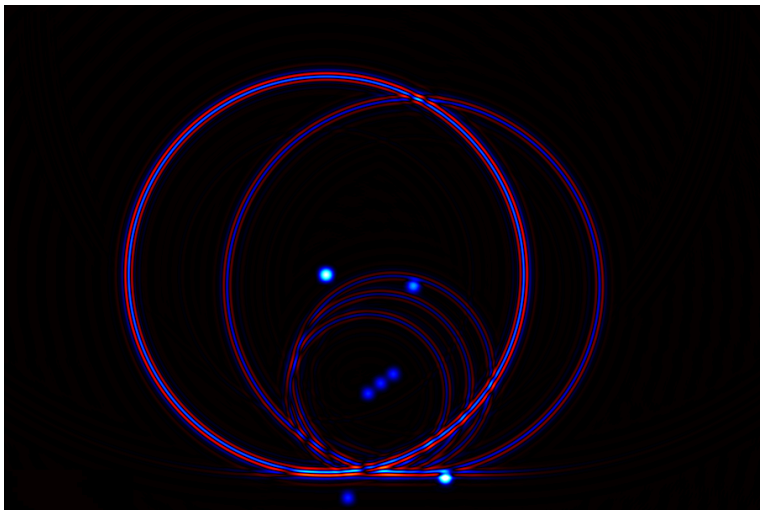
MUST – Matlab UltraSound Toolbox

User guide



Version 2.0.20210117

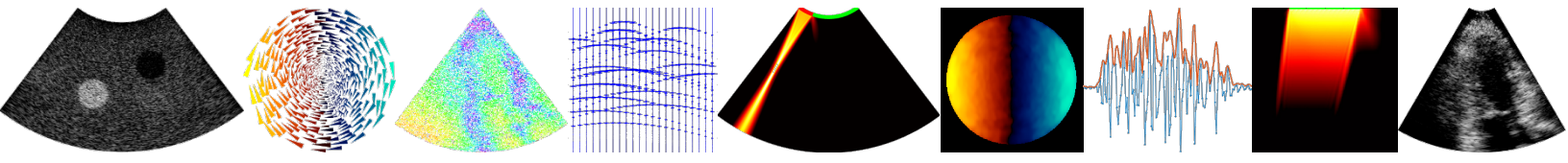
(last update: 2021-01-17)

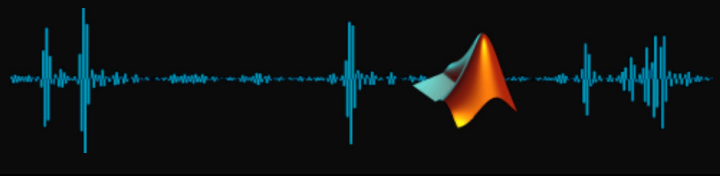
BiomeCardio
Research in Biomechanics & Cardiology

www.biomecardio.com/MUST

Inserm

CREATIS





The MUST toolbox allows one to design transmit sequences, simulate RF signals, post-process and demodulate them, beamform and generate images, estimate Doppler velocities, and make educational figures and animations.

The MUST toolbox is the result of several consecutive research projects. Its development started around 2010. Over the years, I have written Matlab codes for my students and myself to analyze signals acquired with Verasonics scanners. These codes were then included in a toolbox. The Matlab language was chosen for pedagogical and practical reasons. When we had to perform simulations for pre-tests, we first did our simulations with Field II. I then integrated simulators (PFIELD and SIMUS) into the toolbox since I needed fast and open codes for my students. I chose the Fourier domain because it was more appropriate in many ways. The challenge was that the functions could be handled easily after 10 minutes of training. The syntax has been made uniform and the default settings are those commonly used in medical ultrasound imaging. You can enter "help *function_name*" in the Matlab command window to obtain exhaustive descriptions.

The programs of the MUST toolbox are currently for two-dimensional analysis. Two-dimensionality is indeed adapted to most ultrasound medical imaging applications. Given the availability and growth of 3-D ultrasound imaging, the functions of MUST will, one day, also be available in 3-D. This update will occur if MUST meets with some success and if there is a real and significant demand for 3-D imaging.

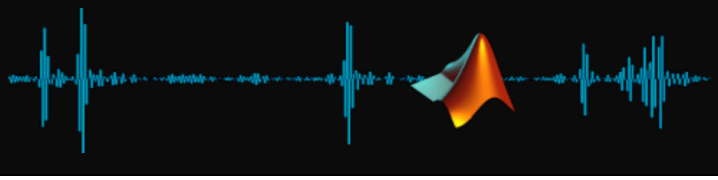
Since version #2 of MUST (2021), simulations with PFIELD and SIMUS can take into account elevation focusing.

It goes without saying that the MUST simulators do not claim to represent reality. They just approximate it. Strong assumptions are underlying them. To list the most important ones: 1) linearity, 2) scatterers acting as monopole sources, 3) weak (single) scattering. The electronic part of the probe is not taken into account, because I have no expertise in this area. The units of the RF signals and pressure fields are therefore arbitrary.

The theory used in MUST has been presented here and there, during IUS short courses and a TUFFC international school, or in some articles (list below). I plan to submit an article that will synthesize it (in 2021). Theoretical pages will be added to the website as I find the time to write them. For the moment, I suggest that you refer to the following documents:

1. Porée J, Posada D, Hodzic A, Tournoux F, Cloutier G, Garcia D. **High-frame-rate echocardiography using coherent compounding with Doppler-based motion-compensation**. *IEEE Trans Med Imaging*, 2016;35:1647-1657 (see the supplemental content in the PDF)
2. Madiena C, Faurie J, Porée J, Garcia D. **Color and vector flow imaging in parallel ultrasound with sub-Nyquist sampling**. *IEEE Trans Ultrason Ferroelectr Freq Control*, 2018;65:795-802 (PDF)
3. Shahriari S, Garcia D. **Meshfree simulations of ultrasound vector flow imaging using smoothed particle hydrodynamics**. *Phys Med Biol*, 2018;63:205011 (PDF)
4. Perrot V, Polichetti M, Varray F, Garcia D. **So you think you can DAS? A viewpoint on delay-and-sum beamforming**. *Ultrasonics*, 2021;111:106309 (PDF)

To better understand how PFIELD works, I also recommend that you read the first chapters of the book ***Fundamentals of Ultrasonic Phased Arrays*** from Lester W. Schmerr Jr. (Springer, see info). These chapters helped me a lot when I programmed PFIELD.



Each code has been used many times by students at CRCHUM (Montreal, Canada) and CREATIS (Lyon, France), and by myself, over the past few years. I believe I can claim that they are free of major theoretical or numerical errors. If you think I am wrong, please contact me and share your suggestions. Also, let me know if you find a bug, be it minor.

I hope that you will find MUST interesting for your research or lectures. The MUST toolbox will evolve. Functions will be added or updated. Update dates are included in each function. How fast MUST evolves will depend on its success and my agenda. If I deem certain requests to be of general interest, I may give them priority consideration. I thank you in advance for your interest and use of MUST.

The MUST toolbox is free and will remain so. If it helps you with your research projects, please cite the articles mentioned in this document and on the website. The MUST toolbox is distributed under the terms of the GNU Lesser General Public License v3.0 (LGPL v3).

Make the Most of MUST!

Damien Garcia
INSERM researcher, Lyon, June 2020 (update December 2021)

garcia.damien+MUST@gmail.com
www.biomecardio.com/MUST

The current version of the MUST toolbox contains the functions listed on this page and the following one. The next pages provide exhaustive descriptions and illustrative examples.

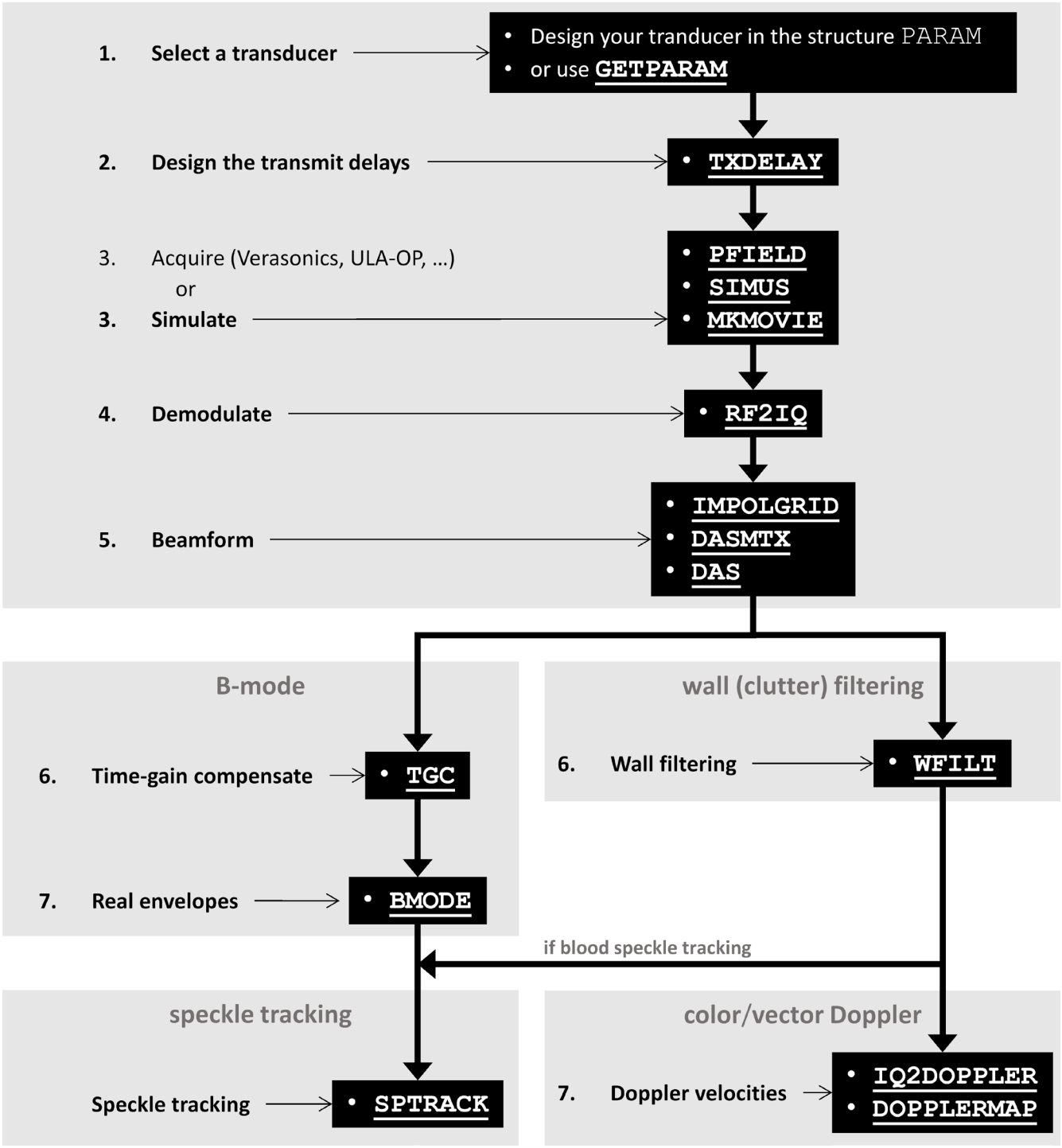
To get a quick overview of the MUST toolbox, I suggest you try the demos

1. **A QUICK START DEMO** – simulations, demodulation, beamforming, compounding
2. **DIVERGING-WAVE ECHOCARDIOGRAPHY** – simulations, beamforming, compounding

- **Functions of the MUST toolbox by alphabetical order**

bmode	Generate 8-bit B-mode images from I/Q signals
das	Delay-and-sum of RF and I/Q signals
dasmtx	Delay-and-sum matrix for beamforming with sparse matrix-vector multiplication
dopplermap	A color map for color Doppler
getparam	Get parameters of a uniform linear or convex array
getpulse	Get the one-way or two-way transmit pulse
impolgrid	Polar-type grid for ultrasound images
iq2doppler	Convert I/Q signals to Doppler velocities by using an auto-correlator
mkmovie	Make movie frames and animated GIF of wave propagation and backscattering
pfield	RMS acoustic pressure field of a linear or convex array
rf2iq	I/Q demodulation of RF data
simus	Simulation of ultrasound RF signals for a linear or convex array
sptrack	Motion estimation by speckle tracking
tgc	Time-gain compensation of RF or I/Q signals
txdelay	Generate transmit delays for a uniform rectilinear or curved array
wfilt	Wall (clutter) filtering based on polynomial regression, truncated DCT, or truncated SVD

This flowchart shows how some MUST functions should typically be used.



Acquire

txdelay	Generate transmit delays for a uniform rectilinear or curved array
rf2iq	I/Q demodulation of RF data
tgc	Time-gain compensation of RF or I/Q signals

Beamform

das	Delay-and-sum of RF and I/Q signals
dasmtx	Delay-and-sum matrix for beamforming with sparse matrix-vector multiplication

Color Doppler & Speckle tracking

wfilt	Wall (clutter) filtering based on polynomial regression, truncated DCT, or truncated SVD
iq2doppler	Convert I/Q signals to Doppler velocities by using an auto-correlator
sptrack	Motion estimation by speckle tracking

Simulate

getparam	Get parameters of a uniform linear or convex array
getpulse	Get the one-way or two-way transmit pulse
pfield	RMS acoustic pressure field of a linear or convex array
simus	Simulation of 2-D ultrasound RF signals for a linear or convex array

Display

impolgrid	Polar-type grid for ultrasound images
bmode	Generate 8-bit B-mode images from I/Q signals
dopplermap	A color map for color Doppler
mkmovie	Make movie frames and animated GIF of wave propagation and backscattering

BMODE B-mode image

BMODE creates B-mode images from I/Q signals.

Syntax

`BMODE(IQ,DR)` converts the I/Q signals (in `IQ`) to 8-bit log-compressed ultrasound images with a dynamic range `DR` (in dB). `IQ` is a complex whose real (imaginary) part contains the inphase (quadrature) component.

`BMODE(IQ)` uses `DR` = 40 dB;

Example: Speckle tracking in B-mode images

This example shows how to obtain the motion field of a rotating disk insonified with plane waves.

A rotating disk (diameter of 2 cm) was insonified by a series of 32 unsteered plane waves with a Verasonics scanner, and a linear transducer, at a PRF (pulse repetition frequency) of 10 kHz. The RF signals were **downsampled** at 4/3 × (5 MHz) = 6.66 MHz. The properties of the linear array were:

- 128 elements
- center frequency = 5 MHz
- pitch = 0.298 mm

Download the experimental RF data. The 3-D array RF contains 128 columns (as the transducer contained 128 elements), and its length is 32 in the third dimension (as 32 plane waves were transmitted).

```
load('PWI_disk.mat')
```

The structure `param` contains some experimental properties that are required for the following steps.

```
disp('''param'' is a structure whose fields are:')
disp(param)
```

```
'param' is a structure whose fields are:
           fs: 6.6667e+06
        pitch: 2.9800e-04
           fc: 5000000
            c: 1480
           t0: 9.9500e-06
          PRF: 10000
        width: 2.6200e-04
    Nelements: 128
      TXdelay: [1×128 double]
    bandwidth: 15
```

Demodulate the RF signals with RF2IQ.

```
IQ = rf2iq(RF,param);
```

Create a 2.5-cm-by-2.5-cm image grid.

```
dx = 1e-4; % grid x-step (in m)
dz = 1e-4; % grid z-step (in m)
[x,z] = meshgrid(-1.25e-2:dx:1.25e-2,1e-2:dz:3.5e-2);
```

Create a Delay-And-Sum DAS matrix with DASMTX.

```
param.fnumber = []; % an f-number will be determined by DASMTX
M = dasmtx(1i*size(IQ),x,z,param,'nearest');
```

Beamform the I/Q signals.

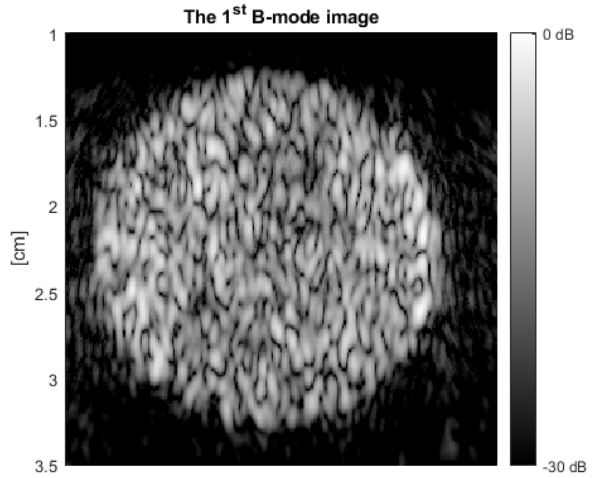
```
IQb = M*reshape(IQ,[],32);
IQb = reshape(IQb,[size(x) 32]);
```

Create the B-mode images with BMODE.

```
I = bmode(IQb,30);
```

Display the first ultrasound image.

```
image(x(1,:)*100,z(:,1)*100,I(:,:,1))
c = colorbar;
c.YTick = [0 255];
c.YTickLabel = {'-30 dB','0 dB'};
colormap gray
title('The 1^{st} B-mode image')
ylabel('[cm]')
axis equal tight ij
set(gca,'XColor','none','box','off')
```



Create an ROI.

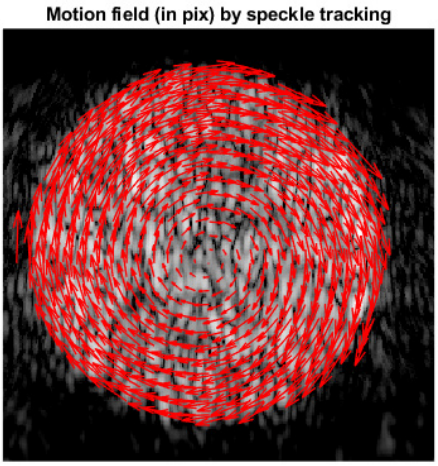
```
param.ROI = median(I,3)>64;
```

Track the speckles with SPTRACK.

```
param.winsize = [32 32; 24 24; 16 16]; % size of the subwindows
param.iminc = 4; % image increment
[Di,Dj,id,jd] = sptrack(I,param);
```

Display the motion field.

```
image(I(:,:,1))
colormap gray
hold on
h = quiver(jd,id,Dj,Di,3,'r');
set(h,'LineWidth',1)
hold off
title('Motion field (in pix) by speckle tracking')
axis equal off ij
```



See also

rf2iq, tgc, sptrack

Last update

2020/06

DAS Delay-And-Sum

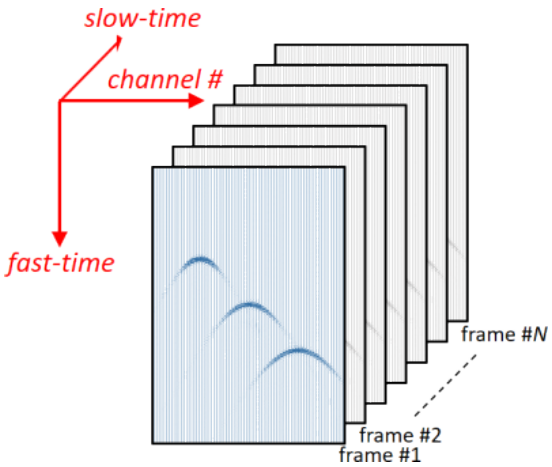
DAS beamforms the RF or I/Q signals.

Syntax

`BFSIG = DAS(SIG,X,Z,DELAYS,PARAM)` beamforms the RF or I/Q signals stored in the array `SIG`, and returns the beamformed signals BFSIG. The signals are beamformed at the points specified by `X` and `Z`.

▶ **TRY IT!** Enter `das` in the command window for an example.

`SIG` must be a 2-D or 3-D array. The first dimension (i.e. each column) corresponds to a single RF or I/Q signal over (fast-) time, with the FIRST COLUMN corresponding to the FIRST ELEMENT. Several 2-D signals can be stacked along the third dimension.



DELAYS are the transmit time delays (in s). The number of elements in DELAYS must be the number of elements in the array (which is equal to `size(SIG,2)`). If a sub-aperture was used during transmission, use `DELAYS(i) = NaN` if element # `i` of the linear array was off.

PARAM is a structure that contains the parameter values required for DAS beamforming (see below for details).

Note: `SIG` must be complex when beamforming I/Q data (i.e. `SIG = complex(I,Q) = I + 1i*Q`).

Note: DASMTX

DAS calls DASMTX. DASMTX creates a DAS matrix.

- **Beamforming using DAS**: `bfSIG = das(SIG,x,z,delays,param)`

- **Beamforming using DASMTX**:
```
M = dasmtx(size(SIG),x,z,delays,param);
bfSIG = M*SIG(:);
bfSIG = reshape(bfSIG,size(x));
```

Other syntaxes

`DAS(SIG,X,Z,PARAM)` uses `DELAYS = param.TXdelay`.

`DAS(...,METHOD)` specifies the interpolation method. The available methods are decribed in NOTE #3 below.

`[M,PARAM] = DAS(...)` also returns the structure `PARAM` with the default values.

The structure PARAM

`PARAM` is a structure that contains the following fields:

1. `PARAM.fs`: sampling frequency (in Hz, **required**)
2. `PARAM.pitch`: pitch of the transducer (in m, **required**)
3. `PARAM.fc`: center frequency (in Hz, **required** for I/Q signals)
4. `PARAM.radius`: radius of curvature (in m). The default is `Inf` (rectilinear array)
5. `PARAM.TXdelay`: transmission law delays (in s, required if the vector `DELAYS` is not given)
6. `PARAM.c`: longitudinal velocity (in m/s, default = 1540 m/s)
7. `PARAM.t0`: start time for reception (in s, default = 0 s)

A note on the *f*-number

The *f*-number is defined by the ratio (depth)/(aperture size). A null f-number, i.e. `PARAM.fnumber = 0`, means that the full aperture is used during DAS-beamforming. This might be a suboptimal strategy since the array elements have some directivity.

Use `PARAM.fnumber = []` to obtain an "optimal" *f*-number, which is estimated from the element directivity (and depends on fc, bandwidth, element width):

- `PARAM.fnumber`: reception f-number (default = 0, i.e. full aperture)
- `PARAM.width`: element width (in m, required if `PARAM.fnumber = []`)
- **or** `PARAM.kerf`: kerf width (in m, kerf = pitch-width, required if `PARAM.fnumber` = [])
- `PARAM.bandwidth`: pulse-echo 6dB fractional bandwidth (in %). The default is 60% (used only if `PARAM.fnumber = []`).

Advanced syntax for vector Doppler

`PARAM.RXangle`: reception angles (in rad, default = 0)

This option can be used for vector Doppler. Beamforming with at least two (sufficiently different) reception angles enables different Doppler directions and, in turn, vector Doppler.

Passive imaging

`PARAM.passive`: must be true for passive imaging (i.e. no transmit). The default is false.

Notes

- **NOTE #1: X- and Z-axes**

The `X` axis is PARALLEL to the transducer and points from the first (leftmost) element to the last (rightmost) element (`X = 0` at the CENTER of the transducer).

The `Z` axis is PERPENDICULAR to the transducer and points downward (`Z = 0` at the level of the transducer, `Z` increases as depth increases). See the figure below.

For a **convex** array, the `X` axis is parallel to the chord and `Z = 0` at the level of the chord.

- **NOTE #2:**

`DAS` uses a standard delay-and-sum. Phase rotations are included if I/Q (complex) signals are beamformed.

- **NOTE #3: interpolation methods**

By default `DAS` uses a linear interpolation. To specify the interpolation method, use `DAS(...,METHOD)`, with `METHOD` being:

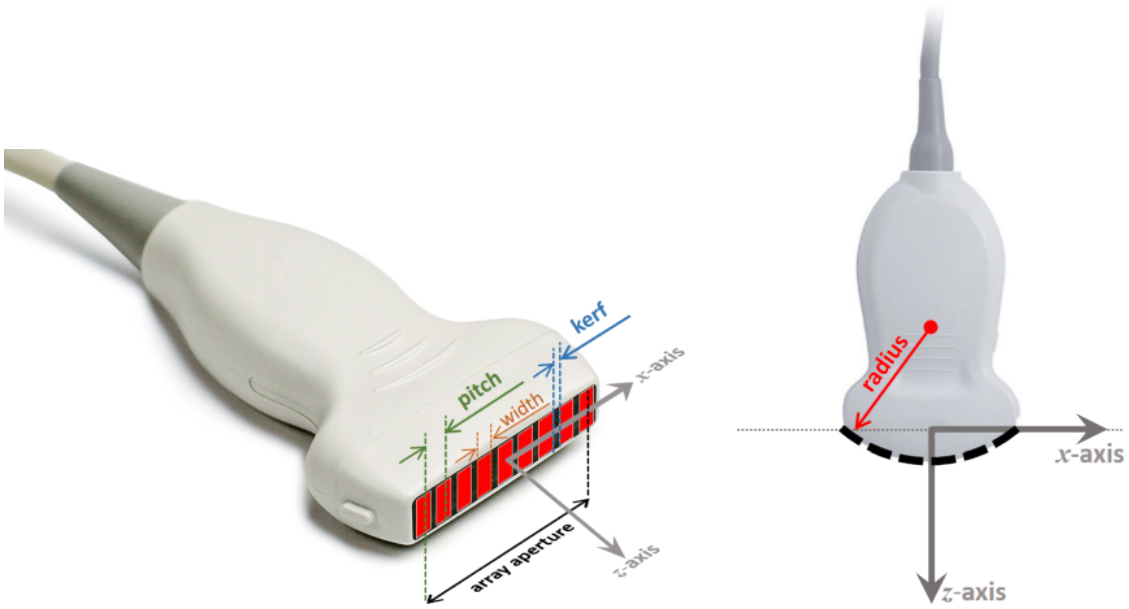
1. `'nearest'` : nearest neighbor interpolation
2. `'linear'` : (default) linear interpolation
3. `'quadratic'` : quadratic interpolation
4. `'lanczos3'` : 3-lobe Lanczos (windowed sinc) interpolation
5. `'5points'` : 5-point least-squares parabolic interpolation
6. `'lanczos5'` : 5-lobe Lanczos (windowed sinc) interpolation

The linear interpolation (it is a 2-point method) returns a matrix twice denser than the nearest-neighbor interpolation. It is 3, 4, 5, 6 times denser for `'quadratic'`, `'lanczos3'`, `'5points'`, `'lanczos5'`, respectively (they are 3-to-6-point methods).

Uniform linear array (ULA)

The ***pitch*** is defined as the center-to-center distance between two adjacent elements. The ***kerf*** width is the distance that separates two adjacent elements. They are constant for a uniform linear array (ULA).

Some functions of the MUST toolbox can also consider curved (convex) ULAs.



Example #1: Beamform signals from a phased array

This example shows how to simulate RF signals then beamform I/Q signals

Download the properties of a 2.7-MHz 64-element cardiac phased array in a structure `param` by using GETPARAM.

```
param = getparam('P4-2v');
```

Calculate the transmit delays to generate a non-tilted 60-degrees wide circular wave.

```
width = pi/3; % width angle in rad
txdel = txdelay(param,0,width); % in s
```

Create the scatterers of a 12-cm-by-12-cm phantom.

```
xs = rand(1,50000)*12e-2-6e-2;
zs = rand(1,50000)*12e-2;
idx = hypot(xs,zs-.05)<1e-2;
xs(idx) = []; % create a 1-cm-radius hole
zs(idx) = [];
RC = 3+randn(size(xs)); % reflection coefficients
```

Simulate RF signals by using SIMUS.

```
param.fs = 4*param.fc; % sampling frequency
RF = simus(xs,zs,RC,txdel,param);
```

Demodulate the RF signals.

```
IQ = rf2iq(RF,param);
```

Create a 256x256 80-degrees wide polar grid with IMPOLGRID.

```
[x,z] = impolgrid([256 256],10e-2,pi/3,param);
```

Beamform the I/Q signals.

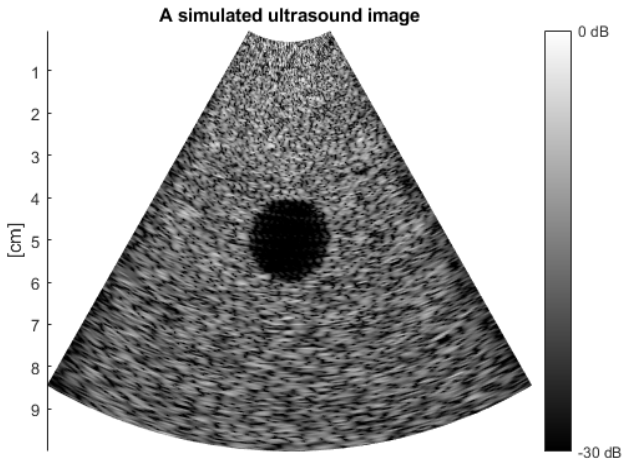
```
IQb = das(IQ,x,z,txdel,param);
```

Create the ultrasound image with BMODE.

```
B = bmode(IQb,30); % log-compressed B-mode image with a -30 dB range
```

Display the ultrasound image

```
pcolor(x*100,z*100,B)
c = colorbar;
c.YTick = [0 255];
c.YTickLabel = {'-30 dB','0 dB'};
colormap gray
title('A simulated ultrasound image')
ylabel('[cm]')
shading interp
axis equal ij tight
set(gca,'XColor','none','box','off')
```



Example #2: Beamform signals from a convex array

This example shows how to simulate RF signals with a convex array and how to beamform the I/Q signals with DAS.

Download the properties of a 2.7-MHz 64-element cardiac phased array in a structure `param` by using GETPARAM.

```
param = getparam('C5-2v');
```

Create the scatterers of a 17-cm-by-10-cm cyst phantom.

```
xs = rand(1,100000)*17e-2-8.5e-2;
zs = rand(1,100000)*10e-2;
idx = hypot(xs-.02,zs-.04)<1e-2; % create a 1-cm-radius hole
xs(idx) = [];
zs(idx) = [];
RC = 3+randn(size(xs)); % reflection coefficients
idx = hypot(xs+.02,zs-.06)<1e-2; % create a 1-cm-radius cyst
RC(idx) = RC(idx)+10;
```

Simulate RF signals by using SIMUS.

```
txdel = zeros(1,128); % transmit delays
param.fs = 4*param.fc; % sampling frequency
opt.ElementSplitting = 1; % to make simulations faster
RF = simus(xs,zs,RC,txdel,param,opt);
```

Demodulate the RF signals.

```
IQ = rf2iq(RF,param);
```

Create a 256x256 polar grid with IMPOLGRID.

```
[x,z] = impolgrid([256 256],10e-2,param);
```

It is recommended to use an "optimal" f-number.

```
param.fnumber = [];
```

Beamform the I/Q signals.

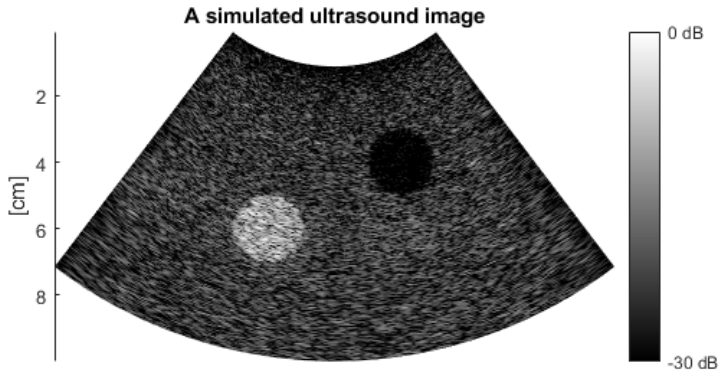
```
IQb = das(IQ,x,z,txdel,param);
```

Create the ultrasound image with BMODE.

```
B = bmode(IQb,30); % log-compressed B-mode image with a -30 dB range
```

Display the ultrasound image

```
pcolor(x*100,z*100,B)
c = colorbar;
c.YTick = [0 255];
c.YTickLabel = {'-30 dB','0 dB'};
colormap gray
title('A simulated ultrasound image')
ylabel('[cm]')
shading interp
axis equal ij tight
set(gca,'XColor','none','box','off')
```



See also

dasmtx, txdelay, simus

References

- Perrot V, Polichetti M, Varray F, Garcia D. **So you think you can DAS? A viewpoint on delay-and-sum beamforming**. *Ultrasonics*, 2021; 111:106309. **(PDF)**
- If you use `PARAM.RXangle` for vector Doppler: Madiena C, Faurie J, Porée J, Garcia D. **Color and vector flow imaging in parallel ultrasound with sub-Nyquist sampling**. *IEEE TUFFC*, 2018;65:795-802. **(PDF)**

Last update

2020/02

DASMTX Delay-And-Sum matrix

DASMTX returns a Delay-And-Sum matrix for linear beamforming.

Syntax

`M = DASMTX(SIG,X,Z,DELAYS,PARAM)` returns the `numel(X)` -by- `numel(SIG)` delay-and-sum DAS matrix. The matrix `M` can be used to beamform `SIG` (RF or I/Q signals) at the points specified by `X` and `Z`.

▶ **TRY IT!** Enter `dasmtx` in the command window for an example.

Because the signals in `SIG` are not required (only its size is needed) to create `M`, the following syntax is recommended:

```
M = DASMTX(size(SIG),X,Z,DELAYS,PARAM)
```

▶ **IMPORTANT!** – With this syntax, use `M = DASMTX(1i*size(SIG),...)` to return a ***complex* DAS matrix for I/Q data**.

`DELAYS` are the transmit time delays (in s). The number of elements in `DELAYS` must be the number of elements in the array (which is equal to `size(SIG,2)`). If a sub-aperture was used during transmission, use `DELAYS(i) = NaN` if element # `i` of the linear array was off.

`PARAM` is a structure that contains the parameter values required for DAS beamforming (see below for details).

DASMTX versus DAS

DASMTX is also called by DAS. DASMTX thus returns the same beamformed results as DAS.

- **Beamforming using DAS**:

```
bfSIG = das(SIG,x,z,delays,param)
```

- **Beamforming using DASMTX**:

```
M = dasmtx(size(SIG),x,z,delays,param);
bfSIG = M*SIG(:);
bfSIG = reshape(bfSIG,size(x));
```

`M` is a large sparse matrix. Computing `M` can be much more cost-effective than using DAS if you need to beamform several `SIG` matrices, because `M` needs to be determined only once.

Let us consider that a series `SIG{1}`, `SIG{2}` ... `SIG{N}` of ultrasound matrices have been generated by sending similar wavefronts with the same ultrasound array. These signals `SIG{i}` are stacked in a 3D array `sig3D` so that `sig3D(:,:,i) = SIG{i}`. To beamform these data with a delay-and-sum approach, the following can be used.

To obtain the DAS matrix, use `dasmtx`:

`M = dastmtx([size(sig3D,1) size(sig3D,2)],x,z,delays,param)` **or**

`M = dastmtx(1i*[size(sig3D,1) size(sig3D,2)],...)` **for I/Q data**.

The beamformed signals are then calculated by a matrix multiplication:

```
bfSIG3D = M*reshape(sig3D,[],size(sig3D,3))
```

It is necessary to reshape the beamformed matrix array:

```
bfSIG3D = reshape(bfSIG3D,size(x,1),size(x,2),[])
```

You can also consider saving the DAS matrix `M` in a MAT file and loading it when needed. The previous syntax is generally much faster than:

```
for k = 1:N, bfSIG{k} = das(SIG{k},x,z,delays,param); end
```

Other syntaxes

`DASMTX(SIG,X,Z,PARAM)` uses `DELAYS = param.TXdelay`.

`DASMTX(...,METHOD)` specifies the interpolation method. The available methods are decribed in NOTE #3 below.

`[M,PARAM] = DASMTX(...)` also returns the structure `PARAM` with the default values.

The structure PARAM

`PARAM` is a structure that contains the following fields:

1. `PARAM.fs`: sampling frequency (in Hz, **required**)
2. `PARAM.pitch`: pitch of the transducer (in m, **required**)
3. `PARAM.fc`: center frequency (in Hz, **required** for I/Q signals)
4. `PARAM.radius`: radius of curvature (in m). The default is `Inf` (rectilinear array)
5. `PARAM.TXdelay`: transmission law delays (in s, required if the vector `DELAYS` is not given)
6. `PARAM.c`: longitudinal velocity (in m/s, default = 1540 m/s)
7. `PARAM.t0`: start time for reception (in s, default = 0 s)

A note on the f-number

The f-number is defined by the ratio (depth)/(aperture size). A null f-number, i.e. `PARAM.fnumber = 0`, means that the full aperture is used during DAS-beamforming. This might be a suboptimal strategy since the array elements have some directivity.

Use PARAM.fnumber = [] to obtain an "optimal" f-number, which is estimated from the element directivity (and depends on fc, bandwidth, element width):

- PARAM.fnumber: reception f-number (default = 0, i.e. full aperture)
- PARAM.width: element width (in m, required if PARAM.fnumber = [])
- **or** PARAM.kerf: kerf width (in m, kerf = pitch-width, required if PARAM.fnumber = [])
- PARAM.bandwidth: pulse-echo 6dB fractional bandwidth (in %). The default is 60% (used only if PARAM.fnumber = []).

Advanced syntax for vector Doppler

PARAM.RXangle: reception angles (in rad, default = 0)

This option can be used for vector Doppler. Beamforming with at least two (sufficiently different) reception angles enables different Doppler directions and, in turn, vector Doppler.

Notes

- **NOTE #1: X- and Z-axes**

The X axis is PARALLEL to the transducer and points from the first (leftmost) element to the last (rightmost) element (X = 0 at the CENTER of the transducer).

The Z axis is PERPENDICULAR to the transducer and points downward (Z = 0 at the level of the transducer, Z increases as depth increases). See the figure below.

For a **convex** array, the X axis is parallel to the chord and Z = 0 at the level of the chord.

- **NOTE #2:**

DASMTX uses a standard delay-and-sum. It is a linear operator. Phase rotations are included if I/Q (complex) signals are beamformed.

- **NOTE #3: interpolation methods**

By default DASMTX uses a linear interpolation to generate the DAS matrix. To specify the interpolation method, use DASMTX(...,METHOD), with METHOD being:

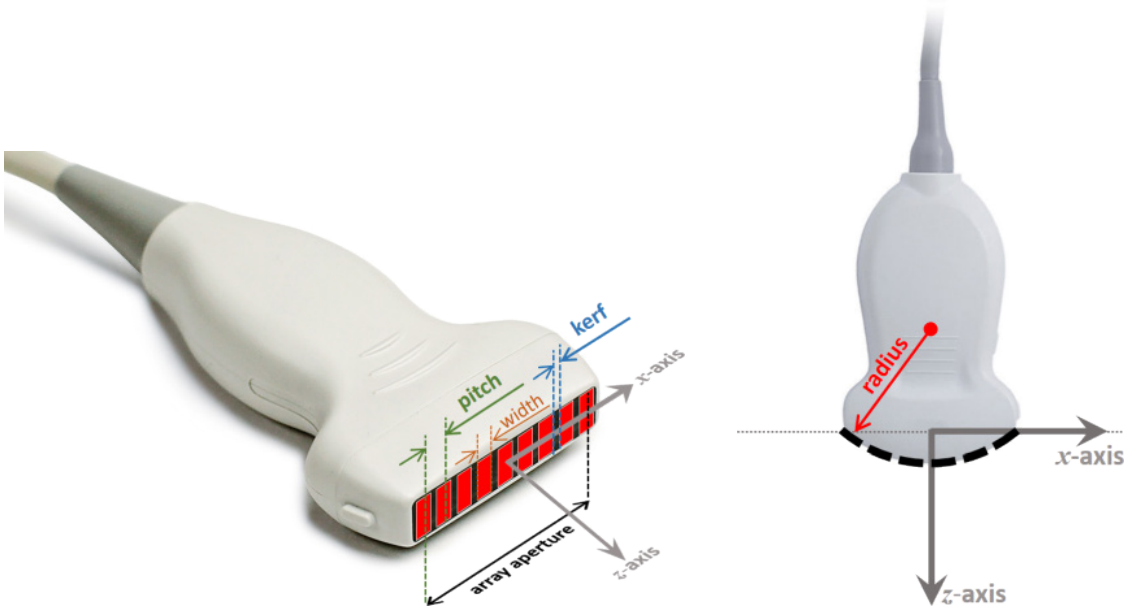
1. 'nearest' : nearest neighbor interpolation
2. 'linear' : (default) linear interpolation
3. 'quadratic' : quadratic interpolation
4. 'lanczos3' : 3-lobe Lanczos (windowed sinc) interpolation
5. '5points' : 5-point least-squares parabolic interpolation
6. 'lanczos5' : 5-lobe Lanczos (windowed sinc) interpolation

The linear interpolation (it is a 2-point method) returns a matrix twice denser than the nearest-neighbor interpolation. It is 3, 4, 5, 6 times denser for `'quadratic'`, `'lanczos3'`, `'5points'`, `'lanczos5'`, respectively (they are 3-to-6-point methods).

Uniform linear array (ULA)

The ***pitch*** is defined as the center-to-center distance between two adjacent elements. The ***kerf*** width is the distance that separates two adjacent elements. They are constant for a uniform linear array (ULA).

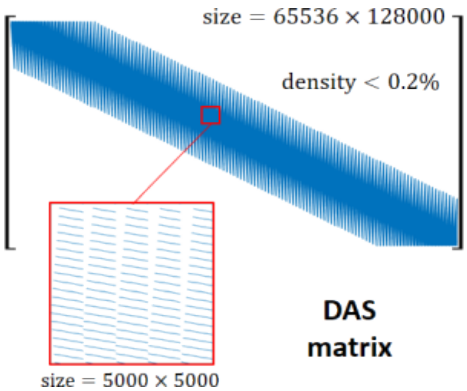
Some functions of the MUST toolbox can also consider curved (convex) ULAs.



The DAS matrix

The DAS matrix is very sparse. It is complex when beamforming I/Q signals, which should be the preferred approach.

Here is an example of a DAS matrix:



Example #1: Beamform signals from a phased array

This example shows how to simulate RF signals then beamform I/Q signals

Download the properties of a 2.7-MHz 64-element cardiac phased array in a structure `param` by using `GETPARAM`.

```
param = getparam('P4-2v');
```

Calculate the transmit delays to generate a non-tilted 80-degrees wide circular wave.

```
width = 60/180*pi; % width angle in rad
txdel = txdelay(param,0,width); % in s
```

Create the scatterers of a 12-cm-by-12-cm phantom.

```
xs = rand(1,50000)*12e-2-6e-2;
zs = rand(1,50000)*12e-2;
idx = hypot(xs,zs-.05)<1e-2;
xs(idx) = []; % create a 1-cm-radius hole
zs(idx) = [];
RC = 3+randn(size(xs)); % reflection coefficients
```

Simulate RF signals by using `SIMUS`.

```
param.fs = 4*param.fc; % sampling frequency
RF = simus(xs,zs,RC,txdel,param);
```

Demodulate the RF signals.

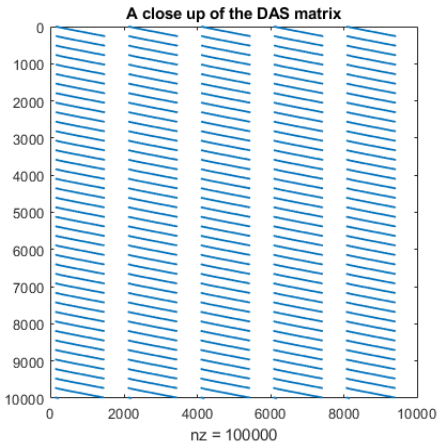
```
IQ = rf2iq(RF,param);
```

Create a 256x256 80-degrees wide polar grid with `IMPOLGRID`.

```
[x,z] = impolgrid([256 256],10e-2,pi/3,param);
```

Create the DAS matrix.

```
Mdas = dasmtx(1i*size(IQ),x,z,txdel,param);
spy(Mdas(1:10000,1:10000))
title('A close up of the DAS matrix')
```



Beamform the I/Q signals.

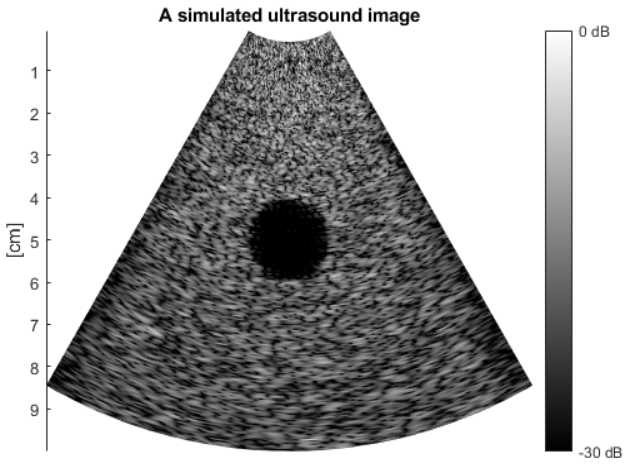
```
IQb = Mdas*IQ(:);
IQb = reshape(IQb,size(x));
```

Create the ultrasound image with BMODE.

```
B = bmode(IQb,30); % log-compressed B-mode image with a -30 dB range
```

Display the ultrasound image

```
pcolor(x*100,z*100,B)
c = colorbar;
c.YTick = [0 255];
c.YTickLabel = {'-30 dB','0 dB'};
colormap gray
title('A simulated ultrasound image')
ylabel('[cm]')
shading interp
axis equal ij tight
set(gca,'XColor','none','box','off')
```



Example #2: Beamform signals from a convex array

This example shows how to simulate RF signals with a convex array and how to beamform the I/Q signals with DAS.

Download the properties of a 2.7-MHz 64-element cardiac phased array in a structure `param` by using GETPARAM.

```
param = getparam('C5-2v');
```

Create the scatterers of a 17-cm-by-10-cm cyst phantom.

```
xs = rand(1,100000)*17e-2-8.5e-2;
zs = rand(1,100000)*10e-2;
idx = hypot(xs-.02,zs-.04)<1e-2; % create a 1-cm-radius hole
xs(idx) = [];
zs(idx) = [];
RC = 3+randn(size(xs)); % reflection coefficients
idx = hypot(xs+.02,zs-.06)<1e-2; % create a 1-cm-radius cyst
RC(idx) = RC(idx)+10;
```

Simulate RF signals by using SIMUS.

```
txdel = zeros(1,128); % transmit delays
param.fs = 4*param.fc; % sampling frequency
opt.ElementSplitting = 1; % to make simulations faster
RF = simus(xs,zs,RC,txdel,param,opt);
```

Demodulate the RF signals.

```
IQ = rf2iq(RF,param);
```

Create a 256x256 polar grid with IMPOLGRID.

```
[x,z] = impolgrid([256 256],10e-2,param);
```

It is recommended to use an "optimal" f-number.

```
param.fnumber = [];
```

Create the DAS matrix.

```
Mdas = dasmtx(1i*size(IQ),x,z,txdel,param);
```

Beamform the I/Q signals.

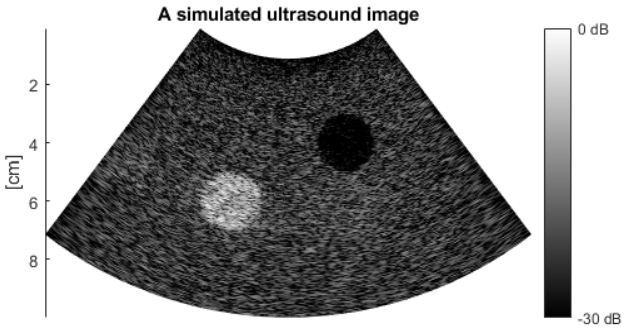
```
IQb = Mdas*IQ(:);
IQb = reshape(IQb,size(x));
```

Create the ultrasound image with BMODE.

```
B = bmode(IQb,30); % log-compressed B-mode image with a -30 dB range
```

Display the ultrasound image

```
pcolor(x*100,z*100,B)
c = colorbar;
c.YTick = [0 255];
c.YTickLabel = {'-30 dB','0 dB'};
colormap gray
title('A simulated ultrasound image')
ylabel('[cm]')
shading interp
axis equal ij tight
set(gca,'XColor','none','box','off')
```



See also

das, txdelay, simus

References

- Perrot V, Polichetti M, Varray F, Garcia D. **So you think you can DAS? A viewpoint on delay-and-sum beamforming**. *Ultrasonics*, 2021; 111:106309. **(PDF)**
- If you use `PARAM.RXangle` for vector Doppler: Madiena C, Faurie J, Porée J, Garcia D. **Color and vector flow imaging in parallel ultrasound with sub-Nyquist sampling**. *IEEE TUFFC*, 2018;65:795-802. **(PDF)**

Last update

2020/10

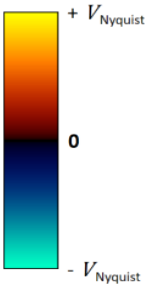
DOPPLERMAP Color map for color Doppler

DOPPLERMAP returns a matrix containing the Doppler colormap.

Syntax

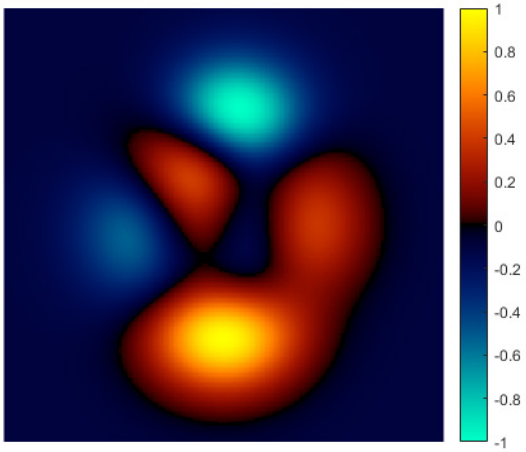
DOPPLERMAP(M) returns an M-by-3 matrix containing a typical color map used by ultrasound scanners in the color Doppler mode.

DOPPLERMAP (no input argument) is the same size as the current figure's color map.



Example

```
P = peaks(256);
P = 2*rescale(P)-1;
imagesc(P)
axis square off
colormap dopplermap
colorbar
```



See also

iq2doppler

Last update

2020/06

GETPARAM Parameters of an ultrasound array

GETPARAM returns parameters of a uniform linear or convex array.

Syntax

PARAM = GETPARAM opens a dialog box which allows you to select a transducer whose parameters are returned in PARAM.

PARAM = GETPARAM(PROBE), where PROBE is a string, returns the prameters of the transducer given by PROBE.

The structure PARAM is used in several functions of MUST (Matlab UltraSound Toolbox). The structure returned by GETPARAM contains only the fields that describe a transducer. Other fields may be required in some MUST functions.

PROBE can be one of the following:

1. `'L11-5v'` (128-element, 7.6-MHz linear array)
2. `'L12-3v'` (192-element, 7.5-MHz linear array)
3. `'C5-2v'` (128-element, 3.6-MHz convex array)
4. `'P4-2v'` (64-element, 2.7-MHz phased array)

These are the Verasonics' transducers. Feel free to modify the code and complete this list for your own use.

The structure PARAM returned by GETPARAM contains the following fields:

1. PARAM.`Nelements`: number of elements in the transducer array
2. PARAM.`fc`: center frequency (in Hz)
3. PARAM.`pitch`: element pitch (in m)
4. PARAM.`width`: element width (in m)
5. PARAM.`kerf`: kerf width (in m)
6. PARAM.`bandwidth`: 6-dB pulse-echo fractional bandwidth (in %)
7. PARAM.`radius`: radius of curvature (in m, Inf for a linear array)
8. PARAM.`focus`: elevation focus (in m)
9. PARAM.`height`: element height (in m)

PARAM fields from GETPARAM

The PARAM fields returned by GETPARAM have the following values:

PARAM fields	units	'L11-5V'	'L12-3V'	'C5-2V'	'P4-2V'
Nelements	-	128	192	128	64
fc	Hz	7.6e6	7.5e6	3.6e6	2.7e6
pitch	m	300e-6	200e-6	508e-6	300e-6
kerf	m	30e-6	30e-6	48e-6	50e-6
width	m	270e-6	170e-6	460e-6	250e-6
bandwidth	%	77	93	80	74
radius	m	Inf	Inf	49.6e-3	Inf
focus	m	18e-3	20e-3	60e-3	60e-3
height	m	5e-3	5e-3	13.5e-3	14e-3

Example: Generate a focused pressure field

This example shows how to simulate a focused pressure field with a cardiac phased-array.

Download the properties of a 2.7-MHz 64-element cardiac phased array in a structure `param` by using GETPARAM.

```
param = getparam('P4-2v');
disp('The properties of the selected probe are:')
disp(param)
```

```
The properties of the selected probe are:
           fc: 2700000
         kerf: 5.0000e-05
        width: 2.5000e-04
        pitch: 3.0000e-04
    Nelements: 64
    bandwidth: 74
       radius: Inf
       height: 0.0140
        focus: 0.0600
```

`param.focusm` is the ELEVATION focus. Define the (azimuthal) focus position.

```
x0 = 2e-2; z0 = 5e-2; % in m
```

Calculate the transmit delays with TXDELAY.

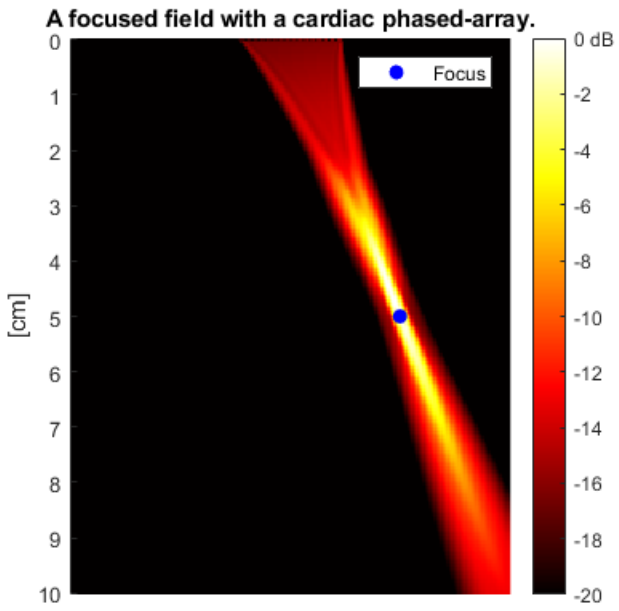
```
dels = txdelay(x0,z0,param);
```

Define a 200-by-200 image grid.

```
x = linspace(-4e-2,4e-2,200);
z = linspace(0,10e-2,200);
[x,z] = meshgrid(x,z);
y = zeros(size(x));
```

Simulate the acoustic pressure field and display it.

```
P = pfield(x,y,z,dels,param);
imagesc(x(1,:)*1e2,z(:,1)*1e2,20*log10(P/max(P(:))))
hold on
plot(x0*1e2,z0*1e2,'bo','MarkerFaceColor','b')
hold off
colormap hot
axis equal tight
caxis([-20 0])
c = colorbar;
c.YTickLabel{end} = '0 dB';
ylabel('[cm]')
title('A focused field with a cardiac phased-array.')
legend('Focus')
set(gca,'XColor','none','box','off')
```



See also

txdelay, pfield, simus, getpulse

Last update

2020/07

GETPULSE Get the transmit pulse

GETPULSE returns the one-way or two-way transmit pulse.

Syntax

`PULSE = GETPULSE(PARAM,WAY)` returns the one-way or two-way transmit pulse with a time sampling of 1 nanosecond. Use `WAY = 1` to get the one-way pulse, or `WAY = 2` to obtain the two-way (pulse-echo) pulse.

`PULSE = GETPULSE(PARAM)` uses `WAY = 1`.

`[PULSE,t] = GETPULSE(...)` also returns the time vector.

The structure PARAM

PARAM is a structure that contains the following fields:

1. `PARAM.fc`: center frequency (in Hz, **required**)
2. `PARAM.bandwidth`: pulse-echo 6dB fractional bandwidth (in %). The default is 75%.
3. `PARAM.TXnow`: number of wavelengths of the TX pulse (default: 1)
4. `PARAM.TXfreqsweep`: frequency sweep for a linear chirp (default: [])

Example #1: Get the one-way pulse of a phased-array probe

This example shows the default transmitted (one way) pulse that is simulated with the 'P4-2v' phased-array probe.

Get the parameters of the 2.7-MHz cardiac phased array with GETPARAM.

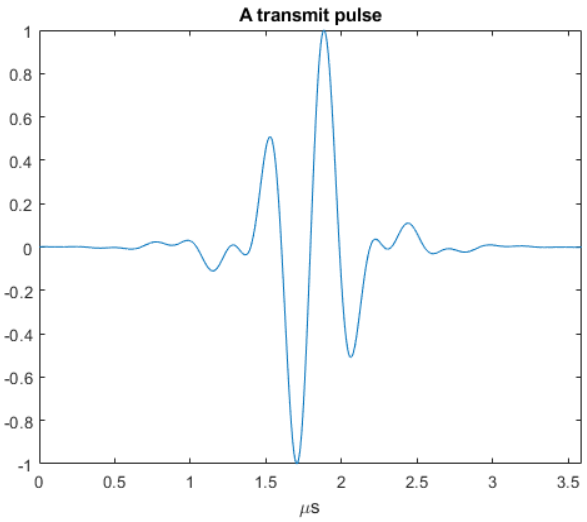
```
param = getparam('P4-2v');
```

Get the one-way transmit pulse with GETPULSE.

```
[pulse,t] = getpulse(param);
```

Display the pulse.

```
plot(t*1e6,pulse)
xlabel('{\mu}s')
axis tight
title('A transmit pulse')
```



Example #2: Check the pulse with a linear chirp

This example shows a linear chirp transmitted by a simulated linear array.

Get the parameters of the 7.6-MHz linear array with GETPARAM.

```
param = getparam('L11-5v');
```

Modify the fractional bandwidth.

```
param.bandwidth = 120;
```

Define the properties of the chirp

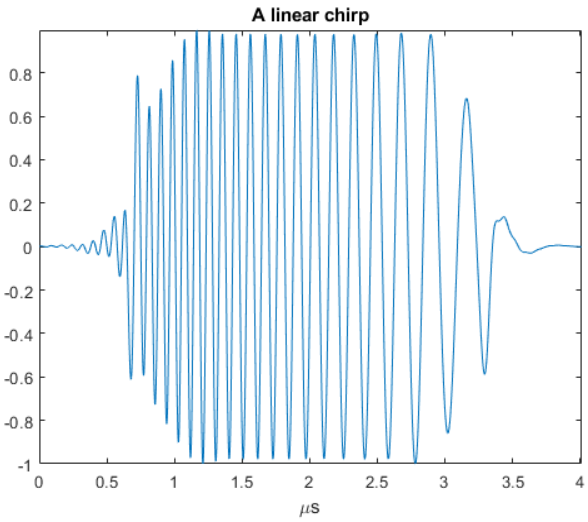
```
param.TXnow = 20;
param.TXfreqsweep = 10e6;
```

Get the one-way transmit pulse with GETPULSE.

```
[pulse,t] = getpulse(param);
```

Display the pulse.

```
plot(t*1e6,pulse)
xlabel('{\mu}s')
axis tight
title('A linear chirp')
```



See also

pfield, simus, getparam

Last update

2020/12

IMPOLGRID Polar-type grid for ultrasound images

IMPOLGRID returns a polar-type (fan-type) grid expressed in Cartesian coordinates. This is the "natural" grid (before scan-conversion) used when beamforming signals obtained with a cardiac phased array or a convex array.

Syntax

`[X,Z] = IMPOLGRID(SIZ,ZMAX,WIDTH,PARAM)` returns the `X,Z` coordinates of the fan-type grid of size `SIZ` and angular width `WIDTH` (in rad) for a phased array described by `PARAM`. The maximal `Z` (maximal depth) is `ZMAX`.

`[X,Z] = IMPOLGRID(SIZ,ZMAX,PARAM)` returns the `X,Z` coordinates of the fan-type grid of size `SIZ` and angular width `WIDTH` (in rad) for a convex array described by `PARAM`. For a convex array, `PARAM.radius` is not `Inf`. The maximal `Z` (maximal depth) is `ZMAX`.

`[X,Z,Z0] = IMPOLGRID(...)` also returns the z-coordinate of the grid origin. Note that `X0 = 0`.

Units: `X,Z,Z0` are in m. `WIDTH` must be in rad.

`PARAM` is a structure which must contain the following fields:

1. `PARAM.pitch`: pitch of the linear array (in m, **required**)
2. `PARAM.Nelements`: number of elements in the transducer array, (**required**)
3. `PARAM.radius`: radius of curvature (in m, default = `Inf`, linear array)

Example #1: Low-resolution grids (for clarity purpose)

This example shows how polar-type grid look like for a phased array or convex array.

Choose a 2.7-MHz 64-element cardiac phased array with `GETPARAM`.

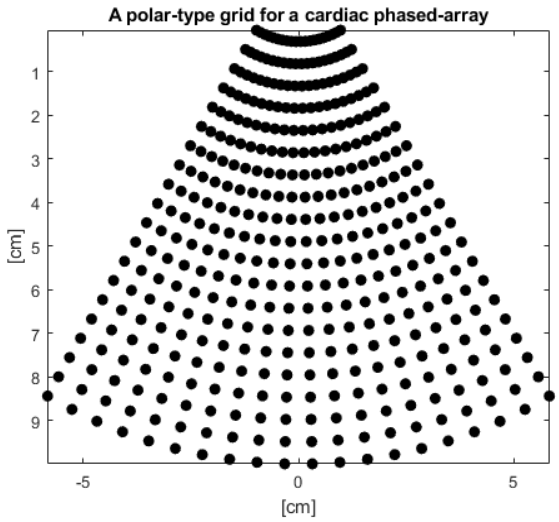
```
param = getparam('P4-2v');
```

Create a 60-degrees wide polar-type 20-by-20 grid.

```
[x,z] = impolgrid([20 20],10e-2,pi/3,param);
```

Display the grid.

```
plot(x*1e2,z*1e2,'ko','MarkerFaceColor','k')
xlabel('[cm]')
ylabel('[cm]')
axis equal ij tight
title('A polar-type grid for a cardiac phased-array')
```



Now choose a 3.6-MHz 128-element curved linear array.

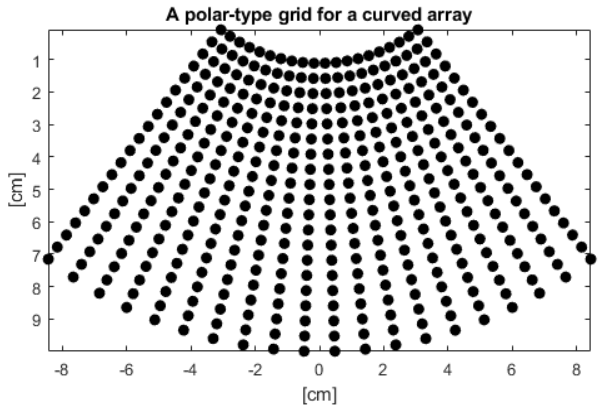
```
param = getparam('C5-2v');
```

Create a polar-type 20-by-20 grid.

```
[x,z] = impolgrid([20 20],10e-2,param);
```

Display the grid.

```
plot(x*1e2,z*1e2,'ko','MarkerFaceColor','k')
xlabel('[cm]')
ylabel('[cm]')
axis equal ij tight
title('A polar-type grid for a curved array')
```



Example #2: A focused pressure field with a phased-array

This example shows how to simulate a pressure field on a polar-type grid.

Choose a 2.7-MHz 64-element cardiac phased array with GETPARAM.

```
param = getparam('P4-2v');
```

Define a focus position.

```
xf = 2e-2; zf = 5e-2;
```

Calculate the transmit delays with TXDELAY.

```
txdel = txdelay(xf,zf,param);
```

Create a 60-degrees wide 10-cm deep 100-by-50 grid with IMPOLGRID.

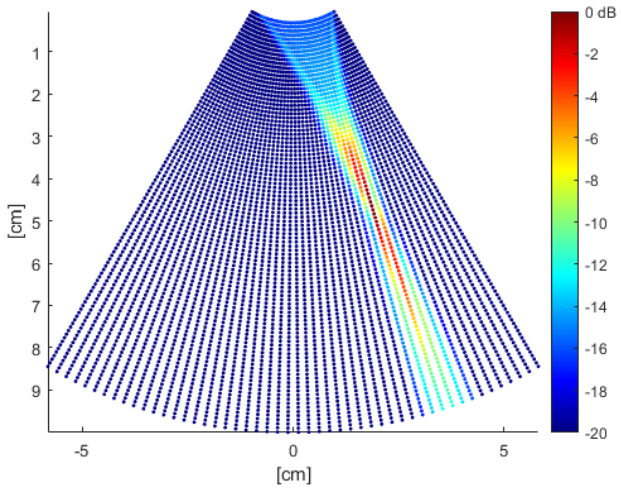
```
[x,z] = impolgrid([100 50],10e-2,pi/3,param);
```

Simulate the RMS pressure field with PFIELD.

```
y = zeros(size(x));
P = pfield(x,y,z,txdel,param);
```

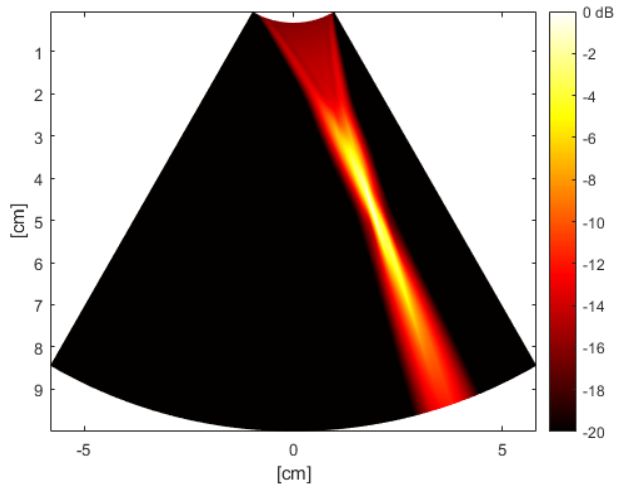
Display a scatter plot of the pressure field.

```
scatter(x(:)*1e2,z(:)*1e2,5,20*log10(P(:)/max(P(:))),'filled')
colormap jet
caxis([-20 0])
c = colorbar;
c.YTickLabel{end} = '0 dB';
axis equal ij tight
xlabel('[cm]')
ylabel('[cm]')
```



Display the pressure field.

```
pcolor(x*1e2,z*1e2,20*log10(P/max(P(:))))
shading interp
colormap hot
axis equal ij tight
xlabel('[cm]')
ylabel('[cm]')
caxis([-20 0])
c = colorbar;
c.YTickLabel{end} = '0 dB';
```



See also

das, dasmtx, pfield

Last update

2020/06

IQ2DOPPLER I/Q data to color Doppler

IQ2DOPPLER converts I/Q data to Doppler velocities.

Syntax

`VD = IQ2DOPPLER(IQ,PARAM)` returns the Doppler velocities from the I/Q time series using a slow-time autocorrelator.

`PARAM` is a structure that must contain the following fields:

1. `PARAM.fc`: center frequency (in Hz, **required**)
2. `PARAM.c`: longitudinal velocity (in m/s, default = 1540 m/s)
3. `PARAM.PRF` (in Hz) or `PARAM.PRP` (in s): pulse repetition frequency or period (**required**)

`VD = IQ2DOPPLER(IQ,PARAM,M)`:

- If `M` is of a two-component vector `[M(1) M(2)]`, the output Doppler velocity is estimated from the `M(1)` -by- `M(2)` neighborhood around the corresponding pixel.
- If `M` is a scalar, then an `M` -by- `M` neighborhood is used.
- If `M` is empty, then `M = 1`. This is the default.

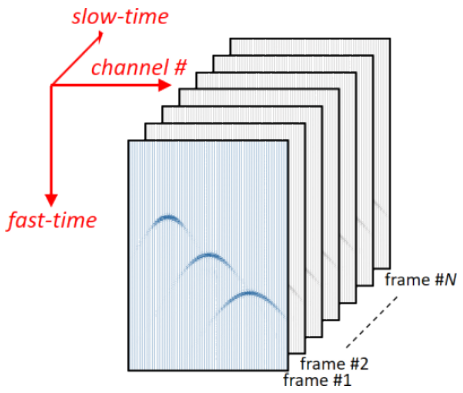
`VD = IQ2DOPPLER(IQ,PARAM,M,LAG)` uses a lag of value `LAG` in the autocorrelator. By default, `LAG = 1`.

`[VD,VarD] = IQ2DOPPLER(...)` also returns an estimated Doppler variance.

Important note

`IQ` must be a **3-D complex array**, where the real and imaginary parts correspond to the in-phase and quadrature components, respectively. The third dimension corresponds to the slow-time axis.

`IQ2DOPPLER` uses a full ensemble length to perform the auto-correlation, i.e. ensemble length (or packet size) = `size(IQ,3)`.



Example #1: Color Doppler with plane wave imaging

This example shows how to obtain color Doppler velocities from RF signals acquired by plane wave imaging.

A rotating disk (diameter of 2 cm) was insonified by a series of 32 unsteered plane waves with a Verasonics scanner, and a linear transducer, at a PRF (pulse repetition frequency) of 10 kHz. The RF signals were **downsampled** at 4/3 × (5 MHz) = 6.66 MHz. The properties of the linear array were:

- 128 elements
- center frequency = 5 MHz
- pitch = 0.298 mm

Download the experimental RF data. The 3-D array RF contains 128 columns (as the transducer contained 128 elements), and its length is 32 in the third dimension (as 32 plane waves were transmitted).

```
load('PWI_disk.mat')
```

The structure `param` contains some experimental properties that are required for the following steps.

```
disp('''param'' is a structure whose fields are:')
disp(param)
```

```
'param' is a structure whose fields are:
           fs: 6.6667e+06
        pitch: 2.9800e-04
           fc: 5000000
            c: 1480
           t0: 9.9500e-06
          PRF: 10000
        width: 2.6200e-04
    Nelements: 128
      TXdelay: [1×128 double]
    bandwidth: 15
```

Demodulate the RF signals with `RF2IQ`.

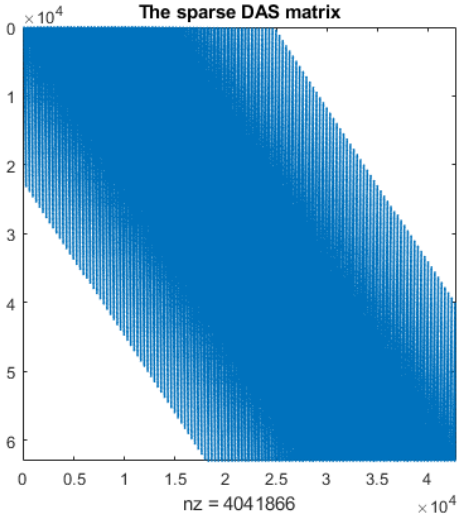
```
IQ = rf2iq(RF,param);
```

Create a 2.5-cm-by-2.5-cm image grid.

```
dx = 1e-4; % grid x-step (in m)
dz = 1e-4; % grid z-step (in m)
[x,z] = meshgrid(-1.25e-2:dx:1.25e-2,1e-2:dz:3.5e-2);
```

Create a Delay-And-Sum DAS matrix with `DASMTX`.

```
param.fnumber = []; % an f-number will be determined by DASMTX
M = dasmtx(1i*size(IQ),x,z,param,'nearest');
spy(M)
axis square
title('The sparse DAS matrix')
```



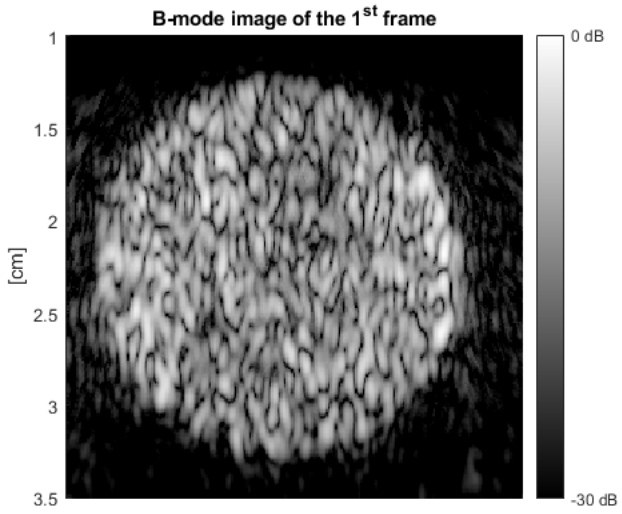
Beamform the I/Q signals.

The 32 I/Q series can be beamformed simultaneously with the DAS matrix.

```
IQb = M*reshape(IQ,[],32);
IQb = reshape(IQb,[size(x) 32]);
```

Create and display an ultrasound image with BMODE.

```
B = bmode(IQb(:,:,1),30); % log-compressed image

imagesc(x(1,:)*100,z(:,1)*100,B)
c = colorbar;
c.YTick = [0 255];
c.YTickLabel = {'-30 dB','0 dB'};
colormap gray
title('B-mode image of the 1^{st} frame')
ylabel('[cm]')
shading interp
axis equal ij tight
set(gca,'XColor','none','box','off')
```

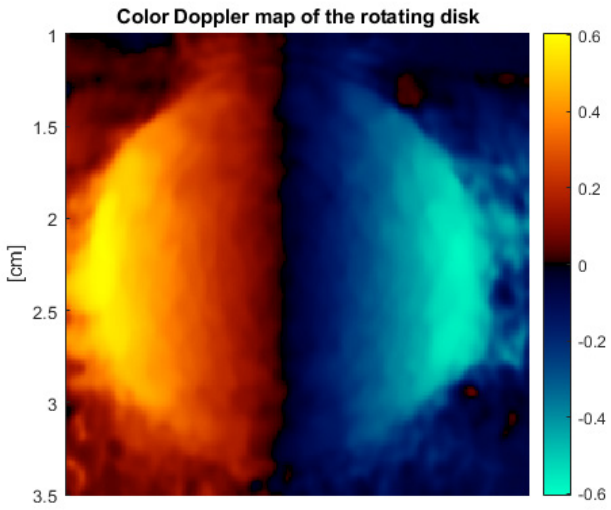


Calculate the Doppler velocities by using IQ2DOPPLER.

```
Vd = iq2doppler(IQb,param,[11 11]);
```

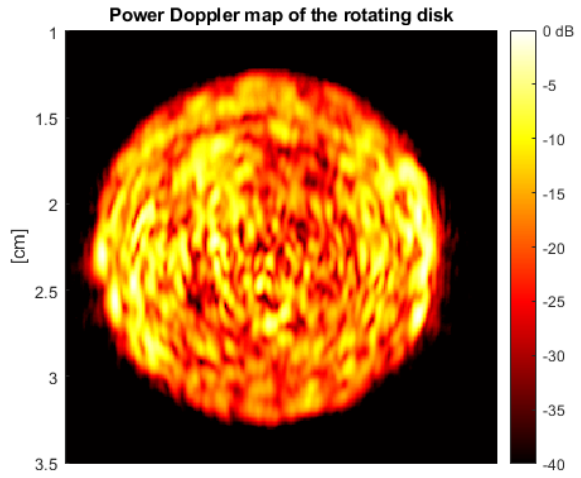
Display the color Doppler map

```
imagesc(x(1,:)*100,z(:,1)*100,Vd)
colormap dopplermap
colorbar
caxis([-1 1]*max(abs(Vd(:))))
title('Color Doppler map of the rotating disk')
ylabel('[cm]')
axis equal ij tight
set(gca,'XColor','none','box','off')
```



Display the power Doppler map.

```
P = sum(abs(IQb).^2,3); % power Doppler
P = 20*log10(P/max(P(:))); % log-compressed power Doppler
imagesc(x(1,:)*100,z(:,1)*100,P)
caxis([-40 0]) % dynamic range = [-40,0] dB
c = colorbar;
c.YTickLabel{end} = '0 dB';
colormap hot
title('Power Doppler map of the rotating disk')
ylabel('[cm]')
axis equal ij tight
set(gca,'XColor','none','box','off')
```

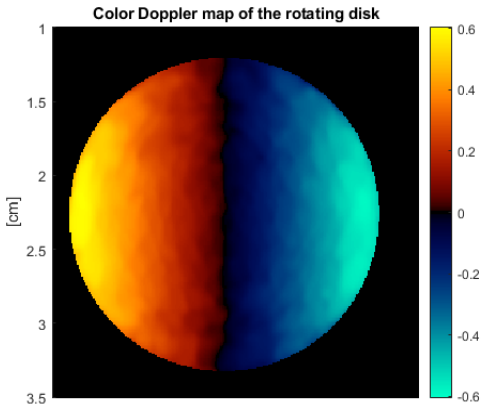


Create a ROI to get a masked color Doppler.

```
idx = P>-40;
xc = mean(x(idx)); zc = mean(z(idx)); % center of the disk
R = sqrt(nnz(idx)*dx*dz/pi); % estimated radius
roi = hypot(x-xc,z-zc)<R; % binary mask
```

Display a nicer color Doppler.

```
Vd(~roi) = 0; % discard the out-of-ROI velocities
imagesc(x(1,:)*100,z(:,1)*100,Vd)
colormap dopplermap
colorbar
caxis([-1 1]*max(abs(Vd(:))))
title('Color Doppler map of the rotating disk')
ylabel('[cm]')
axis equal ij tight
set(gca,'XColor','none','box','off')
```



Example #2: Vector Doppler with plane wave imaging

This example shows how to make vector Doppler imaging from RF signals acquired by plane wave imaging.

A rotating disk (diameter of 2 cm) was insonified by a series of 32 unsteered plane waves with a Verasonics scanner, and a linear transducer, at a PRF (pulse repetition frequency) of 10 kHz. The RF signals were **downsampled** at 4/3 × (5 MHz) = 6.66 MHz. The properties of the linear array were:

- 128 elements
- center frequency = 5 MHz
- pitch = 0.298 mm

Download the experimental RF data. The 3-D array RF contains 128 columns (as the transducer contained 128 elements), and its length is 32 in the third dimension (as 32 plane waves were transmitted).

```
load('PWI_disk.mat');
```

The structure `param` contains some experimental properties that are required for the following steps.

```
disp('''param'' is a structure whose fields are:')
disp(param)
```

```
'param' is a structure whose fields are:
           fs: 6.6667e+06
        pitch: 2.9800e-04
           fc: 5000000
            c: 1480
           t0: 9.9500e-06
          PRF: 10000
        width: 2.6200e-04
    Nelements: 128
      TXdelay: [1×128 double]
```

```
    bandwidth: 15
```

Demodulate the RF signals with RF2IQ.

```
IQ = rf2iq(RF,param);
```

Create a 2.5-cm-by-2.5-cm image grid.

```
dx = 1e-4; % grid x-step (in m)
dz = 1e-4; % grid z-step (in m)
[x,z] = meshgrid(-1.25e-2:dx:1.25e-2,1e-2:dz:3.5e-2);
```

Create a Delay-And-Sum DAS matrix with DASMTX.

Two receive angles of ± 15° are used *for vector Doppler*.

An adequate f-number must be chosen.

```
param.fnumber = [];

param.RXangle = pi/12; % receive angle (in rad)
Mp = dasmtx(1i*size(IQ),x,z,param); % DAS matrix for +15 degrees

param.RXangle = -pi/12; % receive angle (in rad)
Mm = dasmtx(1i*size(IQ),x,z,param); % DAS matrix for -15 degrees
```

Beamform the I/Q signals twice (at + 15° and -15°).

```
IQp = Mp*reshape(IQ,[],32);
IQp = reshape(IQp,[size(x) 32]);

IQm = Mm*reshape(IQ,[],32);
IQm = reshape(IQm,[size(x) 32]);
```

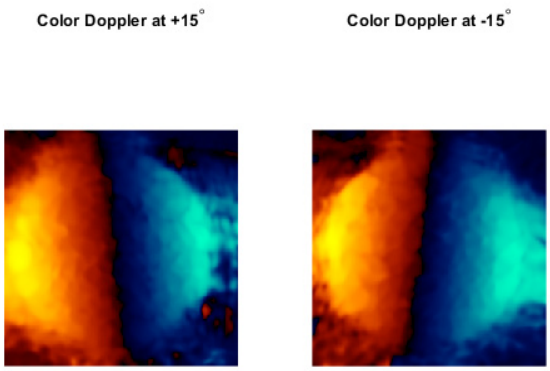
Calculate the Doppler velocities by using IQ2DOPPLER.

```
Vp = iq2doppler(IQp,param,[11 11]); % at +15 degrees
Vm = iq2doppler(IQm,param,[11 11]); % at -15 degrees
```

Display the two color Doppler maps.

```
subplot(121)
imagesc(Vp)
colormap dopplermap
caxis([-1 1]*max(abs(Vp(:))))
title('Color Doppler at +15^{\circ}')
```

```
axis equal ij off

subplot(122)
imagesc(Vm)
colormap dopplermap
caxis([-1 1]*max(abs(Vm(:))))
title('Color Doppler at -15^{\circ}')
axis equal ij off
```



Create a ROI to get masked color Doppler.

```
roi = hypot(x,z-.023)<.01; % binary mask

Vp(~roi) = NaN; % discard the out-of-ROI velocities
Vm(~roi) = NaN; % discard the out-of-ROI velocities
```

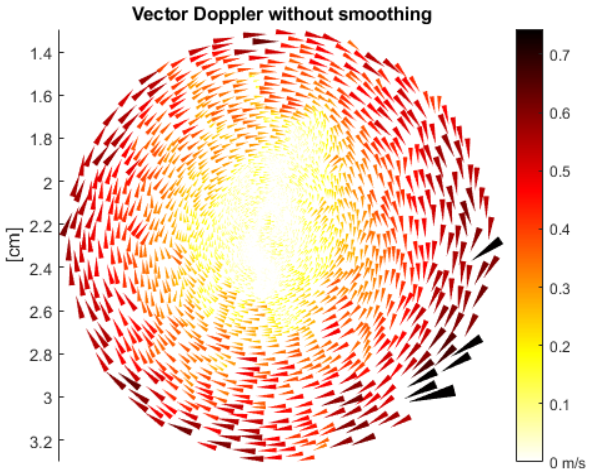
Calculate the x- and z-components of the velocity vectors.

```
Vx = (Vm-Vp)/sind(15);
Vz = -(Vm+Vp)/(1+cosd(15));
```

Display the velocity vector map.

```
figure
vplot(x*100,z*100,Vx,Vz)
ylabel('[cm]')
axis equal ij tight
set(gca,'XColor','none','box','off')
maxV = max(hypot(Vz,Vz),[],'all'); % maximum velocity
caxis([0 0.9*maxV])
c = colorbar;
c.YTickLabel{1} = '0 m/s';
```

```
colormap(flipud(hot))
title('Vector Doppler without smoothing')
```

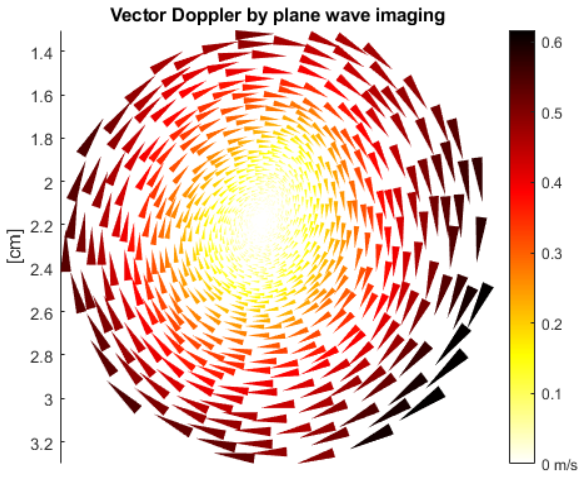


Smooth the vector Doppler map by using SMOOTHN.

```
Vs = smoothn({Vx,Vz},1e4,'robust');
Vs{1}(~roi) = NaN;
Vs{2}(~roi) = NaN;
```

Display a smoothed velocity vector map.

```
figure
vplot(x*100,z*100,Vs{1},Vs{2})
ylabel('[cm]')
axis equal ij tight
set(gca,'XColor','none','box','off')
maxV = max(hypot(Vs{1},Vs{2}),[],'all'); % maximum velocity
caxis([0 0.9*maxV])
c = colorbar;
c.YTickLabel{1} = '0 m/s';
colormap(flipud(hot))
title('Vector Doppler by plane wave imaging')
```



See also

rf2iq, dopplermap, wfilt, dasmtx

Reference

- Madiena C, Faurie J, Porée J, Garcia D. **Color and vector flow imaging in parallel ultrasound with sub-Nyquist sampling**. *IEEE Trans Ultrason Ferroelectr Freq Control*, 2018;65:795-802. **(PDF)**

Last update

2020/06

MKMOVIE Make movie frames and animated GIF

MKMOVIE creates movie frames and animated GIF for wave propagation and backpropagation.

Syntax

`F = MKMOVIE(DELAYS,PARAM)` simulates ultrasound RF radio-frequency signals by using `PFIELD` and returns movie frames. The array elements are excited at different time delays, given by `DELAYS` (in s). The characteristics of the transmit and receive must be given in the structure `PARAM` (see below for details).

▶ **TRY IT!** Enter `mkmovie` in the command window for an example.

By default, the ROI is of size 2L-by-2L, with L being the aperture length of the array [i.e. L = pitch × (Number_of_elements-1)], and its resolution is 50 pix/cm. These values can be modified through `PARAM.movie`:

e.g. `PARAM.movie = [ROI_width ROI_height resolution]`

▶ **IMPORTANT**: Pay attention to the units! Width and height are in **cm**, resolution is in **pix/cm**.

The output variable `F` is an 8-bit 3-D array that contains the frames along the third dimension.

`MKMOVIE` uses `PFIELD` during transmit and receive. The parameters that must be included in the structure `PARAM` are similar as those in `PFIELD`.

The structure PARAM

`PARAM` is a structure that contains the following fields:

- **MOVIE PROPERTIES**

1. `PARAM.movie` = [width height resolution duration fps]

IMPORTANT: *Pay attention to the units*!

ROI width and height are in cm, resolution is in pix/cm, duration is in s, fps is frame per second. The default is `[2L 2L 50 15 10]`, with `L` (in cm) being the aperture length of the array [i.e. `L = pitch*(Number_of_elements-1)`]

- **TRANSDUCER PROPERTIES**

2. `PARAM.fc`: central frequency (in Hz, **required**)
3. `PARAM.pitch`: pitch of the linear array (in m, **required**)
4. `PARAM.width`: element width **OR** `PARAM.kerf`: kerf width (in m, **required**; width = pitch-kerf)
5. `PARAM.radius`: radius of curvature (in m). The default is `Inf` (rectilinear array)
6. `PARAM.bandwidth`: pulse-echo (2-way) 6dB fractional bandwidth (in %). The default is 75%.

7. PARAM.baffle: property of the baffle: 'soft' (default), 'rigid' or a scalar > 0. See ***Note on BAFFLE property*** below for details.

- **MEDIUM PARAMETERS**

1. PARAM.c: longitudinal velocity (in m/s, default = 1540 m/s)
2. PARAM.attenuation: attenuation coefficient (dB/cm/MHz, default: 0). A linear frequency-dependence is assumed. A typical value for soft tissues is ~0.5 dB/cm/MHz.

- **TRANSMIT PARAMETERS**

1. PARAM.TXdelay: transmission law delays (in s). PARAM.TXdelay is required if the vector DELAYS is not given (see ***Other syntaxes*** below).
2. PARAM.TXapodization: transmision apodization (default: no apodization)
3. PARAM.TXnow: number of wavelengths of the TX pulse (default: 1). Use PARAM.TXnow = Inf for a mono-harmonic signal.
4. PARAM.TXfreqsweep: frequency sweep for a linear chirp (default: [ ]). To be used to simulate a linear TX chirp. See ***Note on CHIRP signals*** below for details.

Other syntaxes

F = MKMOVIE(X,Z,RC,DELAYS,PARAM) also simulates backscattered echoes. The scatterers are characterized by their coordinates (X,|Z|) and reflection coefficients RC. X, Z and RC must be of same size.

The syntaxes F = MKMOVIE(PARAM) or F = MKMOVIE(X,Z,RC,PARAM) use DELAYS = param.TXdelay.

[F,INFO] = MKMOVIE(...) returns image information in the structure INFO. INFO.Xgrid and INFO.Zgrid are the x- and z-coordinates of the image. INFO.TimeStep is the time step between two consecutive frames.

[F,INFO,PARAM] = MKMOVIE(...) updates the fields of the PARAM structure.

[...] = MKMOVIE without any input argument provides an interactive example designed to produce a movie using a 2.7 MHz phased-array transducer.

[...] = MKMOVIE(...,OPTIONS) uses the structure OPTIONS to adjust the simulations performed by PFIELD:

Advanced options

- **FREQUENCY SAMPLES**

Only frequency components of the transmitted signal in the range [|0|, 2fc] with significant amplitude are considered. The default relative amplitude is -60 dB in MKMOVIE. You can change this value by using the following:

```
[...] = MKMOVIE(...,OPTIONS),
```

where OPTIONS.dBThresh is the threshold in dB (default = −60).

- **FULL-FREQUENCY DIRECTIVITY**

By default, the directivity of the elements depends on the center frequency only. This makes the algorithm faster. To make the directivities fully frequency-dependent, use:

```
[...] = MKMOVIE(...,OPTIONS),
```

with OPTIONS.FullFrequencyDirectivity = true (default = false).

- **ELEMENT SPLITTING**

Each transducer element of the array is split into small segments. The length of these small segments must be small enough to ensure that the far-field model is accurate. By default, the elements are split into M segments, with M being defined by:

```
M = ceil(element_width/smallest_wavelength);
```

To modify the number M of subelements by splitting, you may adjust OPTIONS.ElementSplitting. For example, OPTIONS.ElementSplitting = 1.

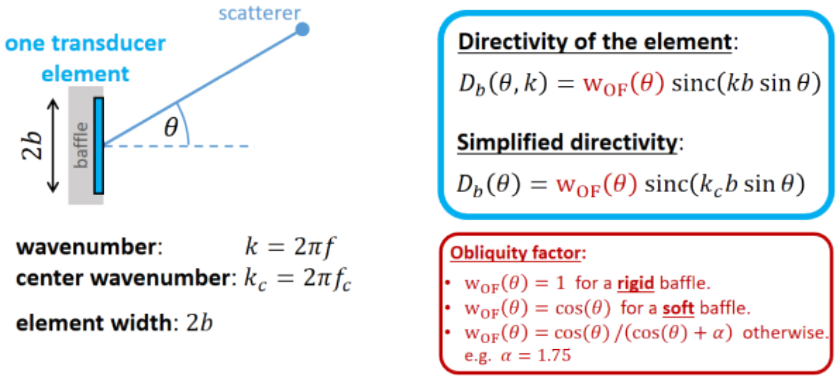
- **WAIT BAR**

If OPTIONS.WaitBar is true, a wait bar appears (only if the number of frequency samples >10). Default is true.

Note on BAFFLE property

In PFIELD, it is assumed by default that the array elements are embedded in an infinite **SOFT** baffle. To modify the property of the baffle, modify the field PARAM.baffle:

1. 'rigid'
2. 'soft' (this is the default)
3. a nonnegative scalar α , with α = (medium impedance)/(baffle impedance). Note: $\alpha = 0 \Longrightarrow$ 'rigid'; $\alpha \gg 1$ 'soft'.



The baffle property affects the obliquity factor included in the directivity of the elements. This obliquity factor is not 1 if the baffle is not rigid. A general case (see case #3 below) can be chosen by specifying an impedance ratio. For details, refer to the corresponding papers.

1. Rigid baffle ⟹ obliquity factor = 1.
2. Soft baffle ⟹ obliquity factor = cos(θ).
3. General baffle ⟹ obliquity factor = cos(θ)/(cos(θ) + α), with α = (medium impedance)/(baffle impedance).

References for baffle models:

- Selfridge *et al*. *A theory for the radiation pattern of a narrow-strip acoustic transducer*. Appl Phys Lett 37(1), 35-36 (1980)
- Pesqué *et al*. *Effect of the planar baffle impedance in acoustic radiation of a phased array element theory and experimentation*. IEEE Ultrasonics Symposium, (1984)

Example: For a baffle of impedance 2.8 MRayl (epoxy) adjacent to soft tissues of impedance 1.6 MRayls, α = 1.75.

Note on CHIRP signals

Linear chirps are characterized by `PARAM.TXnow`, `PARAM.fc` and `PARAM.TXfreqsweep`. The transmitted pulse has a duration of approximately `T` (= `PARAM.TXnow`/`PARAM.fc`), with the amplitude and phase defined over the time interval `-T/2` to `+T/2`.

The total frequency sweep is `DeltaF` (= `PARAM.TXfreqsweep`): the frequencies increase linearly from (`PARAM.fc - DeltaF/2`) to (`PARAM.fc + DeltaF/2`) in the defined time interval.

Documentation: Chirp spectrum in Wikipedia.

Example #1: Display the propagation of a focused wave

This example shows how to simulate and display the wave propagation of RF signals that focus to a given location.

Download the properties of a 2.7-MHz 64-element cardiac phased array in a structure `param` by using GETPARAM.

```
param = getparam('P4-2v');
```

Choose a focus location at xf = 0 cm, zf = 5 cm.

```
xf = 0e-2; zf = 3e-2; % focus position (in m)
```

Obtain the corresponding transmit time delays (in s) with TXDELAY.

```
txdel = txdelay(xf,zf,param); % in s
```

Define the image size (in cm) and its resolution (in pix/cm).

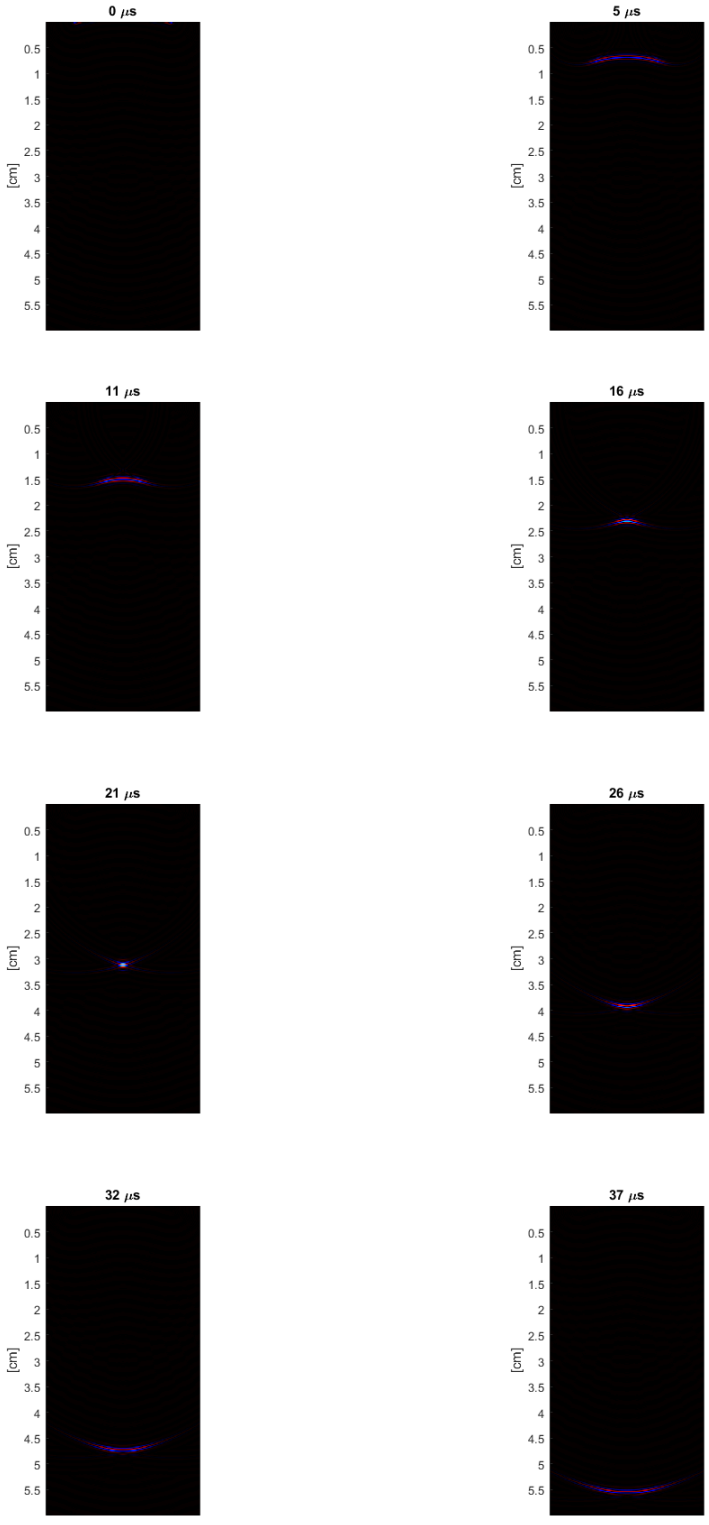
```
param.movie = [3 6 100];
```

Make movie frames by using MKMOVIE and create an animated GIF.

```
[F,info] = mkmovie(txdel,param,'focused_wave.gif');
```

Display nine movie frames.

```
colormap([1-hot(128); hot(128)]);
framenum = round(linspace(1,size(F,3),9));
for k = 1:9
    image(info.Xgrid*100,info.Zgrid*100,F(:,:,framenum(k)))
    axis equal ij tight
    title([int2str(info.TimeStep*framenum(k)*1e6) ' \mus'])
    ylabel('[cm]')
    set(gca,'XColor','none','box','off')
    snapnow
end
```


0 μs
5 μs
11 μs
16 μs
21 μs
26 μs
32 μs
37 μs
[cm]
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
5.5

Example #2: Display the backpropagation of a diverging wave

This example shows how to simulate and display a diverging wave backpropagated by a few scatterers.

Download the properties of a 2.7-MHz 64-element cardiac phased array in a structure `param` by using GETPARAM.

```
param = getparam('P4-2v');
```

Obtain the transmit time delays (in s) for a 90-degree wide diverging wave with TXDELAY.

```
txdel = txdelay(param,0,pi/2); % in s
```

Define 20 random scatterers and their bacskscattering coefficients.

```
n = 20;
x = rand(n,1)*8e-2-4e-2; % (in m)
z = rand(n,1)*10e-2; % (in m)
RC = (rand(n,1)+1)/2;
```

Define the image size (in cm) and its resolution (in pix/cm).

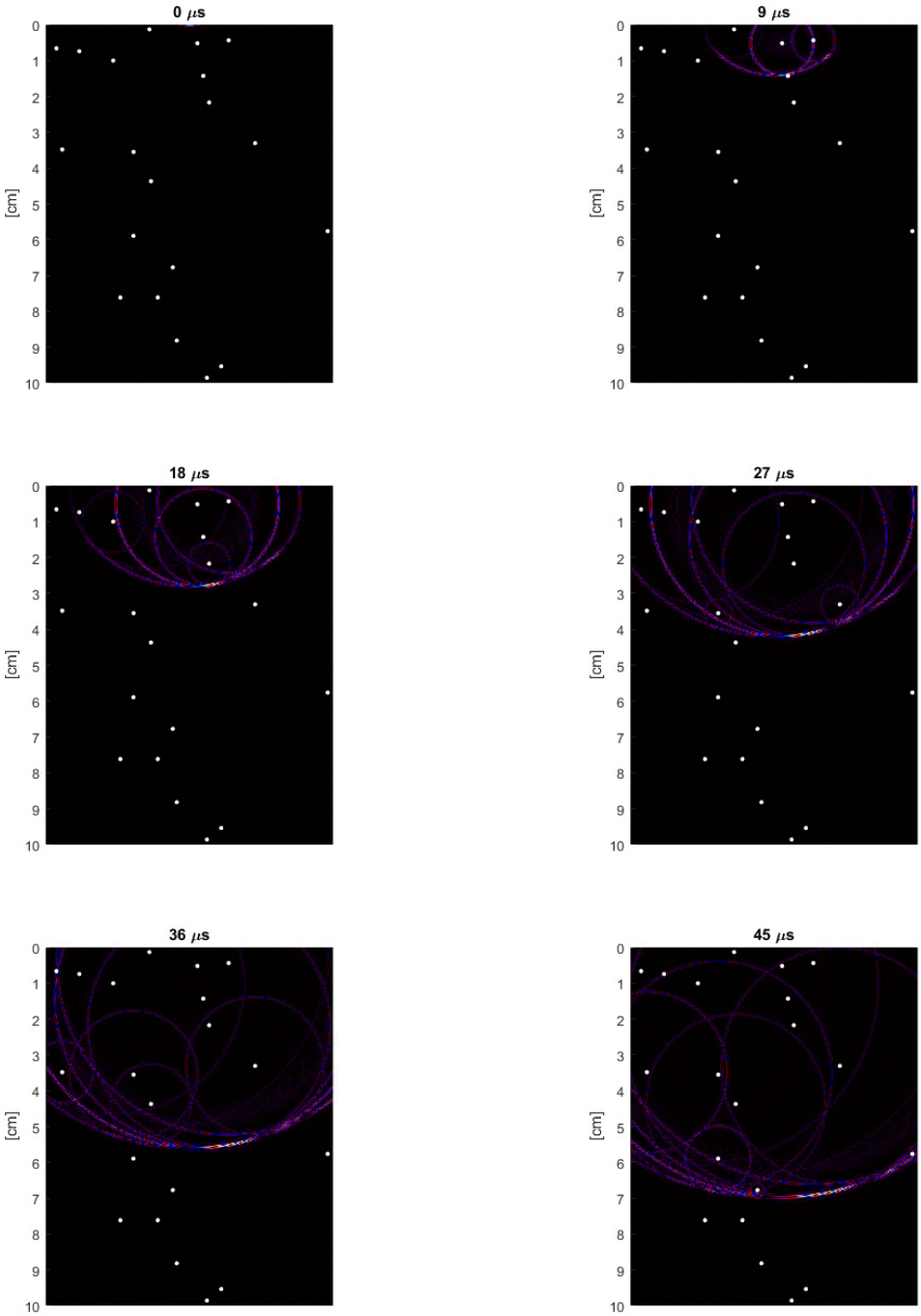
```
param.movie = [8 10];
```

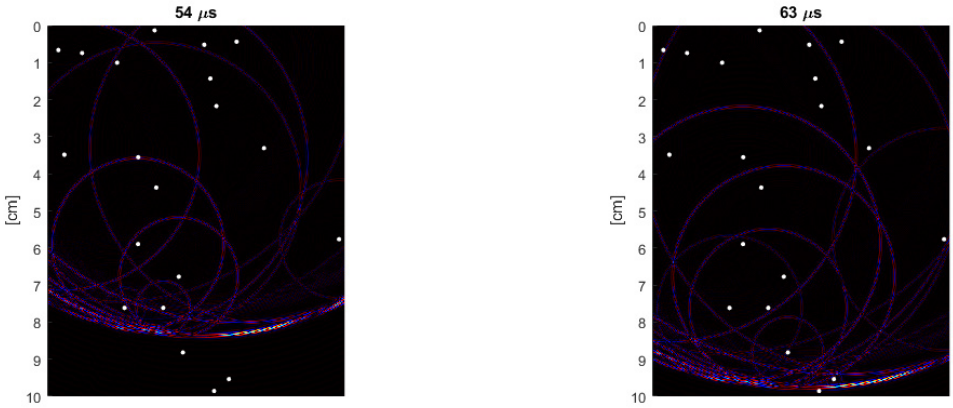
Make movie frames by using MKMOVIE and create an animated GIF.

```
[F,info] = mkmovie(x,z,RC,txdel,param,'diverging_wave.gif');
```

Display nine movie frames.

```
colormap([1-hot(128); hot(128)]);
framenum = round(linspace(1,size(F,3),9));
for k = 1:9
    image(info.Xgrid*100,info.Zgrid*100,F(:,:,framenum(k)))
    hold on
    scatter(x*100,z*100,10,'w','filled')
    hold off
    axis equal ij tight
    title([int2str(info.TimeStep*framenum(k)*1e6) ' \mus'])
    ylabel('[cm]')
    set(gca,'XColor','none','box','off')
    snapnow
end
```


0 μs
9 μs
18 μs
27 μs
36 μs
45 μs
[cm]
0
1
2
3
4
5
6
7
8
9
10



Example #3: Display the backpropagation of a diverging wave from a convex array

This example shows how to simulate and display a diverging wave generated by a convex array and backpropagated by a few scatterers.

Download the properties of a 3.6-MHz 128-element convex array in a structure `param` by using GET-PARAM.

```
param = getparam('C5-2v');
```

Design a few scatterers.

```
xs = [4.3 3.2 1.7 0 -1.7 -3.2 -4.3 0 -2.5 2.5]*1e-2; % in m
zs = [7 8.1 8.8 9 8.8 8.1 7 5 2 2]*1e-2; % in m
```

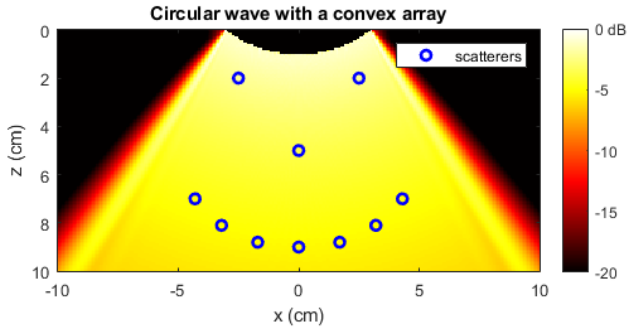
Set all the transmit delays to 0.

```
txdel = zeros(1,param.Nelements);
```

Calculate and display the pressure field.

```
[x,z] = meshgrid(linspace(-.1,.1,256),linspace(0,.1,128));
P = pfield(x,z,txdel,param);

imagesc(x(1,:)*1e2,z(:,1)*1e2,20*log10(P/max(P,[],'all')))
axis equal ij tight
caxis([-20 0]) % dynamic range = [-20,0] dB
c = colorbar;
c.YTickLabel{end} = '0 dB';
colormap hot
xlabel('x (cm)'), ylabel('z (cm)')
title('Circular wave with a convex array')
```

```
hold on
plot(xs*1e2,zs*1e2,'bo','Linewidth',2)
hold off
legend('scatterers')
```

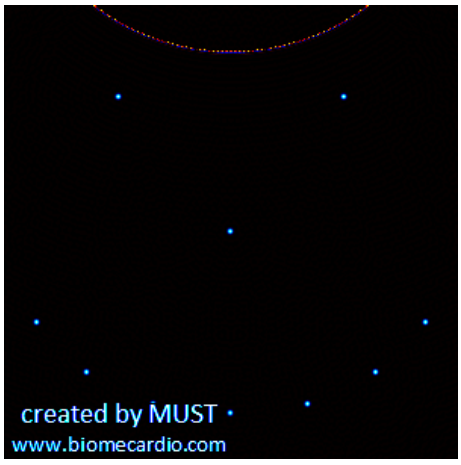


Add a "ghost" scatterer (its reflection coefficient is zero) at 10 cm deep to ensure that RF are simulated till 10-cm depth.

```
RC = [ones(1,numel(xs)) 0];
xs = [xs 0]; zs = [zs 1e-2];
```

Create a GIF movie named 'convexSmiley.gif' with `MKMOVIE`.

```
param.movie = [10 10 30]; % 10-cm-by-10-cm ROI, resolution = 30 pix/cm
mkmovie(xs,zs,RC,txdel,param,'convexSmiley.gif');
```



See also

pfield, simus, txdelay

Last update

2020/01

PFIELD RMS acoustic pressure field of a linear or convex array

PFIELD returns the radiation pattern of a uniform LINEAR or CONVEX array

Syntax

`RP = PFIELD(X,Y,Z,DELAYS,PARAM)` returns the radiation pattern of a uniform linear or convex array whose elements are excited at different time delays (given by the vector `DELAYS`). The radiation pattern `RP` is given in terms of the root-mean-square (RMS) of acoustic pressure. The characteristics of the array and transmission must be given in the structure `PARAM` (see below for details). The radiation pattern is calculated at the points specified by (`X`,`Y`,`Z`).

`RP = PFIELD(X,Z,DELAYS,PARAM)` or `RP = PFIELD(X,[],Z,DELAYS,PARAM)` disregards elevation focusing (`PARAM.focus` is ignored) and assumes that `Y`=0 (2-D space). The computation is faster in 2-D.

▶ **TRY IT!** Enter `pfield` in the command window for an example.

Units: `X`, `Y`, `Z` must be in m; `DELAYS` must be in s.

`DELAYS` can also be a matrix. This alternative can be used to simulate MLT (multi-line transmit) sequences. In this case, each ROW represents a delay series. For example, to create a 4-MLT sequence with a 64-element phased array, `DELAYS` matrix must have 4 rows and 64 columns (size = [4 64]).

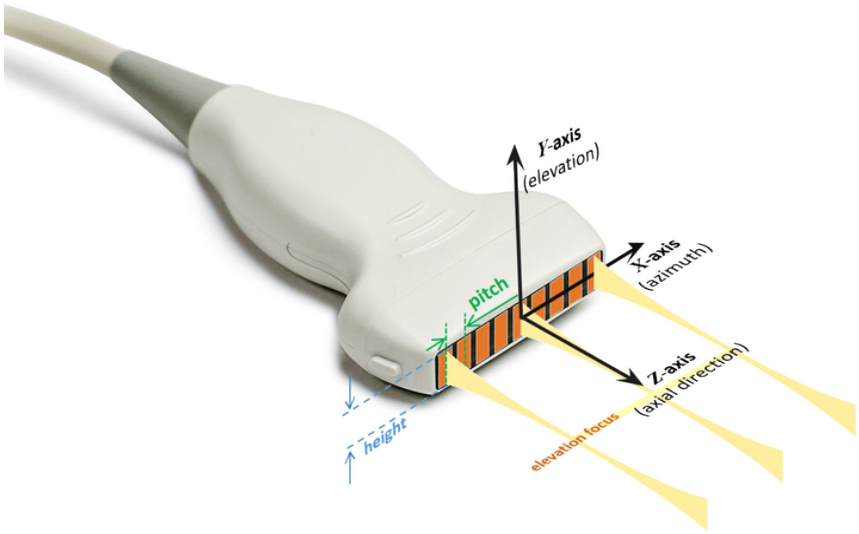
`PFIELD` is called by `SIMUS` to simulate ultrasound RF radio-frequency signals generated by an ultrasound uniform linear or convex array.

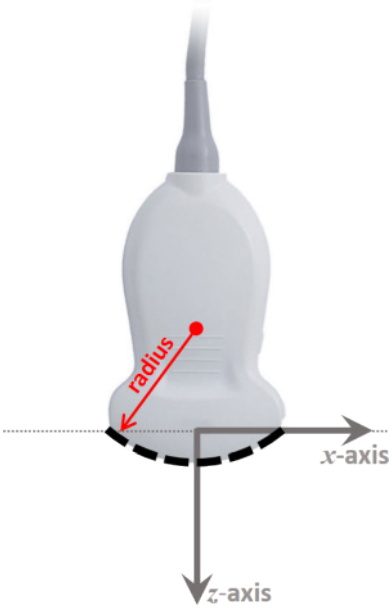
Uniform linear array (ULA)

The ***pitch*** is defined as the center-to-center distance between two adjacent elements. The ***kerf*** width is the distance that separates two adjacent elements. The pitch and kerf are constant for a uniform linear array (ULA).

Some functions of the MUST toolbox can also consider curved (convex) ULAs.

The pitch is required as an input parameter for `PFIELD`. The element width or the kerf width is also required. See the paragraph below entitled **The structure `PARAM`** for details.





The structure PARAM

PARAM is a structure that contains the following fields:

- **TRANSDUCER PROPERTIES**

1. PARAM.`fc`: central frequency (in Hz, **required**)
2. PARAM.`pitch`: pitch of the linear array (in m, **required**)
3. PARAM.`width`: element width **OR** PARAM.`kerf`: kerf width (in m, **required**; width = pitch-kerf)
4. PARAM.`focus`: elevation focus (in m, ignored if Y is not given). The default is Inf (no elevation focusing)
5. PARAM.`height`: element height (in m, ignored if Y is not given). The default is Inf (no elevation focusing)
6. PARAM.`radius`: radius of curvature (in m). The default is `Inf` (rectilinear array)

7. PARAM.bandwidth: pulse-echo (2-way) 6dB fractional bandwidth (in %). The default is 75%.
8. PARAM.baffle: property of the baffle: 'soft' (default), 'rigid' or a scalar > 0. See ***Note on BAFFLE property*** below for details.

- **MEDIUM PARAMETERS**

1. PARAM.c: longitudinal velocity (in m/s, default = 1540 m/s)
2. PARAM.attenuation: attenuation coefficient (dB/cm/MHz, default: 0). A linear frequency-dependence is assumed. A typical value for soft tissues is ~0.5 dB/cm/MHz.

- **TRANSMIT PARAMETERS**

1. PARAM.TXapodization: transmision apodization (default: no apodization)
2. PARAM.TXnow: number of wavelengths of the TX pulse (default: 1). Use PARAM.TXnow = Inf for a mono-harmonic signal.
3. PARAM.TXfreqsweep: frequency sweep for a linear chirp (default: [ ]). To be used to simulate a linear TX chirp. See ***Note on CHIRP signals*** below for details.

Other syntaxes

[RP,PARAM] = PFIELD(...) also returns the complete list of parameters including the default values.

[...] = PFIELD without any input argument provides an interactive example designed to produce a focused ultrasound beam using a 2.7 MHz phased-array transducer.

X-, Y-, and Z-axes

Conventional axes are used:

1. For a **LINEAR** array, the X-axis is PARALLEL to the transducer and points from the first (leftmost) element to the last (rightmost) element (X = 0 at the CENTER of the transducer). The Z-axis is PERPENDICULAR to the transducer and points downward (Z = 0 at the level of the transducer, Z increases as depth increases). The Y-axis is such that the coordinates are right-handed. These axes are represented in the above ULA figure.
2. For a **CONVEX** array, the X-axis is parallel to the chord and Z = 0 at the level of the chord.

Element directivity

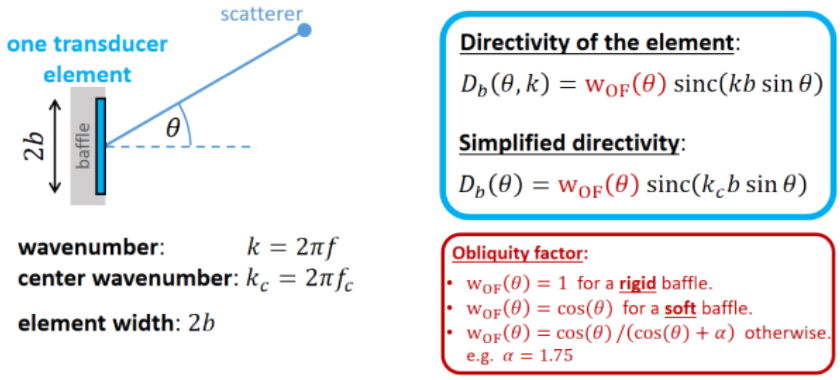
By default, the calculation is made faster by assuming that the directivity of the elements is dependent only on the central frequency (see figure below). This simplification very little affects the pressure field in most situations (except in the vicinity of the array). To turn off this option, use OPTIONS.FullFrequencyDirectivity = true.

See ***ADVANCED OPTIONS*** below.

Note on BAFFLE property

In PFIELD, it is assumed by default that the array elements are embedded in an infinite **SOFT** baffle. To modify the property of the baffle, modify the field PARAM.baffle:

1. `'rigid'`
2. `'soft'` (this is the default)
3. a nonnegative scalar α , with α = (medium impedance)/(baffle impedance). Note: $\alpha = 0 \Longrightarrow$ `'rigid'`; $\alpha \gg 1 \Longrightarrow$ `'soft'`.



The baffle property affects the obliquity factor included in the directivity of the elements. This obliquity factor is not 1 if the baffle is not rigid. A general case (see case #3 below) can be chosen by specifying an impedance ratio. For details, refer to the corresponding papers.

1. Rigid baffle $\Longrightarrow$ obliquity factor = 1.
2. Soft baffle $\Longrightarrow$ obliquity factor = cos(θ).
3. General baffle $\Longrightarrow$ obliquity factor = cos(θ)/(cos(θ) + α), with α = (medium impedance)/(baffle impedance).

References for baffle models:

- Selfridge *et al*. *A theory for the radiation pattern of a narrow-strip acoustic transducer*. Appl Phys Lett 37(1), 35-36 (1980)
- Pesqué *et al*. *Effect of the planar baffle impedance in acoustic radiation of a phased array element theory and experimentation*. IEEE Ultrasonics Symposium, (1984)

Example: For a baffle of impedance 2.8 MRayl (epoxy) adjacent to soft tissues of impedance 1.6 MRayls, α = 1.75.

Note on CHIRP signals

Linear chirps are characterized by `PARAM.TXnow`, `PARAM.fc` and `PARAM.TXfreqsweep`. The transmitted pulse has a duration of approximately `T` (= `PARAM.TXnow/PARAM.fc`), with the amplitude and phase defined over the time interval `-T/2` to `+T/2`.

The total frequency sweep is `DeltaF` (= `PARAM.TXfreqsweep`): the frequencies increase linearly from (`PARAM.fc - DeltaF/2`) to (`PARAM.fc + DeltaF/2`) in the defined time interval.

Documentation: Chirp spectrum in Wikipedia.

Advanced options

- **FREQUENCY SAMPLES**

Only frequency components of the transmitted signal in the range [0, `2fc`] with significant amplitude are considered. The default relative amplitude is -60 dB in `PFIELD`. You can change this value by using the following:

```
[...] = PFIELD(...,OPTIONS),
```

where `OPTIONS.dBThresh` is the threshold in dB (default = `-60`).

- **FULL-FREQUENCY DIRECTIVITY**

By default, the directivity of the elements depends on the center frequency only. This makes the algorithm faster. To make the directivities fully frequency-dependent, use:

```
[...] = PFIELD(...,OPTIONS),
```

with `OPTIONS.FullFrequencyDirectivity = true` (default = `false`).

- **ELEMENT SPLITTING**

Each transducer element of the array is split into small segments. The length of these small segments must be small enough to ensure that the far-field model is accurate. By default, the elements are split into M segments, with `M` being defined by:

```
M = ceil(element_width/smallest_wavelength);
```

To modify the number `M` of subelements by splitting, you may adjust `OPTIONS.ElementSplitting`. For example, `OPTIONS.ElementSplitting = 1`.

- **WAIT BAR**

If `OPTIONS.WaitBar` is `true`, a wait bar appears (only if the number of frequency samples >10). Default is `true`.

Example #1: 2-D focused pressure field with a phased-array transducer

This example shows how to generate a focused pressure field with a phased-array transducer.

Download the properties of a 2.7-MHz 64-element cardiac phased array in a structure `param` by using GETPARAM.

```
param = getparam('P4-2v');
```

Choose a focus location at xf = 2 cm, zf = 5 cm.

```
xf = 2e-2; zf = 5e-2; % focus position (in m)
```

Obtain the corresponding transmit time delays (in s).

```
txdel = txdelay(xf,zf,param); % in s
```

Simulate the pressure field by using PFIELD.

First define an image grid.

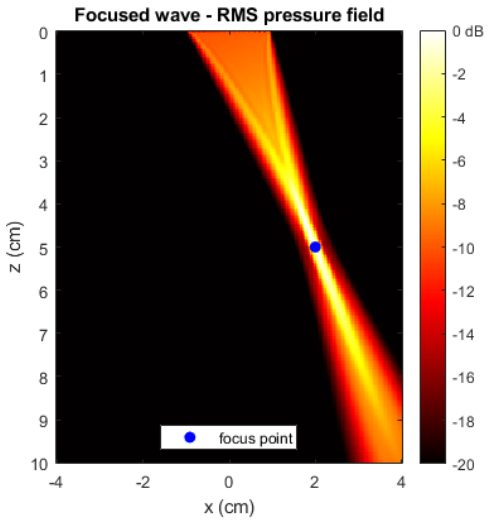
```
x = linspace(-4e-2,4e-2,200); % in m
z = linspace(0,10e-2,200); % in m
[x,z] = meshgrid(x,z);
```

The function PFIELD yields the root-mean-square (RMS) pressure field.

```
P = pfield(x,z,txdel,param);
```

Display the acoustic pressure field.

```
imagesc(x(1,:)*1e2,z(:,1)*1e2,20*log10(P/max(P,[],'all')))
caxis([-20 0]) % dynamic range = [-20,0] dB
c = colorbar;
c.YTickLabel{end} = '0 dB';
colormap hot
axis equal ij tight
xlabel('x (cm)'), ylabel('z (cm)')
title('Focused wave - RMS pressure field')

hold on
plot(xf*1e2,zf*1e2,'bo','MarkerFaceColor','b')
legend('focus point','Location','South')
hold off
```



Example #2: 3-D focused pressure field with a cardiac phased-array

This example shows how to generate a 3-D focused pressure field with a phased-array transducer.

Download the properties of a 2.7-MHz 64-element cardiac phased array in a structure `param` by using GETPARAM.

```
param = getparam('P4-2v');
```

Choose a focus location at xf = 2 cm, zf = 5 cm.

```
xf = -2e-2; zf = 5e-2; % focus position (in m)
```

Obtain the corresponding transmit time delays (in s).

```
txdel = txdelay(xf,zf,param); % in s
```

Simulate the pressure field by using PFIELD.

First define the x-, y-, z-coordinates:

```
x = linspace(-4e-2,4e-2,200);
y = linspace(-0.75,.75,50)*param.height;
z = linspace(0,10e-2,200);
```

Obtain the RMS pressure field on the azimuthal plane:

```
[xaz,zaz] = meshgrid(x,z);
yaz = zeros(size(xaz));
Paz = pfield(xaz,yaz,zaz,txdel,param);
```

Obtain the RMS pressure field on the elevation plane:

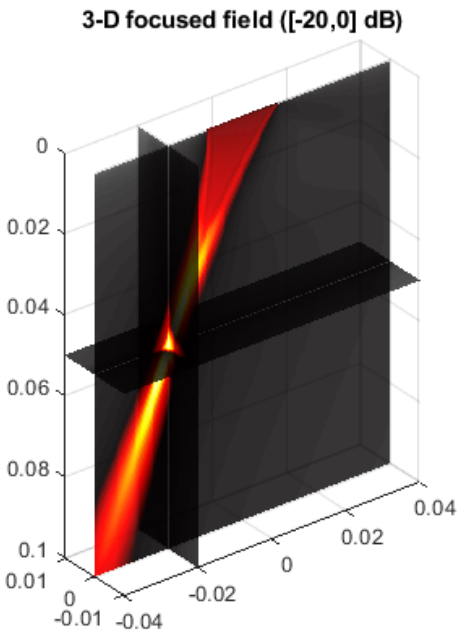
```
[yel,zel] = meshgrid(y,z);
xel = ones(size(yel))*xf;
Pel = pfield(xel,yel,zel,txdel,param);
```

Obtain the RMS pressure field on the focal plane:

```
[xfo,yfo] = meshgrid(x,y);
zfo = ones(size(xfo))*zf;
Pfo = pfield(xfo,yfo,zfo,txdel,param);
```

Display the acoustic pressure field.

```
Pmax = max(Paz(:));
surf(xaz,yaz,zaz,20*log10(Paz/Pmax)), shading flat
hold on
surf(xel,yel,zel,20*log10(Pel/Pmax)), shading flat
surf(xfo,yfo,zfo,20*log10(Pfo/Pmax)), shading flat
hold off
% Fine-tune the figure:
axis equal
set(gca,'ZDir','reverse')
title('3-D focused field ([-20,0] dB)')
caxis([-20 0])
alpha color
```



Example #3: Diverging wave with a phased array

This example shows how to simulate a circular wave with a phased-array transducer with the 2-D approximation.

Download the properties of a 2.7-MHz 64-element cardiac phased array in a structure `param` by using GETPARAM.

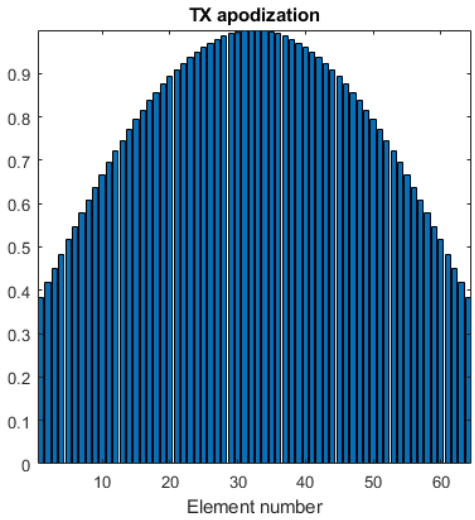
```
param = getparam('P4-2v');
```

Calculate the transmit delays to generate a 75-degrees wide circular wave. Use TXDELAY.

```
width = 75/180*pi; % width angle in rad
tilt = 0; % tilt angle in rad
txdel = txdelay(param,tilt,width); % in s
```

Define a transmit apodization.

```
param.TXapodization = cos(linspace(-3*pi/8,3*pi/8,64));

bar(param.TXapodization)
xlabel('Element number')
title('TX apodization')
axis tight square
```



Simulate the pressure field with `PFIELD`.

First define the image grid.

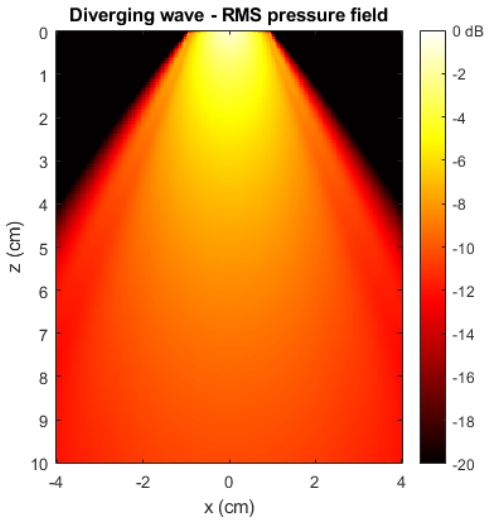
```
x = linspace(-4e-2,4e-2,200); % in m
z = linspace(0,10e-2,200); % in m
[x,z] = meshgrid(x,z);
```

The function `PFIELD` yields the root-mean-square (RMS) pressure field.

```
P = pfield(x,z,txdel,param);
```

Display the acoustic pressure field.

```
imagesc(x(1,:)*1e2,z(:,1)*1e2,20*log10(P/max(P,[],'all')))
caxis([-20 0]) % dynamic range = [-20,0] dB
c = colorbar;
c.YTickLabel{end} = '0 dB';
colormap hot
axis equal ij tight
xlabel('x (cm)'), ylabel('z (cm)')
title('Diverging wave - RMS pressure field')
```



Example #4: Multi-line transmit (MLT) with a phased-array

This example shows how to generate a MLT transmit sequence with a phased-array transducer. In this example, a 3-MLT sequence is designed, i.e. three focused waves are transmitted simultaneously.

Download the properties of a 2.7-MHz 64-element cardiac phased array in a structure `param` by using GETPARAM.

```
param = getparam('P4-2v');
```

Define the TX delays for a 3-MLT transmit sequence.

```
x0 = 2e-2; z0 = 5e-2; % in m
xf = [-x0 0 x0]; zf = [z0 sqrt(x0^2+z0^2) z0]; % focus points (in m)
txdel = txdelay(xf,zf,param); % in s
```

Simulate the pressure field with PFIELD.

Define the image grid.

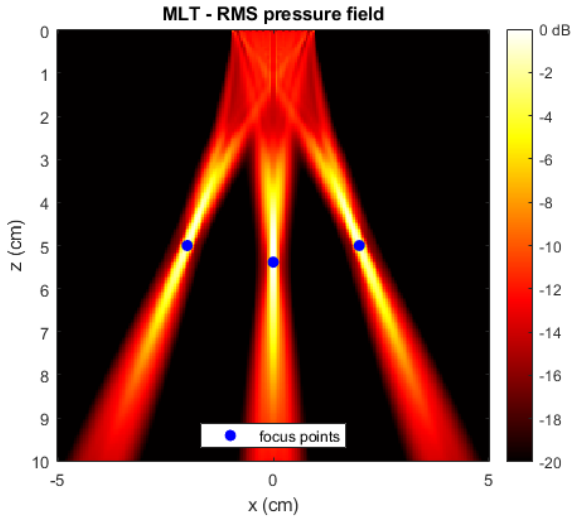
```
x = linspace(-5e-2,5e-2,200); % in m
z = linspace(0,10e-2,200); % in m
[x,z] = meshgrid(x,z); % image grid
y = zeros(size(x));
```

Use PFIELD to obtain the RMS pressure field.

```
P = pfield(x,y,z,txdel,param);
```

Display the pressure field.

```
imagesc(x(1,:)*1e2,z(:,1)*1e2,20*log10(P/max(P,[],'all')))
caxis([-20 0]) % dynamic range = [-20,0] dB
c = colorbar;
c.YTickLabel{end} = '0 dB';
colormap hot
axis equal ij tight
xlabel('x (cm)'), ylabel('z (cm)')
title('MLT - RMS pressure field')

hold on
plot(xf*1e2,zf*1e2,'bo','MarkerFaceColor','b')
legend('focus points','Location','South')
hold off
```



Example #5: Focused pressure field with a subaperture of a linear array

This example shows how to generate a focused pressure field with a subaperture of a linear array transducer.

Download the properties of a 7.6-MHz 128-element uniform linear array in a structure `param` by using GETPARAM.

```
param = getparam('L11-5v');
```

Calculate the positions of the elements (for visualization).

```
L = param.pitch*(param.Nelements-1); % array aperture (in m)
xe = linspace(-0.5,0.5,param.Nelements)*L;
ze = zeros(1,param.Nelements);
```

Create a linear "sub-array" with 24 elements.

```
param_suba = param;
param_suba.Nelements = 24;
```

Define a focus location at xf = 0 cm, zf = 2.5 cm, ***relative to*** the sub-array

```
xf = 0; zf = 2.5e-2; % focus position (in m)
```

Calculate the transmit time delays (in s) for the sub-array.

```
txdel_suba = txdelay(xf,zf,param_suba); % in s
```

Obtain the transmit time delays (in s) for the (complete) array...

... when firing with elements #1 to #24

```
txdel_1to24 = NaN(1,128);
txdel_1to24(1:24) = txdel_suba;
```

... when firing with elements #91 to #114

```
txdel_91to114 = NaN(1,128);
txdel_91to114(91:114) = txdel_suba;
```

Simulate the pressure fields by using `PFIELD`.

First define an image grid.

```
x = linspace(-2.5e-2,2.5e-2,150); % in m
z = linspace(0,5e-2,150); % in m
[x,z] = meshgrid(x,z);
```

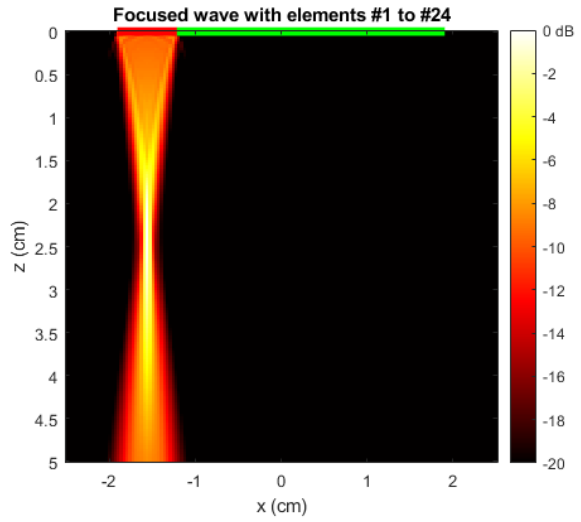
The function `PFIELD` yields the root-mean-square (RMS) pressure fields.

```
P_1to24 = pfield(x,z,txdel_1to24,param);
P_91to114 = pfield(x,z,txdel_91to114,param);
```

Display the acoustic pressure field when firing elements #1 to #24.

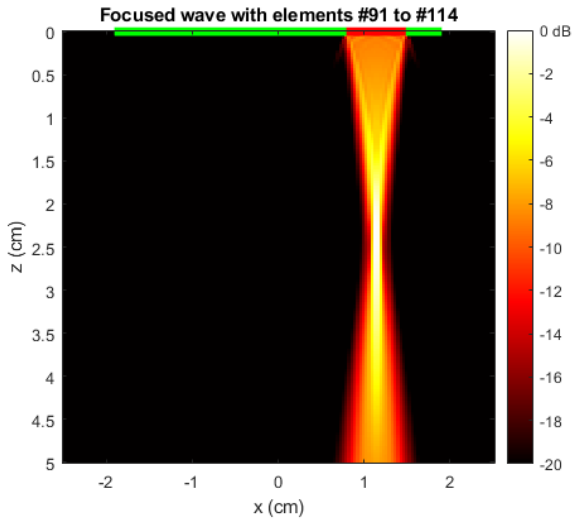
```
imagesc(x(1,:)*1e2,z(:,1)*1e2,20*log10(P_1to24/max(P_1to24,[],'all')))
caxis([-20 0]) % dynamic range = [-20,0] dB
c = colorbar;
c.YTickLabel{end} = '0 dB';
colormap hot
axis equal ij tight
```

```
xlabel('x (cm)'), ylabel('z (cm)')
title('Focused wave with elements #1 to #24')

hold on
plot(xe*1e2,ze*1e2,'g','Linewidth',5)
plot(xe(1:24)*1e2,ze(1:24)*1e2,'r','Linewidth',5)
hold off
```



Display the acoustic pressure field when firing elements #91 to #114.

```
imagesc(x(1,:)*1e2,z(:,1)*1e2,20*log10(P_91to114/max(P_91to114,[],'all')))
caxis([-20 0]) % dynamic range = [-20,0] dB
c = colorbar;
c.YTickLabel{end} = '0 dB';
colormap hot
axis equal ij tight
xlabel('x (cm)'), ylabel('z (cm)')
title('Focused wave with elements #91 to #114')

hold on
plot(xe*1e2,ze*1e2,'g','Linewidth',5)
plot(xe(91:114)*1e2,ze(91:114)*1e2,'r','Linewidth',5)
hold off
```



Example #6: Steered plane wave with a linear array

This example shows how to simulated a steered plane wave with a linear vascular transducer.

Download the properties of a 7.6-MHz 128-element uniform linear array in a structure `param` by using GETPARAM.

```
param = getparam('L11-5v');
```

Calculate the transmit delays for a plane wave steered at +10 degrees

```
tilt = 10/180*pi; % tilt angle in rad
txdel = txdelay(param,tilt); % in s
```

Use PFIELD to simulate the pressure field.

First define the image grid.

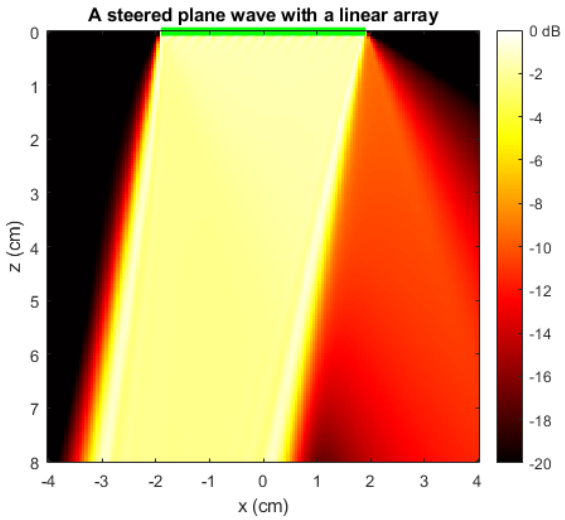
```
x = linspace(-4e-2,4e-2,150); % in m
z = linspace(0,8e-2,150); % in m
[x,z] = meshgrid(x,z);
```

The function PFIELD yields the root-mean-square (RMS) pressure field.

```
P = pfield(x,z,txdel,param);
```

Display the RMS acoustic pressure field.

```
imagesc(x(1,:)*1e2,z(:,1)*1e2,20*log10(P/max(P,[],'all')))
caxis([-20 0]) % dynamic range = [-20,0] dB
c = colorbar;
c.YTickLabel{end} = '0 dB';
colormap hot
axis equal ij tight
xlabel('x (cm)'), ylabel('z (cm)')
title('A steered plane wave with a linear array')

% Calculate the positions of the element centers.
L = param.pitch*(param.Nelements-1); % array aperture (in m)
xe = linspace(-0.5,0.5,param.Nelements)*L;
ze = zeros(1,param.Nelements);
hold on
plot(xe*1e2,ze*1e2,'g','Linewidth',5)
hold off
```



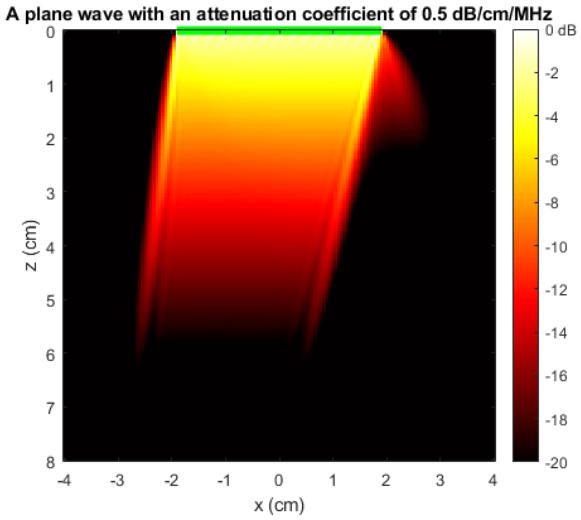
Add an attenuation coefficient.

```
param.attenuation = 0.5; % attenuation coefficient (in dB/cm/MHz)
```

Calculate the pressure field and compare.

```
P = pfield(x,z,txdel,param);

imagesc(x(1,:)*1e2,z(:,1)*1e2,20*log10(P/max(P,[],'all')))
caxis([-20 0]) % dynamic range = [-20,0] dB
c = colorbar;
c.YTickLabel{end} = '0 dB';
colormap hot
axis equal ij tight
xlabel('x (cm)'), ylabel('z (cm)')
title('A plane wave with an attenuation coefficient of 0.5 dB/cm/MHz')
```

```
hold on
plot(xe*1e2,ze*1e2,'g','Linewidth',5)
hold off
```



Example #7: Focused pressure field with a subaperture of a curved array

This example shows how to generate a focused pressure field with a subaperture of a convex array transducer.

Download the properties of a 3.6-MHz 128-element curved linear array in a structure `param` by using GETPARAM.

```
param = getparam('C5-2v');
```

Calculate the positions of the elements (for visualization).

```
R = param.radius;
p = param.pitch;
N = param.Nelements;
L = 2*R*sin(asin(p/2/R)*(N-1)); % chord length
h = sqrt(R^2-L^2/4);
th = linspace(atan2(-L/2,h),atan2(L/2,h),N);
xe = R*sin(th);
ze = R*cos(th)-h;
```

Create a convex "sub-array" with 32 elements.

```
param_suba = param;
param_suba.Nelements = 32;
```

Define a focus location at xf = 0 cm, zf = 7.5 cm, ***relative to*** the sub-array

```
xf = 0; zf = 7.5e-2; % focus position (in m)
```

Calculate the transmit time delays (in s) for the sub-array.

```
txdel_suba = txdelay(xf,zf,param_suba); % in s
```

Obtain the transmit time delays (in s) for the (complete) array...

... when firing with elements #1 to #32

```
txdel_1to32 = NaN(1,128);
txdel_1to32(1:32) = txdel_suba;
```

... when firing with elements #83 to #114

```
txdel_83to114 = NaN(1,128);
txdel_83to114(83:114) = txdel_suba;
```

Simulate the pressure fields by using `PFIELD`.

First define an image polar-type grid by using `IMPOLGRID`.

```
[x,z] = impolgrid([128 128],15e-2,param);
y = zeros(size(x));
```

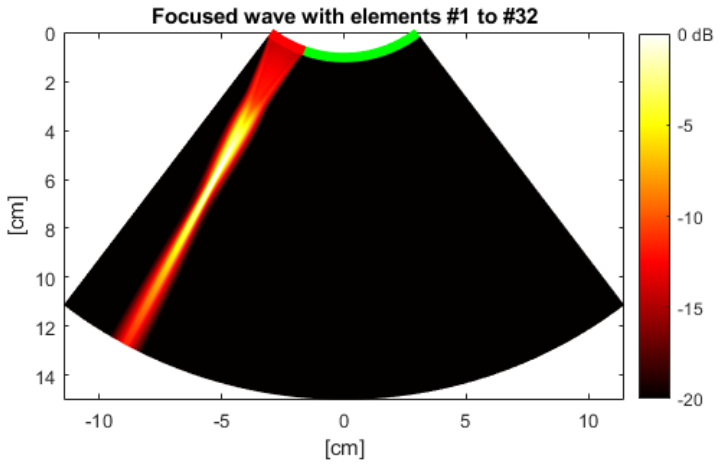
The function `PFIELD` yields the root-mean-square (RMS) pressure fields.

```
P_1to32 = pfield(x,y,z,txdel_1to32,param);
P_83to114 = pfield(x,y,z,txdel_83to114,param);
```

Display the acoustic pressure field when firing elements #1 to #32.

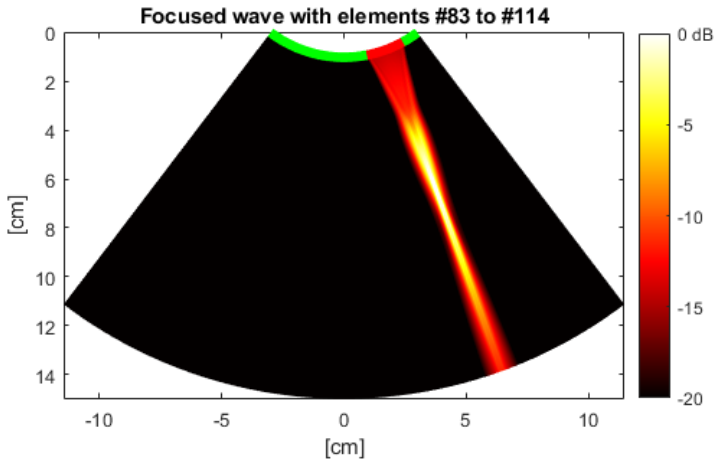
```
pcolor(x*1e2,z*1e2,20*log10(P_1to32/max(P_1to32(:))))
shading interp
colormap hot
axis equal tight ij
caxis([-20 0]) % dynamic range = [-20,0] dB
c = colorbar;
c.YTickLabel{end} = '0 dB';
xlabel('[cm]'), ylabel('[cm]')
title('Focused wave with elements #1 to #32')

hold on
plot(xe*1e2,ze*1e2,'g','Linewidth',5)
```

```
plot(xe(1:32)*1e2,ze(1:32)*1e2,'r','Linewidth',5)
hold off
```



Display the acoustic pressure field when firing elements #83 to #114.

```
pcolor(x*1e2,z*1e2,20*log10(P_83to114/max(P_83to114(:))))
shading interp
colormap hot
axis equal tight ij
caxis([-20 0]) % dynamic range = [-20,0] dB
c = colorbar;
c.YTickLabel{end} = '0 dB';
xlabel('[cm]'), ylabel('[cm]')
title('Focused wave with elements #83 to #114')

hold on
plot(xe*1e2,ze*1e2,'g','Linewidth',5)
plot(xe(83:114)*1e2,ze(83:114)*1e2,'r','Linewidth',5)
hold off
```



See also

getparam, txdelay, simus

References

- The paper that describes the first 2-D version of PFIELD is: Shariari S, Garcia D. **Meshfree simulations of ultrasound vector flow imaging using smoothed particle hydrodynamics**. *Phys Med Biol*, 2018;63:205011. **(PDF)**
- The paper that describes the theory of the full (2-D + 3-D) version of `PFIELD` will be submitted by February 2021.

Last update

2020/12

RF2IQ I/Q demodulation of RF data

RF2IQ demodulates the radiofrequency (RF) bandpass signals and returns the Inphase/Quadrature (I/Q) components.

Syntax

`IQ = RF2IQ(RF,Fs,Fc)` demodulates the radiofrequency (`RF`) bandpass signals and returns the Inphase/Quadrature (I/Q) components. `IQ` is a complex whose real (imaginary) part contains the in-phase (quadrature) component.

1. `Fs` is the sampling frequency of the RF signals (in Hz),
2. `Fc` represents the center frequency (in Hz).

`IQ = RF2IQ(RF,Fs)` or `IQ = RF2IQ(RF,Fs,[],...)` calculates the carrier (center) frequency.

IMPORTANT: `Fc` **must be given** if the RF signal is undersampled (as in bandpass sampling).

`[IQ,Fc] = RF2IQ(...)` also returns the carrier (center) frequency (in Hz).

Note on fractional bandwidth

`RF2IQ` uses a downmixing process followed by low-pass filtering. The low-pass filter is determined by the normalized cut-off frequency `Wn`. By default `Wn = min(2*Fc/Fs,0.5)`. The cut-off frequency Wn can be adjusted if the fractional bandwidth (in %) is given:

`IQ = RF2IQ(RF,Fs,Fc,B)`

The fractional bandwidth (in %) is defined by:

`B` = Bandwidth_in_% = Bandwidth_in_Hz × (100/Fc).

When `B` is an input variable, the cut-off frequency is calculated as follows:

`Wn` = Bandwidth_in_Hz / `Fs`, i.e:

`Wn = B × (Fc/100)/Fs`.

Other syntax

An alternative syntax for `RF2IQ` is the following:

`IQ = RF2IQ(RF,PARAM)`

where the structure `PARAM` must contain the required parameters:

1. `PARAM.fs`: sampling frequency (in Hz, **REQUIRED**)
2. `PARAM.fc`: center frequency (in Hz, OPTIONAL, required for undersampled RF signals)
3. `PARAM.bandwidth`: fractional bandwidth (in %, OPTIONAL)

4. PARAM.t0: time offset (in s, OPTIONAL, default = 0)

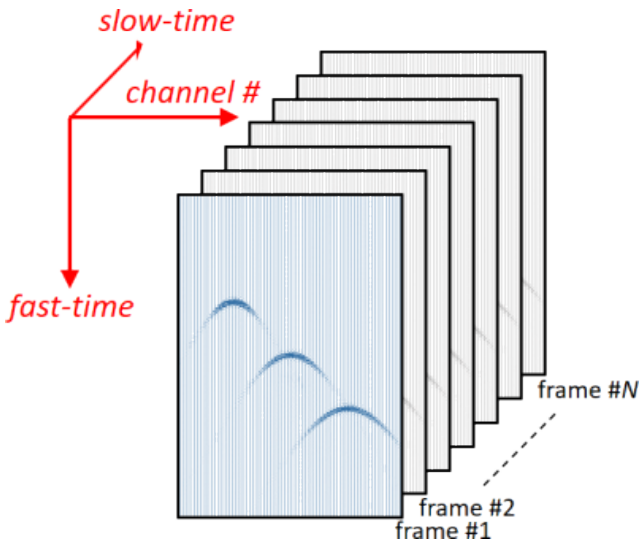
Additional notes

- **NOTE #1: Undersampling (sub-Nyquist sampling)**

If the RF signal is undersampled, the carrier frequency Fc **must be specified**. If a relative bandwidth (B or PARAM.bandwidth) is given, a warning message appears if harmful aliasing is suspected.

- **NOTE #2: Dimensions**

RF2IQ treats the data along the first non-singleton dimension as vectors, i.e. RF2IQ demodulates along columns for 2-D and 3-D RF data. Each column corresponds to a single RF signal over (fast-) time. Use IQ2RF to recover the RF signals.



- **NOTE #3: Method**

RF2IQ multiplies RF by a phasor of frequency Fc (down-mixing) and applies a fifth-order Butterworth lowpass filter using FILTFILT:

```
IQ = RF.*exp(-1i*2*pi*Fc*t);
[b,a] = butter(5,2*Fc/Fs);
IQ = filtfilt(b,a,IQ)*2;
```

Example #1: Envelope of an RF signal

This example shows how to obtain the real envelope of an RF signal by I/Q demodulation

Load an RF signal sampled at 20 MHz.

```
load RFsignal@20MHz.mat
```

Demodulate by using RF2IQ.

```
Fs = 20e6; % sampling frequency (in Hz)
[IQ,Fc] = rf2iq(RF,20e6);
disp(['The center frequency is ' num2str(Fc*1e-3,'%.1f') ' kHz.'])
```

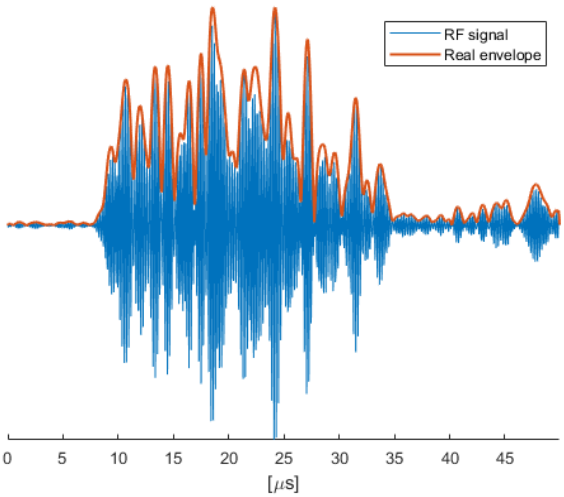
```
The center frequency is 4936.4 kHz.
```

Calculate the real envelope.

```
RE = abs(IQ);
```

Display the RF signal and its envelope.

```
t = (0:numel(RF)-1)/Fs*1e6; % time (in microseconds)
plot(t,RF)
set(gca,'YColor','none','box','off')
xlabel('[\mus]')
hold on
plot(t,RE,'Linewidth',1.5)
hold off
axis tight
legend({'RF signal','Real envelope'})
```



Example #2: Demodulation of an undersampled RF signal

This example illustrates the I/Q demodulation of an undersampled RF signal.

Load an RF signal sampled at 20 MHz (center frequency: 5 MHz).

```
load RFsignal@20MHz.mat
Fc = 5e6; % center frequency (in Hz)
Fs = 20e6; % sampling frequency (in Hz)
```

Demodulate the original RF signal.

```
IQ = rf2iq(RF,Fs,Fc);
```

Create an undersampled RF signal (sampled at Fs/5 = 4 MHz).

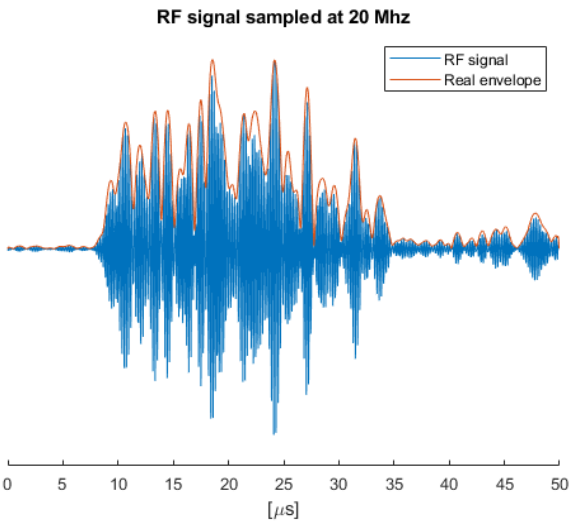
```
factor = 5; % undersampling factor
usRF = RF(1:factor:end);
usFs = Fs/factor; % (under)sampling frequency
```

Demodulate the undersampled RF signal.

```
usIQ = rf2iq(usRF,usFs,Fc);
```

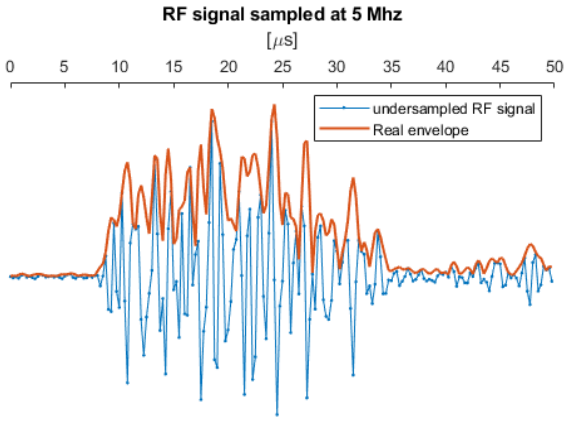
Display the original RF signal and its envelope.

```
t = (0:numel(RF)-1)/Fs*1e6; % time (in microseconds)
plot(t,RF,t,abs(IQ))
set(gca,'YColor','none','box','off')
title('RF signal sampled at 20 Mhz')
xlabel('[\mus]')
legend({'RF signal','Real envelope'})
```



Display the undersampled RF signal and its envelope.

```
t = (0:numel(usRF)-1)/usFs*1e6; % time (in microseconds)
plot(t,usRF,'.-')
hold on
plot(t,abs(usIQ),'LineWidth',1.5)
hold off
set(gca,'YColor','none','box','off','XAxisLocation','top')
xlabel('[\mus]')
title('RF signal sampled at 5 Mhz')
legend({'undersampled RF signal','Real envelope'})
```



See also

iq2doppler, bmode, wfilt

Reference

- Madiena C, Faurie J, Porée J, Garcia D. **Color and vector flow imaging in parallel ultrasound with sub-Nyquist sampling**. *IEEE Trans Ultrason Ferroelectr Freq Control*, 2018;65:795-802. **(PDF)**

Last update

2020/05

SIMUS Simulation of ultrasound RF signals for a linear or convex array

SIMUS simulates ultrasound RF radio-frequency signals generated by an ultrasound uniform LINEAR or CONVEX array insonifying a medium of scatterers.

Syntax

`RF = SIMUS(X,Y,Z,RC,DELAYS,PARAM)` simulates ultrasound RF radio- frequency signals generated by an ultrasound uniform *linear* or *convex* array insonifying a medium of scatterers. The scatterers are characterized by their coordinates (`X`, `Y`, `Z`) and reflection coefficients `RC`.

`X`, `Y`, `Z` and `RC` must be of same size. The elements of the ULA are excited at different time delays, given by the vector `DELAYS`. The transmission and reception characteristics must be given in the structure `PARAM` (see below for details).

`RF = SIMUS(X,Z,RC,DELAYS,PARAM)` or `RF = SIMUS(X,[],Z,RC,DELAYS,PARAM)` disregards elevation focusing (`PARAM.focus` is ignored) and assumes that `Y`=0 (2-D space). The computation is faster in 2-D.

▶ **TRY IT!** Enter `simus` in the command window for an example.

The `RF` output matrix contains `Number_of_Elements` columns. Each column therefore represents an RF signal. The number of rows depends on the depth (estimated from `max(Z)`) and the sampling frequency `PARAM.fs` (see below). By default, the sampling frequency is four times the center frequency.

Units: `X`, `Y`, `Z` must be in m; `DELAYS` must be in s; `RC` has no unit. `RF` has arbitrary unit.

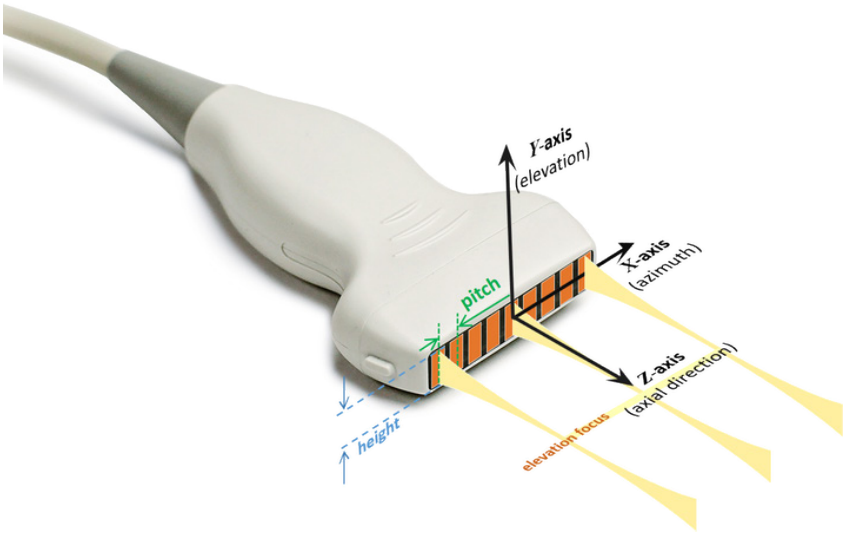
`DELAYS` can also be a matrix. This alternative can be used to simulate MLT (multi-line transmit) sequences. In this case, each ROW represents a delay series. For example, to create a 4-MLT sequence with a 64-element phased array, `DELAYS` matrix must have 4 rows and 64 columns (size = [4 64]).

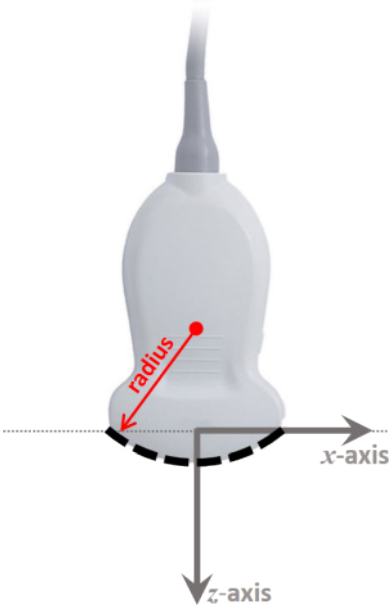
`SIMUS` uses `PFIELD` during transmission and reception. The parameters that must be included in the structure `PARAM` are similar as those in `PFIELD`. Additional parameters are also required (see below).

Uniform linear array (ULA)

The ***pitch*** is defined as the center-to-center distance between two adjacent elements. The ***kerf*** width is the distance that separates two adjacent elements. They are constant for a uniform linear array (ULA).

Some functions of the MUST toolbox can also consider curved (convex) ULAs.





The structure PARAM

PARAM is a structure that contains the following fields:

- **TRANSDUCER PROPERTIES**

1. PARAM.`fc`: central frequency (in Hz, **required**)
2. PARAM.`pitch`: pitch of the linear array (in m, **required**)
3. PARAM.`width`: element width **OR** PARAM.`kerf`: kerf width (in m, **required**; width = pitch-kerf)
4. PARAM.`focus`: elevation focus (in m, ignored if Y is not given). The default is Inf (no elevation focusing)
5. PARAM.`height`: element height (in m, ignored if Y is not given). The default is Inf (no elevation focusing)
6. PARAM.`radius`: radius of curvature (in m). The default is `Inf` (rectilinear array)

7. PARAM.bandwidth: pulse-echo (2-way) 6dB fractional bandwidth (in %). The default is 75%.
8. PARAM.baffle: property of the baffle: 'soft' (default), 'rigid' or a scalar > 0. See ***Note on BAFFLE property*** below for details.

- **MEDIUM PARAMETERS**

1. PARAM.c: longitudinal velocity (in m/s, default = 1540 m/s)
2. PARAM.attenuation: attenuation coefficient (dB/cm/MHz, default: 0). A linear frequency-dependence is assumed. A typical value for soft tissues is ~0.5 dB/cm/MHz.

- **TRANSMIT PARAMETERS**

1. PARAM.TXapodization: transmision apodization (default: no apodization)
2. PARAM.TXnow: number of wavelengths of the TX pulse (default: 1). Use PARAM.TXnow = Inf for a mono-harmonic signal.
3. PARAM.TXfreqsweep: frequency sweep for a linear chirp (default: [ ]). To be used to simulate a linear TX chirp. See ***Note on CHIRP signals*** below for details.

- **RECEIVE PARAMETERS** (not in PFIELD)

1. PARAM.fs: sampling frequency (in Hz, default = 4*param.fc).
2. PARAM.RXdelay: reception law delays (in s, default = 0)

Other syntaxes

[RP,PARAM] = SIMUS(...) also returns the complete list of parameters including the default values.

[...] = SIMUS without any input argument provides an interactive example designed to produce RF signals from a focused ultrasound beam using a 2.7 MHz phased-array transducer.

Parallel computing

SIMUS calls the function PFIELD. If you have the Parallel Computing Toolbox, SIMUS can run several PFIELD functions in parallel. If this option is activated, a parallel pool is created on the default cluster. All workers in the pool are used. The X, Y and Z are split into NW chunks, NW being the number of workers. To execute parallel computing, use:

[...] = SIMUS(...,OPTIONS),

with OPTIONS.ParPool = true (default = false).

X-, Y-, and Z-axes

Conventional axes are used:

1. For a **LINEAR** array, the X-axis is PARALLEL to the transducer and points from the first (leftmost) element to the last (rightmost) element (`X = 0` at the CENTER of the transducer). The Z-axis is PERPENDICULAR to the transducer and points downward (`Z = 0` at the level of the transducer, Z increases as depth increases). The Y-axis is such that the coordinates are right-handed. These axes are represented in the above ULA figure.
2. For a **CONVEX** array, the X-axis is parallel to the chord and `Z = 0` at the level of the chord.

Element directivity

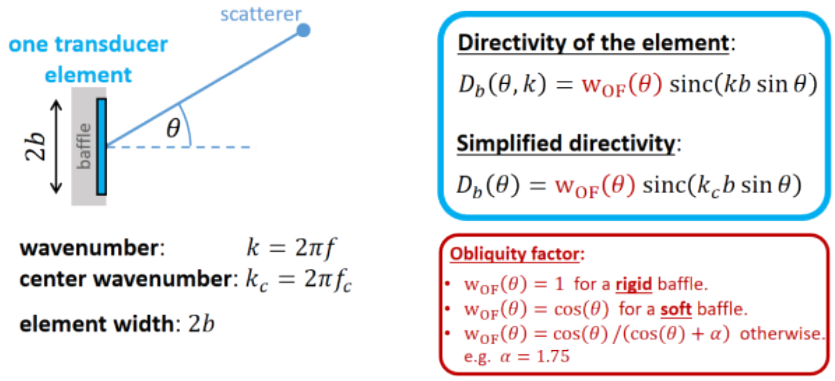
By default, the calculation is made faster by assuming that the directivity of the elements is dependent only on the central frequency (see figure below). This simplification very little affects the pressure field in most situations (except near the array). To turn off this option, use `OPTIONS.FullFrequencyDirectivity = true`.

See ***ADVANCED OPTIONS*** below.

Note on BAFFLE property

In `PFIELD`, it is assumed by default that the array elements are embedded in an infinite **SOFT** baffle. To modify the property of the baffle, modify the field `PARAM.baffle`:

4. `'rigid'`
5. `'soft'` (this is the default)
6. a nonnegative scalar α , with α = (medium impedance)/(baffle impedance). Note: $\alpha = 0 \Longrightarrow$ `'rigid'`; $\alpha \gg 1 \Longrightarrow$ `'soft'`.



The baffle property affects the obliquity factor included in the directivity of the elements. This obliquity factor is not 1 if the baffle is not rigid. A general case (see case #3 below) can be chosen by specifying an impedance ratio. For details, refer to the corresponding papers.

4. Rigid baffle $\Longrightarrow$ obliquity factor = 1.
5. Soft baffle $\Longrightarrow$ obliquity factor = $\cos(\theta)$.

6. General baffle ⟹ obliquity factor = cos(θ)/(cos(θ) + α), with α = (medium impedance)/(baffle impedance).

References for baffle models:

- Selfridge *et al*. *A theory for the radiation pattern of a narrow-strip acoustic transducer*. Appl Phys Lett 37(1), 35-36 (1980)
- Pesqué *et al*. *Effect of the planar baffle impedance in acoustic radiation of a phased array element theory and experimentation*. IEEE Ultrasonics Symposium, (1984)

Example: For a baffle of impedance 2.8 MRayl (epoxy) adjacent to soft tissues of impedance 1.6 MRayls, α = 1.75.

Note on CHIRP signals

Linear chirps are characterized by `PARAM.TXnow`, `PARAM.fc` and `PARAM.TXfreqsweep`. The transmitted pulse has a duration of approximately `T` (= `PARAM.TXnow/PARAM.fc`), with the amplitude and phase defined over the time interval `-T/2` to `+T/2`.

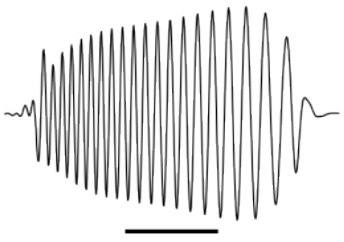
The total frequency sweep is `DeltaF` (= `PARAM.TXfreqsweep`): the frequencies increase linearly from (`PARAM.fc - DeltaF/2`) to (`PARAM.fc + DeltaF/2`) in the defined time interval.

Documentation: Chirp spectrum in Wikipedia.

Examples of pulse responses

Here are two examples of `PFIELD`-derived pulse responses:

```
param.fc = 7.2e6
param.bandwidth = 100

param.TXnow = 20
param.TXfreqsweep = 7e6
```




```
param.fc = 7.2e6
param.bandwidth = 60

param.TXnow = 1
param.TXfreqsweep = []
```

Advanced options

- **FREQUENCY STEP & SAMPLES**

Only frequency components of the transmitted signal in the range [0, 2fc] with significant amplitude are considered. The default relative amplitude is -100 dB in SIMUS. You can change this value by using the following:

```
[...] = SIMUS(...,OPTIONS),
```

where OPTIONS.dBThresh is the threshold in dB (default = −100).

The frequency step is determined automatically to avoid aliasing in the time domain. This step can be adjusted with a scaling factor OPTIONS.FrequencyStep (default = 1). It is not recommended to modify this scaling factor in SIMUS.

- **FULL-FREQUENCY DIRECTIVITY**

By default, the directivity of the elements depends on the center frequency only. This makes the algorithm faster. To make the directivities fully frequency-dependent, use:

```
[...] = SIMUS(...,OPTIONS),
```

with OPTIONS.FullFrequencyDirectivity = true (default = false).

- **ELEMENT SPLITTING**

Each transducer element of the array is split into small segments. The length of these small segments must be small enough to ensure that the far-field model is accurate. By default, the elements are split into M segments, with M being defined by:

```
M = ceil(element_width/smallest_wavelength);
```

To modify the number M of subelements by splitting, you may adjust OPTIONS.ElementSplitting. For example, OPTIONS.ElementSplitting = 1.

- **WAIT BAR**

If OPTIONS.WaitBar is true, a wait bar appears (only if the number of frequency samples >10). Default is true.

Example #1: RF signals using a phased-array transducer

This example shows how to simulate RF signals obtained with a focused wave generated by a cardiac phased-array transducer.

Download the properties of a 2.7-MHz 64-element cardiac phased array in a structure `param` by using GETPARAM.

```
param = getparam('P4-2v');
```

Choose a focus location at xf = 0 cm, zf = 5 cm.

```
xf = 0; zf = 5e-2; % focus position (in m)
```

Obtain the corresponding transmit time delays (in s).

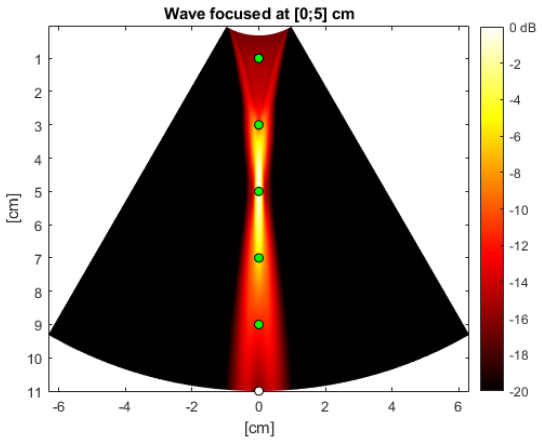
```
txdel = txdelay(xf,zf,param); % in s
```

Define scatterers.

```
x = zeros(1,6); y = zeros(1,6); z = (1:2:11)*1e-2; % scatterers' positions
RC = [ones(1,5) 0]; % reflection coefficients
% The RC of the deepest scatterer is zero. It is a "ghost" scatterer,
% which ensures that |SIMUS| simulates RF signals down to its location.
```

Check the pressure field with PFIELD.

```
[xi,zi] = impolgrid([256 128],11e-2,pi/3,param); % polar-type image grid
yi = zeros(size(xi));
P = pfield(xi,yi,zi,txdel,param);

pcolor(xi*1e2,zi*1e2,20*log10(P/max(P(:))))
shading interp
colormap hot
axis equal tight ij
caxis([-20 0]) % dynamic range = [-20,0] dB
c = colorbar;
c.YTickLabel{end} = '0 dB';
xlabel('[cm]'), ylabel('[cm]')
title('Wave focused at [0;5] cm')

hold on
plot(x(1:5)*1e2,z(1:5)*1e2,'ko','MarkerFaceColor','g')
plot(x(6)*1e2,z(6)*1e2,'ko','MarkerFaceColor','w')
hold off
```



Use a high sampling frequency to obtain RF signals with high temporal resolution (the default is 4 times the center frequency). It does not affect computational time.

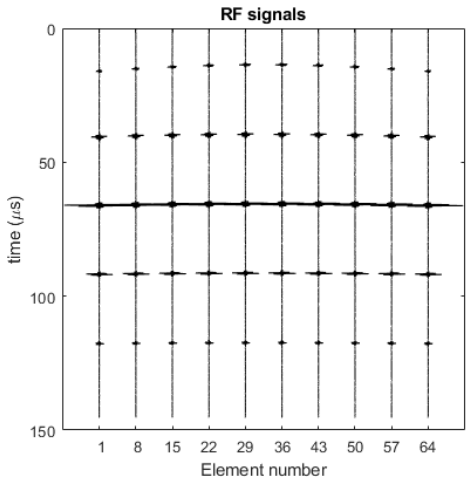
```
param.fs = 20*param.fc; % sampling frequency
```

Simulate RF signal by using `SIMUS`.

```
RF = simus(x,y,z,RC,txdel,param);
```

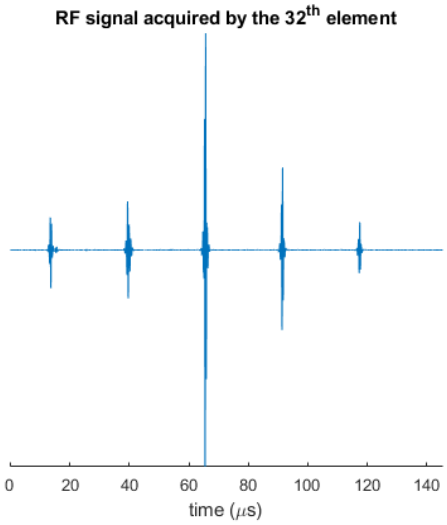
Plot the RF signals.

```
plot((RF(:,1:7:64)/max(RF(:))+(1:10))',...
 (0:size(RF,1)-1)/param.fs*1e6,'k')
set(gca,'XTick',1:10,'XTickLabel',int2str((1:7:64)'))
title('RF signals')
xlabel('Element number'), ylabel('time (\mus)')
xlim([0 11])
axis ij square
```



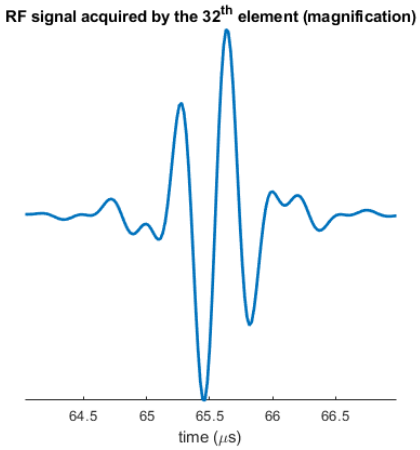
Plot the RF signal acquired by the 32th element.

```
t = (0:size(RF,1)-1)/param.fs; % time (s)
plot(t*1e6,RF(:,32))
set(gca,'YColor','none','box','off')
xlabel('time (\mus)')
title('RF signal acquired by the 32^{th} element')
axis tight square
```



Magnify around the focus location.

```
c = 1540; % 1540 m/s is the default speed of sound
tf = 2*5e-2/c; % 2-way time to focus point
idx = abs(t-tf-txdel(32))<4/param.fc;
plot(t(idx)*1e6,RF(idx,32),'LineWidth',2)
set(gca,'YColor','none','box','off')
xlabel('time (\mus)')
title('RF signal acquired by the 32^{th} element (magnification)')
axis tight square
```



Create a GIF movie named 'focusedWave.gif' with MKMOVIE.

```
param.movie = [5 10 30]; % 5-cm-by-10-cm ROI, resolution = 30 pix/cm
mkmovie(x,z,RC,txdel,param,'focusedWave.gif');
```

Example #2: RF signals in plane wave imaging

This example shows how to simulate RF signals obtained with a plane wave generated by a linear array. The RF signals will be demodulated and beamformed to obtain a B-mode image.

Download the properties of a 7.6-MHz 128-element uniform linear array in a structure param by using GETPARAM.

```
param = getparam('L11-5v');
```

Generate 20 random scatterers.

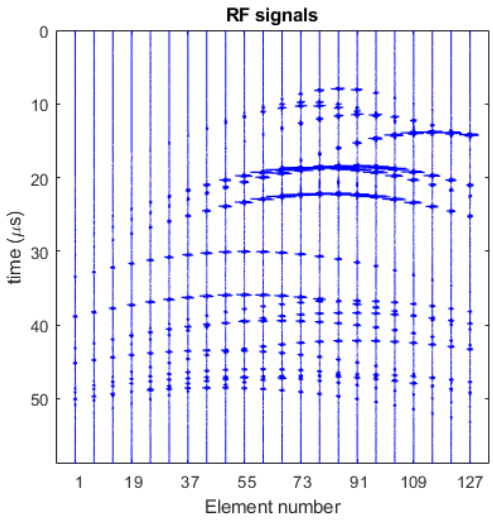
```
n = 20;
x = rand(1,n)*3.8e-2-1.9e-2;
y = zeros(1,n);
z = rand(1,n)*3.5e-2+0.25e-2;
RC = 1/2+rand(1,n)/2; % reflection coefficients
```

Simulate RF signals by using SIMUS. The transmit delays are all zero to create an unsteered plane wave.

```
param.fs = 10*param.fc;
txdel = zeros(1,128); % null delays
RF = simus(x,y,z,RC,txdel,param);
```

Plot the RF signals.

```
RF = RF/max(abs(RF(:)));
t = (0:size(RF,1)-1)/param.fs*1e6; % in microseconds
plot((2*RF(:,1:6:128)+(1:22))',t,'b')
set(gca,'XTick',1:3:22,'XTickLabel',int2str((1:18:128)'))
title('RF signals')
xlabel('Element number'), ylabel('time (\mus)')
axis([0 23 0 t(end)])
axis ij square
```

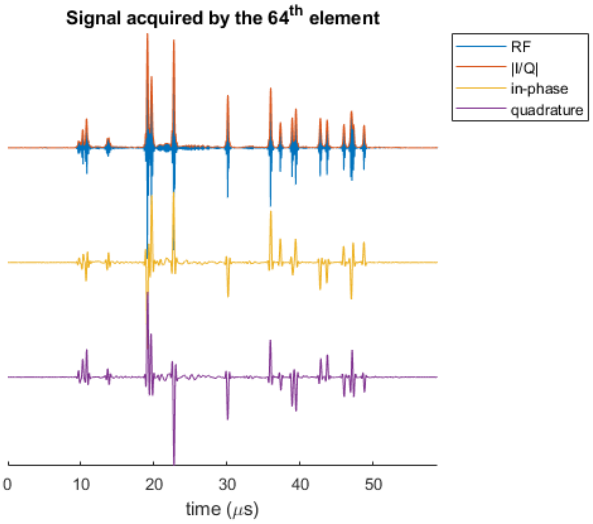


Demodulate the RF signals to obtain I/Q signals

```
IQ = rf2iq(RF,param.fs,param.fc);
```

Plot the RF signal acquired by the 64th element and the corresponding I/Q.

```
RF64 = RF(:,64); IQ64 = IQ(:,64); % 64th signals
p = max(abs(IQ64)); % peak
plot(t,RF64/p+2,t,abs(IQ64)/p+2,t,real(IQ64)/p+1,t,imag(IQ64)/p)
set(gca,'YColor','none','box','off')
xlabel('time (\mus)')
title('Signal acquired by the 64^{th} element')
axis tight
legend({'RF','|I/Q|','in-phase','quadrature'},...
    'Location','NorthEastOutside')
```



Define a 256 × 256 image grid.

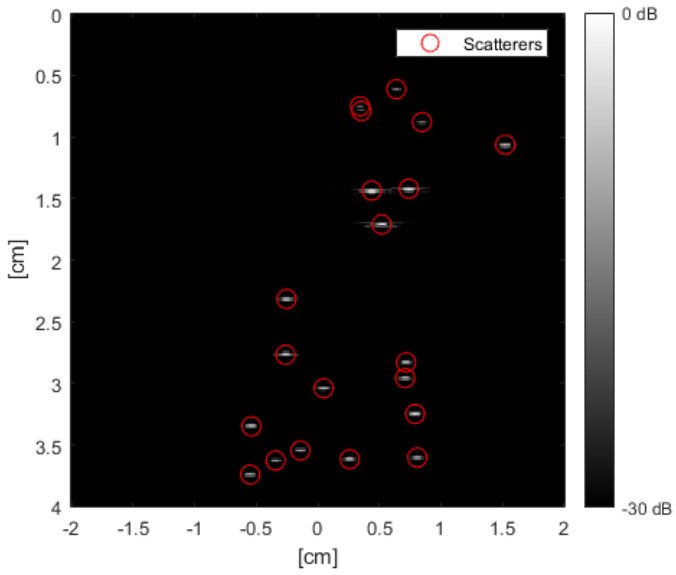
```
xi = linspace(-2e-2,2e-2,256); % in m
zi = linspace(0,4e-2,256); % in m
[xi,zi] = meshgrid(xi,zi);
```

Beamform the I/Q signals by using DAS. Use a void f-number parameter. It will be determined automatically (and optimally).

```
param.fnumber = [];
IQb = das(IQ,xi,zi,txdel,param); % beamformed I/Q
```

Create and display a B-mode image.

```
B = bmode(IQb,30); % log-compressed image
imagesc(xi(1,:)*1e2,zi(:,1)*1e2,B)
c = colorbar;
c.YTick = [0 255];
c.YTickLabel = {'-30 dB','0 dB'};
colormap gray
axis equal tight
xlabel('[cm]'), ylabel('[cm]')

hold on
plot(x*1e2,z*1e2,'ro','MarkerSize',10)
hold off
legend('Scatterers')
```



Example #3: M87-black-hole ultrasound image with diverging waves

This example shows how to simulate an ultrasound image with a series of circular waves generated by a cardiac phased array. The RF signals will be demodulated, beamformed, then merged to obtain a compound B-mode image.

Download the properties of a 2.7-MHz 64-element linear phased array in a structure `param` by using GETPARAM.

```
param = getparam('P4-2v');
```

Center wavelength.

```
param.c = 1540; % speed of sound (m/s)
lambda = param.c/param.fc;
```

Download an image of the supermassive M87 black hole.

```
I = imread(['https://static.projects.iq.harvard.edu/files/styles/',...
    'os_files_xlarge/public/eht/files/20190410-78m-800x466.png']);
```

Simulate scatterers in a 5-cm ✕ 5-cm region.

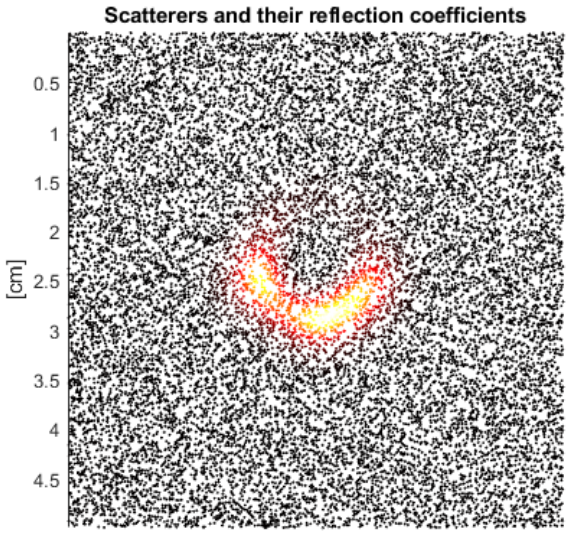
Image grid for a 5-cm deep image

```
[nl,nc,~] = size(I);
L = 5e-2;
[xi,zi] = meshgrid(linspace(0,L,nc)*nc/nl,linspace(0,L,nl));
xi = xi-L/2*nc/nl; % recenter xi
```

Obtain randomly distributed scatterers by interpolation.

```
scatdens = 1.5; % scatterer density per lambda^2 (you may modify it)
Ns = round(scatdens*L^2*nc/nl/lambda^2); % number of scatterers

xs = rand(1,Ns)*L-L/2; % scatterer locations
zs = rand(1,Ns)*L;

Ig = rgb2gray(I); % convert the RGB image to gray

F = scatteredInterpolant(xi(:),zi(:),double(Ig(:))/255);
g = 0.4; % this parameter adjusts the RC values
RC = F(xs,zs).^(1/g); % reflection coefficients

scatter(xs*1e2,zs*1e2,3,RC,'filled')
axis equal ij tight
colormap hot
set(gca,'XColor','none','box','off')
```

```
ylabel('[cm]')
title('Scatterers and their reflection coefficients')
```



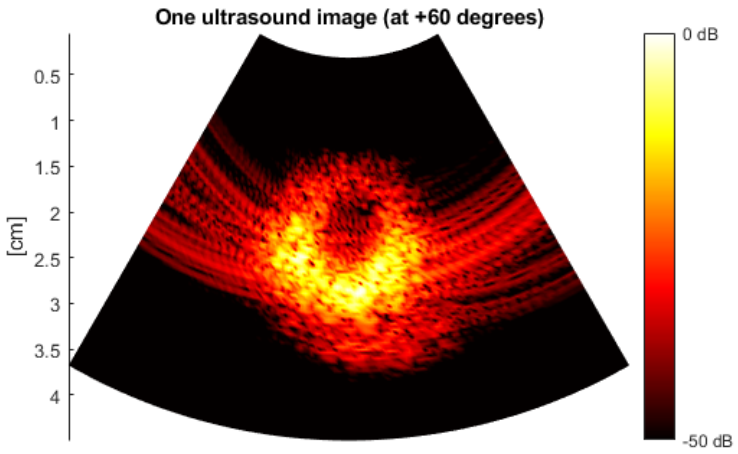
Simulate 21 sets of RF signals obtained with 21 circular waves tilted at different angles, beamform these signals onto a 128 × 128 polar grid, and obtain a compound I/Q dataset.

```
tilt = linspace(-pi/6,pi/6,21); % tilt angles
IQc = zeros(128,128,'like',1i); % will contain the compound I/Q

opt.WaitBar = false; % no progress bar for SIMUS
param.fs = param.fc*4; % RF sampling frequency
[xI,zI] = impolgrid(128,4.5e-2,pi/3,param); % polar-type grid

h = waitbar(0,'SIMUS & DAS...');
for k = 1:21
    dels = txdelay(param,tilt(k),pi/3); % transmit delays
    RF = simus(xs,zs,RC,dels,param,opt); % RF simulation
    IQ = rf2iq(RF,param); % I/Q demodulation
    IQb = das(IQ,xI,zI,dels,param); % DAS beamforming
    IQc = IQc+IQb; % compounding
    waitbar(k/length(tilt),h,...
        ['SIMUS & DAS: ' int2str(k) ' of 21 completed'])
end
close(h)
```

Display the last ultrasound image (at 60-degree steering)

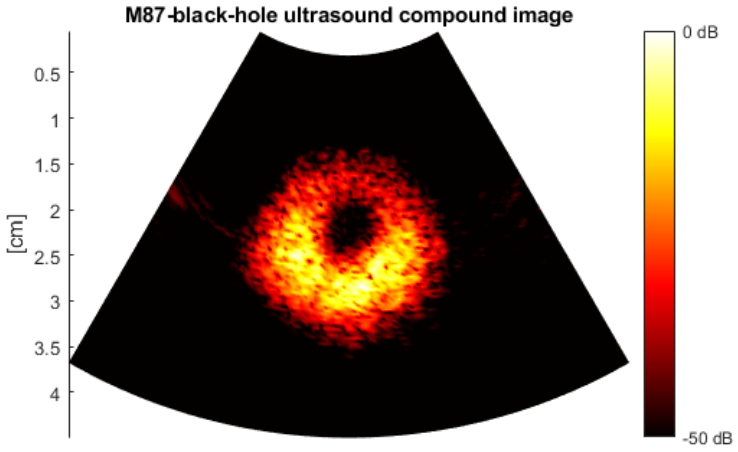
```
lcI = bmode(IQb,50); % log-compressed image
pcolor(xI*1e2,zI*1e2,lcI)
shading interp, axis equal ij tight
c = colorbar;
```

```
c.YTick = [0 255];
c.YTickLabel = {'-50 dB','0 dB'};
colormap gray
axis equal tight
ylabel('[cm]')
set(gca,'XColor','none','box','off')
colormap hot
title('One ultrasound image (at +60 degrees)')
```



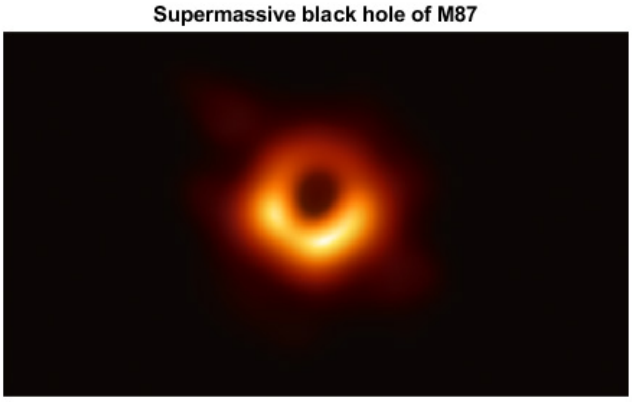
Display the M87-black-hole ultrasound compound image

```
lcI = bmode(IQc,50); % log-compressed image
pcolor(xI*1e2,zI*1e2,lcI)
shading interp, axis equal ij tight
c = colorbar;
c.YTick = [0 255];
c.YTickLabel = {'-50 dB','0 dB'};
colormap gray
axis equal tight
ylabel('[cm]')
set(gca,'XColor','none','box','off')
colormap hot
title('M87-black-hole ultrasound compound image')
```



Display the original picture of the M87 black hole.

```
imshow(I,'InitialMagnification','fit')
title('Supermassive black hole of M87')
```



Example #4: Recover a few scatterers with a convex array

This example shows how to simulate an ultrasound image with one circular wave generated by a curved linear array. The RF signals will be demodulated, then beamformed to obtain a B-mode image.

Download the properties of a 3.6-MHz 128-element convex array in a structure `param` by using GETPARAM.

```
param = getparam('C5-2v');
```

Design a few scatterers.

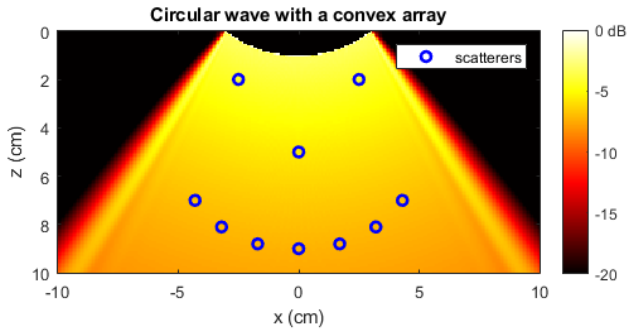
```
xs = [4.3 3.2 1.7 0 -1.7 -3.2 -4.3 0 -2.5 2.5]*1e-2; % in m
zs = [7 8.1 8.8 9 8.8 8.1 7 5 2 2]*1e-2; % in m
```

Set all the transmit delays to 0.

```
txdel = zeros(1,param.Nelements);
```

Calculate and display the pressure field.

```
[x,z] = meshgrid(linspace(-.1,.1,256),linspace(0,.1,128));
P = pfield(x,z,txdel,param);

imagesc(x(1,:)*1e2,z(:,1)*1e2,20*log10(P/max(P,[],'all')))
axis equal ij tight
caxis([-20 0]) % dynamic range = [-20,0] dB
c = colorbar;
c.YTickLabel{end} = '0 dB';
colormap hot
xlabel('x (cm)'), ylabel('z (cm)')
title('Circular wave with a convex array')

hold on
plot(xs*1e2,zs*1e2,'bo','Linewidth',2)
hold off
legend('scatterers')
```



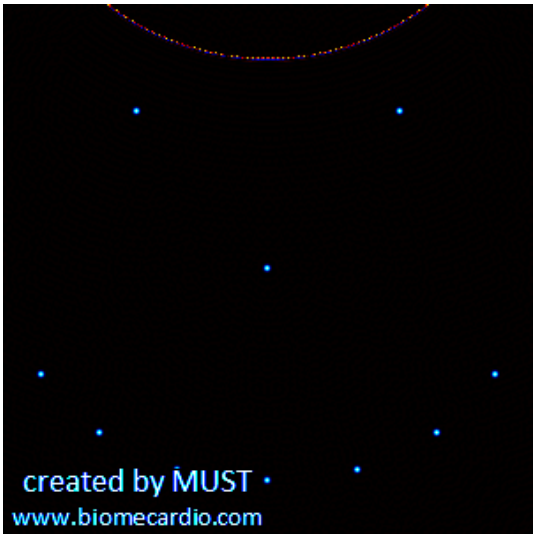
Simulate RF signals with SIMUS.

First add a "ghost" scatterer (its reflection coefficient is zero) at 10 cm deep to ensure that RF are simulated till 10-cm depth.

```
RC = [ones(1,numel(xs)) 0];
xs = [xs 0]; zs = [zs 1e-2];

RF = simus(xs,zs,RC,txdel,param);
```

Create a GIF movie named 'convexSmiley.gif' with MKMOVIE.

```
param.movie = [10 10 30]; % 10-cm-by-10-cm ROI, resolution = 30 pix/cm
mkmovie(xs,zs,RC,txdel,param,'convexSmiley.gif');
```

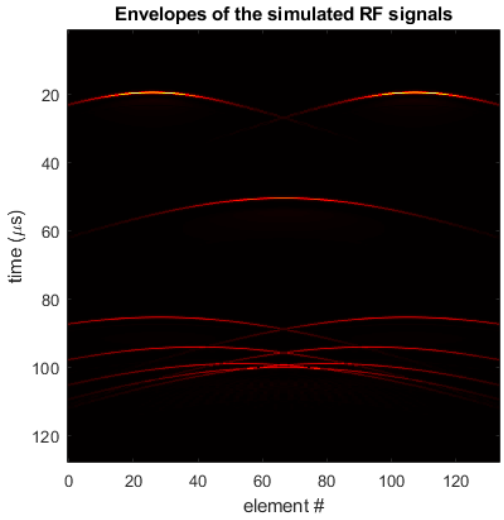


Demodulate the RF signals with RF2IQ.

```
param.fs = 4*param.fc; % it is the default sampling frequency used by SIMUS
IQ = rf2iq(RF,param.fs,param.fc);
```

Display the hyperbolic signatures.

```
imagesc((0:size(RF,1))/param.fs*1e6,1:128,abs(IQ))
xlabel('element #')
ylabel('time (\mus)')
axis square
title('Envelopes of the simulated RF signals')
```



Define a 256 × 512 image grid

```
[x,z] = impolgrid([256 512],0.1,param);
```

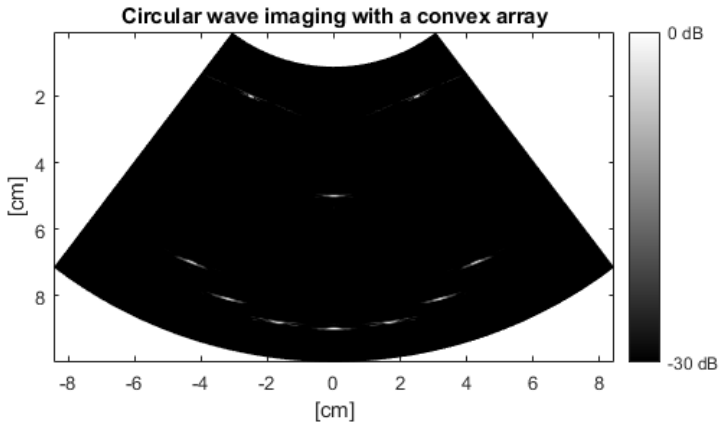
Beamform the I/Q signals onto this grid.

```
param.fnumber = []; % the f-number will be chosen automatically
IQb = das(IQ,x,z,txdel,param);
```

Display the resulting image.

```
B = bmode(IQb,30); % B-mode image
pcolor(x*1e2,z*1e2,B)
shading interp
axis equal ij tight
c = colorbar;
c.YTick = [0 255];
c.YTickLabel = {'-30 dB','0 dB'};
colormap gray
```

```
xlabel('[cm]'), ylabel('[cm]')
title('Circular wave imaging with a convex array')
```



See also

txdelay, impolgrid, pfield, mkmovie

References

- The paper that describes the first 2-D version of PFIELD is: Shariari S, Garcia D. **Meshfree simulations of ultrasound vector flow imaging using smoothed particle hydrodynamics**. *Phys Med Biol*, 2018;63:205011. **(PDF)**
- The paper that describes the theory of the full (2-D + 3-D) version of `PFIELD` will be submitted by February 2021.

Last update

2020/12

SPTRACK Speckle tracking

SPTRACK returns the motion field by speckle tracking.

Syntax

`[Di,Dj] = SPTRACK(I,PARAM)` returns the motion field `[Di,Dj]` that occurs from frame #k `I(:,:,k)` to frame #(k+1) `I(:,:,k+1)`.

`I` must be a 3-D array, with `I(:,:,k)` corresponding to image #k. `I` can contain more than two images (i.e. `size(I,3)`>2). In such a case, an ensemble correlation is used.

▶ **TRY IT!** Enter `sptrack` in the command window for an example.

`Di,Dj` are the displacements (unit = pix) in the IMAGE coordinate system (i.e. the "matrix" axes mode). The `i`-axis is vertical, with values increasing from top to bottom. The `j`-axis is horizontal with values increasing from left to right. The coordinate `(1,1)` corresponds to the center of the upper left pixel. To display the displacement field, you may use: `quiver(Dj,Di), axis ij`

PARAM is a structure that contains the parameter values required for speckle tracking (see below for details).

`[Di,Dj,id,jd] = SPTRACK(...)` also returns the coordinates of the points where the components of the displacement field are estimated.

PARAM is a structure that must contain the following fields:

- PARAM.`winsize`: size of the interrogation windows (**required**)

PARAM.`winsize` must be a 2-COLUMN array. If PARAM.`winsize` contains several rows, a multi-grid, multiple-pass interrogation process is used.

Examples:

a) If PARAM.`winsize = [64 32]`, then a 64 × 32 (64 lines, 32 columns) interrogation window is used.

b) If PARAM.`winsize = [64 64;32 32;16 16]`, a 64 × 64 interrogation window is first used. Then a 32 × 32 window and finally a 16 × 16 window are used.

- PARAM.`overlap`: overlap between the interrogation windows (in %, default = 50)

- PARAM.`iminc`: image increment (for ensemble correlation, default = 1)

The image #k is compared with image #(k+PARAM.iminc): `I(:,:,k)` is compared with `I(:,:,k+PARAM.iminc)`

- `PARAM.ROI`: 2-D region of interest (default = the whole image)

`PARAM.ROI` must be a logical 2-D array with a size of `[size(I,1),size(I,2)]`. The default is `all(isfinite(I),3)`.

Notes

1. The displacement field is returned in PIXELS. Perform an appropriate calibration to get physical units.
2. SPTRACK is based on a multi-step cross-correlation method. The SMOOTHN function (see Reference below) is used at each iterative step for the validation and post-processing.

Example #1: Speckle tracking of a rotating image

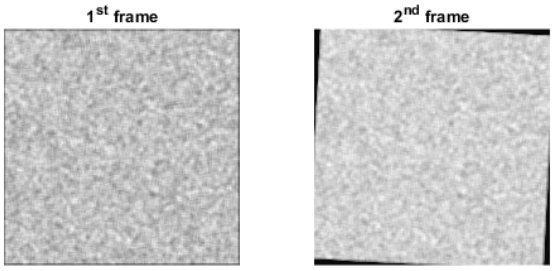
This example shows how to obtain the motion field of a rotating image

Create a 500-by-500 image.

```
I1 = conv2(rand(500,500),ones(10,10),'same');
```

Rotate it in the clockwise direction.

```
I2 = imrotate(I1,-3,'bicubic','crop');
subplot(121), imagesc(I1), axis square off
title('1^{st} frame')
subplot(122), imagesc(I2), axis square off
title('2^{nd} frame')
colormap gray
```

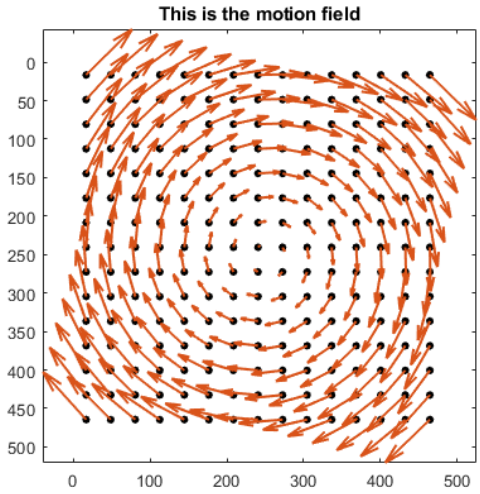


Recover the motion field with SPTRACK.

```
param = [];
param.winsize = [64 64;32 32];
[Di,Dj,id,jd] = sptrack(cat(3,I1,I2),param);
```

Display the motion field.

```
figure
plot(jd(1:2:end,1:2:end),id(1:2:end,1:2:end),'ko',...
    'MarkerFaceColor','k','MarkerSize',4)
hold on
h = quiver(jd(1:2:end,1:2:end),id(1:2:end,1:2:end),...
    Dj(1:2:end,1:2:end),Di(1:2:end,1:2:end),2);
hold off
set(h,'LineWidth',1.5)
axis equal ij tight
title('This is the motion field')
```



Example #2: Speckle tracking with plane wave imaging

This example shows how to obtain the motion field of a rotating disk insonified with plane waves.

A rotating disk (diameter of 2 cm) was insonified by a series of 32 unsteered plane waves with a Verasonics scanner, and a linear transducer, at a PRF (pulse repetition frequency) of 10 kHz. The RF signals were **downsampled** at 4/3 × (5 MHz) = 6.66 MHz. The properties of the linear array were:

- 128 elements
- center frequency = 5 MHz
- pitch = 0.298 mm

Download the experimental RF data. The 3-D array RF contains 128 columns (as the transducer contained 128 elements), and its length is 32 in the third dimension (as 32 plane waves were transmitted).

```
load('PWI_disk.mat')
```

The structure `param` contains some experimental properties that are required for the following steps.

```
disp('''param'' is a structure whose fields are:')
disp(param)
```

```
'param' is a structure whose fields are:
           fs: 6.6667e+06
        pitch: 2.9800e-04
           fc: 5000000
            c: 1480
           t0: 9.9500e-06
          PRF: 10000
        width: 2.6200e-04
    Nelements: 128
      TXdelay: [1×128 double]
    bandwidth: 15
```

Demodulate the RF signals with `RF2IQ`.

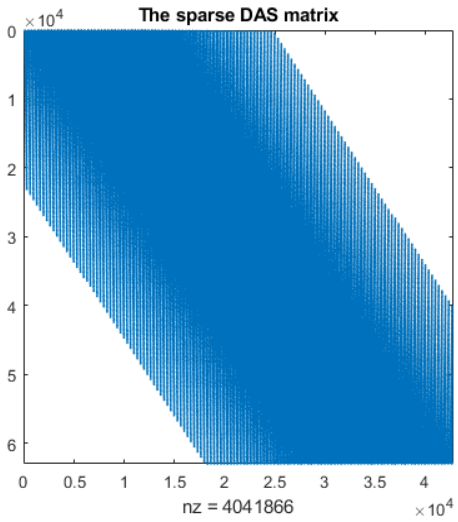
```
IQ = rf2iq(RF,param);
```

Create a 2.5-cm-by-2.5-cm image grid.

```
dx = 1e-4; % grid x-step (in m)
dz = 1e-4; % grid z-step (in m)
[x,z] = meshgrid(-1.25e-2:dx:1.25e-2,1e-2:dz:3.5e-2);
```

Create a Delay-And-Sum DAS matrix with `DASMTX`.

```
param.fnumber = []; % an f-number will be determined by DASMTX
M = dasmtx(1i*size(IQ),x,z,param,'nearest');
spy(M)
axis square
title('The sparse DAS matrix')
```



Beamform the I/Q signals.

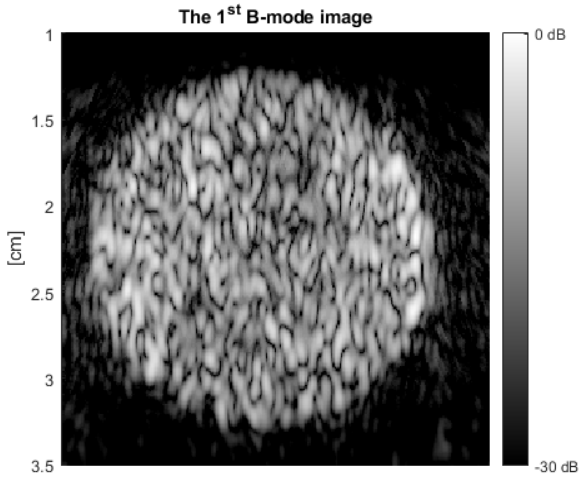
The 32 I/Q series can be beamformed simultaneously with the DAS matrix.

```
tic
IQb = M*reshape(IQ,[],32);
IQb = reshape(IQb,[size(x) 32]);
disp(['Beamforming: time per frame: ' num2str(toc/32*1e3,'%.1f'), ' ms'])
```

```
Beamforming: time per frame: 37.8 ms
```

Create the B-mode images with BMODE and display the first ultrasound image.

```
I = bmode(IQb,30); % B-mode images
image(x(1,:)*100,z(:,1)*100,I(:,:,1))
c = colorbar;
c.YTick = [0 255];
c.YTickLabel = {'-30 dB','0 dB'};
colormap gray
title('The 1^{st} B-mode image')
ylabel('[cm]')
axis equal tight ij
set(gca,'XColor','none','box','off')
```



Create an ROI.

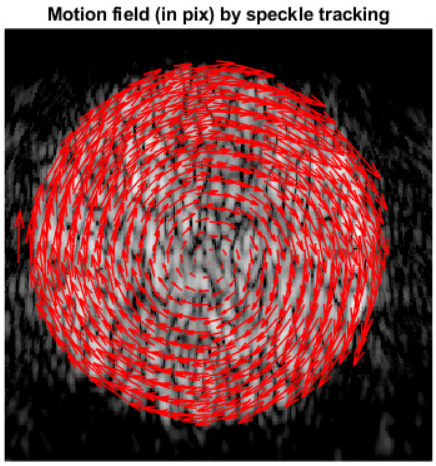
```
param.ROI = median(I,3)>64;
```

Track the speckles with SPTRACK.

```
param.winsize = [32 32; 24 24; 16 16]; % size of the subwindows
param.iminc = 4; % image increment
[Di,Dj,id,jd] = sptrack(I,param);
```

Display the motion field.

```
image(I(:,:,1))
colormap gray
hold on
h = quiver(jd,id,Dj,Di,3,'r');
set(h,'LineWidth',1)
hold off
title('Motion field (in pix) by speckle tracking')
axis equal off ij
```



See also

bmode, iq2doppler, dasmtx

References

- Garcia D, Lantelme P, Saloux É. **Introduction to speckle tracking in cardiac ultrasound imaging**. *Handbook of speckle filtering and tracking in cardiovascular ultrasound imaging and video. Institution of Engineering and Technology*, 2018. **(PDF)**
- Perrot V, Garcia D. **Back to basics in ultrasound velocimetry tracking speckles by using a standard PIV algorithm**. *IEEE International Ultrasonics Symposium (IUS)*, 2018. **(PDF)**
- Garcia D. **A fast all-in-one method for automated post-processing of PIV data**. *Experiments in Fluids*, 2011. **(PDF)**

Date modified

2020/06

TGC Time-gain compensation

Syntax

`TGC(RF)` or `TGC(IQ)` performs a time-gain compensation of the RF or I/Q signals using a decreasing exponential law. Each column of the RF or I/Q array must correspond to a single signal over (fast-) time.

`[~,C] = TGC(RF)` or `[~,C] = TGC(IQ)` also returns the coefficients used for time-gain compensation (i.e. `new_SIGNAL = C.*old_SIGNAL`)

Approach

The signal intensity is assumed to decrease exponentially as distance increases (and thus as fast-time increases). A robust linear fitting is performed on the intensity logarithm to seek the TGC exponential law.

See also

rf2iq, das, bmode

Last update

2020/05

TXDELAY Transmit delays for a uniform linear array

TXDELAY returns the transmit time delays for focused, plane or circular beam patterns in a linear array. **TXDELAY** also returns transmit time delays for focused waves in a curved array.

Syntax

`DELAYS = TXDELAY(X0,Z0,PARAM)` returns the transmit time delays which must be used to generate a pressure field focused at the point `(x0,z0)`. Note: If `z0` is negative, then the point `(x0,z0)` is a virtual source (i.e. emission of circular waves). The properties of the medium and uniform linear array must be given in the structure `PARAM` (see below).

`DELAYS = TXDELAY(PARAM,TILT)` returns the transmit time delays which must be used to get a tilted plane wave. `TILT` is the tilt angle, i.e the angle between the emission axis and `Z`-vertical axis.

`DELAYS = TXDELAY(PARAM,TILT,WIDTH)` yields the transmit time delays necessary for creating a circular wave. The sector enclosed by the circular waves is characterized by the angular width and tilt. `TILT` represents the sector tilt, WIDTH is the sector width (both in radians). **This option is not available for a convex array.**

`X0, Z0, TILT` and `WIDTH` can be vectors. In that case, `DELAYS` is a matrix whose rows contain the different delay laws.

`[DELAYS,PARAM] = TXDELAY(...)` updates the `PARAM` structure parameters including the default values. `PARAM` will also include `PARAM.TXdelay` which is equal to `DELAYS` (in s).

`[...] = TXDELAY` (no input parameter) runs an interactive example simulating a focused pressure field generated by a 2.7 MHz phased array. The user must choose the focus position.

Units: `X0` and `Z0` must be in m; `TILT` and `WIDTH` must be in rad. `DELAYS` are in s.

The structure PARAM

`PARAM` is a structure which must contain the following fields:

1. `PARAM.pitch`: pitch of the linear array (in m, **required**)
2. `PARAM.Nelements`: number of elements in the transducer array (**required**)
3. `PARAM.c`: longitudinal velocity (in m/s, default = 1540 m/s)

Notes

- **NOTE #1: X- and Z-axes**

The `X` axis is PARALLEL to the transducer and points from the first (leftmost) element to the last (rightmost) element (`X = 0` at the CENTER of the transducer).

The Z axis is PERPENDICULAR to the transducer and points downward (Z = 0 at the level of the transducer, Z increases as depth increases). See the figure below.

For a **convex** array, the X axis is parallel to the chord and Z = 0 at the level of the chord.

- **NOTE #2: TILT angle**

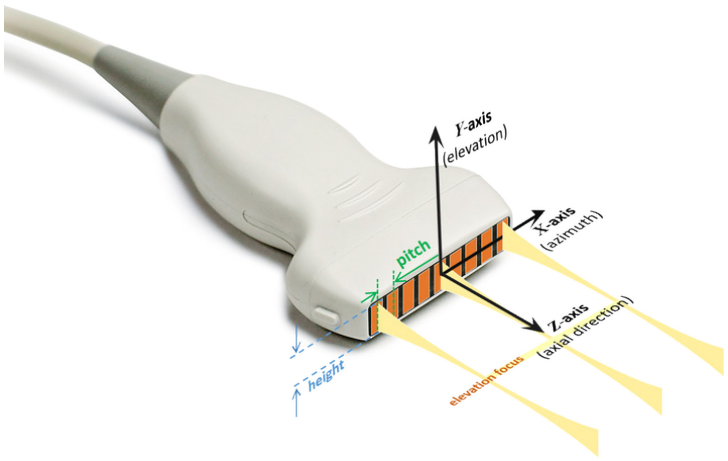
TILT describes the tilt angle in the *trigonometric direction*. See the second figure.

- **NOTE #3: cardiac phased-array**

A cardiac phased-array is also a uniform linear array!

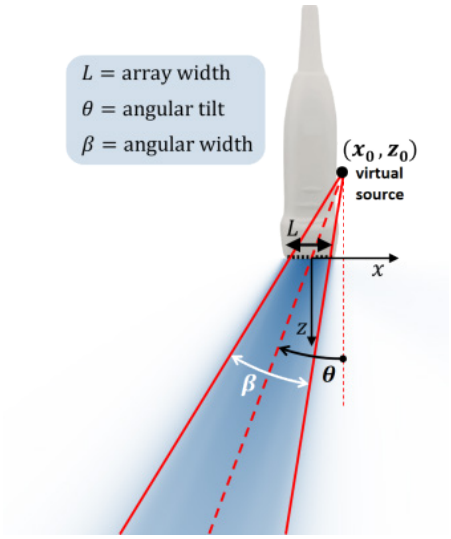
Uniform linear array (ULA)

The ***pitch*** is defined as the center-to-center distance between two adjacent elements. It is constant for a uniform linear array (ULA).



Diverging wave, plane wave, virtual source

The two angles that define a diverging (circular) wave transmit are illustrated in the following figure.



β represents the width angle. θ is the tilt angle.

L is the width of the array aperture, i.e. the center-to-center distance from the first to the last element of the linear array.

If the array has N elements, and if P stands for the pitch, then $L = (N-1)P$.

The coordinates (x_0, z_0) of the virtual source are given by:

$$x_0 = \frac{L}{2}\frac{\sin(2\theta)}{\sin(2\beta)},\ z_0 = \frac{-L}{2}\frac{\cos(\beta)+\cos(2\theta)}{\sin(2\beta)}.$$

A diverging wave becomes a θ-tilted plane wave when $\beta \longrightarrow 0^+$

Example #1: Focused pressure field with a phased-array transducer

This example shows how to generate the transmit delays to obtain a focused pressure field with a phased-array transducer.

Download the properties of a 2.7-MHz 64-element cardiac phased array in a structure `param` by using GETPARAM.

```
param = getparam('P4-2v');
```

Choose a focus location at xf = 2 cm, zf = 5 cm.

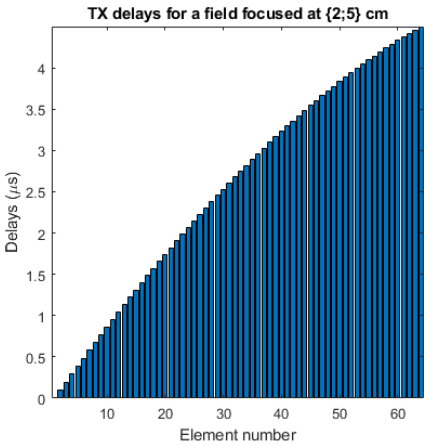
```
xf = 2e-2; zf = 5e-2; % focus position (in m)
```

Obtain the corresponding transmit time delays (in s).

```
txdel = txdelay(xf,zf,param); % in s
```

Display the transmit delays in microseconds.

```
bar(txdel*1e6)
xlabel('Element number')
ylabel('Delays (\mus)')
title('TX delays for a field focused at \{2;5\} cm')
axis tight square
```



Check the pressure field by using `PFIELD`.

First define an image grid.

```
x = linspace(-4e-2,4e-2,200); % in m
z = linspace(0,10e-2,200); % in m
[x,z] = meshgrid(x,z);
```

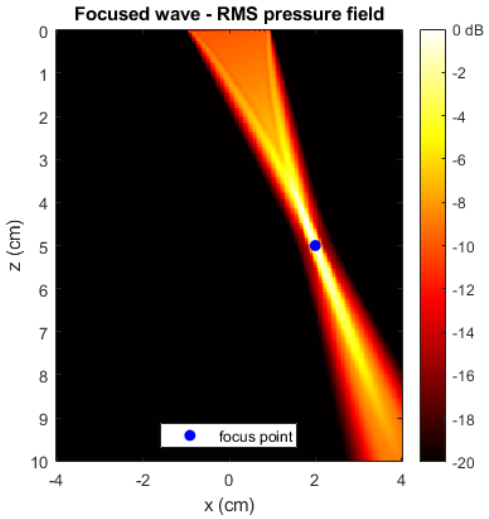
The function `PFIELD` yields the root-mean-square (RMS) pressure field.

```
P = pfield(x,z,txdel,param);
```

Display the acoustic pressure field.

```
imagesc(x(1,:)*1e2,z(:,1)*1e2,20*log10(P/max(P,[],'all')))
caxis([-20 0]) % dynamic range = [-20,0] dB
c = colorbar;
c.YTickLabel{end} = '0 dB';
colormap hot
axis equal tight
xlabel('x (cm)'), ylabel('z (cm)')
title('Focused wave - RMS pressure field')

hold on
plot(xf*1e2,zf*1e2,'bo','MarkerFaceColor','b')
```

```
legend('focus point','Location','South')
hold off
```



Example #2: Diverging wave with a phased-array transducer

This example shows how to generate the transmit delays to obtain a diverging wave with a phased-array transducer.

Download the properties of a 2.7-MHz 64-element cardiac phased array in a structure `param` by using GETPARAM.

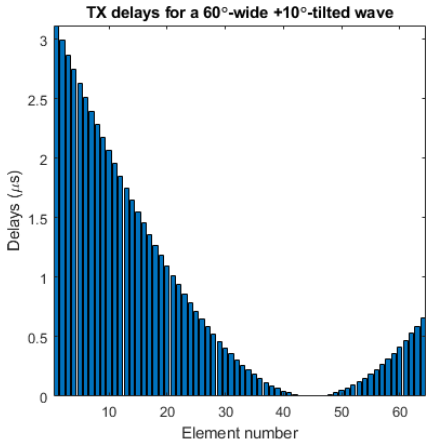
```
param = getparam('P4-2v');
```

Calculate the transmit delays to generate a 60-degrees wide circular wave steered at +10 degrees.

```
width = 60/180*pi; % width angle in rad
tilt = 10/180*pi; % tilt angle in rad
txdel = txdelay(param,tilt,width); % in s
```

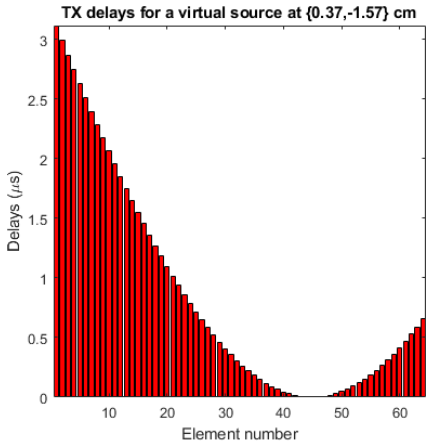
Display the delays.

```
bar(txdel*1e6)
xlabel('Element number')
ylabel('Delays (\mus)')
title('TX delays for a 60{\circ}-wide +10{\circ}-tilted wave')
axis tight square
```



Alternatively, the transmit delays can also be calculated by defining a *virtual source*.

```
p = param.pitch; % pitch (in m)
L = (param.Nelements-1)*p; % array width (in m)
x0 = L/2*sin(2*tilt)/sin(2*width); % in m
z0 = -L/2*(cos(width)+cos(2*tilt))/sin(2*width); % in m
txdel = txdelay(x0,z0,param); % in s

bar(txdel*1e6,'r')
xlabel('Element number')
ylabel('Delays (\mus)')
title('TX delays for a virtual source at \{0.37,-1.57\} cm')
axis tight square
```



Display the resulting pressure field by using `PFIELD`.

First define the image grid.

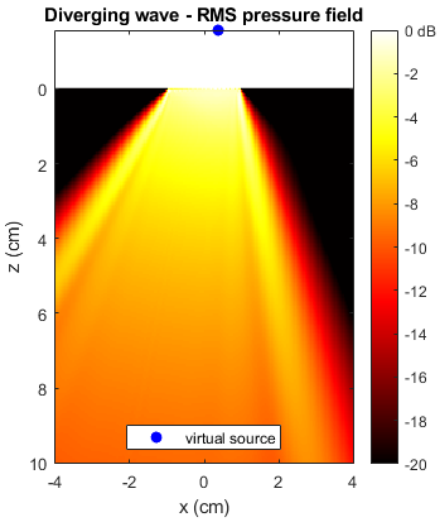
```
x = linspace(-4e-2,4e-2,200); % in m
z = linspace(0,10e-2,200); % in m
[x,z] = meshgrid(x,z);
```

The function PFIELD yields the root-mean-square (RMS) pressure field.

```
P = pfield(x,z,txdel,param);
```

Display the acoustic pressure field.

```
imagesc(x(1,:)*1e2,z(:,1)*1e2,20*log10(P/max(P,[],'all')))
caxis([-20 0]) % dynamic range = [-20,0] dB
c = colorbar;
c.YTickLabel{end} = '0 dB';
colormap hot
axis equal tight
xlabel('x (cm)'), ylabel('z (cm)')
title('Diverging wave - RMS pressure field')

hold on
plot(x0*1e2,z0*1e2,'bo','MarkerFaceColor','b')
legend('virtual source','Location','South')
hold off
```



Example #3: Plane wave with a linear array

This example shows how to generate the transmit delays to obtain a plane wave with a linear transducer.

Download the properties of a 7.6-MHz 128-element uniform linear array in a structure param by using GETPARAM.

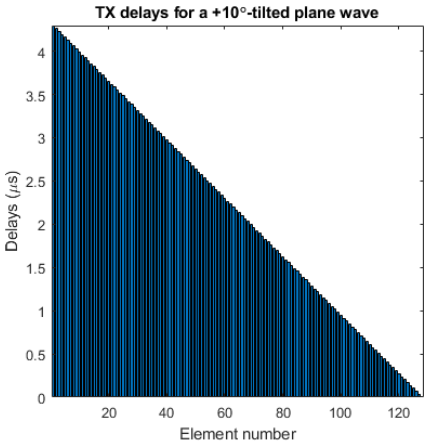
```
param = getparam('L11-5v');
```

Calculate the transmit delays for a plane wave steered at +10 degrees

```
tilt = 10/180*pi; % tilt angle in rad
txdel = txdelay(param,tilt); % in s
```

Display the transmit delays.

```
bar(txdel*1e6)
xlabel('Element number')
ylabel('Delays (\mus)')
title('TX delays for a +10{\circ}-tilted plane wave')
axis tight square
```



Use `PFIELD` to simulate the pressure field.

First define the image grid.

```
x = linspace(-5e-2,5e-2,200); % in m
z = linspace(0,5e-2,150); % in m
[x,z] = meshgrid(x,z);
y = zeros(size(x));
```

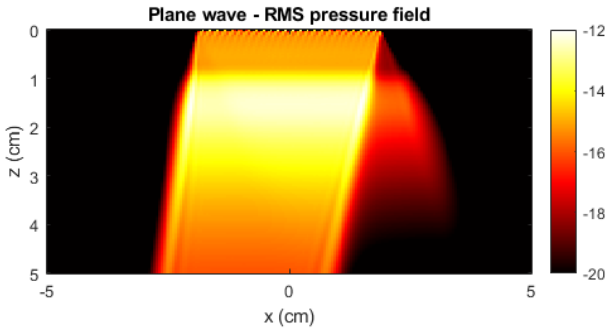
The function `PFIELD` yields the root-mean-square (RMS) pressure field. To make the simulations faster, the number of subelements is set to 1 in the structure `options`. See `pfield` documentation for more details. This will have (small) effects in the very near field.

```
options.ElementSplitting = 1;
P = pfield(x,y,z,txdel,param,options);
```

Display the RMS acoustic pressure field.

```
imagesc(x(1,:)*1e2,z(:,1)*1e2,20*log10(P/max(P,[],'all')))
caxis([-20 0]) % dynamic range = [-20,0] dB
c = colorbar;
```

```
c.YTickLabel{end} = '0 dB';
colormap hot
axis equal tight
xlabel('x (cm)'), ylabel('z (cm)')
title('Plane wave - RMS pressure field')
```



Example #4: Multi-line transmit (MLT) with a phased-array

This example shows how to generate a MLT transmit sequence with a phased-array transducer. In this example, a 3-MLT sequence is designed, i.e. three focused waves are transmitted simultaneously.

Download the properties of a 2.7-MHz 64-element cardiac phased array in a structure `param` with GET-PARAM.

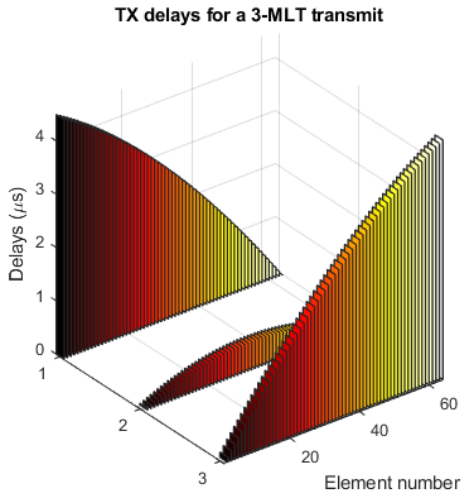
```
param = getparam('P4-2v');
```

Define the TX delays for a 3-MLT transmit sequence.

```
x0 = 2e-2; z0 = 5e-2; % in m
xf = [-x0 0 x0]; zf = [z0 sqrt(x0^2+z0^2) z0]; % focus points (in m)
txdel = txdelay(xf,zf,param); % in s
```

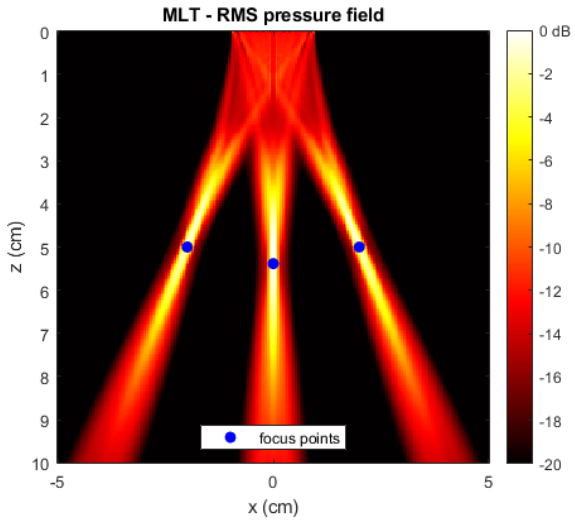
Display the transmit delays.

```
bar3(txdel*1e6,.1)
xlabel('Element number')
zlabel('Delays (\mus)')
title('TX delays for a 3-MLT transmit')
axis tight square
```



Display the pressure field by using `PFIELD`.

```
x = linspace(-5e-2,5e-2,200); % in m
z = linspace(0,10e-2,200); % in m
[x,z] = meshgrid(x,z); % image grid
y = zeros(size(x));

P = pfield(x,y,z,txdel,param); % pressure field

imagesc(x(1,:)*1e2,z(:,1)*1e2,20*log10(P/max(P,[],'all')))
caxis([-20 0]) % dynamic range = [-20,0] dB
c = colorbar;
c.YTickLabel{end} = '0 dB';
colormap hot
axis equal tight
xlabel('x (cm)'), ylabel('z (cm)')
title('MLT - RMS pressure field')

hold on
plot(xf*1e2,zf*1e2,'bo','MarkerFaceColor','b')
legend('focus points','Location','South')
hold off
```



See also

[das](), [pfield](), [simus]()

Date modified

2020/06

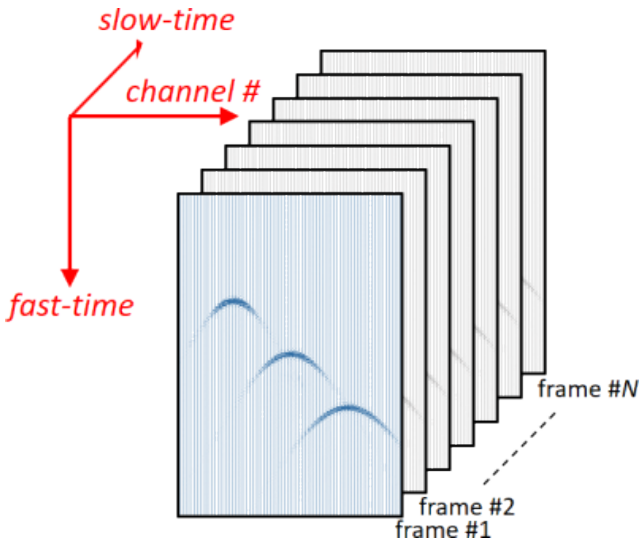
WFILT Wall (clutter) filter

WFILT high-pass (wall) filters the RF or I/Q signals.

Syntax

`fSIG = WFILT(SIG,METHOD,n)` high-pass (wall) filters the RF or I/Q signals stored in the 3-D array `SIG` for Doppler imaging.

The first dimension of `SIG` (i.e. each column) corresponds to a single RF or I/Q signal over (fast-) time, with the first column corresponding to the first transducer element. The third dimension corresponds to the slow-time axis.



Three methods are available .

`METHOD` can be one of the following (case insensitive):

- `'poly'`

Least-squares (*n*th degree) polynomial regression - Orthogonal Legendre polynomials are used. The fitting polynomial is removed from the original I/Q or RF data to keep the high-frequency components. n, with $n \geq 0$, represents the degree of the polynomials. The (slow-time) mean values are removed if $n = 0$ (the polynomials are reduced to constants).

- `'dct'`

Truncated discrete cosine transform - Discrete cosine transforms (DCT) and inverse DCT are performed along the slow-time dimension. The signals are filtered by withdrawing the first n (≥ 1) components, i.e. those corresponding to the n lowest frequencies (with respect to slow-time).

- `'svd'`

Truncated singular value decomposition - An SVD is carried out after a column arrangement of the slow-time dimension. The signals are filtered by withdrawing the top n singular vectors, i.e. those corresponding to the n greatest singular values.

See also

iq2doppler, rf2iq

Reference

The polynomial regression filter was used in the following paper. The polynomial degree was determined automatically by minimizing the Akaike Information Criterion (AIC). Another paper (in preparation) will follow regarding the AIC criterion for truncated DCT and SVD filters. The AIC criterion will be included in a future version, once validated for truncated DCT and SVD.

- Posada D, Porée J, Pellissier A, Chayer B, Tournoux F, Cloutier G, Garcia D. **Staggered multiple-PRF ultrafast color Doppler**. *IEEE Trans Med Imaging*, 2016;35:1510-1521. **(PDF)**

Last update

2020/06