignored.

Threading in FOCUS is implemented at the transducer level, meaning that single transducers will not benefit from this feature.

Specifying a number of threads larger than the number of CPU cores available will not result in a significant additional speed increase and may in fact result in slower speeds due to additional inter-thread communication overhead.

fnm_transient_call

Description

Calculates the transient pressure field from an arbitrary transducer array using the Fast Nearfield Method.

Usage

```
pressure = fnm_transient_call(transducer_array, coordinate_grid, medium, time_samples,\
           ndiv, excitation_function);
pressure = fnm_transient_call(transducer_array, coordinate_grid, medium, time_samples,\
           ndiv, excitation_function, disp_flag);
pressure = fnm_transient_call(transducer_array, coordinate_grid, medium, time_samples,\
           ndiv, excitation_function, disp_flag, nthreads);
pressure, start_time = fnm_transient_call(transducer_array, coordinate_grid, medium,\
           time_samples, ndiv, excitation_function);
pressure, start_time = fnm_transient_call(transducer_array, coordinate_grid, medium,\
           time_samples, ndiv, excitation_function, disp_flag);
pressure, start_time = fnm_transient_call(transducer_array, coordinate_grid, medium,\
           time_samples, ndiv, excitation_function, disp_flag, nthreads);
```

Arguments

- transducer_array: A FOCUS transducer array.
- coordinate_grid: A FOCUS coordinate grid.
- medium: A FOCUS medium.
- time_samples: A time samples struct created by set_time_samples.
- f0: The frequency of the array in Hz.
- ndiv: The number of integral points to use.
- excitation_function: The excitation function to use.
- disp_flag: Display flag, 1 = display, 0 = suppress.
- nthreads: The number of threads to use in the calculation.

Output Parameters

- pressure: A 4-d array representing the pressure at each point in spacetime.
- start_time: A vector containing the start time for the impulse response at each observation point.

Notes

This function only calculates pressures in lossless media, so any attenuation parameter set by the user will be ignored.

Threading in FOCUS is implemented at the transducer level, meaning that single transducers will not benefit from this feature.

Specifying a number of threads larger than the number of CPU cores available will not result in a significant additional speed increase and may in fact result in slower speeds due to additional inter-thread communication overhead.

fnm_transient_parfor

Description

Calculates the transient pressure field from an arbitrary transducer array using the Fast Nearfield Method.

Usage

```
pressure, start_time = fnm_transient_parfor(transducer_array, coordinate_grid,\
medium, time_samples, ndiv, excitation_function, disp_flag);
```

Arguments

- transducer_array: A FOCUS transducer array.
- coordinate_grid: A FOCUS coordinate grid.
- medium: A FOCUS medium.
- time_samples: A time samples struct created by set_time_samples.
- f0: The frequency of the array in Hz.
- ndiv: The number of integral points to use.
- excitation_function: The excitation function to use.
- disp_flag: Display flag, 1 = display, 0 = suppress.

Output Parameters

- pressure: A 4-d array representing the pressure at each point in spacetime.
- start_time: A vector containing the start time for the impulse response at each observation point.

Notes

This function only calculates pressures in lossless media, so any attenuation parameter set by the user will be ignored.

This function uses the MATLAB Parallel Computing Toolkit to perform the pressure calculation using multiple threads. The speed increase realized by this function is entirely dependent on the number of processor cores present in the computer executing the code. If the Parallel Computing Toolkit is not installed, the function will fail to run.

fnm_tsd

Description

Calculates the transient pressure field from an arbitrary transducer array using the Fast Nearfield Method and Time-Space Decomposition or Frequency Domain Time-Space Decomposition, depending on the input signal.

Usage

```
pressure = fnm_tsd(transducer_array, coordinate_grid, medium, time_struct, ndiv,\
           excitation_function);
pressure = fnm_tsd(transducer_array, coordinate_grid, medium, time_struct, ndiv,\
           excitation_function, disp_flag);
pressure = fnm_tsd(transducer_array, coordinate_grid, medium, time_struct, ndiv,\
           excitation_function, disp_flag, num_threads);
pressure = fnm_tsd(transducer_array, coordinate_grid, medium, time_struct, ndiv,\
           excitation_function, disp_flag, num_threads, precision_flag);
```

Arguments

- transducer_array: A transducer_array.
- coordinate_grid: A coordinate grid struct like the ones created by set_coordinate_grid.
- medium: A medium struct like the ones created by set_medium.
- time_struct: A time samples struct created by set_time_samples.
- ndiv: The number of integral points to use.
- excitation_function: An excitation function created by set_excitation_function.
- disp_flag: Display flag, 1 = display, 0 = suppress.
- num_threads: Number of CPU threads to use in the calculation. Default is 1.
- precision_flag: Precision flag, 1 = single precision, 0 = double precision. Default is 0 (double precision).

Output Parameters

- pressure: A 3-d array representing the pressure at each point in space.
- start_time: A vector containing the start time for the impulse response at each observation point.

Notes

This function only calculates pressures in lossless media, so any attenuation parameter set by the user will be ignored.

Threading in FOCUS is implemented at the transducer level, meaning that single transducers will not benefit from this feature.

Specifying a number of threads larger than the number of CPU cores available will not result in a significant additional speed increase and may in fact result in slower speeds due to additional inter-thread communication overhead.

Single precision calculations use about half as much memory as double precision calculations, so setting the precision flag to 1 can help with simulations that run out of memory.

FDTSD calculations are not as fast as TSD calculations. It is therefore recommended that users use one of the analytical TSD expressions if possible for the greatest speed.

fnm_tsd_parfor

Description

Calculates the transient pressure field from an arbitrary transducer array using the Fast Nearfield Method.

Usage

```
pressure = fnm_tsd_parfor(transducer_array, coordinate_grid, medium, time_struct,\
ndiv, excitation_function, disp_flag);
```

Arguments

- transducer_array: A transducer_array.
- coordinate_grid: A coordinate grid struct like the ones created by set_coordinate_grid.
- medium: A medium struct like the ones created by set_medium.
- time_struct: A time samples struct created by set_time_samples.
- ndiv: The number of integral points to use.
- excitation_function: The excitation function to use.
- disp_flag: Display flag, 1 = display, 0 = suppress.

Output Parameters

- pressure: A 3-d array representing the pressure at each point in space.
- start_time: A vector containing the start time for the impulse response at each observation point.

Notes

This function only calculates pressures in lossless media, so any attenuation parameter set by the user will be ignored.

This function uses the MATLAB Parallel Computing Toolkit to perform the pressure calculation using multiple threads. The speed increase realized by this function is entirely dependent on the number of processor cores present in the computer executing the code. If the Parallel Computing Toolkit is not installed, the function will fail to run.

get_apodization

Description

This function is used get a matrix containing the amplitudes of all elements of a transducer array.

Usage

```
amplitudes = get_apodization(transducer_array);
```

Arguments

- transducer_array: A FOCUS transducer array.

Output Parameters

- amplitudes: An m by n matrix where element (m,n) contains the amplitude of transducer array element (m,n).

get_chirp

Description

Returns a set of time points describing a linear chirp between two frequencies.

Usage

```
signal = get_chirp(f1, f2, dt, w);
```

Arguments

- f1: The starting frequency of the chirp in Hz
- f2: The ending frequency of the chirp in Hz
- dt: The sampling period of the signal in s
- w: The duration of the signal in s

Output Parameters

- signal: A set of points in time describing the amplitude of the signal in m/s.

Notes

The chirp signal is described as follows: $v(t) = \sin(2\pi(f_1 t + \frac{f_2 - f_1}{2w} t^2))$

get_circ

Description

This function creates a circular transducer. It is intended to work with the create_circ_arrray functions, but it can be used independently.

Usage

```
transducer = get_circ(radius);
transducer = get_circ(radius, center);
transducer = get_circ(radius, center, euler);
transducer = get_circ(radius, center, euler, piston_apodization_index);
```

Arguments

- radius: radius of the transducer in m.
- center: a 3x1 or 1x3 array with the center coordinate of the element.
- euler_angles: a 3x1 or 1x3 array with the rotation of the element in rad. For details about how FOCUS handles Euler angles, see the documentation for rotate_vector_forward.
- piston_apodization_index: Type of apodization to use. Must be one of the following:
  - 0: uniform - this is the default.
  - 1: cosine
  - 2: raised cosine
  - 3: quadratic
  - 4: quartic

Output Parameters

- transducer: A FOCUS transducer struct.

Notes

The weight of the transducer is set to 1 and the phase is set to zero. If you need to focus via phase shifting, you must manually change the phase after the function has run. Piston apodization is currently defined only for this transducer geometry.

get_excitation_function

Description

Returns a set of points corresponding to the given excitation function sampled with the given sampling period. This function is particularly useful for performing FDTSD calculations with excitation functions based on the analytically defined TSD functions.

Usage

```
points = get_excitation_function(ef_type, a0, f0, w, b, dt);
points = get_excitation_function(excitation_function, dt);
```

Arguments

- excitation_function: A FOCUS excitation function created with set_excitation_function.
- ef_type: A string representing the function type. Three are available:
  - "tone burst": tone burst ($A_0 \sin(2\pi f_0 t)$)
  - "hann pulse": Hann-weighted pulse ($A_0(1-\cos(2\pi \frac{t}{w}))\sin(2\pi f_0 t)$)
  - "tcubed pulse": broadband t^3 pulse ($A_0 t^3 e^{-Bt}\sin(2\pi f_0 t)$)
- a0: Amplitude of the function in m/s
- f0: Center frequency in Hz
- w: width of the pulse in s
- b: Exponential term in type 3 pulse.
- dt: The sampling period in s.

Output Parameters

- points: A set of points in time with the piston face velocity at that time.

Notes

Excitation function types can also be described using their integer identifiers; 1 for tone burst, 2 for Hann-weighted pulse, and 3 for t-cubed pulse.

get_phases

Description

This function is used get a matrix containing the phases of all elements of a transducer array.

Usage

```
phases = get_phases(transducer_array);
```

Arguments

- transducer_array: A FOCUS transducer array.

Output Parameters

- phases: An m by n matrix where element (m,n) contains the phase of transducer array element (m,n).

get_pressure

Description

Extracts a two-dimensional pressure plane from the three-dimensional pressure field returned by FOCUS functions (e.g. fnm_call).

Usage

```
pressure_2d = get_pressure(pressure_3d, plane, index);
```

Arguments

- pressure_3d: The result generated from fft_run or asa_run.
- plane: The 2-d plane to extract. Must be one of the following:
  - 'xy'
  - 'xz'
  - 'yz'
- index: the index of the plane.

Output Parameters

- pressure_2d: A 2-d pressure matrix.

Notes

This function is intended for users who are unfamiliar with the MATLAB colon notation.

get_rect

Description

Creates a rectangular transducer. It is intended to work with the create_rect_array functions, but it can be used independently.

Usage

```
transducer = get_rect(width, height);
transducer = get_rect(width, height, center);
transducer = get_rect(width, height, center, euler);
```

Arguments

- width: Width of the transducer in m.
- height: Height of the transducer in m.
- center: a 3x1 or 1x3 array with the center coordinate of the element.
- euler_angles: a 3x1 or 1x3 array with the rotation of the element in rad. For details about how FOCUS handles Euler angles, see the documentation for rotate_vector_forward.

Output Parameters

- transducer: A FOCUS transducer struct.

Notes

The weight of the transducer is set to 1 and the phase is set to zero. If you need to focus via phase shifting, you must manually change the phase after the function has run.

get_ring

Description

This function creates a planar ring transducer. It is intended to work with the create_ring_arrray functions, but it can be used independently.

Usage

```
transducer = get_ring(inner_radius, outer_radius);
transducer = get_ring(inner_radius, outer_radius, center);
transducer = get_ring(inner_radius, outer_radius, center, euler);
```

Arguments

- inner_radius: inner radius of the transducer in m.
- outer_radius: outer radius of the transducer in m.
- center: a 3x1 or 1x3 array with the center coordinate of the element.
- euler: a 3x1 or 1x3 array with the rotation of the element in rad. For details about how FOCUS handles Euler angles, see the documentation for rotate_vector_forward.

Output Parameters

- transducer: A FOCUS transducer struct.

Notes

The weight of the transducer is set to 1 and the phase is set to zero. If you need to focus via phase shifting, you must manually change the phase after the function has run.

get_spherical_shell

Description

This function creates a spherical shell transducer. It is intended to be used independently.

Usage

```
transducer = get_spherical_shell(radius, rad_curv);
transducer = get_spherical_shell(radius, rad_curv, center);
transducer = get_spherical_shell(radius, rad_curv, center, euler);
```

Arguments

- radius: The radius of the shell in m.
- rad_curv: The radius of curvature of the shell in m.
- center: a 3x1 or 1x3 array with the center coordinate of the element.
- euler_angles: a 3x1 or 1x3 array with the rotation of the element in rad. For details about how FOCUS handles Euler angles, see the documentation for rotate_vector_forward.

Output Parameters

- transducer: A FOCUS transducer struct.

Notes

The weight of the transducer is set to 1 and the phase is set to zero. If you need to focus via phase shifting, you must manually change the phase after the function has run.

get_spherically_focused_ring

Description

This function creates a planar ring transducer. It is intended to work with the create_ring_arrray functions, but it can be used independently.

Usage

```
transducer = get_spherically_focused_ring(inner_radius, outer_radius, geometric_focus);
transducer = get_spherically_focused_ring(inner_radius, outer_radius, geometric_focus,\
center);
transducer = get_spherically_focused_ring(inner_radius, outer_radius, geometric_focus,\
center, euler);
```

Arguments

- inner_radius: inner radius of the transducer in m.
- outer_radius: outer radius of the transducer in m.
- geometric_focus: geometric focus of the transducer in m.
- center: a 3x1 or 1x3 array with the center coordinate of the element.
- euler: a 3x1 or 1x3 array with the rotation of the element in rad. For details about how FOCUS handles Euler angles, see the documentation for rotate_vector_forward.

Output Parameters

- transducer: A FOCUS transducer struct.

Notes

The weight of the transducer is set to 1 and the phase is set to zero. If you need to focus via phase shifting, you must manually change the phase after the function has run.

get_time_delays

Description

This function is used get a matrix containing the time delays of all elements of a transducer array.

Usage

```
delays = get_time_delays(transducer_array);
```

Arguments

- transducer_array: A FOCUS transducer array.

Output Parameters

- delays: An m by n matrix where element (m,n) contains the time delay of transducer array element (m,n).

get_tissuestruct

Description

Notice: This function is deprecated. Please use set_layered_medium instead.

Create a structure representing multiple layers of different tissues.

Usage

```
ts = get_tissuestruct(z, f0, nlayer, z_int, types);
```

Arguments

- z: Grid vector in the z direction.
- f0: Frequency of the transducer array in Hz. Used to calculate frequency-dependent properties of the tissue layers.
- nlayer: Number of tissue layers.
- z_int: Vector containing z coordinates of interfaces between tissue types.
- types: The tissue types of the various layers. Valid types are as follows:
  - 'f': fat
  - 'l': liver
  - 'm': muscle
  - 's': skin
  - 'w': water

Output Parameters

- ts: A MATLAB struct representing the different tissue properties and their relative locations on the z-axis.

Notes

The first layer of the tissue struct starts at z(1) and ends at the boundary of the next layer. The final layer ends at the final z coordinate.

impulse_begin_and_end_times

Description

Finds start and end times for transient calculations.

Usage

```
[t_start, t_end] = impulse_begin_and_end_times(transducer_array, coordinate_grid,\
medium);
[t_start, t_end] = impulse_begin_and_end_times(transducer_array, coordinate_grid,\
medium, nthreads);
```

Arguments

- transducer_array: A transducer_array.
- coordinate_grid: A coordinate grid struct like the ones created by set_coordinate_grid.
- medium: A medium struct like the ones created by set_medium.
- nthreads: The number of threads to use in the calculation.

Output Parameters

- t_start: The start time for the impulse response calculation.
- t_end: The end time for the impulse response calculation.

Notes

Threading in FOCUS is implemented at the transducer level, meaning that single transducers will not benefit from this feature.

Specifying a number of threads larger than the number of CPU cores available will not result in a significant additional speed increase and may in fact result in slower speeds due to additional inter-thread communication overhead.

impulse_response

Description

Calculates the impulse response of a transducer array.

Usage

```
ir = impulse_response(xdcr_array, coord_grid, medium, time_samples);
ir = impulse_response(xdcr_array, coord_grid, medium, time_samples, disp_flag);
ir = impulse_response(xdcr_array, coord_grid, medium, time_samples, disp_flag,\
nthreads);
```

Arguments

- xdcr_array: A FOCUS transducer array.
- coord_grid: A coordinate grid struct like the ones created by set_coordinate_grid.
- medium: A medium struct like the ones created by set_medium.
- time_samples: A time sample struct created by set_time_samples.
- disp_flag: Display flag, 1 = display, 0 = suppress.
- nthreads: The number of threads to use in the calculation.

Output Parameters

- ir: A 4-d array representing the impulse response at each point in space and time.

Notes

This function only calculates pressures in lossless media, so any attenuation parameter set by the user will be ignored.

Threading in FOCUS is implemented at the transducer level, meaning that single transducers will not benefit from this feature.

Specifying a number of threads larger than the number of CPU cores available will not result in a significant additional speed increase and may in fact result in slower speeds due to additional inter-thread communication overhead.

impulse_response_parfor

Description

Calculates the impulse response of a transducer array.

Usage

```
ir = impulse_response_parfor(xdcr_array, coord_grid, medium, time_samples, disp_flag);
```

Arguments

- xdcr_array: A xdcr_array.
- coord_grid: A coordinate grid struct like the ones created by set_coord_grid.
- medium: A medium struct like the ones created by set_medium.
- time_samples: A time sample struct created by set_time_samples.
- disp_flag: Display flag, 1 = display, 0 = suppress.

Output Parameters

- ir: A 3-d array representing the impulse response at each point in space.

Notes

This function only calculates pressures in lossless media, so any attenuation parameter set by the user will be ignored.

This function uses the MATLAB Parallel Computing Toolkit to perform the calculation using multiple threads. The speed increase realized by this function is entirely dependent on the number of processor cores present in the computer executing the code. If the Parallel Computing Toolkit is not installed, the function will fail to run.

kzk_cw

Description

Calculates the continuous-wave pressure field from an arbitrary transducer array using the KZK equation.

Usage

```
pressure = kzk_cw(transducer_array, coordinate_grid, medium, f0, nharm,\
dsig_fd, piston_pts, rho_max, exflag);
```

Arguments

- transducer_array: A transducer_array.
- coordinate_grid: A coordinate grid struct like the ones created by set_coordinate_grid.
- medium: A medium struct like the ones created by set_medium.
- f0: Frequency of the transducer in Hz.
- nharm: The number of harmonics to use in the calculation.
- dsig_fd: Normalized axial step size.
- piston_pts: The number of radial divisions within a piston radius.
- rho_max: Normalized max radial units.
- exflag: For circular transducers only. If 0, the untransformed coordinate system is used in the calculation. If 1, the coordinate system is transformed before the pressure calculation.

Output Parameters

- time: A vector of points in time from tmin to tmax.
- pressure: A 4-d array representing the complex pressure at each point in space for each harmonic such that pressure(x, y, z, n) is the pressure at point (x, y, z) for the nth harmonic.

kzk_transient

Description

Calculates the transient pressure field from an arbitrary transducer array using the KZK equation.

Usage

```
pressure = kzk_transient(transducer_array, coordinate_grid, medium, time_struct,\
excitation_function, di_flag, fd_mode, dsig_fd, piston_pts, rho_max);

[time, pressure] = kzk_transient(transducer_array, coordinate_grid, medium,\
time_struct, excitation_function, di_flag, fd_mode, dsig_fd, piston_pts, rho_max);
```

Arguments

- transducer_array: A transducer_array.
- coordinate_grid: A coordinate grid struct like the ones created by set_coordinate_grid.
- medium: A medium struct like the ones created by set_medium.
- time_struct: A time samples struct created by set_time_samples.
- excitation_function: An excitation function created by set_excitation_function.
- di_flag: Diffraction effect is included if di_flag = 1.
- fd_mode: Implicit backward finite difference (IBFD) method is used when fd_mode = 1 and Crank-Nicolson finite difference (CNFD) method is used when fd_mode = 0.
- dsig_fd: Normalized axial step size.
- piston_pts: The number of radial divisions within a piston radius.
- rho_max: Normalized max radial units.

Output Parameters

- time: A vector of points in time from tmin to tmax.
- pressure: A 4-d array representing the transient pressure at each point in spacetime.

layerasa

Description

This function uses the Angular Spectrum Approach to perform calculations in layered media.

Usage

```
pressure = layerasa(p0, z, medium, nfft, delta, f0);
```

Arguments

- p0: matrix of size [nx,ny], input pressure or velocity source.
- z: vector, location of destination planes.
- medium: A layered medium struct constructed with set_layered_medium.
- nfft: Number of FFT terms to use in the calculation. Must be greater than the largest dimensions of the input pressure matrix. See notes for details on choosing this value.
- delta: scalar, spatial sampling interval in meters.
- f0: Excitation frequency of the source wave.

Output Parameters

- pressure: matrix of size [nx ny nz], calculated pressure.

Notes

This function is currently in a testing phase and the results it computes have not been fully validated. If you encounter any calculation errors or other problems when using this function, please contact the development team so we can work on rectifying the problem.

To choose the number of FFT terms to use, the simplest method is to zero-pad the source pressure and increase the number of terms until an acceptable error is reached. Numbers of terms that are powers of two (e.g. 512, 1024) usually result in the fastest calculations.

make_tissuestruct

Notice: This function is deprecated. Please use get_tissuestruct instead.

Description

Create a structure representing multiple layers of different tissues.

Usage

```
ts = get_tissuestruct(z, nlayer, z_int, types);
```

Arguments

- z: Grid vector in the z direction.
- nlayer: Number of tissue layers.
- z_int: Vector containing z coordinates of interfaces between tissue types.
- types: Vector of tissue types. Valid types are as follows:
  - 'f': fat
  - 'l': liver
  - 'm': muscle
  - 's': skin
  - 'w': water

Output Parameters

- ts: A MATLAB struct representing the different tissue properties and their relative locations on the z-axis.

Notes

The first layer of the tissue struct starts at z(1).

pause2

Description

This is a call back function used by the C++ binary to force an update of the Matlab display buffer. It should not be changed or deleted under any circumstance. If this function is missing, the program WILL NOT WORK.

plot_apodization

Description

Plots the amplitude of each element of a transducer array.

Usage

```
plot_apodization(transducer_array);
```

Arguments

- transducer_array: A FOCUS transducer array.

plot_transient_pressure

Description

Plots the amplitude of each element of a transducer array.

Usage

```
plot_transient_pressure(pressure, coord_grid, time_samples, plane);
plot_transient_pressure(pressure, coord_grid, time_samples, plane, plot_type);
```

Arguments

- pressure: A 4-D matrix containing transient pressure data, e.g. the result of a calculation with fnm_tsd.
- coord_grid: A FOCUS coordinate grid created with set_coordinate_grid.
- time_samples: A FOCUS time samples struct created with set_time_samples.
- plane: The plane to use when displaying the pressure. Options are 'xz', 'yz', and 'xy'.
- plot_type: An optional argument describing the type of plot to use. Options are 'mesh' and 'pcolor'.

rayleigh_cw

Description

Calculates pressure field generated by a transducer array in homogeneous media using the Rayleigh--Sommerfeld Integral.

Usage

```
pressure = rayleigh_cw(transducer_array, coordinate_grid, medium, ndiv, f0);
pressure = rayleigh_cw(transducer_array, coordinate_grid, medium, ndiv, f0, disp_flag);
pressure = rayleigh_cw(transducer_array, coordinate_grid, medium, ndiv, f0, disp_flag,\
nthreads);
```

Arguments

- transducer_array: A transducer_array.
- coordinate_grid: A coordinate grid struct like the ones created by set_coordinate_grid.
- medium: A medium struct like the ones created by set_medium.
- ndiv: The number of integral points to use.
- f0: The frequency of the array.
- disp_flag: Display flag, 1 = display, 0 = suppress.
- nthreads: The number of threads to use in the calculation.

Output Parameters

- pressure: A 3-d array representing the complex pressure at each point in space.

Notes

Threading in FOCUS is implemented at the transducer level, meaning that single transducers will not benefit from this feature.

Specifying a number of threads larger than the number of CPU cores available will not result in a significant additional speed increase and may in fact result in slower speeds due to additional inter-thread communication overhead.

rayleigh_cw_apodized

Description

Calculates pressure field generated by a transducer array in homogeneous media using the Rayleigh--Sommerfeld Integral.

Usage

```
pressure = rayleigh_cw_apodized(xdcr_array, coordinate_grid, medium, ndiv, f0);
pressure = rayleigh_cw_apodized(xdcr_array, coordinate_grid, medium, ndiv, f0,\
disp_flag);
```

Arguments

- xdcr_array: A xdcr_array.
- coordinate_grid: A coordinate grid struct like the ones created by set_coordinate_grid.
- medium: A medium struct like the ones created by set_medium.
- ndiv: The number of integral points to use.
- f0: The frequency of the array.
- disp_flag: Display flag, 1 = display, 0 = suppress.

Output Parameters

- pressure: A 3-d array representing the complex pressure at each point in space.

Notes

Piston apodization only works for circular transducers. See the documentation for get_circ for details.

rayleigh_cw_parfor

Description

Calculates pressure field generated by a transducer array in homogeneous media using the Rayleigh--Sommerfeld Integral.

Usage

```
pressure = rayleigh_cw_parfor(xdcr_array, coordinate_grid, medium, ndiv, f0);
pressure = rayleigh_cw_parfor(xdcr_array, coordinate_grid, medium, ndiv, f0, disp_flag);
```

Arguments

- xdcr_array: A xdcr_array.
- coordinate_grid: A coordinate grid struct like the ones created by set_coordinate_grid.
- medium: A medium struct like the ones created by set_medium.
- ndiv: The number of integral points to use.
- f0: The frequency of the array.
- disp_flag: Display flag, 1 = display, 0 = suppress.

Output Parameters

- pressure: A 3-d array representing the complex pressure at each point in space.

Notes

This function uses the MATLAB Parallel Computing Toolkit to perform the pressure calculation using multiple threads. The speed increase realized by this function is entirely dependent on the number of processor cores present in the computer executing the code. If the Parallel Computing Toolkit is not installed, the function will fail to run.

rayleigh_cw_sse

Description

Calculates pressure field generated by a transducer array in homogeneous media using the Rayleigh--Sommerfeld Integral and Streaming SIMD Extensions (SSE).

Usage

```
pressure = rayleigh_cw_sse(transducer_array, coordinate_grid, medium, ndiv, f0);
pressure = rayleigh_cw_sse(transducer_array, coordinate_grid, medium, ndiv, f0,
disp_flag);
pressure = rayleigh_cw_sse(transducer_array, coordinate_grid, medium, ndiv, f0,
disp_flag,\
nthreads);
```

Arguments

- transducer_array: A transducer_array.
- coordinate_grid: A coordinate grid struct like the ones created by set_coordinate_grid.
- medium: A medium struct like the ones created by set_medium.
- ndiv: The number of integral points to use.
- f0: The frequency of the array.
- disp_flag: Display flag, 1 = display, 0 = suppress.
- nthreads: The number of threads to use in the calculation.

Output Parameters

- pressure: A 3-d array representing the complex pressure at each point in space.

Notes

SSE calculations are about 4x as fast as standard calculations, but they are less accurate (on the order of 10^{-4}) because single precision values are used.

Threading in FOCUS is implemented at the transducer level, meaning that single transducers will not benefit from this feature.

Specifying a number of threads larger than the number of CPU cores available will not result in a significant additional speed increase and may in fact result in slower speeds due to additional inter-thread communication overhead.

rayleigh_transient

Description

Calculates pressure field generated by a transducer array in homogeneous media using the Rayleigh--Sommerfeld Integral.

Usage

```
pressure = rayleigh_transient(transducer_array, coordinate_grid, medium,\
time_struct, ndiv, input_function);
pressure = rayleigh_transient(transducer_array, coordinate_grid, medium,\
time_struct, ndiv, input_function, disp_flag);
pressure = rayleigh_transient(transducer_array, coordinate_grid, medium,\
time_struct, ndiv, input_function, disp_flag, nthreads);
```

Arguments

- transducer_array: A transducer_array.
- coordinate_grid: A coordinate grid struct like the ones created by set_coordinate_grid.
- medium: A medium struct like the ones created by set_medium.
- time_struct: A time struct generated by set_time_samples.
- ndiv: The number of integral points to use.
- input_function: The excitation function. Create using set_excitation_function.
- disp_flag: Display flag, 1 = display, 0 = suppress.
- nthreads: The number of threads to use in the calculation.

Output Parameters

- pressure: A 3-d array representing the pressure at each point in space.

Notes

This function only calculates pressures in lossless media, so any attenuation parameter set by the user will be ignored.

Threading in FOCUS is implemented at the transducer level, meaning that single transducers will not benefit from this feature.

Specifying a number of threads larger than the number of CPU cores available will not result in a significant additional speed increase and may in fact result in slower speeds due to additional inter-thread communication overhead.

rayleigh_transient_parfor

Description

Calculates pressure field generated by a transducer array in homogeneous media using the Rayleigh--Sommerfeld Integral.

Usage

```
pressure = rayleigh_transient_parfor(transducer_array, coordinate_grid, medium,\
time_struct, ndiv, input_function, disp_flag);
```

Arguments

- transducer_array: A transducer_array.
- coordinate_grid: A coordinate grid struct like the ones created by set_coordinate_grid.
- medium: A medium struct like the ones created by set_medium.
- time_struct: A time struct generated by set_time_samples.
- ndiv: The number of integral points to use.
- input_function: The excitation function. Create using set_excitation_function.
- disp_flag: Display flag, 1 = display, 0 = suppress.

Output Parameters

- pressure: A 3-d array representing the pressure at each point in space.

Notes

This function only calculates pressures in lossless media, so any attenuation parameter set by the user will be ignored.

This function uses the MATLAB Parallel Computing Toolkit to perform the pressure calculation using multiple threads. The speed increase realized by this function is entirely dependent on the number of processor cores present in the computer executing the code. If the Parallel Computing Toolkit is not installed, the function will fail to run.

rotate_vector_forward

Description

Interprets Euler angles to rotate vectors.

Usage

```
rotated_vector = rotate_vector_forward(vector, angle);
```

Arguments

- vector: The vectors ([$[x_1, y_1, z_1]$...$[x_n, y_n, z_n]$]) to be rotated.
- angle: Euler angles of the form [θ ψ ϕ]. All angles should be in radians.

Output Parameters

- rotated_vector: The rotated input vector.

Notes

The input vector must be three-dimensional, even if some dimensions are zero. FOCUS interprets Euler angles such that θ rotates the vector about the y axis, ψ rotates the vector about the $x\prime$ axis, and ϕ rotates the vector about the $z\prime\prime$ axis.

set_apodization

Description

This function is used to apodize transducer arrays by setting the amplitudes of the individual transducers.

Usage

```
transducer = set_apodization(transducer, apodization, r);
```

Arguments

- transducer: A FOCUS transducer array.
- apodization: A vector the same size as the transducer array containing the amplitude to be used for each element in the array, e.g. [0 0.5 1 0.5 0] for a five-element array. This argument can also be a string representing the type of apodization to use. Options are:
  - bartlett: Bartlett-Hann window --- see documentation for the MATLAB bartlett() function for details.
  - chebyshev: Chebyshev window --- see documentation for the MATLAB chebwin() function for details.
  - hamming: Hamming window --- $w(n) = 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right)$
  - hann: Hann window --- $w(n) = 0.5\left(1 - \cos\left(\frac{2\pi n}{N-1}\right)\right)$
  - triangle: Triangular window --- $w(n) = \frac{2n}{N-1}\left(\frac{N-1}{2} - \left|n - \frac{N-1}{2}\right|\right)$
  - uniform: Uniform window --- $w(n) = 1$
- r: An optional tuning parameter used only for Chebyshev apodization. r is the desired size of the sidelobes relative to the main lobe in dB. The default value is 100.

Output Parameters

- transducer: The transducer struct with the new amplitude values.

set_center_coordinate_grid

Description

Creates a coordinate grid centered at the given point.

Usage

```
cg = set_center_coordinate_grid(delta, x, y, z, nx, ny, nz);
```

Arguments

- delta: An integer or vector ([dx dy dz]) representing the difference between two points in the grid.
- x: The x coordinate of the center of the grid.
- y: The y coordinate of the center of the grid.
- z: The z coordinate of the center of the grid.
- nx: The number of points in the x direction of the grid.
- ny: The number of points in the y direction of the grid.
- nz: The number of points in the z direction of the grid.

Output Parameters

- cg: A coordinate grid struct with the given center.

Notes

If nx, ny, or nz is set to one, there will be no points in that direction of the grid, resulting in a coordinate plane or coordinate line instead of a 3-d grid.

set_coordinate_grid

Description

Creates a coordinate grid data structure used by FOCUS for various calculations.

Usage

```
grid = set_coordinate_grid(delta, xmin, xmax, ymin, ymax, zmin, zmax);
grid = set_coordinate_grid(x_coords, y_coords, z_coords);
grid = set_coordinate_grid(vector);
```

Arguments

- delta: The difference between two data points in a coordinate direction. Should be a matrix of the form [dx dy dz].
- xmin: The lowest x plane.
- xmax: The highest x plane.
- ymin: The lowest y plane.
- ymax: The highest y plane.
- zmin: The lowest z plane.
- zmax: The highest z plane.
- x_coords: A list of x coordinates.
- y_coords: A list of y coordinates.
- z_coords: A list of z coordinates.
- vector: a list of coordinates in [[x1 y1 z1]; [x2 y2 z2];...] notation.

Output Parameters

- grid: A coordinate grid struct that can be used by FNM functions.

Notes

If x_coords, y_coords, and z_coords are provided, each point is defined as point(i) = [x_coords(i) y_coords(i) z_coords(i)]. If no arguments are provided, the user will be prompted to enter delta, xmin, xmax, ymin, ymax, zmin, and zmax.

Calling this function with no arguments will cause the program to prompt the user for each required value.

set_digitized_time_delays

Description

Notice: This function is deprecated. Please use set_time_delays instead.

Determine the time delays needed to focus a transducer array at the given point. Time delays are shifted to fit the temporal grid rather than being allowed to fall between temporal samples.

Usage

```
transducer = set_digitized_time_delays(transducer, x, y, z, medium, fs);
```

Arguments

- transducer: A FOCUS transducer array.
- x: The x coordinate of the focus.
- y: The y coordinate of the focus.
- z: The z coordinate of the focus.
- medium: A FOCUS medium.
- fs: The sampling frequency in Hz.

Output Parameters

- transducer: The input transducer array with adjusted time delays.

Notes

This function alters the transducer struct, the output transducer should be the same as the input transducer (transducer).

set_excitation_function

Description

A function to create an excitation function suitable for use with transient_pressure and other transient pressure calculation functions.

Usage

```
fn = set_excitation_function(type, f0, w);
fn = set_excitation_function(type, f0, w, b);
fn = set_excitation_function(signal, sample_period, clipping_threshold);
```

Arguments

- type: A string representing the function type. Three are available:
  - "tone burst": tone burst ($A_0 \sin(2\pi f_0 t)$)
  - "hann pulse": Hann-weighted pulse ($A_0(1 - \cos(2\pi \frac{t}{w})) \sin(2\pi f_0 t)$)
  - "tcubed pulse": broadband t^3 pulse ($A_0 t^3 e^{-Bt} \sin(2\pi f_0 t)$)
- f0: Center frequency in Hz
- w: width of the pulse in s
- b: Exponential term in the t-cubed pulse.
- signal: A MATLAB vector containing a set of time samples representing the excitation signal in m/s.
- sample_period: The duration between points in the excitation signal in s.
- clipping_threshold: Optional argument describing the threshold below which frequency components will be ignored. If this value is negative, it will be assumed to be in dB, otherwise any value between zero and one can be used.

Output Parameters

- fn: A MATLAB structure containing the values specified by the user.

Notes

Transient calculations can be performed with arbitrary input signals, however performance will be improved if one of the signal types for which an analytical TSD expression has been derived is used. Otherwise, the calculation will be performed using FDTSD. See the documentation for fnm_tsd for details.

Calling this function with no arguments will cause the program to prompt the user for each required value.

The amplitude of the excitation can be set on each transducer by changing its amplitude property.

Excitation function types can also be described using their integer identifiers; 1 for tone burst, 2 for Hann-weighted pulse, and 3 for t-cubed pulse.

set_fdtsd_excitation

Description

A function to create an excitation function for use in Frequency Domain Time-Space Decomposition (FDTSD) calculations.

Usage

```
fn = set_fdtsd_excitation(signal, sample_period, clipping_threshold);
```

Arguments

- signal: A MATLAB vector containing a set of time samples representing the excitation signal in m/s.
- sample_period: The duration between points in the excitation signal in s.
- clipping_threshold: Optional argument describing the threshold below which frequency components will be ignored. If this value is negative, it will be assumed to be in dB, otherwise any value between zero and one can be used. A value of zero will disable spectral clipping.

Output Parameters

- fn: A MATLAB structure containing the specified values.

Notes

Transient calculations can be performed with arbitrary input signals, however performance will be improved if one of the signal types for which an analytical TSD expression has been derived is used. Otherwise, the calculation will be performed using FDTSD. See the documentation for fnm_tsd for details.

set_layered_medium

Description

Create a medium struct for use in ASA layered medium calculations.

Usage

```
layered_medium = set_layered_medium(z_start, medium_properties);
```

Arguments

- z_start: An array of numbers representing the start point of each layer on the z-axis in m. The first layer must start at z = 0.
- medium_properties: An array of FOCUS medium structs as constructed by set_medium.

Output Parameters

- layered_medium: An array of MATLAB structs with the following properties:
  - specificheatofblood
  - bloodperfusion
  - density
  - soundspeed
  - powerlawexponent
  - attenuationdBcmMHz
  - specificheat
  - thermalconductivity
  - nonlinearityparameter
  - z_start: The start point of the layer on the z-axis in m
  - z_end: The end point of the layer on the z-axis in m. This is equivalent to the start point of the following layer or infinity for the last layer.

Notes

Each layer is constructed such that layer n spans from z_start(n) to z_start($n+1$) and has all of the properties specified in medium_properties(n).

set_medium

Description

Create a medium struct for use with other FOCUS functions.

Usage

```
medium = set_medium(cb, wb, rho, c_sound, b, atten_coeff, ct, kappa, beta);
medium = set_medium('lossless');
medium = set_medium('specificheatofblood',cb,'bloodperfusion',wb,'density',rho,\
'soundspeed',c_sound,'powerlawexponent',b,'attenuationdBcmMHz',atten_coeff,\
'specificheat',ct,'thermalconductivity',kappa,'nonlinearityparameter',beta);
```

Arguments

- cb: Specific heat of blood in J/kg/K
- wb: Blood perfusion in kg/m^3/s
- rho: The density of the medium in kg/m^3
- c_sound: The speed of sound in m/s
- b: Power law coefficient; unitless.
- atten_coeff: Attenuation coefficient in dB/cm/MHz
- ct: Specific heat of the medium in J/kg/K
- kappa: Thermal conductivity in W/m/K
- beta: Nonlinear parameter; unitless.

Output Parameters

- medium: A MATLAB struct with the following properties:
  - specificheatofblood
  - bloodperfusion
  - density
  - soundspeed
  - powerlawexponent
  - attenuationdBcmMHz
  - specificheat
  - thermalconductivity
  - nonlinearityparameter

Notes

Calling set_medium with a string argument will create one of the default media created by define_media. The options for these strings are:

- lossless
- attenuated
- water
- skin
- fat
- muscle
- liver

When the arguments are preceded by their names (e.g. set_medium('density',1500);), they can be in any order and any number of them can be missing. Calling set_medium with no arguments would be equivalent to calling set_medium('lossless');.

Calling this function with no arguments will cause the program to prompt the user for each required value.

set_parameters

Description

Notice: This function is deprecated. Please use set_medium instead.

Create a parameter struct for use with FNM functions.

Usage

```
params = set_parameters('c_sound', c_sound, 'atten_coeff', atten_coeff,\
'rho_density', rho, 'f0', f0, 'fs', fs);
```

Arguments

- c_sound: The speed of sound in m/s
- atten_coeff: The attenuation coefficient in dB/cm/MHz
- rho: The density of the medium in kg/m^3
- f0: The frequemcy of the transducer array in Hz
- fs: The sampling frequency in Hz

Output Parameters

- params: A parameter struct.

Notes

Calling this function with no arguments will cause the program to prompt the user for each required value.

set_phases

Description

This function is used to apodize transducer arrays by setting the amplitudes of the individual transducers.

Usage

```
transducer = set_phases(transducer, phases);
```

Arguments

- transducer: A FOCUS transducer array.
- phases: A vector the same size as the transducer array containing the phase to be used for each element in the array, e.g. [0 $\pi/2$ π $\pi/2$ 0] for a five-element array.

Output Parameters

- transducer: The transducer struct with the new phase values.

set_temporal_coordinate_grid

Description

Notice: This function is deprecated. Please use set_time_samples and set_coordinate_grid instead.

Creates a coordinate grid for use with transient functions.

Usage

```
tcg = set_temporal_coordinate_grid(vector, time_vector);
tcg = set_temporal_coordinate_grid(vector, tmin, tmax, deltat);
tcg = set_temporal_coordinate_grid(delta, xmin, xmax, ymin, ymax, zmin, zmax,\
time_vector);
tcg = set_temporal_coordinate_grid(delta, xmin, xmax, ymin, ymax, zmin, zmax,\
tmin, tmax, deltat);
```

Arguments

- vector: A vector containing a list of coordinates.
- time_vector: A vector containing an explicit list of sampling points.
- delta: The difference between two data points in a coordinate direction. Should be a matrix of the form [dx dy dz].
- xmin: The lowest x plane.
- xmax: The highest x plane.
- ymin: The lowest y plane.
- ymax: The highest y plane.
- zmin: The lowest z plane.
- zmax: The highest z plane.
- tmin: The starting time.
- tmax: The ending time.
- deltat: The difference between adjacent time samples.

Output Parameters

- tcg: A temporal coordinate grid struct.

Notes

Calling this function with no arguments will cause the program to prompt the user for each required value.

set_time_delays

Description

Determine the time delays required to focus a transducer array at a given point. Time delays calculated with set_time_delays may fall between temporal samples unless the sampling frequency is provided, in which case the delays are shifted to fit the temporal grid (digitized) rather than being allowed to fall between temporal samples.

Usage

```
transducer = set_time_delays(transducer, x, y, z, medium);
transducer = set_time_delays(transducer, x, y, z, medium, fs);
transducer = set_time_delays(transducer, x, y, z, medium, fs, linear_array);
transducer = set_time_delays(transducer, delays);
```

Arguments

- transducer: A transducer array, e.g. one created by create_circ_csa.
- x: The x coordinate of the focus.
- y: The y coordinate of the focus.
- z: The z coordinate of the focus.
- medium: A medium struct.
- fs: Optional sampling frequency in Hz. If provided, digitized time delays (i.e. time delays shifted to align with the sampling frequency) will be used.
- linear_array - Whether to treat the array as a linear array, i.e. all array elements with the same x-coordinate will be focused together and have the same delay.
- delays: The time delays to assign to the array elements such that the time delay of transducer(n) = delays(n).

Output Parameters

- transducer: The input transducer array with adjusted time delays.

Notes

This function alters the transducer struct so the output transducer should be the same as the input transducer. If the sampling frequency is provided, the time delays will be digitized, i.e. they will be shifted to occur exactly on the time samples rather than between them.

set_time_samples

Description

Create a time sample struct for use with transient functions.

Usage

```
time_struct = set_time_samples(samples);
time_struct = set_time_samples(deltat, tmin, tmax);
```

Arguments

- samples: A vector containing time samples (e.g. [t1 t2 t3]).
- deltat: The difference between adjacent time samples.
- tmin: The starting time.
- tmax: The ending time.

Output Parameters

- time_struct: A structure containing time sample information.

Notes

Calling set_time_samples with no arguments will cause the function to prompt the user to enter deltat, tmin, and tmax. Additionally, note that set_time_samples(samples) is VERY different from set_time_samples(deltat, tmin, tmax); the former will create a grid that samples only at the specified points, whereas the latter will create a regular grid between tmin and tmax.

trans_rot

Description

This is an internal function used by draw_array and its related functions. Users are not expected to use this function. It is designed to interpret Euler angles when drawing transducer arrays in MATLAB.

transient_pressure

Description

Calculates the transient pressure field from an arbitrary transducer array using the Fast Nearfield Method and Time-Space Decomposition or Frequency Domain Time-Space Decomposition, depending on the input signal.

Usage

```
pressure = transient_pressure(transducer_array, coordinate_grid, medium, time_struct,\
ndiv, excitation_function); pressure = transient_pressure(transducer_array, coor-
dinate_grid, medium, time_struct,\
ndiv, excitation_function, calculation_method);
```

Arguments

- transducer_array: A transducer_array.
- coordinate_grid: A coordinate grid struct like the ones created by set_coordinate_grid.
- medium: A medium struct like the ones created by set_medium.
- time_struct: A time samples struct created by set_time_samples.
- ndiv: The number of integral points to use.
- excitation_function: An excitation function created by set_excitation_function.
- calculation_method: The pressure calculation algorithm to use. Options are:
  - 'fnm': Fast Nearfield Method with Time-Space Decomposition
  - 'fnm transient': Fast Nearfield Method without TSD
  - 'rayleigh': Transient Rayleigh-Sommerfeld method

Output Parameters

- pressure: A 3-d array representing the complex pressure at each point in space.

Notes

Advanced users may wish to use the individual calculation functions directly for access to more features and options.

FDTSD calculations are not as fast as TSD calculations. It is therefore recommended that users use one of the analytical TSD expressions if possible for the greatest speed.

This function only calculates pressures in lossless media, so any attenuation parameter set by the user will be ignored.