Design of the LTFAT toolbox

Peter L. Søndergaard, Zdeněk Průša

September 2, 2014

Contents

1 About this document

The purpose of this document is to collect information on how to contribute and develop LTFAT, and collect information about *why*[1]the toolbox has come to look the way it does.

2 Goals of the software

- High quality code: Easy to read and maintain
- High quality documentation in the code: This is the best place to keep the documentation, as it is easier to keep up to date when the code changes. The documentation can be extracted in HTML and TEX formats.
- Simplicity: Eliminate unnecessary configuration options. Always choose the simplest possible solution. This makes it much easier to read and modify the code.

3 Naming conventions

In order to avoid confusion, parameters and variables should have the same name across different functions. New names will be added when needed.

3.1 Common input/output parameters

a	Integer. Length of time-shift, hop-size.
AF, BF	Scalars. Frame bounds.
c	2D/3D matrix. Gabor/Wilson coefficients
f	Vector/matrix. Signal.
g	Cell array/text string defining a window
ga	Vector. Analysis window.
gd	Vector. Dual window.
gs	Vector. Synthesis window. To avoid doubt in users, the ga and gs combination carries the most meaning.
gt	Vector. Tight window.
$tgrad$	2D/3D matrix. Time gradient of the phase, the local instantaneous frequency.
$fgrad$	2D/3D matrix. Frequency gradient of the phase, the local instantaneous time.
L	Integer. Length of the Gabor transform to be done.
Ls	Integer. Length of a signal.
M	Integer. Number of channels in a Gabor system.
J	Integer. Depth in a wavelet system.
tfr	Scalar. Time-to-frequency ratio.
lt	1×2 vector defining a non-separable lattice. Usually an optional parameter.
dim	Integer. Specifies the dimension to work along. Usually an optional parameter.

[1]For a lot of interesting comments on *why*, read the following Slashdot article: http://tech.slashdot.org/story/11/12/04/1742211/institutional-memory-and-reverse-smuggling

3.2 Common variables used only inside functions

b	Integer. Length of frequency shift.
ii	Integer. The first counting variable. DO NOT USE i only, as this is the imaginary unit.
jj	Integer. The second counting variable. DO NOT USE j only, as this is the imaginary unit. J is used for the Wavelet routines.
N	Integer. Number of time shifts.
p	Integer. The density of a Gabor frame written as a irreducible fraction is $\frac{p}{q}$. Similarly, the redundancy is $\frac{q}{p}$.
q	Integer. See p.
w	Integer. Counting variable that runs in the number of signals. See W.
W	Integer. Number of signals in a multi-components signal (normally the number of columns in a matrix).

3.3 Variables used only in the factorization routines

c	Integer. Equal to $\gcd(a, M)$. When this name is used, a coefficient is called $coef$.
d	Integer. Equal to $\gcd(b, N)$.
r	Integer. Counting variable. Runs in the numbers $0, \ldots, c-1$.
s	Integer. Counting variable. Runs in the numbers $0, \ldots, d-1$.

3.4 Iterative algorithms

Common inputs to iterative algorithms, *listed in order*:

tol	Tolerance. Algorithm is consider to have converges if the *relative* tolerance is less than *tol*.
maxit	Maximum number of iterations to do.

Common outputs from iterative algorithms, *listed in order:*

relres	Relative residual. Vector of length *iter* of the computed relative residuals.
iter	Number of iterations actually done, less than *maxit*.

4 Coding standards

4.1 Matlab specific coding standards

- A variable should never have the same name as an already existing function is Matlab. This makes the code easier to read and less prone to errors. This is in practice almost impossible to accomplish, as all short names are already taken by some function, but please try hard anyway!
- All function names in Matlab should be lowercase. This avoids a lot of confusion because some computer architectures respect upper/lower casing and others do not. Furthermore, function names are traditionally written in uppercase in Matlab documentation.

- It is not allowed to use underscores in function names. They are reserved for structural purposes, i.e. as in `demo_dgt` or `test_dgt`.
- *As much as possible*, functions are named after the function they perform, rather than the algorithm they use, or the person who invented it.
- No global variables. Global variables makes it harder to debug, and the code cannot be paralyzed.
- If a transform works on a matrix, it will per default work along the *columns*. This is a standard in Matlab (`fft` does this, among many other functions).

4.2 C specific coding standards

- Variables and function names are allowed to be both lower and upper case. This convention is called *camelCasing*, see http://en.wikipedia.org/wiki/CamelCase. The general rule is that the word starts with a lower case letter, and then the following words are starts will upper case letters. Use with care:
  - The benefit is that it makes long variable name easier to read.
  - The downside is that the user suddenly has to remember the casing of the variable name, instead of just the name itself.

5 Directory stucture

fourier	Fourier analysis. Transforms and functions that cannot be read/positionen at a 2D time/frequency scale, but only at a 1D frequency scale. The directory contains purely m-functions.
gabor	All functions that work with a linear frequency scale goes in this directory. If a function can be generalized to wavelets etc. then it must be prefixed by "GAB" to avoid confusion. The directory contains purely m-functions.
wavelets	Wavelet tools. Everything connected to wavelets goes here. The directory contains purely m-functions and one subdirectory *wfbtmanip*. The subdirectory contains purely m-functions for manipulating the general filterbank trees.
filterbank	General filterbank routines. A filterbank consists of a cell-array of filters and an array/scalar of hop factors. See sec. for a detailed description of The directory contains purely m-functions.
nonstatgab	Time-domain non-stationary Gabor transform routines. The directory contains purely m-functions.
frames	Routines for the object oriented frames framework. The directory contains purely m-functions.
operators	Routines for the object oriented operator framework. The directory contains purely m-functions.
comp	Computational functions intended to support the other directories. Functions in this directory never takes additional input parameters. An input parameter has one and only one meaning. Input parameters are not checked, this is the job of the calling function. The directory contains purely m-functions.
sigproc	Tools that work on output from all the transforms in the toolbox. Tools that are needed to support the examples. The directory contains purely m-functions.

signals	Signals intended for the demos. Each signal is loaded by calling an m-function without any arguments. The function must complain if called with arguments. The help file should contains an accurate description of the signal including length, sampling rate, where it was obtained from. The directory m-functions and binary data.
auditory	Auditory related functions.
demos	Demo scripts.
deprecated	Deprecated function. These function are kept around (forever), but issue a warning when ever used.
src	All standalone C files goes here.
mex	All Matlab MEX functions goes here.
oct	All Octave C++ functions goes here.

6 Sign and normalization conventions

6.1 Inner product and dft

We use the definition of inner products where they are conjugate linear in the second slot.

The discrete Fourier transform `dft`, is given by

$$c\left(k+1\right)=\frac{1}{\sqrt{L}}\sum_{l=0}^{L-1}f\left(l+1\right)e^{-2\pi ikl/L}.$$

This is a normalization of the `fft`, so that the `dft` preserved the l^2-norm of the input signal.

The dft can be seen as the inner product between the signal and the complex exponentials f_k

$$c\left(k+1\right)=\frac{1}{\sqrt{L}}\left\langle f,f_k\right\rangle, \qquad (1)$$

where $f_k\in\mathbb{C}^L$ is given by

$$f_k\left(l\right)=e^{2\pi ilk/L}. \qquad (2)$$

6.2 dgt

The `dgt` can be seen as inner product between the signal and a translated and modulated window, g:

$$\begin{aligned} c\left(m+1,n+1\right) &= \left\langle f,M_{mb}T_{na}g\right\rangle \\ &= \sum_{l=0}^{L-1}f(l+1)e^{-2\pi imbl/L}\overline{g\left(l-an+1\right)}. \end{aligned}$$

This transform is said to be *frequency-invariant* or *phase unlocked*, and it the default in LTFAT, and it is in many papers / books related to the community around the Numerical Harmonic Analysis Group (NuHAG) in Vienna, Austria. The name refers to the fact that if the signal is modulated by a complex exponential, this will amount to a circular shift in the output coefficients along the frequency axis, and nothing else.

The *time-invariant* or *phase locked* version corresponds to the following

$$c\left(m+1,n+1\right)=\sum_{l=0}^{L-1}f(l+1)e^{-2\pi imb(l-an)/L}\overline{g\left(l-an+1\right)}.$$

In this case, the name comes from the fact that a circular time shift of the input signal exhibits only in a circular shift of the coefficients along the time axis, and nothing else. The `phaselock` and `phaseunlock` routines allows changing the phase of the coefficients from one convention to another.

6.3 Discrete Symplectic Fourier transform, dsft

The `dsft` is given by

$$C(m+1,n+1) = \frac{1}{\sqrt{KL}} \sum_{l=0}^{L-1} \sum_{k=0}^{K-1} F(k+1,l+1)\, e^{2\pi i(kn/K - lm/L)}.$$

This makes it have the following property XXX

6.4 Spreading function and twisted convolution

Twisted convolution `h=tconv(f,g)` is defined by

$$h(m+1,n+1) = \sum_{l=0}^{L-1} \sum_{k=0}^{L-1} f(k+1,l+1)\, g(m-k+1,n-l+1)\, e^{-2\pi i(m-k)l/L}$$

The application of a spreading operator with symbol c applied to a signal f, `h=spreadop(c,f)` is given by

$$h(l+1) = \sum_{n=0}^{L-1} \sum_{m=0}^{L-1} c(m+1,n+1)\, e^{2\pi i l m/L} f(l-n+1)$$

This convention gives the following identities. If

- `h=tconv(c1,c2)`
- `f1=spreadop(spreadop(f,c2),c1)`

then `f2=spreadop(f,h)`.

6.5 Filterbanks

An analysis filterbank is defined by a convolution in each of M frequency channels

$$c_m(n+1) = \sum_{l=0}^{L-1} f(l+1)\, g_m(a_m n - l + 1), \tag{3}$$

for $m = 1,\ldots,M$, $N_m = L/a_m$ and $n = 0,\ldots,N_m - 1$. This means that each window is applied in the reverse order as compared to the DGT, making them incompatible. If a filterbank is used to compute a DGT, you must reverse the window. This inconsistency comes from the scientific communities originally defining the transforms. For symmetric windows, there is no confusion.

A synthesis filterbank is defined as

$$f(l+1) = \sum_{m=1}^{M} \sum_{n=0}^{N_m-1} c_m(n+1)\, g_m^*(a_m n - l + 1), \tag{4}$$

for $l = 0,\ldots,L-1$, $N_m = L/a_m$ where g_m^* denotes involution of g_m such that $g_m^* = \overline{g_m(-l)}$.

Fractional subsampling filterbanks The above equations require a_m and N_m to be integers for all m which means the input signal has to have length $L = k\text{lcm}(a_m)$ for some positive integer k, where lcm denotes least common multiple of all a_m. In case filterbank is band-limited with fractional subsampling, L is not restricted and N_m are defined directly and $a_m = L/N_m$ are rational numbers.

The default phase convention is *time-invariant* phase. In fact, (just to make it confusing) it is frequency-invariant when thinking about it as NSGT in the frequency domain so it is NOT equivalent to doing NSGT on FFT of the signal. This convention is used to be consistent with the non-bandlimited filterbanks phase convention and to preserve correct time position relations of the coefficients in the time domain.

The downside of this convention is that the subbands may themselves stay bandlimited which has to be accounted for when e.g. shifting the subbands around.

6.5.1 Interfaces for pfilt, filterbank and friends

Introduction and motivation

Several transforms in LTFAT are based on filterbanks with bandlimited windows: cqt, erblett and some wavelets. The problem with defining bandlimited filters is that the mask in frequency changes depending on the length of the signal. Therefore, a bandlimited filter must defined in on the torus $\mathbb{T}$ and then properly sampled as soon as the signal length is known.

The goal of this section is to discuss interfaces for the filterbank routines that allows them to both work with FIR and bandlimited filters.

As for the other windowed transforms in LTFAT, we must define two interfaces:

1. A frontend/user interface that it easy to work with, and where common usages can be represented efficiently i.e. dgt(f,'gauss',a,M) or fwt(f,'db4',6);
2. A backend interface that is as simple to parse as possible, but still able to represent everything in a accurate manner.

Backwards compatibility

The new frontend interface must be backwards compatible with the old one. There are no restrictions on the backend interface. The old frontend only defines a single interface, which is a cell array of vectors, where each vector is a zero-delay filter as used by DGT.

Two new filter generators

These two functions serves almost no purpose other than as place of documentation and as helper functions for *comp_fourierwindow*. The both return the filter expressed as a struct, as this requires no additional processing except passing L to an anonymous function.

firfilter – Construct an FIR filter

- First input is one of the names from *firwin*.
- Second input is the width of the filter, as in *firwin*.
- Third input is the delay of the filter, possibly the word *'causal'*. Default value is 0.
- The rest of the parameters are passed to *firwin*.

blfilter – Construct a bandlimited filter

- First input is one of the names from *firwin*.
- Second input is the support of the filter in the frequency domain measured in normalized frequencies.
- Third input is the centre of the filter in the frequency domain measured in normalized frequencies.
- It is possible to normalize the filter as an optional parameter, but is is normalized in the *frequency domain*.

pfilt frontend interface

The interface is parsed by *comp_fourierwindow*. *pfillt* calls *comp_fourierwindow* with the filter definition and the length of the signal L.

Input is a vector

In the input is a vector, it is considered to be a zero-delay FIR window compatible with DGT. A quick way of defining a classical causal FIR filter when having a numerical imp. resp. h is to use struct as follows: *struct('h',h,'offset',0)*.

Input is a string

The following keywords are defined

gauss	Normalized Gaussian (Note: For backwards compatibility, it can be removed)
psech	Normalized sech (Note: For backwards compatibility, it can be removed)

Input is a struct

The struct can have the following fields:
If the filter is an FIR filter, these two fields are mandatory:

h	Vector. The impulse response of the filter
offset	offset of the filter. The total delay is the delay of h when used as a traditional causal filter plus $offset$.

h and $offset$ are defined by the following piece of code:

```
pfilt([1;zeros(L-1,1),g])==circshift(postpad(g.h,L),g.offset);
```

so h and $offset$ decribes how to obtain an impulse reponse of length L.

If the filter is a bandlimited filter, these two fields are mandatory:

H	An anonymous function taking L as input and producing a single vector, which is the frequency response of the filter
foff	An anonymous function taking L as input and producing a single number, which is the frequency offset of the filter measured in samples.

H and $foff$ are defined by the following piece of code:

```
fft(pfilt([1;zeros(L-1,1),g]))==circshift(postpad(g.H,L),g.foff);
```

so H and $foff$ decribes how to obtain the frequency response of length L. Whenever H amd $foff$ get evaluated with a given L it's value should be stored in an additional field $.L$.

Input is a cell array

- If the first value is one of the names from *firfilter,* the whole cell array is passed as parameters to *firfilter.*

filterbank frontend interface

The interface is parsed by *filterbankwin. filterbank* and friends calls *filterbankwin* with the filter definition and the length of the signal L. *filterbankwin* can be called without specifying L if all filters are FIR.

Two basic formats:

- A cell array of stuff that is legal with pfilt
- A cell array where the first element is a string that denotes how to parse the rest. The string can be

'dual'	Dual filterbank of what follows
'realdual'	Dual filterbank for real-valued signals of what follows
'tight'	Tight filterbank of what follows
'tightdual'	Tight filterbank for real-valued signals of what follows

filterbank backend interface

The backend interface in a cell array of filters. Each element in the cell array is a valid struct for the pfilt backend.

6.6 Non-stationary Gabor systems

The non-stationary Gabor transform in the time-domain was defined using a time-invariant phase, making it incompatible with the Gabor transform which by default uses frequency-invariant phase. This was done in order to make a simple formulation of the basic window-and-transform algorithm.

7 The frames object oriented framework

7.1 Rationale

LTFAT contains many different linear transforms that behave similarly. An object oriented framework that captures these similarities is natural extension of LTFAT. The framework consists of an encapsulation of the basic methods for analysis and synthesis, and a set of advanced algorithms that use the framework, to provide transform-agnostic algorithms.

7.2 Design considerations

Two different models have been considered:

1. A frame object contains a single frame, which can be used for either analysis or synthesis

   Pros:

   (a) Very general, analysis and synthesis from different frames can be combined

   (b) Very general construction of dual and tight windows, as they are constructed by a method: F2=frameweirddual(F1);

   Cons:

   (a) Dual window constructed by a method: F2=framedual(F1), which in many cases means an additional setup line.

   (b) The user must track two frames when doing analysis and synthesis:
   c=frana(Fa,f);
   f=frsyn(Fs,c);

2. A frame object contains *two* frames, where one is used for analysis and the other for synthesis

   Pros:

   (a) Simple syntax for working with standard duals: F=frame('dgt','gabor','dual',a,M);

   (b) Simple analysis and synthesis using the same definition:
   c=frana(F,f);
   f=frsyn(F,c);

   Cons:

   (a) Clumsy setup when only one frame is needed: F=frame('dgt','none','gauss',a,M);

   (b) Impossible to combine analysis and synthesis from different frame types.

   (c) Difficult to specify non-standard dual in the new frame definition

Comments to pros and cons:

(1a) and (2b) Due to different coefficient layout, it will often be impossible to combine analysis and synthesis from different frames, and it will require spe

(1b) and (2c) Non-canonical dual and tight methods are difficult express so they work for all frames, meaning that it does not make sense to put them into the framework. This makes this issue less relevant

8 Wavelets

Describe the following:

- Input signal is assumed to be already a projection onto the scale space 0.
- Boundary extension handling.
- How to add new wavelet filters.
- Building general filterbank trees.
- Rationale why there is no continuous wavelet transform in LTFAT yet.

9 Back-end library, MEX and OCT interfaces

The back-end library depends on the FFTW library `http://www.fftw.org` and on some BLAS and LAPACK routines. It is written in C (according to C99 standard[2]). Code formatting

Pros for this choice:

- The C code does not suffer from incompatibilities when linked with another C/C++ code compiled with different compiler as C++ code does. This is due to the fact that C++ function names mangling is not standardized.
- C99 offers transparent complex numbers operations. In the C99 complex number format the real and imaginary numbers are interleaved in memory. This is also how FFTW library and GNU Octave handle complex numbers. Matlab, however, internally stores complex numbers as separate arrays in memory and therefore a conversion is needed.

  There are several ways how to create, cast and work with complex number arrays in C99 (double data type here):

  – The proper way of allocating array of complex numbers:

    ```
    ltfatInt L = 100;
    // Allocation on the stack (this is C99 feature)
    double _Complex f1[L];
    // Memory aligned allocation on the heap
    double _Complex *f2 = ltfat_malloc(L*sizeof*f2)
    ```

  – Accessing the data using two pointers:

    ```
    double *f3re = (double*)f2;
    double *f3im = f3re+1;
    // Advance pointers by 2
    ```

  – Recasting to array of length 2 arrays:

    ```
    double (*f4)[2] = (double(*)[2]) f2;
    // double *f4[2] is length 2 array of pointers to double

    // 10th element
    double f10re = f4[10][0];
    double f10im = f4[10][1];
    ```

- C99 complex numbers should be binary compatible with C++ `complex` class. Correct cast is

  ```
  complex<double> * g;
  double _Complex* h = reinterpret_cast<double _Complex *>(g);
  ```

Cons:

- C99 is not (and will not ever be) supported by the Microsoft compiler.
- Templated functions are not supported in C99.

The following guidelines should be followed in order to make the code consistent:

[2]See `http://www.open-std.org/jtc1/sc22/WG14/www/docs/n1256.pdf`.

- There is a `ltfatInt` integer data type which should be used for array lengths and array indexes. `ltfatInt` is alias for `ptrdiff_t` (or `int` if `LTFAT_COMPAT32` compiler flag is used).
- In C language, the explicit cast from void* returned by malloc (mxGetData, and others) do not have to be (and should not be) used. Compare

  ```
  double* f = ltfat_malloc(L*sizeof*f);
  ```

  and

  ```
  double* f = (double*)ltfat_malloc(L*sizeof(double));
  ```

  Note `sizeof` is not a function but rather a unary operator so f is actually not dereferenced which would result in seg. fault.
- Do not assume the output arrays passed as function arguments to be safely zeroed.
- Creating FFTW plan using opt. flag other than `FFTW_ESTIMATE` rewrites data in the array used for planning. Handle your data accordingly.
- Do not use bool data type in C function headers. The C++ and C definition of bool is not compatible. Use int instead.

There are two compilation targets, DOUBLE (producing ltfat.dll or libltfat.a) and SINGLE (producing ltfatf.dll or libltfatf.a) and in addition, some of the *.c files are processed twice in a single target (real and complex version). In order to reduce the code repetition and to allow template-like functions, the macro set representing data types, function names and constants were introduced:

Data types:

- LTFAT_REAL – real data type, `double` or `float`
- LTFAT_COMPLEX – complex data type, `double _Complex` or `float _Complex`
- LTFAT_TYPE – real or complex data type, LTFAT_REAL or LTFAT_COMPLEX

Function name decorators:

- LTFAT_NAME(name) – adds `_d`, `_s`, `_cd` or `_cs` suffix to name.
- LTFAT_NAME_REAL(name) – adds `_d` or `_s` suffix to name.
- LTFAT_NAME_COMPLEX(name) – adds `_cd` or `_cs` suffix to name.
- LTFAT_FFTW(name) – adds `fftw_` or `fftwf_` prefix to name.
- LTFAT_COMPLEXH(name) – adds noting or `f` suffix to name.

Other (MEX related):

- LTFAT_MX_COMPLEXITY – `mxREAL` or `mxCOMPLEX`
- LTFAT_MX_CLASSID – `mxDOUBLE_CLASS` or `mxSINGLE_CLASS`

The macro definitions are controlled by target specific, mutually exclusive compiler directives `LTFAT_DOUBLE` (for the DOUBLE target), `LTFAT_SINGLE` (for the SINGLE target) and by an additional macro `LTFAT_COMPLEXTYPE` (see `ltfat_types.h`), which is used to compile the real/complex transparent functions. Tab. 1 summarizes macro expansions.

Cons:

- The real and complex versions of a function in the single comp. target are in the same translational unit.

target macro	LTFAT_DOUBLE		LTFAT_SINGLE	
Is LTFAT_COMPLEXTYPE set?	false	true	false	true
LTFAT_TYPE	`double`	`double _Complex`	`float`	`float _Complex`
LTFAT_REAL	`double`	`double`	`float`	`float`
LTFAT_COMPLEX	`double _Complex`	`double _Complex`	`float _Complex`	`float _Complex`
LTFAT_NAME(name)	name_d	name_cd	name_s	name_cs
LTFAT_NAME_REAL(name)	name_d	name_d	name_s	name_s
LTFAT_NAME_COMPLEX(name)	name_cd	name_cd	name_cs	name_cs
LTFAT_FFTW(name)	fftw_name	fftw_name	fftwf_name	fftwf_name
LTFAT_COMPLEXH(name)	name	name	namef	namef
LTFAT_MX_COMPLEXITY	mxREAL	mxCOMPLEX	mxREAL	mxCOMPLEX
LTFAT_MX_CLASSID	mxDOUBLE_CLASS	mxDOUBLE_CLASS	mxSINGLE_CLASS	mxSINGLE_CLASS

Table 1: Macro expansions in LTFAT backend.

9.1 MEX related remarks

- The MEX file should be as simple as possible. Ideally, it should just obtain pointers and reformat other inputs to be passed further to the core functions from the backend lib.
- Use `ltfat_mex_template_helper.h` to reduce the code repetition when handling real/complex single/double input data. Most notable operation is the change of the complex data memory format from split (two separate arrays in memory) used in Matlab to interleaved (real and imaginary parts are in a single array interleaved) used by the backend lib.

The same goes for the OCT interfaces. Internally, Octave uses interleaved memory format. Other remarks:

- MEX API defines its own integer types for array lengths: mwSize, mwIndex, mwSignedIndex, all being `int`, if the `MX_COMPAT_32` compiler flag is set. Otherwise, they are size_t, size_t and ptrdiff_t respectively.
- OCT API defines octave_idx_type used for the array lengths which is `int`.

9.2 Handling dependencies

LTFAT backend lib depends on the following libs:

- BLAS `http://www.netlib.org/blas/`
- LAPACK `http://www.netlib.org/lapack/`
- FFTW `http://www.fftw.org/`

The block-stream processing framework depends on

- PORTAUDIO `http://www.portaudio.com/`
- (Octave) Java package. Java is included in the core since 3.8.

Possible problems with libraries There was a problem with BLAS and LAPACK libraries when changing from 32bit to 64bit compilation. Both require array lengths to be passed by pointers so it is important to use correct data type. This was solved by requiring the lengths to be of `ptrdiff_t` data type in LTFAT's BLAS/LAPACK wrapper functions.

FFTW internally works with 64bit array lengths, but the public API works with `int`s, which are usually 32bit even on 64bit systems. LTFAT backend can use either 32bit or 64bit (signed) integers for lengths depending on `ltfatInt`. This issue was solved by an explicit downcast to `int` prior calling any of the FFTW routines which require defining lengths [3].

Linking libraries The way how to install and link dependencies differ among operating systems, Octave and Matlab and apparently even among different versions of Matlab. Matlab usually already contains all the libs in it's `bin/ARCH` directory, which is convenient on one hand, but it makes it hard to force using different version of a library. One option is to use a symbolic link to a custom version of a custom lib.

[3] FFTW however offers some 64bit guru interface for 64bit length array. See `http://www.fftw.org/doc/64_002dbit-Guru-Interface.html`, but nobody will probably need to do fft of arrays of lengths $> 2^{31}$.

Matlab, Windows All dependencies can be found in `[MATLAB]/bin/win64` as dlls. When using MinGW, we can link them directly without creating import libs [4].

This way, only `libmwblas` and `libmwlapack` are linked. FFTW lib does not contain all the routines we need; namely memory allocation routines and real-to-real transforms are missing. Portaudio lib shipped with Matlab does not perform well (at least on my R2011b). Therefore, a manual installation of FFTW and portaudio libs is required. Simply copy the dlls to the directories next to the MEX files. They will be found by the build script at compile time and linked with at runtime. Here we rely on the Windows property that dlls contained in the same directory as the executable/dll are linked preferably at runtime.

Octave, Windows Using the MinGW octave build, all libs except for portaudio are contained in the installation. Manual installation of portaudio is required as in the previous case. The Java package does not work correctly in the current Octave binary for Windows (3.6.4).

Matlab, Linux In most cases, `bin/ARCH` already contains all dependencies in an usable shape. Just some old versions of Matlab (checked on R2009b) do not yet contain portaudio lib. Matlab sets `LD_LIBRARY_PATH` to contain `bin/ARCH` after start, so it become somewhat difficult to use system-wide or custom libs since `LD_LIBRARY_PATH` is searched first at runtime.

Octave, Linux As Octave uses system-wide libs, all dependencies can be installed using a distro specific package manager.

Octave package, Linux The installation procedure is automated using autotools.

Matlab, Mac OS X The situation is similar to Linux, but it seems that clang is the main compiler nowadays (at least packed with XCode) and it does not understand some of the gcc specific flags. This is the reason why there are separate makefiles for Mac and Linux.

Octave, Mac OS X In Octave 3.8 all dependencies except Portaudio are part of core Octave. For other versions of Octave higher than 3.4, the Java package should be installed. The Java package depends on the Java Development Kit which could be downloaded through the Apple Developer Downloads.

To compile the bank-end libraries it is recommended to have the Xcode Command Line Tools installed since this contains all necessary compilers.

[4]Symbols dll exports can be browsed by the Dependency Walker `http://www.dependencywalker.com/`.