User Manual for the DREAM Toolbox
Version 2.1.3

An ultrasound simulation software for use with MATLAB and GNU OCTAVE

Fredrik Lingvall*

November 13, 2009

*E-mail:Fredrik.Lingvall@ifi.uio.no

Contents

1 Introduction **5**

2 Copyright **5**

2.1 Disclaimer 5

3 System Requirements **5**

4 Installation **6**

4.1 Binary Installation (Matlab only) 6

4.2 Windows Specific Installation Notes 6

4.3 MacOS X Specific Installation Notes 7

4.4 Octave Specific Installation Notes 7

4.5 Installation from Source 7

4.5.1 Build DREAM for Linux/Unix 7

4.5.2 Build the DREAM mex-files for MacOS X (Intel Macs) 8

4.5.3 Build the DREAM Matlab 32-bit `mex`-files for Windows 9

4.5.4 Build the DREAM Octave 32-bit `oct`-files for Windows using the MinGW Compiler 10

4.5.5 Build the DREAM 32-bit Octave `oct`-files for Windows using the MSVC compiler (depreciated) 11

4.5.6 Build the DREAM matlab `mex`-files for 64-bit Windows 11

4.5.7 Build the DREAM Octave `oct`-files for 64-bit Windows 12

4.5.8 Compile with FFTW support for the Attenuation Code 12

5 An Introduction to The Impulse Response Method **12**

5.1 The Baffled Piston Model and the Rayleigh integral 13

5.2 Discrete-time Spatial Impulse Responses 15

5.3 The Discrete Representation (DR) Computational Concept 16

5.4 Lossy Media 17

6 A Quick Start to DREAM Simulations **18**

7 Transducer Function Reference **19**

7.1 Input Parameters Common to all Transducer Functions 19

7.1.1 Observation Point(s) Parameter 19

7.1.2 Sampling Parameters 20

7.1.3 The Delay Parameter 20

7.1.4 Material Parameters 20

7.1.5 Focusing parameters 21

7.1.6 Error Handling 21

7.2 Output Parameters Common to all Transducer Functions 21

7.2.1 The SIR Output Argument 21

7.2.2 The Error Output Argument 22
7.3 Single Element Transducers 22
7.3.1 Line (strip) Transducer 22
7.3.2 Rectangular Transducer 22
7.3.3 Rectangular Focused Transducer 22
7.3.4 Circular Transducer 23
7.3.5 Focused Circular Transducer 23
7.3.6 Spherical Concave Transducer 23
7.3.7 Spherical Convex Transducer 23
7.3.8 Cylindrical Concave Transducer 24
7.3.9 Cylindrical Convex Transducer 24
7.4 Input Parameters Common to the Array Functions 24
7.4.1 The Array Grid Matrix 24
7.4.2 Array Focusing 24
7.4.3 Beam Steering Parameters 25
7.4.4 Apodization Parameters 25
7.5 Array Transducers 26
7.5.1 Array with Rectangular Elements 26
7.5.2 Array with Circular Elements 26
7.5.3 Array with Cylindrical Concave Elements 26
7.5.4 Array with Cylindrical Convex Elements 27
7.5.5 Annular Array 27

8 Parallel Processing Support **27**

9 Analytic Transducer Functions **29**

10 Misc Functions **29**
10.1 Apodization Windows 29
10.2 Attenuation Response 29
10.3 One Dimensional Matrix Convolution Functions 30
10.3.1 Using Pre-computed FFTW Plans 30
10.3.2 Using the In-place Mode 30
10.3.3 Computing Array Responses for Arbitrary Input Signals 31
10.4 Parallel Matrix Copy 32
10.5 The Speed of Sound in Water 32

11 Signal Processing Examples **33**
11.1 Double-path Modeling 33
11.2 Synthetic Aperture Imaging — The Synthetic Aperture Focusing Technique 33
11.3 Delay-and-sum Imaging 34

11.4 Model Based Ultrasonic Array Imaging 35
11.4.1 The Matched Filter 36
11.4.2 The Optimal Linear Estimator 36

12 The Graphical User Interface (Matlab only) 37

13 Known Issues 37

Bibliography 39

A Building the Pthreads Library for 32 and 64 bit Windows 40

B Building the FFTW Library for 32 and 64 bit Windows 40

1 Introduction

THE DREAM (Discrete REpresentation Array Modeling) toolbox is an open source software, released under the GNU General Public License (GPL), for both MATLAB and Octave for simulating acoustic fields radiated from common ultrasonic transducer types and arbitrarily complicated ultrasonic transducers arrays. The DREAM toolbox enables analysis of beam steering, beam focusing, and apodization for wide band (pulse) excitation both in near and far fields. The toolbox is also provided with a user friendly graphical user interface (GUI).

The toolbox consists of a set of routines for computing (discrete) spatial impulse response (SIRs) for various single-element transducer geometries as well as multi-element transducer arrays. Based on linear systems theory, these SIR functions can then be convolved with the transducer's electrical impulse response to obtain the acoustic field at an observation point. Using the DREAM toolbox one can simulate ultrasonic measurement systems for many configurations including phased arrays and measurements performed in lossy media.

The DREAM toolbox uses a numerical procedure based on based on the discrete representation (DR) computational concept [1, 2] which is a method based on the general approach of the spatial impulse responses [3, 4].

2 Copyright

THE DREAM toolbox is an open source software and the source code for the toolbox is freely redistributable under the terms of the GNU General Public License (GPL) as published by the Free Software Foundation (http://www.gnu.org). See also the file `COPYING` which is distributed with the DREAM Toolbox.

The DREAM Toolbox can be downloaded at: http://www.signal.uu.se/Toolbox/dream/. At this website you can also find information how to contact the authors and report bugs etc.

2.1 Disclaimer

The DREAM toolbox is distributed in the hope that it will be useful but WITHOUT ANY WARRANTY. More specifically:

> THE PROGRAM IS PROVIDED "AS-IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL ANY OF THE AUTHORS OF THE DREAM TOOLBOX AND/OR THE DEPARTMENT OF ENGINEERING SCIENCES AT UPPSALA UNIVERSITY, SWEDEN, OR THE INSTITUT D'ELECTRONIQUE ET DE MICRU-ELECTRONIQUE DU NORD (IEMN-DOAE-UMR CNRS 9929), ECOLE CENTRALE DE LILLE, FRANCE, BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA, OR PROFITS, WHETHER OR NOT THE AUTHORS OF THE DREAM TOOLBOX HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, AND/OR ON ANY THEORY OF LIABILITY ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

3 System Requirements

1. MATLAB $\geq$ 7.1 or,
2. Octave $\geq$ 2.9.12 ($\geq$ 3.0.0 recommended).[1]

[1] The graphics routines have been in under heavy development in recent versions of Octave. The plotting and image routines in Octave 3.0.0 and above are, therefore, much more compatible with Matlab now then it used to be. The example

3. FFTW $\geq$ 3.0.1 (optional).[2]

4. Pthread.[3]

4 Installation

THE DREAM Toolbox can be installed both using pre-compiled binaries and from source code. Binaries are currently available for Linux (x86/x86_64), Windows (x86), and for Intel Macs (Mac OS X). The binaries are compiled using generic compiler flags and without support for the fftw library (for the transducer functions) and should, therefore, run on most setups.

If you want higher performance then it is recommended that you compile DREAM from source. There is a bash script and m-files included in the source distribution to facilitate building the toolbox and, furthermore, the `Makefile`s used in build process can easily be edited for further fine tuning of the performance.

4.1 Binary Installation (Matlab only)

1. Download the `dream-xxx-2.x.x.tgz` file, or `dream-xxx-2.x.x.zip` file, using your favorite browser (where `xxx` replaced by the hardware architecture (`unix` [32/64], `win`, or `maci`) and the `2.x.x` is replaced by the actual version number of the toolbox).

2. Copy the file to a suitable directory on your disk and uncompress the file:

   Windows: Use `Winzip` or `Winrar`.

   Linux: Type `tar xvzf dream-xxx-2.x.x.tgz` (or `gunzip dream-xxx-2.x.x.zip`).

3. Add the new directory into the MATLABpath. This can be performed by choosing the *Set Path* command from the **File menu** (MATLAB only) or using the `addpath` command. On Linux you can add the line `addpath('/`*your_installation_dir*`/')` to your `startup.m` file.

Note: since the oct-API often changes between releases of Octave the binaries will probably only work for the version of Octave that they were compiled for. For this reason binaries for Octave are no longer available and you need to install the toolbox from source which easily can be done with the Octave `pkg` command (see Section 4.4).

4.2 Windows Specific Installation Notes

If you want to use the parallel (threaded) DREAM functions you need to install the Pthreads-Win32 library. Download the library from: `http://sourceware.org/pthreads-win32/` and put the pthreadGC2.dll file in a suitable directory, such as, `C:\WINDOWS\System32`; this file is now also available on the DREAM web page. If you want to build the Pthreads library for Windows from source then you can find instructions in Appendix A.

The `fftconv_p` and `sum_fftconv_p` use the FFTW3 library (see Section 10.3). A Windows version of this library can be found at: `http://www.fftw.org/install/windows.html` but it is now also available

scripts in the DREAM toolbox now uses these new features in Octave and it is therefore recommended to install a recent version of Octave. However, the transducer functions in DREAM will probably also work with older versions of Octave.

[2]The attenuation code in DREAM makes heavily use of FFTs. If the optimized FFTW is installed then DREAM can be configured to use this lib.

[3]If you want to use the parallel (threaded) DREAM functions in Windows you need to install the Pthreads-Win32 library.

on the DREAM web page. Copy the `libfftw3-3.dll` file to `C:\WINDOWS\System32`. If you want to build the FFTW library for Windows from source then you can find instructions in Appendix B.

To install DREAM for Octave on Windows see Sections 4.5.4 and 4.5.5.

4.3 MacOS X Specific Installation Notes

The `fftconv_p` and `sum_fftconv_p` functions use the FFTW3 library (see Section 10.3). Instructions for installing fftw on MacOS X can be found at:

http://www.fftw.org/install/mac.html

4.4 Octave Specific Installation Notes

As mentioned above, the DREAM Toolbox must be installed from sources for Octave. Recent versions of Octave have a package manager tool (`pkg`) for that purpose. Given that you have the developer tools for your system installed (for Windows see Sections 4.5.4 and 4.5.5) you should be able to install DREAM by,

1. Download the special DREAM source code file for the Octave package manager,
2. type: `pkg install dream-2.x.x.tar.gz` at the Octave command line.[4]

You should now be able to see the DREAM packege by typing:

```
pkg list
```

at the Octave command line.

4.5 Installation from Source

To build the DREAM Toolbox from sources you need to have developer tools installed for your system (compiler, linker, etc.). Start with downloading (and uncompressing) the (full) source `DREAM-2.x.x.tgz` file. This file contains the source code, the documentation (the user manual), and the html code for the web pages. The build process is based on Makefiles both for C/C++ code and for building the (LaTeX) user manual and html documentation. Therefore, to build the documentation you need to have TeX/LaTeX installed and, furthermore, to build the html documentation you also need the tex4ht and highlight tools.[5]

4.5.1 Build DREAM for Linux/Unix

There are three methods that can be used to build the DREAM Toolbox on unix from sources. The first, and simplest, is to use the m-script `build_mexfiles_unix.m`, the second is to use the bash script `build_dream.sh` and the third is to manually edit the build configuration file `Make.inc` and then compile the sources.

Method 1: Using the `build_mexfiles_unix.m` script.

This is the simplest method but there is no optimization for the used architecture and the attenuation code is build without fftw support (see Section 4.5.8). This will also only build the mex-files (not the oct-files).

[4] Note: this will not build the documentation.

[5] The build of the documentation has only been tested on Linux based systems.

1. Install fftw (if it is not already installed).
2. Start MATLAB and select compiler (gcc is recommended) with `mex -setup` (if you have not already done so).
3. Remove the `-ansi` flag from your

   `~/.matlab/R200Xx/mexopts.sh`

   file (the DREAM sources contain C++ style comments and they will not compile with `-ansi`).
4. Run the m-script: `build_mexfiles_unix`
5. Copy the files in the `gui` and `help_m_files` directories to your DREAM install directory.
6. Add your DREAM install directory to the MATLAB path. That is, add `addpath('/`*your_installation_dir*`/')`to your `matlab/startup.m` file.

Method 2: Using the `build_dream.sh` bash script.

The bash script `build_dream.sh` will probe the machine for cpu, architecture (32 or 64 bits) and then use a pre-defined set of compiler flags for the machine (currently only x86 is supported). This script will both build the mex-/oct-interfaces and the documentation for DREAM. It is assumed that MATLAB is installed in `/usr/local/matlab`. If you have installed MATLAB somewhere else then you can create a symbolic link to the `/usr/local/matlab` dir. Usage:

```
./build_dream.sh
```

Then add your DREAM install directory to the MATLAB/Octave path(s). That is, add `addpath('/`*your_installation_dir*`/')` to your `matlab/startup.m` file and/or `.octaverc` file.

Method 3: Manual configuration.

1. Copy `Make.default` to `Make.inc`, and open the `Make.inc` file with a text-editor.
2. Change the paths for Matlab and/or Octave in `Make.inc` to fit your installation.
3. Change `INSTALL_DIR`, `MEX_EXT` (if you use MATLAB), `CFLAGS`, and `OCTVER` (if you use Octave) in `Make.inc` to fit your architecture.
4. If you don't have both Matlab and Octave then open the `Makefile` and remove the corresponding software on the line "`all:  matlab octave doc`" that you don't use.
5. Type `make`
6. Add your DREAM install dir to the MATLAB/Octave path(s). That is, add `addpath('/`*your_installation_dir*`/')` to your `matlab/startup.m` file and/or `.octaverc` file.

4.5.2 Build the DREAM mex-files for MacOS X (Intel Macs)

There is currently only one method to build the DREAM mex-files for MacOS X and that is to use the `build_mexfiles_unix.m` script (the `Makefiles` do not currently work under Mac OS X).

1. Install fftw (if it is not already installed).
2. Start MATLAB and select compiler (gcc is recommended) with `mex -setup` (if you have not already done so).

3. Remove the `-ansi` flag from your

   `~/.matlab/R200Xx/mexopts.sh`

   file (the DREAM sources contain C++ style comments which will not compile with `-ansi`).

4. Run the m-script: `build_mexfiles_unix`

5. Copy the files in the `gui` and `help_m_files` directories to your DREAM install directory.

6. Add your DREAM install directory to your MATLAB path. That is, add `addpath('/`*your_installation_dir* to your `matlab/startup.m` file (or use the menu in the MATLAB GUI).

4.5.3 Build the DREAM Matlab 32-bit mex-files for Windows

Method 1: Using the MinGW/Gnumex Tools.

1. Download the DREAM Toolbox full source package and uncompress it.

2. Install MinGW and the Gnumex tools (see http://www.mingw.org/ and http://gnumex.sourceforge.net/).

3. Gnumex is normally setup by running the `gnumex.m` script, which is located in the main Gnumex directory, from the Matlab promt which generates a `mexopts.bat` file for the Matlab `mex` command. The `windows/mexopts/` directory contains three pre-build `bat`-files (for MinGW 3.4.5 and Gnumex 2.1.0) which is used to build the toolbox and link with the fftw and Pthread-Win32 libs. You need to edit the `mexopts_mingw.bat`, `mexopts_pthread_mingw.bat`, and `mexopts_pthread_fftw_mingw.bat` files for your installation, that is, set paths to your DREAM source directory, MinGW directories, and Gnumex directories, respectively. The lines you need to make changes in are indicated with numbers 1)–7) in the respective `bat`-file.

4. Start MATLAB and run the m-script `build_mexfiles_mingw.m` (in the main DREAM source directory).[6]

5. Copy the files in the `gui` and `help_m_files` directory to your DREAM install directory.

6. Add your DREAM install directory to your MATLAB path. That is, add `addpath('/`*your_installation_dir* `')` to your `matlab/startup.m` file (or use the menu in the MATLAB GUI).

7. Copy the files `pthreadGC2.dll` (for gcc) and `libfftw3-3.dll` to `C:\WINDOWS\System32`.

Method 2: Using the MSVC 2008 Express Edition

1. Download the DREAM Toolbox full source package and uncompress it.

[6] If you are using an older version of Gnumex you may get an error that `libmex.def` is missing (happens for Matlab R2007a and above) then the files `libmx.def`, `libmat.def`, and `libmex.def` are missing in the `MATLAB/extern/include/` directory. They can, however, be created from the corresponding `dll` files in `MATLAB/bin/win32/` directory with the `pexports` tool which can be found at: http://www.emmestech.com/software/pexports-0.43/download_pexports.html. Type `pexports.exe libmx.dll > libmx.def` etc. in a cmd shell and copy the files to the `MATLAB/extern/include/` directory. This is not needed in newer versions of Gnumex which automatically build the `def`-files if they are missing.

2. Start MATLAB and select compiler with `mex -setup` (if you have not already done so). If Matlab cannot find the MSVC 2008 compiler then read this page:
http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=18508. Then, after you have installed the `bat`-files in their corresponding directories, you need to set the env variable `MSSdk` for the Windows SDK to, for example, `C:\Program Files\Microsoft SDKs\Windows\v6.1`. On Windows XP, this is done by first "right-clicking" on *My Computer* and then selecting *Properties→Advanced→Environment Variables→New* and, second, add the `MSSdk` variable and the path to the Windows SDK that you use (you may need to reboot to make the new variable visible). On Windows Vista, go to the start meny, "right-click" on *Computer* and then select *Properties→Advanced system settings→Environment Variables...→New...* and add `MSSdk` and the path to the Windows SDK.
3. Run the m-script: `build_mexfiles_msvc.m`
4. Copy the files in the `gui` and `help_m_files` directories to your DREAM install directory.
5. Add your DREAM install directory to your MATLAB path. That is, add `addpath('/`*your_installation_dir*`')` to your `matlab/startup.m` file (or use the menu in the MATLAB GUI).
6. Copy the files `pthreadGC2.dll` and `libfftw3-3.dll` to `C:\WINDOWS\System32`.

Method 3: Matlab's build-in LCC compiler (depreciated).

1. Download the DREAM Toolbox full source package and uncompress it.
2. Start MATLAB and select compiler with `mex -setup` (if you have not already done so).
3. Run the m-script: `build_mexfiles_windows.m`
4. Copy the files in the `gui` and `help_m_files` directories to your DREAM install directory.
5. Add your DREAM install directory to your MATLAB path. That is, add `addpath('/`*your_installation_dir*`')` to your `matlab/startup.m` file (or use the menu in the MATLAB GUI).

Note the LCC compiler, at least LCC for Matlab R2007b, cannot build the parallel `mex`-functions since the compiler cannot parse the Pthread-Win32 header files. All parallel functions will, therefore, be unavailable if you are using the LCC compiler.

4.5.4 Build the DREAM Octave 32-bit `oct`-files for Windows using the MinGW Compiler

The pkg source can be build also on Windows with resent versions of Octave (tested with Octave 3.2.3 [MinGW 4.4.0]). To do this, using the native Windows Octave from octave-forge, you only need to install Octave since fftw, the MinGW compiler, and the pthread libs are included in the Octave package).

1. Download the DREAM Toolbox `pkg` source package (i.e., the file `dream-2.x.x.tar.gz`).
2. Download the Windows Octave installer.
3. Install Octave in `C:\Octave`; do not install in `C:\Program Files\Octave` since the `mkoctfile` script that is used to compile the C++ code in the DREAM toolbox do work well with white spaces in the path (the default install location should be OK).
4. Build and install the DREAM Toolbox with: `pkg install dream-2.x.x.tar.gz`.

4.5.5 Build the DREAM 32-bit Octave `oct`-files for Windows using the MSVC compiler (depreciated)

This section describes how to build the DREAM oct-files for Windows using the MSVC compiler. But note that, as of spring 2009, Octave binaries build with the MSVC compiler are no longer available at octave-forge due to license issues. It is therefore recomended that the MinGW compiler is used instead as described in Section 4.5.4! To use MSVC one therefore needs an older MSVC build binary of Octave or one has to build Octave (and all its dependencies) using the MSVC compiler and then follow the procedure below:

1. Download the DREAM Toolbox `pkg` source package and uncompress it.
2. Download the MSVC version of Octave (build with the Microsoft Visual C++ 2008 Express Edition compiler).
3. Install Octave in `C:\Octave`; do not install in `C:\Program Files\Octave` since the `mkoctfile` script that is used to compile the C++ code in the DREAM toolbox do work well with white spaces in the path.
4. Install Microsoft Visual C++ 2008 Express Edition
5. Get the `fftw3.h` file (for version 3.2.2) from `http://www.fftw.org`, or from DREAM web page, and copy it to `C:\Octave\include\`; you don't need to install the fftw3 `dll` file since it is already included in the MSVC version of Octave.
6. Get the `phread.h`, `shed.h`, and `semaphore.h` files from `http://sourceware.org/pthreads-win32/` (version 2.8.0), or from the DREAM web page, and copy them to `C:\Octave\include\`
7. Get the `phreadGC2.lib` file from `http://sourceware.org/pthreads-win32/`, or from the DREAM web page, copy it to `C:\Octave\lib\`, and rename it to `phread.lib`; the Unix style Makefile which is used to build the toolbox expects this name of the lib file.
8. Create a new `bat` file and add the lines:

   ```
   @echo off
   call "C:\Program Files\Microsoft Visual Studio 9.0\VC\vcvarsall.bat"
   "C:\Octave\bin\octave-3.0.1.exe"
   ```

   to the file (change the path to the actual version of Octave that you use).
9. Start Octave by running the `bat` file created above.
10. Build and install the DREAM Toolbox with: `pkg -verbose install dream-2.x.x.tar.gz`.
11. Get the `pthreadGC2.dll` file from `http://sourceware.org/pthreads-win32/` (version 2.8.0), or from the DREAM web page, and copy it to `C:\WINDOWS\System32`.

4.5.6 Build the DREAM matlab `mex`-files for 64-bit Windows

If you have an older 64-bit Matlab installation (eg. 2007x) then follow these steps:

1. Install MSVC 2008 Express Edition SP1.
2. Install the Windows SDK; The current version is *Windows SDK for Windows Server 2008 and .NET Framework 3.5*. Remember to select the "X64 Compilers and Tools" when installing the SDK.

3. If you are using a different Windows SDK version than 6.1 then you need to set the correct path in the bat-file `windows/mexopts/mexopts_msvc_64.bat`.

4. Get the 64-bit versions of fftw and Pthread-win32 from http://sourceforge.net/projects/mingw-w64/ and http://www.fftw.org, respectively, or from the DREAM web page, and copy them to `C:\WINDOWS\System32`.[7]

If you have Matlab 2008b, or above,[8] then read and follow the instructions in Mathworks FAQ: http://www.mathworks.fr/support/solutions/en/data/1-6IJJ3L/index.html?solution=1-6IJJ3L. Then run the `build_mexfiles_windows.m` script.

4.5.7 Build the DREAM Octave oct-files for 64-bit Windows

Currently there is no support for building 64-bit `oct`-files since there is no 64-bit version of Octave for Windows available (yet).

4.5.8 Compile with FFTW support for the Attenuation Code

The attenuation code in the DREAM Toolbox uses FFTs extensively. To speed up the FFT computations (by 10% to 50%) one can compile The DREAM Toolbox with FFTW support for the attenuation code. To do this you need to compile all functions with the flag `-DUSE_FFTW` and link with `-lfftw3`. This can be accomplished by changing:

```
ATT_FFTW        =
FFW_LIB         =
```

to

```
ATT_FFTW = -DUSE_FFTW
FFW_LIB = -lfftw3
```

in the file `Make.default` before compiling the toolbox if you are using *Method 3* described in Section 4.5.1 (the `build_dream.sh` uses fftw by default).

5 An Introduction to The Impulse Response Method

THE impulse response method is an approach based on linear systems theory to model acoustic fields from ultrasonic transducers; the method was introduced by Tupholme and Stepanishen in the late 60's, early 70's [3, 4]. The impulse response method is based on linear acoustics and can be used to model acoustic fields and (double-path) responses for both single transducer setups and for array imaging. The idea is to divide the imaging system in two parts: the first one accounts for acoustical wave propagation effects (i.e., the diffraction effects) from the transducer surface to the observation point, and the second one accounts for the electro-acoustical effects. These two parts are then convolved to obtain a model for the total imaging system.

The impulse response method is very flexible since, (*i*) by linearity the response from multi-element transducers (such as array transducers) can be obtained be means of super-position and (*ii*) arbitrary input signals can be treated by simply convolving the electro-acoustical impulse response with the input

[7] The sources for Pthread-win32 and fftw is now also included in the DREAM source package. There are also scripts and patches for genrerating 64-bit versions of these libs using the MinGW-w64 cross compiler tool chain on Linux (see Appendix A and B.

[8] The method described above (for Matlab 2007x) may work for Matlab 2008b, or above too.

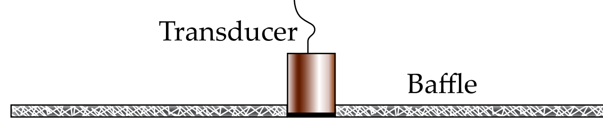


Figure 1: Illustration of a baffled transducer.

signal. In this section we will present a short introduction to the impulse response method and, in particular, discuss how to use the method for discrete-time modeling.

5.1 The Baffled Piston Model and the Rayleigh integral

The impulse response method is based on the assumption that the transducer can be treated as a baffled piston. This assumption implies that we only need to consider the active area of the transducer when modeling the wave propagation. That is, if the source (the transducer) is located in the rigid plane, often referred to as the *rigid baffle* as illustrated in Figure 1, then the baffle (S_{B}) will not contribute to the field. The pressure at an observation point $\mathbf{r}$ is then described by the Rayleigh integral

$$\begin{aligned} p(\mathbf{r},t) =& \rho_0 \frac{\partial}{\partial t} \int_{-\infty}^{\infty} \left(\int_{S_{\text{R}}} v_{\text{n}}(\mathbf{r}_0,t_0) \frac{\delta\left(t-t_0-|\mathbf{r}-\mathbf{r}_0|/c_p\right)}{2\pi|\mathbf{r}-\mathbf{r}_0|} dS_{\text{R}} \right) dt_0, \\ =& \rho_0 \frac{\partial}{\partial t} \int_{-\infty}^{\infty} v_{\text{n}}(t_0) \int_{S_{\text{R}}} \frac{\delta\left(t-t_0-|\mathbf{r}-\mathbf{r}_0|/c_p\right)}{2\pi|\mathbf{r}-\mathbf{r}_0|} dS_{\text{R}} dt_0, \\ =& \rho_0 \frac{\partial}{\partial t} \int_{-\infty}^{\infty} v_{\text{n}}(t_0) h^{\text{f-SIR}}(\mathbf{r},t-t_0) dt_0, \\ =& \rho_0 \frac{\partial}{\partial t} v_{\text{n}}(t) * h^{\text{f-SIR}}(\mathbf{r},t-t_0). \end{aligned} \tag{1}$$

where it is where we for simplicity have assumed that the normal velocity $v_{\text{n}}(\mathbf{r}_0,t) \equiv v_{\text{n}}(t)$ is uniform on the transducer's surface S_{R}. The Rayleigh integral formula (1) simply states that the acoustic field at an observation point is the sum of the contributions from all points of the active area of the transducer. The impulse response $h^{\text{f-SIR}}(\mathbf{r},t)$ in Eq. (1) is usually referred to as the (forward) *spatial impulse response* (SIR).

The normal velocity, $v_{\text{n}}(t)$, depends on both the input signal, $u(t)$, and the electro-acoustical properties of the transducer, which can be described with the (forward) electrical impulse response $h^{\text{ef}}(t)$. Thus, the pressure at $\mathbf{r}$ can be expressed by the convolutions of the input signal and the two (forward) impulse responses according to,

$$p(\mathbf{r},t) = h^{\text{f-SIR}}(\mathbf{r},t) * h^{\text{ef}}(t) * u(t). \tag{2}$$

Double-path (pulse-echo) responses can be treated in a similar way by convolving the forward response (2) with the backward electrical (acousto-electrical) response and the backward SIR for a point source at $\mathbf{r}$.

Analytical solutions to SIRs exist for a few geometries, but one must in general resort to numerical methods. Also, these *time continuous* solutions are normally not practical since all acquired signals (the data) are normally sampled and time discrete models are, therefore, needed; sampling of time continous SIRs is discussed in Section 5.2.

Before we discuss sampled SIRs, and the particular method use by the DREAM Toolbox, let us consider a case where there exist an analytical solution.

Example 1 (The SIR for a Circular Disc)**.** The SIR of a circular disc (see illustration in Figure 2) has an analytical solution [4] which can be divided in two cases: (i) when the observation point is inside

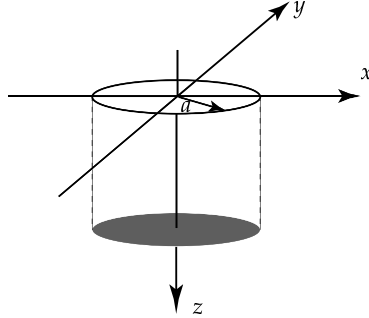


Figure 2: Geometry of a a circular disc source.

the aperture of the disc $\sqrt{x^2+y^2} \leq a$, where a is the transducer radius, and (ii) when the observation point is outside the aperture. The disc is assumed to be located in the x–y plane centered at $x = y = 0$. If we let r denote the distance in the x–y plane from the center axis of the disc to the observation point, $r = \sqrt{x^2+y^2}$, then the circular disc SIR is given by

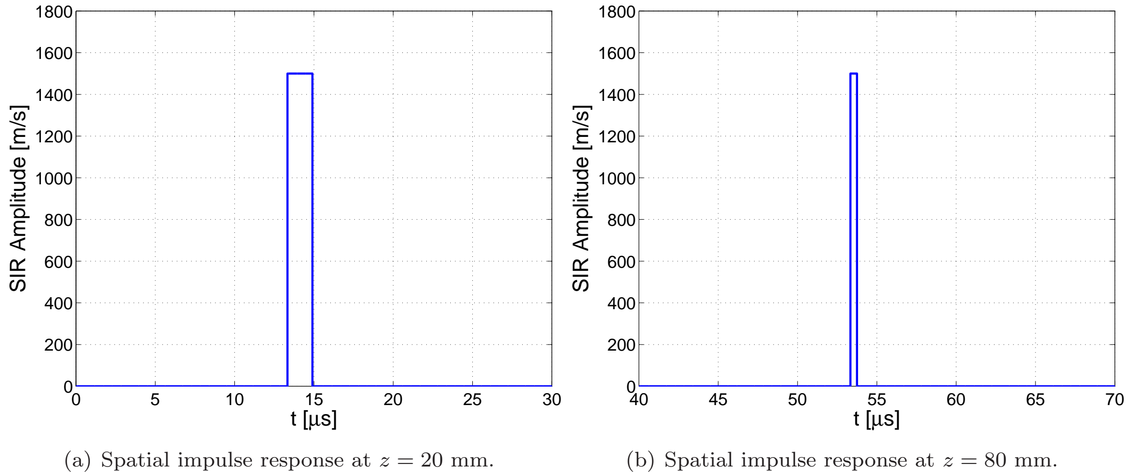
$$
\begin{aligned}
&\text{for } r \leq a \\
&h(\mathbf{r},t) = \begin{cases} 0, & t \leq t_z \\ c_\text{p}, & t_z \leq t \leq t_1 \\ \frac{c_\text{p}}{\pi}\cos^{-1}\left(c_\text{p}^2 \frac{t^2-t_z^2+t_r^2-a/c_\text{p}^2}{2t_r\sqrt{t^2-t_z^2}}\right), & t_1 < t \leq t_2 \\ 0, & t > t_2 \end{cases} \\
&\text{for } r > a \\
&h(\mathbf{r},t) = \begin{cases} 0, & t \leq t_1 \\ \frac{c_\text{p}}{\pi}\cos^{-1}\left(c_\text{p}^2 \frac{t^2-t_z^2+t_r^2-a/c_\text{p}^2}{2t_r\sqrt{t^2-t_z^2}}\right), & t_1 < t \leq t_2 \\ 0, & t > t_2 \end{cases}
\end{aligned}
\tag{3}
$$

where $t_z = z/c_\text{p}$ is the earliest time that the wave reaches the observation point $\mathbf{r}$ when $r \leq a$, $t_r = r/c_\text{p}$, and $t_{1,2} = t_z\sqrt{1+(\frac{a\mp r}{z})^2}$ are the propagation times corresponding to the edges of the disc that are closest and furthermost away from $\mathbf{r}$, respectively.

Noticeable is that the pulse amplitude of the on-axis SIR is constant regardless of the distance to the observation point.[9] The duration of the on-axis SIR is given by $t_1 = a/c_\text{p}$ at $z = 0$. As the distance increases the duration, $t_1 - t_z$, of the SIR becomes shorter, and for large z it approaches to the delta function. The transducer size effects are therefore most pronounced in the near-field. This is illustrated in Figure 3 where the on-axis SIRs at $z = 20$ and $z = 80$ mm, respectively are shown. The duration at $z = 20$ is longer than that of $z = 80$ and if the distance, z, increases then the on-axis SIR will approach to a delta function, cf. Figures 3(a) and (b). □

As mentioned above, there exist no analytical analytical solutions for many transducer geometries, and in such situations numerical methods must be used. The DREAM Toolbox uses a method based on the *discrete representation* (DR) computational concept [1, 2]. The DR method is very flexible in the sense that complex transducer shapes as well as arbitrary focusing methods easily can be modeled. Another benefit of the DR method is that the SIRs are directly computed in a discrete form which is convenient since this directly allows for digital signal processing. The DR method is described in Section 5.3 below, but first we will discuss sampling of spatial impulse responses.

[9] The on-axis SIR has duration $t_1 - t_z$ with the constant amplitude c_p in the time interval $t_z \leq t \leq t_1$.



(a) Spatial impulse response at $z = 20$ mm.

(b) Spatial impulse response at $z = 80$ mm.

Figure 3: On-axis spatial impulse responses for a 10 mm disc where the sound speed, c_p, was 1500 m/s.

5.2 Discrete-time Spatial Impulse Responses

Before we introduce the discrete representation (DR) method, which the DREAM toolbox transducer functions are based on, let us discuss sampling of spatial impulse responses. This is of interest since our ultimate goal normally is to model the total *sampled* imaging system which includes both acoustic propagation effects (the SIRs) as well as input signals and electro-acoustical effects.

To obtain a discrete model we need a discrete representation of the SIRs. That is, the analytical expressions for the SIRs discussed in Section 5.1 must be converted to a discrete form in order to be useful for digital signal processing; a proper discrete representation of the SIRs is necessary so that when the sampled SIR is convolved with the normal velocity the resulting waveform can faithfully represent the sampled measured waveform.

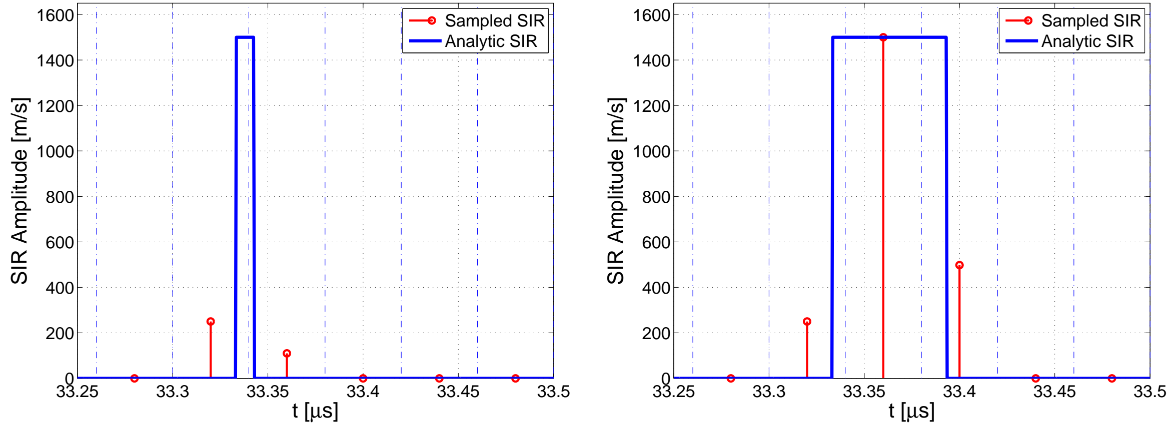
The analytical SIRs have an infinite bandwidth due to the abrupt amplitude changes that, for example, could be seen in the line and disc solutions above. In some situations the duration of a SIR may even be shorter than the sampling interval, T_s, and it is therefore not sufficient to simply sample the analytical SIRs by simply taking the amplitude at the sampling instants since the SIR may actually be zero those time instants. The SIRs are however convolved with a band-limited normal velocity signal, hence the resulting pressure waveform must also be band-limited, cf. (2). Consequently, we only need to sample the SIR in such way that the band-limited received A-scans are properly modeled.

In a sampled system the impulse responses are given at discrete time instants, t_k (given by the sampling period T_s) and to faithfully represent the SIRs we need to collect all contributions from the continuous time SIRs in the corresponding sampling interval $[t_k - T_\mathrm{s}/2, t_k + T_\mathrm{s}/2]$. A discrete version of the a time continuous SIR is then obtained by summing all contributions from the SIR in the actual sampling interval. That is, the sampled SIR is defined as

$$h(\mathbf{r}, t_k) \triangleq \frac{1}{T_\mathrm{s}} \int_{t_k - T_\mathrm{s}/2}^{t_k + T_\mathrm{s}/2} h(\mathbf{r}, t) dt. \tag{4}$$

The division by T_s retains the same unit (m/s) of the sampled SIR as the continuous one. The amplitude of the sampled SIR, at time t_k, is then the mean value of the continuous SIR in the corresponding sampling interval, $[t_k - T_\mathrm{s}/2, t_k + T_\mathrm{s}/2]$. Also, as seen from the analytical solutions above, the SIRs always have a finite length as the transducer has a finite size. The sampled SIRs are therefore naturally represented by finite impulse response filters (FIRs).

The effect of the sampling scheme (4) is illustrated in Figure 4 for two discs with radii 1.2 and 3 mm, respectively, where the sampling interval, T_s, was 0.04μs. In Figure 4(a) the analytic SIR is shorter than



(a) Continuous and sampled spatial impulse responses of a circular disc with radius $r = 1.2$ mm.

(b) Continuous and sampled spatial impulse responses of a circular disc with radius $r = 3$ mm.

Figure 4: Illustration of sampling spatial impulse responses. The continuous and sampled on-axis SIRs for two discs with radii 1.2 and 3 mm, respectively are shown where the sampling interval, T_s, was 0.04 μs (c_p= 1500 [m/s]).

the sampling interval, T_s. The max amplitude of the discrete SIR is therefore lower than then the max amplitude of the continuous SIR. If the duration of the analytic SIR is longer than the sampling interval, as for the 3 mm disc shown in Figure 4(b), then the max amplitudes of the on-axis sampled and analytic disc SIRs will be the same.

5.3 The Discrete Representation (DR) Computational Concept

As mentioned in previous in Section 5.1 the analytical spatial impulse responses are only available for a few simple transducer geometries. Therefore, for a transducer with an arbitrary geometry a numerical method must be used. The numerical method used in this toolbox is based on the *discrete representation* (DR) method, which is based on a discretization of the Rayleigh integral formula (1). In the DR method, the radiating surface is divided into a set of small surface elements (see illustration in Figure 5), and the surface integral in the Rayleigh formula is replaced by a summation. The DR method facilitates computation of SIRs for non-uniform excitation, apodization of the aperture, and arbitrary focusing laws since each surface element can be assigned a different normal velocity, apdodization or time-delay. The DR computational concept can therefore be used for computing SIRs for an arbitrary transducer shape or array layout [1, 2].

A discrete SIR, computed using the DR method, can be found by first dividing the total transducer surface into a set of J surface elements $\{\Delta S_0, \Delta S_1, \ldots, \Delta S_{J-1}\}$. Second, let w_j denote an aperture weight, and $R_j = |\mathbf{r} - \mathbf{r}_j|$ the distance from the jth surface element to the observation point. The discrete SIR can now be approximated by

$$\begin{aligned} h(\mathbf{r}, t_k) &= \frac{1}{2\pi} \sum_{j=0}^{J-1} \frac{w_j \delta(t_k - R_j/c_p - d_j)}{R_j} \Delta S_j \\ &= \sum_{j=0}^{J-1} a_j \delta(t_k - R_j/c_p - d_j), \end{aligned} \tag{5}$$

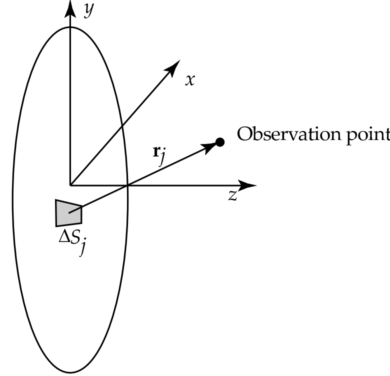


Figure 5: Geometry and notations for the discrete representation method.

where d_j is a user defined focusing delay and $t_k = kT_\mathrm{s}$, for $k = 0, 1, \ldots K-1$. The scaling factor

$$a_j = \frac{w_j \Delta S_j}{2\pi R_j} \tag{6}$$

in (5) represents the amplitude of the impulse response for an elementary surface at $\mathbf{r}_j$ excited by a Dirac pulse. Hence, the total response, at time t_k, is a sum of contributions from those elementary surface elements, ΔS_j, whose response arrive in the time interval $[t_k - T_\mathrm{s}/2, t_k + T_\mathrm{s}/2]$.

The accuracy of the method depends on the size of the discretization surfaces ΔS_j. It should, however, be noted that high frequency numerical noise due to the surface discretization is in practice not critical since the transducer's electrical impulse response has a bandwidth in the low frequency range (for a further discussion see [2]). Also, these errors are small if the elementary surfaces, ΔS_j, are small. The DR-method is very flexible in the sense that beam steering, focusing, apodization, and non-uniform surface velocity can easily be included in the simulation.

5.4 Lossy Media

The computational procedure for an attenuation free medium implies a Dirac-type Green's function [1]. The discrete approach in the DREAM toolbox above is, however, also applicable to the problems characterized by an arbitrarily shaped causal Green's function. In such a case a Dirac function is simply replaced by a sampled version of this function. This characteristic extends the field of applications and allows, for example, the computations for lossy media. For such a case, the free space Green's function $\frac{\delta(t-|\mathbf{r}_\mathrm{s}-\mathbf{r}_\mathrm{o}|/c)}{4\pi|\mathbf{r}_\mathrm{s}-\mathbf{r}_\mathrm{o}|}$, where $\mathbf{r}_\mathrm{s}$is a point in the transducer surface and $\mathbf{r}_\mathrm{o}$is the observation point, should be replaced by its causal counterpart $g_\alpha(t, |\mathbf{r}_\mathrm{s} - \mathbf{r}_\mathrm{o}|)$ related to the medium with absorption. The solution for lossy media used in the DREAM toolbox for g_α has the following frequency-domain form [1, 5]:

$$G_\alpha(j\omega, |\mathbf{r}_\mathrm{s} - \mathbf{r}_\mathrm{o}|) = \frac{1}{4\pi|\mathbf{r}_\mathrm{s} - \mathbf{r}_\mathrm{o}|} e^{j(k_1|\mathbf{r}_\mathrm{s}-\mathbf{r}_\mathrm{o}|-\omega t)} \tag{7}$$

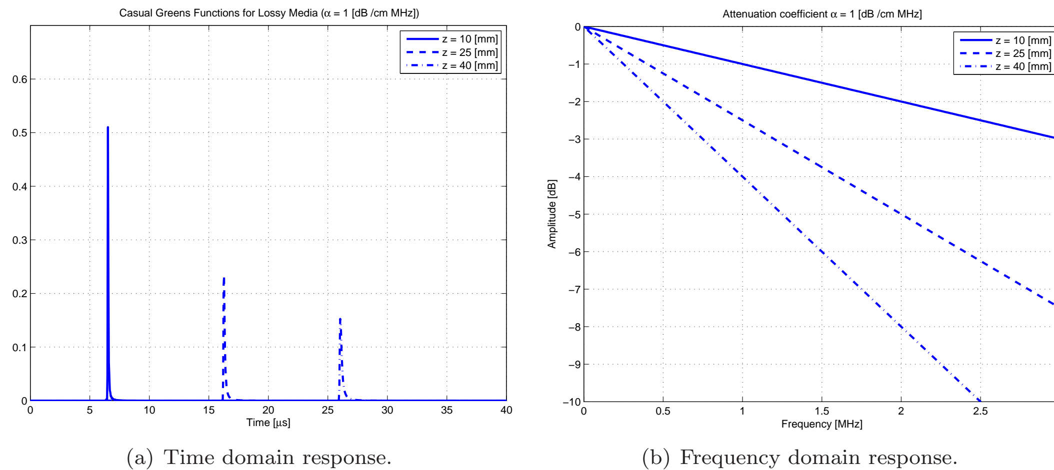
where

$$k_1 = \frac{\omega}{c}\left[1 + \frac{c\alpha_0}{\pi^2}\ln\left(\frac{1}{\alpha_1\omega}\right)\right] + \alpha_0 f, \tag{8}$$

$\alpha_1 = \pi/0.95$, and $\alpha_0(\alpha) = \frac{\alpha}{8.686\times10^4}$. The time-domain transfer function is then obtained by means of the inverse Fourier transform

$$g_\alpha(t, |\mathbf{r}_\mathrm{s} - \mathbf{r}_\mathrm{o}|) = \mathcal{F}^{-1}\{G_\alpha(j\omega, |\mathbf{r}_\mathrm{s} - \mathbf{r}_\mathrm{o}|)\}. \tag{9}$$

In the DREAM toolbox this computation is performed by a discrete Fourier transform for each surface element `dx`$\times$`dy`. An illustration of the effects due to lossy media for an attenuation of coefficient, α of 1 [db/cm MHz] is shown in Figure 6 for 10 mm, 25 mm, and 40 mm respectively.



(a) Time domain response.

(b) Frequency domain response.

Figure 6: Illustration of the attenuation response at three different depths. The attenuation coefficient, α, was 1 [db/cm MHz] and the attenuation response at 10 mm is the solid line, 25 mm is the dashed line, and 40 mm is dash-dotted line, respectively.

6 A Quick Start to DREAM Simulations

IN his section a quick start on how to perform simulations with DREAM is presented. Here only a simple example is shown just to illustrate what is needed to simulate an ultrasonic measurement system. We will use a circular transducers for this example but more advanced examples can be found on the DREAM web page http://www.signal.uu.se/Toolbox/dream/ on the *Examples* page. Also more details of the various functions in the DREAM Toolbox can be found in Sections 7, 8, and 10, respectively.

Let us study a the pressure response for a circular transducer using water as the propagation medium (with a sound speed $c_p = 1500$ [m/s]). We start by setting the sampling frequency and defining the points of interest, the so-called observation points. The observation points are located on a line at $z = 10$ mm, from 1–50 mm with 1 mm between them:

```
Fs = 10;   % Sampling freq. in MHz.
Ts = 1/Fs;

%
% Observation point(s).
%

% Depth
z = 10; % [mm]

%  Points along x-axis.
d  = 1;                          % [mm]
xo = (0:d:50);                   % 0-50 mm.
yo = zeros(length(xo),1);
zo = z*ones(length(xo),1);
Ro = [xo(:) yo(:) zo(:)];
```

Then we need to define the discretization parameters for both the transducer surface and the temporal sampling:

```
% Descretization parameters.
dx = 0.03;                              % [mm].
dy = 0.03;                              % [mm]
dt = Ts;                                % [us].
nt = 400;                               % Length of spatial impulse response vector.
s_par = [dx dy dt nt];
```

We must also define the sound speed of the medium, normal velocity, and attenuation. Here we choose an attenuation free medium ($\alpha = 0$) and a unit normal velocity:

```
% Material parameters.
v     = 1.0;                            % Normal velocity.
cp    = 1500;                           % Sound speed.
alfa  = 0;                              % Absorbtion [dB/(cm MHz)].
m_par = [v cp alfa];
```

The size of the transducer must also be defined, here we use a circular transducer with a 5 mm radius:

```
% Geometrical parameters.
r = 5;                                  % Radius [mm].
geom_par = [r];
```

Finally we set the start point of the SIR and call the DREAM function `dreamcirc` to compute the discrete SIRs:

```
% Delay.
t_z = z*1e3/cp;
%delay = 0;                             % Start at 0 [us].
delay = t_z;                            % Start at t_z [us].

H = dreamcirc(Ro,geom_par,s_par,delay,m_par,'stop');
```

7 Transducer Function Reference

THE transducer functions are implemented as MATLAB *mex*-functions and Octave *oct*-functions; a *mex/oct*-function is pre-compiled code that is dynamically linked to MATLAB/Octave at run time. This greatly increases the computation speed compared to using ordinary m-files since the DR-concept (see [2]) uses `for` and `while` loops extensively. An overview of the transducer functions can be found in Table 1.

By convention, a transducer function name ending with "`_f`" is a focused transducer (or has focused transducer elements) and a transducer function name ending with "`_d`" is defocused (convex) transducer.

The transducer functions can be divided in two groups, *single transducer functions* and *array functions*. The names of both groups starts with "`dream`" and the array functions are distinguished from single transducer functions by "`_arr_`". Note that arbitrary arrays (with mixed forms of transducer elements) can be modeled using the single transducer functions and then adding their corresponding responses.

7.1 Input Parameters Common to all Transducer Functions

7.1.1 Observation Point(s) Parameter

The transducer functions can compute SIRs for multiple observation points. The observation points are given by a $N \times 3$ matrix `Ro` were the first column contains the x-coordinates, the second the y-coordinates, and the third the z-coordinates, respectively (N is the number of observation points).

The single element transducers are, by convention, all centered at $\mathbf{r} = (0, 0, 0)$ and the SIRs are computed relative to this center point. The array functions uses a grid matrix `G` to define the positions

Transducer type	DREAM function
Strip transducer	dreamline
Rectangular transducer	dreamrect
Focused rectangular transducer	dreamrect_f
Circular transducer	dreamcirc
Focused circular transducer	dreamcirc_f
Spherical concave transducer (focused)	dreamsphere_f
Spherical convex transducer (defocused)	dreamsphere_d
Cylindrical concave transducer (focused)	dreamcylind_f
Cylindrical convex transducer (defocused)	dreamcylind_d
Array with rectangular elements	dream_arr_rect
Array with circular elements	dream_arr_circ
Array with cylindrical concave elements (focused)	dream_arr_cylind_f
Array with cylindrical convex elements (defocused)	dream_arr_cylind_d
Annular array	dream_arr_annu

Table 1: Overview of the DREAM toolbox transducer functions.

of the array elements (see Section 7.4). The SIRs are therefore computed relative the positions given by the grid matrix for the arrays.

7.1.2 Sampling Parameters

The four-element vector `s_par = [dx dy dt nt]` determines the spatial discretization and the temporal sampling properties. The spatial discretization of the transducer surface is given by `dx`×`dy`; small values of `dx` and `dy` results in a finer mesh, and a higher numerical accuracy, compared to larger ones. The temporal sampling interval Δt (or T_s) is given by `dt` and the length of the SIR-vector is determined by `nt`. If `nt` is chosen too low so that any non-zero component of the SIR is not within the time-window defined by `[delay dt*(nt-1)+delay]` an error message will be printed (See Section 7.1.3 for a description of the `delay` parameter and Section 7.1.6 for a description of the error handling in the DREAM toolbox).

`dx` Spatial discretization size in x-direction [mm].

`dy` Spatial discretization size in y-direction [mm].

`dt` Temporal discretization size [μs] ($T_s = \Delta t = 1$/sampling freq).

`nt` Length of spatial impulse response vector.

7.1.3 The Delay Parameter

The starting point of the SIR-vector(s) is given by the delay parameter `delay` ([μs]). The delay parameter can either be a scalar, then all SIRs will have the same starting point, or it can be a vector with a length that must be equal to the number of observation points. In the latter case each observation point has a SIR with a different starting point.

7.1.4 Material Parameters

The material parameters are given by the three-element vector `m_par = [v cp alfa]`:

`v` Normal velocity of the transducer surface[m/s].

`cp` Sound velocity of the medium [m/s].

`alfa` Attenuation coefficient [dB/(cm MHz)].

Note: if `alfa` $\neq 0$ then the SIRs are compensated for attenuation. Note that the attenuation calculation involves a computation of an inverse discrete Fourier transform of length `nt` for every surface element `dx`$\times$`dy` [2] This results in a longer computation time compared to when `alfa` $= 0$. See also Section 10.2.

7.1.5 Focusing parameters

The focusing in the DREAM toolbox is controlled with the two parameters `foc_met` and `focal`. The `foc_met` parameters is a text string that selects the focusing method, options are:

`'off'` : focusing not used.

`'x'` : focus in x only, $\left(\sqrt{x^2 + z_f^2}\right)/c_\mathrm{p}$.

`'y'` : focus in y only, $\left(\sqrt{y^2 + z_f^2}\right)/c_\mathrm{p}$.

`'xy'` : focus in both x and y, $\left(\sqrt{x^2 + y^2 + z_f^2}\right)/c_\mathrm{p}$.

`'x+y'` : focus in $x + y$, $\left(\sqrt{x^2 + z_f^2} + \sqrt{y^2 + z_f^2}\right)/c_\mathrm{p}$.

This type of focusing is used for the two single transducer functions (`dreamrect_f` and `dreamcirc_f` see Section 7.3) and for the array functions (Section 7.5). The spherical and cylindrical transducer functions also use focusing but there focusing is controlled by a single parameter `R` (see Sections 7.3.6, 7.3.7, 7.3.8, and 7.3.9).

7.1.6 Error Handling

There are three levels of error reporting for the transducer functions. An error typically occur when the SIR do not fit within the time window, defined by the delay, sampling period, and length parameters. The levels are controlled by the optional `err_level` parameter which is a text string with the following alternatives:

1. `'ignore'`: Here an error is silently ignored,
2. `'warn'`: An error message is printed but computation is not stopped,
3. `'stop'`: An error message is printed and the computation is stopped.

The error message contains a number that tells how many samples outside the time window the SIR is. The default error level is `'stop'` (if the `err_level` is omitted).

7.2 Output Parameters Common to all Transducer Functions

7.2.1 The SIR Output Argument

The first output argument of transducer functions, `H`, is a matrix or vector, containing the spatial impulse response(s).

```
H = dream***( ... );
```

Each column `H` contain the SIR for the corresponding entry in the observation point input matrix `Ro` (see Section 7.1.1).

7.2.2 The Error Output Argument

The second (optional) output argument, `err`, is negative if an error has occurred and 0 otherwise.

```
[H,err] = dream***( ... );
```

If, for example, `err_level = 'ignore'` then no error message will be printed but `err` will be negative if an error occurred so the error can be detected. This is useful for displaying error dialog boxes in GUIs, for example.

7.3 Single Element Transducers

As mentioned above all single element transducers are centered at $\mathbf{r} = (0, 0, 0)$. There is, however, no loss in generality since the response at other transducer positions can simply be obtained by offsetting the coordinate system after computation.

7.3.1 Line (strip) Transducer

The line, or strip, transducer has a length `a` and a (small) thickness equal to `dy` (in `s_par`).

Syntax:

```
H = dreamline(Ro,geom_par,s_par,delay,m_par,err_level);
```

Geometrical parameters:

```
geom_par = [a];
   a [mm] - length of the strip (in x-direction).
```

7.3.2 Rectangular Transducer

The size of the rectangular transducer is determined by `a` and `b`.

Syntax:

```
H = dreamrect(Ro,geom_par,s_par,delay,m_par,err_level);
```

Geometrical parameters:

```
geom_par = [a b];
  a [mm] - x-size.
  b [mm] - y-size.
```

7.3.3 Rectangular Focused Transducer

The size of the rectangular focused transducer is determined by `a` and `b` and the focusing is described in Section 7.1.5.

Syntax:

```
H = dreamrect_f(Ro,geom_par,s_par,delay,m_par,foc_met,focal,err_level);
```

Geometrical parameters:

```
geom_par = [a b];
   a [mm] - x-size.
   b [mm] - y-size.
```

7.3.4 Circular Transducer

The size of the circular transducer is determined by the single parameter `r`.

Syntax:

```
h  = dreamcirc(Ro,geom_par,s_par,delay,m_par,err_level);
```

Geometrical parameters:

```
geom_par = [r];
  r  - Radius of the transducer.
```

7.3.5 Focused Circular Transducer

The size of the focused circular transducer is determined by the parameter `r` and the focusing is described in Section 7.1.5.

Syntax:

```
h  = dreamcirc_f(Ro,geom_par,s_par,delay,m_par,foc_met,focal,err_level);
```

Geometrical parameters:

```
geom_par = [r];
  r  - Radius of the transducer.
```

7.3.6 Spherical Concave Transducer

Syntax:

```
H = dreamsphere_f(Ro,geom_par,s_par,delay,m_par,err_level);
```

Geometrical parameters:

```
geom_par = [r R];
   r [mm] - Radius of the transducer.
   R [mm] - Curvature radius of the transducer.
```

7.3.7 Spherical Convex Transducer

Syntax:

```
H = dreamsphere_d(Ro,geom_par,s_par,delay,m_par)
```

Geometrical parameters:

```
geom_par = [r R];
   r [mm] - Radius of the transducer.
   R [mm] - Curvature radius of the transducer.
```

7.3.8 Cylindrical Concave Transducer

Syntax:

```
H = dreamcylind_f(Ro,geom_par,s_par,delay,m_par,err_level);
```

Geometrical parameters:

```
geom_par = [a b R];
   a - x-size of the transducer.
   b - y-size of the transducer.
   R - Radius of the curvature.
```

7.3.9 Cylindrical Convex Transducer

Syntax:

```
H = dreamcylind_d(Ro,geom_par,s_par,delay,m_par)
```

Geometrical parameters:

```
geom_par = [a b R];
   a - x-size of the transducer.
   b - y-size of the transducer.
   R - Radius of the curvature.
```

7.4 Input Parameters Common to the Array Functions

7.4.1 The Array Grid Matrix

The positions of the array transducer elements are determined by the $L \times 3$ *grid matrix* `G`. The first column contain the (center) x-positions of the elements, the second the y-positions, and the third the z-positions (L is the number of elements), respectively. This approach is very flexible and allows for arbitrary array geometries that not is restricted to equally spaced linear or 2D arrays.

7.4.2 Array Focusing

The array focusing has an extra option, `'ud'`, compared to the focusing methods described in Section 7.1.5:

`'off`' : focusing not used,

`'x'` : see Section 7.1.5,

`'y'` : see Section 7.1.5,

`'xy'` : see Section 7.1.5,

`'x+y'` : see Section 7.1.5.

`'ud'` : user defined focusing.

When user defined focusing is used the `'focal'` parameter is a vector of focusing delays (in μs). Each element in `'focal'` then delays the signal to the corresponding element in the array (given by the grid matrix).

7.4.3 Beam Steering Parameters

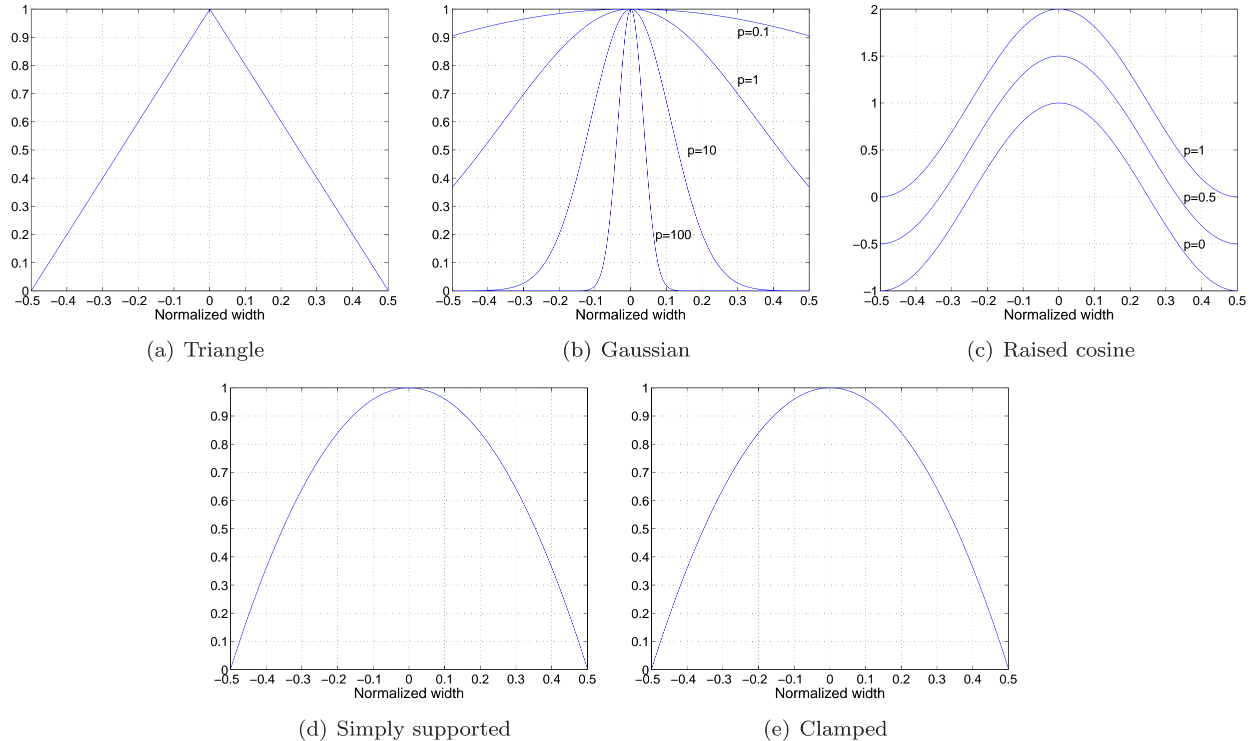
The beam steering in the DREAM toolbox is controlled by the two parameters `steer_met` and `steer_par`. The `steer_met` is a text string with four alternatives: `'off'`, `'x'`, `'y'`, and `'xy'`. The `steer_par` is a two-element vector `steer_par = [theta phi]` where `theta` [deg] is the x-direction steer angle and `phi` [deg] the y-direction steer angle.

7.4.4 Apodization Parameters

The DREAM toolbox has five pre-defined apodization windows that can be used for the array functions:

$$
\begin{aligned}
w_{\text{triangle}}(r) &= 1 - \frac{|r|}{r_{\max}} &(10)\\
w_{\text{gauss}}(r;p) &= \exp(-pr^2)/r_{\max}^2 &(11)\\
w_{\text{raised}}(r;p) &= p + \cos(r\pi/r_{\max}) &(12)\\
w_{\text{simply}}(r) &= 1 - r^2/r_{\max}^2 &(13)\\
w_{\text{clamped}}(r) &= (1 - r^2/r_{\max}^2)^2. &(14)
\end{aligned}
$$

Additionally to the pre-defined apodizations one can have a user defined apodization weights. Figure 7 show some examples of these function.



(a) Triangle (b) Gaussian (c) Raised cosine (d) Simply supported (e) Clamped

Figure 7: The pre-defined apodization window functions in the DREAM toolbox.

There are three parameters that controls the apodization in the DREAM toolbox, `apod_met`, `apod`, and `win_par`. The `apod_met` parameter is string variable for selecting apodization method, options are:

`'off'` - No apodization

`'ud'` - User defined apodization

`'triangle'` - Triangular window

`'gauss'` - Gaussian (bell-shaped) window

`'raised'` - Raised cosine window

`'simply'` - Simply supported

`'clamped'` - Clamped.

The second parameter, `apod`, is a vector of apodization weights that is used for the `'ud'` option. Pass an empty matrix (`[]`) if one of the other options are used. The last parameter, `win_par` (scalar), is used for raised cosine and Gaussian apodization functions. See also Section 10.1.

7.5 Array Transducers

7.5.1 Array with Rectangular Elements

Syntax:

```
H = dream_arr_rect(Ro,geom_par,G,s_par,delay,m_par,foc_met,
      focal,steer_met,steer_par,apod_met,apod,win_par,err_level);
```

Geometrical parameters:

```
geom_par = [a b]:
a  - Element size in x-direction.
b  - Element size in y-direction.
```

7.5.2 Array with Circular Elements

Syntax:

```
H = dream_arr_circ(Ro,a,G,s_par,delay,m_par,foc_met,
      focal,steer_met,steer_par,apod_met,apod,win_par,err_level);
```

Geometrical parameters: `r` - Radius of the transducer elements.

7.5.3 Array with Cylindrical Concave Elements

Syntax:

```
H = dream_arr_cylind_f(Ro,geom_par,G,s_par,delay,m_par,foc_met,
        focal,steer_met,steer_par,apod_met,apod,win_par,err_level);
```

Geometrical parameters:

```
geom_par = [a b r]:
  a  - Element size in x-direction.
  b  - Element size in y-direction.
  r  - Radius (y-direction).
```

Example (linear array):

```
gx = -10:1:10;
gy = zeros(length(gx),1);
gz = zeros(length(gx),1);
gx=gx(:);  gy=gy(:);  gz=gz(:);
G = [gx gy gz];
```

7.5.4 Array with Cylindrical Convex Elements

Syntax:

```
H = dream_arr_cylind_d(Ro,geom_par,G,s_par,delay,m_par,foc_met,
        focal,steer_met,steer_par,apod_met,apod,win_par,err_level);
```

Geometrical parameters:

```
geom_par = [a b r]:
  a  - Element size in x-direction.
  b  - Element size in y-direction.
  r  - Radius (y-direction).
```

7.5.5 Annular Array

Syntax:

```
H = dream_arr_annu(Ro,G,s_par,delay,m_par,foc_met,focal,apod_met,apod,
        win_par,err_level);
```

Geometrical parameters:

```
G  - Vector of annulus radies.
```

The first element in `G` is the radius of the center element; then `G` has two entries for each annulus (the inner and outer radius). Hence, the length of `G` must be odd for the annular array.

The `apod_met` parameter has the following entries for the annular array function:

`'on'` - Focusing on (xy), see Section 7.1.5,

`'off'` - No focusing,

`'ud'` - User defined focusing.

8 Parallel Processing Support

Since the SIR computation for two different observation points is independent one can easily divide the observation points in sets and let a different process handle each set. On a multiprocessor machine, where each process (or thread) can run on a separate CPU simultaneously, this will speed up the computations. In the DREAM toolbox this is performed using (POSIX) threads; the SIRs for each set is computed in its own thread.The DREAM functions with thread support has an extra parameter `n_cpus` and a "`_p`" in its function name. For example, to compute SIRs for a rectangular transducer on 2 CPUs use:

```
n_cpus = 2;
H = dreamrect_p(Ro,geom_par,s_par,delay,m_par,n_cpus,'stop');
```

where `n_cpus` is an integer ≥ 1. To see if the two threads are running using Linux, open new terminal window and use the command `top`. You should see something like:

```
17:10:15  up 62 days, 21:18,  4 users,  load average: 0.77, 0.26, 0.16
80 processes: 77 sleeping, 3 running, 0 zombie, 0 stopped
CPU0 states: 100.0% user   0.0% system    0.0% nice   0.0% iowait   0.0% idle
CPU1 states: 100.0% user   0.0% system    0.0% nice   0.0% iowait   0.0% idle
Mem:  6211732k av, 5835092k used,  376640k free,       0k shrd,  284960k buff
1280556k active,            4156076k inactive
Swap: 4096496k av,     420k used, 4096076k free                 5000568k cached

PID USER     PRI  NI  SIZE  RSS SHARE STAT LC %CPU %MEM   TIME COMMAND
29403 fl        25   0 75916  74M 11820 R     0 99.9  1.2   0:11 matlab
29404 fl        25   0 75916  74M 11820 R     1 99.9  1.2   0:11 matlab
29395 fl        15   0  1304 1304   968 R     2  0.3  0.0   0:00 top
1 root      15   0   472  460   424 S     2  0.0  0.0   0:49 init
2 root      RT   0     0    0     0 SW    0  0.0  0.0   0:00 migration_CPU0
3 root      RT   0     0    0     0 SW    1  0.0  0.0   0:00 migration_CPU1
```

As one can see there are two MATLAB processes running on 99.9% instead on just one. Note that there is an overhead when creating (and running) new threads. If the distributed computations are very short the computations may not be faster for the parallel algorithm than the serial. On uni-processor computers the thread enabled functions may be slower than the non-threaded functions due to this overhead (if the hardware and software do not support hyper threading).

The functions currently available with parallel (thread) support is shown in Table 2.

Transducer type/operation	**DREAM function**	**Linux**	**Mac**	**Windows**
Strip transducer	`dreamline_p`	Yes	Yes	Yes (see Sec. 4.2)
Rectangular transducer	`dreamrect_p`	Yes	Yes	Yes (see Sec. 4.2)
Focused rectangular transducer	`dreamrect_f_p`	Yes	Yes	Yes (see Sec. 4.2)
Circular transducer	`dreamcirc_p`	Yes	Yes	Yes (see Sec. 4.2)
Focused circular transducer	`dreamcirc_f_p`	Yes	Yes	Yes (see Sec. 4.2)
Spherical concave transducer	`dreamsphere_f_p`	Yes	Yes	Yes (see Sec. 4.2)
Spherical convex transducer	`dreamsphere_d_p`	Yes	Yes	Yes (see Sec. 4.2)
Cylindrical concave transducer	`dreamcylind_f_p`	Yes	Yes	Yes (see Sec. 4.2)
Cylindrical convex transducer	`dreamcylind_d_p`	Yes	Yes	Yes (see Sec. 4.2)
Array with rectangular elements	`dream_arr_rect_p`	Yes	Yes	Yes (see Sec. 4.2)
Array with circular elements	`dream_arr_circ_p`	Yes	Yes	Yes (see Sec. 4.2)
Array with cylindrical concave el.	`dream_arr_cylind_f_p`	Yes	Yes	Yes (see Sec. 4.2)
Array with cylindrical convex el.	`dream_arr_cylind_d_p`	Yes	Yes	Yes (see Sec. 4.2)
Annular array	`dream_arr_annu_p`	Yes	Yes	Yes (see Sec. 4.2)
Time-domain convolution	`conv_p`	Yes	Yes	Yes (see Sec. 4.2)
Frequency-domain convolution	`fftconv_p`	Yes	Yes	Yes (see Sec. 4.2)
Frequency-domain convolution	`sum_fftconv_p`	Yes	Yes	Yes (see Sec. 4.2)
Parallel copy of data	`copy_p`	Yes	Yes	Yes (see Sec. 4.2)
Synthetic aperture focusing technique	`saft_p`	Yes	Yes	Yes (see Sec. 4.2)

Table 2: Functions in the DREAM Toolbox with parallel (thread) support.

9 Analytic Transducer Functions

The DREAM Toolbox has two new time-continous analytic functions, and one sampled analytic function, for circular and a rectangular transducers, respectively [4, 6]. The parameters for the functions,

```
[Y] = circ_sir(Ro,geom_par,delay,s_par,m_par);

[Y] = rect_sir(Ro,geom_par,delay,s_par,m_par);

[Y] = scirc_sir(Ro,geom_par,delay,s_par,m_par,n_int);
```

are similar to the other DR-based transducer functions, except for the sampled analytic circular function, `scirc_sir`, which has a parameter `n_int` that determines the number of points to use in the (numerical) temporal integration for each sampling interval [cf. Eq. (4)].

10 Misc Functions

10.1 Apodization Windows

The DREAM toolbox includes function,

```
[w] = dream_apodwin(apod_met,a,win_par)
```

that can be used to compute apodization weights using the methods described is Section 7.4.4. The input parameters are:

`apod_met` - Text string for selecting apodization method. options are: `'off'`, `'ud'`, `'triangle'`, `'gauss'`, `'raised'`, `'simply'`, and `'clamped'`.

`a` - Is the number of points of the apodization window.

`win_par` - Scalar parameter for raised cosine and Gaussian apodization functions.

10.2 Attenuation Response

The function,

```
[H] = dream_att(Ro,s_par,delay,m_par);
```

can be used to compute only the impulse response(s) that is due to attenuation. The input parameters for `dream_att` are:

`Ro` - See Section 7.1.1.

`s_par = [dt,nt];` - See Section 7.1.2.

`delay` - See Section 7.1.3.

`m_par= [cp,alfa];` - See Section 7.1.4.

10.3 One Dimensional Matrix Convolution Functions

The conv_p and fftconv_p functions computes the one dimensional convolution of the columns in a matrix A and matrix (or vector) B using one or more threads. These functions are typically used to compute single-path or double-path (pulse-echo) responses for a large number of observation points and thus avoiding a slow for-loop using the conv function (that only takes vector arguments).

Syntax:

```
[Y] = conv_p(A,B,n_cpus)
      conv_p(A,B,n_cpus,Y)
      conv_p(A,B,n_cpus,Y,mode_string)
```

and

```
[Y] = fftconv_p(A,B,n_cpus)
[Y,wisdom_string] = fftconv_p(A,B,n_cpus)
[Y] = fftconv_p(A,B,n_cpus,wisdom_string)
      fftconv_p(A,B,n_cpus,Y,wisdom_string)
      fftconv_p(A,B,n_cpus,Y,mode_string)
      fftconv_p(A,B,n_cpus,Y,wisdom_string,mode_string)
```

where A is an $M \times N$ matrix, B is a $K \times N$ matrix or a K-length vector. If B is a vector then each column in A is convolved with B. The parameter n_cpus is the number of threads to use, which must be greater or equal to 1, and Y is the $(M + K - 1) \times N$ output matrix.

The fftconv_p performs the convolutions in the frequency domain using the FFTW library [7] which is significantly faster than the time-domain implementation conv_p for long vectors. To use the fftconv_p the FFTW3 lib must be available. The windows version of fftconv_p is linked to the FFTW lib which, as previously described, can be found at ftp://ftp.fftw.org/pub/fftw/fftw3win32mingw.zip.

10.3.1 Using Pre-computed FFTW Plans

To speed up computation of repeated fftconv_p operations (of the same size) one can use pre-computed fftw plans. First compute the plan for the corresponding vector length,

```
[tmp,wisdom_string] = fftconv_p(A(:,1),B(:,1),n_cpus);
```

then use the plan in consecutive computations of the same size,

```
for n=1:N

  % Compute new A and B here.

  [Y] = fftconv_p(A,B,n_cpus,wisdom_string);

end
```

The time-consuming call to fftw plan functions is then avoided at each call to fftconv_p inside the loop.

10.3.2 Using the In-place Mode

If the involved matrices are large then memory allocation can be time consuming. To alleviate this problem both conv_p, fftconv_p and sum_fftconv_p (see Section 10.3.3) have an in-place mode that

uses a pre-allocated output matrix, hence memory allocation for the (large) output matrix Y is avoided. For the conv_p and fftconv_p functions, the in-place operation has three modes, '=', '+=', and '-=', respectively.

The default '=' mode

```
[Y] = fftconv_p(H,u,n_cpus);

for n=2:N

  % Compute new H and u here.

  fftconv_p(H,u,n_cpus,Y,'+=');

end
```

Note the in-place mode have the side effect that, in the code

```
  X = Y;
  fftconv_p(H,u,n_cpus,Y,'+=');
```

both X and Y will be altered, since Malab/Octave do not make a copy of a matrix unless it is changed after X = Y assignment.

10.3.3 Computing Array Responses for Arbitrary Input Signals

The pressure response at an observation point $\mathbf{r}$ can be computed by super-imposing the responses from the individual array elements. Assume that we have an array with K transmit elements. The pressure response $p(\mathbf{r},t)$ is then given by,

$$p(\mathbf{r},t) = \sum_{k=0}^{K-1} h_k^{\text{f-SIR}}(\mathbf{r},t) * h_k^{\text{ef}}(t) * u_k(t), \tag{15}$$

where $h_k^{\text{f-SIR}}(\mathbf{r},t)$ is the forward SIR for the kth transmit element, $h_k^{\text{ef}}(t)$ is the forward electro-acoustical impulse response, and $u_k(t)$ is the kth input signal. The DREAM Toolbox has a function, sum_fftconv_p, to facilitate computation of discrete array responses with arbitrary input signals. Similar to the fftconv_p function, the sum_fftconv_p function uses a FFT based algorithm to compute the convolutions. The sum_fftconv_p function performs an operation similar to the code:

```
YF = zeros(M+K-1,N);
for l=1:L
  for n=1:N
    YF(:,n) = YF(:,n) + fft(A(:,n,l),M+K-1).* fft(B(:,l),M+K-1);
  end
end
Y = real(ifft(Y));
```

where A is a 3D matrix.

```
[Y] = sum_fftconv_p(A,B,n_cpus);
[Y] = sum_fftconv_p(A,B,n_cpus,wisdom_str);
      sum_fftconv_p(A,B,n_cpus,Y);
      sum_fftconv_p(A,B,n_cpus,Y,wisdom_str);
```

Input parameters:

- `A` An $M \times N \times L$ three-dimensional matrix.
- `B` A $K \times L$ matrix.
- `n_cpus` Number of threads to use where `n_cpus` must be greater or equal to 1.
- `wisdom_str` Optional parameter (see Section 10.3).

and the output parameter, `Y` , is an $(M + K - 1) \times N$ matrix. A typical usage is:

```
% Compute a new fftw wisdom string.
[tmp,wisdom_str]  = fftconv_p(A(:,1,1),B(:,1),n_cpus);

for i=1:N

  % Do some stuff here.

  Y = sum_fftconv_p (A,B,n_cpus,wisdom_str);
end
```

where the overhead of calling fftw plan functions is now avoided inside the for loop.

10.4 Parallel Matrix Copy

The impulse response matrices can often become rather large and the time taken to copy data between matrices can therefore be considerable. The DREAM Toolbox comes with a threaded function `copy_p` to speed up data copy.

In-place, threaded copy of data into a matrix:

```
copy_p(B,row_idx,col_idx,A,n_cpus);
```

Input parameters:

```
B                  - Pre-allocated Output matrix of size >= (r2-r1)x(c2-c1).
row_idx = [r1 r2] - Two element vector defining rows in B where
                    the data is copied.
col_idx = [c1 c2] - Two element vector defining columns in B where
                    the data is copied.
B                  - Input matrix of size (r2-r1)x(c2-c1);
n_cpus             - Number of threads to use, n_cpus must be greater or equal to 1.
```

10.5 The Speed of Sound in Water

The SIRs are a function of the sound speed in the propagation medium which often is water. The DREAM toolbox comes with a function `h2o_soundspeed`, based on a method by V.A. Del Grosso [8], to facilitate computation of the water sound speed (as function of temperature, pressure, and salinity).

```
[cp] = h2o_soundspeed(T,P,unit,S)
```

Input parameters:

```
T    - Temperture [in degrees Celsius].
P    - Pressure (optional).
unit - Text string defining the pressure unit of arg 2
       ['Pa','bar','at','atm','mmHg', or, 'psi']  (optional).
S    - Salinity [in parts per thousand] (optional).
```

11 Signal Processing Examples

11.1 Double-path Modeling

Double-path responses can be modeled as convolutions between the forward and backward responses [9]. This operation can be time consuming when the number of observation points is large. The DREAM toolbox has the threaded functions `conv_p` and `fftconv_p` that can be used to speed up this operation. A typical example is:

```
H = fftconv_p(H_t,H_r,n_cpus); % Double-path.

P = fftconv_p(H,h_e,n_cpus); % Electrical impulse response.
```

where we first compute the double-path SIRs and then add (convolve) the double-path electrical impulse response to get the (double-path) propagation response matrix `P`.

11.2 Synthetic Aperture Imaging — The Synthetic Aperture Focusing Technique

Synthetic aperture imaging (SAI) was developed to improve resolution in the along track direction for side-looking radar. The idea was to record data from a sequence of pulses from a single moving real aperture and then, with suitable computation, combine the signals so the output can be treated as a much larger aperture. The first synthetic aperture radar (SAR) systems appeared in the beginning of the 1950's [10, 11]. Later on the method has carried over to ultrasound imaging in areas such as synthetic aperture sonar (SAS) [12], medical imaging, and nondestructive testing [13, 14], where the method is often called the synthetic aperture focusing technique (SAFT).

The conventional time-domain SAFT algorithm performs synthetic focusing by means of coherent summations, of responses from point scatterers, along hyperbolas.[10] These hyperbolas simply express the distances, or time-delays, from transducer positions in the synthetic aperture to the observation points, see illustration in Figure 8. More specifically, to achieve focus at an observation point $(x_{\tilde{n}}, z_m)$, the SAFT algorithm time shifts and performs a summation of the received signals $y(x_n, t)$ measured at transducer positions x_n for all n in the synthetic aperture. The time shifts which aim to compensate for differences in pulse traveling time, are simply calculated using the Pythagorean theorem and the operation is commonly expressed in the continuous time form [15]

$$o(x_{\tilde{n}}, z_m) = \sum_n w_n y(x_n, \frac{2}{c_\mathrm{p}} \sqrt{(x_{\tilde{n}} - x_n)^2 + z_m^2}). \tag{16}$$

where $o(x_{\tilde{n}}, z_m)$ is the beamformed image.

The DREAM Toolbox has two functions `saft` and `saft_p`, respectively, that performs the SAFT operation (with linear interpolation):

```
Y = saft(B,To,delay,s_par,m_par,Ro,a);
Y = saft_p(B,To,delay,s_par,m_par,Ro,a,n_cpus);
```

[10] Linear scanning of the transducer is assumed here.

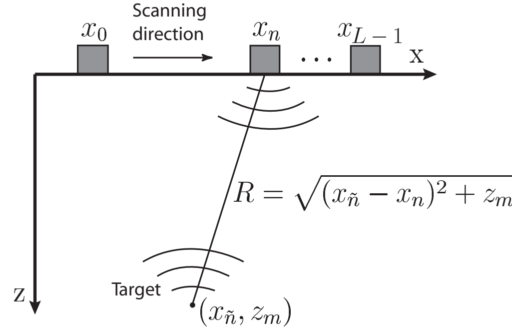


Figure 8: Typical measurement setup for a SAFT experiment. The transducer is mechanically scanned along the x-axis and at each sampling position, x_n, $n = 0, 1, \ldots, L-1$, a data vector (A-scan) of length K is recorded. The distance between the transducer, at $(x_n, z = 0)$, and the observation point, $(x_{\tilde{n}}, z_m)$, is given by R.

Input parameters:

```
B     - KxL Ultrasonic B-scan data matrix.
To    - A matrix of the form  [xo1 yo1 zo2; xo2 yo2 zo2; ... xoL yoL zoL];
        where L is the number of transducer positions.
delay - A scalar (or vector) of starting point(s) of the A-scans in B.

n_cpus - Number of threads to use. n_cpus must be greater or equal to 1.
```

11.3 Delay-and-sum Imaging

The SAFT method described in Section 11.2 is a special case of methods based on so-called delay-and-sum imaging (DAS). The simple idea is just to compensate for the double-path propagation delay from each transmitter/receiver to each observation point and then perform a (coherent) summation. This operation is essentially based on an geometrical optics approach and is analogous to the operation of an acoustical lens [16].

The DREAM Toolbox has two functions to facilitate (matrix based) delay-and-sum processing. The two DAS functions das and das_arr is similar to the transducer functions in the sense that they return a matrix with responses corresponding to each observation point. The difference from the transducer functions is that only the time delay is computed which is represented with a "1" at the corresponding index.

The das function is for single transducers and the das_arr function is for arrays. The input parameters,

```
[D,err] = das(Ro,s_par,delay,m_par,err_level)

[D,err] = das_arr(Ro,G,s_par,delay,m_par,foc_met,...
   focal,steer_met,steer_par,apod_met,apod,win_par,err_level)
```

are similar to the ones used in the transducer functions previously described.

11.4 Model Based Ultrasonic Array Imaging

In this section a short introduction to model based ultrasonic imaging is presented. Model based ultrasonic array imaging [17–19] is different from delay-and-sum imaging in the sense that is based on optimal information processing whereas delay-and-sum imaging is based on geometrical focusing. The idea is to use a (linear) model, taking into account the diffraction effects associated with each transmitter/receiver, the electrical characteristics for each transmitter/receiver, as well as the used input signal(s), and then estimate the parameters (the scattering strengths) of the model based on both data and prior information. The DREAM toolbox can be used in this process for computing the SIRs for the model which can then be convolved with measured electrical impulse responses to obtain a model for a real measurement setup or using only simulated impulse responses to evaluate different array designs, for example.

To obtain a linear mode we need to consider both the forward process and the backward process. As discussed in Section 5.1 [Eq. (2)] the forward response can be divided in three parts: the input signal $u(t)$, the forward electro-acoustical response $h^{\text{ef}}(t)$, and the forward SIR $h^{\text{f-SIR}}(\mathbf{r},t)$. The backward response can similarly be divided in two parts: the backward acousto-electrical impulse response $h^{\text{eb}}(t)$ and the backward SIR $h^{\text{b-SIR}}(\mathbf{r},t)$. Now, consider an array with K transmit elements and L receive elements and contributions from a single observation point, $\mathbf{r} = \begin{bmatrix} x & y & z \end{bmatrix}^T$, where ^{T} denotes the transpose operator. The received signal, $y_l(\mathbf{r},t)$, from the lth receive element can be expressed

$$\begin{aligned} y_l(\mathbf{r},t) = &\overbrace{\left(\sum_{k=0}^{K-1} h_k^{\text{f-SIR}}(\mathbf{r},t) * h_k^{\text{ef}}(t), *u_k(t)\right)}^{\text{Forward impulse response (f)}} o(\mathbf{r}) * \\ &\overbrace{h_l^{\text{b-SIR}}(\mathbf{r},t) * h_l^{\text{eb}}(t)}^{\text{Backward impulse response (b)}} \qquad + e_l(t), \\ &= h^{\text{f}}(\mathbf{r},t) * h_l^{\text{b}}(\mathbf{r},t) o(\mathbf{r}) + e_l(t), \\ &= h_l(\mathbf{r},t) o(\mathbf{r}) + e_l(t), \end{aligned} \tag{17}$$

where $*$ denotes temporal convolution and $e_l(t)$ is the noise for the lth receive element. Note that the total forward impulse response is a superposition of the forward impulse responses corresponding to all transmit elements. The *object function* $o(\mathbf{r})$ is the scattering strength at the observation point $\mathbf{r}$, $h_k^{\text{ef}}(t)$ is the forward electrical impulse response for the kth transmit element, $h_l^{\text{eb}}(t)$ the backward electrical impulse response for the lth receive element, and $u_k(t)$ is the input signal for the kth transmit element.

A discrete-time version of (15) is obtained by sampling the impulse responses and by using discrete-time convolutions. If we consider N observation points then the received discrete waveform from a target at the nth observation point, $\mathbf{r}_n = (x_n, y_n, z_n)$, can be expressed as

$$\mathbf{y}_l^{(n)} = \mathbf{h}_l^{(n)} (\mathbf{o})_n + \mathbf{e}_l, \tag{18}$$

where the column vector $\mathbf{h}_l^{(n)}$ is the discrete system impulse response for the lth receive element.[11] The vector $\mathbf{o}$ represents the N scattering amplitudes in the region-of-interest, and the notation $(\mathbf{n})_n$ denotes the nth element in $\mathbf{o}$.[12]

To obtain the received signal for all observation points we need to perform a summation over n, which equivalently can be expressed as a matrix-vector multiplication, according to

$$\begin{aligned} \mathbf{y}_l &= \sum_n \mathbf{h}_l^{(n)} (\mathbf{o})_n + \mathbf{e}_l, \\ &= \begin{bmatrix} \mathbf{h}_l^{(0)} & \mathbf{h}_l^{(1)} & \cdots & \mathbf{h}_l^{(N-1)} \end{bmatrix} \mathbf{o} + \mathbf{e}_l, \\ &= \mathbf{P}_l \mathbf{o} + \mathbf{e}_l, \end{aligned} \tag{19}$$

[11] Here all vectors are by convention column vectors.

[12] The vector $\mathbf{o}$ can easily be rearrange to form an image when two-dimensional imaging is considered [20].

which gives us a liner model for the data for one receive element.

To obtain a model for all elements we can append all L receive signals $\mathbf{y}_l$ into a single vector $\mathbf{y}$ and we finally have linear model for the total array setup

$$\begin{aligned}\mathbf{y} = \begin{bmatrix}\mathbf{y}_0\\ \mathbf{y}_1\\ \vdots\\ \mathbf{y}_{L-1}\end{bmatrix} &= \begin{bmatrix}\mathbf{P}_0\\ \mathbf{P}_1\\ \vdots\\ \mathbf{P}_{L-1}\end{bmatrix}\mathbf{o} + \begin{bmatrix}\mathbf{e}_0\\ \mathbf{e}_1\\ \vdots\\ \mathbf{e}_{L-1}\end{bmatrix}\\ &= \mathbf{P}\mathbf{o} + \mathbf{e},\end{aligned} \tag{20}$$

for the total array imaging system [20]. The *propagation matrix*, $\mathbf{P}$, in (20) now describes both the transmission and the reception process for an arbitrary focused array. Note that the position of the observation points, and the corresponding scattering amplitudes represented by the vector $\mathbf{o}$, is not restricted to a regular two-dimensional grid, which is often used in ultrasonic imaging. Furthermore, the array elements can in fact, similar to the observation points , be positioned at arbitrary locations in tree-dimensional space space. Thus, the model (20) can also be used to model two-dimensional arrays as well as to model array responses in tree-dimensional space. Also note that the "noise" vector $\mathbf{e}$ describes the *uncertainty* of the model (20). The noise $\mathbf{e}$ does not only model the measurement noise but also all other errors that we may have, such as: multiple scattering effects, cross talk between array elements, non-uniform sound speed in the media, etc.

11.4.1 The Matched Filter

The matched filter has the property of maximizing the signal-to-noise ratio (at a single point). The matched filter for each observation point is given by [18]

$$\hat{\mathbf{o}} = \mathbf{P}^T\mathbf{y}. \tag{21}$$

Note that the structure of the matched filter is similar to delay-and-sum processing which also can be expressed as a matrix-vector multiplication (see the `model_based_example.m` on the DREAM website),

$$\hat{\mathbf{o}} = \mathbf{D}^T\mathbf{y} \tag{22}$$

where the delay matrix $\mathbf{D}$ has ones in the positions corresponding the the propagation delays and zeros otherwise.

11.4.2 The Optimal Linear Estimator

The optimal linear estimator (or the Wiener filter) is given by [17, 19]

$$\begin{aligned}\hat{\mathbf{o}} &= \mathbf{C}_o\mathbf{P}^T(\mathbf{P}\mathbf{C}_o\mathbf{P}^T + \mathbf{C}_e)^{-1}\mathbf{y}\\ &= (\mathbf{P}^T\mathbf{C}_e^{-1}\mathbf{P} + \mathbf{C}_o^{-1})^{-1}\mathbf{P}^T\mathbf{C}_e^{-1}\mathbf{y},\end{aligned} \tag{23}$$

where $\mathbf{C}_o$ is the covariance matrix for $\mathbf{o}$ and $\mathbf{C}_e$ is the covariance matrix for $\mathbf{e}$, respectively. The estimator (23) has the property, not found for the matched filter or for delay-and-sum methods, namely that any beampattern can be compensated given that the signal-to-noise ratio is sufficient.

As a final not on model based imaging is that the matrices involved normally become rather large. It is therefore highly recommended to used a tuned linear algebra library, such as K. Goto's BLAS library or the ATLAS library, for example. These libraries often have thread support so that all CPUs on the computer can be utilized.

An example of model based (and matrix based delay-and-sum) imaging can be found on the DREAM website (see the `model_base_example.m` file on the examples page).

12 The Graphical User Interface (Matlab only)

To launch the DREAM Toolbox GUI, type `dream_gui` in the MATLAB Command Window and a graphical user interface will be raised as shown in Figure 9. After activating the user interface, users can set

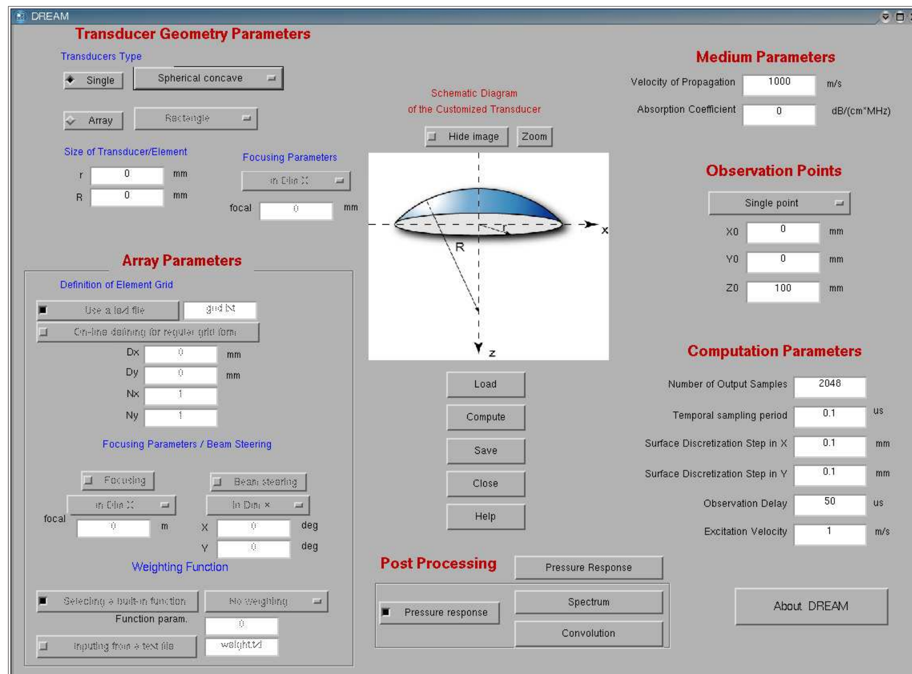


Figure 9: The graphical user interface.

parameters, compute SIRs, and save and process the resulting SIR by using the graphical controls. For setting of geometry parameters, please also refer to the schematic diagram of transducer displayed in the center of the graphic interface. After setting all the parameters, clicking on the *Compute* button starts to compute SIR. In addition, the DREAM GUI provides several post-processing operations to process the SIRs, including computing pressure response, computing the spectrum of the SIR, and convolving the SIR with an excitation signal. Notice that if users tick the checkbox of pressure response, the spectrum and convolution operations will be performed on the pressure instead of on the SIR. The parameters and resulting SIR can be stored in an `*.mat` file by clicking on the button of *Save*. The parameters settings can then be restored by loading the file using the *Load* button.

13 Known Issues

- On Windows platforms the MEX-files can not be aborted by pressing `CTRL-C` since Windows lacks real asynchronous signals, see: `http://www.mathworks.com/support/solutions/data/1-188VX.html` Therefore, when `CTRL-C` is pressed, using Windows, the operation is interrupted after the MEX-file has finished.
- On Linux x86 platforms the MATLAB release 14 library `matlab/sys/os/glnx86/libgcc_s.so` seems to be buggy causing MATLAB to abort when a `mexErrMsg` call is done within a MEX-file. The threaded functions uses the `mexErrMsg` function when `CTRL-C` is pressed. A work-around is to set the environment variable `LD_PRELOAD` to point to a working library. If you, for example, are using

gcc 3.4.4, then you can set

```
export LD_PRELOAD=/usr/lib/gcc/i686-pc-linux-gnu/3.4.4/libgcc_s.so
```

or rename `matlab/sys/os/glnx86/libgcc_s.so` and copy (or link) a working `libgcc_s.so` to the `matlab/sys/os/glnx86/` directory.

- Matlab's memory manager is not thread safe (at least not R2007b and older). The functions that use the fftw library may therefore crash if `n_cpus` > 1. A workaround is to set the env. variable `MATLAB_MEM_MGR=system` (i.e., `export MATLAB_MEM_MGR=system`).
- The file `mex_sum_fftconv_p.c` do not compile unless MATLAB $\geq$ R2006b is installed. The problem is that `mwSize` must be defined in `matlab/extern/include/matrix.h` (`sum_fftconv_p` compiles for Octave).
- The computation of responses for lossy media is rather time consuming; the computation time can be as large as a factor of 1000 times longer than for loss-less media. This due to the fact that the frequency domain attenuation response is computed, and an inverse FFT is performed, for every surface element, `dx`$\times$`dy`, of the transducer surface.

References

[1] B. Piwakowski and B. Delannoy, "Method for computing spatial pulse response: Time-domain approach", J. Acoust. Soc. America **86**, 2422–32 (1989).

[2] B. Piwakowski and K. Sbai, "A new approach to calculate the field radiated from arbitrarily structured transducer arrays", IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control **46**, 422–40 (1999).

[3] G. E. Tupholme, "Generation of acoustic pulses by baffled plane pistons", Mathematika **16**, 209–224 (1969).

[4] P. Stepanishen, "Transient radiation fom pistons in an infinite planar baffle", J. Acoust. Soc. America **49**, 1629–38 (1971).

[5] K. Aki and P. Richards, *Quantitative Seismology* (W.H. Freeman, San Francisco) (1980).

[6] J. Lockwood and J. Willette, "High-speed method for computing the exact solution for the pressure variations in the nearfield of a baffled piston", J. Acoust. Soc. America **53**, 735–741 (1973).

[7] http://www.fftw.org.

[8] V. A. Del Grosso, "New equation for the speed of sound in natural waters (with comparisons to other equations)", J. Acoust. Soc. America **56**, 1084–109 (1974).

[9] P. Stepanishen, "Pulsed transmit/receive response of ultrasonic piezoelectric transducers", J. Acoust. Soc. America **69**, 1815–1827 (1981).

[10] C. Wiley, "Synthetic aperture radars", IEEE Trans. on Aerosp. Electron. Syst. **21**, 440–443 (1985).

[11] C. Sherwin, J. Ruina, and R. Rawcliffe, "Some early developments in synthetic aperture radar systems", IRE Trans. Military Electron. **6**, 111–115 (1962).

[12] P. Gough and D. Hawkins, "Imaging algorithms for strip-map synthetic aperture sonar: Minimizing the effects of aperure errors and aperture undersampling", IEEE Journal of Oceanic Engineering **22**, 27–39 (1997).

[13] J. Seydel, "Ultrasonic synthetic-aperture focusing techniques in NDT", Research Techniques for Nondestructive Testing (Academic Press) (1982).

[14] S. Doctor, T. Hall, and L. Reid, "SAFT—the evolution of a signal processing technology for ultrasonic testing", NDT International **19**, 163–172 (1986).

[15] C. Frazier and J. W.D. O'Brien, "Synthetic aperture techniques with a virtual source element", IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control **45**, 196–207 (1998).

[16] G. S. Kino, *Acoustic Waves: Devices, Imaging and Analog Signal Processing*, volume 6 of *Prentice-Hall Signal Processing Series* (Prentice-Hall) (1987).

[17] R. Stoughton, "Source imaging with minimum mean-squared error", jasa **94**, 827–834 (1993).

[18] F. Lingvall, T. Olofsson, and T. Stepinski, "Synthetic aperture imaging using sources with finite aperture: Deconvolution of the spatial impulse response", J. Acoust. Soc. America **114**, 225–234 (2003).

[19] F. Lingvall and T. Olofsson, "On time-domain model based ultrasonic array imaging", **54**, 1623–1633 (2007).

[20] F. Lingvall, "A method of improving overall resolution in ultrasonic array imaging using spatio-temporal deconvolution", Ultrasonics **42**, 961–968 (2004).

A Building the Pthreads Library for 32 and 64 bit Windows

The DREAM toolbox uses pthreads (POSIX threads) to run computations in parallel on several cores/cpus. To run the parallel functions on Windows the *Pthreads-win32* library is used: http://sourceware.org/pthreads-win32/. The most current version of the Pthreads-win32 library, which is the one used by DREAM, is version 2.8.0 (2006-12-22). A 32-bit binary version of the library can be found at: ftp://sourceware.org/pub/pthreads-win32/. The MinGW-w64 project has a patch to build the Pthreads-win32 lib for 64-bit Windows, and there are some more 64-bit patches at: http://www.cadforte.com/wiki/index.php/Pthreads.

To facilite building the lib we have included our own (experimental) patch, which is a combination of the patches above (with some minor changes), and the source code for the Pthreads-win32 lib in the source code packge of the DREAM toolbox. There are also two (bash) build scripts which can be used for building the 32-bit and 64-bit libs, respectively. This is tested using the MinGW-w32 and MinGW-w64 cross compiler tool chains on Linux. After uncompressing the DREAM source code packge you can find these files in the `windows` folder.

Given that you have installed the MinGW-w32 cross compiler you can build the 32-bit Pthreads-win32 with

```
# ./build_pthreads_win32.sh
```

This will build the `pthreadGC2.dll` file and copy it (the `pthread.def` file, and the header files `pthread.h`, `sched.h` and, `semaphore.h`) to the `windows/dll` folder.

For 64-bit windows you first need to install the MinGW-w64 cross compiler and then the script

```
# ./build_pthreads_win64.sh
```

will copy the Pthreads-win32 sources, patch them for 64-bits support, and build the lib. The 64-bit lib (the `pthread.def` file, and the header files `pthread.h`, `sched.h` and, `semaphore.h`) will be installed in the `windows/dll_64` folder.

Note: if you intend to use these libs with one of the MSVC compilers then you also need the corresponding `lib` files. These `lib` files can be generated using the windows bat-files found in the `windows/dll` and `windows/dll_64` folders, respectively.

B Building the FFTW Library for 32 and 64 bit Windows

For conveniance we have also included the sources for the FFTW library and two scripts to build the library for both 32 and 64 bit Windows. The (bash) scripts uses MinGW-w32 and MinGW-w64 cross compiler tool chains, respectively. Change directory to the `windows` folder and run

```
# ./BUILD-MINGW-FFTW.sh
```

to build for 32-bit Windows or

```
# ./BUILD-MINGW_64-FFTW.sh
```

for 64-bit Windows. The two build scripts are based on the corresponding ones at the FFTW site adapted for building DREAM on Windows. The scripts install the libraries in the `windows/dll` and `windows/dll_64` folders, respectively.

Similary to the Pthreads-win32 lib you need to build the `lib` files if you intend to use these libs with one of the MSVC compilers. These `lib` files can be generated using the corresponding windows bat-files found in the `windows/dll` and `windows/dll_64` folders, respectively.