

## LTFAT Reference manual

Peter L. Søndergaard

Peter Balazs

Monika Dörfler

Nicki Holighaus

Florent Jaillet

Nathanael Perraudin

Zdeněk Průša

Kai Siedenburg

Bruno Torrésani

Christoph Wiesmeyr

Jordy van Velthoven



# Contents

<b>1 LTFAT - Base routines</b>	<b>11</b>
1.1 Basic routines . . . . .	11
1.1.1 ltfatstart . . . . .	11
1.1.2 ltfatstop . . . . .	11
1.1.3 ltfathelp . . . . .	11
1.1.4 ltfatmex . . . . .	12
1.1.5 ltfatbasepath . . . . .	12
1.1.6 isoctave . . . . .	12
1.2 Parameter handling . . . . .	13
1.2.1 ltfatarghelper . . . . .	13
1.2.2 ltfatgetdefaults . . . . .	14
1.2.3 ltfatsetdefaults . . . . .	14
1.2.4 scalardistribute . . . . .	14
1.3 Graphical user interfaces . . . . .	15
1.3.1 mulaclab . . . . .	15
<b>2 LTFAT - Gabor analysis</b>	<b>17</b>
2.1 Basic Time/Frequency analysis . . . . .	17
2.1.1 tconv . . . . .	17
2.1.2 dsft . . . . .	17
2.1.3 zak . . . . .	18
2.1.4 izak . . . . .	19
2.1.5 col2diag . . . . .	19
2.1.6 s0norm . . . . .	19
2.2 Gabor systems . . . . .	20
2.2.1 dgt . . . . .	20
2.2.2 idgt . . . . .	22
2.2.3 isgram . . . . .	24
2.2.4 isgramreal . . . . .	26
2.2.5 dgt2 . . . . .	28
2.2.6 idgt2 . . . . .	29
2.2.7 dgtral . . . . .	29
2.2.8 idgtral . . . . .	31
2.2.9 gabwin . . . . .	33
2.2.10 dgtrlength . . . . .	34
2.3 Wilson bases and WMDCT . . . . .	34
2.3.1 dwilt . . . . .	34
2.3.2 idwilt . . . . .	36
2.3.3 dwilt2 . . . . .	36
2.3.4 idwilt2 . . . . .	38
2.3.5 wmdct . . . . .	38
2.3.6 iwm dct . . . . .	40
2.3.7 wmdct2 . . . . .	41
2.3.8 iwm dct2 . . . . .	41

2.3.9	wil2rect . . . . .	42
2.3.10	rect2wil . . . . .	42
2.3.11	wilwin . . . . .	42
2.3.12	dwiltlength . . . . .	43
2.4	Reconstructing windows . . . . .	44
2.4.1	gabdual . . . . .	44
2.4.2	gabtight . . . . .	45
2.4.3	gabfirdual . . . . .	47
2.4.4	gaboptdual . . . . .	49
2.4.5	gabfirtight . . . . .	51
2.4.6	gabopttight . . . . .	52
2.4.7	gabconvexopt . . . . .	54
2.4.8	gabprojdual . . . . .	56
2.4.9	gabmixdual . . . . .	56
2.4.10	wilorth . . . . .	57
2.4.11	wildual . . . . .	57
2.5	Conditions numbers . . . . .	58
2.5.1	gabframebounds . . . . .	58
2.5.2	gabrieszbounds . . . . .	59
2.5.3	wilbounds . . . . .	59
2.5.4	gabdualnorm . . . . .	60
2.5.5	gabframediag . . . . .	61
2.5.6	wilframediag . . . . .	61
2.6	Phase gradient methods and reassignment . . . . .	62
2.6.1	gabphasegrad . . . . .	62
2.6.2	gabreassign . . . . .	62
2.7	Phase conversions . . . . .	63
2.7.1	phaselock . . . . .	63
2.7.2	phaseunlock . . . . .	64
2.7.3	sympulse . . . . .	64
2.8	Support for non-separable lattices . . . . .	64
2.8.1	matrix2lattice . . . . .	64
2.8.2	lattice . . . . .	66
2.8.3	shearfind . . . . .	66
2.8.4	noshearlength . . . . .	67
2.9	Plots . . . . .	67
2.9.1	tfplot . . . . .	67
2.9.2	plotdgt . . . . .	68
2.9.3	plotdgtreal . . . . .	68
2.9.4	plotdwilt . . . . .	69
2.9.5	plotwmdct . . . . .	69
2.9.6	sgram . . . . .	69
2.9.7	gabimagepars . . . . .	72
2.9.8	resgram . . . . .	72
2.9.9	instfreqplot . . . . .	74
2.9.10	phaseplot . . . . .	75
<b>3</b>	<b>LTFAT - Basic Fourier and DCT analysis.</b>	<b>79</b>
3.1	Support routines . . . . .	79
3.1.1	fftindex . . . . .	79
3.1.2	modcent . . . . .	79
3.1.3	floor23 . . . . .	79
3.1.4	floor235 . . . . .	80
3.1.5	ceil23 . . . . .	81
3.1.6	ceil235 . . . . .	81
3.1.7	nextfastfft . . . . .	82

3.2	Basic Fourier analysis . . . . .	82
3.2.1	dft . . . . .	82
3.2.2	idft . . . . .	83
3.2.3	fftreal . . . . .	83
3.2.4	ifftreal . . . . .	83
3.2.5	gga . . . . .	83
3.2.6	chirpz . . . . .	85
3.2.7	plotfft . . . . .	87
3.2.8	plotfftreal . . . . .	88
3.3	Simple operations on periodic functions . . . . .	89
3.3.1	involute . . . . .	89
3.3.2	peven . . . . .	89
3.3.3	podd . . . . .	90
3.3.4	peconv . . . . .	90
3.3.5	convolve . . . . .	90
3.3.6	pxcorr . . . . .	91
3.3.7	isevenfunction . . . . .	91
3.3.8	middlepad . . . . .	91
3.4	Functions . . . . .	91
3.4.1	expwave . . . . .	91
3.4.2	pchirp . . . . .	92
3.4.3	shah . . . . .	93
3.4.4	pheaviside . . . . .	94
3.4.5	prect . . . . .	94
3.4.6	psinc . . . . .	95
3.5	Window functions . . . . .	96
3.5.1	pgauss . . . . .	96
3.5.2	psech . . . . .	99
3.5.3	pbspline . . . . .	101
3.5.4	firwin . . . . .	102
3.5.5	firkaiser . . . . .	105
3.5.6	fir2long . . . . .	105
3.5.7	long2fir . . . . .	105
3.6	Filtering . . . . .	105
3.6.1	firfilter . . . . .	105
3.6.2	blfilter . . . . .	106
3.6.3	warpedblfilter . . . . .	107
3.6.4	pfilt . . . . .	108
3.6.5	magresp . . . . .	109
3.6.6	transferfunction . . . . .	111
3.6.7	pgrpdelay . . . . .	111
3.7	Hermite functions and fractional Fourier transforms . . . . .	112
3.7.1	pherm . . . . .	112
3.7.2	hermbasis . . . . .	113
3.7.3	dfracft . . . . .	114
3.7.4	ffracft . . . . .	115
3.8	Approximation of continuous functions . . . . .	115
3.8.1	fftresample . . . . .	115
3.8.2	dctresample . . . . .	116
3.8.3	pderiv . . . . .	116
3.9	Cosine and Sine transforms . . . . .	116
3.9.1	dcti . . . . .	116
3.9.2	dctii . . . . .	118
3.9.3	dctiii . . . . .	119
3.9.4	dctiv . . . . .	120
3.9.5	dsti . . . . .	120

3.9.6	dstii . . . . .	121
3.9.7	dstiii . . . . .	122
3.9.8	dstiv . . . . .	123
<b>4</b>	<b>LTFAT - Wavelets</b>	<b>125</b>
4.1	Basic analysis/synthesis . . . . .	125
4.1.1	fwt . . . . .	125
4.1.2	ifwt . . . . .	127
4.1.3	fwt2 . . . . .	128
4.1.4	ifwt2 . . . . .	130
4.1.5	ufwt . . . . .	130
4.1.6	iufwt . . . . .	132
4.1.7	fwtlength . . . . .	133
4.1.8	fwtclength . . . . .	133
4.2	Advanced analysis/synthesis . . . . .	134
4.2.1	wfbt . . . . .	134
4.2.2	iwfbt . . . . .	135
4.2.3	uwfbt . . . . .	136
4.2.4	iuwfbt . . . . .	137
4.2.5	wpfbt . . . . .	138
4.2.6	iwpfbt . . . . .	139
4.2.7	uwpfbt . . . . .	140
4.2.8	iuwpfbt . . . . .	142
4.2.9	wpbest . . . . .	143
4.2.10	wfbtlength . . . . .	145
4.2.11	wfbtclength . . . . .	146
4.2.12	wpfbtclength . . . . .	146
4.3	Dual-tree complex wavelet transform . . . . .	146
4.3.1	dtwfb . . . . .	146
4.3.2	idtwfb . . . . .	150
4.3.3	dtwfbleal . . . . .	151
4.3.4	idtwfbleal . . . . .	154
4.4	Wavelet Filterbank trees manipulation . . . . .	155
4.4.1	wfbtinit . . . . .	155
4.4.2	dtwfbinit . . . . .	156
4.4.3	wfbtput . . . . .	156
4.4.4	wfbtremove . . . . .	158
4.4.5	wfbt2filterbank . . . . .	159
4.4.6	wpfbt2filterbank . . . . .	160
4.4.7	dtwfb2filterbank . . . . .	162
4.4.8	fwtinit . . . . .	163
4.5	Frame properties of wavelet filterbanks: . . . . .	164
4.5.1	wfbtbounds . . . . .	164
4.5.2	wpfbtbounds . . . . .	165
4.5.3	uwfbtbounds . . . . .	165
4.5.4	uwpfbtbounds . . . . .	165
4.6	Plots . . . . .	166
4.6.1	plotwavelets . . . . .	166
4.6.2	wfiltinfo . . . . .	166
4.6.3	wfiltddtinfo . . . . .	167
4.7	Auxiliary . . . . .	168
4.7.1	wavfun . . . . .	168
4.7.2	wavcell2pack . . . . .	169
4.7.3	wavpack2cell . . . . .	170
4.8	Wavelet Filters defined in the time-domain . . . . .	170
4.8.1	wfilt_algmband . . . . .	170

4.8.2	wfilt_cmband . . . . .	171
4.8.3	wfilt_coif . . . . .	173
4.8.4	wfilt_db . . . . .	174
4.8.5	wfilt_dden . . . . .	175
4.8.6	wfilt_dgrid . . . . .	176
4.8.7	wfilt_hden . . . . .	176
4.8.8	wfilt_lemarie . . . . .	177
4.8.9	wfilt_matlabwrapper . . . . .	178
4.8.10	wfilt_mband . . . . .	178
4.8.11	wfilt_remez . . . . .	179
4.8.12	wfilt_symds . . . . .	180
4.8.13	wfilt_spline . . . . .	181
4.8.14	wfilt_sym . . . . .	182
4.8.15	wfilt_symdden . . . . .	183
4.8.16	wfilt_symorth . . . . .	184
4.8.17	wfilt_symtight . . . . .	185
4.8.18	wfilt_qshifta . . . . .	185
4.8.19	wfilt_qshiftb . . . . .	186
4.8.20	wfilt_oddevena . . . . .	187
4.8.21	wfilt_oddevenb . . . . .	188
4.9	Dual-Tree Filters . . . . .	189
4.9.1	wfiltdt_qshift . . . . .	189
4.9.2	wfiltdt_optsym . . . . .	190
4.9.3	wfiltdt_oddeven . . . . .	190
4.9.4	wfiltdt_dden . . . . .	191
<b>5</b>	<b>LTFAT - Filterbanks</b>	<b>193</b>
5.1	Transforms and basic routines . . . . .	193
5.1.1	filterbank . . . . .	193
5.1.2	ufilterbank . . . . .	193
5.1.3	ifilterbank . . . . .	194
5.1.4	filterbankwin . . . . .	194
5.1.5	filterbanklength . . . . .	195
5.1.6	filterbanklengthcoef . . . . .	195
5.2	Auditory inspired filterbanks . . . . .	195
5.2.1	cqt . . . . .	195
5.2.2	icqt . . . . .	197
5.2.3	erblett . . . . .	197
5.2.4	ierblett . . . . .	199
5.2.5	insdgb . . . . .	199
5.3	Filter generators . . . . .	200
5.3.1	cqtfilters . . . . .	200
5.3.2	erbfilters . . . . .	201
5.4	Window construction and bounds . . . . .	204
5.4.1	filterbankdual . . . . .	204
5.4.2	filterbanktight . . . . .	205
5.4.3	filterbankrealdual . . . . .	205
5.4.4	filterbankrealtight . . . . .	205
5.4.5	filterbankbounds . . . . .	206
5.4.6	filterbankrealbounds . . . . .	206
5.4.7	filterbankresponse . . . . .	206
5.5	Auxiliary . . . . .	207
5.5.1	filterbankfreqz . . . . .	207
5.5.2	nonu2ufilterbank . . . . .	207
5.5.3	u2nonucfmt . . . . .	207
5.5.4	nonu2ucfmt . . . . .	208

5.6	Plots . . . . .	208
5.6.1	plotfilterbank . . . . .	208
<b>6</b>	<b>LTFAT - Non-stationary Gabor systems</b>	<b>211</b>
6.1	Transforms . . . . .	211
6.1.1	nsdgt . . . . .	211
6.1.2	unsdgt . . . . .	212
6.1.3	insdgt . . . . .	213
6.1.4	nsdgtreal . . . . .	213
6.1.5	unsdgtreal . . . . .	215
6.1.6	insdgtreal . . . . .	216
6.2	Window construction and bounds . . . . .	216
6.2.1	nsgab dual . . . . .	216
6.2.2	nsgab tight . . . . .	217
6.2.3	nsgab framebounds . . . . .	217
6.2.4	nsgab frame diag . . . . .	218
6.3	Plots . . . . .	218
6.3.1	plotnsdgt . . . . .	218
6.3.2	plotnsdg treal . . . . .	219
<b>7</b>	<b>LTFAT - Frames</b>	<b>221</b>
7.1	Creation of a frame object . . . . .	221
7.1.1	frame . . . . .	221
7.1.2	framepair . . . . .	223
7.1.3	frame dual . . . . .	223
7.1.4	frame tight . . . . .	224
7.1.5	frame accel . . . . .	224
7.2	Linear operators . . . . .	225
7.2.1	frana . . . . .	225
7.2.2	frsyn . . . . .	226
7.2.3	frsynmatrix . . . . .	226
7.2.4	frame diag . . . . .	227
7.2.5	frana iter . . . . .	227
7.2.6	frsyn iter . . . . .	229
7.3	Visualization . . . . .	230
7.3.1	plotframe . . . . .	230
7.3.2	frame gram . . . . .	231
7.4	Information about a frame . . . . .	231
7.4.1	frame bounds . . . . .	231
7.4.2	frame red . . . . .	232
7.4.3	frame length . . . . .	233
7.4.4	frame length coef . . . . .	233
7.4.5	frame length . . . . .	233
7.5	Coefficients conversions . . . . .	233
7.5.1	frame coef 2 native . . . . .	233
7.5.2	frame native 2 coef . . . . .	233
7.5.3	frame coef 2 tf . . . . .	234
7.5.4	frame tf 2 coef . . . . .	234
7.5.5	frame coef 2 tf plot . . . . .	234
7.6	Non-linear analysis and synthesis . . . . .	234
7.6.1	fran abp . . . . .	234
7.6.2	fran alasso . . . . .	238
7.6.3	fran group lasso . . . . .	242
7.6.4	frsyn abp . . . . .	243

<b>8 LTFAT - Signal processing tools</b>	<b>245</b>
8.1 General . . . . .	245
8.1.1 rms . . . . .	245
8.1.2 normalize . . . . .	245
8.1.3 gaindb . . . . .	246
8.1.4 crestfactor . . . . .	246
8.1.5 uquant . . . . .	246
8.2 Ramping . . . . .	247
8.2.1 rampup . . . . .	247
8.2.2 rampdown . . . . .	247
8.2.3 rampsignal . . . . .	247
8.3 Thresholding methods . . . . .	248
8.3.1 thresh . . . . .	248
8.3.2 largestr . . . . .	249
8.3.3 largestn . . . . .	249
8.3.4 dynlimit . . . . .	250
8.3.5 groupthresh . . . . .	250
8.4 Image processing . . . . .	251
8.4.1 rgb2jpeg . . . . .	251
8.4.2 jpeg2rgb . . . . .	252
8.5 Tools for OFDM . . . . .	252
8.5.1 qam4 . . . . .	252
8.5.2 iqam4 . . . . .	253
<b>9 LTFAT - Simple auditory processing</b>	<b>255</b>
9.1 Plots . . . . .	255
9.1.1 semiaudplot . . . . .	255
9.2 Auditory scales . . . . .	255
9.2.1 audtufreq . . . . .	255
9.2.2 freqtoaud . . . . .	255
9.2.3 audspace . . . . .	256
9.2.4 audspacebw . . . . .	256
9.2.5 erbtotfreq . . . . .	257
9.2.6 freqtoerb . . . . .	257
9.2.7 erbspace . . . . .	258
9.2.8 erbspacebw . . . . .	258
9.2.9 audfiltbw . . . . .	258
9.3 Range compression . . . . .	258
9.3.1 rangecompress . . . . .	258
9.3.2 rangeexpand . . . . .	259
9.4 Auditory filters . . . . .	259
9.4.1 gammatonefir . . . . .	259
<b>10 LTFAT - Signals</b>	<b>261</b>
10.1 Signal generators . . . . .	261
10.1.1 ctestfun . . . . .	261
10.1.2 noise . . . . .	261
10.1.3 pinknoise . . . . .	263
10.1.4 expchirp . . . . .	263
10.2 Sound signals. . . . .	264
10.2.1 bat . . . . .	264
10.2.2 batmask . . . . .	265
10.2.3 greasy . . . . .	265
10.2.4 cocktailparty . . . . .	267
10.2.5 gspi . . . . .	267
10.2.6 linus . . . . .	267

10.2.7	ltfatlogo . . . . .	268
10.2.8	otoclick . . . . .	268
10.2.9	traindoppler . . . . .	268
10.3	Images. . . . .	269
10.3.1	cameraman . . . . .	269
10.3.2	lichtenstein . . . . .	269
10.3.3	ltfattext . . . . .	270
<b>11</b>	<b>LTFAT - Demos</b>	<b>271</b>
11.1	Basic demos . . . . .	271
11.1.1	demo_dgt . . . . .	271
11.1.2	demo_gabfir . . . . .	273
11.1.3	demo_wavelets . . . . .	276
11.2	Compression . . . . .	277
11.2.1	demo_imagecompression . . . . .	277
11.2.2	demo_audiocompression . . . . .	278
11.3	Denoising . . . . .	279
11.3.1	demo_audiodenoise . . . . .	279
11.4	Applications . . . . .	279
11.4.1	demo_ofdm . . . . .	279
11.4.2	demo_audioshrink . . . . .	281
11.4.3	demo_gabmulappr . . . . .	281
11.4.4	demo_bpframemul . . . . .	282
11.4.5	demo_frsynabs . . . . .	283
11.5	Aspects of particular functions . . . . .	286
11.5.1	demo_nsdt . . . . .	286
11.5.2	demo_pgauss . . . . .	286
11.5.3	demo_pbspline . . . . .	288
11.5.4	demo_gabmixdual . . . . .	289
11.5.5	demo_framemul . . . . .	290
11.5.6	demo_phaseplot . . . . .	290
11.5.7	demo_phaseret . . . . .	291
11.5.8	demo_nextfastfft . . . . .	292
11.5.9	demo_filterbanks . . . . .	293
11.6	Auditory scales and filters . . . . .	295
11.6.1	demo_audscales . . . . .	295
11.6.2	demo_auditoryfilterbank . . . . .	296
11.6.3	demo_wfbt . . . . .	298
11.7	Block-processing demos . . . . .	299
11.7.1	demo_blockproc_basicloop . . . . .	299
11.7.2	demo_blockproc_paramequalizer . . . . .	301
11.7.3	demo_blockproc_denoising . . . . .	301
11.7.4	demo_blockproc_slidingsgram . . . . .	302
11.7.5	demo_blockproc_slidingcqt . . . . .	302
11.7.6	demo_blockproc_slidingerblets . . . . .	303
11.7.7	demo_blockproc_dgtequalizer . . . . .	303
11.7.8	demo_blockproc_effects . . . . .	304

# Chapter 1

## LTFAT - Base routines

### 1.1 Basic routines

#### 1.1.1 LTFATSTART - Start the LTFAT toolbox

##### Usage

```
ltfatstart;
```

##### Description

ltfatstart starts the LTFAT toolbox. This command must be run before using any of the functions in the toolbox.

To configure default options for functions, you can use the ltfatsetdefaults function in your startup script. A typical startup file could look like:

```
addpath('/path/to/my/work/ltfat');
ltfatstart;
ltfatsetdefaults('sgram','nocolorbar');
```

This will add the main LTFAT directory to you path, start the toolbox, and configure sgram to not display the colorbar.

The function walks the directory tree and adds a subdirectory to path if the directory contain a [subdirectory, init.m] script setting a status variable to some value greater than 0. status==1 identifies a toolbox module any other value just a directory to be added to path.

#### 1.1.2 LTFATSTOP - Stops the LTFAT toolbox

##### Usage

```
ltfatstop;
```

##### Description

ltfatstop removes all LTFAT subdirectories from the path.

#### 1.1.3 LTFATHHELP - Help on the LTFAT toolbox

##### Usage

```
ltfathelp;
v=ltfathelp('version');
mlist=ltfathelp('modules');
```

**Description**

`ltfathelp` displays some general help on the LTFAT toolbox.

`ltfathelp('version')` returns the version number.

`ltfathelp('modules')` returns a cell array of installed modules and corresponding version numbers.

**1.1.4 LTFATMEX - Compile Mex/Oct interfaces****Usage**

```
ltfatmex;
ltfatmex(...);
```

**Description**

`ltfatmex` compiles the C backend in order to speed up the execution of the toolbox. The C backend is linked to Matlab and Octave through Mex and Octave C++ interfaces.

The action of `ltfatmex` is determined by one of the following flags:

<code>'compile'</code>	Compile stuff. This is the default.
<code>'clean'</code>	Removes the compiled functions.
<code>'test'</code>	Run some small tests that verify that the compiled functions work.

The target to work on is determined by one of the following flags.

General LTFAT:

<code>'lib'</code>	Perform action on the LTFAT C library.
<code>'mex'</code>	Perform action on the mex / oct interfaces.
<code>'gpc'</code>	Perform action on the GPC code for use with MULACLAB
<code>'auto'</code>	Choose automatically which targets to work on from the previous ones based on the operation system etc. This is the default.

Block-processing framework related:

<code>'playrec'</code>	Perform action on the playrec code for use with real-time block streaming framework.
<code>'java'</code>	Perform compilation of JAVA classes into the bytecode. The classes makes the GUI for the blockproc. framework.

**1.1.5 LTFATBASEPATH - The base path of the LTFAT installation****Usage**

```
bp = ltfatbasepath;
```

**Description**

`ltfatbasepath` returns the top level directory in which the LTFAT files are installed.

**1.1.6 ISOCTAVE - True if the operating environment is octave****Usage**

```
t=isoctave();
```

**Description**

`is octave` returns 1 if the operating environment is Octave, otherwise it returns 0 (Matlab)

## 1.2 Parameter handling

### 1.2.1 LTFATARGHELPER - Parse arguments for LTFAT

**Usage**

```
[flags, varargout] = ltfatarghelper(posdepnames, definput, arglist, callfun);
```

**Input parameters**

<b>posdepnames</b>	Names of the position dependant parameters.
<b>definput</b>	Struct to define the allowed input
<b>arglist</b>	Commandline of the calling function (varargin)
<b>callfun</b>	Name of calling function (optional)

**Output parameters**

<b>flags</b>	Struct with information about flags.
<b>keyvals</b>	Struct with key / values.
<b>varargout</b>	The position dependant pars. properly initialized

**Description**

`[flags, keyvals]=ltfatarghelper (posdepnames, definput, arglist)` assists in parsing input parameters for a function in LTFAT. Parameters come in four categories:

- Position dependant parameters. These must not be strings. These are the first parameters passed to a function, and they are really just a short way of specifying key/value pairs. See below.
- Flags. These are single string appearing after the position dependant parameters.
- Key/value pairs. The key is always a string followed by the value, which can be anything.
- Expansions. These appear as flags, that expand into a pre-defined list of parameters. This is a short-hand way of specifying standard sets of flags and key/value pairs.

The parameters are parsed in order, so parameters appearing later in varargin will override previously set values.

The following example for calling `ltfatarghelper` is taken from dgt:

```
definput.keyvals.L=[];
definput.flags.phase={'freqinv','timeinv'};
[flags,kv]=ltfatarghelper({'L'},definput,varargin);
```

The first line defines a key/value pair with the key '`L`' having an initial value of [] (the empty matrix).

The second line defines a group of flags by the name of `phase`. The group `phase` contains the flags '`freqinv`' and '`timeinv`', which can both be specified on the command line by the user. The group-name `phase` is just for internal use, and does not appear to the user. The flag mentioned first in the list will be selected by default, and only one flag in a group can be selected at any time. A group can contain as many flags as desired.

The third line is the actual call to `ltfatarghelper` which defines the output `flags` and `kv`. The input `{'L'}` indicates that the value of the parameter '`L`' can also be given as the very first value in varargin.

The output struct `kv` contains the key/value pairs, so the value associated to '`L`' is stored in `kv.L`.

The output struct `flags` contains information about the flags chosen by the user. The value of `flags.phase` will be set to the selected flag in the group `phase` and additionally, the value of `flags.do_timeinv` will be 1 if '`timeinv`' was selected and 0 otherwise, and similarly for '`freqinv`'. This allows for easy checking of selected flags.

### 1.2.2 LTFATGETDEFAULTS - Get default parameters of function

`ltfatgetdefaults(fname)` returns the default parameters of the function `fname` as a cell array.  
`ltfatgetdefaults('all')` returns all the set defaults.

### 1.2.3 LTFATSETDEFAULTS - Set default parameters of function

`ltfatsetdefaults(fname, ...)` sets the default parameters to be the parameters specified at the end of the list of input arguments.

`ltfatsetdefaults(fname)` clears any default parameters for the function `fname`.  
`ltfatsetdefaults('clearall')` clears all defaults from all functions.

### 1.2.4 SCALARDISTRIBUTE - Copy scalar to array shape for parameter handling

#### Usage

```
[...] = scalardistribute(...);
```

#### Description

`[...] = scalardistribute(...)` copies the input parameters to the output parameters.

- If one of the input parameters is an array, all the output parameters will be column vectors containing the same number of elements. If one of the other input parameters is a scalar, it will be replicated to the correct length. This allows a scalar value to be repeated for all conditions.
- If two or more input parameters are arrays, they must have the exact same size. They will be converted to vectors and returned in the output parameters. This allows two arrays to co-vary at the same time.

This operator is useful for sanitizing input parameters: The user is allowed to enter scalars or arrays as input parameters. These inputs are in turn passed to `scaldardistribute`, which makes sure that the arrays have the same shape, and that scalars are replicated. The user of `scaldardistribute` can now generate conditions based on all the parameters, and be sure they have the right sizes.

As an example, consider:

```
[a,b,c]=scaldardistribute(1,[2,3],[4,5])
```

*This code produces the following output:*

```
a =
```

```
1
1
```

```
b =
```

```
2
3
```

```
c =
```

```
4
5
```

## 1.3 Graphical user interfaces

### 1.3.1 MULACLAB - Graphical interface for audio processing using frame multipliers

#### Usage

```
mulaclab;
```

#### Description

When starting the interface, the user is asked to choose the processed signal, named original signal in the interface. Possible signals are .wav files and .mat files containing decompositions preliminarily saved using the mulaclab interface. The interface only handles monochannel signals. So for a multichannel .wav files, the first channel is used as the original signal.

After choosing the original signal, the user is presented with the main interface. This interface is divided in two areas:

- The right part of the figure contains the visualizations, which represent the spectrograms of the original and modified signals.

The 'Original signal' visualization is used to display the original signal spectrogram and to graphically define the symbol of the Gabor multiplier that will be applied on this signal.

The 'Overview of original signal' visualization also represents the original signal spectrogram and can be used for fast zooming and moving of the other visualizations. Zooming and moving is controlled by mouse interaction with the white rectangle displayed on this visualization.

The 'Modified signal' visualization is used to display the spectrogram of the modified signal after application of the multiplier.

It is possible to hide the 'Overview of original signal' and 'Modified signal' visualizations using the 'Visualization' menu.

- The left part of the figure contains panels with tools for user interaction.

The 'Audioplayer' panel contains the controls for audio playback of the original and modified signal.

The 'Visualization' panel contains tools used to adapt the display of the visualizations.

The 'Selection' panel contains tools and information concerning the multilayered selection used to graphically specify the symbol of the multiplier.

#### Known Matlab limitations:

- When using Matlab on Linux with multiple screens, there might be a Matlab bug preventing the display of the multiplier symbol. This can be solved by docking the figure.
- When using a Matlab version prior to 7.3 (R2006b), the rectangle displayed on the 'Overview of original signal' visualization is not automatically updated when using the zoom and pan tools of the 'Zoom' panel. It can be manually updated by re-clicking on the currently selected tool or by changing the current tool.

The Matlab Image Processing Toolbox is required by the mulaclab function.

MULACLAB uses the GPC library available from <http://www.cs.man.ac.uk/~toby/gpc/>.

This library is distributed alongside LTFAT, but under different licensing conditions. Please see the `ltfat/thirdparty/gpc` file for the exact conditions.



# Chapter 2

## LTFAT - Gabor analysis

### 2.1 Basic Time/Frequency analysis

#### 2.1.1 TCONV - Twisted convolution

##### Usage

```
h=tconv(f,g);
```

##### Description

`tconv(f,g)` computes the twisted convolution of the square matrices  $f$  and  $g$ .

Let  $h = tconv(f, g)$  for  $f, g$  being  $L \times L$  matrices. Then  $h$  is given by

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

where  $m - k$  and  $n - l$  are computed modulo  $L$ .

If both  $f$  and  $g$  are of class `sparse` then  $h$  will also be a sparse matrix. The number of non-zero elements of  $h$  is usually much larger than the numbers for  $f$  and  $g$ . Unless  $f$  and  $g$  are very sparse, it can be faster to convert them to full matrices before calling `tconv`.

The routine ?? can be used to calculate an inverse convolution. Define  $h$  and  $r$  by:

```
h=tconv(f,g);
r=tconv(spreadinv(f),h);
```

then  $r$  is equal to  $g$ .

#### 2.1.2 DSFT - Discrete Symplectic Fourier Transform

##### Usage

```
C=dsft(F);
```

##### Description

`dsft(F)` computes the discrete symplectic Fourier transform of  $F$ .  $F$  must be a matrix or a 3D array. If  $F$  is a 3D array, the transformation is applied along the first two dimensions.

Let  $F$  be a  $K \times L$  matrix. Then the DSFT of  $F$  is given by

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

for  $m = 0, \dots, L-1$  and  $n = 0, \dots, K-1$ .

The `dsft` is its own inverse.

**References:** [28]

### 2.1.3 ZAK - Zak transform

#### Usage

```
c=zak(f,a);
```

#### Description

`zak(f,a)` computes the Zak transform of  $f$  with parameter  $a$ . The coefficients are arranged in an  $a \times L/a$  matrix, where  $L$  is the length of  $f$ .

If  $f$  is a matrix then the transformation is applied to each column. This is then indexed by the third dimension of the output.

Assume that  $c = \text{zak}(f,a)$ , where  $f$  is a column vector of length  $L$  and  $N = L/a$ . Then the following holds for  $m = 0, \dots, a-1$  and  $n = 0, \dots, N-1$

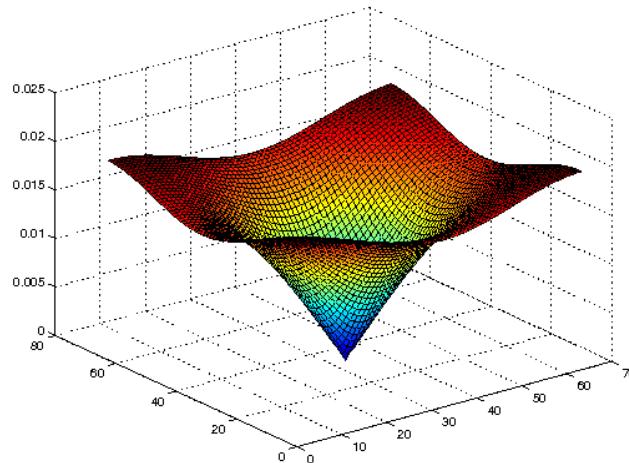
$$c(m+1, n+1) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} f(m-ka+1) e^{2\pi i nk/M}$$

#### Examples:

This figure shows the absolute value of the Zak-transform of a Gaussian. Notice that the Zak-transform is 0 in only a single point right in the middle of the plot

```
a=64;
L=a^2;
g=pgauss(L);
zg=zak(g,a);

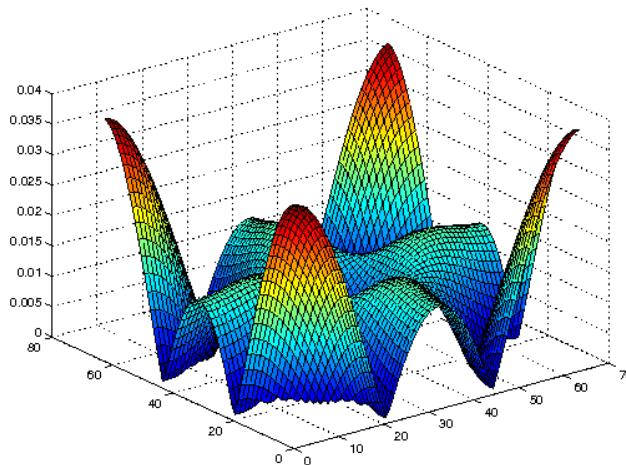
surf(abs(zg));
```



This figure shows the absolute value of the Zak-transform of a 4th order Hermite function. Notice how the Zak transform of the Hermite functions is zero on a circle centered on the corner

```
a=64;
L=a^2;
g=pherm(L,4);
zg=zak(g,a);

surf(abs(zg));
```



**References:** [41], [14]

#### 2.1.4 IZAK - Inverse Zak transform

##### Usage

```
f=izak(c);
```

##### Description

`izak(c)` computes the inverse Zak transform of  $c$ . The parameter of the Zak transform is deduced from the size of  $c$ .

**References:** [41], [14]

#### 2.1.5 COL2DIAG - Move columns of a matrix to diagonals

##### Usage

```
cout=col2diag(cin);
```

##### Description

`col2diag(cin)` will rearrange the elements in the square matrix `cin` so that columns of `cin` appears as diagonals. Column number  $n$  will appear as diagonal number  $-n$  and  $L - n$ , where  $L$  is the size of the matrix.

The function is its own inverse.

`col2diag` performs the underlying coordinate transform for spreading function and Kohn-Nirenberg calculus in the finite, discrete setting.

#### 2.1.6 S0NORM - S<sub>0</sub>-norm of signal

##### Usage

```
y = s0norm(f);
y = s0norm(f, ...);
```

##### Description

`s0norm(f)` computes the  $S_0$ -norm of a vector.

If the input is a matrix or ND-array, the  $S_0$ -norm is computed along the first (non-singleton) dimension, and a vector of values is returned.

**WARNING:** The  $S_0$ -norm is computed by computing a full Short-time Fourier transform of a signal, which can be quite time-consuming. Use this function with care for long signals.

`s0norm` takes the following flags at the end of the line of input parameters:

<b>'dim',d</b>	Work along specified dimension. The default value of [] means to work along the first non-singleton one.
<b>'rel'</b>	Return the result relative to the $l^2$ norm (the energy) of the signal.

## 2.2 Gabor systems

### 2.2.1 DGT - Discrete Gabor transform

#### Usage

```
c=dgt(f,g,a,M);
c=dgt(f,g,a,M,L);
c=dgt(f,g,a,M,'lt',lt);
[c,Ls]=dgt(...);
```

#### Input parameters

<b>f</b>	Input data.
<b>g</b>	Window function.
<b>a</b>	Length of time shift.
<b>M</b>	Number of channels.
<b>L</b>	Length of transform to do.
<b>lt</b>	Lattice type (for non-separable lattices).

#### Output parameters

<b>c</b>	$M \times N$ array of coefficients.
<b>Ls</b>	Length of input signal.

#### Description

`dgt(f,g,a,M)` computes the Gabor coefficients (also known as a windowed Fourier transform) of the input signal  $f$  with respect to the window  $g$  and parameters  $a$  and  $M$ . The output is a vector/matrix in a rectangular layout.

The length of the transform will be the smallest multiple of  $a$  and  $M$  that is larger than the signal.  $f$  will be zero-extended to the length of the transform. If  $f$  is a matrix, the transformation is applied to each column. The length of the transform done can be obtained by `L=size(c,2)*a;`

The window  $g$  may be a vector of numerical values, a text string or a cell array. See the help of `gabwin` for more details.

`dgt(f,g,a,M,L)` computes the Gabor coefficients as above, but does a transform of length  $L$ .  $f$  will be cut or zero-extended to length  $L$  before the transform is done.

`[c,Ls]=dgt(f,g,a,M)` or `[c,Ls]=dgt(f,g,a,M,L)` additionally returns the length of the input signal  $f$ . This is handy for reconstruction:

```
[c,Ls]=dgt(f,g,a,M);
fr=idgt(c,gd,a,Ls);
```

will reconstruct the signal  $f$  no matter what the length of  $f$  is, provided that  $gd$  is a dual window of  $g$ .

`[c, Ls, g]=dgt(...)` additionally outputs the window used in the transform. This is useful if the window was generated from a description in a string or cell array.

The Discrete Gabor Transform is defined as follows: Consider a window  $g$  and a one-dimensional signal  $f$  of length  $L$  and define  $N = L/a$ . The output from `c=dgt(f, g, a, M)` is then given by:

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

where  $m = 0, \dots, M-1$  and  $n = 0, \dots, N-1$  and  $l-an$  is computed modulo  $L$ .

#### Non-separable lattices:

`dgt(f, g, a, M, 'lt', lt)` computes the DGT for a non-separable lattice given by the time-shift  $a$ , number of channels  $M$  and lattice type  $lt$ . Please see the help of `matrix2lattice` for a precise description of the parameter  $lt$ .

The non-separable discrete Gabor transform is defined as follows: Consider a window  $g$  and a one-dimensional signal  $f$  of length  $L$  and define  $N = L/a$ . The output from `c=dgt(f, g, a, M, L, lt)` is then given by:

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

where  $m = 0, \dots, M-1$  and  $n = 0, \dots, N-1$  and  $l-an$  are computed modulo  $L$ . The additional offset  $w$  is given by  $w(n) = \text{mod}(n \cdot lt_1, lt_2)/lt_2$  in the formula above.

#### Additional parameters:

`dgt` takes the following flags at the end of the line of input arguments:

<b>'freqinv'</b>	Compute a DGT using a frequency-invariant phase. This is the default convention described above.
<b>'timeinv'</b>	Compute a DGT using a time-invariant phase. This convention is typically used in FIR-filter algorithms.

#### Examples:

In the following example we create a Hermite function, which is a complex-valued function with a circular spectrogram, and visualize the coefficients using both `imagesc` and `plotdgt`:

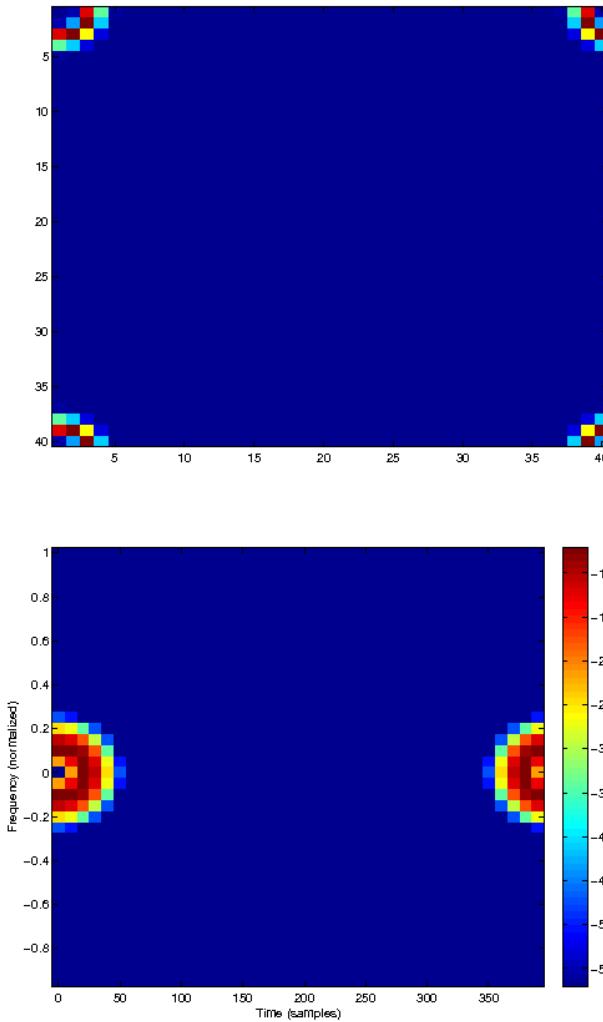
```

a=10;
M=40;
L=a*M;
h=pherm(L, 4); % 4th order hermite function.
c=dgt(h, 'gauss', a, M);

% Simple plot: The squared modulus of the coefficients on
% a linear scale
figure(1);
imagesc(abs(c).^2);

% Better plot: zero-frequency is displayed in the middle,
% and the coefficients are show on a logarithmic scale.
figure(2);
plotdgt(c, a, 'dynrange', 50);

```



**References:** [29], [36]

### 2.2.2 IDGT - Inverse discrete Gabor transform

#### Usage

```
f=idgt(c,g,a);
f=idgt(c,g,a,Ls);
f=idgt(c,g,a,Ls,lt);
```

#### Input parameters

<b>c</b>	Array of coefficients.
<b>g</b>	Window function.
<b>a</b>	Length of time shift.
<b>Ls</b>	Length of signal.
<b>lt</b>	Lattice type (for non-separable lattices)

#### Output parameters

<b>f</b>	Signal.
----------	---------

### Description

`idgt(c, g, a)` computes the Gabor expansion of the input coefficients  $c$  with respect to the window  $g$  and time shift  $a$ . The number of channels is deduced from the size of the coefficients  $c$ .

`idgt(c, g, a, Ls)` does as above but cuts or extends  $f$  to length  $Ls$ .

`[f, g]=idgt(...)` additionally outputs the window used in the transform. This is useful if the window was generated from a description in a string or cell array.

For perfect reconstruction, the window used must be a dual window of the one used to generate the coefficients.

The window  $g$  may be a vector of numerical values, a text string or a cell array. See the help of `gabwin` for more details.

If  $g$  is a row vector, then the output will also be a row vector. If  $c$  is 3-dimensional, then `idgt` will return a matrix consisting of one column vector for each of the TF-planes in  $c$ .

Assume that `f=idgt(c, g, a, L)` for an array  $c$  of size  $M \times N$ . Then the following holds for  $k = 0, \dots, L-1$ :

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

### Non-separable lattices:

`idgt(c, g, a, 'lt', lt)` computes the Gabor expansion of the input coefficients  $c$  with respect to the window  $g$ , time shift  $a$  and lattice type  $lt$ . Please see the help of `matrix2lattice` for a precise description of the parameter  $lt$ .

Assume that `f=dgt(c, g, a, L, lt)` for an array  $c$  of size  $M \times N$ . Then the following holds for  $k = 0, \dots, L-1$ :

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

### Additional parameters:

`idgt` takes the following flags at the end of the line of input arguments:

<b>'freqinv'</b>	Compute an IDGT using a frequency-invariant phase. This is the default convention described above.
<b>'timeinv'</b>	Compute an IDGT using a time-invariant phase. This convention is typically used in FIR-filter algorithms.

### Examples:

The following example demonstrates the basic principles for getting perfect reconstruction (short version):

```

f=greasy;           % test signal
a=32;              % time shift
M=64;              % frequency shift
gs={'blackman',128}; % synthesis window
ga={'dual',gs};    % analysis window

[c,Ls]=dgt(f,ga,a,M);    % analysis

% ... do interesting stuff to c at this point ...

r=idgt(c,gs,a,Ls); % synthesis

norm(f-r)            % test

```

*This code produces the following output:*

```
ans =
```

```
5.8386e-15
```

The following example does the same as the previous one, with an explicit construction of the analysis and synthesis windows:

```
f=greasy; % test signal
a=32; % time shift
M=64; % frequency shift
Ls=length(f); % signal length

% Length of transform to do
L=dgtlength(Ls,a,M);

% Analysis and synthesis window
gs=firwin('blackman',128);
ga=gabdual(gs,a,M,L);

c=dgt(f,ga,a,M); % analysis

% ... do interesting stuff to c at this point ...

r=idgt(c,gs,a,Ls); % synthesis

norm(f-r) % test
```

*This code produces the following output:*

```
ans =
```

```
5.8775e-15
```

### 2.2.3 ISGRAM - Spectrogram inversion

#### Usage

```
f=isgram(c,g,a);
f=isgram(c,g,a,Ls);
[f,relres,iter]=isgram(...);
```

#### Input parameters

<b>c</b>	Array of coefficients.
<b>g</b>	Window function.
<b>a</b>	Length of time shift.
<b>Ls</b>	length of signal.

#### Output parameters

<b>f</b>	Signal.
<b>relres</b>	Vector of residuals.
<b>iter</b>	Number of iterations done.

### Description

`isgram(s,g,a)` attempts to invert a spectrogram computed by

```
s = abs(dgt(f,g,a,M)).^2;
```

using an iterative method.

`isgram(c,g,a,Ls)` does as above but cuts or extends  $f$  to length  $L_s$ .

If the phase of the spectrogram is known, it is much better to use `idgt`.

`[f,relres,iter]=isgram(...)` additionally return the residuals in a vector `relres` and the number of iteration steps `iter`.

Generally, if the spectrogram has not been modified, the iterative algorithm will converge slowly to the correct result. If the spectrogram has been modified, the algorithm is not guaranteed to converge at all.

`isgram` takes the following parameters at the end of the line of input arguments:

<code>'lt',lt</code>	Specify the lattice type. See the help on <code>matrix2latticetype</code> .
<code>'zero'</code>	Choose a starting phase of zero. This is the default
<code>'rand'</code>	Choose a random starting phase.
<code>'int'</code>	Construct a starting phase by integration. Only works for Gaussian windows.
<code>'griflim'</code>	Use the Griffin-Lim iterative method, this is the default.
<code>'bfgs'</code>	Use the limited-memory Broyden Fletcher Goldfarb Shanno (BFGS) method.
<code>'tol',t</code>	Stop if relative residual error is less than the specified tolerance.
<code>'maxit',n</code>	Do at most n iterations.
<code>'print'</code>	Display the progress.
<code>'quiet'</code>	Don't print anything, this is the default.
<code>'printstep',p</code>	If <code>'print'</code> is specified, then print every p'th iteration. Default value is p=10;

To use the BFGS method, please install the `minFunc` software from <http://www.cs.ubc.ca/~schmidtm/Software/minFunc.html>.

### Examples:

To reconstruct the phase of 'greasy', use the following:

```
% Setup the problem and the coefficients
f=greasy;
g='gauss';
a=20; M=200;
c=dgt(f,g,a,M);
s=abs(c).^2;
theta=angle(c);

% Reconstruct and get spectrogram and angle
r=isgram(s,g,a);
c_r=dgt(r,g,a,M);
s_r=abs(c_r).^2;
theta_r=angle(c_r);

% Compute the angular difference
```

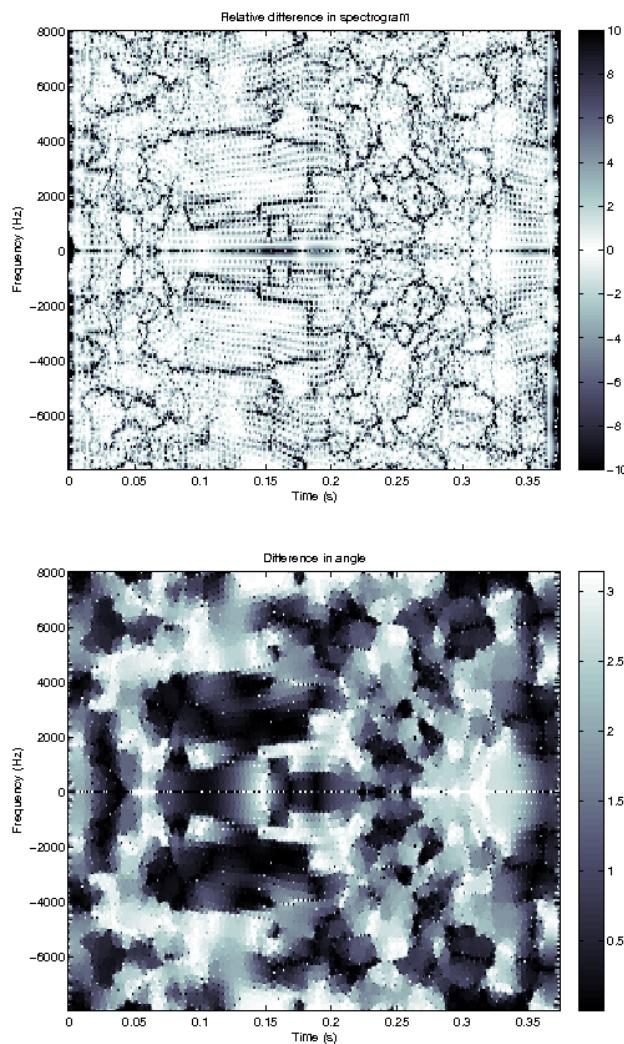
```

d1=abs(theta-theta_r);
d2=2*pi-d1;
anglediff=min(d1,d2);

% Plot the difference in spectrogram and phase
figure(1);
plotdgt(s./s_r,a,16000,'clim',[ -10,10]);
colormap([bone;flipud(bone)])
title('Relative difference in spectrogram');

figure(2);
plotdgt(anglediff,a,16000,'lin');
colormap(bone);
title('Difference in angle');

```



**References:** [35], [25], [53]

#### 2.2.4 ISGRAMREAL - Spectrogram inversion (real signal)

##### Usage

```

f=isgramreal(s,g,a,M);
f=isgramreal(s,g,a,M,Ls);

```

```
[f, relres, iter]=isgramreal(...);
```

**Input parameters**

<b>c</b>	Array of coefficients.
<b>g</b>	Window function.
<b>a</b>	Length of time shift.
<b>M</b>	Number of channels.
<b>Ls</b>	length of signal.

**Output parameters**

<b>f</b>	Signal.
<b>relres</b>	Vector of residuals.
<b>iter</b>	Number of iterations done.

**Description**

`isgramreal(s, g, a, M)` attempts to invert a spectrogram computed by

```
s = abs(dgtreal(f, g, a, M)).^2;
```

by an iterative method.

`isgramreal(s, g, a, M, Ls)` does as above but cuts or extends  $f$  to length  $Ls$ .

If the phase of the spectrogram is known, it is much better to use `dgtreal`

`f, relres, iter]=isgramreal(...)` additionally returns the residuals in a vector `relres` and the number of iteration steps done.

Generally, if the spectrogram has not been modified, the iterative algorithm will converge slowly to the correct result. If the spectrogram has been modified, the algorithm is not guaranteed to converge at all.

`isgramreal` takes the following parameters at the end of the line of input arguments:

<b>'lt',lt</b>	Specify the lattice type. See the help on <code>matrix2lattice</code> . Only the rectangular or quinque lattices can be specified.
<b>'zero'</b>	Choose a starting phase of zero. This is the default
<b>'rand'</b>	Choose a random starting phase.
<b>'int'</b>	Construct a starting phase by integration. Only works for Gaussian windows.
<b>'griflim'</b>	Use the Griffin-Lim iterative method, this is the default.
<b>'bfgs'</b>	Use the limited-memory Broyden Fletcher Goldfarb Shanno (BFGS) method.
<b>'tol',t</b>	Stop if relative residual error is less than the specified tolerance.
<b>'maxit',n</b>	Do at most n iterations.
<b>'print'</b>	Display the progress.
<b>'quiet'</b>	Don't print anything, this is the default.
<b>'printstep',p</b>	If 'print' is specified, then print every p'th iteration. Default value is p=10.

To use the BFGS method, please install the `minFunc` software from <http://www.cs.ubc.ca/~schmidtm/Software/minFunc.html>

**References:** [35], [25], [53]

### 2.2.5 DGT2 - 2-D Discrete Gabor transform

#### Usage

```
c=dgt2(f,g,a,M);
c=dgt2(f,g1,g2,[a1,a2],[M1,M2]);
c=dgt2(f,g1,g2,[a1,a2],[M1,M2],[L1,L2]);
[c,Ls]=dgt2(f,g1,g2,[a1,a2],[M1,M2]);
[c,Ls]=dgt2(f,g1,g2,[a1,a2],[M1,M2],[L1,L2]);
```

#### Input parameters

<b>f</b>	Input data, matrix.
<b>g, g1, g2</b>	Window functions.
<b>a, a1, a2</b>	Length of time shifts.
<b>M, M1, M2</b>	Number of modulations.
<b>L1, L2</b>	Length of transform to do

#### Output parameters

<b>c</b>	array of coefficients.
<b>Ls</b>	Original size of input matrix.

#### Description

`dgt2(f,g,a,M)` will calculate a separable two-dimensional discrete Gabor transformation of the input signal  $f$  with respect to the window  $g$  and parameters  $a$  and  $M$ .

For each dimension, the length of the transform will be the smallest possible that is larger than the length of the signal along that dimension.  $f$  will be appropriately zero-extended.

`dgt2(f,g,a,M,L)` computes a Gabor transform as above, but does a transform of length  $L$  along each dimension.  $f$  will be cut or zero-extended to length  $L$  before the transform is done.

`[c,Ls]=dgt2(f,g,a,M)` or `[c,Ls]=dgt2(f,g,a,M,L)` additionally returns the length of the input signal  $f$ . This is handy for reconstruction:

```
[c,Ls]=dgt2(f,g,a,M);
fr=idgt2(c,gd,a,Ls);
```

will reconstruct the signal  $f$  no matter what the size of  $f$  is, provided that  $gd$  is a dual window of  $g$ .

`dgt2(f,g1,g2,a,M)` makes it possible to use a different window along the two dimensions.

The parameters  $a$ ,  $M$ ,  $L$  and  $Ls$  can also be vectors of length 2. In this case the first element will be used for the first dimension and the second element will be used for the second dimension.

The output  $c$  has 4 or 5 dimensions. The dimensions index the following properties:

1. Number of translation along 1st dimension of input.
2. Number of channel along 1st dimension of input
3. Number of translation along 2nd dimension of input.
4. Number of channel along 2nd dimension of input
5. Plane number, corresponds to 3rd dimension of input.

### 2.2.6 IDGT2 - 2D Inverse discrete Gabor transform

#### Usage

```
f=idgt2(c,g,a);
f=idgt2(c,g1,g2,a);
f=idgt2(c,g1,g2,[a1 a2]);
f=idgt2(c,g,a,Ls);
f=idgt2(c,g1,g2,a,Ls);
f=idgt2(c,g1,g2,[a1 a2],Ls);
```

#### Input parameters

<b>c</b>	Array of coefficients.
<b>g, g1, g2</b>	Window function(s).
<b>a, a1, a2</b>	Length(s) of time shift.
<b>Ls</b>	Length(s) of reconstructed signal (optional).

#### Output parameters

<b>f</b>	Output data, matrix.
----------	----------------------

#### Description

`idgt2(c,g,a,M)` will calculate a separable two dimensional inverse discrete Gabor transformation of the input coefficients  $c$  using the window  $g$  and parameters  $a$ , along each dimension. The number of channels is deduced from the size of the coefficients  $c$ .

`idgt2(c,g1,g2,a)` will do the same using the window  $g1$  along the first dimension, and window  $g2$  along the second dimension.

`idgt2(c,g,a,Ls)` or `idgt2(c,g1,g2,a,Ls)` will cut the signal to size  $Ls$  after the transformation is done.

The parameters  $a$  and  $Ls$  can also be vectors of length 2. In this case the first element will be used for the first dimension and the second element will be used for the second dimension.

### 2.2.7 DGTREAL - Discrete Gabor transform for real-valued signals

#### Usage

```
c=dgtreal(f,g,a,M);
c=dgtreal(f,g,a,M,L);
[c,Ls]=dgtreal(f,g,a,M);
[c,Ls]=dgtreal(f,g,a,M,L);
```

#### Input parameters

<b>f</b>	Input data
<b>g</b>	Window function.
<b>a</b>	Length of time shift.
<b>M</b>	Number of modulations.
<b>L</b>	Length of transform to do.

## Output parameters

**c**  $M \times N$  array of coefficients.

**Ls** Length of input signal.

## Description

`dgtreal(f, g, a, M)` computes the Gabor coefficients (also known as a windowed Fourier transform) of the real-valued input signal  $f$  with respect to the real-valued window  $g$  and parameters  $a$  and  $M$ . The output is a vector/matrix in a rectangular layout.

As opposed to `dgt` only the coefficients of the positive frequencies of the output are returned. `dgtreal` will refuse to work for complex valued input signals.

The length of the transform will be the smallest multiple of  $a$  and  $M$  that is larger than the signal.  $f$  will be zero-extended to the length of the transform. If  $f$  is a matrix, the transformation is applied to each column. The length of the transform done can be obtained by `L=size(c, 2)*a`.

The window  $g$  may be a vector of numerical values, a text string or a cell array. See the help of `gabwin` for more details.

`dgtreal(f, g, a, M, L)` computes the Gabor coefficients as above, but does a transform of length  $L$ .  $f$  will be cut or zero-extended to length  $L$  before the transform is done.

`[c, Ls]=dgtreal(f, g, a, M)` or `[c, Ls]=dgtreal(f, g, a, M, L)` additionally returns the length of the input signal  $f$ . This is handy for reconstruction:

```
[c, Ls]=dgtreal(f, g, a, M);
fr=idgtreal(c, gd, a, M, Ls);
```

will reconstruct the signal  $f$  no matter what the length of  $f$  is, provided that  $gd$  is a dual window of  $g$ .

`[c, Ls, g]=dgtreal(...)` additionally outputs the window used in the transform. This is useful if the window was generated from a description in a string or cell array.

See the help on `dgt` for the definition of the discrete Gabor transform. This routine will return the coefficients for channel frequencies from 0 to `floor(M/2)`.

`dgtreal` takes the following flags at the end of the line of input arguments:

**'freqinv'** Compute a `dgtreal` using a frequency-invariant phase. This is the default convention described in the help for `dgt`.

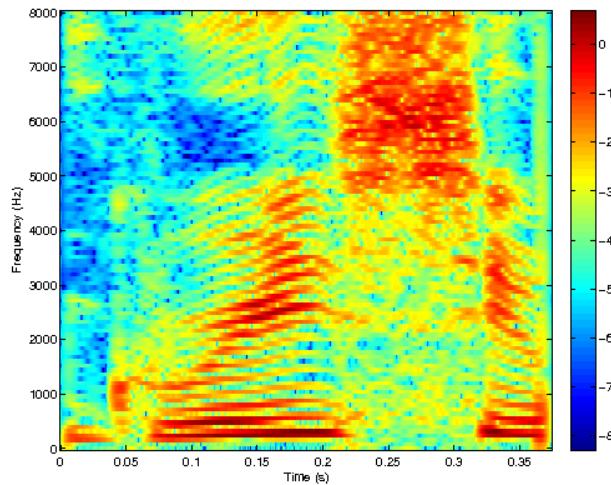
**'timeinv'** Compute a `dgtreal` using a time-invariant phase. This convention is typically used in filter bank algorithms.

`dgtreal` can be used to manually compute a spectrogram, if you want full control over the parameters and want to capture the output

```
f=greasy; % Input test signal
fs=16000; % The sampling rate of this particular test signal
a=10; % Downsampling factor in time
M=200; % Total number of channels, only 101 will be computed

% Compute the coefficients using a 20 ms long Hann window
c=dgtreal(f, {'hann', 0.02*fs}, a, M);

% Visualize the coefficients as a spectrogram
dynrange=90; % 90 dB dynamical range for the plotting
plotdgtreal(c, a, M, fs, dynrange);
```



**References:** [29], [36]

### 2.2.8 IDGTRREAL - Inverse discrete Gabor transform for real-valued signals

#### Usage

```
f=idgtreal(c,g,a,M);
f=idgtreal(c,g,a,M,Ls);
```

#### Input parameters

<b>c</b>	Array of coefficients.
<b>g</b>	Window function.
<b>a</b>	Length of time shift.
<b>M</b>	Number of channels.
<b>Ls</b>	length of signal.

#### Output parameters

<b>f</b>	Signal.
----------	---------

#### Description

`idgtreal(c, g, a, M)` computes the Gabor expansion of the input coefficients  $c$  with respect to the real-valued window  $g$ , time shift  $a$  and number of channels  $M$ .  $c$  is assumed to be the positive frequencies of the Gabor expansion of a real-valued signal.

It must hold that `size(c, 1) == floor(M/2) + 1`. Note that since the correct number of channels cannot be deduced from the input, `idgtreal` takes an additional parameter as opposed to `idgt`.

The window  $g$  may be a vector of numerical values, a text string or a cell array. See the help of `gabwin` for more details.

`idgtreal(c, g, a, M, Ls)` does as above but cuts or extends  $f$  to length  $Ls$ .

`[f, g] = idgtreal(...)` additionally outputs the window used in the transform. This is useful if the window was generated from a description in a string or cell array.

For perfect reconstruction, the window used must be a dual window of the one used to generate the coefficients.

If  $g$  is a row vector, then the output will also be a row vector. If  $c$  is 3-dimensional, then `idgtreal` will return a matrix consisting of one column vector for each of the TF-planes in  $c$ .

See the help on `idgt` for the precise definition of the inverse Gabor transform.

`idgtreal` takes the following flags at the end of the line of input arguments:

- 'freqinv'** Use a frequency-invariant phase. This is the default convention described in the help for `dgt`.
- 'timeinv'** Use a time-invariant phase. This convention is typically used in filter bank algorithms.

### Examples:

The following example demonstrates the basic principles for getting perfect reconstruction (short version):

```
f=greasy; % test signal
a=32; % time shift
M=64; % frequency shift
gs={'blackman',128}; % synthesis window
ga={'dual',gs}; % analysis window

[c,Ls]=dgtreal(f,ga,a,M); % analysis

% ... do interesting stuff to c at this point ...

r=idgtreal(c,gs,a,M,Ls); % synthesis

norm(f-r) % test
```

*This code produces the following output:*

```
ans =
5.7333e-15
```

The following example does the same as the previous one, with an explicit construction of the analysis and synthesis windows:

```
f=greasy; % test signal
a=32; % time shift
M=64; % frequency shift
Ls=length(f); % signal length

% Length of transform to do
L=dgtlength(Ls,a,M);

% Analysis and synthesis window
gs=firwin('blackman',128);
ga=gabdual(gs,a,M,L);

c=dgtral(f,ga,a,M); % analysis

% ... do interesting stuff to c at this point ...

r=idgtral(c,gs,a,M,Ls); % synthesis

norm(f-r) % test
```

*This code produces the following output:*

```
ans =
5.9870e-15
```

### 2.2.9 GABWIN - Compute a Gabor window from text or cell array

#### Usage

```
[g,info] = gabwin(g,a,M,L);
```

#### Description

`[g,info]=gabwin(g,a,M,L)` computes a window that fits well with the specified number of channels  $M$ , time shift  $a$  and transform length  $L$ . The window itself is specified by a text description or a cell array containing additional parameters.

The window can be specified directly as a vector of numerical values. In this case, `gabwin` only checks assumptions about transform sizes etc.

`[g,info]=gabwin(g,a,M)` does the same, but the window must be a FIR window, as the transform length is unspecified.

`gabwin(g,a,M,L,lt)` or `gabwin(g,a,M,[],lt)` does as above but for a non-separable lattice specified by `lt`. Please see the help of `matrix2latticetype` for a precise description of the parameter `lt`.

The window can be specified as one of the following text strings:

<b>'gauss'</b>	Gaussian window fitted to the lattice, i.e. $tfr = a \cdot M / L$ .
<b>'dualgauss'</b>	Canonical dual of Gaussian window.
<b>'tight'</b>	Tight window generated from a Gaussian.

In these cases, a long window is generated with a length of  $L$ .

It is also possible to specify one of the window names from `firwin`. In such a case, `gabwin` will generate the specified FIR window with a length of  $M$ .

The window can also be specified as cell array. The possibilities are:

- {'gauss', ...} Additional parameters are passed to `pgauss`.
- {'dual', ...} Canonical dual window of whatever follows. See the examples below.
- {'tight', ...} Canonical tight window of whatever follows.

It is also possible to specify one of the window names from `firwin` as the first field in the cell array. In this case, the remaining entries of the cell array are passed directly to `firwin`.

Some examples: To compute a Gaussian window of length  $L$  fitted for a system with time-shift  $a$  and  $M$  channels use:

```
g=gabwin('gauss',a,M,L);
```

To compute Gaussian window with equal time and frequency support irrespective of  $a$  and  $M$ :

```
g=gabwin({'gauss',1},a,M,L);
```

To compute the canonical dual of a Gaussian window fitted for a system with time-shift  $a$  and  $M$  channels:

```
gd=gabwin('gaussdual',a,M,L);
```

To compute the canonical tight window of the Gaussian window fitted for the system:

```
gd=gabwin({'tight','gauss'},a,M,L);
```

To compute the dual of a Hann window of length 20:

```
g=gabwin({'dual',{'hann',20}},a,M,L);
```

The structure `info` provides some information about the computed window:

**info.gauss** True if the window is a Gaussian.

**info.tfr** Time/frequency support ratio of the window. Set whenever it makes sense.

```
info.wasrow Input was a row window
info.isfir Input is an FIR window
info.isdual Output is the dual window of the auxiliary window.
info.istight Output is known to be a tight window.
info.auxinfo Info about auxiliary window.
info.g1 Length of window.
```

### 2.2.10 DGTLENGTH - DGT length from signal

#### Usage

```
L=dgtlength(Ls,a,M);
L=dgtlength(Ls,a,M,lt);
```

#### Description

`dgtlength(Ls,a,M)` returns the length of a Gabor system that is long enough to expand a signal of length  $L_s$ . Please see the help on `dgt` for an explanation of the parameters  $a$  and  $M$ .

If the returned length is longer than the signal length, the signal will be zero-padded by `dgt`.

A valid transform length must be divisible by both  $a$  and  $M$ . This means that the minimal admissible transform length is

```
Lsmallest = lcm(a,M);
```

and all valid transform lengths are multipla of  $L_{smallest}$

#### Non-separable lattices:

`dgtlength(Ls,a,M,lt)` does as above for a non-separable lattice with lattice-type  $lt$ . For non-separable lattices, there is the additinal requirement on the transform length, that the structure of the lattice must be periodic. This gives a minimal transform length of

```
Lsmallest = lcm(a,M)*lt(2);
```

## 2.3 Wilson bases and WMDCT

### 2.3.1 DWILT - Discrete Wilson transform

#### Usage

```
c=dwilt(f,g,M);
c=dwilt(f,g,M,L);
[c,Ls]=dwilt(...);
```

#### Input parameters

<b>f</b>	Input data
<b>g</b>	Window function.
<b>M</b>	Number of bands.
<b>L</b>	Length of transform to do.

#### Output parameters

<b>c</b>	$2M \times N$ array of coefficients.
<b>Ls</b>	Length of input signal.

### Description

`dwilt(f, g, M)` computes a discrete Wilson transform with  $M$  bands and window  $g$ .

The length of the transform will be the smallest possible that is larger than the signal.  $f$  will be zero-extended to the length of the transform. If  $f$  is a matrix, the transformation is applied to each column.

The window  $g$  may be a vector of numerical values, a text string or a cell array. See the help of `wilwin` for more details.

`dwilt(f, g, M, L)` computes the Wilson transform as above, but does a transform of length  $L$ .  $f$  will be cut or zero-extended to length  $L$  before the transform is done.

`[c, Ls] = dwilt(f, g, M)` or `[c, Ls] = dwilt(f, g, M, L)` additionally return the length of the input signal  $f$ . This is handy for reconstruction:

```
[c, Ls] = dwilt(f, g, M);
fr = idwilt(c, gd, M, Ls);
```

will reconstruct the signal  $f$  no matter what the length of  $f$  is, provided that  $gd$  is a dual Wilson window of  $g$ .

`[c, Ls, g] = dwilt(...)` additionally outputs the window used in the transform. This is useful if the window was generated from a description in a string or cell array.

A Wilson transform is also known as a maximally decimated, even-stacked cosine modulated filter bank.

Use the function `wil2rect` to visualize the coefficients or to work with the coefficients in the TF-plane.

Assume that the following code has been executed for a column vector  $f$ :

```
c = dwilt(f, g, M); % Compute a Wilson transform of f.
N = size(c, 2) * 2; % Number of translation coefficients.
```

The following holds for  $m = 0, \dots, M - 1$  and  $n = 0, \dots, N/2 - 1$ :

If  $m = 0$ :

$$c(1, n+1) = \sum_{l=0}^{L-1} f(l+1)g(l - 2nM + 1)$$

If  $m$  is odd and less than  $M$

$$c(m+1, n+1) = \sqrt{2} \sum_{l=0}^{L-1} f(l+1) \sin\left(\pi \frac{m}{M} l\right) g(l - 2nM + 1)$$

$$c(m+M+1, n+1) = \sqrt{2} \sum_{l=0}^{L-1} f(l+1) \cos\left(\pi \frac{m}{M} l\right) g(l - (2n+1)M + 1)$$

If  $m$  is even and less than  $M$

$$c(m+1, n+1) = \sqrt{2} \sum_{l=0}^{L-1} f(l+1) \cos\left(\pi \frac{m}{M} l\right) g(l - 2nM + 1)$$

$$c(m+M+1, n+1) = \sqrt{2} \sum_{l=0}^{L-1} f(l+1) \sin\left(\pi \frac{m}{M} l\right) g(l - (2n+1)M + 1)$$

if  $m = M$  and  $M$  is even:

$$c(M+1, n+1) = \sum_{l=0}^{L-1} f(l+1) (-1)^l g(l - 2nM + 1)$$

else if  $m = M$  and  $M$  is odd

$$c(M+1, n+1) = \sum_{k=0}^{L-1} f(k+1) (-1)^k g(k - (2n+1)M + 1)$$

**References:** [12], [52], [24]

### 2.3.2 IDWILT - Inverse discrete Wilson transform

#### Usage

```
f=idwilt(c,g);
f=idwilt(c,g,Ls);
```

#### Input parameters

<b>c</b>	$M \times N$ array of coefficients.
<b>g</b>	Window function.
<b>Ls</b>	Final length of function (optional)

#### Output parameters

<b>f</b>	Input data
----------	------------

#### Description

`idwilt(c,g)` computes an inverse discrete Wilson transform with window  $g$ . The number of channels is deduced from the size of the coefficient array  $c$ .

The window  $g$  may be a vector of numerical values, a text string or a cell array. See the help of `wilwin` for more details.

`idwilt(f,g,Ls)` does the same, but cuts of zero-extend the final result to length  $Ls$ .

`[f,g]=idwilt(...)` additionally outputs the window used in the transform. This is usefull if the window was generated from a description in a string or cell array.

**References:** [12], [52]

### 2.3.3 DWILT2 - 2D Discrete Wilson transform

#### Usage

```
c=dwilt2(f,g,M);
c=dwilt2(f,g1,g2,[M1,M2]);
c=dwilt2(f,g1,g2,[M1,M2],[L1,L2]);
[c,Ls]=dwilt2(f,g1,g2,[M1,M2],[L1,L2]);
```

#### Input parameters

<b>f</b>	Input data, matrix.
<b>g, g1, g2</b>	Window functions.
<b>M, M1, M2</b>	Number of bands.
<b>L1, L2</b>	Length of transform to do.

#### Output parameters

<b>c</b>	array of coefficients.
<b>Ls</b>	Original size of input matrix.

### Description

`dwilt2(f, g, M)` calculates a two dimensional discrete Wilson transform of the input signal  $f$  using the window  $g$  and parameter  $M$  along each dimension.

For each dimension, the length of the transform will be the smallest possible that is larger than the length of the signal along that dimension.  $f$  will be appropriately zero-extended.

All windows must be whole-point even.

`dwilt2(f, g, M, L)` computes a Wilson transform as above, but does a transform of length  $L$  along each dimension.  $f$  will be cut or zero-extended to length  $L$  before the transform is done.

`[c, Ls] = dwilt(f, g, M)` or `[c, Ls] = dwilt(f, g, M, L)` additionally returns the length of the input signal  $f$ . This is handy for reconstruction.

`c = dwilt2(f, g1, g2, M)` makes it possible to use a different window along the two dimensions.

The parameters  $L$ ,  $M$  and  $Ls$  can also be vectors of length 2. In this case the first element will be used for the first dimension and the second element will be used for the second dimension.

The output  $c$  has 4 or 5 dimensions. The dimensions index the following properties:

1. Number of translations along 1st dimension of input.
2. Number of channels along 1st dimension of input
3. Number of translations along 2nd dimension of input.
4. Number of channels along 2nd dimension of input
5. Plane number, corresponds to 3rd dimension of input.

### Examples:

The following example visualize the `dwilt2` coefficients of a test image. For clarity, only the 50 dB largest coefficients are show::

```

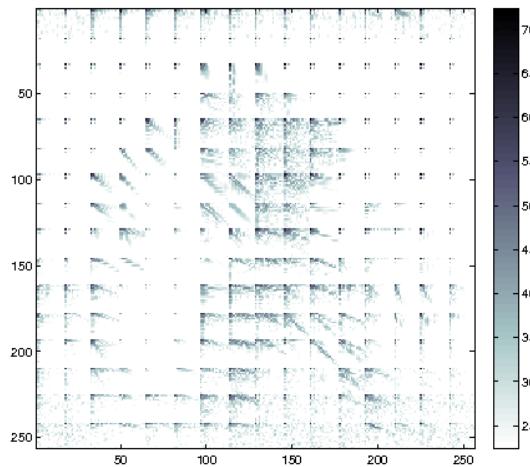
c=dwilt2(cameraman,'itersine',16);
c=reshape(c,256,256);

figure(1);
imagesc(cameraman), colormap(gray), axis('image');

figure(2);
cc=dynlimit(20*log10(abs(c)),50);
imagesc(cc), colormap(flipud(bone)), axis('image'), colorbar;

```





### 2.3.4 IDWILT2 - 2D Inverse Discrete Wilson transform

#### Usage

```
f=idwilt2(c,g);
f=idwilt2(c,g1,g2);
f=idwilt2(c,g1,g2,Ls);
```

#### Input parameters

<b>c</b>	Array of coefficients.
<b>g, g1, g2</b>	Window functions.
<b>Ls</b>	Size of reconstructed signal.

#### Output parameters

<b>f</b>	Output data, matrix.
----------	----------------------

#### Description

`idwilt2(c, g)` calculates a separable two dimensional inverse discrete Wilson transformation of the input coefficients  $c$  using the window  $g$ . The number of channels is deduced from the size of the coefficients  $c$ .

`idwilt2(c, g1, g2)` does the same using the window  $g1$  along the first dimension, and window  $g2$  along the second dimension.

`idwilt2(c, g1, g2, Ls)` cuts the signal to size  $Ls$  after the transformation is done.

### 2.3.5 WMDCT - Windowed MDCT transform

#### Usage

```
c=wmdct(f,g,M);
c=wmdct(f,g,M,L);
[c,Ls]=wmdct(...);
```

**Input parameters**

<b>f</b>	Input data
<b>g</b>	Window function.
<b>M</b>	Number of bands.
<b>L</b>	Length of transform to do.

**Output parameters**

<b>c</b>	$M \times N$ array of coefficients.
<b>Ls</b>	Length of input signal.

**Description**

wmdct (*f*, *g*, *M*) computes a Windowed Modified Discrete Cosine Transform with *M* bands and window *g*.

The length of the transform will be the smallest possible that is larger than the signal. *f* will be zero-extended to the length of the transform. If *f* is a matrix, the transformation is applied to each column. *g* must be whole-point even.

The window *g* may be a vector of numerical values, a text string or a cell array. See the help of wilwin for more details.

wmdct (*f*, *g*, *M*, *L*) computes the MDCT transform as above, but does a transform of length *L*. *f* will be cut or zero-extended to length *L* before the transform is done.

[*c*, *Ls*] =wmdct (*f*, *g*, *M*) or [*c*, *Ls*] =wmdct(*f*, *g*, *M*, *L*)' additionally returns the length of the input signal *f*. This is handy for reconstruction:

```
[c, Ls]=wmdct (f, g, M);
fr=iwmdct (c, gd, M, Ls);
```

will reconstruct the signal *f* no matter what the length of *f* is, provided that *gd* is a dual Wilson window of *g*.

[*c*, *Ls*, *g*] =wmdct (...) additionally outputs the window used in the transform. This is useful if the window was generated from a description in a string or cell array.

The WMDCT is sometimes known as an odd-stacked cosine modulated filter bank. The WMDCT defined by this routine is slightly different from the most common definition of the WMDCT, in order to be able to use the same window functions as the Wilson transform.

Assume that the following code has been executed for a column vector *f* of length *L*:

```
c=wmdct (f, g, M); % Compute the WMDCT of f.
N=size(c, 2); % Number of translation coefficients.
```

The following holds for  $m = 0, \dots, M - 1$  and  $n = 0, \dots, N - 1$ :

If  $m + n$  is even:

$$c(m+1, n+1) = \sqrt{2} \sum_{l=0}^{L-1} f(l+1) \cos \left( \frac{\pi}{M} \left( m + \frac{1}{2} \right) l + \frac{\pi}{4} \right) g(l - nM + 1)$$

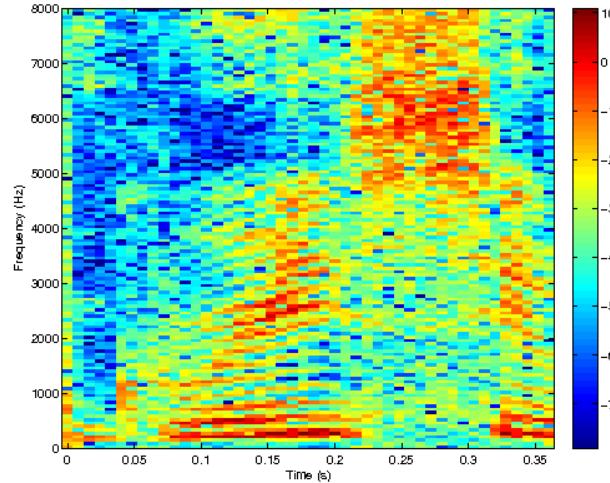
If  $m + n$  is odd:

$$c(m+1, n+1) = \sqrt{2} \sum_{l=0}^{L-1} f(l+1) \sin \left( \frac{\pi}{M} \left( m + \frac{1}{2} \right) l + \frac{\pi}{4} \right) g(l - nM + 1)$$

**Examples:**

The following example shows the WMDCT coefficients (128 channels) of the greasy test signal:

```
fs=16000; % Sampling rate
c=wmdct(greasy,{'hann',0.02*fs},128);
plotwmdct(c,fs,90);
```



Compare the visual difference with the redundant expansion of the same signal given in the example of the dgtreal function.

**References:** [66], [67], [58], [13]

### 2.3.6 IWMDCT - Inverse MDCT

#### Usage

```
f=iwmdct(c,g);
f=iwmdct(c,g,Ls);
```

#### Input parameters

<b>c</b>	M*N array of coefficients.
<b>g</b>	Window function.
<b>Ls</b>	Final length of function (optional)

#### Output parameters

<b>f</b>	Input data
----------	------------

#### Description

`iwmdct (c, g)` computes an inverse windowed MDCT with window  $g$ . The number of channels is deduced from the size of the coefficient array  $c$ .

The window  $g$  may be a vector of numerical values, a text string or a cell array. See the help of `wilwin` for more details.

`iwmdct (f, g, Ls)` does the same, but cuts or zero-extends the final result to length  $Ls$ .

`[f, g]=iwmdct(...)` additionally outputs the window used in the transform. This is useful if the window was generated from a description in a string or cell array.

**References:** [66], [67], [58], [13]

### 2.3.7 WMDCT2 - 2D Discrete windowed MDCT transform

#### Usage

```
c=wmdct2(f,g,M);
c=wmdct2(f,g1,g2,[M1,M2]);
c=wmdct2(f,g1,g2,[M1,M2],[L1,L2]);
[c,L]=wmdct2(f,g1,g2,[M1,M2],[L1,L2]);
```

#### Input parameters

<b>f</b>	Input data, matrix.
<b>g, g1, g2</b>	Window functions.
<b>M, M1, M2</b>	Number of bands.
<b>L1, L2</b>	Length of transform to do.

#### Output parameters

<b>c</b>	array of coefficients.
<b>Ls</b>	Original size of input matrix.

#### Description

wmdct2 (*f*, *g*, *M*) calculates a two dimensional Modified Discrete Cosine transform of the input signal *f* using the window *g* and parameter *M* along each dimension.

For each dimension, the length of the transform will be the smallest possible that is larger than the length of the signal along that dimension. *f* will be appropriately zero-extended.

All windows must be whole-point even.

wmdct2 (*f*, *g*, *M*, *L*) computes a 2D windowed MDCT as above, but does a transform of length *L* along each dimension. *f* will be cut or zero-extended to length *L* before the transform is done.

[*c*, *Ls*] = wmdct2 (*f*, *g*, *M*) or [*c*, *Ls*] = wmdct2 (*f*, *g*, *M*, *L*) additionally return the length of the input signal *f*. This is handy for reconstruction.

*c*=wmdct2 (*f*, *g1*, *g2*, *M*) makes it possible to use different windows along the two dimensions.

The parameters *L*, *M* and *Ls* can also be vectors of length 2. In this case the first element will be used for the first dimension and the second element will be used for the second dimension.

The output *c* has 4 or 5 dimensions. The dimensions index the following properties:

1. Number of translation along 1st dimension of input.
2. Number of channel along 1st dimension of input
3. Number of translation along 2nd dimension of input.
4. Number of channel along 2nd dimension of input
5. Plane number, corresponds to 3rd dimension of input.

### 2.3.8 IWMDCT2 - 2D Inverse windowed MDCT transform

#### Usage

```
f=iwmdct2(c,g);
f=iwmdct2(c,g1,g2);
f=iwmdct2(c,g1,g2,Ls);
```

**Input parameters**

<b>c</b>	Array of coefficients.
<b>g, g1, g2</b>	Window functions.
<b>Ls</b>	Size of reconstructed signal.

**Output parameters**

<b>f</b>	Output data, matrix.
----------	----------------------

**Description**

`iwmdct2(c, g)` calculates a separable two dimensional inverse wmdct transform of the input coefficients  $c$  using the window  $g$ . The number of channels is deduced from the size of the coefficients  $c$ .

`iwmdct2(c, g1, g2)` does the same using the window  $g1$  along the first dimension, and window  $g2$  along the second dimension.

`iwmdct2(c, g1, g2, Ls)` cuts the signal to size  $Ls$  after the transform is done.

**2.3.9 WIL2RECT - Arrange Wilson coefficients in a rectangular layout****Usage**

```
c=wil2rect(c);
```

**Description**

`wil2rect(c)` rearranges the coefficients  $c$  in a rectangular shape. The coefficients must have been obtained from dwilt. After rearrangement the coefficients are placed correctly on the time/frequency-plane.

The rearranged array is larger than the input array: it contains zeros on the spots where the Wilson transform is missing a DC or Nyquest component.

**2.3.10 RECT2WIL - Inverse of WIL2RECT****Usage**

```
c=rect2wil(c);
```

**Description**

`rect2wil(c)` takes Wilson coefficients processed by `wil2rect` and puts them back in the compact form used by `dwilt` and `idwilt`. The coefficients can then be processed by `idwilt`.

**2.3.11 WILWIN - Compute a Wilson/WMDCT window from text or cell array****Usage**

```
[g, info] = wilwin(g, M, L);
```

**Description**

`[g, info]=wilwin(g, M, L)` computes a window that fits well with the specified number of channels  $M$  and transform length  $L$ . The window itself is specified by a text description or a cell array containing additional parameters.

The window can be specified directly as a vector of numerical values. In this case, `wilwin` only checks assumptions about transform sizes etc.

`[g, info]=wilwin(g, M)` does the same, but the window must be a FIR window, as the transform length is unspecified.

The window can be specified as one of the following text strings:

<b>'gauss'</b>	Gaussian window with optimal concentration
<b>'dualgauss'</b>	Riesz dual of Gaussian window with optimal concentration.
<b>'tight'</b>	Window generating an orthonormal basis

In these cases, a long window is generated with a length of  $L$ .

It is also possible to specify one of the window names from firwin. In such a case, `wilwin` generates the specified FIR window with a length of  $M$ .

The window can also be specified as cell array. The possibilities are:

- {'gauss', ...} Additional parameters are passed to PGAUSS
- {'dual', ...} Dual window of whatever follows. See the examples below.
- {'tight', ...} Orthonormal window of whatever follows.

It is also possible to specify one of the window names from firwin as the first field in the cell array. In this case, the remaining entries of the cell array are passed directly to firwin.

Some examples:

```
g=wilwin('gauss',M,L);
```

This computes a Gaussian window of length  $L$  fitted for a system with  $M$  channels.

```
g=wilwin({'gauss',1},M,L);
```

This computes a Gaussian window with equal time and frequency support irrespective of  $M$ .

```
gd=wilwin('gaussdual',M,L);
```

This computes the dual of a Gaussian window fitted for a system with  $M$  channels.

```
gd=wilwin({'tight','gauss'},M,L);
```

This computes the orthonormal window of the Gaussian window fitted for the system.

```
g=wilwin({'dual',{'hann',20}},M,L);
```

This computes the dual of a Hann window of length 20.

The structure `info` provides some information about the computed window:

- info.gauss** True if the window is a Gaussian.
- info.tfr** Time/frequency support ratio of the window. Set whenever it makes sense.
- info.wasrow** Input was a row window
- info.isfir** Input is an FIR window
- info.isdual** Output is the dual window of the auxiliary window.
- info.istight** Output is known to be a tight window.
- info.auxinfo** Info about auxiliary window.
- info.g1** Length of window.

### 2.3.12 DWILTLENGTH - DWILT/WMDCT length from signal

#### Usage

```
L=dwiltlength(Ls,M);
```

### Description

`dwiltlength(Ls, M)` returns the length of a Wilson / WMDCT system with  $M$  channels system is long enough to expand a signal of length  $L_s$ . Please see the help on `dwilt` or `wmdct` for an explanation of the parameter  $M$ .

If the returned length is longer than the signal length, the signal will be zero-padded by `dwilt` or `wmdct`.

A valid transform length must be divisible by  $2M$ . This means that the minimal admissible transform length is

```
Lsmallest = 2*M;
```

and all valid transform lengths are multiples of  $L_{smallest}$

## 2.4 Reconstructing windows

### 2.4.1 GABDUAL - Canonical dual window of Gabor frame

#### Usage

```
gd=gabdual(g,a,M);
gd=gabdual(g,a,M,L);
gd=gabdual(g,a,M,'lt',lt);
```

#### Input parameters

<b>g</b>	Gabor window.
<b>a</b>	Length of time shift.
<b>M</b>	Number of channels.
<b>L</b>	Length of window. (optional)
<b>lt</b>	Lattice type (for non-separable lattices).

#### Output parameters

<b>gd</b>	Canonical dual window.
-----------	------------------------

#### Description

`gabdual(g, a, M)` computes the canonical dual window of the discrete Gabor frame with window  $g$  and parameters  $a, M$ .

The window  $g$  may be a vector of numerical values, a text string or a cell array. See the help of `gabwin` for more details.

If the length of  $g$  is equal to  $M$ , then the input window is assumed to be an FIR window. In this case, the canonical dual window also has length of  $M$ . Otherwise the smallest possible transform length is chosen as the window length.

`gabdual(g, a, M, L)` returns a window that is the dual window for a system of length  $L$ . Unless the dual window is a FIR window, the dual window will have length  $L$ .

`gabdual(g, a, M, 'lt', lt)` does the same for a non-separable lattice specified by  $lt$ . Please see the help of `matrix2latticetype` for a precise description of the parameter  $lt$ .

If  $a > M$  then the dual window of the Gabor Riesz sequence with window  $g$  and parameters  $a$  and  $M$  will be calculated.

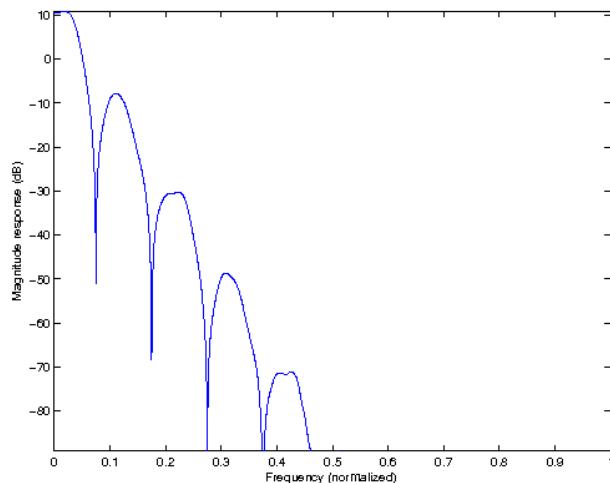
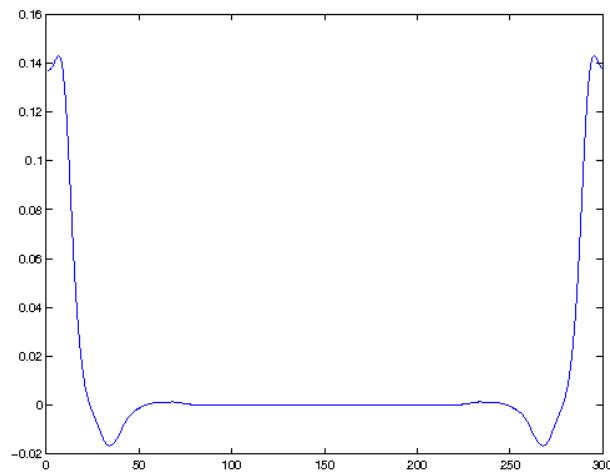
**Examples:**

The following example shows the canonical dual window of the Gaussian window:

```
a=20;
M=30;
L=300;
g=pgauss(L,a*M/L);
gd=gabdual(g,a,M);

% Simple plot in the time-domain
figure(1);
plot(gd);

% Frequency domain
figure(2);
magresp(gd,'dynrange',100);
```



### 2.4.2 GABTIGHT - Canonical tight window of Gabor frame

**Usage**

```
gt=gabtight(a,M,L);
```

```
gt=gabtight (g,a,M);
gt=gabtight (g,a,M,L);
gd=gabtight (g,a,M,'lt',lt);
```

### Input parameters

<b>g</b>	Gabor window.
<b>a</b>	Length of time shift.
<b>M</b>	Number of modulations.
<b>L</b>	Length of window. (optional)
<b>lt</b>	Lattice type (for non-separable lattices).

### Output parameters

<b>gt</b>	Canonical tight window, column vector.
-----------	--

### Description

`gabtight (a, M, L)` computes a nice tight window of length  $L$  for a lattice with parameters  $a$ ,  $M$ . The window is not an FIR window, meaning that it will only generate a tight system if the system length is equal to  $L$ .

`gabtight (g, a, M)` computes the canonical tight window of the Gabor frame with window  $g$  and parameters  $a$ ,  $M$ .

The window  $g$  may be a vector of numerical values, a text string or a cell array. See the help of `gabwin` for more details.

If the length of  $g$  is equal to  $M$ , then the input window is assumed to be a FIR window. In this case, the canonical dual window also has length of  $M$ . Otherwise the smallest possible transform length is chosen as the window length.

`gabtight (g, a, M, L)` returns a window that is tight for a system of length  $L$ . Unless the input window  $g$  is a FIR window, the returned tight window will have length  $L$ .

`gabtight (g, a, M, 'lt', lt)` does the same for a non-separable lattice specified by  $lt$ . Please see the help of `matrix2latticetype` for a precise description of the parameter  $lt$ .

If  $a > M$  then an orthonormal window of the Gabor Riesz sequence with window  $g$  and parameters  $a$  and  $M$  will be calculated.

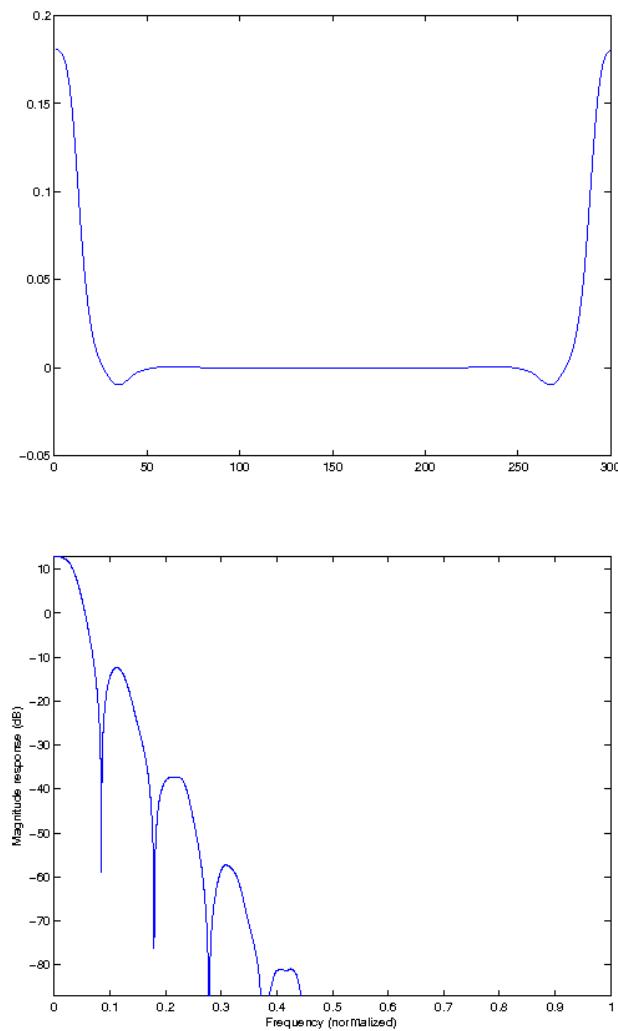
### Examples:

The following example shows the canonical tight window of the Gaussian window. This is calculated by default by `gabtight` if no window is specified:

```
a=20;
M=30;
L=300;
gt=gabtight (a,M,L);

% Simple plot in the time-domain
figure(1);
plot(gt);

% Frequency domain
figure(2);
magresp(gt,'dynrange',100);
```



### 2.4.3 GABFIRDUAL - Compute FIR dual window

#### Usage

```
gd=gabfirdual(Ldual,g,a,M);
gd=gabfirdual(Ldual,g,a,M,varargin);
```

#### Input parameters

<b>Ldual</b>	Length of dual window
<b>g</b>	Window function
<b>a</b>	Time shift
<b>M</b>	Number of Channels
<b>alpha1</b>	Weight of $l^1$ -norm in the time domain
<b>alpha2</b>	Weight of $l^1$ -norm in the freq. domain

#### Output parameters

<b>gd</b>	Dual window
-----------	-------------

### Description

`gabfirdual(Ldual, g, a, M)` computes an FIR window  $gd$  which is an approximate dual window of the Gabor system defined by  $g$ ,  $a$  and  $M$ . The FIR dual window will be supported on  $Ldual$  samples.

This function solve a convex optimization problem that can be written as:

$$\begin{aligned} gd = & \arg \min_x \|\alpha x\|_1 + \|\beta \mathcal{F}x\|_1 \\ & + \|\omega(x - g_l)\|_2^2 \\ & \delta \|x\|_{S0} + \mu \|\nabla x\|_2^2 + \gamma \|\nabla \mathcal{F}x\|_2^2 \end{aligned}$$

such that  $x$  is a dual window of  $g$

**Note:** This function require the unlocbox. You can download it at <http://unlocbox.sourceforge.net>

The function uses an iterative algorithm to compute the approximate FIR dual. The algorithm can be controlled by the following flags:

<b>'alpha'</b> , <b>alpha</b>	Weight in time. If it is a scalar, it represent the weights of the entire L1 function in time. If it is a vector, it is the associated weight assotiated to each component of the L1 norm (length: $Ldual$ ). Default value is $\alpha = 0$ . <b>Warning:</b> this value should not be too big in order to avoid the the L1 norm proximal operator kill the signal. No L1-time constraint: $\alpha = 0$
<b>'beta'</b> , <b>beta</b>	Weight in frequency. If it is a scalar, it represent the weights of the entire L1 function in frequency. If it is a vector, it is the associated weight assotiated to each component of the L1 norm in frequency. (length: $Ldual$ ). Default value is $\beta = 0$ . <b>Warning:</b> this value should not be too big in order to avoid the the L1 norm proximal operator kill the signal. No L1-frequency constraint: $\beta = 0$
<b>'omega'</b> , <b>omega</b>	Weight in time of the L2-norm. If it is a scalar, it represent the weights of the entire L2 function in time. If it is a vector, it is the associated weight assotiated to each component of the L2 norm (length: $Ldual$ ). Default value is $\omega = 0$ . No L2-time constraint: $\omega = 0$
<b>'glike'</b> , <b>g_l</b>	$g_l$ is a windows in time. The algorithm try to shape the dual window like $g_l$ . Normalization of $g_l$ is done automatically. To use option omega should be different from 0. By default $g_d = 0$ .
<b>'mu'</b> , <b>mu</b>	Weight of the smooth constraint Default value is 1. No smooth constraint: $\mu = 0$
<b>'gamma'</b> , <b>gamma</b>	Weight of the smooth constraint in frequency. Default value is 1. No smooth constraint: $\gamma = 0$
<b>'delta'</b> , <b>delta</b>	Weight of the S0-norm. Default value is 0. No S0-norm: $\delta = 0$
<b>'dual'</b>	Look for a dual windows (default)
<b>'painless'</b>	Construct a starting guess using a painless-case approximation. This is the default
<b>'zero'</b>	Choose a starting guess of zero.
<b>'rand'</b>	Choose a random starting phase.

<b>'tol',t</b>	Stop if relative residual error is less than the specified tolerance.
<b>'maxit',n</b>	Do at most n iterations. default 200
<b>'print'</b>	Display the progress.
<b>'debug'</b>	Display all the progresses.
<b>'quiet'</b>	Don't print anything, this is the default.
<b>'fast'</b>	Fast algorithm, this is the default.
<b>'slow'</b>	Safer algorithm, you can try this if the fast algorithm is not working. Before using this, try to iterate more.
<b>'printstep',p</b>	If 'print' is specified, then print every p'th iteration. Default value is p=10;
<b>'hardconstraint'</b>	Force the projection at the end (default)
<b>'softconstraint'</b>	Do not force the projection at the end

#### 2.4.4 GABOPTDUAL - Compute dual window

##### Usage

```
gd=gaboptdual(g,a,M);
gd=gaboptdual(g,a,M, varargin);
```

##### Input parameters

<b>g</b>	Window function
<b>a</b>	Time shift
<b>M</b>	Number of Channels

##### Output parameters

<b>gd</b>	Dual window
-----------	-------------

##### Description

`gaboptdual(g,a,M)` computes a window  $gd$  which is an approximate dual window of the Gabor system defined by  $g$ ,  $a$  and  $M$ .

This function solve a convex optimization problem that can be written as:

$$\begin{aligned} gd = & \arg \min_x \|\alpha x\|_1 + \|\beta \mathcal{F}x\|_1 \\ & + \|\omega(x - g_l)\|_2^2 \\ & \delta \|x\|_{S_0} + \mu \|\nabla x\|_2^2 + \gamma \|\nabla \mathcal{F}x\|_2^2 \end{aligned}$$

such that  $x$  is a dual window of  $g$

**Note:** This function require the unlocbox. You can download it at <http://unlocbox.sourceforge.net>

The function uses an iterative algorithm to compute the approximate optimized dual. The algorithm can be controlled by the following flags:

<b>'alpha',alpha</b>	Weight in time. If it is a scalar, it represent the weights of the entire L1 function in time. If it is a vector, it is the associated weight assotiated to each component of the L1 norm (length: Ldual). Default value is $\alpha = 0$ . <b>Warning:</b> this value should not be too big in order to avoid the the L1 norm proximal operator kill the signal. No L1-time constraint: $\alpha = 0$
<b>'beta',beta</b>	Weight in frequency. If it is a scalar, it represent the weights of the entire L1 function in frequency. If it is a vector, it is the associated weight assotiated to each component of the L1 norm in frequency. (length: Ldual). Default value is $\beta = 0$ . <b>Warning:</b> this value should not be too big in order to avoid the the L1 norm proximal operator kill the signal. No L1-frequency constraint: $\beta = 0$
<b>'omega',omega</b>	Weight in time of the L2-norm. If it is a scalar, it represent the weights of the entire L2 function in time. If it is a vector, it is the associated weight assotiated to each component of the L2 norm (length: Ldual). Default value is $\omega = 0$ . No L2-time constraint: $\omega = 0$
<b>'glike',g_l</b>	$g_l$ is a windows in time. The algorithm try to shape the dual window like $g_l$ . Normalization of $g_l$ is done automatically. To use option omega should be different from 0. By default $g_d = 0$ .
<b>'mu',mu</b>	Weight of the smooth constraint Default value is 1. No smooth constraint: $\mu = 0$
<b>'gamma',gamma</b>	Weight of the smooth constraint in frequency. Default value is 1. No smooth constraint: $\gamma = 0$
<b>'delta',delta</b>	Weight of the S0-norm. Default value is 0. No S0-norm: $\delta = 0$
<b>'dual'</b>	Look for a dual windows (default)
<b>'painless'</b>	Construct a starting guess using a painless-case approximation. This is the default
<b>'zero'</b>	Choose a starting guess of zero.
<b>'rand'</b>	Choose a random starting phase.
<b>'tol',t</b>	Stop if relative residual error is less than the specified tolerance.
<b>'maxit',n</b>	Do at most n iterations. default 200
<b>'print'</b>	Display the progress.
<b>'debug'</b>	Display all the progresses.
<b>'quiet'</b>	Don't print anything, this is the default.
<b>'fast'</b>	Fast algorithm, this is the default.
<b>'slow'</b>	Safer algorithm, you can try this if the fast algorithm is not working. Before using this, try to iterate more.
<b>'printstep',p</b>	If 'print' is specified, then print every p'th iteration. Default value is p=10;
<b>'hardconstraint'</b>	Force the projection at the end (default)
<b>'softconstraint'</b>	Do not force the projection at the end

### 2.4.5 GABFIRTIGHT - Compute FIR tight window

#### Usage

```
gt=gabfirtight (Lsupport, g, a, M) ;
gt=gabfirtight (Lsupport, g, a, M, varargin) ;
```

#### Input parameters

<b>Lsupport</b>	Length of the tight window
<b>g</b>	Initial window function
<b>a</b>	Time shift
<b>M</b>	Number of Channels

#### Output parameters

<b>gt</b>	Tight window
-----------	--------------

#### Description

`gabfirtight (Lsupport, g, a, M)` computes an FIR window  $gd$  which is tight. The FIR dual window will be supported on  $Lsupport$  samples.

This function solve a convex optimization problem that can be written as:

$$\begin{aligned} gd = & \arg \min_x \|\alpha x\|_1 + \|\beta \mathcal{F}x\|_1 \\ & + \|\omega(x - g_l)\|_2^2 \\ & \delta \|x\|_{S_0} + \mu \|\nabla x\|_2^2 + \gamma \|\nabla \mathcal{F}x\|_2^2 \end{aligned}$$

such that  $x$  is a tight FIR window

**Note:** This function require the unlocbox. You can download it at <http://unlocbox.sourceforge.net>

The function uses an iterative algorithm to compute the approximate FIR tight windows. Warning The algorithm solve a non convex problem and might be stuck in bad local minima. The algorithm can be controlled by the following flags:

<b>'alpha',alpha</b>	Weight in time. If it is a scalar, it represent the weights of the entire L1 function in time. If it is a vector, it is the associated weight associated to each component of the L1 norm (length: Ldual). Default value is $\alpha = 0$ . <b>Warning:</b> this value should not be too big in order to avoid the the L1 norm proximal operator kill the signal. No L1-time constraint: $\alpha = 0$
<b>'beta',beta</b>	Weight in frequency. If it is a scalar, it represent the weights of the entire L1 function in frequency. If it is a vector, it is the associated weight associated to each component of the L1 norm in frequency. (length: Ldual). Default value is $\beta = 0$ . <b>Warning:</b> this value should not be too big in order to avoid the the L1 norm proximal operator kill the signal. No L1-frequency constraint: $\beta = 0$

<b>'omega',omega</b>	Weight in time of the L2-norm. If it is a scalar, it represent the weights of the entire L2 function in time. If it is a vector, it is the associated weight assotiated to each component of the L2 norm (length: Ldual). Default value is $\omega = 0$ . No L2-time constraint: $\omega = 0$
<b>'glike',g_l</b>	$g_l$ is a windows in time. The algorithm try to shape the dual window like $g_l$ . Normalization of $g_l$ is done automatically. To use option omega should be different from 0. By default $g_d = 0$ .
<b>'mu',mu</b>	Weight of the smooth constraint Default value is 1. No smooth constraint: $\mu = 0$
<b>'gamma',gamma</b>	Weight of the smooth constraint in frequency. Default value is 1. No smooth constraint: $\gamma = 0$
<b>'delta',delta</b>	Weight of the S0-norm. Default value is 0. No S0-norm: $\delta = 0$
<b>'dual'</b>	Look for a dual windows (default)
<b>'painless'</b>	Construct a starting guess using a painless-case approximation. This is the default
<b>'zero'</b>	Choose a starting guess of zero.
<b>'rand'</b>	Choose a random starting phase.
<b>'tol',t</b>	Stop if relative residual error is less than the specified tolerance.
<b>'maxit',n</b>	Do at most n iterations. default 200
<b>'print'</b>	Display the progress.
<b>'debug'</b>	Display all the progresses.
<b>'quiet'</b>	Don't print anything, this is the default.
<b>'fast'</b>	Fast algorithm, this is the default.
<b>'slow'</b>	Safer algorithm, you can try this if the fast algorithm is not working. Before using this, try to iterate more.
<b>'printstep',p</b>	If 'print' is specified, then print every p'th iteration. Default value is p=10;
<b>'hardconstraint'</b>	Force the projection at the end (default)
<b>'softconstraint'</b>	Do not force the projection at the end

#### 2.4.6 GABOPTTIGHT - Compute a optimized tight window

##### Usage

```
gt=gabopttight(Ltight,g,a,M);
gt=gabopttight(Ltight,g,a,M, varargin);
```

##### Input parameters

<b>g</b>	Initial window function
<b>a</b>	Time shift
<b>M</b>	Number of Channels

### Output parameters

<b>gt</b>	Tight window
-----------	--------------

### Description

`gabopttight(g, a, M)` computes a tight window  $gt$  for a frame of parameter  $a$  and  $M$

This function solves a convex optimization problem that can be written as:

$$\begin{aligned} gd = \arg \min_x & \| \alpha x \|_1 + \| \beta \mathcal{F}x \|_1 \\ & + \| \omega(x - g_l) \|_2^2 \\ & \delta \| x \|_{S0} + \mu \| \nabla x \|_2^2 + \gamma \| \nabla \mathcal{F}x \|_2^2 \end{aligned}$$

such that  $x$  is tight window

**Note:** This function require the unlocbox. You can download it at <http://unlocbox.sourceforge.net>

The function uses an iterative algorithm to compute the approximate optimized tight window. Warning The algorithm solve a non convex problem and might be stuck in bad local minima. The algorithm can be controlled by the following flags:

<b>'alpha',alpha</b>	Weight in time. If it is a scalar, it represent the weights of the entire L1 function in time. If it is a vector, it is the associated weight assotiated to each component of the L1 norm (length: Ldual). Default value is $\alpha = 0$ . <b>Warning:</b> this value should not be too big in order to avoid the the L1 norm proximal operator kill the signal. No L1-time constraint: $\alpha = 0$
----------------------	--

<b>'beta',beta</b>	Weight in frequency. If it is a scalar, it represent the weights of the entire L1 function in frequency. If it is a vector, it is the associated weight assotiated to each component of the L1 norm in frequency. (length: Ldual). Default value is $\beta = 0$ . <b>Warning:</b> this value should not be too big in order to avoid the the L1 norm proximal operator kill the signal. No L1-frequency constraint: $\beta = 0$
--------------------	---

<b>'omega',omega</b>	Weight in time of the L2-norm. If it is a scalar, it represent the weights of the entire L2 function in time. If it is a vector, it is the associated weight assotiated to each component of the L2 norm (length: Ldual). Default value is $\omega = 0$ . No L2-time constraint: $\omega = 0$
----------------------	---

<b>'glike',g_l</b>	$g_l$ is a windows in time. The algorithm try to shape the dual window like $g_l$ . Normalization of $g_l$ is done automatically. To use option omega should be different from 0. By default $g_d = 0$ .
--------------------	--

**'mu', mu Weight of the smooth constraint Default value is 1.** No smooth constraint:  $\mu = 0$

**'gamma', gamma Weight of the smooth constraint in frequency. Default value is 1.** No smooth constraint:  $\gamma = 0$

**'delta', delta Weight of the S0-norm. Default value is 0.** No S0-norm:  $\delta = 0$

<b>'dual'</b>	Look for a dual windows (default)
---------------	-----------------------------------

<b>'painless'</b>	Construct a starting guess using a painless-case approximation. This is the default
-------------------	---

<b>'zero'</b>	Choose a starting guess of zero.
<b>'rand'</b>	Choose a random starting phase.
<b>'tol',t</b>	Stop if relative residual error is less than the specified tolerance.
<b>'maxit',n</b>	Do at most n iterations. default 200
<b>'print'</b>	Display the progress.
<b>'debug'</b>	Display all the progresses.
<b>'quiet'</b>	Don't print anything, this is the default.
<b>'fast'</b>	Fast algorithm, this is the default.
<b>'slow'</b>	Safer algorithm, you can try this if the fast algorithm is not working. Before using this, try to iterate more.
<b>'printstep',p</b>	If 'print' is specified, then print every p'th iteration. Default value is p=10;
<b>'hardconstraint'</b>	Force the projection at the end (default)
<b>'softconstraint'</b>	Do not force the projection at the end

#### 2.4.7 GABCONVEXOPT - Compute a window using convex optimization

##### Usage

```
gout=gabconvexopt (g,a,M);
gout=gabconvexopt (g,a,M, varargin);
```

##### Input parameters

<b>g</b>	Window function /initial point (tight case)
<b>a</b>	Time shift
<b>M</b>	Number of Channels

##### Output parameters

<b>gout</b>	Output window
<b>iter</b>	Number of iterations
<b>relres</b>	Reconstruction error

##### Description

`gabconvexopt (g, a, M)` computes a window *gout* which is the optimal solution of the convex optimization problem below

$$gd = \arg \min_x \|\alpha x\|_1 + \|\beta \mathcal{F}x\|_1$$

$$+ \|\omega(x - g_l)\|_2^2$$

$$\delta \|x\|_{S0} + \mu \|\nabla x\|_2^2 + \gamma \|\nabla \mathcal{F}x\|_2^2$$

such that *x* satisfies the constraints

Three constraints are possible:

- $x$  is dual with respect of  $g$
- $x$  is tight
- $x$  is compactly supported on  $L_{dual}$

**Note:** This function require the unlocbox. You can download it at <http://unlocbox.sourceforge.net>

The function uses an iterative algorithm to compute the approximate. The algorithm can be controlled by the following flags:

<b>'alpha',alpha</b>	Weight in time. If it is a scalar, it represent the weights of the entire L1 function in time. If it is a vector, it is the associated weight assotiated to each component of the L1 norm (length: $L_{dual}$ ). Default value is $\alpha = 0$ . <b>Warning:</b> this value should not be too big in order to avoid the the L1 norm proximal operator kill the signal. No L1-time constraint: $\alpha = 0$
<b>'beta',beta</b>	Weight in frequency. If it is a scalar, it represent the weights of the entire L1 function in frequency. If it is a vector, it is the associated weight assotiated to each component of the L1 norm in frequency. (length: $L_{dual}$ ). Default value is $\beta = 0$ . <b>Warning:</b> this value should not be too big in order to avoid the the L1 norm proximal operator kill the signal. No L1-frequency constraint: $\beta = 0$
<b>'omega',omega</b>	Weight in time of the L2-norm. If it is a scalar, it represent the weights of the entire L2 function in time. If it is a vector, it is the associated weight assotiated to each component of the L2 norm (length: $L_{dual}$ ). Default value is $\omega = 0$ . No L2-time constraint: $\omega = 0$
<b>'glike',g_l</b>	$g_l$ is a windows in time. The algorithm try to shape the dual window like $g_l$ . Normalization of $g_l$ is done automatically. To use option omega should be different from 0. By default $g_d = 0$ .

**'mu', mu Weight of the smooth constraint Default value is 1.** No smooth constraint:  $\mu = 0$

**'gamma', gamma Weight of the smooth constraint in frequency. Default value is 1.** No smooth constraint:  $\gamma = 0$

**'delta', delta Weight of the S0-norm. Default value is 0.** No S0-norm:  $\delta = 0$

<b>'support'</b>	$L_{dual}$ Add a constraint on the support. The windows should be compactly supported on $L_{dual}$ .
<b>'tight'</b>	Look for a tight windows
<b>'dual'</b>	Look for a dual windows (default)
<b>'painless'</b>	Construct a starting guess using a painless-case approximation. This is the default
<b>'zero'</b>	Choose a starting guess of zero.
<b>'rand'</b>	Choose a random starting phase.
<b>'tol',t</b>	Stop if relative residual error is less than the specified tolerance.
<b>'maxit',n</b>	Do at most n iterations. default 200
<b>'print'</b>	Display the progress.

<b>'debug'</b>	Display all the progresses.
<b>'quiet'</b>	Don't print anything, this is the default.
<b>'fast'</b>	Fast algorithm, this is the default.
<b>'slow'</b>	Safer algorithm, you can try this if the fast algorithm is not working. Before using this, try to iterate more.
<b>'printstep',p</b>	If 'print' is specified, then print every p'th iteration. Default value is p=10;
<b>'hardconstraint'</b>	Force the projection at the end (default)
<b>'softconstraint'</b>	Do not force the projection at the end

### 2.4.8 GABPROJDUAL - Gabor Dual window by projection

#### Usage

```
gd=gabprojdual(gm,g,a,M)
gd=gabprojdual(gm,g,a,M,L)
```

#### Input parameters

<b>gm</b>	Window to project.
<b>g</b>	Window function.
<b>a</b>	Length of time shift.
<b>M</b>	Number of modulations.
<b>L</b>	Length of transform to consider

#### Output parameters

<b>gd</b>	Dual window.
-----------	--------------

#### Description

`gabprojdual(gm,g,a,M)` calculates the dual window of the Gabor frame given by  $g$ ,  $a$  and  $M$  closest to  $gm$  measured in the  $l^2$  norm. The function projects the suggested window  $gm$  onto the subspace of admissible dual windows, hence the name of the function.

`gabprojdual(gm,g,a,M,L)` first extends the windows  $g$  and  $gm$  to length  $L$ .

`gabprojdual(...,'lt',lt)` does the same for a non-separable lattice specified by  $lt$ . Please see the help of `matrix2latticetype` for a precise description of the parameter  $lt$ .

### 2.4.9 GABMIXDUAL - Computes the mixdual of g1

#### Usage

```
gamma=mixdual(g1,g2,a,M)
```

#### Input parameters

<b>g1</b>	Window 1
<b>g2</b>	Window 2
<b>a</b>	Length of time shift.
<b>M</b>	Number of modulations.

**Output parameters**

**gammaf** Mixdual of window 1.

**Description**

`gabmixdual(g1, g2, a, M)` computes a dual window of  $g1$  from a mix of the canonical dual windows of  $g1$  and  $g2$ .

**References:** [85]

**2.4.10 WILORTH - Wilson orthonormal window****Usage**

```
gt=wilorth(M, L);
gt=wilorth(g, M);
gt=wilorth(g, M, L);
```

**Input parameters**

**g** Auxiliary window window function (optional).

**M** Number of modulations.

**L** Length of window (optional).

**Output parameters**

**gt** Window generating an orthonormal Wilson basis.

**Description**

`wilorth(M, L)` computes a nice window of length  $L$  generating an orthonormal Wilson or WMDCT basis with  $M$  frequency bands for signals of length  $L$ .

`wilorth(g, M)` computes a window generating an orthonormal basis from the window  $g$  and number of channels  $M$ .

The window  $g$  may be a vector of numerical values, a text string or a cell array. See the help of `wilwin` for more details.

If the length of  $g$  is equal to  $2 \times M$ , then the input window is assumed to be a FIR window. In this case, the orthonormal window also has length of  $2 \times M$ . Otherwise the smallest possible transform length is chosen as the window length.

`wilorth(g, M, L)` pads or truncates  $g$  to length  $L$  before calculating the orthonormal window. The output will also be of length  $L$ .

The input window  $g$  must be real whole-point even. If  $g$  is not whole-point even, the computed window will not generate an orthonormal system (i.e. reconstruction will not be perfect). For a random window  $g$ , the window closest to  $g$  that satisfies these restrictions can be found by

```
g_wpe = real(peven(g));
```

All Gabor windows in the toolbox satisfies these restrictions unless clearly stated otherwise.

**2.4.11 WILDUAL - Wilson dual window****Usage**

```
gamma=wildual(g, M);
gamma=wildual(g, M, L);
```

**Input parameters**

<b>g</b>	Gabor window.
<b>M</b>	Number of modulations.
<b>L</b>	Length of window. (optional)

**Output parameters**

<b>gamma</b>	Canonical dual window.
--------------	------------------------

**Description**

`wildual(g, M)` returns the dual window of the Wilson or WMDCT basis with window  $g$ , parameter  $M$  and length equal to the length of the window  $g$ .

The window  $g$  may be a vector of numerical values, a text string or a cell array. See the help of `wilwin` for more details.

If the length of  $g$  is equal to  $2 \cdot M$  then the input window is assumed to be an FIR window. In this case, the dual window also has length of  $2 \cdot M$ . Otherwise the smallest possible transform length is chosen as the window length.

`wildual(g, M, L)` does the same, but now  $L$  is used as the length of the Wilson basis.

The input window  $g$  must be real and whole-point even. If  $g$  is not whole-point even, then reconstruction using the dual window will not be perfect. For a random window  $g$ , the window closest to  $g$  that satisfies these restrictions can be found by

```
g_wpe = real(peven(g));
```

All windows in the toolbox satisfies these restrictions unless clearly stated otherwise.

## 2.5 Conditions numbers

### 2.5.1 GABFRAMEBOUNDS - Calculate frame bounds of Gabor frame

**Usage**

```
fcond=gabframebounds(g,a,M);
[A,B]=gabframebounds(g,a,M);
[A,B]=gabframebounds(g,a,M,L);
[A,B]=gabframebounds(g,a,M,'lt',lt);
```

**Input parameters**

<b>g</b>	The window function.
<b>a</b>	Length of time shift.
<b>M</b>	Number of channels.
<b>L</b>	Length of transform to consider.
<b>lt</b>	Lattice type (for non-separable lattices).

**Output parameters**

<b>fcond</b>	Frame condition number (B/A)
<b>A, B</b>	Frame bounds.

**Description**

`gabframebounds(g, a, M)` calculates the ratio  $B/A$  of the frame bounds of the Gabor system with window  $g$ , and parameters  $a, M$ .

`[A, B]=gabframebounds(...)` returns the frame bounds  $A$  and  $B$  instead of just the ratio.

The window  $g$  may be a vector of numerical values, a text string or a cell array. See the help of `gabwin` for more details.

`gabframebounds(g, a, M, L)` will cut or zero-extend the window to length  $L$ .

`gabframebounds(g, a, M, 'lt', lt)` does the same for a non-separable lattice specified by  $lt$ . Please see the help of `matrix2latticetype` for a precise description of the parameter  $lt$ .

**2.5.2 GABRIESZBOUNDS - Calculate Riesz sequence/basis bounds of Gabor frame****Usage**

```
fcond=gabrieszbounds(g, a, M);
[A, B]=gabrieszbounds(g, a, M);
[A, B]=gabrieszbounds(g, a, M, L);
```

**Input parameters**

<b>g</b>	The window function.
<b>a</b>	Length of time shift.
<b>M</b>	Number of channels.
<b>L</b>	Length of transform to consider.

**Output parameters**

<b>fcond</b>	Frame condition number ( $B/A$ )
<b>A, B</b>	Frame bounds.

**Description**

`gabrieszbounds(g, a, M)` calculates the ratio  $B/A$  of the Riesz bounds of the Gabor system with window  $g$ , and parameters  $a, M$ .

`[A, B]=gabrieszbounds(g, a, M)` calculates the Riesz bounds  $A$  and  $B$  instead of just the ratio.

The window  $g$  may be a vector of numerical values, a text string or a cell array. See the help of `gabwin` for more details.

If the optional parameter  $L$  is specified, the window is cut or zero-extended to length  $L$ .

**2.5.3 WILBOUNDS - Calculate frame bounds of Wilson basis****Usage**

```
[AF, BF]=wilbounds(g, M)
[AF, BF]=wilbounds(g, M, L)
```

**Input parameters**

<b>g</b>	Window function.
<b>M</b>	Number of channels.
<b>L</b>	Length of transform to do (optional)

**Output parameters**

**AF, BF** Frame bounds.

**Description**

`wilbounds(g, M)` calculates the frame bounds of the Wilson frame operator of the Wilson basis with window  $g$  and  $M$  channels.

`[A, B]=wilbounds(g, a, M)` returns the frame bounds  $A$  and  $B$  instead of just the ratio.

The window  $g$  may be a vector of numerical values, a text string or a cell array. See the help of `wilwin` for more details.

If the length of  $g$  is equal to  $2 \cdot M$  then the input window is assumed to be a FIR window. Otherwise the smallest possible transform length is chosen as the window length.

If the optional parameter  $L$  is specified, the window is cut or zero-extended to length  $L$ .

### 2.5.4 GABDUALNORM - Measure of how close a window is to being a dual window

**Usage**

```
dn=gabdualnorm(g, gamma, a, M) ;
dn=gabdualnorm(g, gamma, a, M, L) ;
dn=gabdualnorm(g, gamma, a, M, 'lt', lt) ;
[scal,res]=gabdualnorm(...);
```

**Input parameters**

<b>gamma</b>	input window..
<b>g</b>	window function.
<b>a</b>	Length of time shift.
<b>M</b>	Number of modulations.
<b>L</b>	Length of transform to consider

**Output parameters**

<b>dn</b>	dual norm.
<b>scal</b>	Scaling factor
<b>res</b>	Residual

**Description**

`gabdualnorm(g, gamma, a, M)` calculates how close  $\gamma$  is to being a dual window of the Gabor frame with window  $g$  and parameters  $a$  and  $M$ .

The windows  $g$  and  $\gamma$  may be vectors of numerical values, text strings or cell arrays. See the help of `gabwin` for more details.

`[scal,res]=gabdualnorm(...)` computes two entities: `scal` determines if the windows are scaled correctly, it must be 1 for the windows to be dual. `res` is close to zero if the windows (scaled correctly) are dual windows.

`gabdualnorm(g, gamma, a, M, L)` does the same, but considers a transform length of  $L$ .

`gabdualnorm(g, gamma, a, M, 'lt', lt)` does the same for a non-separable lattice specified by  $lt$ . Please see the help of `matrix2lattice` for a precise description of the parameter  $lt$ .

`gabdualnorm` can be used to get the maximum relative reconstruction error when using the two specified windows. Consider the following code for some signal  $f$ , windows  $g$ ,  $\gamma$ , parameters  $a$  and  $M$  and transform-length  $L$  (See help on `dgt` on how to obtain  $L$ ):

```

fr=idgt (dgt (f, g, a, M), gamma, a);
er=norm(f-fr)/norm(f);
eest=gabdualnorm(g, gamma, a, M, L);

```

Then  $er < eest$  for all possible input signals  $f$ .

To get a similar estimate for an almost tight window  $gt$ , simply use

```
eest=gabdualnorm(gt, gt, a, M, L);
```

### 2.5.5 GABFRAMEDIAG - Diagonal of Gabor frame operator

#### Usage

```

d=gabframediag(g, a, M, L);
d=gabframediag(g, a, M, L, 'lt', lt);

```

#### Input parameters

<b>g</b>	Window function.
<b>a</b>	Length of time shift.
<b>M</b>	Number of channels.
<b>L</b>	Length of transform to do.
<b>lt</b>	Lattice type (for non-separable lattices).

#### Output parameters

<b>d</b>	Diagonal stored as a column vector
----------	------------------------------------

#### Description

`gabframediag(g, a, M, L)` computes the diagonal of the Gabor frame operator with respect to the window  $g$  and parameters  $a$  and  $M$ . The diagonal is stored as column vector of length  $L$ .

The diagonal of the frame operator can for instance be used as a preconditioner.

### 2.5.6 WILFRAMEDIAG - Diagonal of Wilson and WMDCT frame operator

#### Usage

```
d=wilframediag(g, M, L);
```

#### Input parameters

<b>g</b>	Window function.
<b>M</b>	Number of channels.
<b>L</b>	Length of transform to do.

#### Output parameters

<b>d</b>	Diagonal stored as a column vector
----------	------------------------------------

#### Description

`wilframediag(g, M, L)` computes the diagonal of the Wilson or WMDCT frame operator with respect to the window  $g$  and number of channels  $M$ . The diagonal is stored as column vector of length  $L$ .

The diagonal of the frame operator can for instance be used as a preconditioner.

## 2.6 Phase gradient methods and reassignment

### 2.6.1 GABPHASEGRAD - Phase gradient of the DGT

#### Usage

```
[tgrad,fgrad,c] = gabphasegrad('dgt',f,g,a,M);
[tgrad,fgrad]   = gabphasegrad('phase',cphase,a);
[tgrad,fgrad]   = gabphasegrad('abs',s,g,a);
```

#### Description

`[tgrad,fgrad]=gabphasegrad(method,...)` computes the time-frequency gradient of the phase of the dgt of a signal. The derivative in time *tgrad* is the instantaneous frequency while the frequency derivative *fgrad* is the local group delay.

*tgrad* and *fgrad* measure the deviation from the current time and frequency, so a value of zero means that the instantaneous frequency is equal to the center frequency of the considered channel.

*tgrad* is scaled such that distances are measured in samples. Similarly, *fgrad* is scaled such that the Nyquist frequency (the highest possible frequency) corresponds to a value of  $L/2$ .

The computation of *tgrad* and *fgrad* is inaccurate when the absolute value of the Gabor coefficients is low. This is due to the fact the the phase of complex numbers close to the machine precision is almost random. Therefore, *tgrad* and *fgrad* may attain very large random values when *abs(c)* is close to zero.

The computation can be done using three different methods.

'dgt'	Directly from the signal. This is the default method.
'phase'	From the phase of a DGT of the signal. This is the classic method used in the phase vocoder.
'abs'	From the absolute value of the DGT. Currently this method works only for Gaussian windows.

`[tgrad,fgrad]=gabphasegrad('dgt',f,g,a,M)` computes the time-frequency gradient using a DGT of the signal *f*. The DGT is computed using the window *g* on the lattice specified by the time shift *a* and the number of channels *M*. The algorithm used to perform this calculation computes several DGTs, and therefore this routine takes the exact same input parameters as *dgt*.

The window *g* may be specified as in *dgt*. If the window used is 'gauss', the computation will be done by a faster algorithm.

`[tgrad,fgrad,c]=gabphasegrad('dgt',f,g,a,M)` additionally returns the Gabor coefficients *c*, as they are always computed as a byproduct of the algorithm.

`[tgrad,fgrad]=gabphasegrad('phase',cphase,a)` computes the phase gradient from the phase *cphase* of a DGT of the signal. The original DGT from which the phase is obtained must have been computed using a time-shift of *a*.

`[tgrad,fgrad]=gabphasegrad('abs',s,g,a)` computes the phase gradient from the spectrogram *s*. The spectrogram must have been computed using the window *g* and time-shift *a*.

`[tgrad,fgrad]=gabphasegrad('abs',s,g,a,difforder)` uses a centered finite difference scheme of order *difforder* to perform the needed numerical differentiation. Default is to use a 4th order scheme.

Currently the 'abs' method only works if the window *g* is a Gaussian window specified as a string or cell array.

**References:** [8], [20], [30]

### 2.6.2 GABREASSIGN - Reassign time-frequency distribution

#### Usage

```
sr = gabreassign(s,tgrad,fgrad,a);
sr = gabreassign(s,tgrad,fgrad,a,'aa');
```

### Description

`gabreassign(s,tgrad,fgrad,a)` reassigns the values of the positive time-frequency distribution  $s$  using the phase gradient given by  $fgrad$  and  $tgrad$ . The lattice is determined by the time shift  $a$  and the number of channels deduced from the size of  $s$ .

$fgrad$  and  $tgrad$  can be obtained by the routine `gabphasegrad`.

### Examples:

The following example demonstrates how to manually create a reassigned spectrogram. An easier way is to just call `resgram`:

```
% Create reassigned vector field of the bat signal.
a=4;
M=100
[tgrad, fgrad, c] = gabphasegrad('dgt',bat,'gauss',a,M);

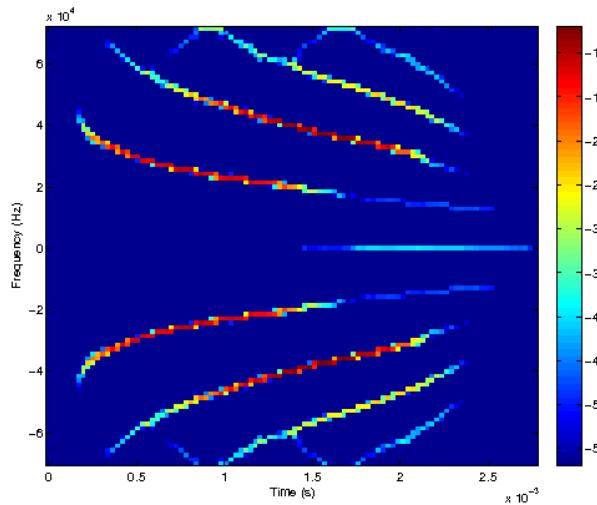
% Perform the actual reassignment
sr = gabreassign(abs(c).^2,tgrad,fgrad,a);

% Display it using plotdgt
plotdgt(sr,a,143000,50);
```

This code produces the following output:

M =

100



References: [8]

## 2.7 Phase conversions

### 2.7.1 PHASELOCK - Phaselock Gabor coefficients

#### Usage

```
c=phaselock(c,a);
```

**Description**

`phaselock (c, a)` phaselocks the Gabor coefficients  $c$ . The coefficients must have been obtained from a dgt with parameter  $a$ .

Phaselocking the coefficients modifies them so as if they were obtained from a time-invariant Gabor system. A filter bank produces phase locked coefficients.

Phaselocking of Gabor coefficients correspond to the following transform: Consider a signal  $f$  of length  $L$  and define  $N = L/a$ . The output from `c=phaselock (dgt (f, g, a, M), a)` is given by

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

where  $m = 0, \dots, M-1$  and  $n = 0, \dots, N-1$  and  $l-an$  is computed modulo  $L$ .

`phaselock (c, a, 'lt', lt)` does the same for a non-separable lattice specified by  $lt$ . Please see the help of `matrix2latticetype` for a precise description of the parameter  $lt$ .

**References:** [68]

**2.7.2 PHASEUNLOCK - Undo phase lock of Gabor coefficients****Usage**

```
c=phaseunlock (c, a);
```

**Description**

`phaseunlock (c, a)` removes phase locking from the Gabor coefficients  $c$ . The coefficient must have been obtained from a dgt with parameter  $a$ .

Phaselocking the coefficients modifies them so as if they were obtained from a time-invariant Gabor system. A filter bank produces phase locked coefficients.

**References:** [68]

**2.7.3 SYMPHASE - Change Gabor coefficients to symmetric phase****Usage**

```
c=sympphase (c, a);
```

**Description**

`sympphase (c, a)` alters the phase of the Gabor coefficients  $c$  so as if they were obtained from a Gabor transform based on symmetric time/frequency shifts. The coefficient must have been obtained from a dgt with parameter  $a$ .

**2.8 Support for non-separable lattices****2.8.1 MATRIX2LATTICETYPE - Convert matrix form to standard lattice description****Usage**

```
[a, M, lt] = matrix2latticetype (L, V);
```

**Description**

`[a, M, lt]=matrix2latticetype (L, V)` converts a  $2 \times 2$  integer matrix description into the standard description of a lattice using the  $a$ ,  $M$  and  $lt$ . The conversion is *only* valid for the specified transform length  $L$ .

The lattice type  $lt$  is a  $1 \times 2$  vector  $[lt_1, lt_2]$  denoting an irreducible fraction  $lt_1/lt_2$ . This fraction describes the distance in frequency (counted in frequency channels) that each coefficient is offset when moving in time by the time-shift of  $a$ . Some examples:  $lt=[0 1]$  defines a square lattice,  $lt=[1 2]$  defines the quinqua (almost hexagonal) lattice,  $lt=[1 3]$  describes a lattice with a  $1/3$  frequency offset for each time shift and so forth.

An example:

```
[a,M,lt] = matrix2latticetype(120,[10 0; 5 10])
```

This code produces the following output:

```
a =
```

```
10
```

```
M =
```

```
12
```

```
lt =
```

```
1      2
```

### Coefficient layout:

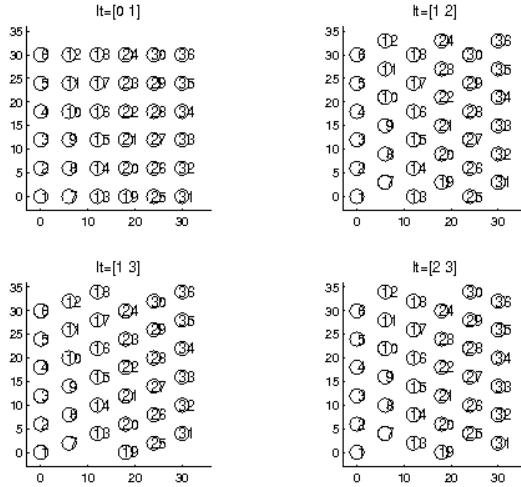
The following code generates plots which show the coefficient layout and enumeration of the first 4 lattices in the time-frequency plane:

```
a=6;
M=6;
L=36;
b=L/M;
N=L/a;
cw=3;
ftz=12;

[x,y]=meshgrid(a*(0:N-1),b*(0:M-1));

lt1=[0 1 1 2];
lt2=[1 2 3 3];

for fignum=1:4
    subplot(2,2,fignum);
    z=y;
    if lt2(fignum)>0
        z=z+mod(lt1(fignum)*x/lt2(fignum),b);
    end;
    for ii=1:M*N
        text(x(ii)-cw/4,z(ii),sprintf('%2.0i',ii),'FontSize',ftz);
        rectangle('Curvature',[1 1], 'Position',[x(ii)-cw/2,z(ii)-cw/2,cw,cw]);
    end;
    axis([-cw L -cw L]);
    axis('square');
    title(sprintf('lt=[%i %i]',lt1(fignum),lt2(fignum)),'FontSize',ftz);
end;
```



## 2.8.2 LATTICETYPE2MATRIX - Convert lattice description to matrix form

### Usage

```
V=latticetype2matrix(L,a,M,lt);
```

### Description

`V=latticetype2matrix(L,a,M,lt)` converts a standard description of a lattice using the  $a$ ,  $M$  and  $lt$  parameters into a  $2 \times 2$  integer matrix description. The conversion is *only* valid for the specified transform length  $L$ .

The output will be in lower triangular Hermite normal form.

For more information, see [http://en.wikipedia.org/wiki/Hermite\\_normal\\_form](http://en.wikipedia.org/wiki/Hermite_normal_form).

An example:

```
V = latticetype2matrix(120,10,12,[1 2])
```

This code produces the following output:

```
V =
```

```
10      0
 5     10
```

## 2.8.3 SHEARFIND - Shears for transformation of a general lattice to separable

### Usage

```
[s0,s1,br] = shearfind(L,a,M,lt);
```

### Description

`[s0,s1,br]=shearfind(L,a,M,lt)` computes three numbers, the first two represent a frequency and time shear respectively. With the returned choices of  $s_0$  and  $s_1$  one can transform an initial lattice given by  $a$ ,  $M$  and  $lt$  into a separable (rectangular) lattice given by

$$a_r = \frac{aL}{b_r M}, \quad M_r = \frac{L}{b_r}.$$

If  $s_0$  is non-zero, the transformation from general to separable lattice requires a frequency-side shear. Similarly, if  $s_1$  is non-zero, a time-side shear is required.

## 2.8.4 NOSHEARLENGTH - Transform length that does not require a frequency shear

### Usage

```
L=noshearlength(Ls,a,M,lt)
```

### Description

`noshearlength(Ls,a,M,lt)` computes the next larger transform length bigger or equal to  $L_s$  for which the shear algorithm does not require a frequency side shear for a non-separable Gabor system specified by  $a$ ,  $M$  and  $lt$ .

This property makes computation of the canonical dual and tight Gabor windows `gabdual` and `gabtight` and the `dgt` for a full length window faster, if this transform length is chosen.

## 2.9 Plots

### 2.9.1 TFPLT - Plot coefficient matrix on the TF plane

#### Usage

```
tfplot(coef,step,yr);
tfplot(coef,step,yr,...);
```

#### Description

`tfplot(coef,step,yr)` will plot a rectangular coefficient array on the TF-plane. The shift in samples between each column of coefficients is given by the variable `step`. The vector `yr` is a  $1 \times 2$  vector containing the lowest and highest normalized frequency.

`C=tfplot(...)` returns the processed image data used in the plotting. Inputting this data directly to `imagesc` or similar functions will create the plot. This is useful for custom post-processing of the image data.

`tfplot` is not meant to be called directly. Instead, it is called by other plotting routines to give a uniform display format.

`tfplot` (and all functions that call it) takes the following arguments.

<b>'dynrange',r</b>	Limit the dynamical range to $r$ by using a colormap in the interval $[chigh-r, chigh]$ , where $chigh$ is the highest value in the plot. The default value of [] means to not limit the dynamical range.
<b>'db'</b>	Apply $20 \cdot \log_{10}$ to the coefficients. This makes it possible to see very weak phenomena, but it might show too much noise. A logarithmic scale is more adapted to perception of sound. This is the default.
<b>'dbsq'</b>	Apply $10 \cdot \log_{10}$ to the coefficients. Same as the ' <code>db</code> ' option, but assume that the input is already squared.
<b>'lin'</b>	Show the coefficients on a linear scale. This will display the raw input without any modifications. Only works for real-valued input.
<b>'linsq'</b>	Show the square of the coefficients on a linear scale.
<b>'linabs'</b>	Show the absolute value of the coefficients on a linear scale.
<b>'tc'</b>	Time centering. Move the beginning of the signal to the middle of the plot.
<b>'clim',clim</b>	Use a colormap ranging from <code>clim(1)</code> to <code>clim(2)</code> . These values are passed to <code>imagesc</code> . See the help on <code>imagesc</code> .
<b>'image'</b>	Use <code>imagesc</code> to display the plot. This is the default.

<b>'contour'</b>	Do a contour plot.
<b>'surf'</b>	Do a surf plot.
<b>'colorbar'</b>	Display the colorbar. This is the default.
<b>'nocolorbar'</b>	Do not display the colorbar.
<b>'display'</b>	Display the figure. This is the default.
<b>'nodisplay'</b>	Do not display figure. This is useful if you only want to obtain the output for further processing.

If both '`clim`' and '`dynrange`' are specified, then '`clim`' takes precedence.

It is possible to customize the text by setting the following values:

'time',  $t$  The word denoting time. Default is 'Time'

<b>'frequency',<math>f</math></b>	The word denoting frequency. Default is 'Frequency'
<b>'samples',<math>s</math></b>	The word denoting samples. Default is 'samples'
<b>'normalized',<math>n</math></b>	Default value is 'normalized'.

## 2.9.2 PLOTDGT - Plot DGT coefficients

### Usage

```
plotdgt(coef,a);
plotdgt(coef,a,fs);
plotdgt(coef,a,fs,dynrange);
```

### Description

`plotdgt(coef,a)` plots the Gabor coefficients  $\text{coef}$ . The coefficients must have been produced with a time shift of  $a$ .

`plotdgt(coef,a,fs)` does the same assuming a sampling rate of  $fs$  Hz of the original signal.

`plotdgt(coef,a,fs,dynrange)` additionally limits the dynamic range.

The figure generated by this function places the zero-frequency in the center of the y-axis, with positive frequencies above and negative frequencies below.

`C=plotdgt(...)` returns the processed image data used in the plotting. Inputting this data directly to `imagesc` or similar functions will create the plot. This is useful for custom post-processing of the image data.

`plotdgt` supports all the optional parameters of `tfplot`. Please see the help of `tfplot` for an exhaustive list.

## 2.9.3 PLOTDGTRREAL - Plot DGTRREAL coefficients

### Usage

```
plotdgtrreal(coef,a,M);
plotdgtrreal(coef,a,M,fs);
plotdgtrreal(coef,a,M,fs,dynrange);
```

### Description

`plotdgtrreal(coef,a,M)` plots Gabor coefficient from `dgtreal`. The parameters  $a$  and  $M$  must match those from the call to `dgtreal`.

`plotdgtrreal(coef,a,M,fs)` does the same assuming a sampling rate of  $fs$  Hz of the original signal.

`plotdgtrreal(coef,a,M,fs,dynrange)` additionally limits the dynamic range.

`C=plotdgtreal(...)` returns the processed image data used in the plotting. Inputting this data directly to `imagesc` or similar functions will create the plot. This is useful for custom post-processing of the image data.

`plotdgtreal` supports all the optional parameters of `tfplot`. Please see the help of `tfplot` for an exhaustive list.

## 2.9.4 PLOTDWILT - Plot DWILT coefficients

### Usage

```
plotdwilt(coef,);
plotdwilt(coef,fs);
plotdwilt(coef,fs,dynrange);
```

### Description

`plotdwilt(coef)` will plot coefficients from `dwilt`.

`plotdwilt(coef,fs)` will do the same assuming a sampling rate of  $fs$  Hz of the original signal. Since a Wilson representation does not contain coefficients for all positions on a rectangular TF-grid, there will be visible 'holes' among the lowest (DC) and highest (Nyquist rate) coefficients. See the help on `wil2rect`.

`plotdwilt(coef,fs,dynrange)` will additionally limit the dynamic range.

`C=plotdwilt(...)` returns the processed image data used in the plotting. Inputting this data directly to `imagesc` or similar functions will create the plot. This is useful for custom post-processing of the image data.

`plotdwilt` supports all the optional parameters of `tfplot`. Please see the help of `tfplot` for an exhaustive list.

## 2.9.5 PLOTWMDCT - Plot WMDCT coefficients

### Usage

```
plotwmdct(coef,);
plotwmdct(coef,fs);
plotwmdct(coef,fs,dynrange);
```

### Description

`plotwmdct(coef)` plots coefficients from `wmdct`.

`plotwmdct(coef,fs)` does the same assuming a sampling rate of  $fs$  Hz of the original signal.

`plotwmdct(coef,fs,dynrange)` additionally limits the dynamic range.

`C=plotwmdct(...)` returns the processed image data used in the plotting. Inputting this data directly to `imagesc` or similar functions will create the plot. This is useful for custom post-processing of the image data.

`plotwmdct` supports all the optional parameters of `tfplot`. Please see the help of `tfplot` for an exhaustive list.

## 2.9.6 SGRAM - Spectrogram

### Usage

```
sgram(f,op1,op2, ... );
sgram(f,fs,op1,op2, ... );
C=sgram(f, ... );
```

## Description

`sgram(f)` plots a spectrogram of `f` using a Discrete Gabor Transform (DGT).

`sgram(f, fs)` does the same for a signal with sampling rate `fs` (sampled with `fs` samples per second);

`sgram(f, fs, dynrange)` additionally limits the dynamic range of the plot. See the description of the '`dynrange`' parameter below.

`C=sgram(f, ...)` returns the image to be displayed as a matrix. Use this in conjunction with `imwrite` etc. These coefficients are **only** intended to be used by post-processing image tools. Numerical Gabor signal analysis and synthesis should **always** be done using the `dgt`, `idgt`, `dgtreal` and `idgtreal` functions.

Additional arguments can be supplied like this:

```
sgram(f, fs, 'dynrange', 50)
```

The arguments must be character strings possibly followed by an argument:

<b>'dynrange',r</b>	Limit the dynamical range to <code>r</code> by using a colormap in the interval $[chigh - r, chigh]$ , where <code>chigh</code> is the highest value in the plot. The default value of <code>[]</code> means to not limit the dynamical range.
<b>'db'</b>	Apply $20 * \log_{10}$ to the coefficients. This makes it possible to see very weak phenomena, but it might show too much noise. A logarithmic scale is more adapted to perception of sound. This is the default.
<b>'lin'</b>	Show the energy of the coefficients on a linear scale.
<b>'tfr',v</b>	Set the ratio of frequency resolution to time resolution. A value <code>v = 1</code> is the default. Setting <code>v &gt; 1</code> will give better frequency resolution at the expense of a worse time resolution. A value of $0 < v < 1$ will do the opposite.
<b>'wlen',s</b>	Window length. Specifies the length of the window measured in samples. See help of <code>pgauss</code> on the exact details of the window length.
<b>'posfreq'</b>	Display only the positive frequencies. This is the default for real-valued signals.
<b>'nf'</b>	Display negative frequencies, with the zero-frequency centered in the middle. For real signals, this will just mirror the upper half plane. This is standard for complex signals.
<b>'tc'</b>	Time centering. Move the beginning of the signal to the middle of the plot. This is useful for visualizing the window functions of the toolbox.
<b>'image'</b>	Use <code>imagesc</code> to display the spectrogram. This is the default.
<b>'clim',clim</b>	Use a colormap ranging from <code>clim(1)</code> to <code>clim(2)</code> . These values are passed to <code>imagesc</code> . See the help on <code>imagesc</code> .
<b>'thr',r</b>	Keep only the largest fraction <code>r</code> of the coefficients, and set the rest to zero.
<b>'fmax',y</b>	Display <code>y</code> as the highest frequency. Default value of <code>[]</code> means to use the Nyquist frequency.
<b>'xres',xres</b>	Approximate number of pixels along x-axis / time. The default value is 800
<b>'yres',yres</b>	Approximate number of pixels along y-axis / frequency. The Default value is 600

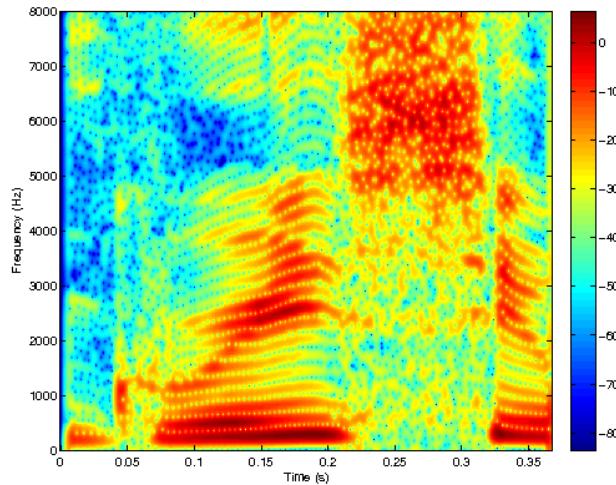
<b>'contour'</b>	Do a contour plot to display the spectrogram.
<b>'surf'</b>	Do a surf plot to display the spectrogram.
<b>'mesh'</b>	Do a mesh plot to display the spectrogram.
<b>'colorbar'</b>	Display the colorbar. This is the default.
<b>'nobar'</b>	Do not display the colorbar.

In addition to these parameters, `sgram` accepts any of the flags from `normalize`. The window used to calculate the spectrogram will be normalized as specified.

### Examples:

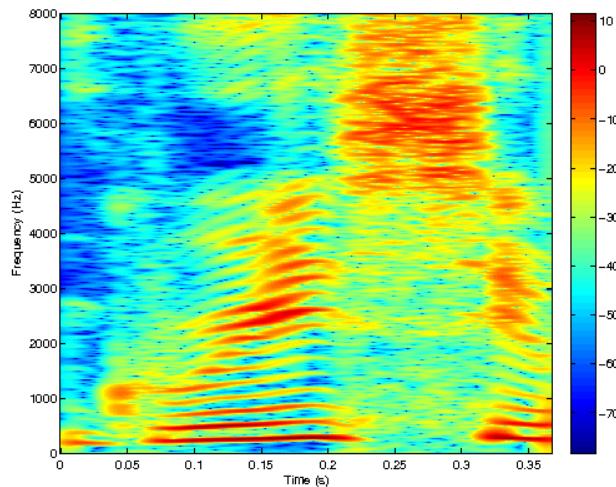
The greasy signal is sampled using a sampling rate of 16 kHz. To display a spectrogram of greasy with a dynamic range of 90 dB, use:

```
sgram(greasy, 16000, 90);
```



To create a spectrogram with a window length of 20ms (which is typically used in speech analysis) use

```
fs=16000;
sgram(greasy, fs, 90, 'wlen', round(20/1000*fs));
```



### 2.9.7 GABIMAGEPARS - Find Gabor parameters to generate image

#### Usage

```
[a,M,L,N,Ngood]=gabimagepars(Ls,x,y);
```

#### Description

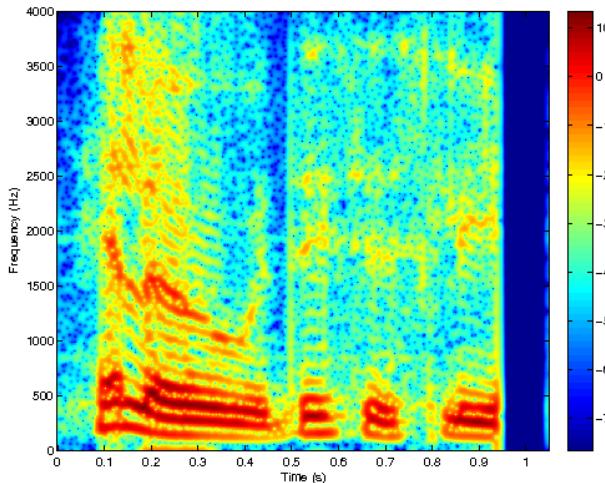
`[a,M,L,N,Ngood]=gabimagepars(Ls,x,y)` will compute a reasonable set of parameters  $a$ ,  $M$  and  $L$  to produce a nice Gabor 'image' of a signal of length  $L_s$ . The approximate number of pixels in the time direction is given as  $x$  and the number of pixels in the frequency direction is given as  $y$ .

The output parameter  $Ngood$  contains the number of time steps (columns in the coefficients matrix) that contains relevant information. The columns from  $Ngood$  until  $N$  only contains information from a zero-extension of the signal.

If you use this function to calculate a grid size for analysis of a real-valued signal (using `dgtreal`), please input twice of the desired size  $y$ . This is because `dgtreal` only returns half as many coefficients in the frequency direction as `dgt`.

An example: We wish to compute a Gabor image of a real valued signal  $f$  of length 7500. The image should have an approximate resolution of  $800 \times 600$  pixels:

```
[f,fs]=linus; f=f(4001:4000+7500);
[a,M,L,N,Ngood] = gabimagepars(7500,800,2*600);
c = dgtreal(f,'gauss',a,M);
plotdgtreal(c,a,M,fs,90);
```



The size of  $c$  is  $N \times (M/2) + 1$  equal to  $801 \times 600$  pixels.

For this function to work properly, the specified numbers for  $x$  and  $y$  must not be large prime numbers.

### 2.9.8 RESGRAM - Reassigned spectrogram plot

#### Usage

```
resgram(f,op1,op2, ... );
resgram(f,fs,op1,op2, ... );
```

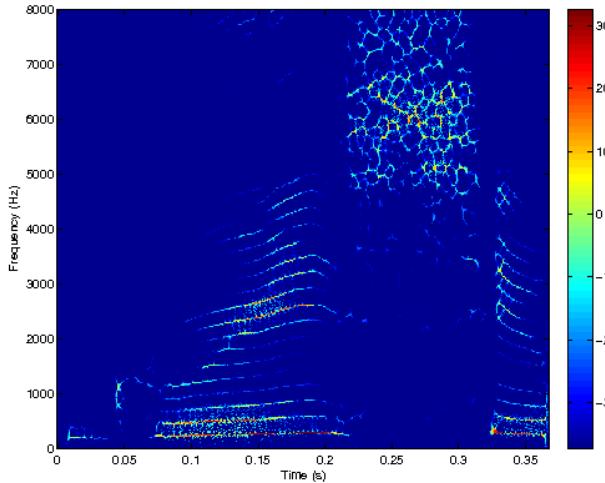
#### Description

`resgram(f)` plots a reassigned spectrogram of  $f$ .

`resgram(f,fs)` does the same for a signal with sampling rate  $fs$  (sampled with  $fs$  samples per second).

Because reassigned spectrograms can have an extreme dynamical range, consider always using the '`dynrange`' or '`clim`' options (see below) in conjunction with the '`db`' option (on by default). An example:

```
resgram(greasy, 16000, 'dynrange', 70);
```



This will produce a reassigned spectrogram of the greasy signal without drowning the interesting features in noise.

`C=sgram(f, ...)` returns the image to be displayed as a matrix. Use this in conjunction with `imwrite` etc. These coefficients are **only** intended to be used by post-processing image tools. Reassignment should be done using the `gabreassign` function instead.

`resgram` accepts the following additional arguments:

<b>'dynrange',r</b>	Limit the dynamical range to $r$ by using a colormap in the interval $[chigh - r, chigh]$ , where $chigh$ is the highest value in the plot. The default value of [] means to not limit the dynamical range.
<b>'db'</b>	Apply $20 * \log_{10}$ to the coefficients. This makes it possible to see very weak phenomena, but it might show too much noise. A logarithmic scale is more adapted to perception of sound. This is the default.
<b>'sharp',alpha</b>	Set the sharpness of the plot. If $alpha = 0$ the regular spectrogram is obtained. $alpha = 1$ means full reassignment. Anything in between will produce a partially sharpened picture. Default is $alpha = 1$ .
<b>'lin'</b>	Show the energy of the coefficients on a linear scale.
<b>'tfr',v</b>	Set the ratio of frequency resolution to time resolution. A value $v = 1$ is the default. Setting $v > 1$ will give better frequency resolution at the expense of a worse time resolution. A value of $0 < v < 1$ will do the opposite.
<b>'wlen',s</b>	Window length. Specifies the length of the window measured in samples. See help of <code>pgauss</code> on the exact details of the window length.
<b>'posfreq'</b>	Display only the positive frequencies. This is the default for real-valued signals.
<b>'nf'</b>	Display negative frequencies, with the zero-frequency centered in the middle. For real signals, this will just mirror the upper half plane. This is standard for complex signals.

<b>'tc'</b>	Time centering. Move the beginning of the signal to the middle of the plot. This is useful for visualizing the window functions of the toolbox.
<b>'image'</b>	Use <code>imagesc</code> to display the spectrogram. This is the default.
<b>'clim',<i>clim</i></b>	Use a colormap ranging from <i>clim</i> (1) to <i>clim</i> (2). These values are passed to <code>imagesc</code> . See the help on <code>imagesc</code> .
<b>'thr',<i>r</i></b>	Keep only the largest fraction <i>r</i> of the coefficients, and set the rest to zero.
<b>'fmax',<i>y</i></b>	Display <i>y</i> as the highest frequency. Default value of [ ] means to use the Nyquist frequency.
<b>'xres',<i>xres</i></b>	Approximate number of pixels along x-axis / time. The default value is 800
<b>'yres',<i>yres</i></b>	Approximate number of pixels along y-axis / frequency. The default value is 600
<b>'contour'</b>	Do a contour plot to display the spectrogram.
<b>'surf'</b>	Do a surf plot to display the spectrogram.
<b>'mesh'</b>	Do a mesh plot to display the spectrogram.
<b>'colorbar'</b>	Display the colorbar. This is the default.
<b>'nocolorbar'</b>	Do not display the colorbar.

In addition to these parameters, `sgram` accepts any of the flags from `normalize`. The window used to calculate the spectrogram will be normalized as specified.

### 2.9.9 INSTFREQPLOT - Plot of instantaneous frequency

#### Usage

```
instfreqplot(f,op1,op2, ... );
instfreqplot(f,fs,op1,op2, ... );
```

#### Description

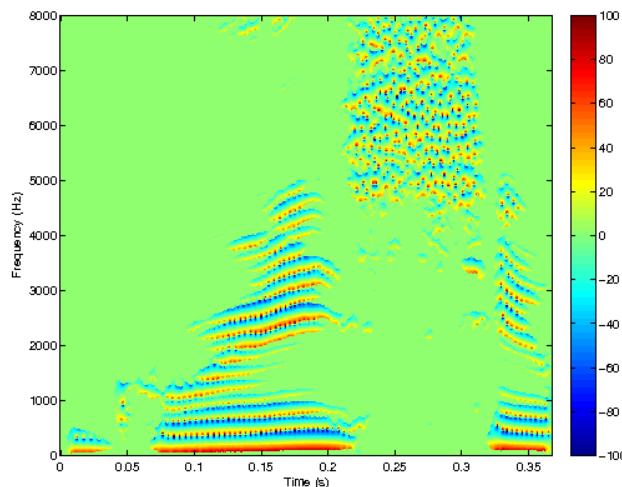
`instfreqplot(f)` plots the instantaneous frequency of *f* using a dgt.

`instfreqplot(f,fs)` does the same for a signal with sampling rate *fs* Hz.

The instantaneous frequency contains extreme spikes in regions where the spectrogram is close to zero. These points are usually uninteresting and destroy the visibility of the plot. Use the '`'thr'` or '`'clim'` or '`'climsym'`' options (see below) to remove these points.

An example:

```
instfreqplot(greasy,16000,'thr',.03,'climsym',100);
```



will produce a nice instantaneous frequency plot of the greasy signal.

`instfreqplot` accepts the following optional arguments:

<b>'tfr',v</b>	Set the ratio of frequency resolution to time resolution. A value $v = 1$ is the default. Setting $v > 1$ will give better frequency resolution at the expense of a worse time resolution. A value of $0 < v < 1$ will do the opposite.
<b>'wlen',s</b>	Window length. Specifies the length of the window measured in samples. See help of <code>pgauss</code> on the exact details of the window length.
<b>'thr',r</b>	Keep the coefficients with a magnitude larger than $r$ times the largest magnitude. Set the instantaneous frequency of the rest of the coefficients to zero
<b>'nf'</b>	Display negative frequencies, with the zero-frequency centered in the middle. For real signals, this will just mirror the upper half plane. This is standard for complex signals.
<b>'tc'</b>	Time centering. Move the beginning of the signal to the middle of the plot. This is useful for visualizing the window functions of the toolbox.
<b>'image'</b>	Use <code>imagesc</code> to display the spectrogram. This is the default.
<b>'dgt'</b>	Use the ' <code>dgt</code> ' method to compute the instantaneous frequency. This is the default.
<b>'clim',clim</b>	Use a colormap ranging from <code>clim(1)</code> to <code>clim(2)</code> . These values are passed to <code>imagesc</code> . See the help on <code>imagesc</code> .
<b>'climsym',cval</b>	Use a colormap ranging from $-cval$ to $cval$
<b>'fmax',y</b>	Display $y$ as the highest frequency.

## 2.9.10 PHASEPLOT - Phase plot

### Usage

```
phaseplot(f,op1,op2, ... );
phaseplot(f,fs,op1,op2, ... );
```

## Description

`phaseplot(f)` plots the phase of  $f$  using a dgt.

`phaseplot(f, fs)` does the same for a signal with sampling rate  $fs$  Hz.

`phaseplot` should only be used for short signals (shorter than the resolution of the screen), as there will otherwise be some visual aliasing, such that very fast changing areas will look very smooth. `phaseplot` always calculates the phase of the full time/frequency plane (as opposed to `sgram`), and you therefore risk running out of memory for long signals.

`phaseplot` takes the following flags at the end of the line of input arguments:

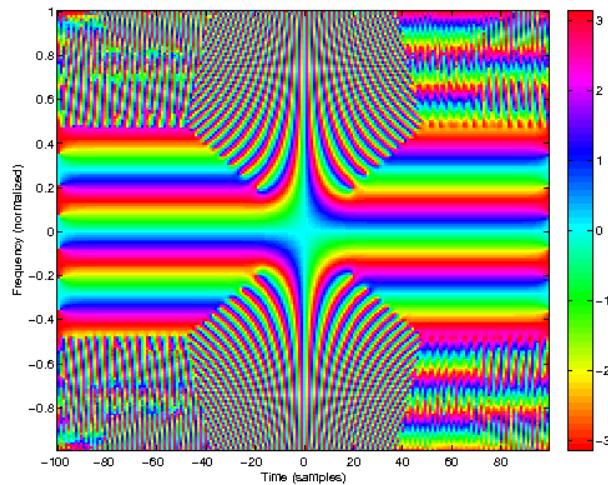
<b>'tfr',v</b>	Set the ratio of frequency resolution to time resolution. A value $v = 1$ is the default. Setting $v > 1$ will give better frequency resolution at the expense of a worse time resolution. A value of $0 < v < 1$ will do the opposite.
<b>'wlen',s</b>	Window length. Specifies the length of the window measured in samples. See help of <code>pgauss</code> on the exact details of the window length.
<b>'nf'</b>	Display negative frequencies, with the zero-frequency centered in the middle. For real signals, this will just mirror the upper half plane. This is standard for complex signals.
<b>'tc'</b>	Time centering. Move the beginning of the signal to the middle of the plot. This is useful for visualizing the window functions of the toolbox.
<b>'thr',r</b>	Keep the coefficients with a magnitude larger than $r$ times the largest magnitude. Set the phase of the rest of the coefficients to zero. This is useful, because for small amplitude the phase values can be meaningless.
<b>'timeinv'</b>	Display the phase as computed by a time-invariant dgt. This is the default.
<b>'freqinv'</b>	Display the phase as computed by a frequency-invariant dgt.
<b>'fmax',y</b>	Display $y$ as the highest frequency.
<b>'colorbar'</b>	Display the colorbar. This is the default.
<b>'nocolorbar'</b>	Do not display the colorbar.

For the best result when using `phaseplot`, use a circulant color map, for instance `hsv`.

## Examples:

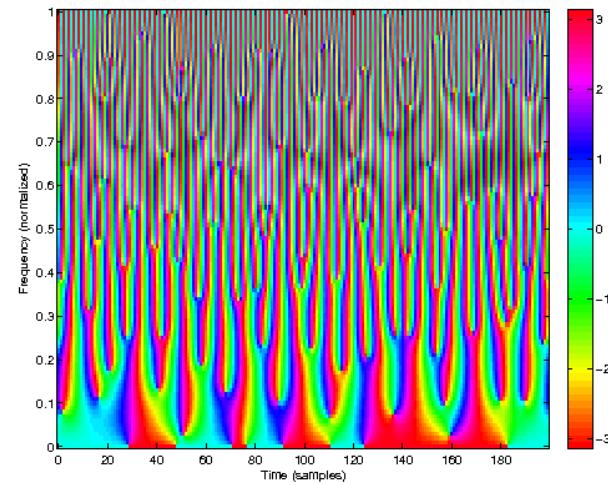
The following code shows the `phaseplot` of a periodic, hyperbolic secant visualized using the `hsv` colormap:

```
phaseplot(psech(200), 'tc', 'nf');
colormap(hsv);
```



The following phaseplot shows the phase of white, Gaussian noise:

```
phaseplot(randn(200,1));  
colormap(hsv);
```



**References:** [18]



# Chapter 3

## LTFAT - Basic Fourier and DCT analysis.

### 3.1 Support routines

#### 3.1.1 FFTINDEX - Frequency index of FFT modulations

##### Usage

```
n=fftindex(N);
```

##### Description

`fftindex(N)` returns the index of the frequencies of the standard FFT of length  $N$  as they are ordered in the output from the `fft` routine. The numbers returned are in the range `-ceil(N/2)+1:floor(N/2)`.  
`fftindex(N, 0)` does as above, but sets the Nyquist frequency to zero.

#### 3.1.2 MODCENT - Centered modulo

##### Usage

```
y=modcent(x, r);
```

##### Description

`modcent(x, r)` computes the modulo of  $x$  in the range  $[-r/2, r/2[$ .  
As an example, to compute the modulo of  $x$  in the range  $[-\pi, \pi[$  use the call:

```
y = modcent(x, 2*pi);
```

#### 3.1.3 FLOOR23 - Previous number with only 2,3 factors

##### Usage

```
nceil=floor23(n);
```

##### Description

`floor23(n)` returns the first number less than or equal to  $n$ , which can be written as a product of powers of 2 and 3.

The algorithm will look up the best size in a table, which is computed the first time the function is run. If the input size is larger than the largest value in the table, the input size will be reduced by factors of 2, until it is in range.

`[nceil, table]=floor23(n)` additionally returns the table used for lookup.

**Examples:**

Return the first number smaller or equal to 26 that can be written solely as products of powers of 2 and 3:

```
floor23(26)
```

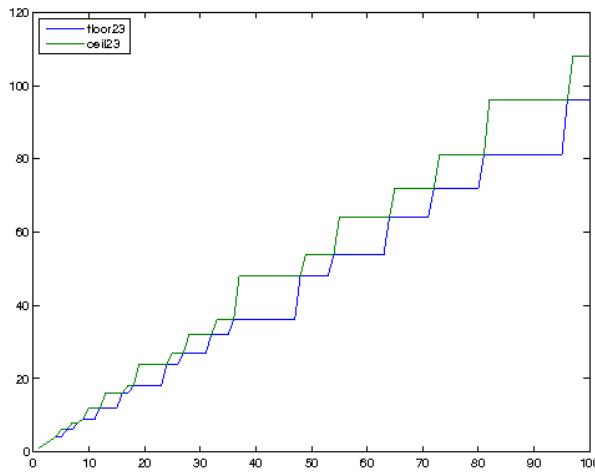
*This code produces the following output:*

```
ans =
```

```
24
```

This plot shows the behaviour of floor23 and ceil23 for numbers up to 100:

```
x=1:100;
plot(x,floor23(x),x,ceil23(x));
legend('floor23','ceil23','Location','Northwest');
```

**3.1.4 FLOOR235 - Previous number with only 2,3 and 5 factors****Usage**

```
nfloor=floor235(n);
```

**Description**

`floor235(n)` returns the next number greater than or equal to  $n$ , which can be written as a product of powers of 2, 3 and 5.

The algorithm will look up the best size in a table, which is computed the first time the function is run. If the input size is larger than the largest value in the table, the input size will be reduced by factors of 2, until it is in range.

`[nfloor,table]=floor235(n)` additionally returns the table used for lookup.

**Examples:**

Return the first number smaller or equal to 26 that can be written solely as products of powers of 2, 3 and 5:

```
floor235(26)
```

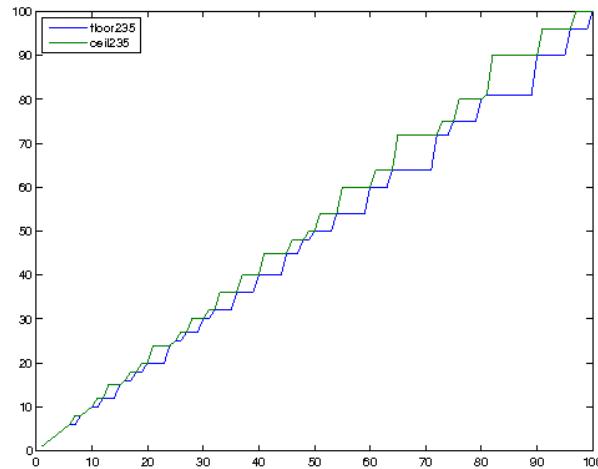
*This code produces the following output:*

```
ans =
```

```
25
```

This plot shows the behaviour of floor235 and ceil235 for numbers up to 100:

```
x=1:100;
plot(x,floor235(x),x,ceil235(x));
legend('floor235','ceil235','Location','Northwest');
```



### 3.1.5 CEIL23 - Next number with only 2,3 factors

#### Usage

```
nceil=ceil23(n);
```

#### Description

`ceil23(n)` returns the next number greater than or equal to  $n$ , which can be written as a product of powers of 2 and 3.

The algorithm will look up the best size in a table, which is computed the first time the function is run. If the input size is larger than the largest value in the table, the input size will be reduced by factors of 2, until it is in range.

`[nceil,table]=ceil23(n)` additionally returns the table used for lookup.

#### Examples:

Return the first number larger or equal to 19 that can be written solely as products of powers of 2 and 3:

```
ceil23(19)
```

*This code produces the following output:*

```
ans =
```

```
24
```

### 3.1.6 CEIL235 - Next number with only 2,3 and 5 factors

#### Usage

```
nceil=ceil235(n);
```

**Description**

`ceil235(n)` returns the next number greater than or equal to  $n$ , which can be written as a product of powers of 2, 3 and 5.

The algorithm will look up the best size in a table, which is computed the first time the function is run. If the input size is larger than the largest value in the table, the input size will be reduced by factors of 2, until it is in range.

`[nceil,table]=ceil235(n)` additionally returns the table used for lookup.

**Examples:**

Return the first number larger or equal to 19 that can be written solely as products of powers of 2, 3 and 5:

```
ceil235(19)
```

*This code produces the following output:*

```
ans =
```

```
20
```

**3.1.7 NEXTFASTFFT - Next higher number with a fast FFT****Usage**

```
nfft=nextfastfft(n);
```

**Description**

`nextfastfft(n)` returns the next number greater than or equal to  $n$ , for which the computation of a FFT is fast. Such a number is solely comprised of small prime-factors of 2, 3, 5 and 7.

`nextfastfft` is intended as a replacement of `nextpow2`, which is often used for the same purpose. However, a modern FFT implementation (like FFTW) usually performs well for sizes which are powers of 2, 3, 5 and 7, and not only just for powers of 2.

The algorithm will look up the best size in a table, which is computed the first time the function is run. If the input size is larger than the largest value in the table, the input size will be reduced by factors of 2, until it is in range.

`[n,nfft]=nextfastfft(n)` additionally returns the table used for lookup.

**References:** [31], [21], [79]

**3.2 Basic Fourier analysis****3.2.1 DFT - Normalized Discrete Fourier Transform****Usage**

```
f=dft(f);
f=dft(f,N,dim);
```

**Description**

`dft` computes a normalized discrete Fourier transform. This is nothing but a scaled version of the output from `fft`. The function takes exactly the same arguments as `fft`. See the help on `fft` for a thorough description.

### 3.2.2 IDFT - Inverse DFT

#### Usage

```
f=idft(f);
f=idft(f,N,dim);
```

#### Description

This function computes a normalized inverse discrete Fourier transform. This is nothing but a scaled version of the output from `ifft`. The function takes exactly the same arguments as `ifft`. See the help on `ifft` for a thorough description.

### 3.2.3 FFTREAL - FFT for real valued input data

#### Usage

```
f=fftreal(f);
f=fftreal(f,N,dim);
```

#### Description

`fftreal(f)` computes the coefficients corresponding to the positive frequencies of the FFT of the real valued input signal  $f$ .

The function takes exactly the same arguments as `fft`. See the help on `fft` for a thorough description.

### 3.2.4 IFFTREAL - Inverse FFT for real valued signals

#### Usage

```
f=ifftreal(c,N);
f=ifftreal(c,N,dim);
```

#### Description

`ifftreal(c,N)` computes an inverse FFT of the positive frequency Fourier coefficients  $c$ . The length  $N$  must always be specified, because the correct transform length cannot be determined from the size of  $c$ .

`ifftreal(c,N,dim)` does the same along dimension  $dim$ .

### 3.2.5 GGA - Generalized Goertzel algorithm

#### Usage

```
c = gga(x,fvec)
c = gga(x,fvec,fs)
```

#### Input parameters

<b>x</b>	Input data.
<b>fvec</b>	Indices to calculate.
<b>fs</b>	Sampling frequency.

#### Output parameters

<b>c</b>	Coefficient vector.
----------	---------------------

### Description

`c=gga(f,fvec)` computes the discrete-time fourier transform DTFT of  $f$  at frequencies in `fvec` as  $c(k) = F(2\pi f_{vec}(k))$  where  $F = DTFT(f)$ ,  $k = 1, \dots, K$  and  $K = \text{length}(fvec)$  using the generalized second-order Goertzel algorithm. Thanks to the generalization, values in `fvec` can be arbitrary numbers in range  $0 - 1$  and not restricted to  $l/L_s$ ,  $l = 0, \dots, L_s - 1$  (usual DFT samples) as the original Goertzel algorithm is.  $L_s$  is the length of the first non-singleton dimension of  $f$ . If `fvec` is empty or omitted, `fvec` is assumed to be  $(0:L_s-1)/L_s$  and results in the same output as `fft`.

`c=gga(f,fvec,fs)` computes the same with `fvec` in Hz relative to `fs`.

The input  $f$  is processed along the first non-singleton dimension or along dimension `dim` if specified.

**Remark:** Besides the generalization the algorithm is also shortened by one iteration compared to the conventional Goertzel.

### Examples:

Calculating DTFT samples of interest:

```
% Generate input signal
fs = 8000;
L = 2^10;
k = (0:L-1).';
freq = [400,510,620,680,825];
phase = [pi/4,-pi/4,-pi/8,pi/4,-pi/3];
amp = [5,3,4,1,2];
f = arrayfun(@(a,f,p) a*sin(2*pi*k*f/fs+p),...
    amp,freq,phase,'UniformOutput',0);
f = sum(cell2mat(f),2);

% This is equal to fft(f)
ck = gga(f);

%GGA to FFT error:
norm(ck-fft(f))

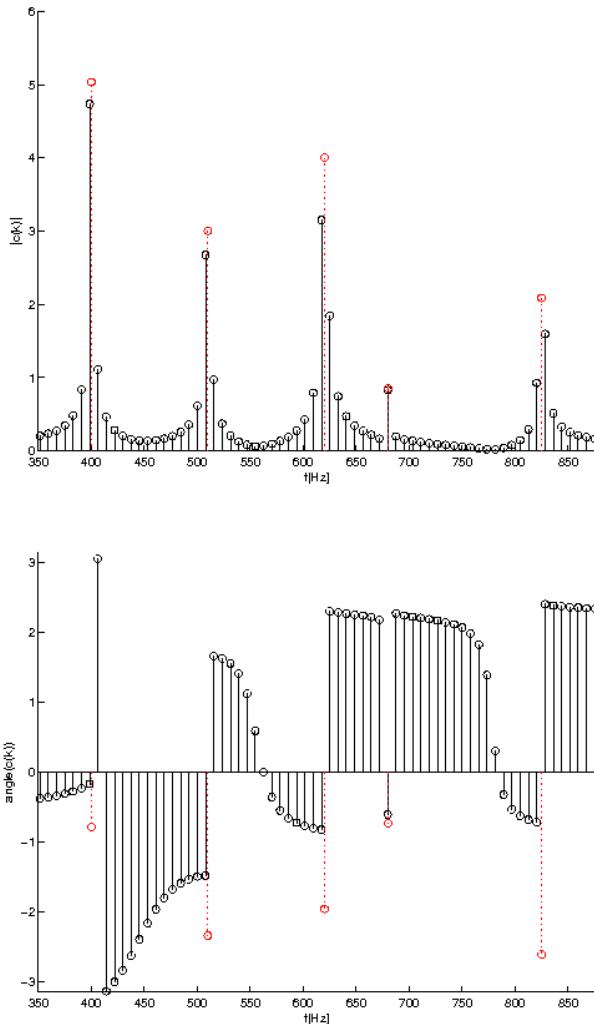
% DTFT samples at 400,510,620,680,825 Hz
ckgga = gga(f,freq,fs);

% Plot modulus of coefficients
figure(1);clf;hold on;
stem(k/L*fs,2*abs(ck)/L,'k');
stem(freq,2*abs(ckgga)/L,'r:');
set(gca,'XLim',[freq(1)-50,freq(end)+50]);
set(gca,'YLim',[0 6]);
xlabel('f[Hz]');
ylabel('|c(k)|');
hold off;

% Plot phase of coefficients
figure(2);clf;hold on;
stem(k/L*fs,angle(ck),'k');
stem(freq,angle(ckgga),'r:');
set(gca,'XLim',[freq(1)-50,freq(end)+50]);
set(gca,'YLim',[-pi pi]);
xlabel('f[Hz]');
ylabel('angle(c(k))');
hold off;
```

This code produces the following output:

```
ans =
1.6050e-09
```



**References:** [81]

### 3.2.6 CHIRPZT - Chirped Z-transform

#### Usage

```
c = chirpz(f,K,fdiff)
c = chirpz(f,K,fdiff,foff)
c = chirpz(f,K,fdiff,foff,fs)
```

#### Input parameters

<b>f</b>	Input data.
<b>K</b>	Number of values.
<b>fdiff</b>	Frequency increment.
<b>foff</b>	Starting frequency.
<b>fs</b>	Sampling frequency.

## Output parameters

**c** Coefficient vector.

## Description

`c = chirpzt(f, K, fdiff, foff)` computes  $K$  samples of the discrete-time fourier transform DTFT  $c$  of  $f$  at values  $c(k+1) = F(2\pi(f_{off} + kf_{diff}))$  for  $k = 0, \dots, K-1$  where  $F = DTFT(f)$ . Values `foff` and `fdiff` should be in range of 0–1. If `foff` is omitted or empty, it is considered to be 0. If `fdiff` is omitted or empty,  $K$  equidistant values  $c(k+1) = F(2\pi k/K)$  are computed. If even  $K$  is omitted or empty, input length is used instead resulting in the same values as `fft` does.

`c = chirpzt(f, K, fdiff, foff, fs)` computes coefficients using frequency values relative to `fs`  $c(k+1) = F(2\pi(f_{off} + kf_{diff})/fs)$  for  $k = 0, \dots, K-1$ .

The input `f` is processed along the first non-singleton dimension or along dimension `dim` if specified.

## Examples:

Calculating DTFT samples of interest (aka zoom FFT):

```
% Generate input signal
fs = 8000;
L = 2^10;
k = (0:L-1).';
f1 = 400;
f2 = 825;
f = 5*sin(2*pi*k*f1/fs + pi/4) + 2*sin(2*pi*k*f2/fs - pi/3);

% This is equal to fft(f)
ck = chirpzt(f,L);

%chirpzt to FFT error:
norm(ck-fft(f))

% Frequency "resolution" in Hz
fdiff = 0.4;
% Frequency offset in Hz
foff = 803.9;
% Number of frequency values
K = 125;
% DTFT samples. The frequency range of interest is 803.9-853.5 Hz
ckchzt = chirpzt(f,K,fdiff,foff,fs);

% Plot modulus of coefficients
figure(1);
fax=foff+fdiff.* (0:K-1);
hold on;
stem(k/L*fs,abs(ck),'k');
stem(fax,abs(ckchzt),'r:');
set(gca,'XLim',[foff,foff+K*fdiff]);
set(gca,'YLim',[0 1065]);
xlabel('f[Hz]');
ylabel('|ck|');

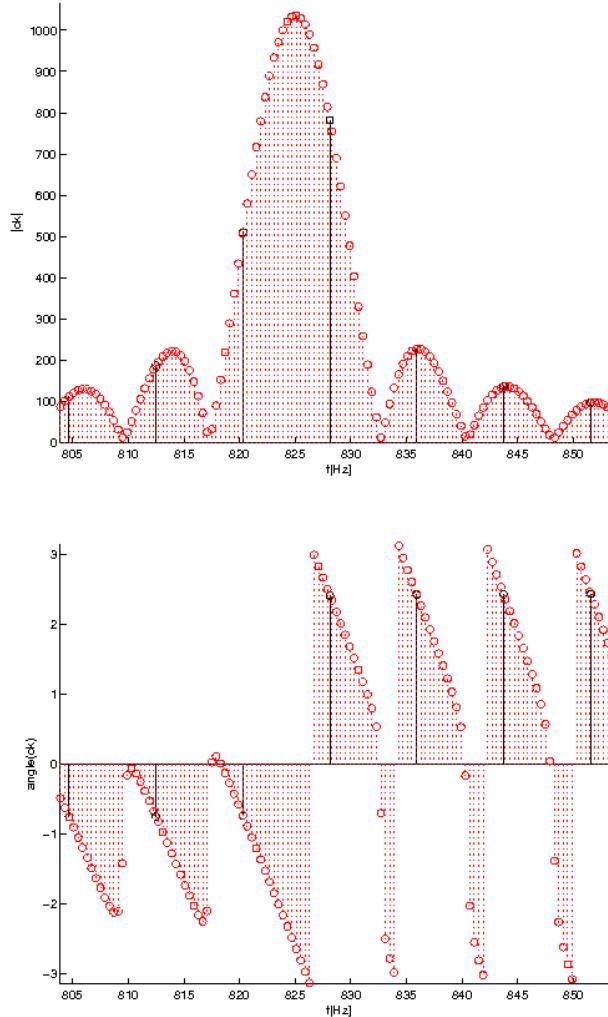
% Plot phase of coefficients
figure(2);
hold on;
stem(k/L*fs,angle(ck),'k');
```

```
stem(fax,angle(ckchzt),'r:');
set(gca,'XLim',[foff,foff+K*fdiff]);
set(gca,'YLim',[-pi pi]);
xlabel('f[Hz]');
ylabel('angle(ck)');
```

This code produces the following output:

```
ans =
```

```
5.9440e-10
```



**References:** [69]

### 3.2.7 PLOTFFT - Plot the output from FFT

#### Usage

```
plotfft(coef);
plotfft(coef,fs);
```

#### Description

`plotfft(coef)` plots the output from the `fft` function. The frequency axis will use normalized frequencies between 0 and 1 (the Nyquist frequency).

`plotfft (coef, fs)` does the same for the FFT of a signal sampled at a sampling rate of  $fs$  Hz.  
`plotfft (coef, fs, dynrange)` additionally limits the dynamic range of the plot. See the description of the 'dynrange' parameter below.  
`plotfft` accepts the following optional arguments:

<b>'dynrange',r</b>	Limit the dynamical range to $r$ by using a colormap in the interval $[chigh-r, chigh]$ , where $chigh$ is the highest value in the plot. The default value of [] means to not limit the dynamical range.
<b>'db'</b>	Apply $20 \cdot \log_{10}$ to the coefficients. This makes it possible to see very weak phenomena, but it might show too much noise. This is the default.
<b>'dbsq'</b>	Apply $10 \cdot \log_{10}$ to the coefficients. Same as the 'db' option, but assumes that the input is already squared.
<b>'lin'</b>	Show the coefficients on a linear scale. This will display the raw input without any modifications. Only works for real-valued input.
<b>'linsq'</b>	Show the square of the coefficients on a linear scale.
<b>'linabs'</b>	Show the absolute value of the coefficients on a linear scale.
<b>'nf'</b>	Display negative frequencies, with the zero-frequency centered in the middle. This is the default.
<b>'posfreq'</b>	Display only the positive frequencies.
<b>'dim',dim</b>	If <code>coef</code> is multidimensional, <code>dim</code> indicates the dimension along which are the individual channels oriented. Value 1 indicates columns, value 2 rows.
<b>'flog'</b>	Use logarithmic scale for the frequency axis. This flag is only valid in conjunction with the 'posfreq' flag.

In addition to these parameters, `plotfft` accepts any of the flags from `normalize`. The coefficients will be normalized as specified before plotting.

### 3.2.8 PLOTFFTREAL - Plot the output from FFTREAL

#### Usage

```
plotfftreal(coef);
plotfftreal(coef, fs);
```

#### Description

`plotfftreal (coef)` plots the output from the `fftreal` function. The frequency axis will use normalized frequencies between 0 and 1 (the Nyquist frequency). It is assumed that the length of the original transform was even.

`plotfftreal (coef, fs)` does the same for the `fftreal` of a signal sampled at a sampling rate of  $fs$  Hz.

`plotfftreal (coef, fs, dynrange)` additionally limits the dynamic range of the plot. See the description of the 'dynrange' parameter below.

`plotfftreal` accepts the following optional arguments:

<b>'dynrange',r</b>	Limit the dynamical range to $r$ by using a colormap in the interval $[chigh-r, chigh]$ , where $chigh$ is the highest value in the plot. The default value of [] means to not limit the dynamical range.
---------------------	---

<b>'db'</b>	Apply $20 \cdot \log_{10}$ to the coefficients. This makes it possible to see very weak phenomena, but it might show too much noise. This is the default.
<b>'dbsq'</b>	Apply $10 \cdot \log_{10}$ to the coefficients. Same as the ' <code>db</code> ' option, but assumes that the input is already squared.
<b>'lin'</b>	Show the coefficients on a linear scale. This will display the raw input without any modifications. Only works for real-valued input.
<b>'linsq'</b>	Show the square of the coefficients on a linear scale.
<b>'linabs'</b>	Show the absolute value of the coefficients on a linear scale.
<b>'N',N</b>	Specify the transform length $N$ . Use this if you are unsure if the original input signal was of even length.
<b>'dim',dim</b>	If <code>coef</code> is multidimensional, <code>dim</code> indicates the dimension along which are the individual channels oriented. Value 1 indicates columns, value 2 rows.
<b>'flog'</b>	Use logarithmic scale for the frequency axis.

In addition to these parameters, `plotfftreal` accepts any of the flags from `normalize`. The coefficients will be normalized as specified before plotting.

### 3.3 Simple operations on periodic functions

#### 3.3.1 INVOLUTE - Involution

##### Usage

```
finv=involute(f);
finv=involute(f,dim);
```

##### Description

`involute(f)` will return the involution of  $f$ .

`involute(f, dim)` will return the involution of  $f$  along dimension  $dim$ . This can for instance be used to calculate the 2D involution:

```
f=involute(f,1);
f=involute(f,2);
```

The involution  $finv$  of  $f$  is given by:

```
finv(l+1)=conj(f(mod(-l,L)+1));
```

for  $l = 0, \dots, L-1$ .

The relation between conjugation, Fourier transformation and involution is expressed by:

```
conj(dft(f)) == dft(involute(f))
```

for all signals  $f$ . The inverse discrete Fourier transform can be expressed by:

```
idft(f) == conj(involute(dft(f)));
```

#### 3.3.2 PEVEN - Even part of periodic function

##### Usage

```
fe=peven(f);
fe=peven(f,dim);
```

**Description**

`peven(f)` returns the even part of the periodic sequence  $f$ .  
`peven(f, dim)` does the same along dimension  $dim$ .

**3.3.3 PODD - Odd part of periodic function****Usage**

```
fe=podd(f);
fe=podd(f, dim);
```

**Description**

`podd(f)` returns the odd part of the periodic sequence  $f$ .  
`podd(f, dim)` does the same along dimension  $dim$ .

**3.3.4 PCONV - Periodic convolution****Usage**

```
h=pconv(f, g)
h=pconv(f, g, ftype);
```

**Description**

`pconv(f, g)` computes the periodic convolution of  $f$  and  $g$ . The convolution is given by

$$h(l+1) = \sum_{k=0}^{L-1} f(k+1)g(l-k+1)$$

`pconv(f, g, 'r')` computes the convolution where  $g$  is reversed (involtuted) given by

$$h(l+1) = \sum_{k=0}^{L-1} f(k+1)\overline{g(k-l+1)}$$

This type of convolution is also known as cross-correlation.

`pconv(f, g, 'rr')` computes the alternative where both  $f$  and  $g$  are reversed given by

$$h(l+1) = \sum_{k=0}^{L-1} f(-k+1)g(l-k+1)$$

In the above formulas,  $l-k$ ,  $k-l$  and  $-k$  are computed modulo  $L$ .

**3.3.5 CONVOLVE - Convolution****Usage**

```
h=convolve(f, g);
```

**Description**

`convolve(f, g)` will convolve two vectors  $f$  and  $g$ . The convolution is not periodic, so the output will have a length of

```
output_len = length(f)+length(g)-1;
```

This function works entirely similar to the Matlab routine `conv` using instead a fast FFT algorithm, making it much faster if one or more of the signals are long.

### 3.3.6 PXCORR - Periodic cross correlation

#### Usage

```
h=pxcorr(f, g)
```

#### Description

`pxcorr(f, g)` computes the periodic cross correlation of the input signals  $f$  and  $g$ . The cross correlation is defined by

$$h(l+1) = \sum_{k=0}^{L-1} f(k+1) \overline{g(k-l+1)}$$

In the above formula,  $k-l$  is computed modulo  $L$ .

`pxcorr(f, g, 'normalize')` does the same, but normalizes the output by the product of the  $l^2$ -norm of  $f$  and  $g$ .

### 3.3.7 ISEVENFUNCTION - True if function is even

#### Usage

```
t=isevenfunction(f);  
t=isevenfunction(f, tol);
```

#### Description

`isevenfunction(f)` returns 1 if  $f$  is whole point even. Otherwise it returns 0.

`isevenfunction(f, tol)` does the same, using the tolerance  $tol$  to measure how large the error between the two parts of the vector can be. Default is  $1e-10$ .

Adding the flag '`hp`' as the last argument does the same for half point even functions.

### 3.3.8 MIDDLEPAD - Symmetrically zero-extends or cuts a function

#### Usage

```
h=middlepad(f, L);  
h=middlepad(f, L, dim);  
h=middlepad(f, L, ...);
```

#### Description

`middlepad(f, L)` cuts or zero-extends  $f$  to length  $L$  by inserting zeros in the middle of the vector, or by cutting in the middle of the vector.

If  $f$  is whole-point even, `middlepad(f, L)` will also be whole-point even.

`middlepad(f, L, dim)` does the same along dimension  $dim$ .

If  $f$  has even length, then  $f$  will not be purely zero-extended, but the last element will be repeated once and multiplied by 1/2. That is, the support of  $f$  will increase by one!

Adding the flag '`wp`' as the last argument will cut or extend whole point even functions. Adding '`hp`' will do the same for half point even functions.

## 3.4 Functions

### 3.4.1 EXPWAVE - Complex exponential wave

#### Usage

```
h=expwave(L, m);  
h=expwave(L, m, cent);
```

**Description**

`expwave(L, m)` returns an exponential wave revolving  $m$  times around the origin. The collection of all waves with wave number  $m = 0, \dots, L - 1$  forms the basis of the discrete Fourier transform.

The wave has absolute value 1 everywhere. To get an exponential wave with unit  $l^2$ -norm, divide the wave by  $\sqrt{L}$ . This is the normalization used in the `dft` function.

`expwave(L, m, cent)` makes it possible to shift the sampling points by the amount *cent*. Default is *cent* = 0.

**3.4.2 PCHIRP - Periodic chirp****Usage**

```
g=pchirp(L, n);
```

**Description**

`pchirp(L, n)` returns a periodic, discrete chirp of length  $L$  that revolves  $n$  times around the time-frequency plane in frequency.  $n$  must be an integer number.

To get a chirp that revolves around the time-frequency plane in time, use

```
dft(pchirp(L, N));
```

The chirp is computed by:

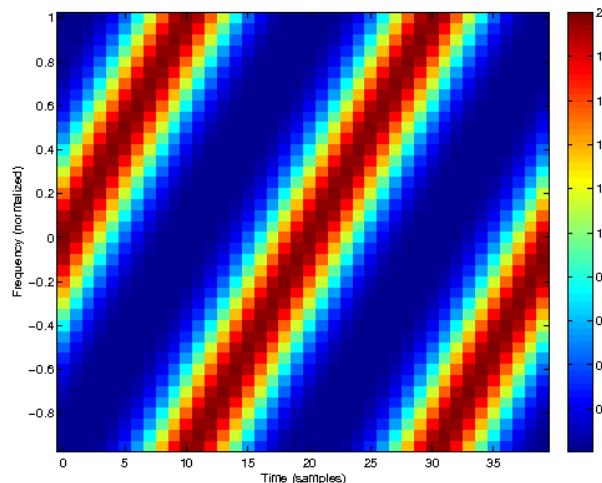
$$g(l+1) = e^{\pi i n(l-\lceil L/2 \rceil)^2(L+1)/L}, \quad l = 0, \dots, L-1$$

The chirp has absolute value 1 everywhere. To get a chirp with unit  $l^2$ -norm, divide the chirp by  $\sqrt{L}$ .

**Examples:**

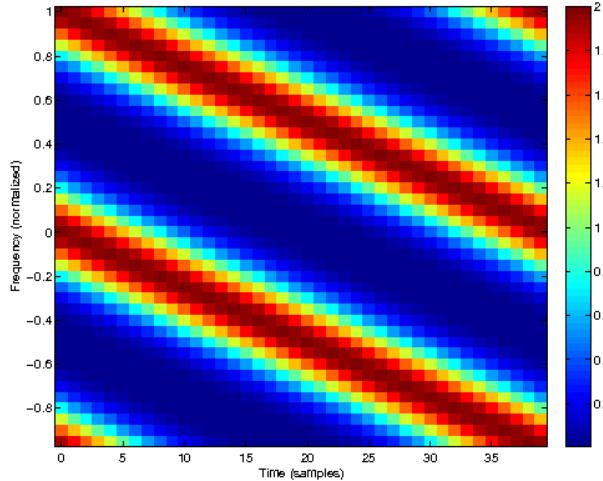
A spectrogram on a linear scale of an even length chirp:

```
sgram(pchirp(40, 2), 'lin');
```



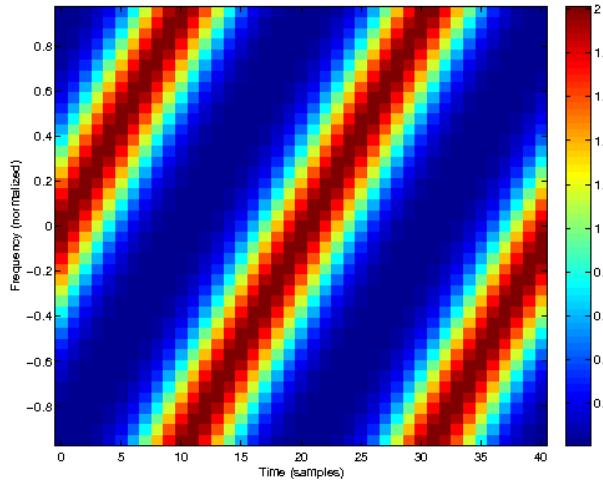
The DFT of the same chirp, now revolving around in time:

```
sgram(dft(pchirp(40, 2)), 'lin');
```



An odd-length chirp. Notice that the chirp starts at a frequency between two sampling points:

```
sgram(pchirp(41,2),'lin');
```



**References:** [28]

### 3.4.3 SHAH - Discrete Shah-distribution

#### Usage

```
f=shah(L,a);
```

#### Description

`shah(L,a)` computes the discrete, normalized Shah-distribution of length  $L$  with a distance of  $a$  between the spikes.

The Shah distribution is defined by

$$f(n \cdot a + 1) = \frac{1}{\sqrt{(L/a)}}$$

for integer  $n$ , otherwise  $f$  is zero.

This is also known as an impulse train or as the comb function, because the shape of the function resembles a comb. It is the sum of unit impulses ('diracs') with the distance  $a$ .

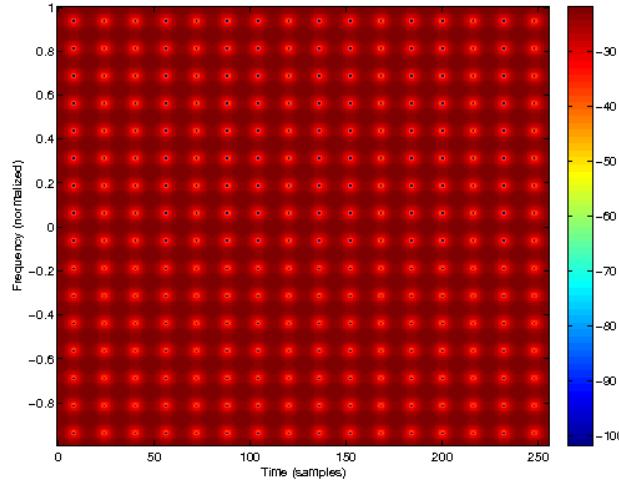
If  $a$  divides  $L$ , then the dft of  $\text{shah}(L, a)$  is  $\text{shah}(L, L/a)$ .

The Shah function has an extremely bad time-frequency localization. It does not generate a Gabor frame for any  $L$  and  $a$ .

### Examples:

A simple spectrogram of the Shah function (includes the negative frequencies to display the whole TF-plane):

```
sgram(shah(256,16), 'dynrange', 80, 'nf')
```



### 3.4.4 PHEAVISIDE - Periodic Heaviside function

#### Usage

```
h=pheaviside(L);
```

#### Description

`pheaviside(L)` returns a periodic Heaviside function. The periodic Heaviside function takes on the value 1 for indices corresponding to positive frequencies, 0 corresponding to negative frequencies and the value .5 for the zero and Nyquist frequencies.

To get a function that weights the negative frequencies by 1 and the positive by 0, use `involute(pheaviside(L))`

As an example, the `pheaviside` function can be used to calculate the Hilbert transform for a column vector  $f$ :

```
h=2*ifft(fft(f).*pheaviside(length(f)));
```

### 3.4.5 PRECT - Periodic rectangle

#### Usage

```
f=prect(L,n);
```

#### Description

`psinc(L,n)` computes the periodic rectangle (or square) function of length  $L$  supported on  $n$  samples. The dft of the periodic rectangle function is the periodic sinc function, `psinc`.

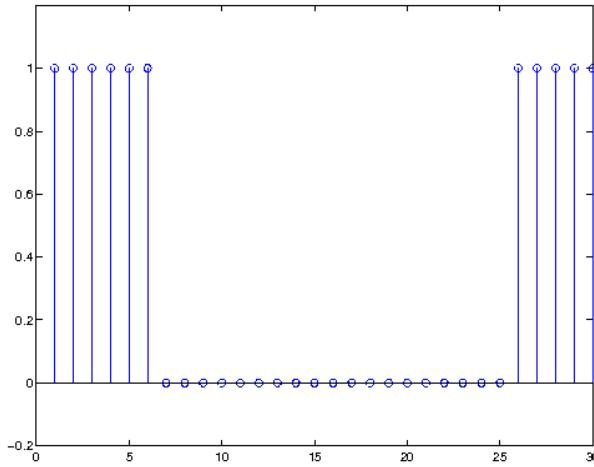
- If  $n$  is odd, the output will be supported on exactly  $n$  samples centered around the first sample.

- If  $n$  is even, the output will be supported on exactly  $n+1$  samples centered around the first sample. The function value on the two samples on the edge of the function will have half the magnitude of the other samples.

**Examples:**

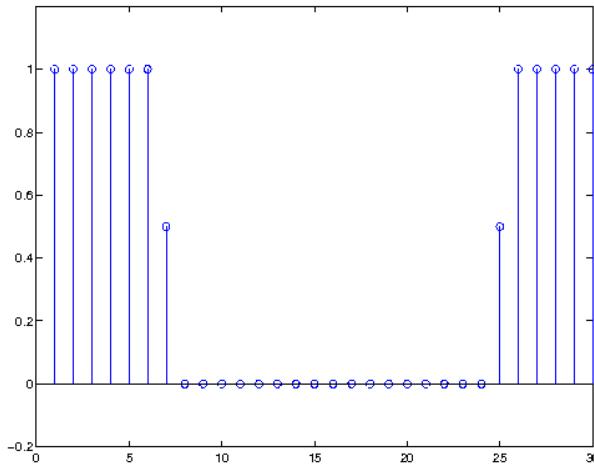
This figure displays an odd length periodic rectangle:

```
stem(prect(30,11));
ylim([- .2 1.2]);
```



This figure displays an even length periodic rectangle. Notice the border points:

```
stem(prect(30,12));
ylim([- .2 1.2]);
```



### 3.4.6 PSINC - Periodic Sinc function (Dirichlet function)

**Usage**

```
f=psinc(L,n);
```

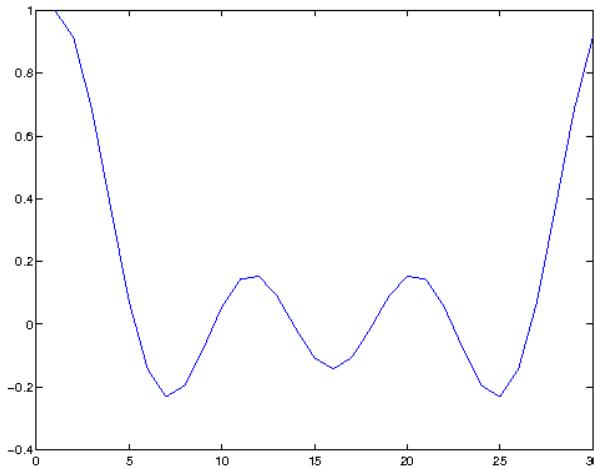
### Description

`psinc(L, n)` computes the periodic Sinc function of length  $L$  with  $n - 1$  local extrema. The dft of the periodic Sinc function is the periodic rectangle, `prect`, of length  $n$ .

### Examples:

This figure displays a the periodic sinc function with 6 local extremas:

```
plot(psinc(30, 7));
```



## 3.5 Window functions

### 3.5.1 PGAUSS - Sampled, periodized Gaussian

#### Usage

```
g=pgauss(L);
g=pgauss(L,tfr);
g=pgauss(L,...);
[g,tfr]=pgauss( ... );
```

#### Input parameters

- |            |   |
|------------|---|
| <b>L</b>   | Length of vector.                         |
| <b>tfr</b> | ratio between time and frequency support. |

#### Output parameters

- |          |                          |
|----------|--------------------------|
| <b>g</b> | The periodized Gaussian. |
|----------|--------------------------|

### Description

`pgauss(L, tfr)` computes samples of a periodized Gaussian. The function returns a regular sampling of the periodization of the function  $\exp(-pi*(x.^2/tfr))$ .

The  $l^2$  norm of the returned Gaussian is equal to 1.

The parameter `tfr` determines the ratio between the effective support of `g` and the effective support of the DFT of `g`. If `tfr > 1` then `g` has a wider support than the DFT of `g`.

`pgauss(L)` does the same setting `tfr=1`.

`[g,tfr] = pgauss( ... )` will additionally return the time-to-frequency support ratio. This is useful if you did not specify it (i.e. used the '`width`' or '`bw`' flag).

The function is whole-point even. This implies that `fft(pgauss(L,tfr))` is real for any  $L$  and  $tfr$ . The DFT of  $g$  is equal to `pgauss(L, 1/tfr)`.

In addition to the '`width`' flag, `pgauss` understands the following flags at the end of the list of input parameters:

<b>'fs',fs</b>	Use a sampling rate of $fs$ Hz as unit for specifying the width, bandwidth, centre frequency and delay of the Gaussian. Default is $fs=1$ which indicates to measure everything in samples.
<b>'width',s</b>	Set the width of the Gaussian such that it has an effective support of $s$ samples. This means that approx. 96% of the energy or 79% of the area under the graph is contained within $s$ samples. This corresponds to a -6 db cutoff. This is equivalent to calling <code>pgauss(L, s^2/L)</code> .
<b>'bw',bw</b>	As for the ' <code>width</code> ' argument, but specifies the width in the frequency domain. The bandwidth is measured in normalized frequencies, unless the ' <code>fs</code> ' value is given.
<b>'cf',cf</b>	Set the centre frequency of the Gaussian to $fc$ .
<b>'wp'</b>	Output is whole point even. This is the default.
<b>'hp'</b>	Output is half point even, as most Matlab filter routines.
<b>'delay',d</b>	Delay the output by $d$ . Default is zero delay.

In addition to these parameteres, `pgauss` accepts any of the flags from `normalize`. The output will be normalized as specified.

If this function is used to generate a window for a Gabor frame, then the window giving the smallest frame bound ratio is generated by `pgauss(L, a*M/L)`.

### Examples:

This example creates a Gaussian function, and demonstrates that it is its own Discrete Fourier Transform:

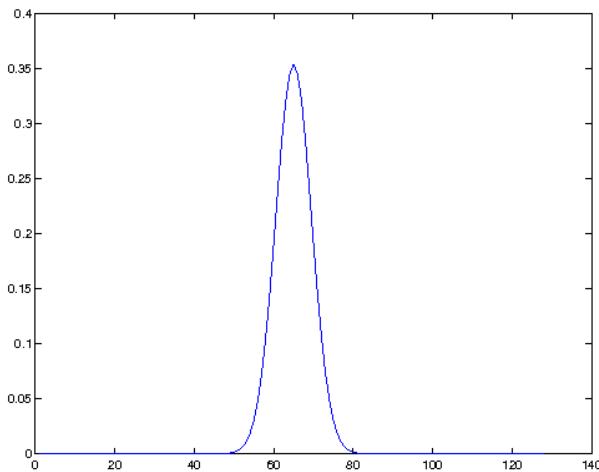
```
g=pgauss(128);
% Test of DFT invariance: Should be close to zero.
norm(g-dft(g))
```

*This code produces the following output:*

```
ans =
2.4815e-15
```

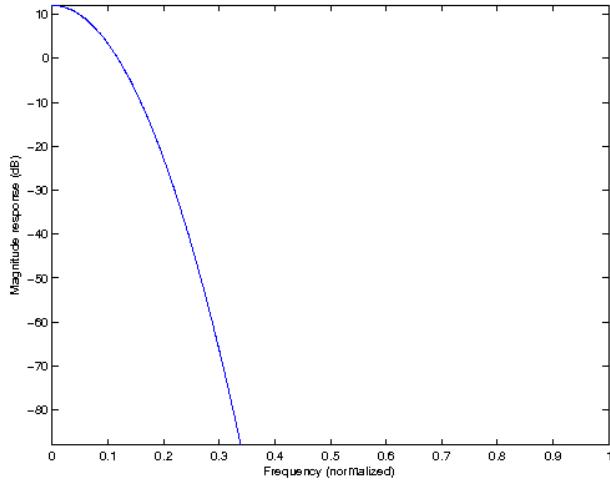
The next plot shows the Gaussian in the time domain:

```
plot(fftshift(pgauss(128)));
```



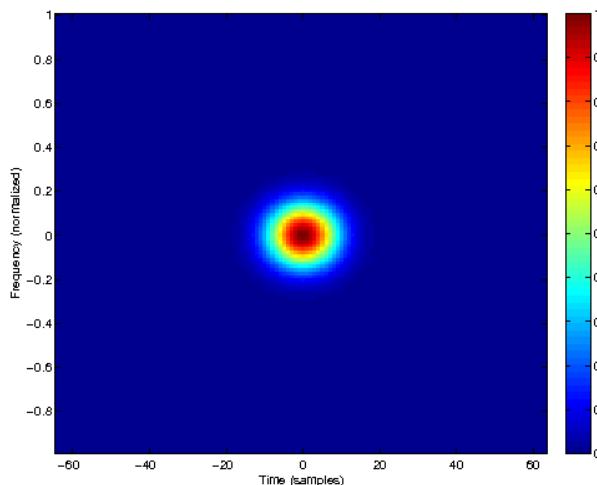
The next plot shows the Gaussian in the frequency domain on a log scale:

```
magresp(pgauss(128), 'dynrange', 100);
```



The next plot shows the Gaussian in the time-frequency plane:

```
sgram(pgauss(128), 'tc', 'nf', 'lin');
```



**References:** [56]

### 3.5.2 PSECH - Sampled, periodized hyperbolic secant

#### Usage

```
g=psech(L);
g=psech(L,tfr);
g=psech(L,s,'samples');
[g,tfr]=psech( ... );
```

#### Input parameters

**L** Length of vector.

**tfr** ratio between time and frequency support.

#### Output parameters

**g** The periodized hyperbolic cosine.

#### Description

`psech(L,tfr)` computes samples of a periodized hyperbolic secant. The function returns a regular sampling of the periodization of the function

The returned function has norm equal to 1.

The parameter *tfr* determines the ratio between the effective support of *g* and the effective support of the DFT of *g*. If *tfr* > 1 then *g* has a wider support than the DFT of *g*.

`psech(L)` does the same setting *tfr* = 1.

`psech(L,s,'samples')` returns a hyperbolic secant with an effective support of *s* samples. This means that approx. 96% of the energy or 74% of the area under the graph is contained within *s* samples. This is equivalent to `psech(L,s^2/L)`.

`[g,tfr] = psech( ... )` additionally returns the time-to-frequency support ratio. This is useful if you did not specify it (i.e. used the '*'samples'* input format).

The function is whole-point even. This implies that `fft(psech(L,tfr))` is real for any *L* and *tfr*.

If this function is used to generate a window for a Gabor frame, then the window giving the smallest frame bound ratio is generated by `psech(L,a*M/L)`.

**Examples:**

This example creates a `psech` function, and demonstrates that it is its own Discrete Fourier Transform:

```
g=psech(128);

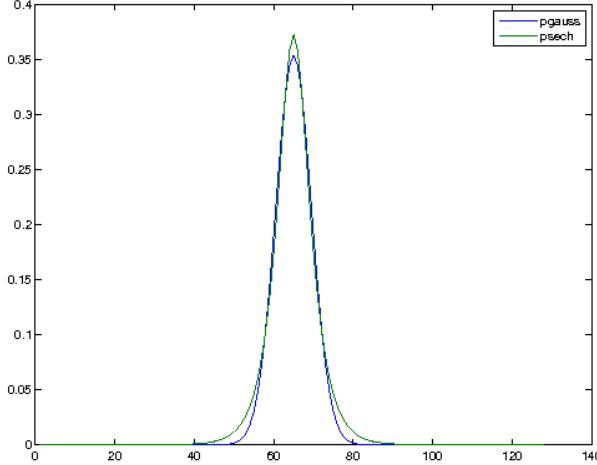
% Test of DFT invariance: Should be close to zero.
norm(g-dft(g))
```

*This code produces the following output:*

```
ans =
2.4003e-15
```

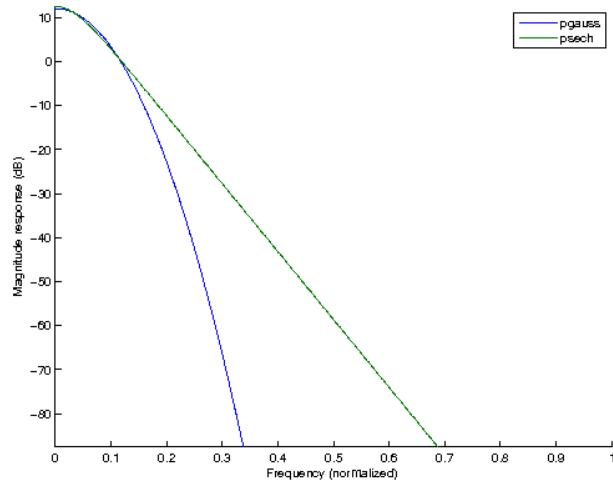
The next plot shows the `psech` in the time domain compared to the Gaussian:

```
plot((1:128)',fftshift(pgauss(128)),...
      (1:128)',fftshift(psech(128)));
legend('pgauss','psech');
```



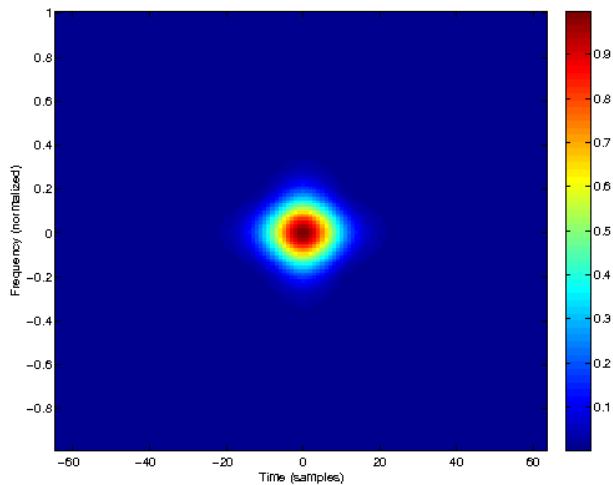
The next plot shows the `psech` in the frequency domain on a log scale compared to the Gaussian:

```
hold all;
magresp(pgauss(128),'dynrange',100);
magresp(psech(128),'dynrange',100);
legend('pgauss','psech');
```



The next plot shows psech in the time-frequency plane:

```
sgram(psech(128), 'tc', 'nf', 'lin');
```



**References:** [42]

### 3.5.3 PBSPLINE - Periodized B-spline

#### Usage

```
g=pbspline(L,order,a,...);
[g,nlen]=pbspline(L,order,a,...);
```

#### Input parameters

- L** Length of window.
- order** Order of B-spline.
- a** Time-shift parameter for partition of unity.

#### Output parameters

- g** Fractional B-spline.
- nlen** Number of non-zero elements in out.

**Description**

`pbspline(L, order, a)` computes a (slightly modified) B-spline of order *order* of total length *L*.

If shifted by the distance *a*, the returned function will form a partition of unity. The result is normalized such that the functions sum to  $1/\sqrt(a)$ .

`pbspline` takes the following flags at the end of the input arguments:

'ed'	Even discrete fractional spline. This is the default
'xd'	'flat' discrete fractional spline.
'stard'" pointy discrete fractional spline	
'ec''	Even fractional spline by sampling.
'xc'	'flat' fractional spline by sampling.
'starc'" pointy fractional spline by sampling.	
'wp''	Generate whole point centered splines. This is the default.
'hp'	Generate half point centered splines.

The different types are accurately described in the referenced paper. Generally, the 'd' types of splines are very fast to compute, while the 'c' types are samplings of the continuous splines. The 'e' types coincides with the regular B-splines for integer orders. The 'x' types do not coincide, but generate Gabor frames with favorable frame bounds. The default type is 'ed' to guarantee fast computation and a familiar shape of the splines.

`[out, nlen]=pbspline( . . . )` will additionally compute the number of non-zero elements in *out*.

If *nlen* = *L*, the function returned will be a periodization of a B-spline.

If *nlen* < *L*, you can choose to remove the additional zeros by calling `g=middlepad(g, nlen)`.

Additionally, `pbspline` accepts flags to normalize the output. Please see the help of `normalize`. Default is to use 'peak' normalization.

**References:** [78]

**3.5.4 FIRWIN - FIR window****Usage**

```
g=firwin(name, M);
g=firwin(name, M, . . . );
g=firwin(name, x);
```

**Description**

`firwin(name, M)` will return an FIR window of length *M* of type *name*.

All windows are symmetric and generate zero delay and zero phase filters. They can be used for the Wilson and WMDCT transform, except when noted otherwise.

`firwin(name, x)` where *x* is a vector will sample the window definition as the specified points. The normal sampling interval for the windows is  $-0.5 < x < 0.5$ .

In the following PSL means "Peak Sidelobe level", and the main lobe width is measured in normalized frequencies.

If a window *g* forms a "partition of unity" (PU) it means specifically that:

```
g+fftshift(g)==ones(L, 1);
```

A PU can only be formed if the window length is even, but some windows may work for odd lengths anyway.

If a window is the square root of a window that forms a PU, the window will generate a tight Gabor frame / orthonormal Wilson/WMDCT basis if the number of channels is less than *M*.

The windows available are:

<b>'hann'</b>	von Hann window. Forms a PU. The Hann window has a mainlobe width of 8/M, a PSL of -31.5 dB and decay rate of 18 dB/Octave.
<b>'sine'</b>	Sine window. This is the square root of the Hanning window. The sine window has a mainlobe width of 8/M, a PSL of -22.3 dB and decay rate of 12 dB/Octave. Aliases: 'cosine', 'sqrthann'
<b>'rect'</b>	(Almost) rectangular window. The rectangular window has a mainlobe width of 4/M, a PSL of -13.3 dB and decay rate of 6 dB/Octave. Forms a PU. Alias: 'square'
<b>'sqrtrect'</b>	Square root of the rectangular window.
<b>'tria'</b>	(Almost) triangular window. Forms a PU. Alias: 'bartlett'
<b>'sqrttria'</b>	Square root of the triangular window.
<b>'hamming'</b>	Hamming window. Forms a PU that sums to 1.08 instead of 1.0 as usual. The Hamming window has a mainlobe width of 8/M, a PSL of -42.7 dB and decay rate of 6 dB/Octave. This window should not be used for a Wilson basis, as a reconstruction window cannot be found by wildual.
<b>'blackman'</b>	Blackman window. The Blackman window has a mainlobe width of 12/M, a PSL of -58.1 dB and decay rate of 18 dB/Octave.
<b>'blackman2'</b>	Alternate Blackman window. This window has a mainlobe width of 12/M, a PSL of -68.24 dB and decay rate of 6 dB/Octave.
<b>'itersine'</b>	Iterated sine window. Generates an orthonormal Wilson/WMDCT basis. This window is described in Wesfreid and Wickerhauser (1993) and is used in the ogg sound codec. Alias: 'ogg'
<b>'nuttall'</b>	Nuttall window. The Nuttall window has a mainlobe width of 16/M, a PSL of -93.32 dB and decay rate of 18 dB/Octave.
<b>'nuttall10'</b>	2-term Nuttall window with 1 continuous derivative. Alias: 'hann', 'hanning'.
<b>'nuttall01'</b>	2-term Nuttall window with 0 continuous derivatives. This is a slightly improved Hamming window. It has a mainlobe width of 8/M, a PSL of -43.19 dB and decay rate of 6 dB/Octave.
<b>'nuttall20'</b>	3-term Nuttall window with 3 continuous derivatives. The window has a mainlobe width of 12/M, a PSL of -46.74 dB and decay rate of 30 dB/Octave.
<b>'nuttall11'</b>	3-term Nuttall window with 1 continuous derivative. The window has a mainlobe width of 12/M, a PSL of -64.19 dB and decay rate of 18 dB/Octave.
<b>'nuttall02'</b>	3-term Nuttall window with 0 continuous derivatives. The window has a mainlobe width of 12/M, a PSL of -71.48 dB and decay rate of 6 dB/Octave.
<b>'nuttall30'</b>	4-term Nuttall window with 5 continuous derivatives. The window has a mainlobe width of 16/M, a PSL of -60.95 dB and decay rate of 42 dB/Octave.
<b>'nuttall21'</b>	4-term Nuttall window with 3 continuous derivatives. The window has a mainlobe width of 16/M, a PSL of -82.60 dB and decay rate of 30 dB/Octave.

- 'nuttall12'** 4-term Nuttall window with 1 continuous derivatives. Alias: 'nuttall'.
- 'nuttall03'** 4-term Nuttall window with 0 continuous derivatives. The window has a mainlobe width of  $16/M$ , a PSL of -98.17 dB and decay rate of 6 dB/Octave.

`firwin` understands the following flags at the end of the list of input parameters:

- 'shift',s** Shift the window by  $s$  samples. The value can be a fractional number.
- 'wp'** Output is whole point even. This is the default. It corresponds to a shift of  $s = 0$ .
- 'hp'** Output is half point even, as most Matlab filter routines. This corresponds to a shift of  $s = -.5$
- 'taper',t** Extend the window by a flat section in the middle. The argument  $t$  is the ratio of the rising and falling parts as compared to the total length of the window. The default value of 1 means no tapering. Accepted values lie in the range from 0 to 1.

Additionally, `firwin` accepts flags to normalize the output. Please see the help of `normalize`. Default is to use '`peak`' normalization, which is useful for using the output from `firwin` for windowing in the time-domain. For filtering in the time-domain, a normalization of '`1`' or '`area`' is preferable.

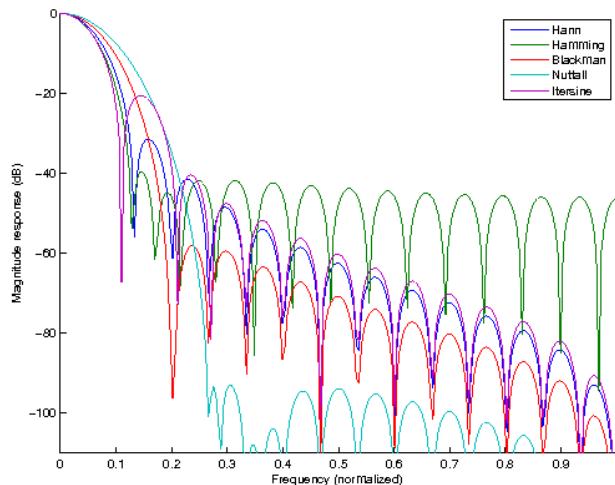
### Examples:

The following plot shows the magnitude response for some common windows:

```
hold all;
L=30;
dr=110;

magresp(firwin('hanning',L,'1'),'fir','dynrange',dr);
magresp(firwin('hamming',L,'1'),'fir','dynrange',dr);
magresp(firwin('blackman',L,'1'),'fir','dynrange',dr);
magresp(firwin('nuttall',L,'1'),'fir','dynrange',dr);
magresp(firwin('itersine',L,'1'),'fir','dynrange',dr);

legend('Hann','Hamming','Blackman','Nuttall','Itersine');
```



**References:** [62], [38], [61], [86]

### 3.5.5 FIRKAISER - Kaiser-Bessel window

#### Usage

```
g=firkaiser(L,beta);
g=firkaiser(L,beta,...);
```

#### Description

`firkaiser(L,beta)` computes the Kaiser-Bessel window of length  $L$  with parameter  $\beta$ . The smallest element of the window is set to zero when the window has an even length. This gives the window perfect whole-point even symmetry, and makes it possible to use the window for a Wilson basis.

`firkaiser` takes the following flags at the end of the input arguments:

<b>'normal'</b>	Normal Kaiser-Bessel window. This is the default.
<b>'derived'</b>	Derived Kaiser-Bessel window.
<b>'wp'</b>	Generate a whole point even window. This is the default.
<b>'hp'</b>	Generate half point even window.

Additionally, `firkaiser` accepts flags to normalize the output. Please see the help of `normalize`. Default is to use '`peak`' normalization.

**References:** [62]

### 3.5.6 FIR2LONG - Extend FIR window to LONG

#### Usage

```
g=fir2long(g,Llong);
```

#### Description

`fir2long(g,Llong)` will extend the FIR window  $g$  to a length  $L_{\text{long}}$  window by inserting zeros. Note that this is a slightly different behaviour than `middlepad`.

`fir2long` can also be used to extend a FIR window to a longer FIR window, for instance in order to satisfy the usual requirement that the window length should be divisible by the number of channels.

If the input to `fir2long` is a cell, `fir2long` will recurse into the cell array.

### 3.5.7 LONG2FIR - Cut LONG window to FIR

#### Usage

```
g=long2fir(g,L);
```

#### Description

`long2fir(g,L)` will cut the LONG window  $g$  to a length  $L$  FIR window by cutting out the middle part. Note that this is a slightly different behaviour than `middlepad`.

`long2fir(g,L,'wp')` or `long2fir(g,L,'hp')` does the same assuming the input window is a whole-point even or half-point even window, respectively.

## 3.6 Filtering

### 3.6.1 FIRFILTER - Construct an FIR filter

#### Usage

```
g=firfilter(name,M);
g=firfilter(name,M,...);
```

### Description

`firfilter(name, M)` creates an FIR filter of length  $M$ . This is exactly the same as calling `firwin`. The name must be one of the accepted window types of `firwin`.

`firfilter(name, M, fc)` constructs a filter with a centre frequency of  $fc$  measured in normalized frequencies.

If one of the inputs is a vector, the output will be a cell array with one entry in the cell array for each element in the vector. If more input are vectors, they must have the same size and shape and the filters will be generated by stepping through the vectors. This is a quick way to create filters for filterbank and ufilterbank.

`firfilter` accepts the following optional parameters:

<b>'fs',fs</b>	If the sampling frequency $fs$ is specified then the length $M$ is specified in seconds and the centre frequency $fc$ in Hz.
<b>'complex'</b>	Make the filter complex valued if the centre frequency is non-zero. This is the default.
<b>'real'</b>	Make the filter real-valued if the centre frequency is non-zero.
<b>'delay',d</b>	Set the delay of the filter. Default value is zero.
<b>'causal'</b>	Create a causal filter starting at the first sample. If specified, this flag overwrites the delay setting.

It is possible to normalize the impulse response of the filter by passing any of the flags from the `normalize` function. The default normalization is '`energy`'.

The filter can be used in the `pfilter` routine to filter a signal, or it can be placed in a cell-array for use with filterbank or ufilterbank.

### 3.6.2 BLFILTER - Construct a band-limited filter

#### Usage

```
g=blfilter(winname, fsupp, fc);
g=blfilter(winname, fsupp, fc, ...);
```

#### Input parameters

<b>winname</b>	Name of prototype
<b>fsupp</b>	Support length of the prototype

#### Description

`blfilter(winname, fsupp)` constructs a band-limited filter. The parameter `winname` specifies the shape of the frequency response. The name must be one of the shapes accepted by `firwin`. The support of the frequency response measured in normalized frequencies is specified by `fsupp`.

`blfilter(winname, fsupp, fc)` constructs a filter with a centre frequency of  $fc$  measured in normalized frequencies.

If one of the inputs is a vector, the output will be a cell array with one entry in the cell array for each element in the vector. If more input are vectors, they must have the same size and shape and the filters will be generated by stepping through the vectors. This is a quick way to create filters for filterbank and ufilterbank.

`blfilter` accepts the following optional parameters:

<b>'fs',fs</b>	If the sampling frequency $fs$ is specified then the support $fsupp$ and the centre frequency $fc$ is specified in Hz.
<b>'complex'</b>	Make the filter complex valued if the centre frequency is non-zero.necessary. This is the default.

<b>'real'</b>	Make the filter real-valued if the centre frequency is non-zero.
<b>'delay',d</b>	Set the delay of the filter. Default value is zero.
<b>'scal',s</b>	Scale the filter by the constant <i>s</i> . This can be useful to equalize channels in a filterbank.
<b>'pedantic'</b>	Force window frequency offset (g. <i>foff</i> ) to a subsample precision by a subsample shift of the firwin output.

It is possible to normalize the transfer function of the filter by passing any of the flags from the normalize function. The default normalization is '`energy`'.

The filter can be used in the `pfilter` routine to filter a signal, or it can be placed in a cell-array for use with `filterbank` or `ufilterbank`.

#### Output format:

The output *g* from `b1filter` is a structure. This type of structure can be used to describe any bandlimited filter defined in terms of its transfer function. The structure contains the following fields:

<b>--Xg_DOT_H</b>	This is an anonymous function taking the transform length <i>L</i> as input and producing the bandlimited transfer function in the form of a vector.
<b>--Xg_DOT_foff</b>	This is an anonymous function taking the transform length <i>L</i> as input and producing the frequency offset of <i>H</i> as an integer. The offset is the value of the lowest frequency of <i>H</i> measured in frequency samples. <i>foff</i> is used to position the bandlimited transfer function stored in <i>H</i> correctly when multiplying in the frequency domain.
<b>--Xg_DOT_delay</b>	This is the desired delay of the filter measured in samples.
<b>--Xg_DOT_realonly</b>	This is an integer with value 1 if the filter defined a real-valued filter. In this case, the bandlimited transfer function <i>H</i> will be mirrored from the positive frequencies to the negative frequencies. If the filter is a natural lowpass filter correctly centered around 0, <code>realonly</code> does not need to be 1.
<b>--Xg_DOT_fs</b>	The intended sampling frequency. This is an optional parameter that is <b>only</b> used for plotting and visualization.

### 3.6.3 WARPEDBLFILTER - Construct a warped band-limited filter

#### Usage

```
g=warpedblfilter(winname,fsupp,fc,fs,freqtoscale);
g=warpedblfilter(winname,fsupp,fc,...);
```

#### Input parameters

<b>winname</b>	Name of prototype.
<b>fsupp</b>	Support length of the prototype (in scale units).
<b>fc</b>	Centre frequency (in Hz).
<b>fs</b>	Sampling rate
<b>freqtoscale</b>	Function handle to convert Hz to scale units
<b>scaletofreq</b>	Function to convert scale units into Hz.

### Output parameters

<b>g</b>	Filter definition, see blfilter.
----------	----------------------------------

### Description

`warpedblfilter(winname, fsupp)` constructs a band-limited filter that is warped on a given frequency scale. The parameter `winname` specifies the basic shape of the frequency response. The name must be one of the shapes accepted by `firwin`. The support of the frequency response measured on the selected frequency scale is specified by `fsupp`, the centre frequency by `fc` and the scale by the function handle `freqtoscale` of a function that converts Hz into the chosen scale.

If one of the inputs is a vector, the output will be a cell array with one entry in the cell array for each element in the vector. If more input are vectors, they must have the same size and shape and the filters will be generated by stepping through the vectors. This is a quick way to create filters for filterbank and ufilterbank.

`warpedblfilter` accepts the following optional parameters:

<b>'complex'</b>	Make the filter complex valued if the centre frequency is non-zero.necessary. This is the default.
<b>'real'</b>	Make the filter real-valued if the centre frequency is non-zero.
<b>'delay',d</b>	Set the delay of the filter. Default value is zero.
<b>'scal',s</b>	Scale the filter by the constant <code>s</code> . This can be useful to equalize channels in a filterbank.

It is possible to normalize the transfer function of the filter by passing any of the flags from the normalize function. The default normalization is '`energy`'.

The filter can be used in the `pfilt` routine to filter a signal, or in can be placed in a cell-array for use with filterbank or ufilterbank.

The output format is the same as that of `blfilter`.

### 3.6.4 PFILT - Apply filter with periodic boundary conditions

#### Usage

```
h=pfilt(f,g);
h=pfilt(f,g,a,dim);
```

#### Description

`pfilt(f,g)` applies the filter `g` to the input `f`. If `f` is a matrix, the filter is applied along each column.

`pfilt(f,g,a)` does the same, but downsamples the output keeping only every `a`'th sample (starting with the first one).

`pfilt(f,g,a,dim)` filters along dimension `dim`. The default value of [] means to filter along the first non-singleton dimension.

The filter `g` can be a vector, in which case the vector is treated as a zero-delay FIR filter.

The filter `g` can be a cell array. The following options are possible:

- If the first element of the cell array is the name of one of the windows from `firwin`, the whole cell array is passed onto `firfilter`.
- If the first element of the cell array is '`bl`', the rest of the cell array is passed onto `blfilter`.
- If the first element of the cell array is '`pgauss`', '`psech`', the rest of the parameters is passed onto the respective function. Note that you do not need to specify the length `L`.

The coefficients obtained from filtering a signal  $f$  by a filter  $g$  are defined by

$$c(n+1) = \sum_{l=0}^{L-1} f(l+1) g(an - l + 1)$$

where  $an - l$  is computed modulo  $L$ .

### 3.6.5 MAGRESP - Magnitude response plot of window

#### Usage

```
magresp(g, ...);
magresp(g, fs, ...);
magresp(g, fs, dynrange, ...);
```

#### Description

`magresp(g)` will display the magnitude response of the window on a log scale (dB);

`magresp(g, fs)` does the same for windows that are intended to be used with signals with sampling rate  $fs$ . The x-axis will display Hz.

`magresp(g, fs, dynrange)` will limit the dynamic range (see below).

`magresp` takes the following parameters at the end of the line of input arguments.

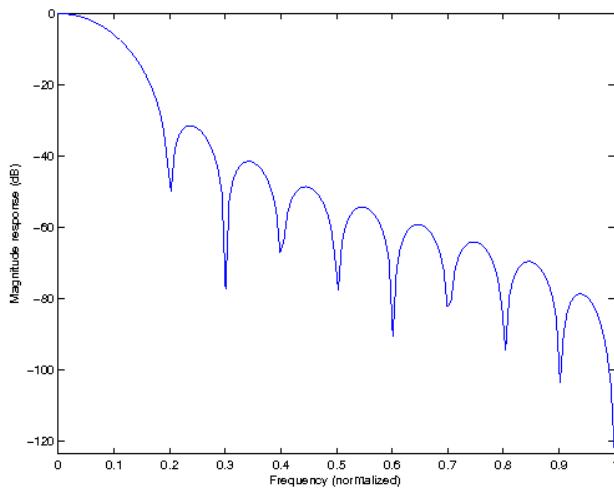
<b>'dynrange',r</b>	Limit the dynamic range of the plot to r dB.
<b>'fir'</b>	Indicate that the input is an FIR window. <code>magresp</code> will zero-extend the window to display a smooth magnitude response.
<b>'L',L</b>	Zero-extend the window to length L.
<b>'posfreq'</b>	Show only positive frequencies.
<b>'nf'</b>	Show also negative frequencies
<b>'autoposfreq'</b>	Show positive frequencies for real-valued signals, otherwise show also the negative frequencies. This is the default.
<b>'opts',op</b>	Pass options onto the plot command. The extra options <i>op</i> are specified as a cell array

In addition to these flags, it is possible to specify any of the normalization flags from `normalize` to normalize the input before calculation of the magnitude response. Specifying '`1`' or '`area`' will display a magnitude response which peaks at 0 dB.

#### Examples:

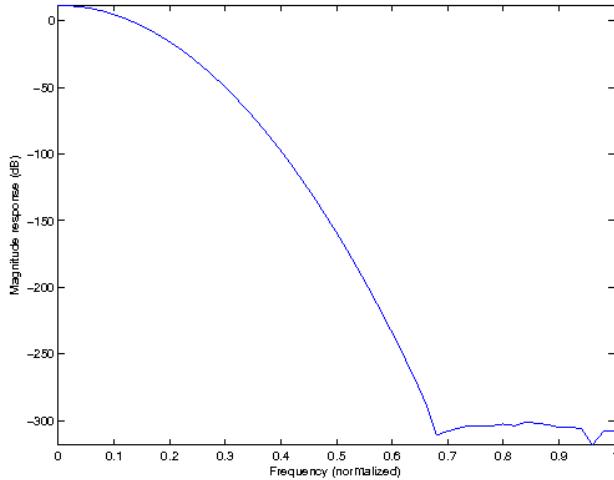
The following will display the magnitude response of a Hann window of length 20 normalized to a peak of 0 dB:

```
magresp({'hann', 20}, '1');
```



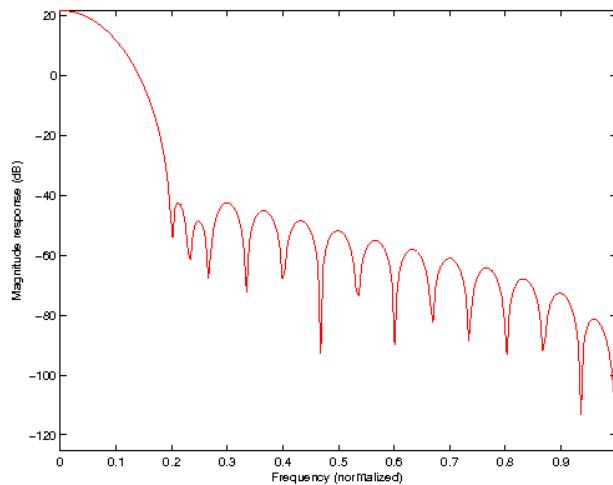
The following will display the magnitude response of a Gaussian window of length 100:

```
magresp('gauss','L',100)
```



The following passes additional options to the plot command to draw in red:

```
magresp({'nuttall11',30}, 'opts', {'r'});
```



### 3.6.6 TRANSFERFUNCTION - The transferfunction of a filter

#### Usage

```
H=transferfunction(g,L);
```

#### Description

`transferfunction(g,L)` computes the transferfunction of length  $L$  of the filter defined by  $g$ .

### 3.6.7 PGRPDELAY - Group delay of a filter with periodic boundaries

#### Usage

```
ggd = pgrpdelay(g,L);
```

#### Description

`pgrpdelay(g,L)` computes group delay of filter  $g$  as a negative derivative of the phase frequency response of filter  $g$  assuming periodic (cyclic) boundaries i.e. the delay may be a negative number. The derivative is calculated using the second order centered difference approximation. The resulting group delay is in samples.

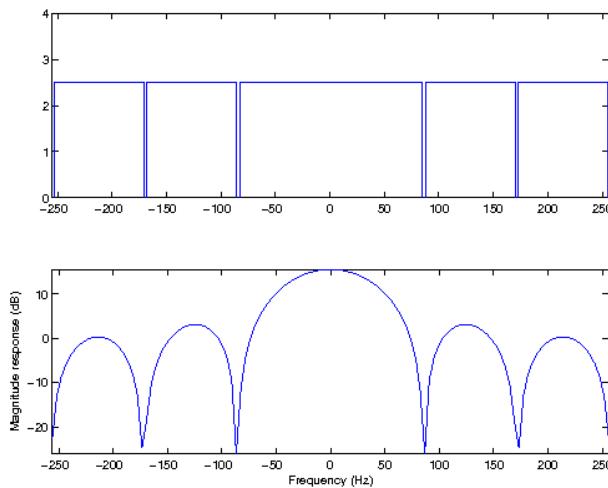
#### Example:

The following example shows a group delay of causal, moving average 6tap FIR filter and it's magnitude frequency response for comparison. The dips in the group delay correspond to places where modulus of the frequency response falls to zero.:

```

g = struct(struct('h',ones(6,1),'offset',0));
L = 512;
figure(1);
subplot(2,1,1);
plot(-L/2+1:L/2,fftshift(pgrpdelay(g,512)));
axis tight; ylim([0,4]);

subplot(2,1,2);
magresp(g,L,'nf');
```



## 3.7 Hermite functions and fractional Fourier transforms

### 3.7.1 PHERM - Periodized Hermite function

#### Usage

```
g=pherm(L,order);
g=pherm(L,order,tfr);
[g,D]=phermd(...);
```

#### Input parameters

<b>L</b>	Length of vector.
<b>order</b>	Order of Hermite function.
<b>tfr</b>	ratio between time and frequency support.

#### Output parameters

<b>g</b>	The periodized Hermite function
----------	---------------------------------

#### Description

`phermd(L,order,tfr)` computes samples of a periodized Hermite function of order *order*. *order* is counted from 0, so the zero'th order Hermite function is the Gaussian.

The parameter *tfr* determines the ratio between the effective support of *g* and the effective support of the DFT of *g*. If *tfr* > 1 then *g* has a wider support than the DFT of *g*.

`phermd(L,order)` does the same setting *tfr* = 1.

If *order* is a vector, `phermd` will return a matrix, where each column is a Hermite function with the corresponding *order*.

`[g,D]=phermd(...)` also returns the eigenvalues *D* of the Discrete Fourier Transform corresponding to the Hermite functions.

The returned functions are eigenvectors of the DFT. The Hermite functions are orthogonal to all other Hermite functions with a different eigenvalue, but eigenvectors with the same eigenvalue are not orthogonal (but see the flags below).

`phermd` takes the following flags at the end of the line of input arguments:

<b>'accurate'</b>	Use a numerically very accurate that computes each Hermite function individually. This is the default.
-------------------	--

<b>'fast'</b>	Use a less accurate algorithm that calculates all the Hermite up to a given order at once.
<b>'noorth'</b>	No orthonormalization of the Hermite functions. This is the default.
<b>'polar'</b>	Orthonormalization of the Hermite functions using the polar decomposition orthonormalization method.
<b>'qr'</b>	Orthonormalization of the Hermite functions using the Gram-Schmidt orthonormalization method (using qr).

If you just need to compute a single Hermite function, there is no speed difference between the 'accurate' and 'fast' algorithm.

### Examples:

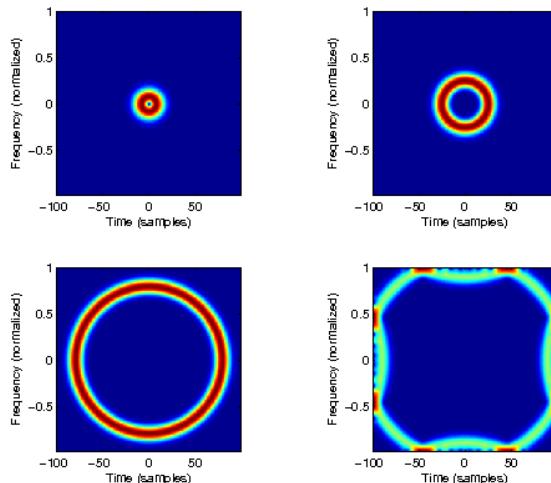
The following plot shows the spectrograms of 4 Hermite functions of length 200 with order 1, 10, 100, and 190:

```
subplot(2,2,1);
sgram(pher(200,1),'nf','tc','lin','nocolorbar'); axis('square');

subplot(2,2,2);
sgram(pher(200,10),'nf','tc','lin','nocolorbar'); axis('square');

subplot(2,2,3);
sgram(pher(200,100),'nf','tc','lin','nocolorbar'); axis('square');

subplot(2,2,4);
sgram(pher(200,190),'nf','tc','lin','nocolorbar'); axis('square');
```



### 3.7.2 HERMBASIS - Orthonormal basis of discrete Hermite functions

#### Usage

```
V=hermbasis(L,p);
V=hermbasis(L);
[V,D]=hermbasis(...);
```

### Description

`hermbasis(L,p)` computes an orthonormal basis of discrete Hermite functions of length  $L$ . The vectors are returned as columns in the output.  $p$  is the order of approximation used to construct the position and difference operator.

All the vectors in the output are eigenvectors of the discrete Fourier transform, and resemble samplings of the continuous Hermite functions to some degree (for low orders).

`[V,D]=hermbasis(...)` also returns the eigenvalues  $D$  of the Discrete Fourier Transform corresponding to the Hermite functions.

### Examples:

The following plot shows the spectrograms of 4 Hermite functions of length 200 with order 1, 10, 100, and 190:

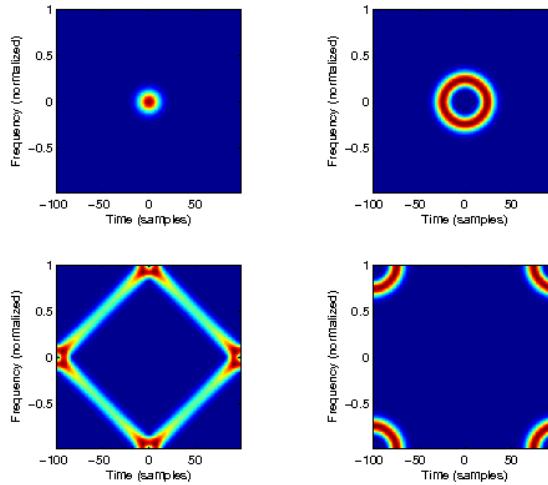
```
H=hermbasis(200);

subplot(2,2,1);
sgram(H(:,1),'nf','tc','lin','nocolorbar'); axis('square');

subplot(2,2,2);
sgram(H(:,10),'nf','tc','lin','nocolorbar'); axis('square');

subplot(2,2,3);
sgram(H(:,100),'nf','tc','lin','nocolorbar'); axis('square');

subplot(2,2,4);
sgram(H(:,190),'nf','tc','lin','nocolorbar'); axis('square');
```



**References:** [63, 17]

### 3.7.3 DFRACFT - Discrete Fractional Fourier transform

#### Usage

```
V=dfracft(f,a,p);
V=dfracft(f,a);
```

#### Description

`dfracft(f,a)` computes the discrete fractional Fourier Transform of the signal  $f$  to the power  $a$ . For  $a=1$  it corresponds to the ordinary discrete Fourier Transform. If  $f$  is multi-dimensional, the transformation

is applied along the first non-singleton dimension.

`dfracft(f, a, dim)` does the same along dimension `dim`.

`dfracft(f, a, [], p)` or `dfracft(f, a, dim, p)` allows to choose the order of approximation of the second difference operator (default:  $p=2$ ).

**References:** [63, 17]

### 3.7.4 FFRACFT - Approximate fast fractional Fourier transform

#### Usage

```
frf=ffracft(f, a)
frf=ffracft(f, a, dim)
```

#### Description

`ffracft(f, a)` computes an approximation of the fractional Fourier transform of the signal  $f$  to the power  $a$ . If  $f$  is multi-dimensional, the transformation is applied along the first non-singleton dimension.

`ffracft(f, a, dim)` does the same along dimension `dim`.

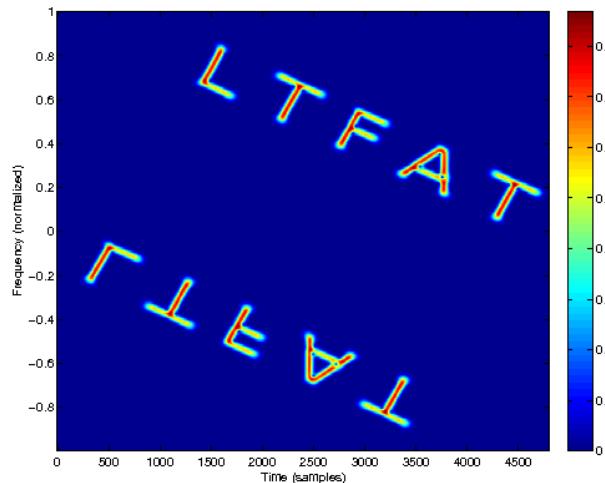
`ffracft` takes the following flags at the end of the line of input arguments:

<b>'origin'</b>	Rotate around the origin of the signal. This is the same action as the <code>dft</code> , but the signal will split in the middle, which may not be the correct action for data signals. This is the default.
<b>'middle'</b>	Rotate around the middle of the signal. This will not break the signal in the middle, but the <code>dft</code> cannot be obtained in this way.

#### Examples:

The following example shows a rotation of the `ltfatlogo` test signal:

```
sgram(ffracft(ltfatlogo,.3,'middle'),'lin','nf');
```



**References:** [17]

## 3.8 Approximation of continuous functions

### 3.8.1 FFTRESAMPLE - Resample signal using Fourier interpolation

#### Usage

```
h=fftresample(f,L);
h=fftresample(f,L,dim);
```

**Description**

`fftresample(f, L)` returns a Fourier interpolation of the signal  $f$  to length  $L$ . If the function is applied to a matrix, it will apply to each column.

`fftresample(f, L, dim)` does the same along dimension  $dim$ .

If the input signal is **not** a periodic signal (or close to), the `dctresample` method gives much better results at the endpoints.

**3.8.2 DCTRESAMPLE - Resample signal using Fourier interpolation****Usage**

```
h=dctresample(f,L);
h=dctresample(f,L,dim);
```

**Description**

`dctresample(f, L)` returns a discrete cosine interpolation of the signal  $f$  to length  $L$ . If the function is applied to a matrix, it will apply to each column.

`dctresample(f, L, dim)` does the same along dimension  $dim$ .

If the input signal is not a periodic signal (or close to), this method will give much better results than `fftresample` at the endpoints, as this method assumes that the signal is even at the endpoints.

The algorithm uses a DCT type iii.

**3.8.3 PDERIV - Derivative of smooth periodic function****Usage**

```
fd=pderiv(f);
fd=pderiv(f,dim);
fd=pderiv(f,dim,difforder);
```

**Description**

`pderiv(f)` will compute the derivative of  $f$  using a 4th order centered finite difference scheme.  $f$  must have been obtained by a regular sampling. If  $f$  is a matrix, the derivative along the columns will be found.

`pderiv(f, dim)` will do the same along dimension  $dim$ .

`pderiv(f, dim, difforder)` uses a centered finite difference scheme of order  $difforder$  instead of the default.

`pderiv(f, dim, Inf)` will compute the spectral derivative using a DFT.

`pderiv` assumes that  $f$  is a regular sampling of a function on the torus  $[0, 1]$ . The derivative of a function on a general torus  $[0, T]$  can be found by scaling the output by  $1/T$ .

**3.9 Cosine and Sine transforms.****3.9.1 DCTI - Discrete Cosine Transform type I****Usage**

```
c=dcti(f);
c=dcti(f,L);
c=dcti(f,[],dim);
c=dcti(f,L,dim);
```

### Description

`dcti(f)` computes the discrete cosine transform of type I of the input signal  $f$ . If  $f$  is a matrix then the transformation is applied to each column. For N-D arrays, the transformation is applied to the first non-singleton dimension.

`dcti(f, L)` zero-pads or truncates  $f$  to length  $L$  before doing the transformation.

`dcti(f, [ ], dim)` or `dcti(f, L, dim)` applies the transformation along dimension  $dim$ .

The transform is real (output is real if input is real) and it is orthonormal.

This transform is its own inverse.

Let  $f$  be a signal of length  $L$ , let  $c = \text{dcti}(f)$  and define the vector  $w$  of length  $L$  by

$$w(n) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } n = 0 \text{ or } n = L - 1 \\ 1 & \text{otherwise} \end{cases}$$

Then

$$c(n+1) = \sqrt{\frac{2}{L-1}} \sum_{m=0}^{L-1} w(n) w(m) f(m+1) \cos\left(\frac{\pi nm}{L-1}\right)$$

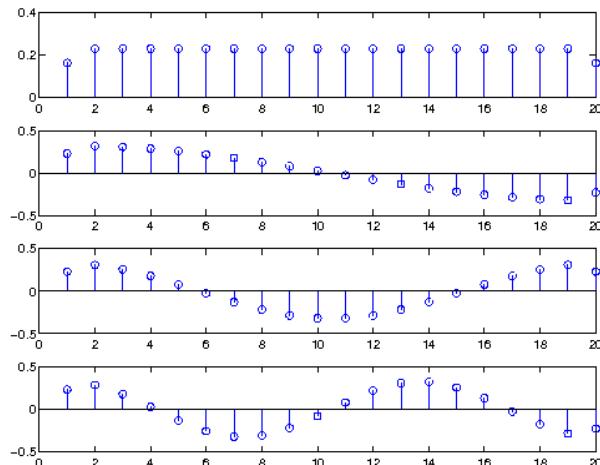
The implementation of this function uses a simple algorithm that require an FFT of length  $2L-2$ , which might potentially be the product of a large prime number. This may cause the function to sometimes execute slowly. If guaranteed high speed is a concern, please consider using one of the other DCT transforms.

### Examples:

The following figures show the first 4 basis functions of the DCTI of length 20:

```
% The dcti is its own adjoint.
F=dcti(eye(20));

for ii=1:4
    subplot(4,1,ii);
    stem(F(:,ii));
end;
```



**References:** [70], [88]

### 3.9.2 DCTII - Discrete Cosine Transform type II

#### Usage

```
c=dctii(f);
c=dctii(f,L);
c=dctii(f,[],dim);
c=dctii(f,L,dim);
```

#### Description

`dctii(f)` computes the discrete cosine transform of type II of the input signal  $f$ . If  $f$  is multi-dimensional, the transformation is applied along the first non-singleton dimension.

`dctii(f, L)` zero-pads or truncates  $f$  to length  $L$  before doing the transformation.

`dctii(f, [], dim)` or `dctii(f, L, dim)` applies the transformation along dimension  $dim$ .

The transform is real (output is real if input is real) and orthonormal.

This is the inverse of `dctiii`.

Let  $f$  be a signal of length  $L$ , let `c=dctii(f)` and define the vector  $w$  of length  $L$  by

$$w(n) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } n = 0 \\ 1 & \text{otherwise} \end{cases}$$

Then

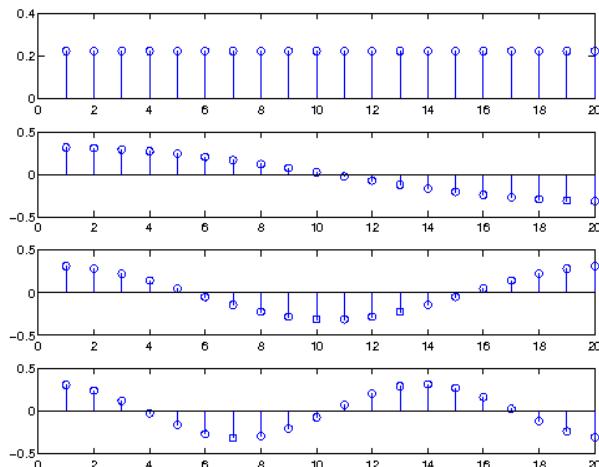
$$c(n+1) = \sqrt{\frac{2}{L}} \sum_{m=0}^{L-1} w(m) f(m+1) \cos\left(\frac{\pi}{L}n\left(m+\frac{1}{2}\right)\right)$$

#### Examples:

The following figures show the first 4 basis functions of the DCTII of length 20:

```
% The dctiii is the adjoint of dctii.
F=dctiii(eye(20));

for ii=1:4
    subplot(4,1,ii);
    stem(F(:,ii));
end;
```



**References:** [70], [88]

### 3.9.3 DCTIII - Discrete Cosine Transform type III

#### Usage

```
c=dctiii(f);
c=dctiii(f,L);
c=dctiii(f,[],dim);
c=dctiii(f,L,dim);
```

#### Description

`dctiii(f)` computes the discrete cosine transform of type III of the input signal  $f$ . If  $f$  is multi-dimensional, the transformation is applied along the first non-singleton dimension.

`dctiii(f,L)` zero-pads or truncates  $f$  to length  $L$  before doing the transformation.

`dctiii(f,[],dim)` or `dctiii(f,L,dim)` applies the transformation along dimension  $dim$ .

The transform is real (output is real if input is real) and orthonormal.

This is the inverse of `dctii`.

Let  $f$  be a signal of length  $L$ , let  $c = \text{dctiii}(f)$  and define the vector  $w$  of length  $L$  by

$$w(n) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } n = 0 \\ 1 & \text{otherwise} \end{cases}$$

Then

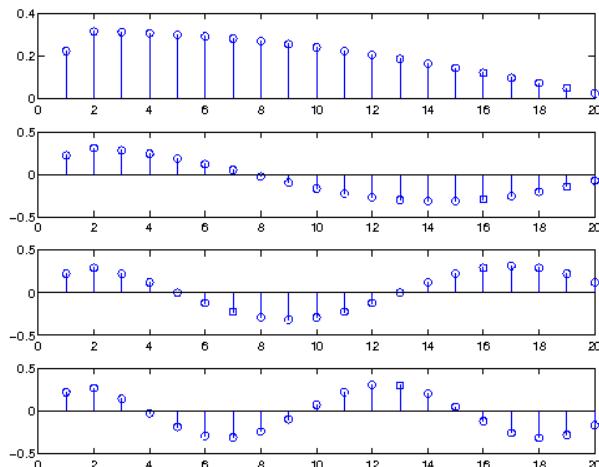
$$c(n+1) = \sqrt{\frac{2}{L}} \sum_{m=0}^{L-1} w(m) f(m+1) \cos\left(\frac{\pi}{L}\left(n+\frac{1}{2}\right)m\right)$$

#### Examples:

The following figures show the first 4 basis functions of the DCTIII of length 20:

```
% The dctii is the adjoint of dctiii.
F=dctii(eye(20));

for ii=1:4
    subplot(4,1,ii);
    stem(F(:,ii));
end;
```



**References:** [70], [88]

### 3.9.4 DCTIV - Discrete Cosine Transform type IV

#### Usage

```
c=dctiv(f);
```

#### Description

`dctiv(f)` computes the discrete cosine transform of type IV of the input signal  $f$ . If  $f$  is multi-dimensional, the transformation is applied along the first non-singleton dimension.

`dctiv(f, L)` zero-pads or truncates  $f$  to length  $L$  before doing the transformation.

`dctiv(f, [], dim)` or `dctiv(f, L, dim)` applies the transformation along dimension  $dim$ .

The transform is real (output is real if input is real) and orthonormal. It is its own inverse.

Let  $f$  be a signal of length  $L$  and let `c=dctiv(f)`. Then

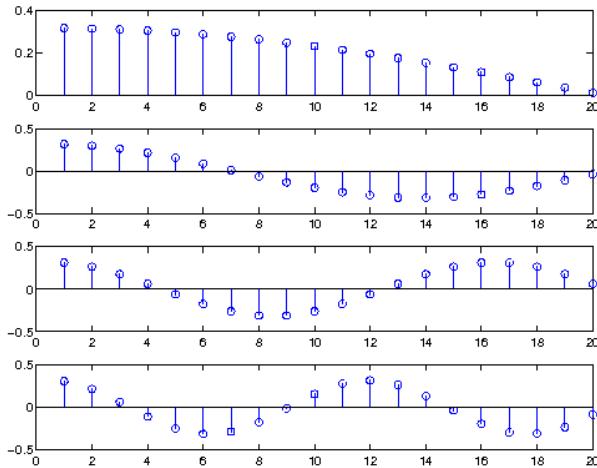
$$c(n+1) = \sqrt{\frac{2}{L}} \sum_{m=0}^{L-1} f(m+1) \cos\left(\frac{\pi}{L}\left(n+\frac{1}{2}\right)\left(m+\frac{1}{2}\right)\right)$$

#### Examples:

The following figures show the first 4 basis functions of the DCTIV of length 20:

```
% The dctiv is its own adjoint.
F=dctiv(eye(20));

for ii=1:4
    subplot(4,1,ii);
    stem(F(:,ii));
end;
```



**References:** [70], [88]

### 3.9.5 DSTI - Discrete Sine Transform type I

#### Usage

```
c=dsti(f);
c=dsti(f,L);
c=dsti(f,[],dim);
c=dsti(f,L,dim);
```

### Description

`dsti(f)` computes the discrete sine transform of type I of the input signal  $f$ . If  $f$  is multi-dimensional, the transformation is applied along the first non-singleton dimension.

`dsti(f, L)` zero-pads or truncates  $f$  to length  $L$  before doing the transformation.

`dsti(f, [], dim)` or `dsti(f, L, dim)` applies the transformation along dimension  $dim$ .

The transform is real (output is real if input is real) and orthonormal.

This transform is its own inverse.

Let  $f$  be a signal of length  $L$  and let  $c = dsti(f)$ . Then

$$c(n+1) = \sqrt{\frac{2}{L+1}} \sum_{m=0}^{L-1} f(m+1) \sin\left(\frac{\pi(n+1)(m+1)}{L+1}\right)$$

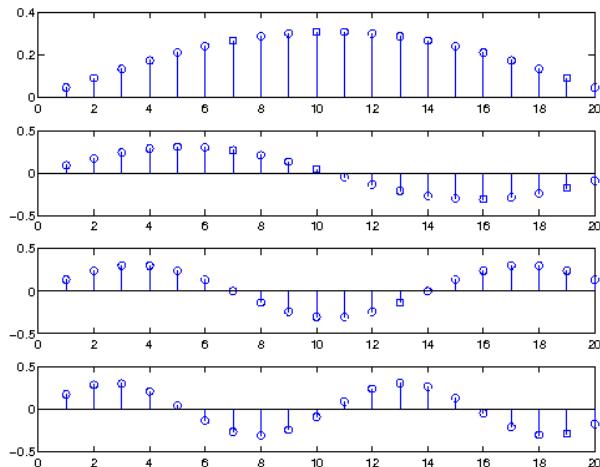
The implementation of this function uses a simple algorithm that requires an FFT of length  $2N + 2$ , which might potentially be the product of a large prime number. This may cause the function to sometimes execute slowly. If guaranteed high speed is a concern, please consider using one of the other DST transforms.

### Examples:

The following figures show the first 4 basis functions of the DSTI of length 20:

```
% The dsti is its own adjoint.
F=dsti(eye(20));

for ii=1:4
    subplot(4,1,ii);
    stem(F(:,ii));
end;
```



**References:** [70], [88]

### 3.9.6 DSTII - Discrete Sine Transform type II

#### Usage

```
c=dstii(f);
c=dstii(f,L);
c=dstii(f,[],dim);
c=dstii(f,L,dim);
```

### Description

`dstii(f)` computes the discrete sine transform of type II of the input signal  $f$ . If  $f$  is multi-dimensional, the transformation is applied along the first non-singleton dimension.

`dstii(f, L)` zero-pads or truncates  $f$  to length  $L$  before doing the transformation.

`dstii(f, [], dim)` or `dstii(f, L, dim)` applies the transformation along dimension  $dim$ .

The transform is real (output is real if input is real) and orthonormal.

The inverse transform of `dstii` is `dstiii`.

Let  $f$  be a signal of length  $L$ , let  $c = \text{dstii}(f)$  and define the vector  $w$  of length  $L$  by

$$w(n) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } n = L - 1 \\ 1 & \text{otherwise} \end{cases}$$

Then

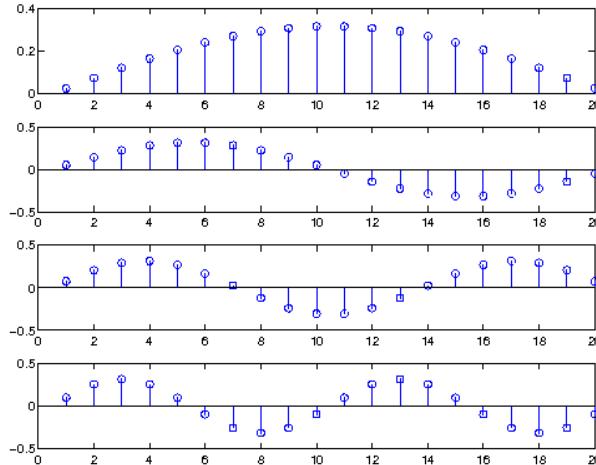
$$c(n+1) = \sqrt{\frac{2}{L}} \sum_{m=0}^{L-1} w(m) f(m+1) \sin\left(\frac{\pi}{L} m \left(n + \frac{1}{2}\right)\right)$$

### Examples:

The following figures show the first 4 basis functions of the DSTII of length 20:

```
% The dstiii is the adjoint of dstii.
F=dstiii(eye(20));

for ii=1:4
    subplot(4,1,ii);
    stem(F(:,ii));
end;
```



**References:** [70], [88]

### 3.9.7 DSTIII - Discrete sine transform type III

#### Usage

```
c=dstiii(f);
c=dstiii(f,L);
c=dstiii(f,[],dim);
c=dstiii(f,L,dim);
```

### Description

`dstiii(f)` computes the discrete sine transform of type III of the input signal  $f$ . If  $f$  is multi-dimensional, the transformation is applied along the first non-singleton dimension.

`dstiii(f,L)` zero-pads or truncates  $f$  to length  $L$  before doing the transformation.

`dstiii(f,[],dim)` or `dstiii(f,L,dim)` applies the transformation along dimension  $dim$ .

The transform is real (output is real if input is real) and orthonormal.

This is the inverse of `dstii`.

Let  $f$  be a signal of length  $L$ , let  $c=dstiii(f)$  and define the vector  $w$  of length  $L$  by

$$w(n) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } n = L - 1 \\ 1 & \text{otherwise} \end{cases}$$

Then

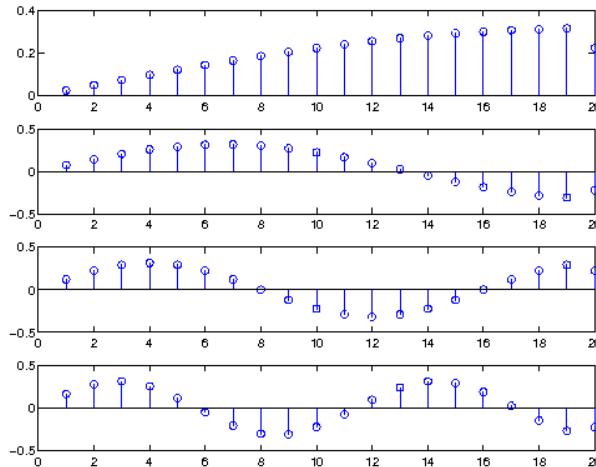
$$c(n+1) = \sqrt{\frac{2}{L}} \sum_{m=0}^{L-1} w(m) f(m+1) \sin\left(\frac{\pi}{L}\left(n+\frac{1}{2}\right)m\right)$$

### Examples:

The following figures show the first 4 basis functions of the DSTIII of length 20:

```
% The dstii is the adjoint of dstiii.
F=dstii(eye(20));

for ii=1:4
    subplot(4,1,ii);
    stem(F(:,ii));
end;
```



**References:** [70], [88]

### 3.9.8 DSTIV - Discrete Sine Transform type IV

#### Usage

```
c=dstiv(f);
c=dstiv(f,L);
c=dstiv(f,[],dim);
c=dstiv(f,L,dim);
```

### Description

`dstiv(f)` computes the discrete sine transform of type IV of the input signal  $f$ . If  $f$  is a matrix, then the transformation is applied to each column. For N-D arrays, the transformation is applied to the first non-singleton dimension.

`dstiv(f, L)` zero-pads or truncates  $f$  to length  $L$  before doing the transformation.

`dstiv(f, [], dim)` applies the transformation along dimension  $dim$ . `dstiv(f, L, dim)` does the same, but pads or truncates to length  $L$ .

The transform is real (output is real if input is real) and it is orthonormal. It is its own inverse.

Let  $f$  be a signal of length  $L$  and let  $c = \text{dstiv}(f)$ . Then

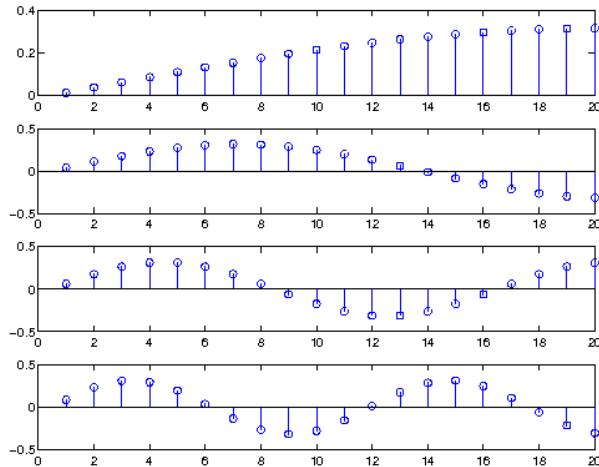
$$c(n+1) = \sqrt{\frac{2}{L}} \sum_{m=0}^{L-1} f(m+1) \sin\left(\frac{\pi}{L}\left(n + \frac{1}{2}\right)\left(m + \frac{1}{2}\right)\right)$$

### Examples:

The following figures show the first 4 basis functions of the DSTIV of length 20:

```
% The dstiv is its own adjoint.
F=dstiv(eye(20));

for ii=1:4
    subplot(4,1,ii);
    stem(F(:,ii));
end;
```



**References:** [70], [88]

# Chapter 4

## LTFAT - Wavelets

### 4.1 Basic analysis/synthesis

#### 4.1.1 FWT - Fast Wavelet Transform

##### Usage

```
c = fwt(f,w,J);  
c = fwt(f,w,J,dim);  
[c,info] = fwt(...);
```

##### Input parameters

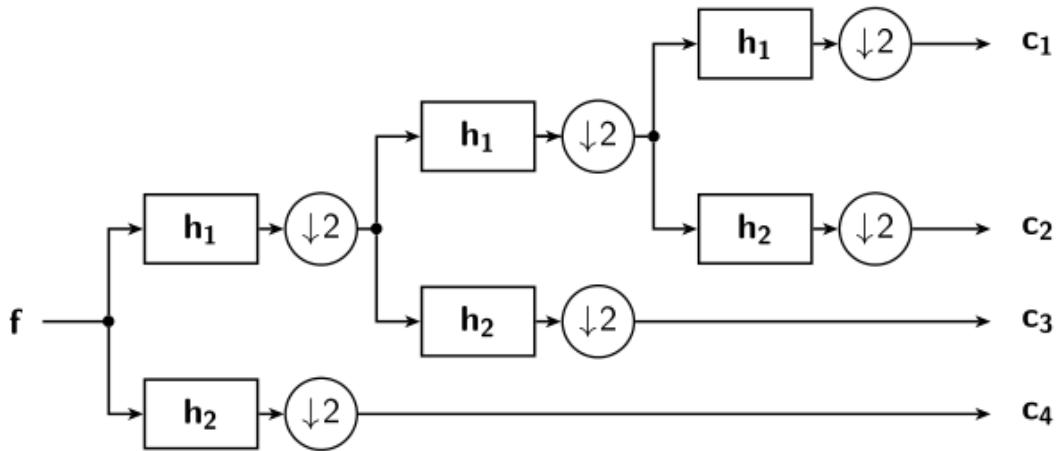
<b>f</b>	Input data.
<b>w</b>	Wavelet definition.
<b>J</b>	Number of filterbank iterations.
<b>dim</b>	Dimension to along which to apply the transform.

##### Output parameters

<b>c</b>	Coefficient vector.
<b>info</b>	Transform parameters struct.

##### Description

`fwt(f,w,J)` returns discrete wavelet coefficients of the input signal  $f$  using  $J$  iterations of the basic wavelet filterbank defined by  $w$  using the fast wavelet transform algorithm (Mallat's algorithm). The coefficients are the Discrete Wavelet transform (DWT) of the input signal  $f$ , if  $w$  defines two-channel wavelet filterbank. The following figure shows DWT with  $J=3$ .



The function can apply the Mallat's algorithm using basic filterbanks with any number of the channels. In such case, the transform have a different name.

Several formats of the basic filterbank definition  $w$  are recognized. One of them is a text string formed by a concatenation of a function name with the `wfilt_` prefix followed by a list of numerical arguments delimited by `:`. For example '`db10`' will result in a call to `wfilt_db(10)` or '`spline4:4`' in call to `wfilt_spline(4, 4)` etc. All filter defining functions can be listed by running `dir([ltfatbasepath, filesep, 'wavelets'])`. Please see help of the respective functions and follow references therein.

For other recognized formats of  $w$  please see `fwtinit`.

`[c, info] = fwt(f, w, J)` additionally returns struct. `info` containing transform parameters. It can be conveniently used for the inverse transform `ifwt` e.g. as `fhat = ifwt(c, info)`. It is also required by the `plotwavelets` function.

If  $f$  is row/column vector, the subbands  $c$  are stored in a single row/column in a consecutive order with respect to the increasing central frequency. The lengths of subbands are stored in `info.Lc` so the subbands can be easily extracted using `wavpack2cell`. Moreover, one can pass an additional flag '`cell`' to obtain the coefficient directly in a cell array. The cell array can be again converted to a packed format using `wavcell2pack`.

If the input  $f$  is a matrix, the transform is applied to each column if `dim==1` (default) and `[Ls, W]=size(f)`. If `dim==2` the transform is applied to each row `[W, Ls]=size(f)`. The output is then a matrix and the input orientation is preserved in the orientation of the output coefficients. The `dim` parameter has to be passed to the `wavpack2cell` and `wavcell2pack` when used.

### Boundary handling:

`fwt(f, w, J, 'per')` (default) uses the periodic extension which considers the input signal as it was a one period of some infinite periodic signal as is natural for transforms based on the FFT. The resulting wavelet representation is non-expansive, that is if the input signal length is a multiple of a  $J$ -th power of the subsampling factor and the filterbank is critically subsampled, the total number of coefficients is equal to the input signal length. The input signal is padded with zeros to the next legal length  $L$  internally.

The default periodic extension can result in "false" high wavelet coefficients near the boundaries due to the possible discontinuity introduced by the zero padding and periodic boundary treatment.

`fwt(f, w, J, ext)` with `ext` other than '`per`' computes a slightly redundant wavelet representation of the input signal  $f$  with the chosen boundary extension `ext`. The redundancy (expansivity) of the representation is the price to pay for using general filterbank and custom boundary treatment. The extensions are done at each level of the transform internally rather than doing the prior explicit padding.

The supported possibilities are:

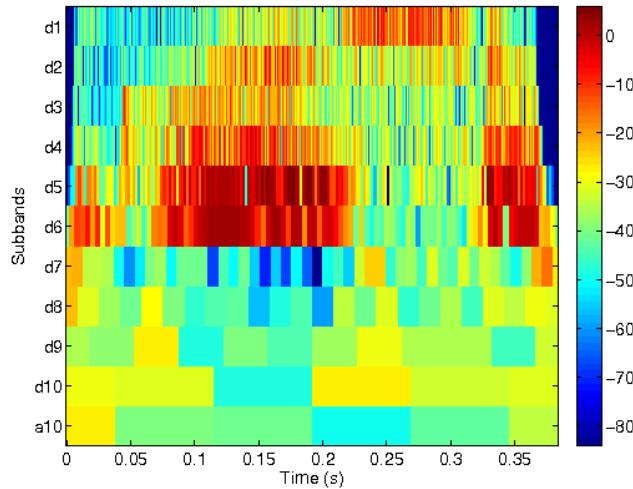
<code>'zero'</code>	Zeros are considered outside of the signal (coefficient) support.
<code>'even'</code>	Even symmetric extension.
<code>'odd'</code>	Odd symmetric extension.

Note that the same flag has to be used in the call of the inverse transform function ifwt if the info struct is not used.

### Examples:

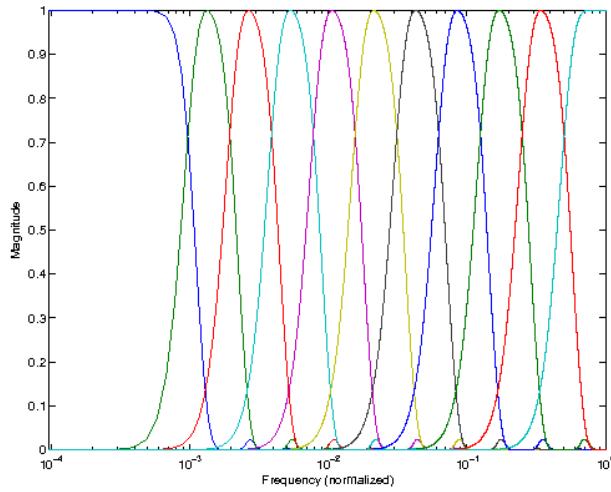
A simple example of calling the fwt function using 'db8' wavelet filters.:

```
[f, fs] = greasy;
J = 10;
[c, info] = fwt(f, 'db8', J);
plotwavelets(c, info, fs, 'dynrange', 90);
```



Frequency bands of the transform with x-axis in a log scale and band peaks normalized to 1. Only positive frequency band is shown.

```
[g, a] = wfbt2filterbank({'db8', 10, 'dwt'});
filterbankfreqz(g, a, 20*1024, 'linabs', 'posfreq', 'plot', 'inf', 'flog');
```



**References:** [54]

### 4.1.2 IFWT - Inverse Fast Wavelet Transform

#### Usage

```
f = ifwt(c, info)
```

```
f = ifwt(c,w,J,Ls)
f = ifwt(c,w,J,Ls,dim)
```

### Input parameters

<b>c</b>	Wavelet coefficients.
<b>info, w</b>	Transform parameters struct/Wavelet filters definition.
<b>J</b>	Number of filterbank iterations.
<b>Ls</b>	Length of the reconstructed signal.
<b>dim</b>	Dimension to along which to apply the transform.

### Output parameters

<b>f</b>	Reconstructed data.
----------	---------------------

### Description

`f = ifwt(c, info)` reconstructs signal  $f$  from the wavelet coefficients  $c$  using parameters from `info` struct. both returned by `fwt` function.

`f = ifwt(c, w, J, Ls)` reconstructs signal  $f$  from the wavelet coefficients  $c$  using  $J$ -iteration synthesis filterbank build from the basic filterbank defined by  $w$ . The  $Ls$  parameter is mandatory due to the ambiguity of lengths introduced by the subsampling operation and by boundary treatment methods. Note that the same flag as in the `fwt` function have to be used, otherwise perfect reconstruction cannot be obtained.

In both cases, the fast wavelet transform algorithm (Mallat's algorithm) is employed. The format of  $c$  can be either packed, as returned by the `fwt` function or cell-array as returned by `wavpack2cell` function.

Please see the help on `fwt` for a detailed description of the parameters.

### Examples:

A simple example showing perfect reconstruction:

```
f = gspi;
J = 8;
c = fwt(f,'db8',J);
fhat = ifwt(c,'db8',J,length(f));
% The following should give (almost) zero
norm(f-fhat)
```

*This code produces the following output:*

```
ans =
3.5680e-14
```

**References:** [54]

### 4.1.3 FWT2 - Fast Wavelet Transform 2D

#### Usage

```
c = fwt2(f,w,J);
c = fwt2(f,w,J,...);
```

**Input parameters**

<b>f</b>	Input data.
<b>w</b>	Wavelet Filterbank definition.
<b>J</b>	Number of filterbank iterations.

**Output parameters**

<b>c</b>	Coefficients stored in a matrix.
----------	----------------------------------

**Description**

`c=fwt2(f,w,J)` returns wavelet coefficients  $c$  of the input matrix  $f$  using  $J$  iterations of the basic wavelet filterbank defined by  $w$ . Please see fwt for description of  $w$  and  $J$ .

`fwt2` supports just the non-expansive boundary condition 'per' and critically subsampled filterbanks in order to be able to pack the coefficients in a matrix. Also the  $J$  is limited to some maximum value for the same reason.

Additional flags make it possible to specify how the algorithm should subdivide the matrix:

'standard' Standard behaviour of the JPEG 2000 standard. This is the default.

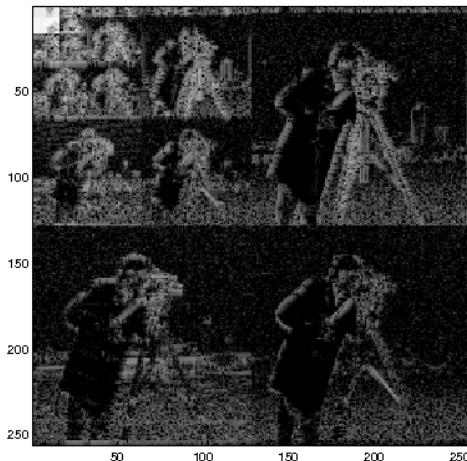
'tensor' This corresponds to doing a full fwt along each dimension of the matrix.

**Examples:**

Some simple example of calling the `fwt2` function, compare with the cameraman image. Only the 70 dB largest coefficients are shown, to make the structures more visible.

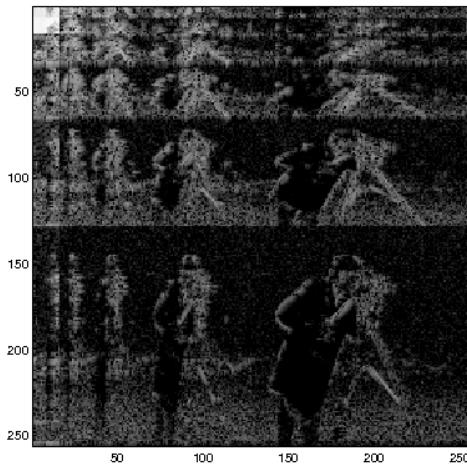
The first example uses the standard layout:

```
c = fwt2(cameraman,'db8',4);
imagesc(dynlimit(20*log10(abs(c)),70));
axis('image'); colormap(gray);
```



The second example uses the tensor product layout:

```
c = fwt2(cameraman,'db8',4,'tensor');
imagesc(dynlimit(20*log10(abs(c)),70));
axis('image'); colormap(gray);
```



**References:** [54]

#### 4.1.4 IFWT2 - Inverse Fast Wavelet Transform

##### Usage

```
f = ifwt2(c, w, J)
f = ifwt2(c, w, J, Ls, ...)
```

##### Input parameters

<b>c</b>	Coefficients stored in a matrix.
<b>w</b>	Wavelet filters definition.
<b>J</b>	Number of filterbank iterations.
<b>Ls</b>	Size of the reconstructed signal.

##### Output parameters

<b>f</b>	Reconstructed data.
----------	---------------------

##### Description

`f = ifwt2(c, w, J)` reconstructs signal  $f$  from the wavelet coefficients  $c$  using a  $J$ -iteration synthesis filterbank build from the basic synthesis filterbank defined by  $w$ .  $f$  is a matrix with `size(f) == size(c)`.

`f = ifwt2(c, w, J, Ls)` works as above but the result  $f$  is cut or extended to size  $Ls$  if  $Ls$  is a two-element vector or to  $[Ls, Ls]$  if  $Ls$  is a scalar.

This function takes the same optional parameters as `fwt2`. Please see the help on `fwt2` for a description of the parameters.

**References:** [54]

#### 4.1.5 UFWT - Undecimated Fast Wavelet Transform

##### Usage

```
c = ufw(f, w, J);
[c, info] = ufw(...);
```

**Input parameters**

<b>f</b>	Input data.
<b>w</b>	Wavelet Filterbank.
<b>J</b>	Number of filterbank iterations.

**Output parameters**

<b>c</b>	Coefficients stored in $L \times J + 1$ matrix.
<b>info</b>	Transform parameters struct.

**Description**

`ufwt (f, w, J)` computes redundant time (or shift) invariant wavelet representation of the input signal  $f$  using wavelet filters defined by  $w$  in the "a-trous" algorithm.

For all accepted formats of the parameter  $w$  see the `fwtinit` function.

`[c, info] = ufw (f, w, J)` additionally returns the `info` struct. containing the transform parameters. It can be conveniently used for the inverse transform `iufwt` e.g. `fhat = iufwt (c, info)`. It is also required by the `plotwavelets` function.

The coefficients  $c$  are so called undecimated Discrete Wavelet transform of the input signal  $f$ , if  $w$  defines two-channel wavelet filterbank. Other names for this version of the wavelet transform are: the time-invariant wavelet transform, the stationary wavelet transform, maximal overlap discrete wavelet transform or even the "continuous" wavelet transform (as the time step is one sample). However, the function accepts any number filters (referred to as  $M$ ) in the basic wavelet filterbank and the number of columns of  $c$  is then  $J(M - 1) + 1$ .

For one-dimensional input  $f$  of length  $L$ , the coefficients  $c$  are stored as columns of a matrix. The columns are ordered with increasing central frequency of the respective subbands.

If the input  $f$  is  $L \times W$  matrix, the transform is applied to each column and the outputs are stacked along third dimension in the  $L \times J(M - 1) + 1 \times W$  data cube.

**Filter scaling**

When compared to `fwt`, `ufwt` subbands are gradually more and more redundant with increasing level of the subband. If no scaling of the filters is introduced, the energy of subbands tends to grow with increasing level. There are 3 flags defining filter scaling:

**'sqrt'** Each filter is scaled by  $1/\sqrt{a}$ , where  $a$  is the hop factor associated with it. If the original filterbank is orthonormal, the overall undecimated transform is a tight frame.  
This is the default.

**'noscale'** Uses filters without scaling.

**'scale'** Each filter is scaled by  $1/a$ .

If '`noscale`' is used, '`scale`' has to be used in `iufwt` (and vice versa) in order to obtain a perfect reconstruction.

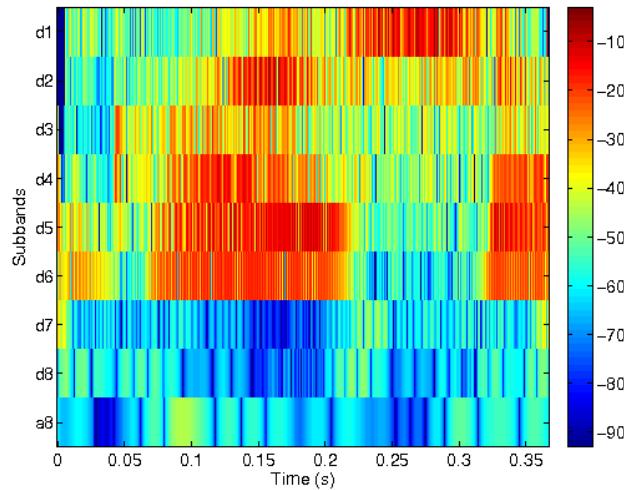
**Boundary handling:**

`c=ufwt (f, w, J)` uses periodic boundary extension. The extensions are done internally at each level of the transform, rather than doing the prior explicit padding.

**Examples:**

A simple example of calling the `ufwt` function:

```
[f, fs] = greasy;
J = 8;
[c, info] = ufwt(f, 'db8', J);
plotwavelets(c, info, fs, 'dynrange', 90);
```



**References:** [40]

#### 4.1.6 IUFWT - Inverse Undecimated Fast Wavelet Transform

##### Usage

```
f = iufwt(c, info)
f = iufwt(c, w, J);
```

##### Input parameters

**c** Coefficients stored in  $L \times J + 1$  matrix.

**info, w** Transform parameters struct/Wavelet filters definition.

**J** Number of filterbank iterations.

##### Output parameters

**f** Reconstructed data.

##### Description

`f = iufwt(c, info)` reconstructs signal  $f$  from the wavelet coefficients  $c$  using parameters from `info` struct. both returned by `ufwt` function.

`f = iufwt(c, w, J)` reconstructs signal  $f$  from the wavelet coefficients  $c$  using the wavelet filterbank consisting of the  $J$  levels of the basic synthesis filterbank defined by  $w$  using the "a-trous" algorithm. Note that the same flag as in the `ufwt` function have to be used.

Please see the help on `ufwt` for a description of the parameters.

### Filter scaling

As in ufw, 3 flags defining scaling of filters are recognized:

- **'sqrt'** Each filter is scaled by  $1/\sqrt{a}$ , where  $a$  is the hop factor associated with it. If the original filterbank is orthonormal, the overall undecimated transform is a tight frame. This is the default.
- **'noscale'** Uses filters without scaling.
- **'scale'** Each filter is scaled by  $1/a$ .

If 'noscale' is used, 'scale' must have been used in ufw (and vice versa) in order to obtain a perfect reconstruction.

### Examples:

A simple example showing perfect reconstruction:

```
f = gspi;
J = 8;
c = ufw(f,'db8',J);
fhat = iufwt(c,'db8',J);
% The following should give (almost) zero
norm(f-fhat)
```

*This code produces the following output:*

```
ans =
6.4884e-14
```

**References:** [54]

### 4.1.7 FWTLENGTH - FWT length from signal

#### Usage

```
L=fwtlength(Ls,w,J);
```

#### Description

`fwtlength(Ls,w,J)` returns the length of a Wavelet system that is long enough to expand a signal of length  $L_s$ . Please see the help on fwt for an explanation of the parameters  $w$  and  $J$ .

If the returned length is longer than the signal length, the signal will be zero-padded by fwt to length  $L$ .

In addition, the function accepts flags defining boundary extension technique as in fwt. The returned length can be longer than the signal length only in case of 'per' (periodic extension).

### 4.1.8 FWTCLENGTH - FWT subbands lengths from a signal length

#### Usage

```
Lc=fwtclength(Ls,w,J);
[Lc,L]=fwtclength(...);
```

#### Description

`Lc=fwtclength(Ls,w,J)` returns the lengths of the wavelet coefficient subbands for a signal of length  $L_s$ . Please see the help on fwt for an explanation of the parameters  $w$  and  $J$ .

`[Lc,L]=fwtclength(...)` additionally the function returns the next legal length of the input signal for the given extension type.

The function support the same boundary-handling flags as the fwt does.

## 4.2 Advanced analysis/synthesis

### 4.2.1 WFBT - Wavelet FilterBank Tree

#### Usage

```
c=wfbt(f,wt);
c=wfbt(f,wt,ext);
[c,info]=wfbt(...);
```

#### Input parameters

<b>f</b>	Input data.
<b>wt</b>	Wavelet filterbank tree definition.

#### Output parameters

<b>c</b>	Coefficients stored in a cell-array.
<b>info</b>	Transform parameters struct.

#### Description

wfbt(f,wt) returns coefficients *c* obtained by applying a wavelet filterbank tree defined by *wt* to the input data *f*.

[*c*,*info*]=wfbt(f,wt) additionally returns struct. *info* containing transform parameters. It can be conveniently used for the inverse transform iwfbt e.g. *fhat* = iwfbt(*c*,*info*). It is also required by the plotwavelets function.

*wt* defines a tree shaped filterbank structure build from the elementary two (or more) channel wavelet filters. The tree can have any shape and thus provide a flexible frequency covering. The outputs of the tree leaves are stored in *c*.

The *wt* parameter can have two formats:

- 1) Cell array containing 3 elements {*w*,*J*,treetype}, where *w* is the basic wavelet filterbank definition as in fwt function, *J* stands for the depth of the tree and the flag treetype defines the type of the tree to be used. Supported options are:

- '**dwt**' Plain DWT tree (default). This gives one band per octave freq. resolution when using 2 channel basic wavelet filterbank and produces coefficients identical to the ones in fwt.
- '**full**' Full filterbank tree. Both (all) basic filterbank outputs are decomposed further up to depth *J* achieving linear frequency band division.
- '**doubleband**', '**quadband**', '**octaband**' The filterbank is designed such that it mimics 4-band, 8-band or 16-band complex wavelet transform provided the basic filterbank is 2 channel. In this case, *J* is treated such that it defines number of levels of 4-band, 8-band or 16-band transform.

- 2) Structure returned by the wfbtinit function and possibly modified by wfbtput and wfbtremove.

Please see wfbtinit for a detailed description and more options.

If *f* is row/column vector, the coefficient vectors *c*{*j*} are columns.

If *f* is a matrix, the transformation is by default applied to each of *W* columns [*Ls*, *W*]=size(*f*). In addition, the following flag groups are supported:

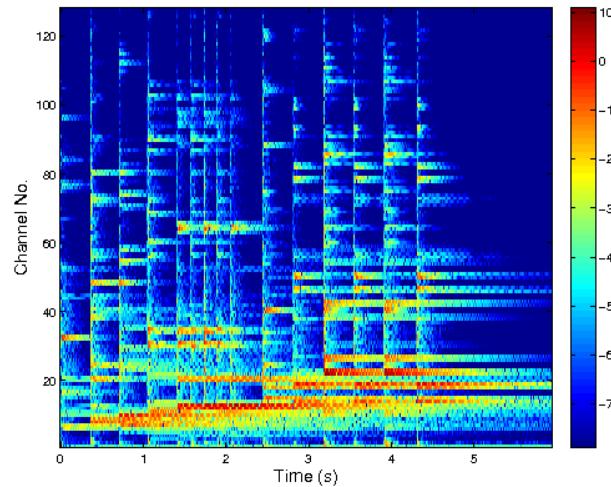
'**per**' '(**default**) , '**zero**' , '**odd**' , '**even**' Type of the boundary handling. Please see the help on fwt for a description of the boundary condition flags.

'**freq**' '(**default**) , '**nat**' Frequency or natural ordering of the coefficient subbands. The direct usage of the wavelet tree ('nat' option) does not produce coefficient subbands ordered according to the frequency. To achieve that, some filter shuffling has to be done ('freq' option).

**Examples:**

A simple example of calling the wfbt function using the "full decomposition" wavelet tree:

```
f = gspi;
J = 7;
[c,info] = wfbt(f,'sym10',J,'full');
plotwavelets(c,info,44100,'dynrange',90);
```



### 4.2.2 IWFBT - Inverse Wavelet Filterbank Tree

**Usage**

```
f=iwfbt(c,info);
f=iwfbt(c,wt,Ls);
```

**Input parameters**

**c** Coefficients stored in a cell-array.

**info, wt** Transform parameters struct/Wavelet Filterbank tree

**Ls** Length of the reconstructed signal.

**Output parameters**

**f** Reconstructed data.

**Description**

`f = iwfbt(c,info)` reconstructs signal  $f$  from the coefficients  $c$  using parameters from `info` struct. both returned by `wfbt` function.

`f = iwfbt(c,wt,Ls)` reconstructs signal  $f$  from the coefficients  $c$  using filterbank tree defined by `wt`. Please see `wfbt` function for possible formats of `wt`. The `Ls` parameter is mandatory due to the ambiguity of reconstruction lengths introduced by the subsampling operation and by boundary treatment methods. Note that the same flag as in the `wfbt` function have to be used, otherwise perfect reconstruction cannot be obtained. Please see help for `wfbt` for description of the flags.

**Examples:**

A simple example showing perfect reconstruction using idtwfb:

```
f = gspi;
J = 7;
wt = {'db6',J};
c = wfbt(f,wt);
fhat = iwfbt(c,wt,length(f));
% The following should give (almost) zero
norm(f-fhat)
```

*This code produces the following output:*

```
ans =
1.3895e-13
```

### 4.2.3 UWFBT - Undecimated Wavelet FilterBank Tree

**Usage**

```
c=uwfbt(f,wt);
[c,info]=uwfbt(...);
```

**Input parameters**

<b>f</b>	Input data.
<b>wt</b>	Wavelet Filterbank tree

**Output parameters**

<b>c</b>	Coefficients stored in $L \times M$ matrix.
----------	---

**Description**

`uwfbt(f,wt)` computes redundant time (or shift) invariant representation of the input signal  $f$  using the filterbank tree definition in  $wt$  and using the "a-trous" algorithm. Number of columns in  $c$  ( $M$ ) is defined by the total number of outputs of nodes of the tree.

`[c,info]=uwfbt(f,wt)` additionally returns struct. `info` containing the transform parameters. It can be conveniently used for the inverse transform `iuwfbt` e.g. `fhat = iuwfbt(c,info)`. It is also required by the `plotwavelets` function.

If  $f$  is a matrix, the transformation is applied to each of  $W$  columns and the coefficients in  $c$  are stacked along the third dimension.

Please see help for `wfbt` description of possible formats of  $wt$  and description of frequency and natural ordering of the coefficient subbands.

**Filter scaling**

When compared to `wfbt`, the subbands produced by `uwfbt` are gradually more and more redundant with increasing depth in the tree. This results in energy grow of the coefficients. There are 3 flags defining filter scaling:

- 'sqrt'** Each filter is scaled by  $1/\sqrt{a}$ , where  $a$  is the hop factor associated with it. If the original filterbank is orthonormal, the overall undecimated transform is a tight frame.

This is the default.

- 'noscale'** Uses filters without scaling.

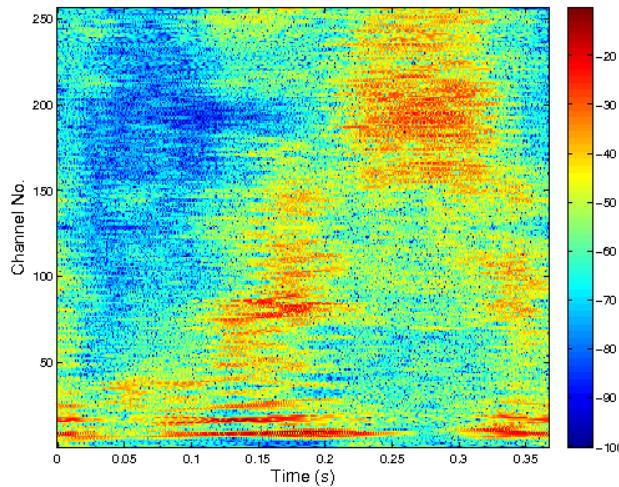
- 'scale'** Each filter is scaled by  $1/a$ .

If '`noscale`' is used, '`scale`' has to be used in `iuwfbt` (and vice versa) in order to obtain a perfect reconstruction.

**Examples:**

A simple example of calling the uwfbt function using the "full decomposition" wavelet tree:

```
f = greasy;
J = 8;
[c,info] = uwfbt(f,'sym10',J,'full');
plotwavelets(c,info,16000,'dynrange',90);
```



#### 4.2.4 IUWFBT - Inverse Undecimated Wavelet Filterbank Tree

**Usage**

```
f = iuwfbt(c,info)
f = iuwfbt(c,wt)
```

**Input parameters**

<b>c</b>	Coefficients stored in $L \times M$ matrix.
<b>info, wt</b>	Transform parameters struct/Wavelet tree definition.

**Output parameters**

<b>f</b>	Reconstructed data.
----------	---------------------

**Description**

`f = iuwfbt(c,info)` reconstructs signal  $f$  from the coefficients  $c$  using parameters from `info` struct, both returned by the uwfbt function.

`f = iuwfbt(c,wt)` reconstructs signal  $f$  from the wavelet coefficients  $c$  using the undecimated wavelet filterbank tree described by `wt`.

Please see help for wfbt description of possible formats of `wt`.

**Filter scaling:**

As in uwfbt, the function recognizes three flags controlling scaling of the filters:

**'sqrt'** Each filter is scaled by  $1/\sqrt{a}$ , where  $a$  is the hop factor associated with it. If the original filterbank is orthonormal, the overall undecimated transform is a tight frame. This is the default.

**'noscale'** Uses filters without scaling.

**'scale'** Each filter is scaled by  $1/a$ .

If '**'noscale'**' is used, '**'scale'**' must have been used in `uwfbt` (and vice versa) in order to obtain a perfect reconstruction.

### Examples:

A simple example showing perfect reconstruction using the "full decomposition" wavelet tree:

```
f = greasy;
J = 6;
c = uwfbt(f, {'db8', J, 'full'});
fhat = iuwfbt(c, {'db8', J, 'full'});
% The following should give (almost) zero
norm(f-fhat)
```

*This code produces the following output:*

```
ans =
2.6500e-14
```

## 4.2.5 WPFBT - Wavelet Packet FilterBank Tree

### Usage

```
c=wpfbt(f,wt);
[c,info]=wpfbt(...);
```

### Input parameters

<b>f</b>	Input data.
<b>wt</b>	Wavelet Filterbank tree definition.

### Output parameters

<b>c</b>	Coefficients stored in a cell-array.
<b>info</b>	Transform parameters struct.

### Description

`c=wpfbt(f,wt)` returns wavelet packet coefficients  $c$  obtained by applying a wavelet filterbank tree defined by `wt` to the input data  $f$ .

`[c,info]=wpfbt(f,wt)` additionally returns struct. `info` containing transform parameters. It can be conveniently used for the inverse transform `iwpfbt` e.g. `fhat = iwpfbt(c,info)`. It is also required by the `plotwavelets` function.

In contrast to `wfbt`, the cell array  $c$  contain every intermediate output of each node in the tree.  $c\{j,j\}$  are ordered according to nodes taken in the breadth-first order.

If  $f$  is row/column vector, the coefficient vectors  $c\{j,j\}$  are columns. If  $f$  is a matrix, the transformation is applied to each of column of the matrix.

### Scaling of intermediate outputs:

The following flags control scaling of intermediate outputs and therefore the energy relations between coefficient subbands. An intermediate output is an output of a node which is further used as an input to a descendant node.

**'intsqrt'** Each intermediate output is scaled by  $1/\sqrt{2}$ . If the filterbank in each node is orthonormal, the overall undecimated transform is a tight frame. This is the default.

**'intnoscale'** No scaling of intermediate results is used. This is necessary for the wpbest function to correctly work with the cost measures.

**'intscale'** Each intermediate output is scaled by  $1/2$ .

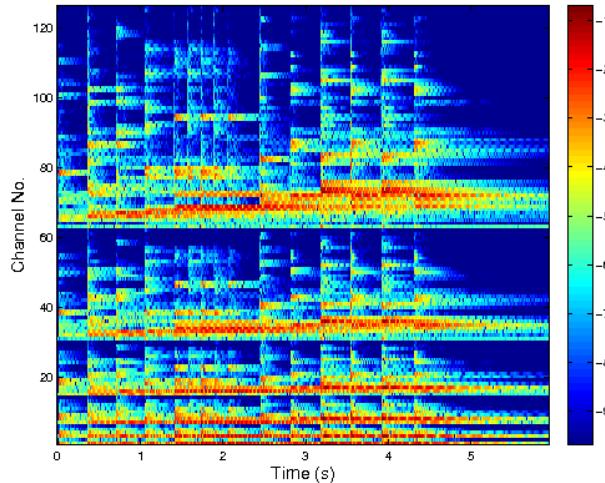
If 'intnoscale' is used, 'intscale' must be used in iwpfbt (and vice versa) in order to obtain a perfect reconstruction.

Please see help for wfbt description of possible formats of *wt* and of the additional flags defining boundary handling.

### Examples:

A simple example of calling the wpfbt function using the "full decomposition" wavelet tree:

```
f = gspi;
J = 6;
[c,info] = wpfbt(f,{'sym10',J,'full'});
plotwavelets(c,info,44100,'dynrange',90);
```



### 4.2.6 IWPFBT - Inverse Wavelet Packet Filterbank Tree

#### Usage

```
f=iwpfbt(c,info);
f=iwpfbt(c,wt,Ls);
```

#### Input parameters

**c** Coefficients stored in a cell-array.

**info, wt** Transform parameters struct/Wavelet Filterbank tree.

**Ls** Length of the reconstructed signal.

### Output parameters

<b>f</b>	Reconstructed data.
----------	---------------------

### Description

`f = iwpfbt(c, info)` reconstructs signal  $f$  from the coefficients  $c$  using parameters from `info` struct, both returned by `wfbt` function.

`f = iwpfbt(c, wt, Ls)` reconstructs signal  $f$  from the coefficients  $c$  using filter bank tree defined by `wt`. Please see `wfbt` function for possible formats of `wt`. The  $Ls$  parameter is mandatory due to the ambiguity of reconstruction lengths introduced by the subsampling operation and by boundary treatment methods.

Please see help for `wfbt` description of possible formats of `wt` and of the additional flags.

### Scaling of intermediate outputs:

The following flags control scaling of the intermediate coefficients. The intermediate coefficients are outputs of nodes which are also inputs to nodes further in the tree.

- **'intsqrt'** Each intermediate output is scaled by  $1/\sqrt{2}$ . If the filterbank in each node is orthonormal, the overall undecimated transform is a tight frame. This is the default.
- **'intnoscale'** No scaling of intermediate results is used.
- **'intscale'** Each intermediate output is scaled by  $1/2$ .

If `'intnoscale'` is used, `'intscale'` must have been used in `wpfbt` (and vice versa) in order to obtain a perfect reconstruction.

### Examples:

A simple example showing perfect reconstruction using the "full decomposition" wavelet tree:

```
f = gspi;
J = 7;
wt = {'db10', J, 'full'};
c = wpfbt(f, wt);
fhat = iwpfbt(c, wt, length(f));
% The following should give (almost) zero
norm(f-fhat)
```

This code produces the following output:

```
ans =
7.9931e-14
```

## 4.2.7 UWPFBT - Undecimated Wavelet Packet FilterBank Tree

### Usage

```
c=uwpfbt(f,wt);
[c,info]=uwpfbt(...);
```

### Input parameters

<b>f</b>	Input data.
<b>wt</b>	Wavelet Filterbank tree

### Output parameters

**c** Coefficients in a  $L \times M$  matrix.

### Description

`c=uwpfbt(f,wt)` returns coefficients  $c$  obtained by applying the undecimated wavelet filterbank tree defined by  $wt$  to the input data  $f$  using the "a-trous" algorithm. Number of columns in  $c$  ( $M$ ) is defined by the total number of outputs of each node. The outputs  $c(:, j)$  are ordered in the breadth-first node order manner.

`[c,info]=uwpfbt(f,wt)` additionally returns struct. `info` containing the transform parameters. It can be conveniently used for the inverse transform `iuwpfbt` e.g. `fhat = iuwpfbt(c,info)`. It is also required by the `plotwavelets` function.

If  $f$  is a matrix, the transformation is applied to each of  $W$  columns and the coefficients in  $c$  are stacked along the third dimension.

Please see help for `wfbt` description of possible formats of  $wt$ .

### Scaling of intermediate outputs:

The following flags control scaling of intermediate outputs and therefore the energy relations between coefficient subbands. An intermediate output is an output of a node which is further used as an input to a descendant node.

**'intsqrt'** Each intermediate output is scaled by  $1/\sqrt{2}$ . If the filterbank in each node is orthonormal, the overall undecimated transform is a tight frame. This is the default.

**'intnoscale'** No scaling of intermediate results is used. This is necessary for the `wpbest` function to correctly work with the cost measures.

**'intscale'** Each intermediate output is scaled by  $1/2$ .

If '`intnoscale`' is used, '`intscale`' must be used in `iuwpfbt` (and vice versa) in order to obtain a perfect reconstruction.

### Scaling of filters:

When compared to `wpfbt`, the subbands produced by `uwpfbt` are gradually more and more redundant with increasing depth in the tree. This results in energy grow of the coefficients. There are 3 flags defining filter scaling:

**'sqrt'** Each filter is scaled by  $1/\sqrt{a}$ , where  $a$  is the hop factor associated with it. If the original filterbank is orthonormal, the overall undecimated transform is a tight frame. This is the default.

**'noscale'** Uses filters without scaling.

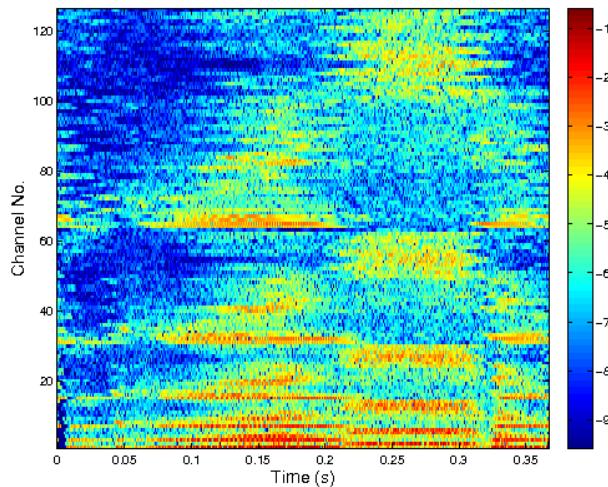
**'scale'** Each filter is scaled by  $1/a$ .

If '`noscale`' is used, '`scale`' must be used in `iuwpfbt` (and vice versa) in order to obtain a perfect reconstruction.

### Examples:

A simple example of calling the `uwpfbt` function using the "full decomposition" wavelet tree:

```
[f,fs] = greasy;
J = 6;
[c,info] = uwpfbt(f,{'db10',J,'full'});
plotwavelets(c,info,fs,'dynrange',90);
```



#### 4.2.8 IUWPFBT - Inverse Undecimated Wavelet Packet Filterbank Tree

##### Usage

```
f=iuwpfbt(c,info);
f=iuwpfbt(c,wt);
```

##### Input parameters

**c** Coefficients stored in  $L \times M$  matrix.

**info, wt** Transform parameters struct/Wavelet tree definition.

##### Output parameters

**f** Reconstructed data.

##### Description

`f = iuwpfbt(c, info)` reconstructs signal  $f$  from the wavelet packet coefficients  $c$  using parameters from `info` struct. both returned by the `uwpfbt` function.

`f = iuwpfbt(c, wt)` reconstructs signal  $f$  from the wavelet packet coefficients  $c$  using the undecimated wavelet filterbank tree described by `wt`.

Please see help for `wfbt` description of possible formats of `wt`.

##### Filter scaling:

As in `uwpfbt`, the function recognizes three flags controlling scaling of filters:

**'sqrt'** Each filter is scaled by  $1/\sqrt{a}$ , there  $a$  is the hop factor associated with it. If the original filterbank is orthonormal, the overall undecimated transform is a tight frame. This is the default.

**'noscale'** Uses filters without scaling.

**'scale'** Each filter is scaled by  $1/a$ .

If '`noscale`' is used, '`scale`' must have been used in `uwpfbt` (and vice versa) in order to obtain a perfect reconstruction.

**Scaling of intermediate outputs:**

The following flags control scaling of the intermediate coefficients. The intermediate coefficients are outputs of nodes which are also inputs to nodes further in the tree.

- 'intsqrt'** Each intermediate output is scaled by  $1/\sqrt{2}$ . If the filterbank in each node is orthonormal, the overall undecimated transform is a tight frame. This is the default.
- 'intnoscale'** No scaling of intermediate results is used.
- 'intscale'** Each intermediate output is scaled by  $1/2$ .

If 'intnoscale' is used, 'intscale' must have been used in uwpfbt (and vice versa) in order to obtain a perfect reconstruction.

**Examples:**

A simple example showing perfect reconstruction using the "full decomposition" wavelet tree:

```
f = greasy;
J = 7;
wtdef = {'db10',J,'full'};
c = uwpfbt(f,wtdef);
fhat = iuwpfbt(c,wtdef);
% The following should give (almost) zero
norm(f-fhat)
```

*This code produces the following output:*

```
ans =
3.2776e-14
```

**4.2.9 WPBEST - Best Tree selection****Usage**

```
c = wpbest(f,w,J,cost);
[c,info] = wpbest(...);
```

**Input parameters**

<b>f</b>	Input data.
<b>w</b>	Wavelet Filterbank.
<b>J</b>	Maximum depth of the tree.

**Output parameters**

<b>c</b>	Coefficients stored in a cell-array.
<b>info</b>	Transform parameters struct.

**Description**

[c,info]=wpbest(f,w,J,cost) selects the best sub-tree info.wt from the full tree with max. depth J, which minimizes the cost function.

Only one-dimensional input f is accepted. The supported formats of the parameter w can be found in help for fwt. The format of the coefficients c and the info struct is the same as in wfbt.

Please note that w should define orthonormal wavelet filters.

First, the depth  $J$  wavelet packet decomposition is performed using wfpfbt. Then the nodes are traversed in the breadth-first and bottom-up order and the value of the cost function of the node input and cost of the combined node outputs is compared. If the node input cost function value is less than the combined output cost, the current node and all possible descendant nodes are marked to be deleted, if not, the input is assigned the combined output cost. At the end, the marked nodes are removed and the resulting tree is considered to be a best basis (or near-best basis) in the chosen cost function sense.

The `cost` parameter can be a cell array or an user-defined function handle, accepting a single column vector. The cell array should consist of a string, followed by a numerical arguments. The possible formats are listed in the following text.

#### Additive costs:

The additive cost  $E$  of a vector  $x$  is a real valued cost function such that:

$$E(x) = \sum_k E(x(k))$$

and  $E(0) = 0$ . Given a collection of vectors  $x_i$  being coefficients in orthonormal bases  $B_i$ , the best basis relative to  $E$  is the one for which the  $E(x_i)$  is minimal.

Additive cost functions allows using the fast best-basis search algorithm since the costs can be precomputed and combined cost of two vectors is just a sum of their costs.

**{'shannon'}** A cost function derived from the Shannon entropy:

$$E_{sh}(x) = - \sum_{k:x(k) \neq 0} |x(k)|^2 \log |x(k)|^2$$

**{'log'}** A logarithm of energy:

$$E_{log}(x) = \sum_{k:x(k) \neq 0} \ln |x(k)|^2$$

**{'lpnorm', p}** Concentration in  $l^p$  norm:

$$E_{lp}(x) = \left( \sum_k |x(k)|^p \right)$$

**{'thre', th}** Number of coefficients above a threshold  $th$ .

#### Non-additive costs:

Cost function, which is not additive cost but which is used for the basis selection is called a non-additive cost. The resulting basis for which the cost is minimal is called near-best, because the non-additive cost cannot guarantee the selection of a best basis relative to the cost function.

**{'wlpnorm', p}** The weak- $l^p$  norm cost function:

$$E_{wlp}(x) = \max k^{1/p} v_k(x),$$

where  $0 < p \leq 2$  and  $v_k(x)$  denotes the  $k$ -th largest absolute value of  $x$ .

**{'compn', p, f}** Compression number cost:

$$E_{cn}(x) = \arg \min_k |w_k(x, p) - f|,$$

where  $0 < p \leq 2$ ,  $0 < f < 1$  and  $w_k(u, p)$  denotes decreasingly sorted, powered, cumulatively summed and renormalized vector:

$$w_k(x, p) = \frac{\sum_{j=1}^k v_j^p(x)}{\sum_{j=1}^N v_j^p(x)}$$

where  $v_k(x)$  denotes the  $k$ -th largest absolute value of  $x$  and  $N$  is the number of elements of  $x$ .

{'compa', p} Compression area cost:

$$E_{ca}(x) = N - \sum_k w_k(x, p),$$

where  $0 < p \leq 2$  and  $w_k(u, p)$  and  $N$  as in the previous case.

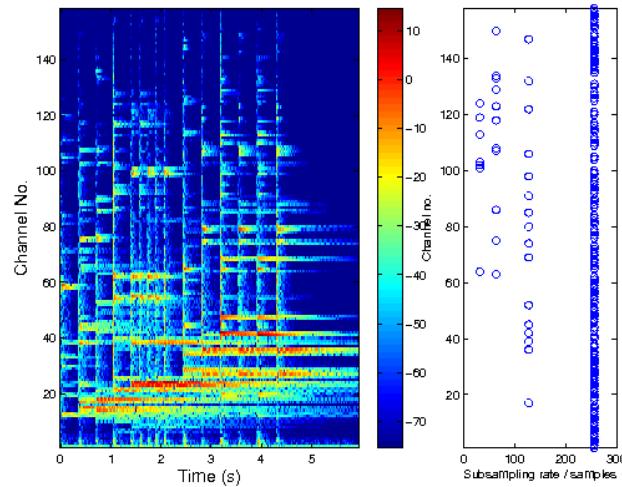
### Examples:

A simple example of calling wpbest

```
f = gspi;
J = 8;
[c,info] = wpbest(f,'sym10',J,'cost','shannon');

% Use 2/3 of the space for the first plot, 1/3 for the second.
subplot(3,3,[1 2 4 5 7 8]);
plotwavelets(c,info,44100,90);

subplot(3,3,[3 6 9]);
N=cellfun(@numel,c); L=sum(N); a=L./N;
plot(a,'o');
xlim([1,numel(N)]);
view(90,-90);
xlabel('Channel no.');
ylabel('Subsampling rate / samples');
```



**References:** [87], [82]

#### 4.2.10 WFBTLENGTH - WFBT length from signal

##### Usage

```
L=wfbtlength(Ls,wt);
```

**Description**

`wfbtlength(Ls, wt)` returns the length of a Wavelet system that is long enough to expand a signal of length  $L_s$ . Please see the help on `wfbt` for an explanation of the parameter `wt`.

If the returned length is longer than the signal length, the signal will be zero-padded by `wfbt` to length  $L$ .

In addition, the function accepts flags defining boundary extension technique as in `wfbt`. The returned length can be longer than the signal length only in case of '`per`' (periodic extension).

**4.2.11 WFBTCLENGTH - WFBT subband lengths from a signal length****Usage**

```
Lc=wfbtclength(Ls,wt);
[Lc,L]=wfbtclength(...);
```

**Description**

`Lc=wfbtclength(Ls, wt)` returns the lengths of coefficient subbands obtained from `wfbt` for a signal of length  $L_s$ . Please see the help on `wfbt` for an explanation of the parameters `wt`.

`[Lc, L]=wfbtclength(...)` additionally returns the next legal length of the input signal for the given extension type.

The function support the same boundary-handling flags as the `fwt` does.

**4.2.12 WPFBTCLENGTH - WPFBT subband length from a signal length****Usage**

```
Lc=wpfbtclength(Ls,wt);
[Lc,L]=wpfbtclength(Ls,wt);
```

**Description**

`Lc=wpfbtclength(Ls, wt)` returns the lengths of coefficient subbands obtained from `wpfbt` for a signal of length  $L_s$ . Please see the help on `wpfbt` for an explanation of the parameter `wt`.

`[Lc, L]=wpfbtclength(...)` additionally returns the next legal length of the input signal for the given extension type.

The function support the same boundary-handling flags as the `fwt` does.

**4.3 Dual-tree complex wavelet transform****4.3.1 DTWFB - Dual-Tree Wavelet FilterBank****Usage**

```
c=dtwfb(f,dualwt);
c=dtwfb(f,{dualw,J});
[c,info]=dtwfb(...);
```

**Input parameters**

**f** Input data.

**dualwt** Dual-tree Wavelet Filterbank definition.

**Output parameters**

**c** Coefficients stored in a cell-array.

**info** Additional transform parameters struct.

### Description

`c=dtwfbt(f, dualwt)` computes dual-tree complex wavelet coefficients of the signal  $f$ . The representation is approximately time-invariant and provides analytic behavior. Due to these facts, the resulting subbands are nearly aliasing free making them suitable for severe coefficient modifications. The representation is two times redundant, provided critical subsampling of all involved filterbanks.

The shape of the filterbank tree and filters used is controlled by `dualwt` (for possible formats see below). The output  $c$  is a cell-array with each element containing a single subband. The subbands are ordered with the increasing subband center frequency.

In addition, the function returns `struct. info` containing transform parameters. It can be conveniently used for the inverse transform `idtwfbreal` e.g. `fhat = idtwfbreal(c, info)`. It is also required by the `plotwavelets` function.

If  $f$  is a matrix, the transform is applied to each column.

Two formats of `dualwt` are accepted:

- 1) Cell array of parameters. First two elements of the array are mandatory `{dualw, J}`.

**dualw** Basic dual-tree filters

**J** Number of levels of the filterbank tree

Possible formats of `dualw` are the same as in `fwtnit` except the `wfilt dt_` prefix is used when searching for function specifying the actual impulse responses. These filters were designed specially for the dual-tree filterbank to achieve the half-sample shift ultimately resulting in analytic (complex) behavior of the transform.

The default shape of the filterbank tree is DWT i.e. only low-pass output is decomposed further ( $J$  times in total).

Different filterbank tree shapes can be obtained by passing additional flag in the cell array. Supported flags (mutually exclusive) are:

**'dwt'** Plain DWT tree (default). This gives one band per octave freq. resolution when using 2 channel basic wavelet filterbank.

**'full'** Full filterbank tree. Both (all) basic filterbank outputs are decomposed further up to depth  $J$  achieving linear frequency band division.

**'doubleband','quadband','octaband'** The filterbank is designed such that it mimics 4-band, 8-band or 16-band complex wavelet transform provided the basic filterbank is 2 channel. In this case,  $J$  is treated such that it defines number of levels of 4-band, 8-band or 16-band transform.

The dual-tree wavelet filterbank can use any basic wavelet filterbank in the first stage of both trees, provided they are shifted by 1 sample (done internally). A custom first stage filterbank can be defined by passing the following key-value pair in the cell array:

**'first','w'**  $w$  defines a regular basic filterbank. Accepted formats are the same as in `fwtnit` assuming the `wfilt_` prefix.

Similarly, when working with a filterbank tree containing decomposition of high-pass outputs, some filters in both trees must be replaced by a regular basic filterbank in order to achieve the approximately analytic behavior. A custom filterbank can be specified by passing another key-value pair in the cell array:

**'leaf','w'**  $w$  defines a regular basic filterbank. Accepted formats are the same as in `fwtnit` assuming the `wfilt_` prefix.

- 2) Another possibility is to pass directly a `struct.` returned by `dtwfbinit` and possibly modified by `wfbtremove`.

**Optional args.:**

In addition, the following flag groups are supported:

**'freq','nat'** Frequency or natural (Paley) ordering of coefficient subbands. By default, subbands are ordered according to frequency. The natural ordering is how the subbands are obtained from the filterbank tree without modifications. The ordering differs only in non-plain DWT case.

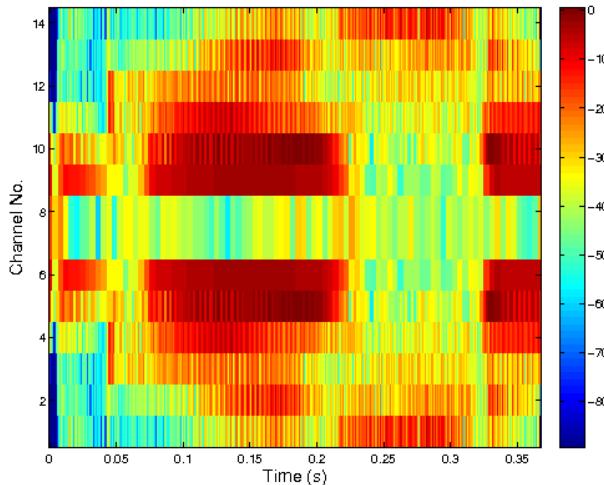
**Boundary handling:**

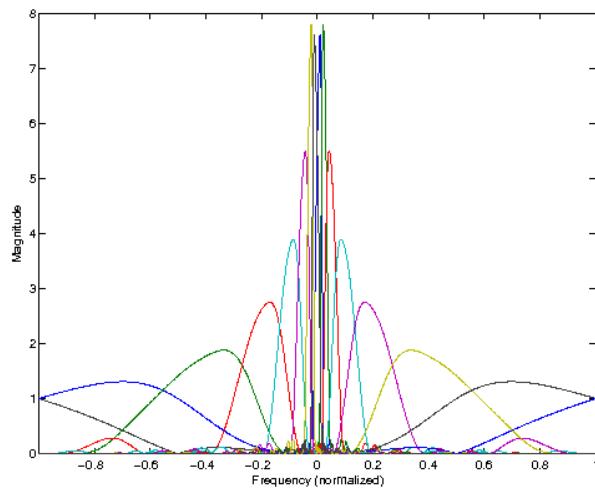
In contrast with fwt, wfbt and wpfbt, this function supports periodic boundary handling only.

**Examples:**

A simple example of calling the dtwfb function using the regular DWT iterated filterbank. The second figure shows a magnitude frequency response of an identical filterbank.:

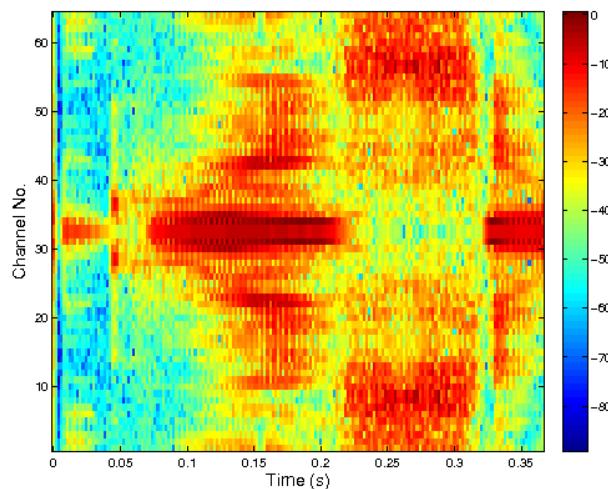
```
[f,fs] = greasy;
J = 6;
[c,info] = dtwfb(f,{'qshift3',J});
figure(1);
plotwavelets(c,info,fs,'dynrange',90);
figure(2);
[g,a] = dtwfb2filterbank({'qshift3',J});
filterbankfreqz(g,a,1024,'plot','linabs');
```

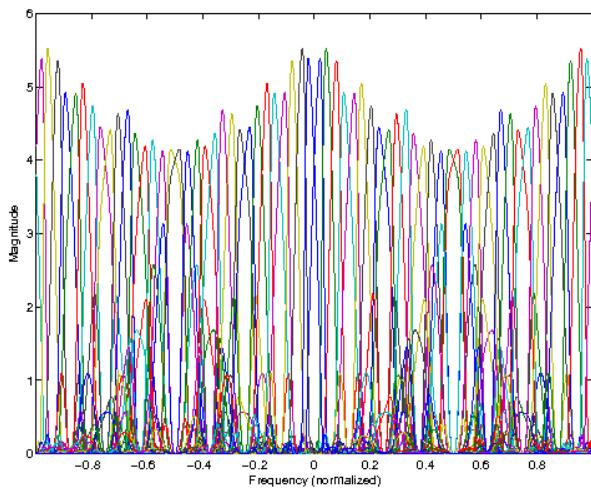




The second example shows a decomposition using a full filterbank tree of depth  $J$ :

```
[f,fs] = greasy;
J = 5;
[c,info] = dtwfb(f,{'qshift4',J,'full'});
figure(1);
plotwavelets(c,info,fs,'dynrange',90);
figure(2);
[g,a] = dtwfb2filterbank({'qshift4',J,'full'});
filterbankfreqz(g,a,1024,'plot','linabs');
```





**References:** [44], [77], [10]

### 4.3.2 IDTWFB - Inverse Dual-tree Filterbank

#### Usage

```
f=idtwfb(c,info);
f=idtwfb(c,dualwt,Ls);
```

#### Input parameters

<b>c</b>	Input coefficients.
<b>info</b>	Transform params. struct
<b>dualwt</b>	Dual-tree Wavelet Filterbank definition
<b>Ls</b>	Length of the reconstructed signal.

#### Output parameters

<b>f</b>	Reconstructed data.
----------	---------------------

#### Description

`f = idtwfb(c,info)` reconstructs signal  $f$  from the coefficients  $c$  using parameters from `info` struct, both returned by `dtwfb` function.

`f = idtwfb(c,dualwt,Ls)` reconstructs signal  $f$  from the coefficients  $c$  using dual-tree filterbank defined by `dualwt`. Please see `dtwfb` for supported formats. The `Ls` parameter is mandatory due to the ambiguity of reconstruction lengths introduced by the subsampling operation. Note that the same flag as in the `dtwfb` function have to be used, otherwise perfect reconstruction cannot be obtained. Please see help for `dtwfb` for description of the flags.

#### Examples:

A simple example showing perfect reconstruction using `idtwfb`:

```
f = gspi;
J = 7;
wtdef = {'qshift3',J};
c = dtwfb(f,wtdef);
fhat = idtwfb(c,wtdef,length(f));
```

```
% The following should give (almost) zero
norm(f-fhat)
```

*This code produces the following output:*

```
ans =
9.6082e-14
```

### 4.3.3 DTWFBREAL - Dual-Tree Wavelet FilterBank for real-valued signals

#### Usage

```
c=dtwfbreal(f,dualwt);
c=dtwfbreal(f,{dualw,J});
[c,info]=dtwfbreal(...);
```

#### Input parameters

<b>f</b>	Input data.
<b>dualwt</b>	Dual-tree Wavelet Filterbank definition.

#### Output parameters

<b>c</b>	Coefficients stored in a cell-array.
<b>info</b>	Additional transform parameters struct.

#### Description

`c=dtwfbtreal(f,dualwt)` computes dual-tree complex wavelet coefficients of the real-valued signal  $f$ . The representation is approximately time-invariant and provides analytic behavior. Due to these facts, the resulting subbands are nearly aliasing free making them suitable for severe coefficient modifications. The representation is two times redundant, provided critical subsampling of all involved filterbanks, but one half of the coefficients is complex conjugate of the other.

The shape of the filterbank tree and filters used is controlled by `dualwt` (for possible formats see below). The output  $c$  is a cell-array with each element containing a single subband. The subbands are ordered with increasing subband center frequency.

In addition, the function returns struct. `info` containing transform parameters. It can be conveniently used for the inverse transform `idtwfbreal` e.g. `fhat = idtwfbreal(c, info)`. It is also required by the `plotwavelets` function.

If  $f$  is a matrix, the transform is applied to each column.

Two formats of `dualwt` are accepted:

- 1) Cell array of parameters. First two elements of the array are mandatory `{dualw, J}`.

**dualw** Basic dual-tree filters

**J** Number of levels of the filterbank tree

Possible formats of `dualw` are the same as in `fwtinit` except the `wfilt dt_` prefix is used when searching for function specifying the actual impulse responses. These filters were designed specially for the dual-tree filterbank to achieve the half-sample shift ultimately resulting in analytic (complex) behavior of the transform.

The default shape of the filterbank tree is DWT i.e. only low-pass output is decomposed further ( $J$  times in total).

Different filterbank tree shapes can be obtained by passing additional flag in the cell array. Supported flags (mutually exclusive) are:

**'dwt'** Plain DWT tree (default). This gives one band per octave freq. resolution when using 2 channel basic wavelet filterbank.

**'full'** Full filterbank tree. Both (all) basic filterbank outputs are decomposed further up to depth  $J$  achieving linear frequency band division.

**'doubleband','quadband','octaband'** The filterbank is designed such that it mimics 4-band, 8-band or 16-band complex wavelet transform provided the basic filterbank is 2 channel. In this case,  $J$  is treated such that it defines number of levels of 4-band, 8-band or 16-band transform.

The dual-tree wavelet filterbank can use any basic wavelet filterbank in the first stage of both trees, provided they are shifted by 1 sample (done internally). A custom first stage filterbank can be defined by passing the following key-value pair in the cell array:

**'first','w'**  $w$  defines a regular basic filterbank. Accepted formats are the same as in fwtinit assuming the wfilt\_ prefix.

Similarly, when working with a filterbank tree containing decomposition of high-pass outputs, some filters in both trees must be replaced by a regular basic filterbank in order to achieve the approximately analytic behavior. A custom filterbank can be specified by passing another key-value pair in the cell array:

**'leaf','w'**  $w$  defines a regular basic filterbank. Accepted formats are the same as in fwtinit assuming the wfilt\_ prefix.

- 2) Another possibility is to pass directly a struct. returned by dtwfbinit and possibly modified by wfbtremove.

### Optional args.:

In addition, the following flag groups are supported:

**'freq','nat'** Frequency or natural (Paley) ordering of coefficient subbands. By default, subbands are ordered according to frequency. The natural ordering is how the subbands are obtained from the filterbank tree without modifications. The ordering differ only in non-plain DWT case.

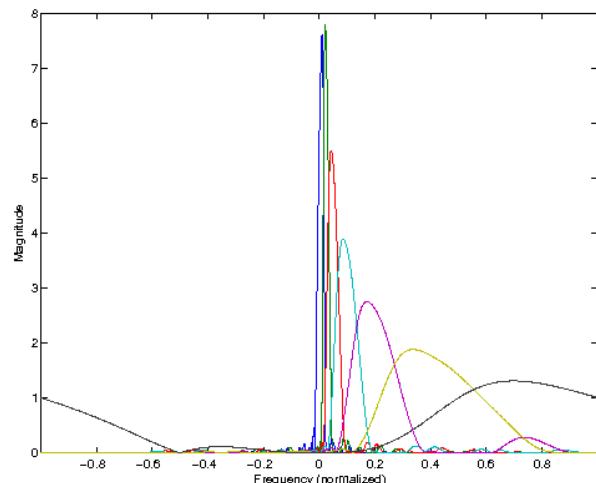
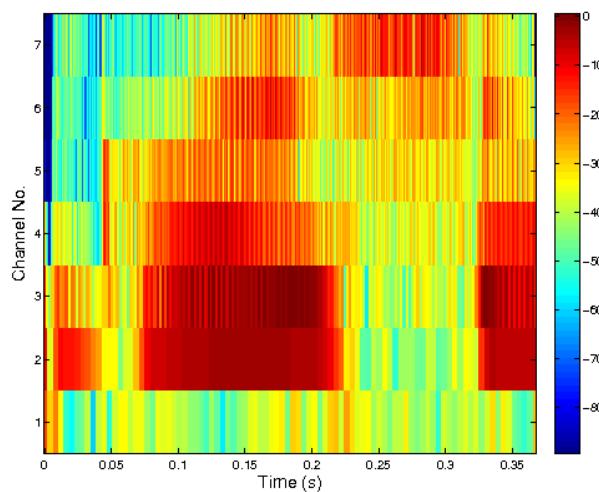
### Boundary handling:

In contrast with fwt, wfbt and wpfbt, this function supports periodic boundary handling only.

### Examples:

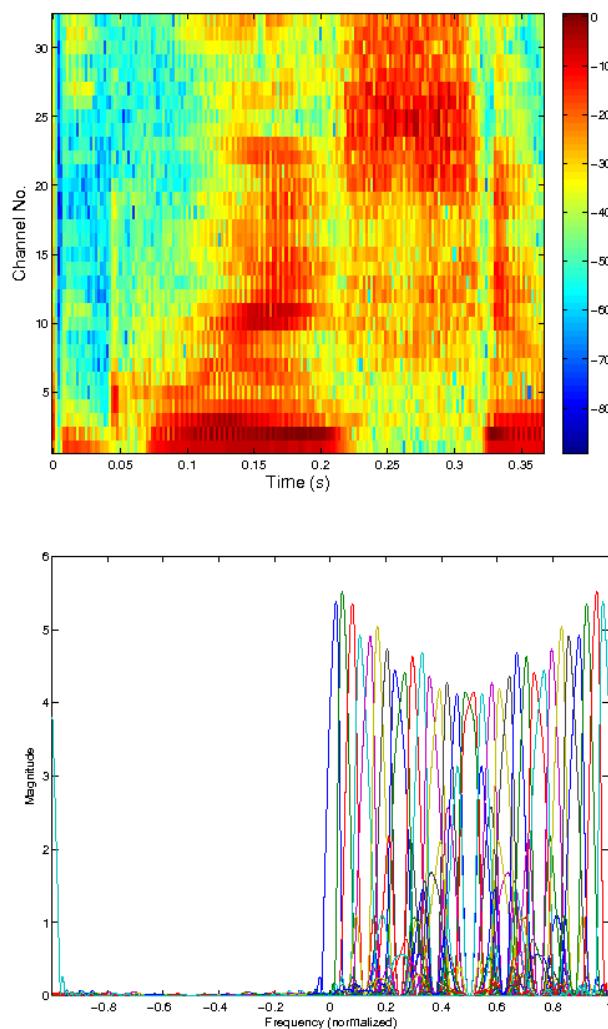
A simple example of calling the dtwfbreal function using the regular DWT iterated filterbank. The second figure shows a magnitude frequency response of an identical filterbank.:

```
[f,fs] = greasy;
J = 6;
[c,info] = dtwfbreal(f,'qshift3',J);
figure(1);
plotwavelets(c,info,fs,'dynrange',90);
figure(2);
[g,a] = dtwfb2filterbank({'qshift3',J},'real');
filterbankfreqz(g,a,1024,'plot','linabs');
```



The second example shows a decomposition using a full filterbank tree of depth  $J$ :

```
[f,fs] = greasy;
J = 5;
[c,info] = dtwfbleal(f,'qshift4',J,'full');
figure(1);
plotwavelets(c,info,fs,'dynrange',90);
figure(2);
[g,a] = dtwfb2filterbank({'qshift4',J,'full'},'real');
filterbankfreqz(g,a,1024,'plot','linabs');
```



**References:** [44], [77], [10]

#### 4.3.4 IDTWFBRREAL - Inverse Dual-tree Filterbank for real-valued signals

##### Usage

```
f=idtwfbreal(c,info);
f=idtwfbreal(c,dualwt,Ls);
```

##### Input parameters

<b>c</b>	Input coefficients.
<b>info</b>	Transform params. struct
<b>dualwt</b>	Dual-tree Wavelet Filterbank definition
<b>Ls</b>	Length of the reconstructed signal.

##### Output parameters

<b>f</b>	Reconstructed data.
----------	---------------------

**Description**

`f = idtwfbreal(c, info)` reconstructs real-valued signal  $f$  from the coefficients  $c$  using parameters from `info` struct. both returned by `dwtfbreal` function.

`f = idtwfbreal(c, dualwt, Ls)` reconstructs real-valued signal  $f$  from the coefficients  $c$  using dual-tree filterbank defined by `dualwt`. Please see `dwtfbreal` for supported formats. The `Ls` parameter is mandatory due to the ambiguity of reconstruction lengths introduced by the subsampling operation. Note that the same flag as in the `dwtfbreal` function have to be used, otherwise perfect reconstruction cannot be obtained. Please see help for `dwtfbreal` for description of the flags.

**Examples:**

A simple example showing perfect reconstruction using `idtwfbreal`:

```
f = gspi;
J = 7;
wtdef = {'qshift3', J};
c = dwtfbreal(f, wtdef);
fhat = idtwfbreal(c, wtdef, length(f));
% The following should give (almost) zero
norm(f-fhat)
```

*This code produces the following output:*

```
ans =
9.6082e-14
```

## 4.4 Wavelet Filterbank trees manipulation

### 4.4.1 WFBTINIT - Initialize Filterbank Tree

**Usage**

```
wt = wfbtinit(wtdef);
```

**Input parameters**

<b>wtdef</b>	Filterbank tree definition.
--------------	-----------------------------

**Output parameters**

<b>wt</b>	Structure describing the filter tree.
-----------	---------------------------------------

**Description**

`wfbtinit({w, J, flag})` creates a filterbank tree of depth  $J$ . The parameter  $w$  defines a basic wavelet filterbank. For all possible formats see `fwt`. The following optional flags (still inside of the cell-array) are recognized:

**'dwt','full','doubleband','quadband','octaband'** Type of the tree to be created.

`wfbtinit({w, J, flag, 'mod', mod})` creates a filterbank tree as before, but modified according to the value of `mod`. Recognized options:

**'powshiftable'** Changes subsampling factors of the root to 1. This results in redundant near-shift invariant representation.

The returned structure `wt` has the following fields:

**.nodes** Cell-array of structures obtained from `fwtinit`. Each element define a basic wavelet filterbank.

- .children** Indexes of children nodes
- .parents** Indexes of a parent node
- .forder** Frequency ordering of the resultant frequency bands.

The structure together with functions from the wfbtmanip subdirectory acts as an abstract data structure tree.

Regular wfbtinit flags:

- 'freq','nat'** Frequency or natural ordering of the coefficient subbands. The direct usage of the wavelet tree (' nat' option) does not produce coefficient subbands ordered according to the frequency. To achieve that, some filter shuffling has to be done (' freq' option).

#### 4.4.2 DTWFBINIT - Dual-Tree Wavelet Filterbank Initialization

##### Usage

```
dualwt=dtwfbinit (dualwtdef);
```

##### Input parameters

<b>dualwtdef</b>	Dual-tree filterbank definition.
------------------	----------------------------------

##### Output parameters

<b>dualwt</b>	Dual-tree filterbank structure.
---------------	---------------------------------

##### Description

dtwfbinit () (a call without arguments) creates an empty structure. It has the same fields as the struct returned from wfbtinit plus a field to hold nodes from the second tree:

- .nodes Filterbank nodes of the first tree
- .dualnodes Filterbank nodes of the second tree
- .children Indexes of children nodes
- .parents Indexes of a parent node
- .forder Frequency ordering of the resultant frequency bands.

dtwfbinit ({dualw, J, flag}) creates a structure representing a dual-tree wavelet filterbank of depth  $J$ , using dual-tree wavelet filters specified by `dualw`. The shape of the tree is controlled by `flag`. Please see help on `dtwfb` or `dtwfbreal` for description of the parameters.

[`dualwt, info`]=dtwfbinit (...) additionally returns `info` struct which provides some information about the computed window:

**info.tight** True if the overall tree construct a tight frame.

**info.dw** A structure containing basic dual-tree filters as returned from `fwtinit (dualwtdef, 'wfiltdt_')`.

##### Additional optional flags

- 'freq','nat'** Frequency or natural ordering of the resulting coefficient subbands. Default ordering is ' freq'.

#### 4.4.3 WFBTPUT - Put node to the filterbank tree

##### Usage

```
wt = wfbtput (d, k, w, wt);
wt = wfbtput (d, k, w, wt, 'force');
```

**Input parameters**

<b>d</b>	Level in the tree (0 - root).
<b>k</b>	Index (array of indexes) of the node at level <i>d</i> (starting at 0).
<b>w</b>	Node, basic wavelet filterbank.
<b>wt</b>	Wavelet filterbank tree structure (as returned from wfbtinit).

**Output parameters**

<b>wt</b>	Modified filterbank structure.
-----------	--------------------------------

**Description**

`wfbtput(d, k, w, wt)` puts the basic filterbank *w* to the filter tree structure *wt* at level *d* and index(es) *k*. The output is a modified tree structure. *d* and *k* have to specify unconnected output of the leaf node. Error is issued if *d* and *k* points to already existing node. For possible formats of parameter *w* see help of `fwt`. Parameter *wt* has to be a structure returned by `wfbtinit`.

`wfbtput(d, k, w, wt, 'force')` does the same but replaces node at *d* and *k* if it already exists. If the node to be replaced has any children, the number of outputs of the replacing node have to be equal to number of outputs of the node beeing replaced.

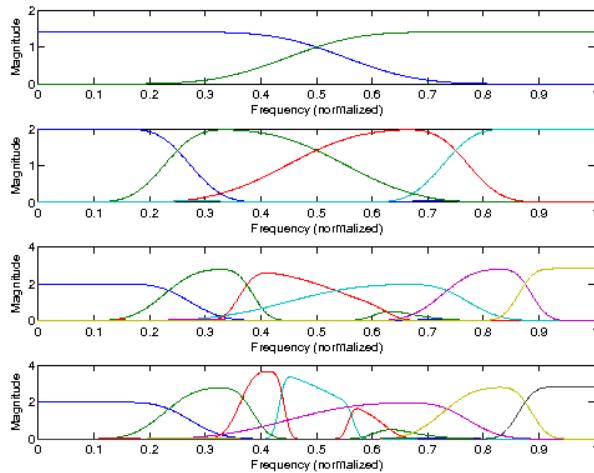
**Examples:**

This example shows magnitude frequency responses of a tree build from the root:

```
% Initialize empty struct
wt = wfbtinit();
% Put root node to the empty struct
wt1 = wfbtput(0,0,'db8',wt);
% Connect a different nodes to both outputs of the root
wt2 = wfbtput(1,[0,1],'db10',wt1);
% Connect another nodes just to high-pass outputs of nodes just added
wt3 = wfbtput(2,[1,3],'db10',wt2);
% Add another node at level 3
wt4 = wfbtput(3,1,'db16',wt3);

% Create identical filterbanks
[g1,a1] = wfbt2filterbank(wt1,'freq');
[g2,a2] = wfbt2filterbank(wt2,'freq');
[g3,a3] = wfbt2filterbank(wt3,'freq');
[g4,a4] = wfbt2filterbank(wt4,'freq');

% Plot frequency responses of the growing tree. Linear scale
% (both axis) is used and positive frequencies only are shown.
subplot(4,1,1);
filterbankfreqz(g1,a1,1024,'plot','linabs','posfreq');
subplot(4,1,2);
filterbankfreqz(g2,a2,1024,'plot','linabs','posfreq');
subplot(4,1,3);
filterbankfreqz(g3,a3,1024,'plot','linabs','posfreq');
subplot(4,1,4);
filterbankfreqz(g4,a4,1024,'plot','linabs','posfreq');
```



#### 4.4.4 WFBTREMOVE - Remove node(s) from the filterbank tree

##### Usage

```
wt = wbftremove(d,kk,wt);
wt = wfbtremove(d,kk,wt,'force');
```

##### Input parameters

- |           |  |
|-----------|--|
| <b>d</b>  | Level in the tree (0 - root).  |
| <b>kk</b> | Index of the node at level <i>d</i> (starting at 0) or array of indexes. |
| <b>wt</b> | Wavelet filterbank tree structure (as returned from wfbtinit).           |

##### Output parameters

- |           |                                |
|-----------|--------------------------------|
| <b>wt</b> | Modified filterbank structure. |
|-----------|--------------------------------|

##### Description

wfbtremove (*d*, *kk*, *wt*) removes existing node at level *d* and index *kk* from the filterbank tree structure *wt*. The function fails if the node has any children (it is not a leaf node).

wfbtremove (*d*, *k*, *wt*, 'force') does the same, but any children of the node are removed too.

##### Examples:

The following example shows magnitude frequency responses of filterbank tree before and after pruning.:

```
% Create a full filterbank tree usinf 'db10' basic filterbank.
wt1 = wfbtinit({'db10',4,'full'});
% Remove a subtree starting by root's high-pass filter. Force flag
% is used because we are removing a non-leaf node.
wt2 = wfbtremove(1,1,wt1,'force');

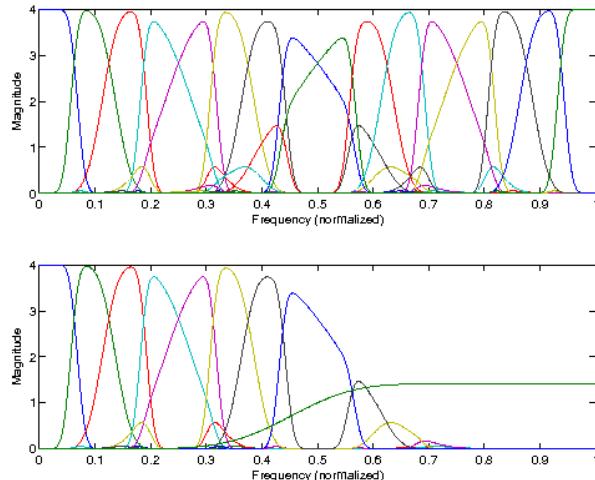
% Create identical filterbanks
[g1,a1] = wfbt2filterbank(wt1,'freq');
[g2,a2] = wfbt2filterbank(wt2,'freq');

% Plot the frequency responses
```

```

subplot(2,1,1);
filterbankfreqz(g1,a1,1024,'plot','posfreq','linabs');
subplot(2,1,2);
filterbankfreqz(g2,a2,1024,'plot','posfreq','linabs');

```



#### 4.4.5 WFBT2FILTERBANK - WFBT equivalent non-iterated filterbank

##### Usage

```
[g, a] = wfbt2filterbank(wt)
```

##### Input parameters

**wt** Wavelet filter tree definition

##### Output parameters

**g** Cell array containing filters

**a** Vector of sub-/upsampling factors

##### Description

`[g, a] = wfbt2filterbank(wt)` calculates the impulse responses *g* and the subsampling factors *a* of non-iterated filterbank, which is equivalent to the wavelet filterbank tree described by *wt*. The returned parameters can be used directly in filterbank and other routines.

`[g, a] = wfbt2filterbank({w, J, 'dwt'})` does the same for the DWT (fwt) filterbank tree.

Please see help on *wfbt* for description of *wt*. The function additionally support the following flags:

**'freq' '(default)', 'nat'** The filters are ordered to produce subbands in the same order as *wfbt* with the same flag.

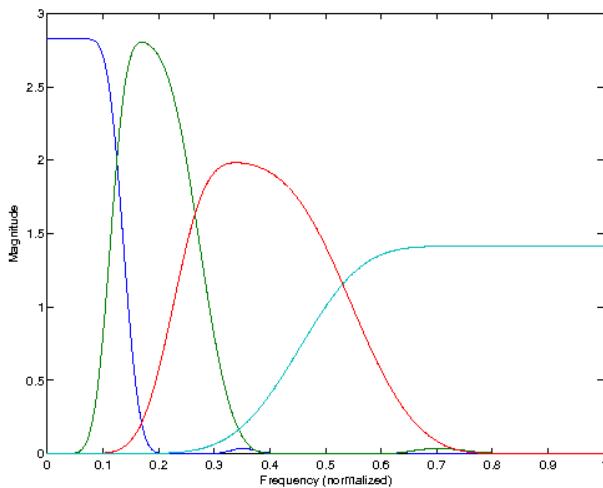
##### Examples:

The following two examples create a multirate identity filterbank using a tree of depth 3. In the first example, the filterbank is identical to the DWT tree:

```

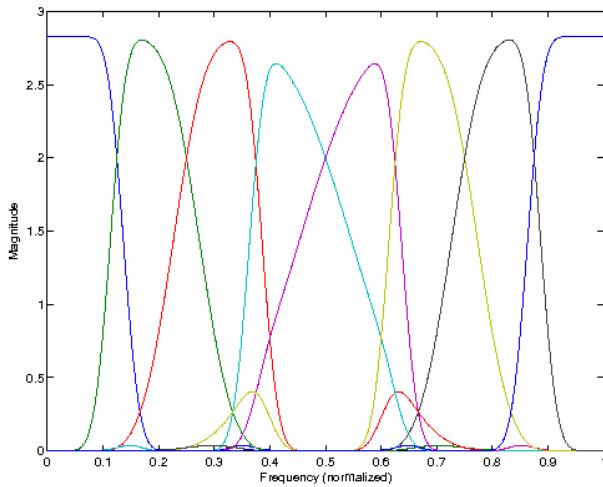
[g, a] = wfbt2filterbank({'db10', 3, 'dwt'});
filterbankfreqz(g, a, 1024, 'plot', 'linabs', 'posfreq');

```



In the second example, the filterbank is identical to the full wavelet tree:

```
[g,a] = wfbt2filterbank({'db10',3,'full'});
filterbankfreqz(g,a,1024,'plot','linabs','posfreq');
```



#### 4.4.6 WPFBT2FILTERBANK - WPFBT equivalent non-iterated filterbank

##### Usage

```
[g,a] = wpfbt2filterbank(wt)
```

##### Input parameters

**wt** Wavelet filter tree definition

##### Output parameters

**g** Cell array containing filters

**a** Vector of sub-/upsampling factors

### Description

`wpfbt2filterbank(wt)` calculates the impulse responses  $g$  and the subsampling factors  $a$  of non-iterated filterbank, which is equivalent to the wavelet packet filterbank tree described by  $wt$ . The returned parameters can be used directly in filterbank, ufilterbank or filterbank.

Please see help on `wfbt` for description of  $wt$ . The function additionally support the following flags:

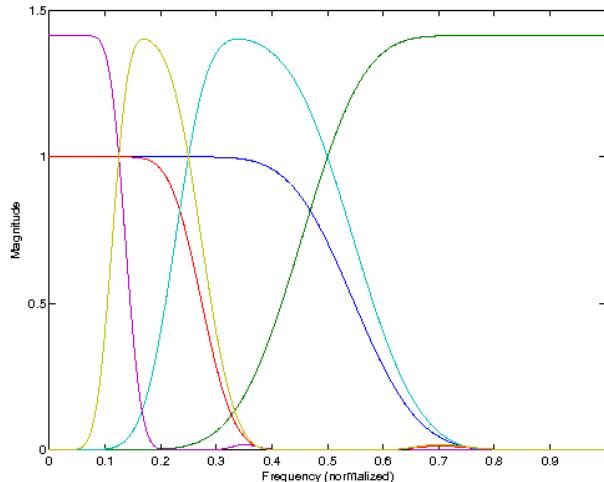
'**freq**' '(**default**) , '**nat**' The filters are ordered to produce subbands in the same order as `wpfbt` with the same flag.

'**intsqrt**' '(**default**) , '**intnoscale**' , '**intscale**' The filters in the filterbank tree are scaled to reflect the behavior of `wpfbt` and `iwpfbt` with the same flags.

### Examples:

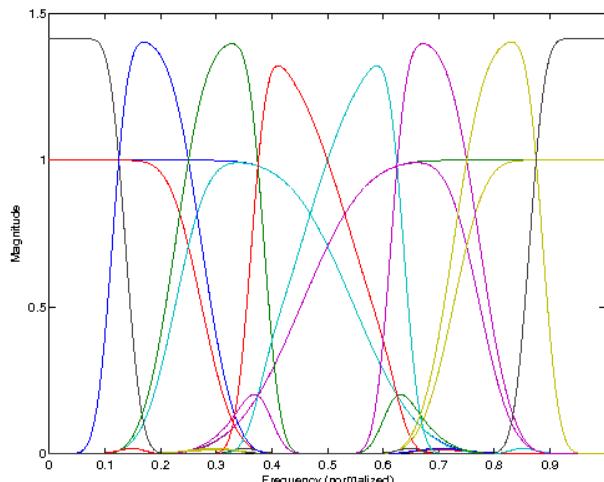
The following two examples create a multirate identity filterbank using a tree of depth 3. In the first example, the filterbank is identical to the DWT tree:

```
[g,a] = wpfbt2filterbank({'db10',3,'dwt'});
filterbankfreqz(g,a,1024,'plot','linabs','posfreq');
```



In the second example, the filterbank is identical to the full wavelet tree:

```
[g,a] = wpfbt2filterbank({'db10',3,'full'});
filterbankfreqz(g,a,1024,'plot','linabs','posfreq');
```



#### 4.4.7 DTWFB2FILTERBANK - DTWFB equivalent non-iterated filterbank

##### Usage

```
[g,a] = dtwfb2filterbank(dualwt)
[g,a,info] = dtwfb2filterbank(...)
```

##### Input parameters

**dualwt** Dual-tree wavelet filterbank specification.

##### Output parameters

**g** Cell array of filters.

**a** Downsampling rate for each channel.

**info** Additional information.

##### Description

[*g*,*a*] = dtwfb2filterbank(*dualwt*) constructs a set of filters *g* and subsampling factors *a* of a non-iterated filterbank, which is equivalent to the dual-tree wavelet filterbank defined by *dualwt*. The returned parameters can be used directly in filterbank and other routines. The format of *dualwt* is the same as in dtwfb and dtwfbleal.

The function internally calls dtwfbinit and passes *dualwt* and all additional parameters to it.

[*g*,*a*,*info*] = dtwfb2filterbank(...) additionally outputs a *info* struct containing equivalent filterbanks of individual real-valued trees as fields *info.g1* and *info.g2*.

##### Additional parameters:

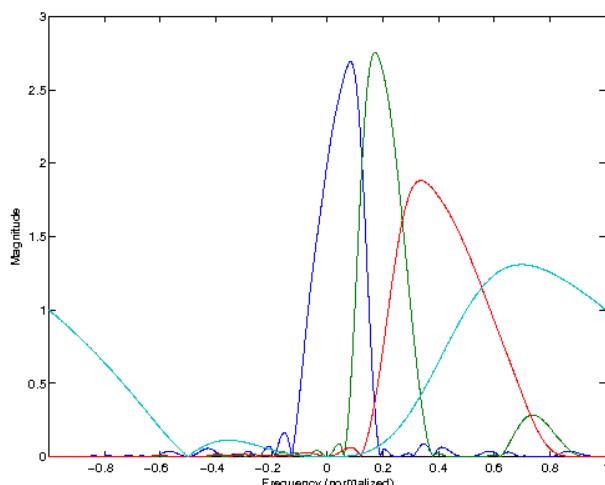
**'real'** By default, the function returns a filterbank equivalent to dtwfb. The filters can be restricted to cover only the positive frequencies and to be equivalent to dtwfbleal by passing a '**'real'**' flag.

**'freq' '(default)', 'nat'** The filters are ordered to produce subbands in the same order as dtwfb or dtwfbleal with the same flag.

##### Examples:

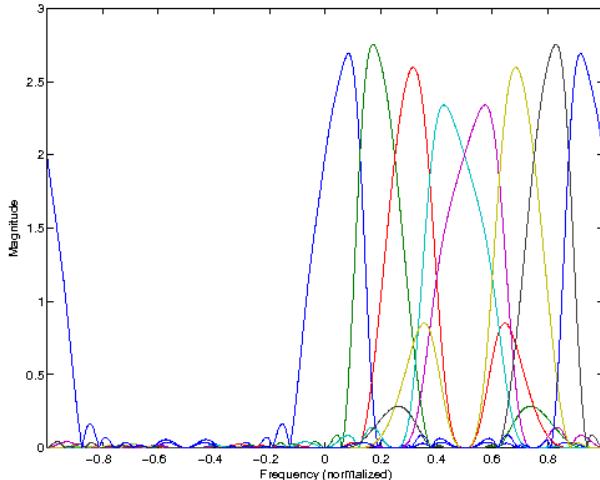
The following two examples create a multirate identity filterbank using a dual-tree of depth 3:

```
[g,a] = dtwfb2filterbank({'qshift3',3}, 'real');
filterbankfreqz(g,a,1024, 'plot', 'linabs');
```



In the second example, the filterbank is identical to the full wavelet tree:

```
[g,a] = dtwfb2filterbank({'qshift3',3,'full'},'real');
filterbankfreqz(g,a,1024,'plot','linabs');
```



#### 4.4.8 FWTINIT - Wavelet Filterbank Structure Initialization

##### Usage

```
w = fwtinit(wdef);
w = fwtinit(wdef,prefix);
[w,info]=fwtinit(...)
```

##### Input parameters

**wdef** Wavelet filters specification.

**prefix** Function name prefix

##### Output parameters

**w** Structure defining the filterbank.

##### Description

`fwtinit(wdef)` produces a structure describing the analysis (field `w.h`) and synthesis (field `w.g`) filterbanks and a hop factors (field `w.a`) of a basic wavelet-type filterbank defined by `wdef`.

The analysis filterbank `w.h` is by default used in `fwt` and the synthesis filterbank `w.g` in `ifwt`.

Both `w.h` and `w.g` are cell arrays of structs defining FIR filters compatible with `filterbank`, `ifilterbank` and related functions. More precisely, each element of either cell array is a struct with fields `.h` and `.offset` defining impulse response and the initial shift respectively.

`[w,info]=fwtinit(...)` additionally returns a `info` struct which provides some information about the wavelet filterbank:

**info.istight** Wavelet filterbank forms a tight frame. In such case, `w.h` and `w.g` are identical.

The function is a wrapper for calling all the functions with the `wfilt_` prefix defined in the LTFAT wavelets directory.

The possible formats of the `wdef` are the following:

- 1) Cell array with first element being the name of the function defining the basic wavelet filters (`wfilt_` prefix) and the other elements are the parameters of the function.
- 2) Character string as concatenation of the name of the wavelet filters defining function (as above) and the numeric parameters delimited by `:` character. Examples:
  - `{'db', 10} or 'db10'` Daubechies with 10 vanishing moments. It calls `wfilt_db(10)` internally.
  - `{'spline', 4, 4} or 'spline4:4'` Biorthogonal spline wavelet filters with 4 vanishing moments. Calls `wfilt_spline(4, 4)` internally.
  - `{'dden', 1} or 'dden1'` Double density wavelet filters. Calls `wfilt_dden(1)` where the filters are stored as numerical vectors.
- 3) Cell array of one dimensional numerical vectors directly defining the wavelet filter impulse responses. By default, outputs of the filters are subsampled by a factor equal to the number of the filters. Pass additional key-value pair `'a',a` (still inside of the cell array) to define the custom subsampling factors, e.g.: `{h1,h2,'a',[2,2]}`.
- 4) The fourth option is to pass again the structure obtained from the `fwtinit` function. The structure is checked whether it has a valid format.
- 5) Two element cell array. First element is the string `'dual'` and the second one is in format 1), 2) or 4). This returns a dual of whatever is passed as the second argument.
- 6) Two element cell array. First element is the string `'strict'` and the second one is in format 1), 2), 4) or 5). This in the non tight case the filters has to be defined explicitly using `'ana'` and `'syn'` identifiers. See below.

One can interchange the filter in `w.h` and `w.g` and use the filterbank intended for synthesis in `fwt` and vice versa by re-using the items 1) and 2) in the following way:

- 1) Add `'ana'` or `'syn'` as the first element in the cell array e.g. `{'ana', 'spline', 4, 4}` or `{'syn', 'spline', 4, 4}`.
- 2) Add `'ana:'` or `'syn:'` to the beginning of the string e.g. `'ana:spline4:4'` or `'syn:spline4:4'`.

This only makes difference if the filterbanks are biorthogonal (e.g. `wfilt_spline`) or a general frame (e.g. `'symds2'`), in other cases, the analysis and synthesis filters are identical.

Please note that using e.g. `c=fwt(f, 'ana:spline4:4', J)` and `fhat=ifwt(c, 'ana:spline4:4', J, size(f,` will not give a perfect reconstruction.

The output structure has the following additional field:

`w.origArgs` Original parameters in format 1).

**References:** [55]

## 4.5 Frame properties of wavelet filterbanks:

### 4.5.1 WFBTBOUNDS - Frame bounds of WFBT

#### Usage

```
fcond=wfbtbounds(wt,L);
[A,B]=wfbtbounds(wt,L);
```

#### Description

`wfbtbounds(wt, L)` calculates the ratio  $B/A$  of the frame bounds of the filterbank specified by `wt` for a system of length  $L$ . The ratio is a measure of the stability of the system.

`wfbtbounds({w, J, 'dwt'}, L)` calculates the ratio  $B/A$  of the frame bounds of the DWT (`fwt`) filterbank specified by `w` and `J` for a system of length  $L$ .

`[A, B]=wfbtbounds(..., L)` returns the lower and upper frame bounds explicitly.

See `wfbt` for explanation of parameter `wt` and `fwt` for explanation of parameters `w` and `J`.

### 4.5.2 WPFBTBOUNDS - Frame bounds of WPFBT

#### Usage

```
fcond=wpfbtbounds (wt,L);
[A,B]=wpfbtbounds (wt,L);
```

#### Description

`wpfbtbounds (wt,L)` calculates the ratio  $B/A$  of the frame bounds of the wavelet packet filterbank specified by `wt` for a system of length  $L$ . The ratio is a measure of the stability of the system.

`[A,B]=wpfbtbounds (wt,L)` returns the lower and upper frame bounds explicitly.

See `wfbt` for explanation of parameter `wt`.

Additionally, the function accepts the following flags:

`'intsqrt' '(default)', 'intnoscale', ''intscale'` The filters in the filterbank tree are scaled to reflect the behavior of `wpfbt` and `iwpfbt` with the same flags.

### 4.5.3 UWFBTBOUNDS - Frame bounds of Undecimated WFBT

#### Usage

```
fcond=uwfbtbounds (wt,L);
[A,B]=uwfbtbounds (wt,L);
```

#### Description

`uwfbtbounds (wt,L)` calculates the ratio  $B/A$  of the frame bounds of the undecimated filterbank specified by `wt` for a system of length  $L$ . The ratio is a measure of the stability of the system.

`uwfbtbounds ({w,J,'dwt'},L)` calculates the ratio  $B/A$  of the frame bounds of the undecimated DWT (`ufwt`) filterbank specified by `w` and `J` for a system of length  $L$ .

`[A,B]=uwfbtbounds (...,L)` returns the lower and upper frame bounds explicitly.

See `wfbt` for explanation of parameter `wt` and `fwt` for explanation of parameters `w` and `J`.

The function supports the following flags:

`'sqrt' '(default)', 'noscale', 'scale'` The filters in the filterbank tree are scaled to reflect the behavior of `uwfbt` and `iuwfbt` with the same flags.

### 4.5.4 UWPFBTBOUNDS - Frame bounds of Undecimated WPFBT

#### Usage

```
fcond=uwpfbtbounds (wt,L);
[A,B]=uwpfbtbounds (wt,L);
```

#### Description

`uwpfbtbounds (wt,L)` calculates the ratio  $B/A$  of the frame bounds of the undecimated wavelet packet filterbank specified by `wt` for a system of length  $L$ . The ratio is a measure of the stability of the system.

`[A,B]=uwpfbtbounds (wt,L)` returns the lower and upper frame bounds explicitly.

See `wfbt` for explanation of parameter `wt`.

Additionally, the function accepts the following flags:

`'intsqrt' '(default)', 'intnoscale', ''intscale'` The filters in the filterbank tree are scaled to reflect the behavior of `uwpfbt` and `iuwpfbt` with the same flags.

`'sqrt' '(default)', 'noscale', 'scale'` The filters in the filterbank tree are scaled to reflect the behavior of `uwpfbt` and `iuwpfbt` with the same flags.

## 4.6 Plots

### 4.6.1 PLOTWAVELETS - Plot wavelet coefficients

#### Usage

```
plotwavelets(c,info,fs)
plotwavelets(c,info,fs,'dynrange',dynrange,...)
```

#### Description

`plotwavelets(c,info)` plots the wavelet coefficients `c` using additional parameters from struct. `info`. Both parameters are returned by any forward transform function in the wavelets directory.

`plotwavelets(c,info,fs)` does the same plot assuming a sampling rate `fs` Hz of the original signal.

`plotwavelets(c,info,fs,'dynrange',dynrange)` additionally limits the dynamic range.

`C=plotwavelets(...)` returns the processed image data used in the plotting. Inputting this data directly to `imagesc` or similar functions will create the plot. This is usefull for custom post-processing of the image data.

`plotwavelets` supports optional parameters of `tfplot`. Please see the help of `tfplot` for an exhaustive list.

### 4.6.2 WFILTINFO - Plots filters info

#### Usage

```
wfiltinfo(w);
```

#### Input parameters

<code>w</code>	Basic wavelet filterbank.
----------------	---------------------------

#### Description

`wfiltinfo(w)` plots impulse responses, frequency responses and approximation of the scaling and of the wavelet function(s) associated with the wavelet filters defined by `w` in a single figure. Format of `w` is the same as in `fwt`.

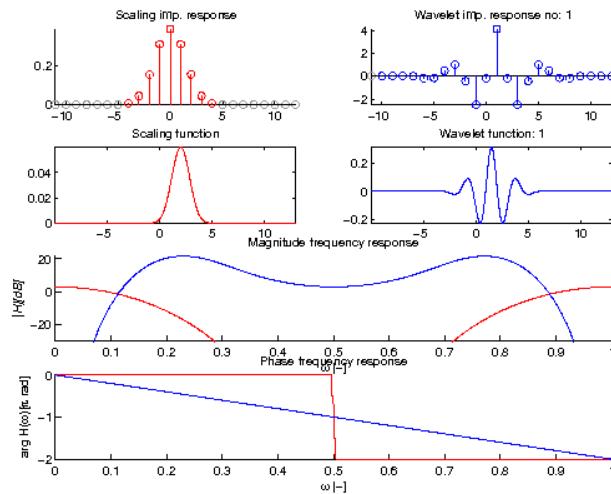
Optionally it is possible to define scaling of the y axis of the frequency seponses. Supported are:

'`db`', '`lin`' dB or linear scale respectively. By deault a dB scale is used.

#### Examples:

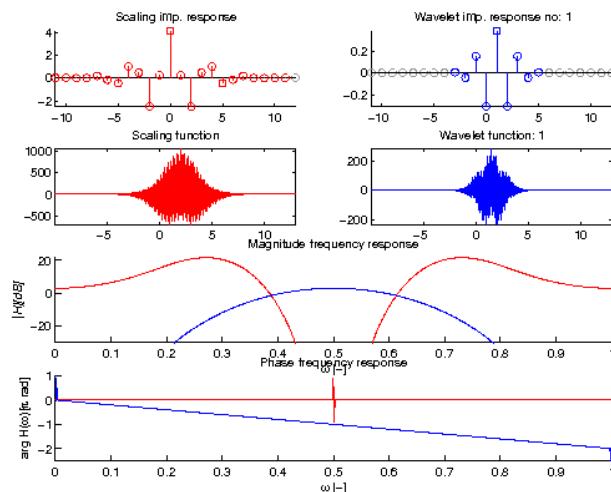
Details of the '`syn:spline8:8`' wavelet filters (see `wfilt_spline`):

```
wfiltinfo('syn:spline8:8');
```



Details of the 'ana:spline8:8' wavelet filters:

```
wfiltinfo('ana:spline8:8');
```



### 4.6.3 WFILTDTINFO - Plots dual-tree filters info

#### Usage

```
wfiltdtinfo(dw);
```

#### Input parameters

<b>dw</b>	Wavelet dual-tree filterbank
-----------	------------------------------

#### Description

`wfiltdtinfo(w)` plots impulse responses, frequency responses and approximation of the scaling and of the wavelet function(s) associated with the dual-tree wavelet filters defined by `w` in a single figure. Format of `dw` is the same as in `dwtfb`.

The figure is organized as follows:

First row shows impulse responses of the first (real) tree.

Second row shows impulse responses of the second (imag) tree.

Third row contains plots of real (green), imaginary (red) and absolute (blue) parts of approximation of scaling and wavelet function(s).

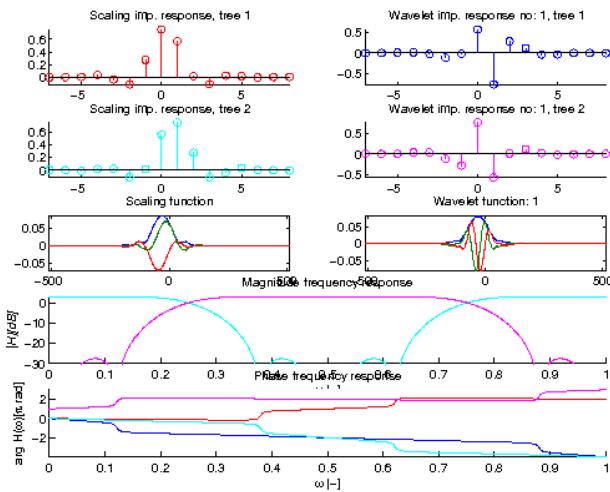
Fourth and fifth row show magnitude and phase frequency responses respectively of filters from rows 1 and 2 with matching colors.

Optionally it is possible to define scaling of the y axis of the frequency seponses. Supported are:

'db', 'lin' dB or linear scale respectivelly. By deault a dB scale is used.

### Examples:

```
wfiltdtinfo('qshift4');
```



## 4.7 Auxilary

### 4.7.1 WAVFUN - Wavelet Function

#### Usage

```
[w,s,xvals] = wavfun(g)
[w,s,xvals] = wavfun(g,N)
```

#### Input parameters

**w** Wavelet filterbank

**N** Number of iterations

#### Output parameters

**wfunc** Approximation of wavelet function(s)

**sfunc** Approximation of the scaling function

**xvals** Correct x-axis values

### Description

Iteratively generate ( $N$  iterations) a discrete approximation of wavelet and scaling functions using filters obtained from  $w$ . The possible formats of  $w$  are the same as for the fwt function. The algorithm is equal to the DWT reconstruction of a single coefficient at level  $N + 1$  set to 1.  $xvals$  contains correct x-axis values. All but last columns belong to the  $wfunc$ , last one to the  $sfunc$ .

The following flags are supported (first is default):

**'fft'**, **'conv'** How to do the computations. Whatever is faster depends on the speed of the `conv2` function.

**WARNING:** The output array lengths  $L$  depend on  $N$  exponentially like:

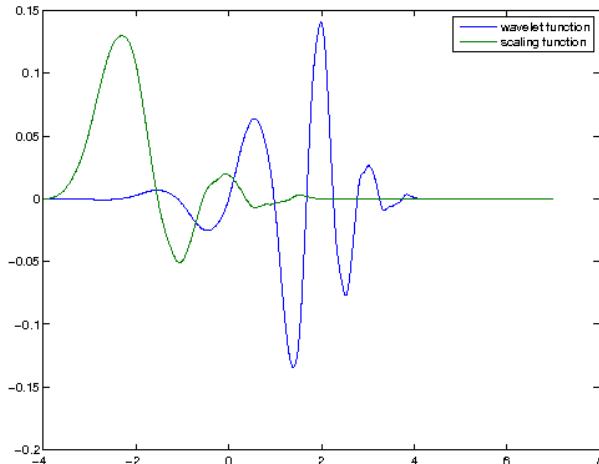
$$L = \frac{a^N - 1}{a - 1} (m - 1) + 1$$

where  $a$  is subsampling factor after the lowpass filter in the wavelet filterbank and  $m$  is length of the filters. Expect issues for high  $N$  e.g. '`db10`' ( $m = 20$ ) and  $N = 20$  yields a ~150MB array.

### Examples:

Approximation of a Daubechies wavelet and scaling functions from the 12 tap filters:

```
[wfn, sfn, xvals] = wavfun('db6');
plot(xvals, [wfn, sfn]);
legend('wavelet function', 'scaling function');
```



### 4.7.2 WAVCELL2PACK - Changes wavelet coefficients storing format

#### Usage

```
[cvec, Lc] = wavcell2pack(ccell);
[cvec, Lc] = wavcell2pack(ccell, dim);
```

#### Input parameters

**ccell** Coefficients stored in a column cell-array.

**dim** Dimension along which the data were transformed.

**Output parameters**

<b>cvec</b>	Coefficients in packed format.
<b>Lc</b>	Vector containing coefficients lengths.

**Description**

[cvec, Lc] = wavcell2pack(ccell) assembles a column vector or a matrix *cvec* using elements of the cell-array *ccell* in the following manner:

```
cvec(1+sum(Lc(1:j-1)) : sum(Lc(1:j), :) ) = ccell{j};
```

where *Lc* is a vector of length *numel(ccell)* containing number of rows of each element of *ccell*.

[cvec, Lc] = wavcell2pack(ccell, dim) with *dim*=2 returns a transposition of the previous.

**4.7.3 WAVPACK2CELL - Changes wavelet coefficients storing format****Usage**

```
ccell = wavpack2cell(cvec, Lc);
ccell = wavpack2cell(cvec, Lc, dim);
```

**Input parameters**

<b>cvec</b>	Coefficients in packed format.
<b>Lc</b>	Vector containing coefficients lengths.
<b>dim</b>	Dimension along which the data were transformed.

**Output parameters**

<b>ccell</b>	Coefficients stored in a cell-array. Each element is a column vector or a matrix.
<b>dim</b>	Return used dim. Usefull as an input of the complementary function wavcell2pack.

**Description**

ccell = wavpack2cell(cvec, Lc) copies coefficients from a single column vector or columns of a matrix *cvec* of size [sum(*Lc*), *W*] to the cell array *ccell* of length *length(Lc)*. Size of *j*-th element of *ccell* is [*Lc(j)*, *W*] and it is obtained by:

```
ccell{j}=cvec(1+sum(Lc(1:j-1)) : sum(Lc(1:j), :) );
```

ccell = wavpack2cell(cvec, Lc, dim) allows specifying along which dimension the coefficients are stored in *cvec*. *dim*=1 (default) considers columns (as above) and *dim*=2 rows to be coefficients belonging to separate channels. Other values are not supported. For *dim*=2, *cvec* size is [*W*, sum(*Lc*)], Size of *j*-th element of *ccell* is [*Lc(j)*, *W*] and it is obtained by:

```
ccell{j}=cvec(:, 1+sum(Lc(1:j-1)) : sum(Lc(1:j)) .';
```

**4.8 Wavelet Filters defined in the time-domain****4.8.1 WFILT\_ALGMBAND - An ALgebraic construction of orthonormal M-BAND wavelets with perfect reconstruction****Usage**

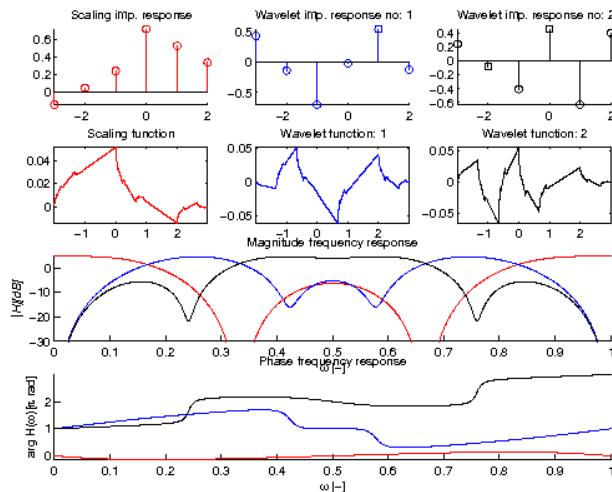
```
[h, g, a] = wfilt_algmband(K);
```

### Description

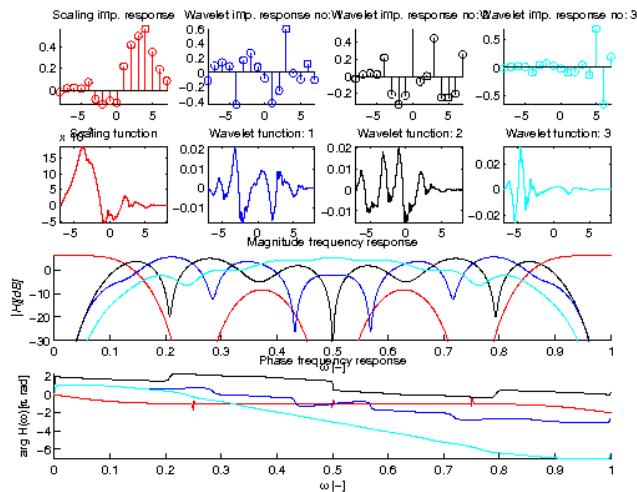
`[h, g, a]=wfilt_algmband(K)` with  $K \in 1, 2$  returns wavelet filters from the reference paper. The filters are 3-band ( $K == 1$ ) and 4-band ( $K == 2$ ) with critical subsampling.

### Examples:

```
wfiltinfo('algmband1');
```



```
wfiltinfo('algmband2');
```



References: [51]

### 4.8.2 WFILT\_CMBAND - Generates M-Band cosine modulated wavelet filters

#### Usage

```
[h, g, a] = wfilt_cmband(M);
```

#### Input parameters

**M**

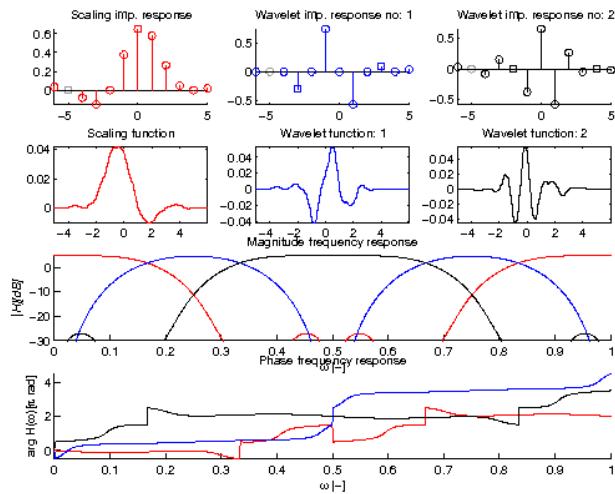
Number of channels.

### Description

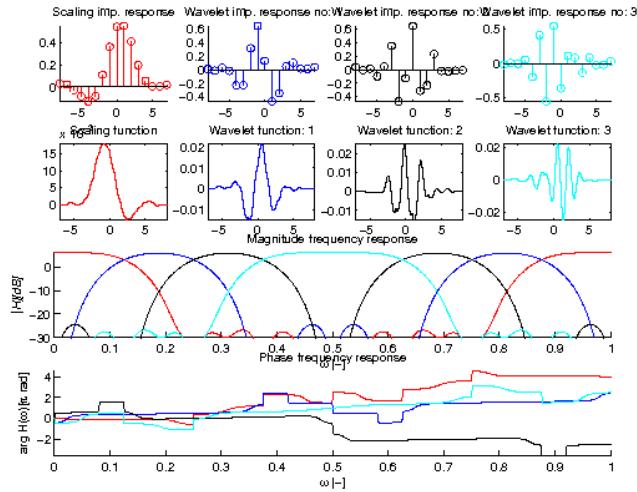
`[h, g, a]=wfilt_cmband(M)` with  $M \in 2, 3, \dots$  returns smooth, 1-regular cosine modulated  $M$ -band wavelet filters according to the reference paper. The length of the filters is  $4M$ .

### Examples:

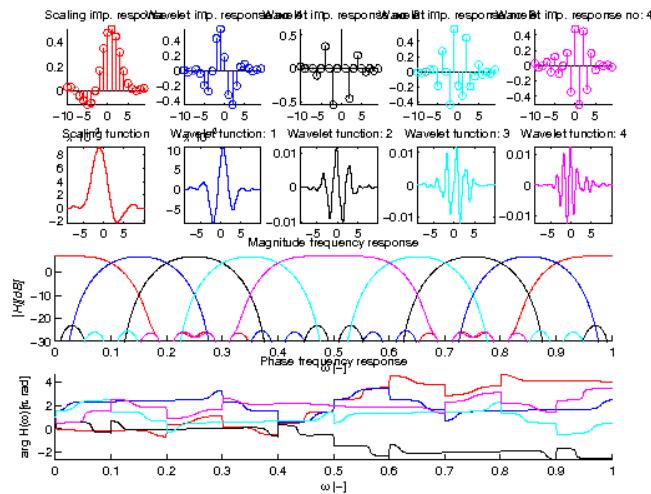
```
wfiltinfo('cmband3');
```



```
wfiltinfo('cmband4');
```



```
wfiltinfo('cmband5');
```



**References:** [34]

### 4.8.3 WFILT\_COIF - Coiflets

#### Usage

```
[h, g, a] = wfilt_coif(K);
```

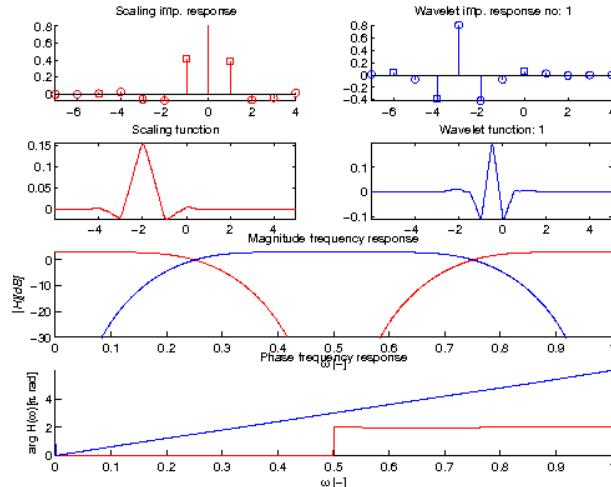
#### Description

[h, g, a]=wfilt\_coif(K) with  $K \in 1, 2, 3, 4, 5$  returns a Coiflet filters of order  $2K$  the number of vanishing moments of both the scaling and the wavelet functions.

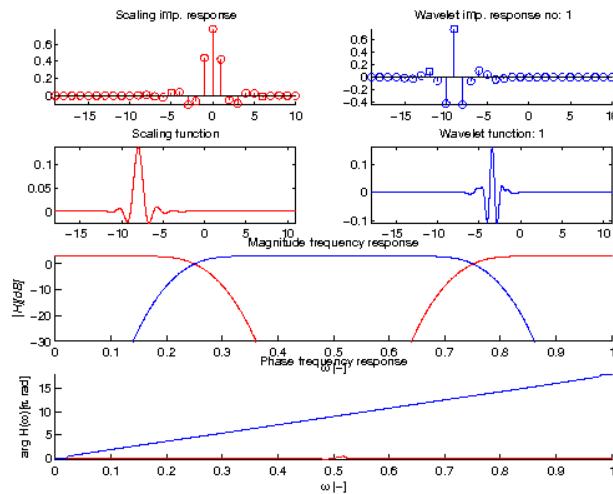
Values are taken from table 8.1 from the reference. REMARK: There is a typo in 2nd element for  $K == 1$ .

#### Examples:

```
wfiltinfo('coif2');
```



```
wfiltinfo('coif5');
```



**References:** [22]

#### 4.8.4 WFILT\_DB - Daubechies FIR filterbank

##### Usage

```
[h, g] = wfilt_db(N);
```

##### Input parameters

**N** Order of Daubechies filters.

##### Output parameters

**H** cell array of analysing filters impulse responses

**G** cell array of synthetizing filters impulse responses

**a** array of subsampling (or hop) factors accociated with corresponding filters

##### Description

$[H, G] = \text{dbfilt}(N)$  computes a two-channel Daubechies FIR filterbank from prototype maximum-phase analysing lowpass filter obtained by spectral factorization of the Lagrange interpolator filter.  $N$  also denotes the number of zeros at  $z = -1$  of the lowpass filters of length  $2N$ . The prototype lowpass filter has the following form (all roots of  $R(z)$  are outside of the unit circle):

$$H_l(z) = (1 + z^{-1})^N R(z),$$

where  $R(z)$  is a spectral factor of the Lagrange interpolator  $P(z) = 2R(z) * R(z^{-1})$ . All subsequent filters of the two-channel filterbank are derived as follows:

$$H_h(z) = H_l((-z)^{-1})$$

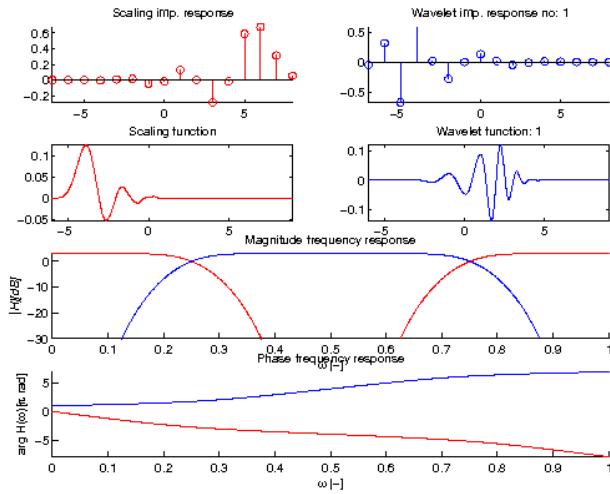
$$G_l(z) = H_l(z^{-1})$$

$$G_h(z) = -H_l(-z)$$

making them an orthogonal perfect-reconstruction QMF.

**Examples:**

```
wfiltinfo('db8');
```



**References:** [22]

**4.8.5 WFILT\_DDEN - Double-DENsity DWT filters (tight frame)****Usage**

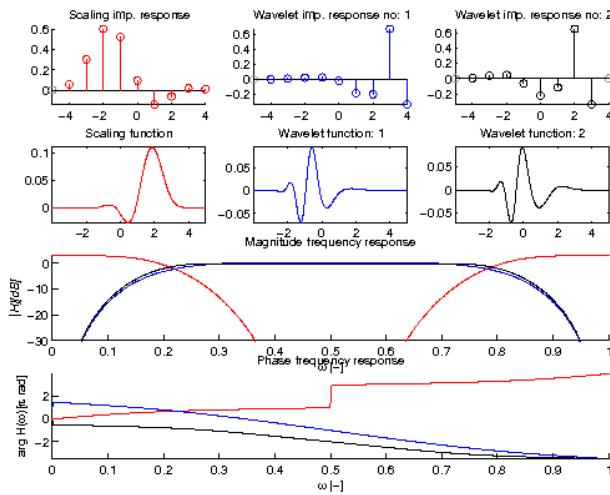
```
[h, g, a] = wfilt_dden(N);
```

**Description**

$[h, g, a] = \text{wfilt\_dden}(N)$  computes oversampled dyadic double-density DWT filters. The redundancy of the basic filterbank is equal to 1.5.

**Examples:**

```
wfiltinfo('dden5');
```



**References:** [65]

#### 4.8.6 WFILT\_DGRID - Dense GRID framelets (tight frame, symmetric)

##### Usage

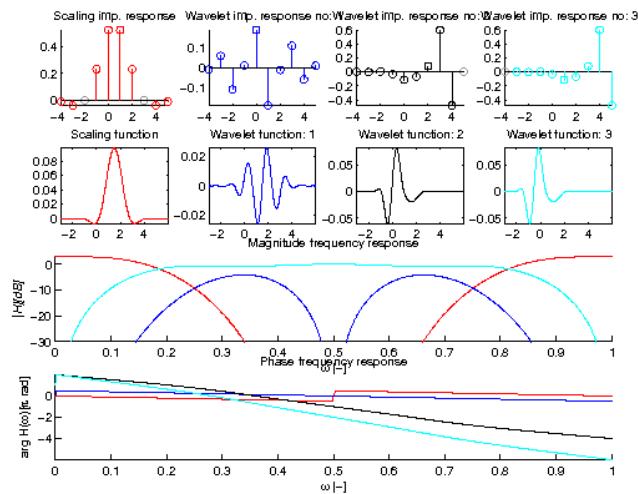
```
[h, g, a] = wfilt_dgrid(N);
```

##### Description

[h, g, a]=wfilt\_dgrid(N) computes Dense GRID framelets. Redundancy equal to 2.

##### Examples:

```
wfiltinfo('dgrid3');
```



##### References: [1]

#### 4.8.7 WFILT\_HDEN - Higher DENsity dwt filters (tight frame, frame)

##### Usage

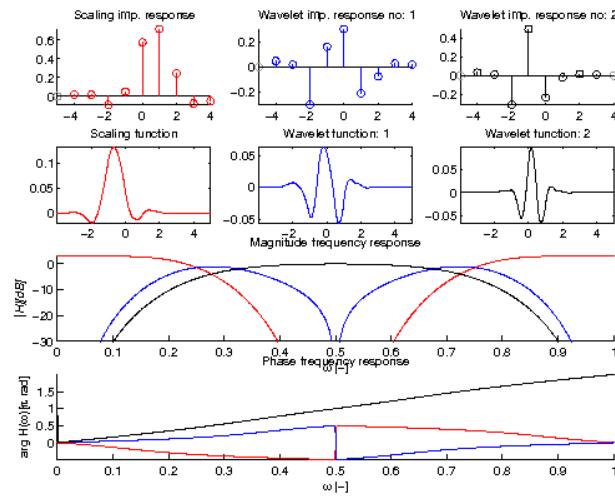
```
[h, g, a] = wfilt_hden(K);
```

##### Description

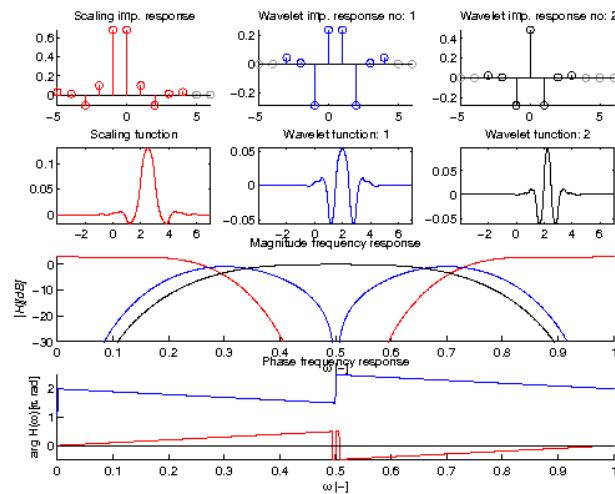
[h, g, a]=wfilt\_hden(K) with  $K \in \{1, 2, 3, 4\}$  returns Higher DENsity dwt filters (tight frame, frame) from the reference. The filterbanks have 3 channels and unusual non-uniform subsampling factors  $[2, 2, 1]$ .

##### Examples:

```
wfiltinfo('hdens');
```



```
wfiltinfo('ana:hden4');
```



**References:** [74]

#### 4.8.8 WFILT\_LEMARIE - Battle and Lemarie filters

##### Usage

```
[h, g, a]=wfilt_lemarie(N)
```

##### Input parameters

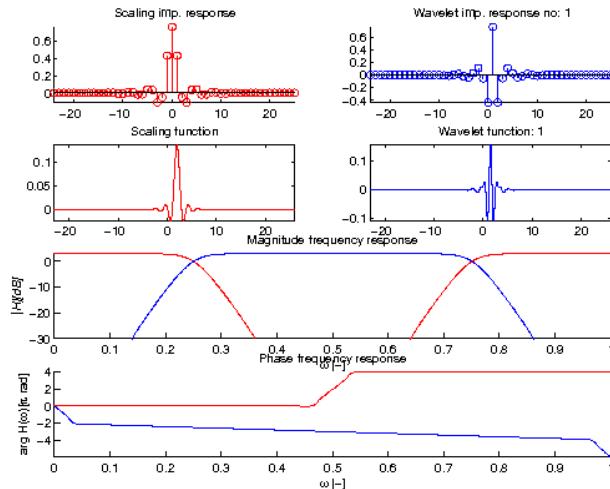
**N** Filter length, must be even.

##### Description

`[h, g, a]=wfilt_lemarie(N)` calculates  $N$  (even) truncated coefficients of orthonormal Battle-Lemarie wavelets. Filter coefficients are obtained by frequency domain sampling and truncating the impulse response. Due to the truncation, the filterbank might not achieve a perfect reconstruction. The filters are included nevertheless since they were the original ones used in the first MRA paper.

**Examples:**

```
wfiltinfo('lemarie50');
```



**References:** [57]

### 4.8.9 WFILT\_MATLABWRAPPER - Wrapper of the Matlab Wavelet Toolbox wfilters function

**Usage**

```
[h, g, a] = wfilt_matlabwrapper(wname);
```

**Description**

`[h, g, a]=wfilt_matlabwrapper(wname)` calls Matlab Wavelet Toolbox function `wfilters` and passes the parameter `wname` to it.

This function requires the Matlab Wavelet Toolbox.

### 4.8.10 WFILT\_MBAND - Generates 4-band coder

**Usage**

```
[h, g, a] = wfilt_mband(N);
```

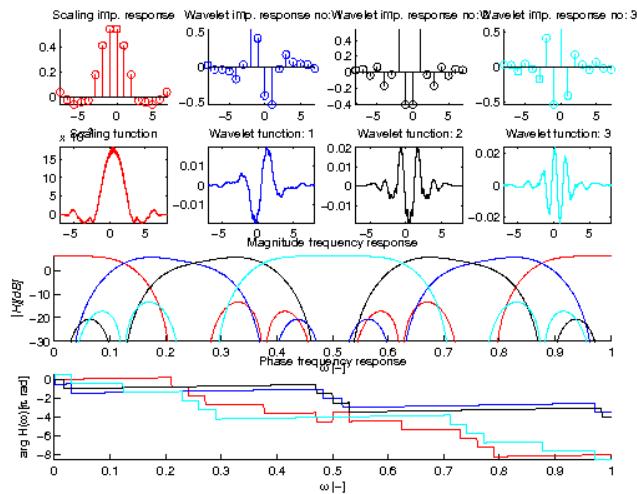
**Description**

`[h, g, a]=wfilt_mband(1)` returns linear-phase 4-band filters from the reference.

The filters are not actually proper wavelet filters, because the scaling filter is not regular, therefore it is not stable under iterations (does not converge to a scaling function).

**Examples:**

```
wfiltinfo('mband1');
```



**References:** [7]

#### 4.8.11 WFILT\_REMEZ - Filters designed using Remez exchange algorithm

##### Usage

```
[h, g, a] = wfilt_remez(L, K, B)
```

##### Input parameters

**L** Length of the filters.

**K** Degree of flatness (regularity) at  $z = -1$ .

**B** Normalized transition bandwidth.

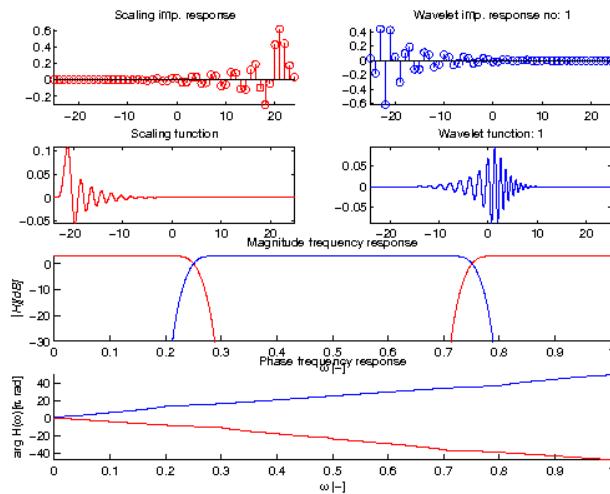
##### Description

[h, g, a] = wfilt\_remez(L, K, B) calculates a set of wavelet filters. Regularity, frequency selectivity, and length of the filters can be controlled by  $K$ ,  $B$  and  $L$  parameters respectively.

The filter design algorithm is based on a Remez algorithm and a factorization of the complex cepstrum of the polynomial.

##### Examples:

```
wfiltinfo('remez50:2:0.1');
```



**References:** [71]

#### 4.8.12 WFILT\_SYMDS - Symmetric wavelets dyadic sibling

##### Usage

```
[h, g, a] = wfilt_symds (K);
```

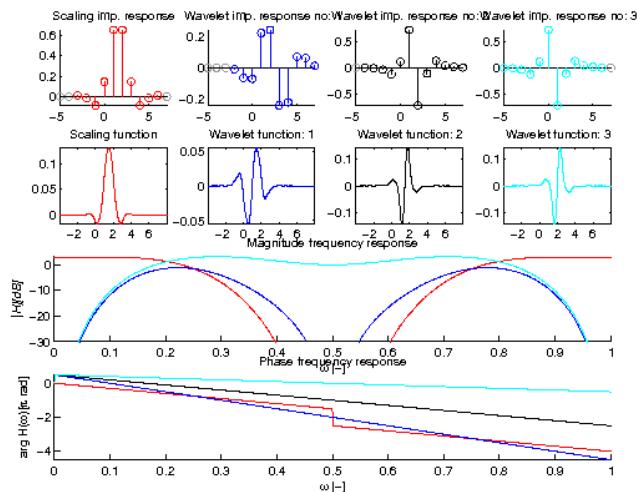
##### Description

[h, g, a]=wfilt\_symds (K) with  $K \in 1, 2, 3, 4, 5$  returns symmetric dyadic sibling wavelet frame filters from the reference.

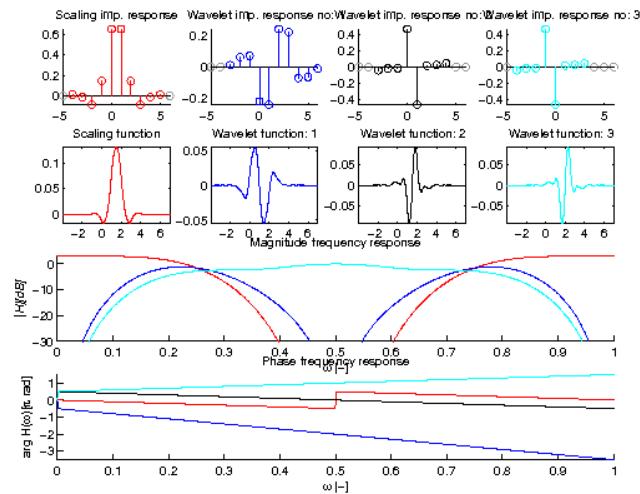
The returned filterbank has redundancy equal to 2 and it does not form a tight frame.

##### Examples:

```
wfiltinfo('ana:symds3');
```



```
wfiltinfo('syn:symds3');
```



**References:** [3]

#### 4.8.13 WFILT\_SPLINE - Birthogonal spline wavelets

##### Usage

```
[h, g, a]=wfilt_spline(m, n);
```

##### Input parameters

**m** Number of zeros at  $z = -1$  of the lowpass filter in  $g\{1\}$

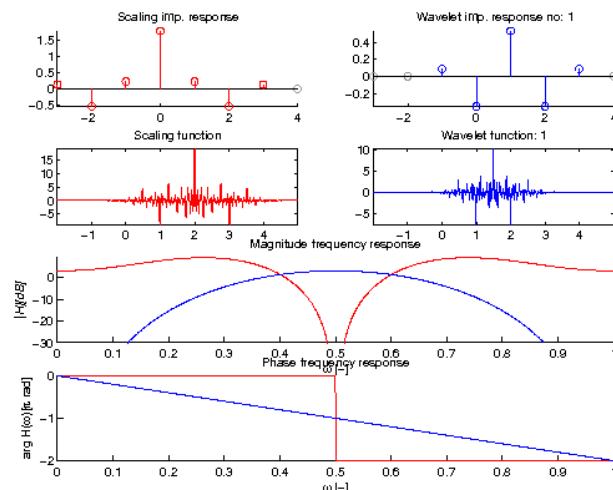
**n** Number of zeros at  $z = -1$  of the lowpass filter in  $h\{1\}$

##### Description

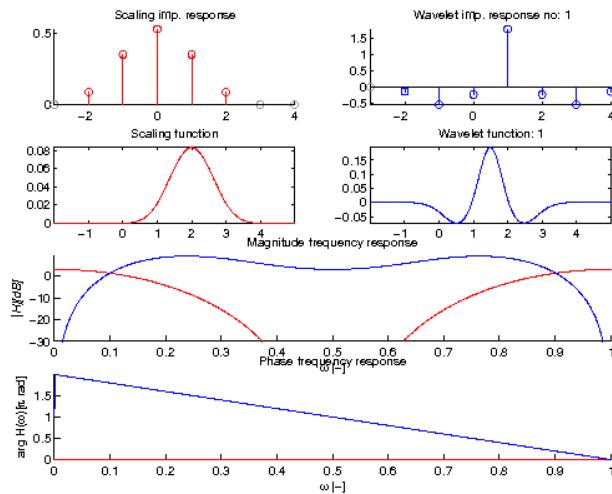
`[h, g, a]=wfilt_spline(m, n)` with  $m+n$  being even returns biorthogonal spline wavelet filters.

##### Examples:

```
wfiltinfo('ana:spline4:2');
```



```
wfiltinfo('syn:spline4:2');
```



#### 4.8.14 WFILT\_SYM - Symlet filters

##### Usage

```
[h, g, a] = wfilt_sym(N);
```

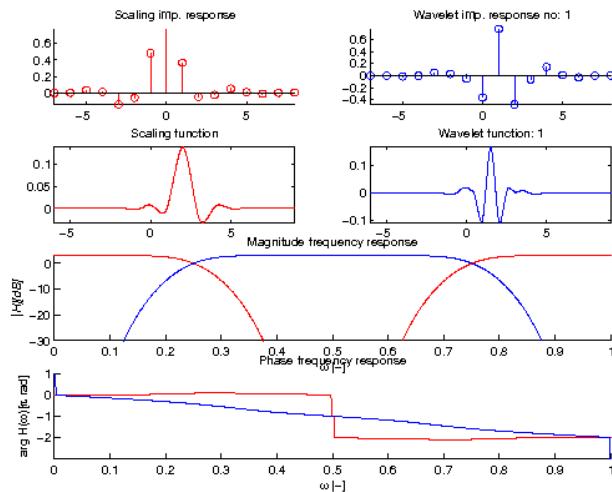
##### Description

[h, g, a] = wfilt\_sym(N) generates the "least asymmetric" Daubechies' orthogonal wavelets or "symlets" with N vanishing moments and length 2N. Zeros of the trigonometrical polynomial the filters consist of in the Z-plane are selected alternatingly inside and outside the unit circle.

Remark: Filters generated by this routine differ slightly from the ones in the reference (table 6.3, figure. 6.4) because of the ambiguity in the algorithm.

##### Examples:

```
wfiltinfo('sym8');
```



**References:** [22]

### 4.8.15 WFILT\_SYMDDEN - Symmetric Double-Density DWT filters (tight frame)

#### Usage

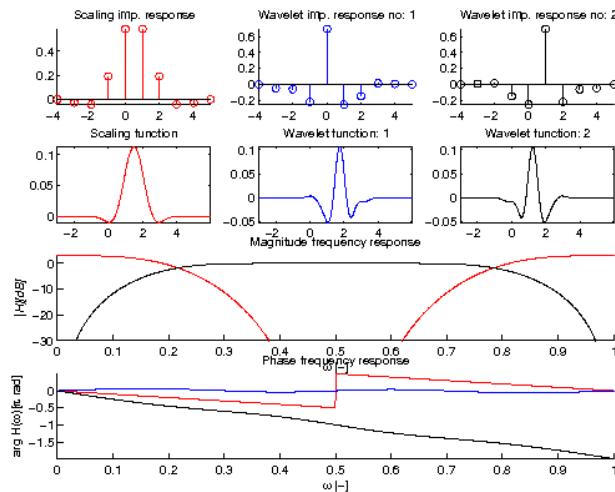
```
[h, g, a] = wfilt_symdden(K);
```

#### Description

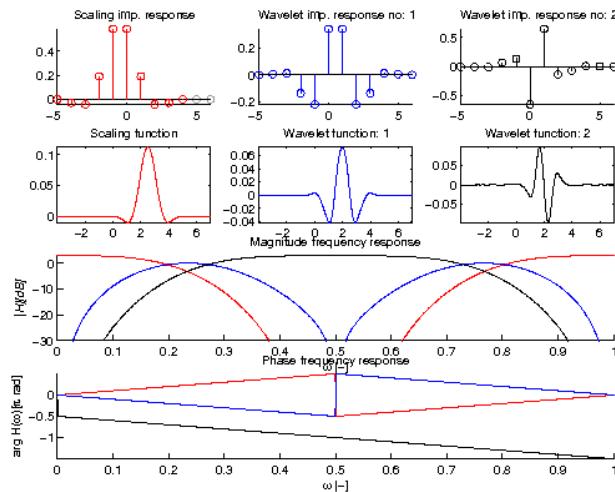
$[h, g, a] = \text{wfilt\_symdden}(K)$  with  $K \in 1, 2$  returns oversampled symmetric double-density DWT filters. The redundancy of the basic filterbank is equal to 1.5.

#### Examples:

```
wfiltinfo('symdden1');
```



```
wfiltinfo('symdden2');
```



**References:** [76]

### 4.8.16 WFILT\_SYMORTH - Symmetric nearly-orthogonal and orthogonal nearly-symmetric

#### Usage

```
[h, g, a] = wfilt_symorth(N);
```

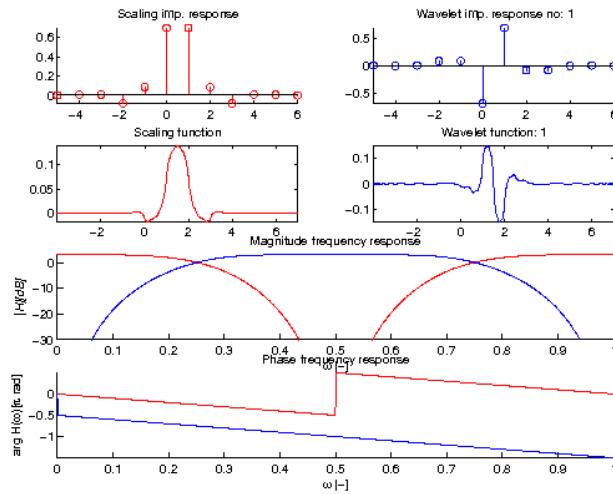
#### Description

[h, g, a]=wfilt\_symorth(N) with  $N \in 1, 2, 3$  returns orthogonal near-symmetric ( $N == 1$ ) and symmetric near-orthogonal ( $N == [2, 3]$ ) wavelet filters from the reference.

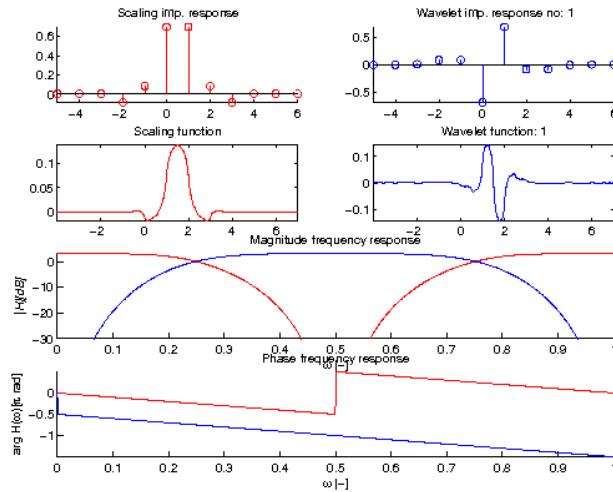
The filters exhibit a coiflet-like behavior i.e. the scaling filter has vanishing moments too.

#### Examples:

```
wfiltinfo('ana:symorth2');
```



```
wfiltinfo('syn:symorth2');
```



#### References: [4]

### 4.8.17 WFILT\_SYMTIGHT - Symmetric Nearly Shift-Invariant Tight Frame Wavelets

#### Usage

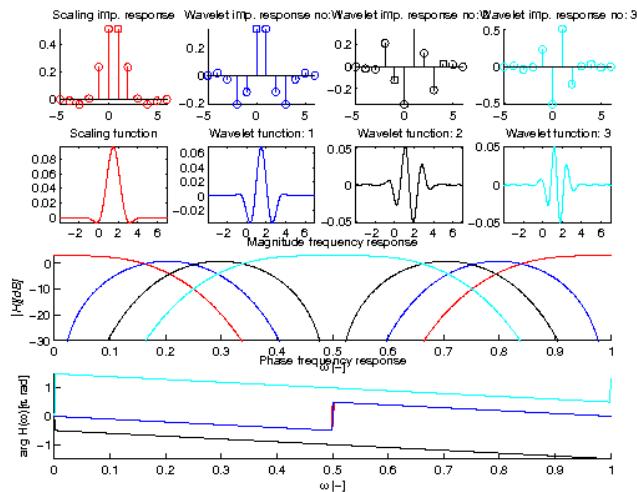
```
[h, g, a] = wfilt_symtight(K);
```

#### Description

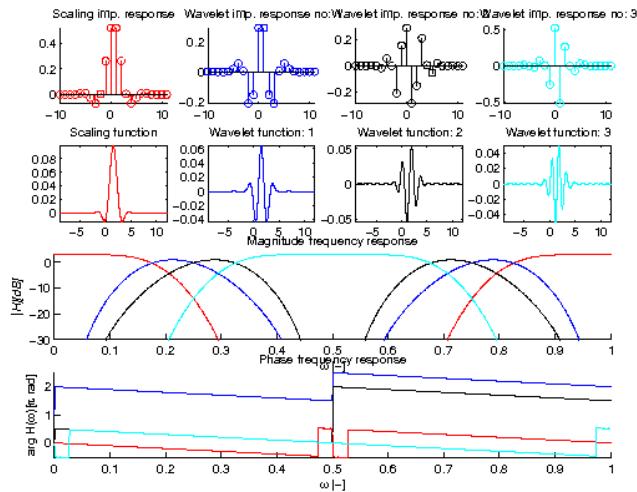
[h, g, a]=wfilt\_symtight(K) with  $K \in 1, 2$  returns 4-band symmetric nearly shift-invariant tight framelets.

#### Examples:

```
wfiltinfo('symtight1');
```



```
wfiltinfo('symtight2');
```



#### References: [2]

### 4.8.18 WFILT\_QSHIFTA - Improved Orthogonality and Symmetry properties

#### Usage

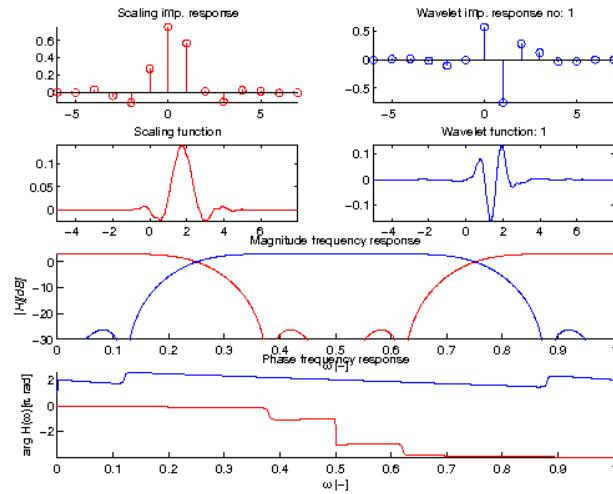
```
[h, g, a] = wfilt_qshifta(N);
```

### Description

`[h, g, a]=wfilt_qshift(N)` with  $N \in 1, 2, 3, 4, 5, 6, 7$  returns Kingsbury's Q-shift wavelet filters for tree A.

### Examples:

```
figure(1);
wfiltinfo('qshifta3');
```



References: [46], [45]

### 4.8.19 WFILT\_QSHIFTB - Improved Orthogonality and Symmetry properties

#### Usage

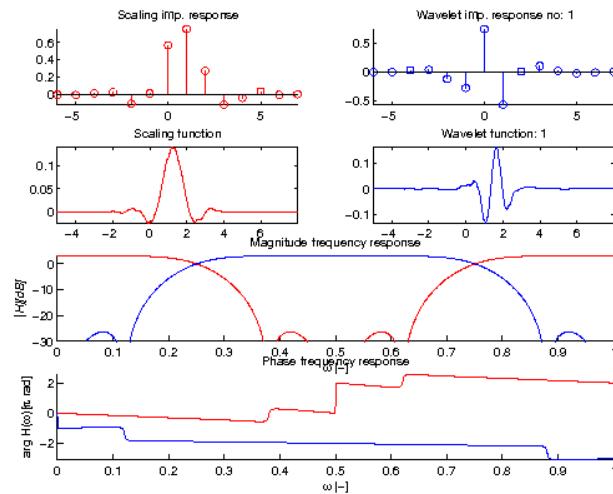
```
[h, g, a] = wfilt_qshiftb(N);
```

### Description

`[h, g, a]=wfilt_qshiftb(N)` with  $N \in 1, 2, 3, 4, 5, 6, 7$  returns Kingsbury's Q-shift wavelet filters for tree B.

### Examples:

```
figure(1);
wfiltinfo('qshiftb3');
```



**References:** [46], [45]

#### 4.8.20 WFILT\_ODDEVENA - Kingsbury's symmetric even filters

##### Usage

```
[h, g, a] = wfilt_oddevena (N);
```

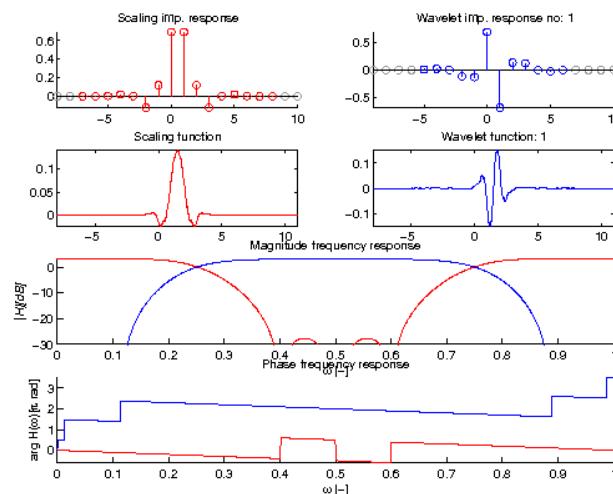
##### Description

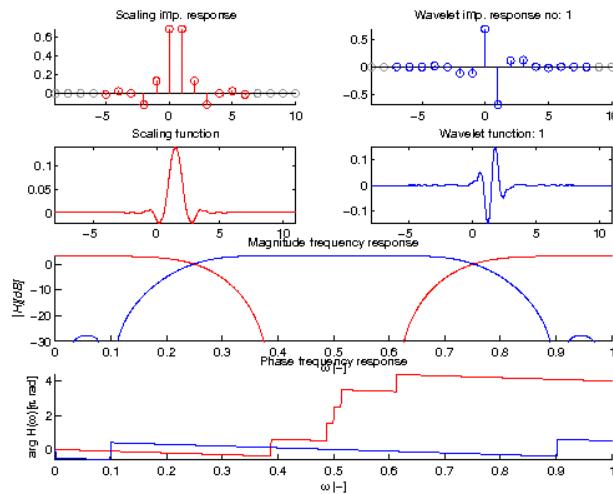
[h, g, a]=wfilt\_oddevena (N) with  $N \in \mathbb{N}$  returns Kingsbury's even filters.

##### Examples:

```
figure(1);
wfiltinfo('ana:oddevenal');

figure(2);
wfiltinfo('syn:oddevenal');
```





References: [44]

#### 4.8.21 WFILT\_ODDEVENB - Kingsbury's symmetric odd filters

##### Usage

```
[h, g, a] = wfilt_oddevenb(N);
```

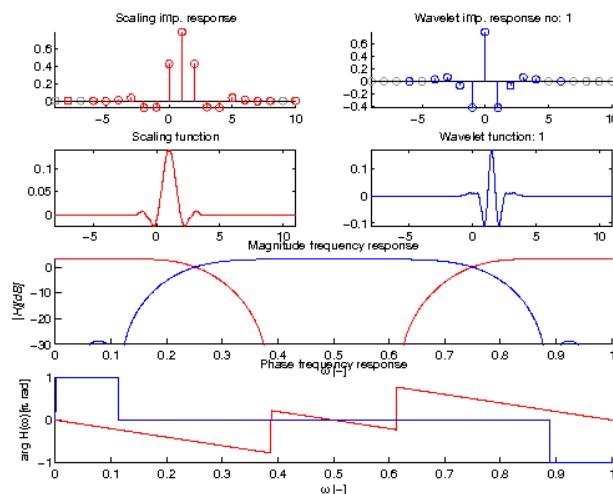
##### Description

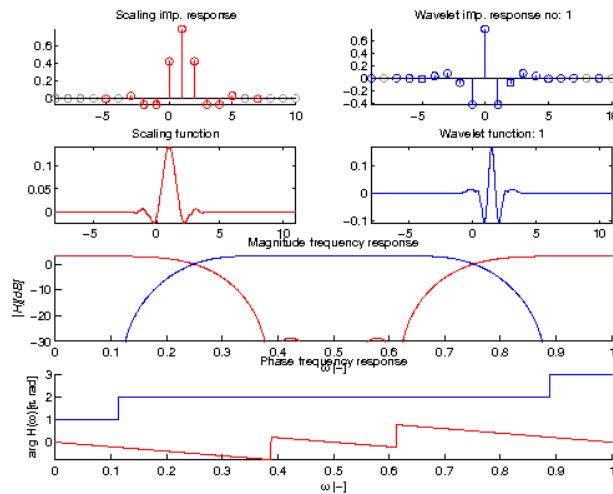
[h, g, a]=wfilt\_oddevenb(N) with  $N \in \mathbb{N}$  returns Kingsbury's odd filters.

##### Examples:

```
figure(1);
wfiltinfo('ana:oddevenb1');

figure(2);
wfiltinfo('syn:oddevenb1');
```





References: [44]

## 4.9 Dual-Tree Filters

### 4.9.1 WFILTDT\_QSHIFT - Improved Orthogonality and Symmetry properties

#### Usage

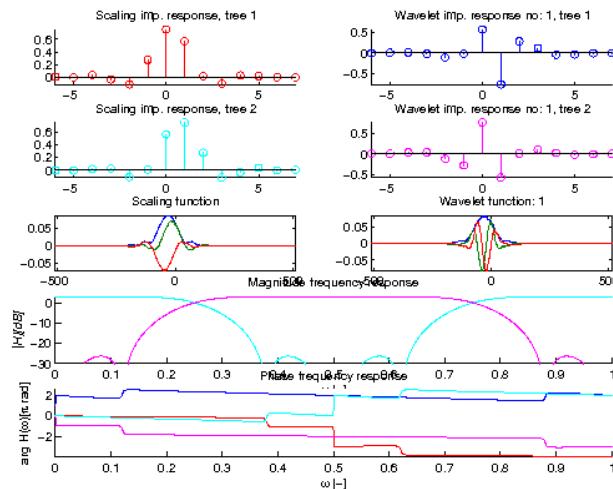
```
[h, g, a] = wfilt dt_qshift (N);
```

#### Description

$[h, g, a] = \text{wfilt dt\_qshift}(N)$  with  $N \in 1, 2, 3, 4, 5, 6, 7$  returns Kingsbury's Q-shift filters suitable for dual-tree complex wavelet transform. Filters in both trees are orthogonal and based on a single prototype low-pass filter with a quarter sample delay. Other filters are derived by modulation and time reversal such that they fulfil the half-sample delay difference between the trees.

#### Examples:

```
wfilt dtinfo ('qshift3');
```



References: [46], [45]

### 4.9.2 WFILTDAT\_OPTSYM - Optimizatized Symmetric Self-Hilbertian Filters

#### Usage

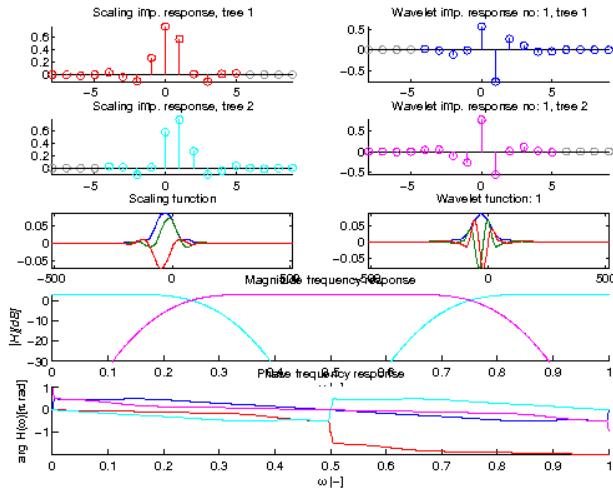
```
[h, g, a] = wfiltadt_optsym(N);
```

#### Description

$[h, g, a] = wfiltadt\_optsym(N)$  with  $N \in 1, 2, 3$  returns filters suitable for dual-tree complex wavelet transform with optimized symmetry.

#### Examples:

```
wfiltadtinfo('optsym3');
```



References: [26]

### 4.9.3 WFILTDAT\_ODDEVEN - Kingsbury's symmetric odd and even filters

#### Usage

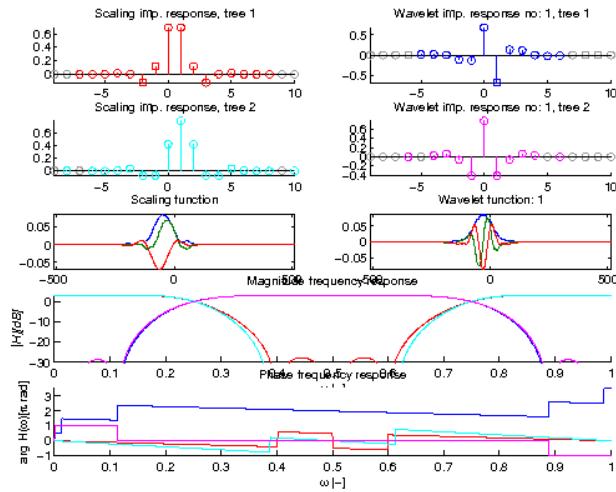
```
[h, g, a] = wfiltadt_oddeven(N);
```

#### Description

$[h, g, a] = wfilt\_oddeven(N)$  with  $N \in 1$  returns the original odd and even symmetric filters suitable for dual-tree complex wavelet transform. The filters in individual trees are biorthogonal.

#### Examples:

```
wfiltadtinfo('ana:oddeven1');
```



References: [44]

#### 4.9.4 WFILTDT\_DDEN - Double-Density Dual-Tree DWT filters

##### Usage

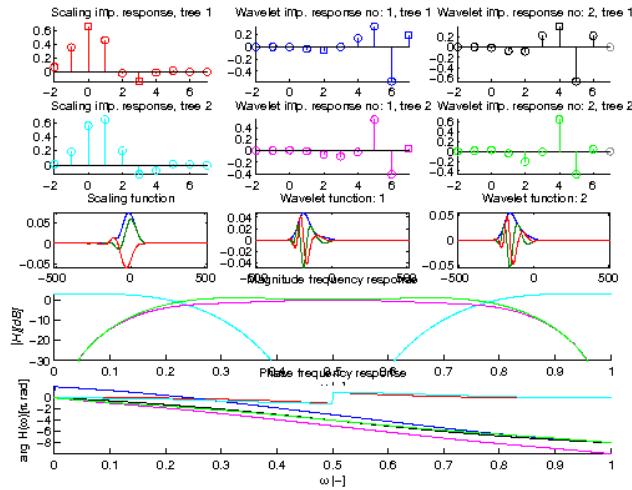
```
[h, g, a] = wfiltddt_dden(N);
```

##### Description

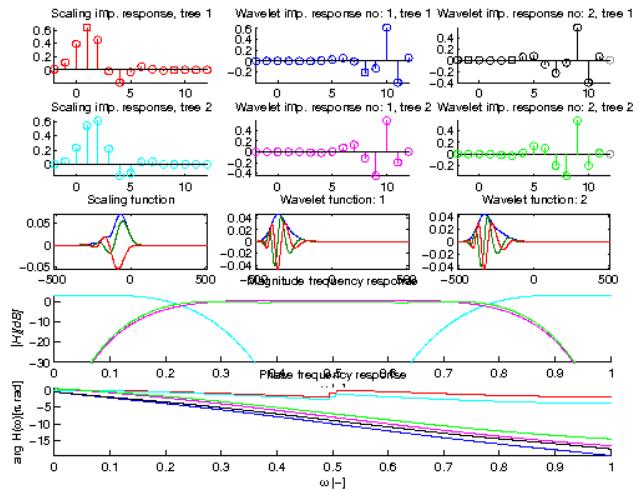
`[h, g, a]=wfiltddt_dden(N)` with  $N \in 1, 2$  returns filters suitable for dual-tree double density complex wavelet transform.

##### Examples:

```
wfiltddtinfo('dden1');
```



```
wfiltddtinfo('dden2');
```



**References:** [73]

# Chapter 5

## LTFAT - Filterbanks

### 5.1 Transforms and basic routines

#### 5.1.1 FILTERBANK - Apply filterbank

##### Usage

```
c=filterbank(f,g,a);
```

##### Description

`filterbank(f,g,a)` applies the filters given in  $g$  to the signal  $f$ . Each subband will be subsampled by a factor of  $a$  (the hop-size). In contrast to `ufilterbank`,  $a$  can be a vector so the hop-size can be channel-dependant. If  $f$  is a matrix, the transformation is applied to each column.

The filters  $g$  must be a cell-array, where each entry in the cell array corresponds to an FIR filter.

The output coefficients are stored a cell array. More precisely, the  $n$ 'th cell of  $c$ ,  $c\{m\}$ , is a 2D matrix of size  $M(n) \times W$  and containing the output from the  $m$ 'th channel subsampled at a rate of  $a(m)$ .  $c\{m\}(n,l)$  is thus the value of the coefficient for time index  $n$ , frequency index  $m$  and signal channel  $l$ .

The coefficients  $c$  computed from the signal  $f$  and the filterbank with windows  $g\_m$  are defined by

$$c_m(n+1) = \sum_{l=0}^{L-1} f(l+1) g(a_m n - l + 1)$$

where  $an - l$  is computed modulo  $L$ .

**References:** [15]

#### 5.1.2 UFILTERBANK - Apply Uniform filterbank

##### Usage

```
c=ufilterbank(f,g,a);
```

##### Description

`ufilterbank(f,g,a)` applies the filter given in  $g$  to the signal  $f$ . Each subband will be subsampled by a factor of  $a$  (the hop-size). If  $f$  is a matrix, the transformation is applied to each column.

The filters  $g$  must be a cell-array, where each entry in the cell array corresponds to a filter.

If  $f$  is a single vector, then the output will be a matrix, where each column in  $f$  is filtered by the corresponding filter in  $g$ . If  $f$  is a matrix, the output will be 3-dimensional, and the third dimension will correspond to the columns of the input signal.

The coefficients  $c$  computed from the signal  $f$  and the filterbank with windows  $g\_m$  are defined by

$$c(n+1,m+1) = \sum_{l=0}^{L-1} f(l+1) g(an - l + 1)$$

### 5.1.3 IFILTERBANK - Filter bank inversion

#### Usage

```
f = ifilterbank(c, g, a);
```

#### Description

`ifilterbank(c, g, a)` synthesizes a signal  $f$  from the coefficients  $c$  using the filters stored in  $g$  for a channel subsampling rate of  $a$  (the hop-size). The coefficients has to be in the format returned by either filterbank or ufilterbank.

The filter format for  $g$  is the same as for filterbank.

If perfect reconstruction is desired, the filters must be the duals of the filters used to generate the coefficients. See the help on filterbankdual.

**References:** [15]

### 5.1.4 FILTERBANKWIN - Compute set of filter bank windows from text or cell array

#### Usage

```
[g, info] = filterbankwin(g, a, L);
```

#### Description

`[g, info] = filterbankwin(g, a, L)` computes a window that fits well with time shift  $a$  and transform length  $L$ . The window itself is as a cell array containing additional parameters.

The window can be specified directly as a cell array of vectors of numerical values. In this case, filterbankwin only checks assumptions about transform sizes etc.

`[g, info] = filterbankwin(g, a)` does the same, but the windows must be FIR windows, as the transform length is unspecified.

`filterbankwin(..., 'normal')` computes a window for regular filterbanks, while `filterbankwin(..., 'real')` does the same for the positive-frequency only filterbanks.

The window can also be specified as cell array. The possibilities are:

- {'dual', ...} Canonical dual window of whatever follows. See the examples below.
- {'realdual', ...} Canonical dual window for a positive-frequency filterbank of whatever follows. See the examples below.
- {'tight', ...} Canonical tight window of whatever follows. See the examples below.
- {'realtight', ...} Canonical tight window for a real-valued for a positive frequency filterbank of whatever follows.

The structure `info` provides some information about the computed window:

- `info.M` Number of windows (equal to the number of channels)
- `info.longestfilter` Length of the longest filter
- `info.gauss` True if the windows are Gaussian.
- `info.tfr` Time/frequency support ratios of the window. Set whenever it makes sense.
- `info.isfir` Input is an FIR window
- `info.isdual` Output is the dual window of the auxiliary window.
- `info.istight` Output is known to be a tight window.
- `info.auxinfo` Info about auxiliary window.
- `info.g1` Length of windows.
- `info.isfac` True if the frame generated by the window has a fast factorization.

### 5.1.5 FILTERBANKLENGTH - Filterbank length from signal

#### Usage

```
L=filterbanklength(Ls,a);
```

#### Description

`filterbanklength(Ls,a)` returns the length of a filterbank with time shifts  $a$ , such that it is long enough to expand a signal of length  $L_s$ .

If the filterbank length is longer than the signal length, the signal will be zero-padded by filterbank or `ufilterbank`.

If instead a set of coefficients are given, call `filterbanklengthcoef`.

### 5.1.6 FILTERBANKLENGTHCOEF - Filterbank length from coefficients

#### Usage

```
L=filterbanklengthcoef(coef,a);
```

#### Description

`filterbanklengthcoef(coef,a)` returns the length of a filterbank with time-shifts  $a$ , such that the filterbank is long enough to expand the coefficients  $coef$ .

If instead a signal is given, call `filterbanklength`.

## 5.2 Auditory inspired filterbanks

### 5.2.1 CQT - Constant-Q nonstationary Gabor filterbank

#### Usage

```
[c,Ls,g,shift,M] = cqt(f,fmin,fmax,bins,fs,M)
[c,Ls,g,shift,M] = cqt(f,fmin,fmax,bins,fs)
[c,Ls,g,shift] = cqt(...)
[c,Ls] = cqt(...)
c = cqt(...)
```

#### Input parameters

**f** The signal to be analyzed (For multichannel signals, input should be a matrix which each column storing a channel of the signal).

**fmin** Minimum frequency (in Hz)

**fmax** Maximum frequency (in Hz)

**bins** Vector consisting of the number of bins per octave

**fs** Sampling rate (in Hz)

**M** Number of time channels (optional) If M is constant, the output is converted to a matrix

## Output parameters

<b>c</b>	Transform coefficients (matrix or cell array)
<b>Ls</b>	Original signal length (in samples)
<b>g</b>	Cell array of Fourier transforms of the analysis windows
<b>shift</b>	Vector of frequency shifts
<b>M</b>	Number of time channels

## Description

This function computes a constant-Q transform via nonstationary Gabor filterbanks. Given the signal  $f$ , the constant-Q parameters  $f_{min}$ ,  $f_{max}$  and  $bins$ , as well as the sampling rate  $fs$  of  $f$ , the corresponding constant-Q coefficients  $c$  are given as output. For reconstruction, the length of  $f$  and the filterbank parameters can be returned also.

The transform produces phase-locked coefficients in the sense that each filter is considered to be centered at 0 and the signal itself is modulated accordingly.

Optional input arguments can be supplied like this:

```
cqt(f,fmin,fmax,bins,fs,'min_win',min_win)
```

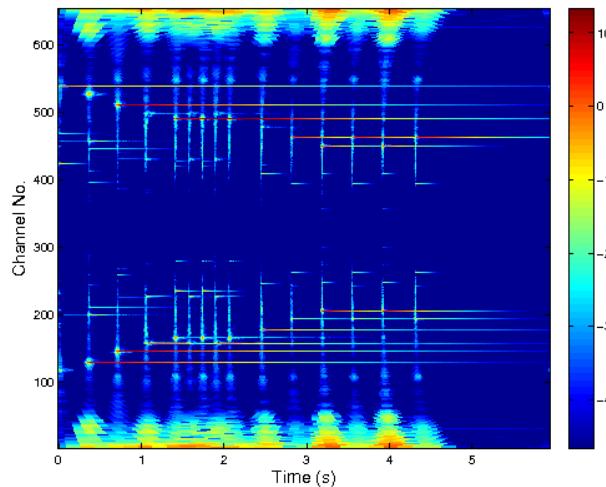
The arguments must be character strings followed by an argument:

<b>'min_win'</b> , <b>min_win</b>	Minimum admissible window length (in samples)
<b>'Qvar'</b> , <b>Qvar</b>	Bandwidth variation factor
<b>'M_fac'</b> , <b>M_fac</b>	Number of time channels are rounded to multiples of this
<b>'wfun'</b> , <b>wfun</b>	Filter prototype (see firwin for available filters)
<b>'fractional'</b>	Allow fractional shifts and bandwidths

## Example:

The following example shows analysis and synthesis with cqt and icqt:

```
[f,fs] = gspi;
fmin = 200;
fmax = fs/2;
[c,Ls,g,shift,M] = cqt(f,fmin,fmax,48,fs);
fr = icqt(c,g,shift,Ls);
rel_err = norm(f-fr)/norm(f);
plotfilterbank(c,Ls./M,[],fs,'dynrange',60);
```



**References:** [84], [39]

### 5.2.2 ICQT - Constant-Q nonstationary Gabor synthesis

#### Usage

```
fr = icqt(c,g,shift,Ls,dual)
fr = icqt(c,g,shift,Ls)
fr = icqt(c,g,shift)
```

#### Input parameters

<b>c</b>	Transform coefficients (matrix or cell array)
<b>g</b>	Cell array of Fourier transforms of the analysis windows
<b>shift</b>	Vector of frequency shifts
<b>Ls</b>	Original signal length (in samples)
<b>dual</b>	Synthesize with the dual frame

#### Output parameters

<b>fr</b>	Synthesized signal (Channels are stored in the columns)
-----------	---

#### Description

Given the cell array  $c$  of non-stationary Gabor coefficients, and a set of filters  $g$  and frequency shifts  $shift$  this function computes the corresponding constant-Q synthesis.

If  $dual$  is set to 1 (default), an attempt is made to compute the canonical dual frame for the system given by  $g$ ,  $shift$  and the size of the vectors in  $c$ . This provides perfect reconstruction in the painless case, see the references for more information.

This is just a dummy routine calling insdgbf.

**References:** [84], [39]

### 5.2.3 ERBLETT - ERBlet nonstationary Gabor filterbank

#### Usage

```
[c,Ls,g,shift,M] = erblett(f,bins,fs,varargin)
[c,Ls,g,shift] = erblett(...)
```

```
[c,Ls] = erblett(...)
c = erblett(...)
```

### Input parameters

<b>f</b>	The signal to be analyzed (For multichannel signals, input should be a matrix which each column storing a channel of the signal)
<b>bins</b>	Desired bins per ERB
<b>fs</b>	Sampling rate of f (in Hz)
<b>varargin</b>	Optional input pairs (see table below)

### Output parameters

<b>c</b>	Transform coefficients (matrix or cell array)
<b>Ls</b>	Original signal length (in samples)
<b>g</b>	Cell array of Fourier transforms of the analysis windows
<b>shift</b>	Vector of frequency shifts
<b>M</b>	Number of time channels

### Description

This function computes an ERBlet constant-Q transform via nonstationary Gabor filterbanks. Given the signal  $f$ , the ERBlet parameter  $bins$ , as well as the sampling rate  $fs$  of  $f$ , the corresponding ERBlet coefficients  $c$  are given as output. For reconstruction, the length of  $f$  and the filterbank parameters can be returned also.

The transform produces phase-locked coefficients in the sense that each filter is considered to be centered at 0 and the signal itself is modulated accordingly.

Optional input arguments can be supplied like this:

```
erblett(f,bins,fs,'Qvar',Qvar)
```

The arguments must be character strings followed by an argument:

<b>'Qvar',Qvar</b>	Bandwidth variation factor
<b>'M_fac',M_fac</b>	Number of time channels are rounded to multiples of this
<b>'wifun',wifun</b>	Filter prototype (see firwin for available filters)

### Examples:

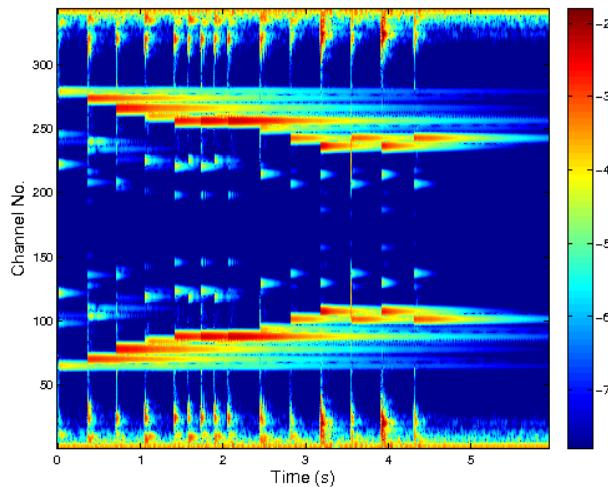
The following example shows analysis and synthesis with erblett and ierblett:

```
[f,fs] = gspi;
binsPerERB = 4;
[c,Ls,g,shift,M] = erblett(f,binsPerERB,fs);
fr = ierblett(c,g,shift,Ls);
rel_err = norm(f-fr)/norm(f)
plotfilterbank(c,Ls./M,[],fs,'dynrange',60);
```

*This code produces the following output:*

```
rel_err =
```

```
4.9941e-16
```



**References:** [60]

#### 5.2.4 IERBLETT - ERBlet nonstationary Gabor synthesis

##### Usage

```
fr = ierblett(c,g,shift,Ls,dual)
fr = ierblett(c,g,shift,Ls)
fr = ierblett(c,g,shift)
```

##### Input parameters

<b>c</b>	Transform coefficients (matrix or cell array)
<b>g</b>	Cell array of Fourier transforms of the analysis windows
<b>shift</b>	Vector of frequency shifts
<b>Ls</b>	Original signal length (in samples)
<b>dual</b>	Synthesize with the dual frame

##### Output parameters

<b>fr</b>	Synthesized signal (Channels are stored in the columns)
-----------	---

##### Description

Given the cell array *c* of non-stationary Gabor coefficients, and a set of filters *g* and frequency shifts *shift* this function computes the corresponding ERBlet synthesis.

If *dual* is set to 1 (default), an attempt is made to compute the canonical dual frame for the system given by *g*, *shift* and the size of the vectors in *c*. This provides perfect reconstruction in the painless case, see the references for more information.

This is just a dummy routine calling insdgfb.

**References:** [60]

#### 5.2.5 INSDGFB - Nonstationary Gabor filterbank synthesis

##### Usage

```
fr = insdgfb(c,g,shift,Ls,dual)
fr = insdgfb(c,g,shift,Ls)
fr = insdgfb(c,g,shift)
```

**Input parameters**

<b>c</b>	Transform coefficients (matrix or cell array)
<b>g</b>	Cell array of Fourier transforms of the analysis windows
<b>shift</b>	Vector of frequency shifts
<b>Ls</b>	Original signal length (in samples)
<b>dual</b>	Synthesize with the dual frame

**Output parameters**

<b>fr</b>	Synthesized signal (Channels are stored in the columns)
-----------	---

**Description**

Given the cell array  $c$  of non-stationary Gabor coefficients, and a set of filters  $g$  and frequency shifts  $shift$  this function computes the corresponding nonstationary Gabor filterbank synthesis.

If  $dual$  is set to 1 (default), an attempt is made to compute the canonical dual frame for the system given by  $g$ ,  $shift$  and the size of the vectors in  $c$ . This provides perfect reconstruction in the painless case, see the references for more information.

**References:** [9], [39]

## 5.3 Filter generators

### 5.3.1 CQTFILTERS - CQT-spaced filters

**Usage**

```
[g, a, fc]=cqtfilters(fs);
[g, a, fc]=cqtfilters(fs, ...);
```

**Input parameters**

<b>fs</b>	Sampling rate (in Hz).
<b>fmin</b>	Minimum frequency (in Hz)
<b>fmax</b>	Maximum frequency (in Hz)
<b>bins</b>	Vector consisting of the number of bins per octave.
<b>Ls</b>	Signal length.

**Output parameters**

<b>g</b>	Cell array of filters.
<b>a</b>	Downsampling rate for each channel.
<b>fc</b>	Center frequency of each channel.
<b>L</b>	Next admissible length suitable for the generated filters.

### Description

`[g, a, fc]=cqtfilters(fs, fmin, fmax, bins, Ls)` constructs a set of band-limited filters  $g$  which cover the required frequency range  $f_{\min}$ - $f_{\max}$  with  $\text{bins}$  filters per octave starting at  $f_{\min}$ . All filters have (approximately) equal  $Q = f_c/f_b$ , hence constant-Q. The remaining frequency intervals not covered by these filters are captured by two additional filters (low-pass, high-pass). The signal length  $L_s$  is mandatory, since we need to avoid too narrow frequency windows.

By default, a Hann window on the frequency side is chosen, but the window can be changed by passing any of the window types from firwin as an optional parameter.

Because the downsampling rates of the channels must all divide the signal length, filterbank will only work for multiples of the least common multiple of the downsampling rates. See the help of filterbanklength.

`[g, a]=cqtfilters(..., 'regsampling')` constructs a non-uniform filterbank. The downsampling rates are constant in the octaves but can differ among octaves. This approach was chosen in order to minimize the least common multiple of  $a$ , which determines a granularity of admissible input signal lengths.

`[g, a]=cqtfilters(..., 'uniform')` constructs a uniform filterbank where the downsampling rate is the same for all the channels. This results in most redundant representation, which produces nice plots.

`[g, a]=cqtfilters(..., 'fractional')` constructs a filterbank with fractional downsampling rates  $a$ . The rates are constructed such that the filterbank can handle signal lengths that are multiples of  $L$ , so the benefit of the fractional downsampling is that you get to choose the value returned by filterbanklength. This results in the least redundant system.

`[g, a]=cqtfilters(..., 'fractionaluniform')` constructs a filterbank with fractional downsampling rates  $a$ , which are uniform for all filters except the "filling" low-pass and high-pass filters can have different fractional downsampling rates. This is useful when uniform subsampling and low redundancy at the same time are desirable.

The filters are intended to work with signals with a sampling rate of  $fs$ .

`cqtfilters` accepts the following optional parameters:

<b>'wifun'</b> , <b>wifun</b>	Filter prototype (see firwin for available filters). Default is ' <code>'hann'</code> '.
<b>'Qvar'</b> , <b>Qvar</b>	Bandwidth variation factor. Multiplies the calculated bandwidth. Default value is <i>1</i> . If the value is less than one, the system may no longer be painless.
<b>'subprec'</b>	Allow subsample window positions and bandwidths to better approximate the constant-Q property.
<b>'complex'</b>	Construct a filterbank that covers the entire frequency range. When missing, only positive frequencies are covered.
<b>'min_win'</b> , <b>min_win</b>	Minimum admissible window length (in samples). Default is <i>4</i> . This restricts the windows not to become too narrow when $L$ is low. This however brakes the constant-Q property for such windows and creates rippling in the overall frequency response.
<b>'redmul'</b> , <b>redmul</b>	Redundancy multiplier. Increasing the value of this will make the system more redundant by lowering the channel downsampling rates. Default value is <i>1</i> . If the value is less than one, the system may no longer be painless.

**References:** [84], [39], [72]

### 5.3.2 ERBFILTERS - ERB-spaced filters

#### Usage

```
[g, a, fc]=erbfilters(fs, Ls);
[g, a, fc]=erbfilters(fs, Ls, ...);
```

**Input parameters**

<b>fs</b>	Sampling rate (in Hz).
<b>Ls</b>	Signal length.

**Output parameters**

<b>g</b>	Cell array of filters.
<b>a</b>	Downsampling rate for each channel.
<b>fc</b>	Center frequency of each channel.
<b>L</b>	Next admissible length suitable for the generated filters.

**Description**

`[g, a, fc]=erbfilters(fs, Ls)` constructs a set of filters  $g$  that are equidistantly spaced on the ERB-scale (see freqtoerb) with bandwidths that are proportional to the width of the auditory filters audfiltbw. The filters are intended to work with signals with a sampling rate of  $fs$ . The signal length  $Ls$  is mandatory, since we need to avoid too narrow frequency windows.

By default, a Hann window on the frequency side is chosen, but the window can be changed by passing any of the window types from firwin as an optional parameter.

Because the downsampling rates of the channels must all divide the signal length, filterbank will only work for multiples of the least common multiple of the downsampling rates. See the help of filterbanklength.

`[g, a]=erbfilters(..., 'regSampling')` constructs a non-uniform filterbank with integer subsampling factors.

`[g, a]=erbfilters(..., 'uniform')` constructs a uniform filterbank where the downsampling rate is the same for all the channels. This results in most redundant representation, which produces nice plots.

`[g, a]=erbfilters(..., 'fractional')` constructs a filterbank with fractional downsampling rates  $a$ . The rates are constructed such that the filterbank can handle signal lengths that are multiples of  $L$ , so the benefit of the fractional downsampling is that you get to choose the value returned by filterbanklength. This results in the least redundant system.

`[g, a]=erbfilters(..., 'fractionaluniform')` constructs a filterbank with fractional downsampling rates  $a$ , which are uniform for all filters except the "filling" low-pass and high-pass filters can have different fractional downsampling rates. This is useful when uniform subsampling and low redundancy at the same time are desirable.

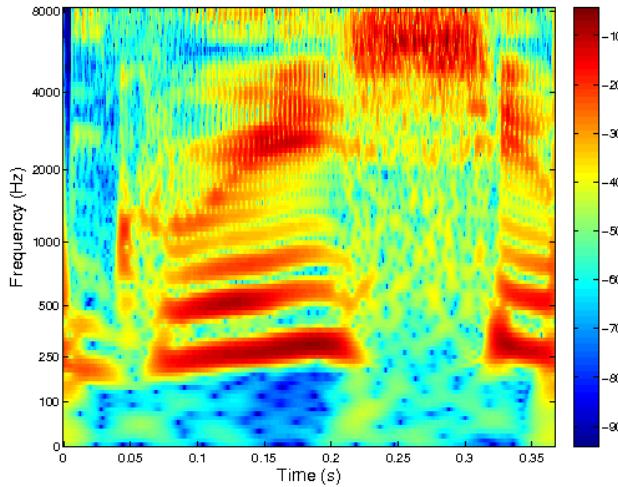
`erbfilters` accepts the following optional parameters:

<b>'spacing',b</b>	Specify the spacing in ERBS between the filters. Default value is $b=1$ .
<b>'M',M</b>	Specify the number of filters, $M$ . If this parameter is specified, it overwrites the ' <code>spacing</code> ' parameter.
<b>'redmul',redmul</b>	Redundancy multiplier. Increasing the value of this will make the system more redundant by lowering the channel downsampling rates. It is only used if the filterbank is a non-uniform filterbank. Default value is $1$ . If the value is less than one, the system may no longer be painless.
<b>'symmetric'</b>	Create filters that are symmetric around their centre frequency. This is the default.
<b>'warped'</b>	Create asymmetric filters that are symmetric on the Erb-scale.
<b>'complex'</b>	Construct a filterbank that covers the entire frequency range.
<b>'bwmul',bwmul</b>	Bandwidth of the filters relative to the bandwidth returned by audfiltbw. Default is $bwmul = 1$ .
<b>'min_win',min_win</b>	Minimum admissible window length (in samples). Default is $4$ . This restrict the windows not to become too narrow when $L$ is low.

**Examples:**

In the first example, we construct a highly redundant uniform filterbank and visualize the result:

```
[f,fs]=greasy; % Get the test signal
[g,a,fc]=erbfilters(fs,length(f),'uniform','M',100);
c=filterbank(f,g,a);
plotfilterbank(c,a,fc,fs,90,'audtck');
```



In the second example, we construct a non-uniform filterbank with fractional sampling that works for this particular signal length, and test the reconstruction. The plot displays the response of the filterbank to verify that the filters are well-behaved both on a normal and an ERB-scale. The second plot shows frequency responses of filters used for analysis (top) and synthesis (bottom).

```
[f,fs]=greasy; % Get the test signal
L=length(f);
[g,a,fc]=erbfilters(fs,L,'fractional');
c=filterbank(f,{ 'realdual' , g },a);
r=2*real(ifilterbank(c,g,a));
norm(f-r)

% Plot the response
figure(1);
subplot(2,1,1);
R=filterbankresponse(g,a,L,fs,'real','plot');

subplot(2,1,2);
semiaudplot(linspace(0,fs/2,L/2+1),R(1:L/2+1));
ylabel('Magnitude');

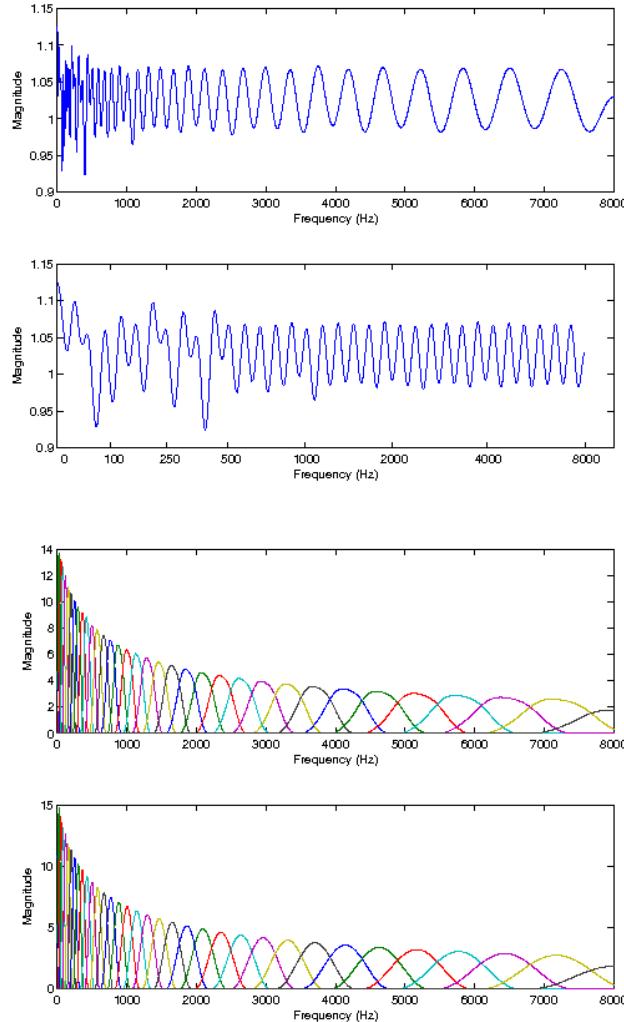
% Plot frequency responses of individual filters
gd=filterbankrealdual(g,a,L);
figure(2);
subplot(2,1,1);
filterbankfreqz(gd,a,L,fs,'plot','linabs','posfreq');

subplot(2,1,2);
filterbankfreqz(g,a,L,fs,'plot','linabs','posfreq');
```

*This code produces the following output:*

```
ans =
```

$8.1396e-15$



**References:** [60]

## 5.4 Window construction and bounds

### 5.4.1 FILTERBANKDUAL - Dual filters

#### Usage

```
gd=filterbankdual(g,a,L);
gd=filterbankdual(g,a);
```

#### Description

`filterbankdual(g, a, L)` computes the canonical dual filters of  $g$  for a channel subsampling rate of  $a$  (hop-size) and system length  $L$ .  $L$  must be compatible with subsampling rate  $a$  as  $L == \text{filterbanklength}(L, a)$ .

`filterbankrealDual(g, a)` does the same, but the filters must be FIR filters, as the transform length is unspecified.  $L$  will be set to next suitable length equal or bigger than the longest impulse response.

The input and output format of the filters  $g$  are described in the help of `filterbank`.

In addition, the function recognizes a 'forcepainless' flag which forces treating the filterbank  $g$  and  $a$  as a painless case filterbank.

To actually invert the output of a filterbank, use the dual filters together with the ifilterbank function.

REMARK: Perfect reconstruction can be obtained for signals of length  $L$ . In some cases, using dual system calculated for shorter  $L$  might work but check the reconstruction error.

### 5.4.2 FILTERBANKTIGHT - Tight filterbank

#### Usage

```
gt=filterbanktight(g,a,L);
gt=filterbanktight(g,a);
```

#### Description

`filterbanktight(g,a,L)` computes the canonical tight filters of  $g$  for a channel subsampling rate of  $a$  (hop-size) and a system length  $L$ .  $L$  must be compatible with subsampling rate  $a$  as  $L==filterbanklength(L,a)$ .

`filterbanktight(g,a,L)` does the same, but the filters must be FIR filters, as the transform length is unspecified.  $L$  will be set to next suitable length equal or bigger than the longest impulse response.

The input and output format of the filters  $g$  are described in the help of filterbank.

REMARK: The resulting system is tight for length  $L$ . In some cases, using tight system calculated for shorter  $L$  might work but check the reconstruction error.

### 5.4.3 FILTERBANKREALDUAL - Dual filters of filterbank for real signals only

#### Usage

```
gd=filterbankrealdual(g,a,L);
gd=filterbankrealdual(g,a);
```

#### Description

`filterbankrealdual(g,a,L)` computes the canonical dual filters of  $g$  for a channel subsampling rate of  $a$  (hop-size) and a system length  $L$ .  $L$  must be compatible with subsampling rate  $a$  as  $L==filterbanklength(L,a)$ . The dual filters work only for real-valued signals. Use this function on the common construction where the filters in  $g$  only covers the positive frequencies.

`filterbankrealdual(g,a)` does the same, but the filters must be FIR filters, as the transform length is unspecified.  $L$  will be set to next suitable length equal or bigger than the longest impulse response.

The format of the filters  $g$  are described in the help of filterbank.

In addition, the function recognizes a 'forcepainless' flag which forces treating the filterbank  $g$  and  $a$  as a painless case filterbank.

To actually invert the output of a filterbank, use the dual filters together with `2*real(ifilterbank(...))`.

REMARK: Perfect reconstruction can be obtained for signals of length  $L$ . In some cases, using dual system calculated for shorter  $L$  might work but check the reconstruction error.

### 5.4.4 FILTERBANKREALTIGHT - Tight filters of filterbank for real signals only

#### Usage

```
gt=filterbankrealtight(g,a,L);
gt=filterbankrealtight(g,a);
```

#### Description

`filterbankrealtight(g,a,L)` computes the canonical tight filters of  $g$  for a channel subsampling rate of  $a$  (hop-size) and a system length  $L$ .  $L$  must be compatible with subsampling rate  $a$  as  $L==filterbanklength(L,a)$ . The tight filters work only for real-valued signals. Use this function on the common construction where the filters in  $g$  only covers the positive frequencies.

`filterbankrealtight(g,a)` does the same, but the filters must be FIR filters, as the transform length is unspecified.  $L$  will be set to next suitable length equal or bigger than the longest impulse response.

The format of the filters  $g$  are described in the help of filterbank.

REMARK: The resulting system is tight for length  $L$ . In some cases, using tight system calculated for shorter  $L$  might work but check the reconstruction error.

#### 5.4.5 FILTERBANKBOUNDS - Frame bounds of filter bank

##### Usage

```
fcond=filterbankbounds(g,a);
[A,B]=filterbankbounds(g,a);
```

##### Description

`filterbankbounds(g,a,L)` calculates the ratio  $B/A$  of the frame bounds of the filterbank specified by  $g$  and  $a$  for a system of length  $L$ . The ratio is a measure of the stability of the system.

`[A,B]=filterbankbounds(g,a,L)` returns the lower and upper frame bounds explicitly.

#### 5.4.6 FILTERBANKREALBOUNDS - Frame bounds of filter bank for real signals only

##### Usage

```
fcond=filterbankrealbounds(g,a);
[A,B]=filterbankrealbounds(g,a);
```

##### Description

`filterbankrealbounds(g,a,L)` calculates the ratio  $B/A$  of the frame bounds of the filterbank specified by  $g$  and  $a$  for a system of length  $L$ . The ratio is a measure of the stability of the system. Use this function on the common construction where the filters in  $g$  only covers the positive frequencies.

`[A,B]=filterbankrealbounds(g,a)` returns the lower and upper frame bounds explicitly.

#### 5.4.7 FILTERBANKRESPONSE - Response of filterbank as function of frequency

##### Usage

```
gf=filterbankresponse(g,a,L);
```

##### Description

`gf=filterbankresponse(g,a,L)` computes the total response in frequency of a filterbank specified by  $g$  and  $a$  for a signal length of  $L$ . This corresponds to summing up all channels. The output is a useful tool to investigate the behaviour of the windows, as peaks indicate that a frequency is overrepresented in the filterbank, while a dip indicates that it is not well represented.

CAUTION: This function computes a sum of squares of modulus of the frequency responses, which is also the diagonal of the Fourier transform of the frame operator. Use `filterbankfreqz` for evaluation or plotting of frequency responses of filters.

`filterbankresponse(g,a,L,'real')` does the same for a filterbank intended for positive-only filterbank.

`filterbankresponse(g,a,L,fs)` specifies the sampling rate  $fs$ . This is only used for plotting purposes.

`gf=filterbankresponse(g,a,L,'individual')` returns responses in frequency of individual filters as columns of a matrix. The total response can be obtained by `gf = sum(gf,2)`.

`filterbankresponse` takes the following optional parameters:

**'fs', $fs$**  Sampling rate, used only for plotting.

**'complex'** Assume that the filters cover the entire frequency range. This is the default.

**'real'** Assume that the filters only cover the positive frequencies (and is intended to work with real-valued signals only).

**'noplott'** Don't plot the response, just return it.

**'plot'** Plot the response using `plotfftreal` or `plotfft`.

## 5.5 Auxilary

### 5.5.1 FILTERBANKFREQZ - Filterbank frequency responses

#### Usage

```
gf = filterbankfreqz(g, a, L)
```

#### Description

`gf = filterbankfreqz(g, a, L)` calculates length  $L$  frequency responses of filters in  $g$  and returns them as columns of  $gf$ .

If an optional parameters 'plot' is passed to `filterbankfreqz`, the frequency responses will be plotted using `plotfft`. Any optional parameter understood by `plotfft` can be passed in addition to 'plot'.

### 5.5.2 NONU2UFILTERBANK - Non-uniform to uniform filterbank transform

#### Usage

```
[gu, au]=nonu2ufilterbank(g, a)
```

#### Input parameters

**g** Filters as a cell array of structs.

**a** Subsampling factors.

#### Output parameters

**gu** Filters as a cell array of structs.

**au** Uniform subsampling factor.

**pk** Numbers of copies of each filter.

#### Description

`[gu, au]=nonu2ufilterbank(g, a)` calculates uniform filterbank  $gu$ ,  $au=\text{lcm}(a)$  which is identical to the (possibly non-uniform) filterbank  $g, a$  in terms of the equal output coefficients. Each filter  $g\{k\}$  is replaced by  $p(k) = au/a(k)$  advanced versions of itself such that  $z^{ma(k)}G_k(z)$  for  $m = 0, \dots, p-1$ .

This allows using the factorisation algorithm when determining filterbank frame bounds in `filterbankbounds` and `filterbankrealbounds` and in the computation of the dual filterbank in `filterbankdual` and `filterbankreal-dual` which do not work with non-uniform filterbanks.

One can change between the coefficient formats of  $gu$ ,  $au$  and  $g$ ,  $a$  using `nonu2ucfmt` and `u2nonucfmt` in the reverse direction.

**References:** [6]

### 5.5.3 U2NONUCFMT - Uniform to non-uniform filterbank coefficient format

#### Usage

```
c=u2nonucfmt(cu, pk)
```

**Input parameters**

**cu** Uniform filterbank coefficients.

**Output parameters**

**c** Non-uniform filterbank coefficients.

**p** Numbers of copies of each filter.

**Description**

`c = u2nonucfmt (cu, pk)` changes the coefficient format from uniform filterbank coefficients  $cu$  ( $M=\text{sum}(p)$  channels) to non-uniform coefficients  $c$  ( $\text{numel}(p)$  channels) such that each channel of  $c$  consists of  $p$  ( $m$ ) interleaved channels of  $cu$ .

The output  $c$  is a cell-array in any case.

**References:** [6]

**5.5.4 NONU2UCFMT - Non-uniform to uniform filterbank coefficient format****Usage**

```
cu=nonu2ucfmt (c, pk)
```

**Input parameters**

**c** Non-uniform filterbank coefficients.

**Output parameters**

**cu** Uniform filterbank coefficients.

**p** Numbers of copies of each filter.

**Description**

`cu = nonu2ucfmt (c, p)` changes the coefficient format from non-uniform filterbank coefficients  $c$  ( $M=\text{numel}(p)$  channels) to uniform coefficients  $cu$  ( $\text{sum}(p)$  channels) such that each channel of  $cu$  consists of de-interleaved samples of channels of  $c$ .

The output  $cu$  is a cell-array in any case.

**References:** [6]

**5.6 Plots****5.6.1 PLOTFILTERBANK - Plot filterbank and ufilterbank coefficients****Usage**

```
plotfilterbank (coef, a);
plotfilterbank (coef, a, fc);
plotfilterbank (coef, a, fc, fs);
plotfilterbank (coef, a, fc, fs, dynrange);
```

### Description

`plotfilterbank(coef, a)` plots filterbank coefficients *coef* obtained from either the filterbank or ufilterbank functions. The coefficients must have been produced with a time-shift of *a*. For more details on the format of the variables *coef* and *a*, see the help of the filterbank or ufilterbank functions.

`plotfilterbank(coef, a, fc)` makes it possible to specify the center frequency for each channel in the vector *fc*.

`plotfilterbank(coef, a, fc, fs)` does the same assuming a sampling rate of *fs* Hz of the original signal.

`plotfilterbank(coef, a, fc, fs, dynrange)` makes it possible to specify the dynamic range of the coefficients.

`C=plotfilterbank(...)` returns the processed image data used in the plotting. Inputting this data directly to `imagesc` or similar functions will create the plot. This is usefull for custom post-processing of the image data.

`plotfilterbank` supports all the optional parameters of `tfplot`. Please see the help of `tfplot` for an exhaustive list.

In addition to the flags and key/values in `tfplot`, `plotfilterbank` supports the following optional arguments:

<b>'fc',fc</b>	Centre frequencies of the channels. <i>fc</i> must be a vector with the length equal to the number of channels. The default value of [] means to plot the channel no. instead of its frequency.
<b>'tickpos',n</b>	Number of tick positions along the y-axis. The position of the ticks are determined automatically. Default value is 10.
<b>'tick',t</b>	Array of tick positions on the y-axis. Use this option to specify the tick position manually.
<b>'audtick'</b>	Use ticks suitable for visualizing an auditory filterbank. Same as ' <code>tick', [0,100,250,500,1000]</code>



# Chapter 6

## LTFAT - Non-stationary Gabor systems

### 6.1 Transforms

#### 6.1.1 NSDGT - Non-stationary Discrete Gabor transform

##### Usage

```
c=nsdgt(f,g,a,M);  
[c,Ls]=nsdgt(f,g,a,M);
```

##### Input parameters

<b>f</b>	Input signal.
<b>g</b>	Cell array of window functions.
<b>a</b>	Vector of time shifts.
<b>M</b>	Vector of numbers of frequency channels.

##### Output parameters

<b>c</b>	Cell array of coefficients.
<b>Ls</b>	Length of input signal.

##### Description

`nsdgt(f,g,a,M)` computes the nonstationary Gabor coefficients of the input signal  $f$ . The signal  $f$  can be a multichannel signal, given in the form of a 2D matrix of size  $L_s \times W$ , with  $L_s$  the signal length and  $W$  the number of signal channels.

The nonstationary Gabor theory extends standard Gabor theory by enabling the evolution of the window over time. It is therefore necessary to specify a set of windows instead of a single window. This is done by using a cell array for  $g$ . In this cell array, the  $n$ 'th element  $g\{n\}$  is a row vector specifying the  $n$ 'th window.

The resulting coefficients also require a storage in a cell array, as the number of frequency channels is not constant over time. More precisely, the  $n$ 'th cell of  $c$ ,  $c\{n\}$ , is a 2D matrix of size  $M(n) \times W$  and containing the complex local spectra of the signal channels windowed by the  $n$ 'th window  $g\{n\}$  shifted in time at position  $a(n)$ .  $c\{n\}(m,w)$  is thus the value of the coefficient for time index  $n$ , frequency index  $m$  and signal channel  $w$ .

The variable  $a$  contains the distance in samples between two consecutive blocks of coefficients. The variable  $M$  contains the number of channels for each block of coefficients. Both  $a$  and  $M$  are vectors of integers.

The variables  $g$ ,  $a$  and  $M$  must have the same length, and the result  $c$  will also have the same length.

The time positions of the coefficients blocks can be obtained by the following code. A value of 0 correspond to the first sample of the signal:

```
timepos = cumsum(a)-a(1);
```

`[c, Ls]=nsdgt(f, g, a, M)` additionally returns the length  $L_s$  of the input signal  $f$ . This is handy for reconstruction:

```
[c, Ls]=nsdgt(f, g, a, M);
fr=insdgt(c, gd, a, Ls);
```

will reconstruct the signal  $f$  no matter what the length of  $f$  is, provided that  $gd$  are dual windows of  $g$ .

#### Notes:

`nsdgt` uses circular border conditions, that is to say that the signal is considered as periodic for windows overlapping the beginning or the end of the signal.

The phaselocking convention used in `nsdgt` is different from the convention used in the `dgt` function. `nsdgt` results are phaselocked (a phase reference moving with the window is used), whereas `dgt` results are not phaselocked (a fixed phase reference corresponding to time 0 of the signal is used). See the help on `phaselock` for more details on phaselocking conventions.

**References:** [9]

### 6.1.2 UNSDGT - Uniform Nonstationary Discrete Gabor transform

#### Usage

```
c=unsdgt(f, g, a, M);
[c, Ls]=unsdgt(f, g, a, M);
```

#### Input parameters

<b>f</b>	Input signal.
<b>g</b>	Cell array of window functions.
<b>a</b>	Vector of time positions of windows.
<b>M</b>	Numbers of frequency channels.

#### Output parameters

<b>c</b>	Cell array of coefficients.
<b>Ls</b>	Length of input signal.

#### Description

`unsdgt(f, g, a, M)` computes the uniform nonstationary Gabor coefficients of the input signal  $f$ . The signal  $f$  can be a multichannel signal, given in the form of a 2D matrix of size  $L_s \times W$ , with  $L_s$  being the signal length and  $W$  the number of signal channels.

The non-stationary Gabor theory extends standard Gabor theory by enabling the evolution of the window over time. It is therefor necessary to specify a set of windows instead of a single window. This is done by using a cell array for  $g$ . In this cell array, the  $n$ 'th element  $g\{n\}$  is a row vector specifying the  $n$ 'th window. However, the uniformity means that the number of channels is fixed.

The resulting coefficients is stored as a  $M \times N \times W$  array.  $c(m, n, w)$  is thus the value of the coefficient for time index  $n$ , frequency index  $m$  and signal channel  $w$ .

The variable  $a$  contains the distance in samples between two consecutive blocks of coefficients.  $a$  is a vectors of integers. The variables  $g$  and  $a$  must have the same length.

The time positions of the coefficients blocks can be obtained by the following code. A value of 0 correspond to the first sample of the signal:

```
timepos = cumsum(a)-a(1);
```

$[c, Ls] = nsdgt(f, g, a, M)$  additionally returns the length  $Ls$  of the input signal  $f$ . This is handy for reconstruction:

```
[c, Ls] = unsdgt(f, g, a, M);
fr = iunsdgt(c, gd, a, Ls);
```

will reconstruct the signal  $f$  no matter what the length of  $f$  is, provided that  $gd$  are dual windows of  $g$ .

#### Notes:

`unsdgt` uses circular border conditions, that is to say that the signal is considered as periodic for windows overlapping the beginning or the end of the signal.

The phaselocking convention used in `unsdgt` is different from the convention used in the `dgt` function. `unsdgt` results are phaselocked (a phase reference moving with the window is used), whereas `dgt` results are not phaselocked (a fixed phase reference corresponding to time 0 of the signal is used). See the help on `phaselock` for more details on phaselocking conventions.

**References:** [9]

### 6.1.3 INSDGT - Inverse nonstationary discrete Gabor transform

#### Usage

```
f = insdgt(c, g, a, Ls);
```

#### Input parameters

<b>c</b>	Cell array of coefficients.
<b>g</b>	Cell array of window functions.
<b>a</b>	Vector of time positions of windows.
<b>Ls</b>	Length of input signal.

#### Output parameters

<b>f</b>	Signal.
----------	---------

#### Description

`insdgt(c, g, a, Ls)` computes the inverse non-stationary Gabor transform of the input coefficients  $c$ .

`insdgt` is used to invert the functions `nsdgt` and `unsdgt`. Please read the help of these functions for details of variables format and usage.

For perfect reconstruction, the windows used must be dual windows of the ones used to generate the coefficients. The windows can be generated using `nsgabdual` or `nsgabtight`.

**References:** [9]

### 6.1.4 NSDGTRREAL - Non-stationary Discrete Gabor transform for real valued signals

#### Usage

```
c = nsdgtrreal(f, g, a, M);
[c, Ls] = nsdgtrreal(f, g, a, M);
```

**Input parameters**

<b>f</b>	Input signal.
<b>g</b>	Cell array of window functions.
<b>a</b>	Vector of time positions of windows.
<b>M</b>	Vector of numbers of frequency channels.

**Output parameters**

<b>c</b>	Cell array of coefficients.
<b>Ls</b>	Length of input signal.

**Description**

`nsdgtreal(f, g, a, M)` computes the nonstationary Gabor coefficients of the input signal  $f$ . The signal  $f$  can be a multichannel signal, given in the form of a 2D matrix of size  $L_s \times W$ , with  $L_s$  the signal length and  $W$  the number of signal channels.

As opposed to `nsdgt` only the coefficients of the positive frequencies of the output are returned. `nsdgtreal` will refuse to work for complex valued input signals.

The nonstationary Gabor theory extends standard Gabor theory by enabling the evolution of the window over time. It is therefore necessary to specify a set of windows instead of a single window. This is done by using a cell array for  $g$ . In this cell array, the  $n$ 'th element  $g\{n\}$  is a row vector specifying the  $n$ 'th window.

The resulting coefficients also require a storage in a cell array, as the number of frequency channels is not constant over time. More precisely, the  $n$ 'th cell of  $c$ ,  $c\{n\}$ , is a 2D matrix of size  $M(n)/2 + 1 \times W$  and containing the complex local spectra of the signal channels windowed by the  $n$ 'th window  $g\{n\}$  shifted in time at position  $a(n)$ .  $c\{n\}(m, 1)$  is thus the value of the coefficient for time index  $n$ , frequency index  $m$  and signal channel  $l$ .

The variable  $a$  contains the distance in samples between two consecutive blocks of coefficients. The variable  $M$  contains the number of channels for each block of coefficients. Both  $a$  and  $M$  are vectors of integers.

The variables  $g$ ,  $a$  and  $M$  must have the same length, and the result  $c$  will also have the same length.

The time positions of the coefficients blocks can be obtained by the following code. A value of 0 correspond to the first sample of the signal:

```
timepos = cumsum(a) - a(1);
```

`[c, Ls] = nsdgtreal(f, g, a, M)` additionally returns the length  $L_s$  of the input signal  $f$ . This is handy for reconstruction:

```
[c, Ls] = nsdgtreal(f, g, a, M);
fr = insdgreal(c, gd, a, Ls);
```

will reconstruct the signal  $f$  no matter what the length of  $f$  is, provided that  $gd$  are dual windows of  $g$ .

**Notes:**

`nsdgtreal` uses circular border conditions, that is to say that the signal is considered as periodic for windows overlapping the beginning or the end of the signal.

The phaselocking convention used in `nsdgtreal` is different from the convention used in the `dgt` function. `nsdgtreal` results are phaselocked (a phase reference moving with the window is used), whereas `dgt` results are not phaselocked (a fixed phase reference corresponding to time 0 of the signal is used). See the help on `phaselock` for more details on phaselocking conventions.

**References:** [9]

### 6.1.5 UNSDGTREAL - Uniform non-stationary Discrete Gabor transform

#### Usage

```
c=unsdgtreal(f,g,a,M);
[c,Ls]=unsdgtreal(f,g,a,M);
```

#### Input parameters

<b>f</b>	Input signal.
<b>g</b>	Cell array of window functions.
<b>a</b>	Vector of time positions of windows.
<b>M</b>	Vector of numbers of frequency channels.

#### Output parameters

<b>c</b>	Cell array of coefficients.
<b>Ls</b>	Length of input signal.

#### Description

`unsdgtreal(f,g,a,M)` computes the nonstationary Gabor coefficients of the input signal  $f$ . The signal  $f$  can be a multichannel signal, given in the form of a 2D matrix of size  $L_s \times W$ , with  $L_s$  the signal length and  $W$  the number of signal channels.

As opposed to `nsdgt` only the coefficients of the positive frequencies of the output are returned. `unsdgtreal` will refuse to work for complex valued input signals.

The non-stationary Gabor theory extends standard Gabor theory by enabling the evolution of the window over time. It is therefore necessary to specify a set of windows instead of a single window. This is done by using a cell array for  $g$ . In this cell array, the  $n$ 'th element  $g\{n\}$  is a row vector specifying the  $n$ 'th window. The uniformity means that the number of channels is not allowed to vary over time.

The resulting coefficients is stored as a  $M/2 + 1 \times N \times W$  array.  $c(m, n, l)$  is thus the value of the coefficient for time index  $n$ , frequency index  $m$  and signal channel  $l$ .

The variable  $a$  contains the distance in samples between two consecutive blocks of coefficients. The variable  $M$  contains the number of channels for each block of coefficients. Both  $a$  and  $M$  are vectors of integers.

The variables  $g$ ,  $a$  and  $M$  must have the same length, and the result  $c$  will also have the same length.

The time positions of the coefficients blocks can be obtained by the following code. A value of 0 correspond to the first sample of the signal:

```
timepos = cumsum(a)-a(1);
```

`[c,Ls]=unsdgtreal(f,g,a,M)` additionally returns the length  $L_s$  of the input signal  $f$ . This is handy for reconstruction:

```
[c,Ls]=unsdgtreal(f,g,a,M);
fr=insdgtreal(c,gd,a,Ls);
```

will reconstruct the signal  $f$  no matter what the length of  $f$  is, provided that  $gd$  are dual windows of  $g$ .

#### Notes:

`unsdgtreal` uses circular border conditions, that is to say that the signal is considered as periodic for windows overlapping the beginning or the end of the signal.

The phaselocking convention used in `unsdgtreal` is different from the convention used in the `dgt` function. `unsdgtreal` results are phaselocked (a phase reference moving with the window is used), whereas `dgt` results are not phaselocked (a fixed phase reference corresponding to time 0 of the signal is used). See the help on `phaselock` for more details on phaselocking conventions.

**References:** [9]

### 6.1.6 INSDGTREAL - Inverse NSDGT for real-valued signals

#### Usage

```
f=insdgt(c,g,a,M,Ls);
```

#### Input parameters

<b>c</b>	Cell array of coefficients.
<b>g</b>	Cell array of window functions.
<b>a</b>	Vector of time positions of windows.
<b>M</b>	Vector of numbers of frequency channels.
<b>Ls</b>	Length of input signal.

#### Output parameters

<b>f</b>	Signal.
----------	---------

#### Description

`insdgt(c,g,a,Ls)` computes the inverse non-stationary Gabor transform of the input coefficients  $c$ .  
`insdgt` is used to invert the functions `nsdgt` and `unsdgt`. Please read the help of these functions for details of variables format and usage.

For perfect reconstruction, the windows used must be dual windows of the ones used to generate the coefficients. The windows can be generated using `nsgabdual` or `nsgabtight`.

**References:** [9]

## 6.2 Window construction and bounds

### 6.2.1 NSGABDUAL - Canonical dual window for non-stationary Gabor frames

#### Usage

```
gd=nsgabdual(g,a,M);
gd=nsgabdual(g,a,M,L);
```

#### Input parameters

<b>g</b>	Cell array of windows.
<b>a</b>	Vector of time shift.
<b>M</b>	Vector of numbers of channels.
<b>L</b>	Transform length.

#### Output parameters

<b>gd</b>	Cell array of canonical dual windows
-----------	--------------------------------------

**Description**

`nsgabdual(g, a, M, L)` computes the canonical dual windows of the non-stationary discrete Gabor frame defined by windows given in  $g$  and time-shifts given by  $a$ .

`nsgabdual` is designed to be used with the functions `nsdgt` and `insdgt`. See the help on `nsdgt` for more details about the variables structure.

The computed dual windows are only valid for the 'painless case', that is to say that they ensure perfect reconstruction only if for each window the number of frequency channels used for computation of `nsdgt` is greater than or equal to the window length. This correspond to cases for which the frame operator is diagonal.

**References:** [9]

**6.2.2 NSGABTIGHT - Canonical tight window for non-stationary Gabor frames****Usage**

```
gt=nsgabtight(g, a, M);
gt=nsgabtight(g, a, M, L);
```

**Input parameters**

<b>g</b>	Cell array of windows
<b>a</b>	Vector of time shifts of windows.
<b>M</b>	Vector of numbers of channels.
<b>L</b>	Transform length.

**Output parameters**

<b>gt</b>	Cell array of canonical tight windows
-----------	---------------------------------------

**Description**

`nsgabtight(g, a, M)` computes the canonical tight windows of the non-stationary discrete Gabor frame defined by windows given in  $g$  and time-shifts given by  $a$ .

`nsgabtight` is designed to be used with functions `nsdgt` and `insdgt`. Read the help on `nsdgt` for more details about the variables structure.

The computed tight windows are only valid for the 'painless case', that is to say that they ensure perfect reconstruction only if for each window the number of frequency channels used for computation of `nsdgt` is greater than or equal to the window length. This correspond to cases for which the frame operator is diagonal.

**References:** [9]

**6.2.3 NSGABFRAMEBOUNDS - Frame bounds of nonstationnary Gabor frame****Usage**

```
fcond=nsgabframebounds(g, a, M);
[A, B]=nsgabframebounds(g, a, M);
```

**Input parameters**

<b>g</b>	Cell array of windows
<b>a</b>	Vector of time positions of windows.
<b>M</b>	Vector of numbers of frequency channels.

**Output parameters**

<b>fcond</b>	Frame condition number ( $B/A$ )
<b>A, B</b>	Frame bounds.

**Description**

`nsgabframebounds(g, a, Ls)` calculates the ratio  $B/A$  of the frame bounds of the nonstationary discrete Gabor frame defined by windows given in  $g$  at positions given by  $a$ . Please see the help on `nsdgt` for a more thorough description of  $g$  and  $a$ .

`[A, B] = nsgabframebounds(g, a, Ls)` returns the actual frame bounds  $A$  and  $B$  instead of just the their ratio.

The computed frame bounds are only valid for the 'painless case' when the number of frequency channels used for computation of `nsdgt` is greater than or equal to the window length. This correspond to cases for which the frame operator is diagonal.

**References:** [9]

**6.2.4 NSGABFRAMEDIAG - Diagonal of Gabor frame operator****Usage**

```
d=nsgabframediag(g, a, M);
```

**Input parameters**

<b>g</b>	Window function.
<b>a</b>	Length of time shift.
<b>M</b>	Number of channels.

**Output parameters**

<b>d</b>	Diagonal stored as a column vector
----------	------------------------------------

**Description**

`nsgabframediag(g, a, M)` computes the diagonal of the non-stationary Gabor frame operator with respect to the window  $g$  and parameters  $a$  and  $M$ . The diagonal is stored as a column vector of length  $L=\text{sum}(a)$ .

The diagonal of the frame operator can for instance be used as a preconditioner.

**6.3 Plots****6.3.1 PLOTNSDGT - Plot spectrogram from non-stationary Gabor coefficients****Usage**

```
plotnsdgt(c, a, fs, dynrange);
```

**Input parameters**

<b>coef</b>	Cell array of coefficients.
<b>a</b>	Vector of time positions of windows.
<b>fs</b>	signal sample rate in Hz (optional)
<b>dynrange</b>	Color scale dynamic range in dB (optional).

### Description

`plotnsdgt (coef, a)` plots coefficients computed using nsdgt or unsdgt. For more details on the format of the variables *coef* and *a*, please read the function help for these functions.

`plotnsdgt (coef, a, fs)` does the same assuming a sampling rate of *fs* Hz of the original signal.

`plotnsdgt (coef, a, fs, dynrange)` additionally limits the dynamic range.

`C=plotnsdgt (...)` returns the processed image data used in the plotting. Inputting this data directly to `imagesc` or similar functions will create the plot. This is useful for custom post-processing of the image data.

`plotnsdgt` supports all the optional parameters of `tfplot`. Please see the help of `tfplot` for an exhaustive list. In addition, the following parameters may be specified:

**'xres',xres**      Approximate number of pixels along x-axis / time. The default value is 800

**'yres',yres**      Approximate number of pixels along y-axis / frequency The default value is 600

### 6.3.2 PLOTNSDGTREAL - Plot spectrogram from NSDGTRREAL coefficients

#### Usage

```
plotnsdgtrreal(c,a,fs,dynrange);
```

#### Input parameters

**coef**      Cell array of coefficients.

**a**      Vector of time positions of windows.

**fs**      signal sample rate in Hz (optional).

**dynrange**      Colorscale dynamic range in dB (optional).

#### Description

`plotnsdgtrreal (coef, a)` plots coefficients computed using nsdgtrreal or unsdgtrreal. For more details on the format of the variables *coef* and *a*, please read the function help for these functions.

`plotnsdgtrreal (coef, a, fs)` does the same assuming a sampling rate of *fs* Hz of the original signal.

`plotnsdgtrreal (coef, a, fs, dynrange)` additionally limits the dynamic range.

`C=plotnsdgtrreal (...)` returns the processed image data used in the plotting. Inputting this data directly to `imagesc` or similar functions will create the plot. This is usefull for custom post-processing of the image data.

`plotnsdgtrreal` supports all the optional parameters of `tfplot`. Please see the help of `tfplot` for an exhaustive list. In addition, the following parameters may be specified:

**'xres',xres**      Approximate number of pixels along x-axis /time. Default value is 800

**'yres',yres**      Approximate number of pixels along y-axis / frequency Default value is 600



# Chapter 7

## LTFAT - Frames

### 7.1 Creation of a frame object

#### 7.1.1 FRAME - Construct a new frame

##### Usage

```
F=frame (ftype, ...);
```

##### Description

`F=frame (ftype, ...)` constructs a new frame object  $F$  of type  $ftype$ . Arguments following  $ftype$  are specific to the type of frame chosen.

##### Time-frequency frames

`frame ('dgt', g, a, M)` constructs a Gabor frame with window  $g$ , time-shift  $a$  and  $M$  channels. See the help on `dgt` for more information.

`frame ('dgtrreal', g, a, M)` constructs a Gabor frame for real-valued signals with window  $g$ , time-shift  $a$  and  $M$  channels. See the help on `dgtrreal` for more information.

`frame ('dwilt', g, M)` constructs a Wilson basis with window  $g$  and  $M$  channels. See the help on `dwilt` for more information.

`frame ('wmdct', g, M)` constructs a windowed MDCT basis with window  $g$  and  $M$  channels. See the help on `wmdct` for more information.

`frame ('filterbank', g, a, M)` constructs a filterbank with filters  $g$ , time-shifts of  $a$  and  $M$  channels. For the ease of implementation, it is necessary to specify  $M$ , even though it strictly speaking could be deduced from the size of the windows. See the help on `filterbank` for more information on the parameters. Similarly, you can construct a uniform filterbank by selecting '`ufilterbank`', a positive-frequency filterbank by selecting '`filterbankreal`' or a uniform positive-frequency filterbank by selecting '`ufilterbankreal`'.

`frame ('nsdgt', g, a, M)` constructs a non-stationary Gabor frame with filters  $g$ , time-shifts of  $a$  and  $M$  channels. See the help on `nsdgt` for more information on the parameters. Similarly, you can construct a uniform NSDGT by selecting '`unsdgt`', an NSDGT for real-valued signals only by selecting '`nsdgtreal`' or a uniform NSDGT for real-valued signals by selecting '`unsdgtreal`'.

##### Wavelet frames

`frame ('fwt', w, J)` constructs a wavelet frame with wavelet definition  $w$  and  $J$  number of filterbank iterations. Similarly, a redundant time invariant wavelet representation can be constructed by selecting '`ufwt`'. See the help on `fwt` and `ufwt` for more information.

`frame ('wfbt', wt)` constructs a wavelet filterbank tree defined by the wavelet filterbank tree definition  $wt$ . Similarly, an undecimated wavelet filterbank tree can be constructed by selecting '`uwfbt`'. See the help on `wfbt` and `uwfbt` for more information.

`frame('wpfbt', wt)` constructs a wavelet packet filterbank tree defined by the wavelet filterbank tree definition `wt`. Similarly, an undecimated wavelet packet filterbank tree can be constructed by selecting '`uwpfbt`'. See the help on `wpfbt` and `uwpfbt` for more information.

### Pure frequency frames

`frame('dft')` constructs a basis where the analysis operator is the dft, and the synthesis operator is its inverse, `idft`. Completely similar to this, you can enter the name of any of the cosine or sine transforms `dcti`, `dctii`, `dctiii`, `dctiv`, `dsti`, `dstii`, `dstiii` or `dstiv`.

`frame('dftreal')` constructs a normalized `fftreal` basis for real-valued signals of even length only. The basis is normalized the dft to ensure that it is orthonormal.

### Special / general frames

`frame('gen', g)` constructs an general frame with analysis matrix `g`. The frame atoms must be stored as column vectors in the matrices.

`frame('identity')` constructs the canonical orthonormal basis, meaning that all operators return their input as output, so it is the dummy operation.

### Container frames

`frame('fusion', w, F1, F2, ...)` constructs a fusion frame, which is the collection of the frames specified by `F1, F2, ...`. The vector `w` contains a weight for each frame. If `w` is a scalar, this weight will be applied to all the sub-frames.

`frame('tensor', F1, F2, ...)` constructs a tensor product frame, where the frames `F1, *F2, ...` are applied along the 1st, 2nd etc. dimensions. If you don't want any action along a specific dimension, use the `identity` frame along that dimension. Any remaining dimensions in the input signal are left alone.

### Wrapper frames

Frames types in this section are "virtual". They serve as a wrapper for a different type of frame.

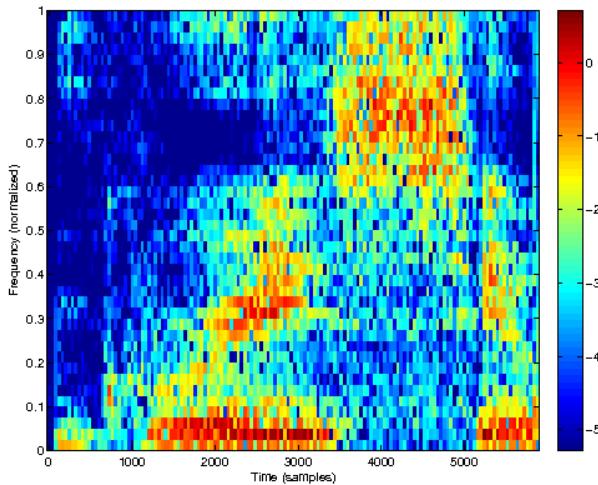
`frame('erbletfb', fs, Ls, ...)` constructs an Erblet filterbank frame for a given samp. frequency `fs` working with signals of length `Ls`. See `erbfilters` for a description of additional parameters as all parameters other than the frame type string '`erbletfb`' are passed to it. NOTE: The resulting frame is defined only for a single signal length `Ls`. Shorter signals will be zero-padded, signals longer than `Ls` cannot be processed. The actual frame type is '`filterbank`' or '`filterbankreal`'.

`frame('cqtfb', fs, fmin, fmax, bins, Ls, ...)` constructs a CQT filterbank frame for a given samp. frequency `fs` working with signals of length `Ls`. See `cqtfilters` for a description of other parameters. NOTE: The resulting frame is defined only for a single signal length `Ls`. Shorter signals will be zero-padded, signals longer than `Ls` cannot be processed. The actual frame type is '`filterbank`' or '`filterbankreal`'.

### Examples

The following example creates a Modified Discrete Cosine Transform frame, analyses an input signal and plots the frame coefficients:

```
F=frame('wmdct','gauss',40);
c=frana(F,greasy);
plotframe(F,c,'dynrange',60);
```



### 7.1.2 FRAMEPAIR - Construct a new frame

#### Usage

```
[F1,F2]=framepair(ftype,g1,g2,...);
```

#### Description

`[F1,F2]=framepair(ftype,g1,g2,...)` constructs two new frame objects *F1* and *F2* of the same type *ftype* using the windows *g1* and *g2*. The windows are specific to chosen frame type. See the help on *frame* for the windows and arguments.

This function makes it easy to create a pair of canonical dual frames: simply specify '*dual*' as window if one frame should be the dual of the other.

This is most easily explained through some examples. The following example creates a Gabor frame for real-valued signals with a Gaussian analysis window and its canonical dual frame as the synthesis frame:

```
f=greasy;
[Fa,Fs]=framepair('dgtreal','gauss','dual',20,294);
c=frana(Fa,f);
r=frsyn(Fs,c);
norm(f-r)
```

*This code produces the following output:*

```
ans =
5.4587e-15
```

The following example creates a Wilson basis with a Gaussian synthesis window, and its canonical dual frame as the analysis frame:

```
[Fa,Fs]=framepair('dwilt','dual','gauss',20);
```

### 7.1.3 FRAMEDUAL - Construct the canonical dual frame

#### Usage

```
Fd=framedual(F);
```

**Description**

`Fd=framedual(F)` returns the canonical dual frame of  $F$ .

The canonical dual frame can be used to get perfect reconstruction as in the following example:

```
% Create a frame and its canonical dual
F=frame('dgt','hamming',32,64);
Fd=framedual(F);

% Compute the frame coefficients and test for perfect
% reconstruction
f=gspi;
c=frana(F,f);
r=frsyn(Fd,c);
norm(r(1:length(f))-f)
```

*This code produces the following output:*

```
ans =
9.7284e-15
```

**7.1.4 FRAMETIGHT - Construct the canonical tight frame****Usage**

```
Ft=frametight(F);
```

**Description**

`Ft=frametight(F)` returns the canonical tight frame of  $F$ .

The canonical tight frame can be used to get perfect reconstruction if it is used for both analysis and synthesis. This is demonstrated in the following example:

```
% Create a frame and its canonical tight
F=frame('dgt','hamming',32,64);
Ft=frametight(F);

% Compute the frame coefficients and test for perfect
% reconstruction
f=gspi;
c=frana(Ft,f);
r=frsyn(Ft,c);
norm(r(1:length(f))-f)
```

*This code produces the following output:*

```
ans =
1.0650e-14
```

**7.1.5 FRAMEACCEL - Precompute structures****Usage**

```
F=frameaccel(F,Ls);
```

**Description**

`F=frameaccel(F,Ls)` precomputes certain structures that makes the basic frame operations `frana` and `frsyn` faster (like instantiating the window from a textual description). If you only need to call the routines once, calling `frameaccel` first will not provide any total gain, but if you are repeatedly calling these routines, for instance in an iterative algorithm, it will be a benefit.

Notice that you need to input the signal length  $L_s$ , so this routines will only be a benefit if  $L_s$  stays fixed. If `frameaccel` is called twice for the same transform length, no additional computations will be done.

## 7.2 Linear operators

### 7.2.1 FRANA - Frame analysis operator

**Usage**

```
c=frana(F,f);
```

**Description**

`c=frana(F,f)` computes the frame coefficients  $c$  of the input signal  $f$  using the frame  $F$ . The frame object  $F$  must have been created using `frame` or `framepair`.

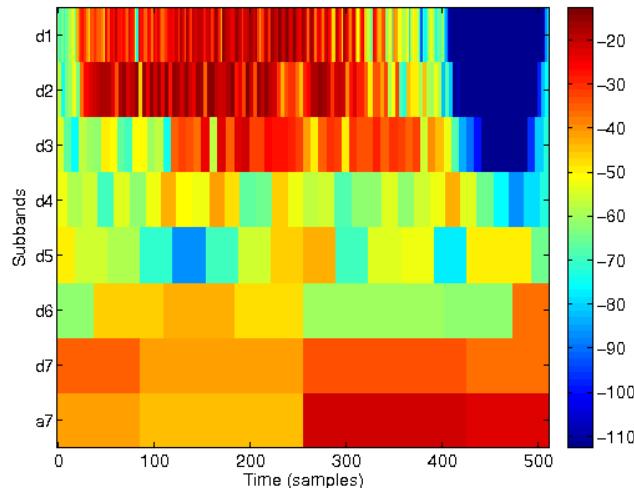
If  $f$  is a matrix, the transform will be applied along the columns of  $f$ . If  $f$  is an N-D array, the transform will be applied along the first non-singleton dimension.

The output coefficients are stored as columns. This is usually **not** the same format as the 'native' format of the frame. As an examples, the output from `frana` for a gabor frame cannot be passed to `idgt` without a reshape.

**Examples:**

In the following example the signal `bat` is analyzed through a wavelet frame. The result are the frame coefficients associated with the input signal `bat` and the analysis frame '`fwt`':

```
f = bat;
w = 'sym8';
J = 7;
F = frame('fwt', w, J);
c = frana(F, f);
% A plot of the frame coefficients
plotframe(F, c, 'dynrange', 100);
```



## 7.2.2 FRSYN - Frame synthesis operator

### Usage

```
f=frsyn(F, c);
```

### Description

`f=frsyn(F, c)` constructs a signal  $f$  from the frame coefficients  $c$  using the frame  $F$ . The frame object  $F$  must have been created using `frame`.

### Examples:

In the following example a signal  $f$  is constructed through the frame synthesis operator using a Gabor frame. The coefficients associated with this Gabor expansion are contained in an identity matrix. The identity matrix corresponds to a diagonal in the time-frequency plane, that is, one atom at each time position with increasing frequency.:

```
a = 10;
M = 40;

F = frame('dgt', 'gauss', a, M);

c = frmenative2coef(F, eye(40));

f = frsyn(F, c);
```

## 7.2.3 FRSYNMATRIX - Frame synthesis operator matrix

### Usage

```
G=frsynmatrix(F, L);
```

### Description

`G=frsynmatrix(F, L)` returns the matrix representation  $G$  of the frame synthesis operator for a frame  $F$  of length  $L$ . The frame object  $F$  must have been created using `frame`.

The frame synthesis operator matrix contains all the frame atoms as column vectors. It has dimensions  $L \times Ncoef$ , where  $Ncoef$  is the number of coefficients. The number of coefficients can be found as `Ncoef=frameclength(L)`. This means that the frame matrix is usually **very** large, and this routine should only be used for small values of  $L$ .

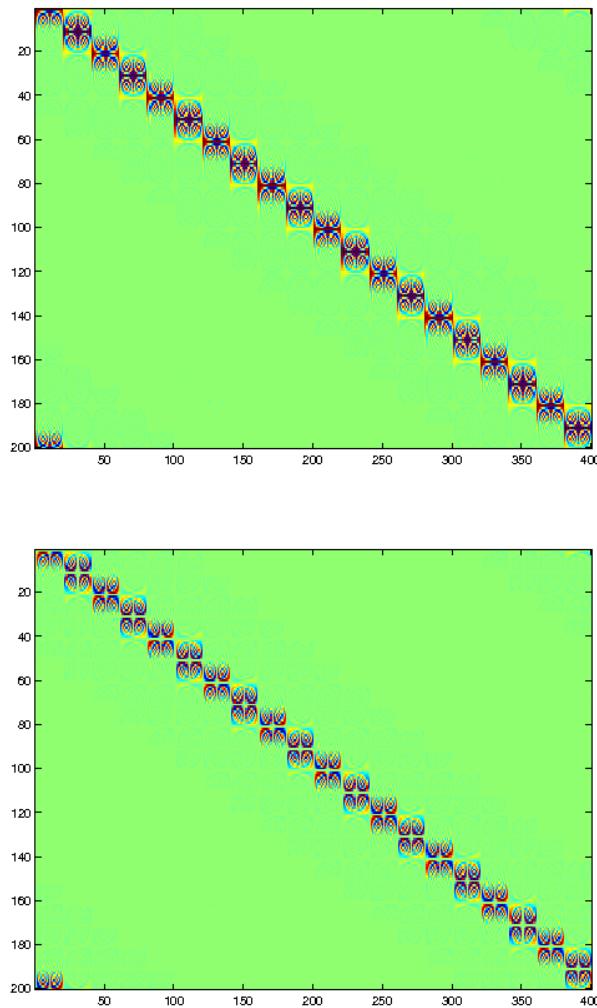
The action of the frame analysis operator `frana` is equal to multiplication with the Hermitean transpose of the frame matrix. Consider the following simple example:

```
L=200;
F=frame('dgt','gauss',10,20);
G=frsynmatrix(F,L);
testsig = randn(L,1);
res = frana(F,testsig)-G'*testsig;
norm(res)
% Show the matrix (real and imaginary parts)
figure(1); imagesc(real(G));
figure(2); imagesc(imag(G));
```

This code produces the following output:

```
ans =
```

```
7.1068e-15
```



#### 7.2.4 FRAMEDIAG - Compute the diagonal of the frame operator

##### Usage

```
d=framediag(F,L);
```

##### Description

`framediag(F,L)` computes the diagonal of the frame operator for a frame of type  $F$  of length  $L$ .

The diagonal of the frame operator can for instance be used as a preconditioner.

#### 7.2.5 FRANAITER - Iterative analysis

##### Usage

```
c=franaiter(F,f);
[c,relres,iter]=franaiter(F,f,...);
```

##### Input parameters

**F** Frame.

**f** Signal.

**Ls** Length of signal.

### Output parameters

<b>c</b>	Array of coefficients.
<b>relres</b>	Vector of residuals.
<b>iter</b>	Number of iterations done.

### Description

`c=franaiter(F, f)` computes the frame coefficients  $c$  of the signal  $f$  using an iterative method such that perfect reconstruction can be obtained using `frsyn`. `franaiter` always works, even when `frana` cannot generate perfect reconstruction coefficients.

`[c, relres, iter]=franaiter(...)` additionally returns the relative residuals in a vector `relres` and the number of iteration steps `iter`.

**Note:** If it is possible to explicitly calculate the canonical dual frame then this is usually a much faster method than invoking `franaiter`.

`franaiter` takes the following parameters at the end of the line of input arguments:

<b>'tol',t</b>	Stop if relative residual error is less than the specified tolerance. Default is 1e-9 (1e-5 for single precision)
<b>'maxit',n</b>	Do at most n iterations.
<b>'pg'</b>	Solve the problem using the Conjugate Gradient algorithm. This is the default.
<b>'pcg'</b>	Solve the problem using the Preconditioned Conjugate Gradient algorithm.
<b>'print'</b>	Display the progress.
<b>'quiet'</b>	Don't print anything, this is the default.

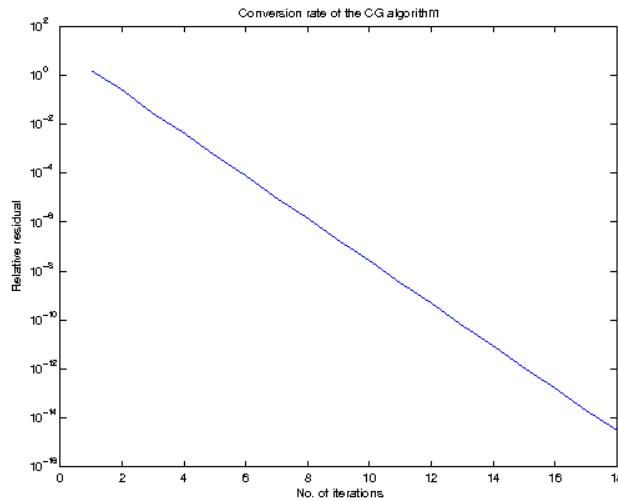
### Examples

The following example shows how to reconstruct a signal without ever using the dual frame:

```
f=greasy;
F=frame('dgtreal','gauss',40,60);
[c,relres,iter]=franaiter(F,f,'tol',1e-14);
r=frsyn(F,c);
norm(f-r)/norm(f)
semilogy(relres);
title('Conversion rate of the CG algorithm');
xlabel('No. of iterations');
ylabel('Relative residual');
```

*This code produces the following output:*

```
ans =
2.0225e-15
```



### 7.2.6 FRSYNITER - Iterative synthesis

#### Usage

```
f=frsyniter(F,c);
f=frsyniter(F,c,Ls);
[f,relres,iter]=frsyniter(F,c,...);
```

#### Input parameters

<b>F</b>	Frame
<b>c</b>	Array of coefficients.
<b>Ls</b>	length of signal.

#### Output parameters

<b>f</b>	Signal.
<b>relres</b>	Vector of residuals.
<b>iter</b>	Number of iterations done.

#### Description

`f=frsyniter(F,c)` iteratively inverts the analysis operator of  $F$ , so `frsyniter` always performs the inverse operation of `frana`, even when a perfect reconstruction is not possible by using `frsyn`.

`[f,relres,iter]=frsyniter(...)` additionally returns the relative residuals in a vector `relres` and the number of iteration steps `iter`.

**Note:** If it is possible to explicitly calculate the canonical dual frame then this is usually a much faster method than invoking `frsyniter`.

`frsyniter` takes the following parameters at the end of the line of input arguments:

<b>'tol',t</b>	Stop if relative residual error is less than the specified tolerance. Default is 1e-9 (1e-5 for single precision)
<b>'maxit',n</b>	Do at most n iterations.
<b>'cg'</b>	Solve the problem using the Conjugate Gradient algorithm. This is the default.

<b>'pcg'</b>	Solve the problem using the Preconditioned Conjugate Gradient algorithm.
<b>'print'</b>	Display the progress.
<b>'quiet'</b>	Don't print anything, this is the default.

### Examples

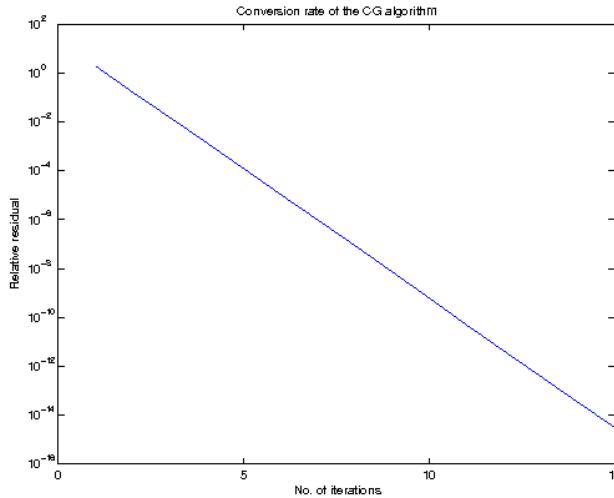
The following example shows how to reconstruct a signal without ever using the dual frame:

```
F=frame('dgtreal','gauss',10,20);
c=frana(F,bat);
[r,relres]=frsyniter(F,c,'tol',1e-14);
norm(bat-r)/norm(bat)
semilogy(relres);
title('Conversion rate of the CG algorithm');
xlabel('No. of iterations');
ylabel('Relative residual');
```

*This code produces the following output:*

```
ans =
```

```
1.5916e-15
```



## 7.3 Visualization

### 7.3.1 PLOTFRAME - Plot frame coefficients

#### Usage

```
plotframe(F, insig, ...);
C = plotframe(...);
```

#### Description

`plotframe(F, c)` plots the frame coefficients  $c$  using the plot command associated to the frame  $F$ .  
`C=plotframe(...)` for frames with time-frequency plots returns the processed image data used in the plotting. The function produces an error for frames which does not have a time-frequency plot.

`plotframe(F, c, ...)` passes any additional parameters to the native plot routine. Please see the help on the specific plot routine for a complete description.

The following common set of parameters are supported by all plotting routines:

<b>'dynrange',r</b>	Limit the dynamical range to $r$ . The default value of [] means to not limit the dynamical range.
<b>'db'</b>	Apply $20 \cdot \log_{10}$ to the coefficients. This makes it possible to see very weak phenomena, but it might show too much noise. A logarithmic scale is more adapted to perception of sound. This is the default.
<b>'dbsq'</b>	Apply $10 \cdot \log_{10}$ to the coefficients. Same as the ' <code>db</code> ' option, but assume that the input is already squared.
<b>'lin'</b>	Show the coefficients on a linear scale. This will display the raw input without any modifications. Only works for real-valued input.
<b>'linsq'</b>	Show the square of the coefficients on a linear scale.
<b>'linabs'</b>	Show the absolute value of the coefficients on a linear scale.
<b>'clim',clim</b>	Only show values in between $\text{clim}(1)$ and $\text{clim}(2)$ . This is usually done by adjusting the colormap. See the help on <code>imagesc</code> .

### 7.3.2 FRAMEGRAM - Easy visualization of energy in frame space

#### Usage

```
framegram(F, c, ...);
```

#### Description

`framegram(F, c)` plots the energy of the coefficients computed from the input signal  $x$  using the frame  $F$  for analysis. This is just a shorthand for:

```
plotframe(F, abs(frana(F, c)) .^2);
```

Any additional arguments given to `framegram` are passed onto `plotframe`.

## 7.4 Information about a frame

### 7.4.1 FRAMEBOUNDS - Frame bounds

#### Usage

```
fcond=framebounds(F);
[A,B]=framebounds(F);
[...]=framebounds(F,Ls);
```

#### Description

`framebounds(F)` calculates the ratio  $B/A$  of the frame bounds of the frame given by  $F$ .

`framebounds(F, Ls)` additionally specifies a signal length for which the frame should work.

`[A,B]=framebounds(F)` returns the frame bounds  $A$  and  $B$  instead of just their ratio.

'framebounds' accepts the following optional parameters:

<b>'fac'</b>	Use a factorization algorithm. The function will throw an error if no algorithm is available.
<b>'iter'</b>	Call <code>eigs</code> to use an iterative algorithm.

<b>'full'</b>	Call <code>eig</code> to solve the full problem.
<b>'auto'</b>	Choose the <code>fac</code> method if possible, otherwise use the <code>full</code> method for small problems and the <code>iter</code> method for larger problems. This is the default.
<b>'crossover',c</b>	Set the problem size for which the 'auto' method switches between <code>full</code> and <code>iter</code> . Default is 200.

The following parameters specifically related to the `iter` method:

<b>'tol',t</b>	Stop if relative residual error is less than the specified tolerance. Default is 1e-9
<b>'maxit',n</b>	Do at most n iterations.
<b>'p',p</b>	The number of Lanczos basis vectors to use. More vectors will result in faster convergence, but a larger amount of memory. The optimal value of p is problem dependent and should be less than L. The default value is 2.
<b>'print'</b>	Display the progress.
<b>'quiet'</b>	Don't print anything, this is the default.

#### 7.4.2 FRAMERED - Redundancy of a frame

##### Usage

```
red=framered(F);
```

##### Description

`framered(F)` computes the redundancy of a given frame  $F$ . If the redundancy is larger than 1 (one), the frame transform will produce more coefficients than it consumes. If the redundancy is exactly 1 (one), the frame is a basis.

##### Examples:

The following simple example shows how to obtain the redundancy of a Gabor frame:

```
F=frame('dgt','gauss',30,40);
framered(F)
```

*This code produces the following output:*

```
ans =
1.3333
```

The redundancy of a basis is always one:

```
F=frame('wmdct','gauss',40);
framered(F)
```

*This code produces the following output:*

```
ans =
```

### 7.4.3 FRAMELENGTH - Frame length from signal

#### Usage

```
L=framelength(F,Ls);
```

#### Description

`framelength(F,Ls)` returns the length of the frame  $F$ , such that  $F$  is long enough to expand a signal of length  $L_s$ .

If the frame length is longer than the signal length, the signal will be zero-padded by frana.

If instead a set of coefficients are given, call `framelengthcoef`.

### 7.4.4 FRAMELENGTHCOEF - Frame length from coefficients

#### Usage

```
L=framelengthcoef(F,Ncoef);
```

#### Description

`framelengthcoef(F,Ncoef)` returns the length of the frame  $F$ , such that  $F$  is long enough to expand the coefficients of length  $N_{coef}$ .

If instead a signal is given, call `framelength`.

### 7.4.5 FRAMECLLENGTH - Number of coefficients from length of signal

#### Usage

```
Ncoef=framecllength(F,Ls);
[Ncoef,L]=framecllength(...);
```

#### Description

`Ncoef=framecllength(F,Ls)` returns the total number of coefficients obtained by applying the analysis operator of frame  $F$  to a signal of length  $L_s$  i.e. `size(frana(F,f),1)` for  $L_s=\text{length}(f)$ .

`[Ncoef,L]=framecllength(F,Ls)` additionally returns  $L$ , which is the same as returned by `framelength`.

If the frame length  $L$  is longer than the signal length  $L_s$ , the signal will be zero-padded to  $L$  by frana.

## 7.5 Coefficients conversions

### 7.5.1 FRAMECOEF2NATIVE - Convert coefficients to native format

#### Usage

```
coef=framecoef2native(F,coef);
```

#### Description

`framecoef2native(F,coef)` converts the frame coefficients  $coef$  into the native coefficient format of the frame. The frame object  $F$  must have been created using `frame`.

### 7.5.2 FRAMENATIVE2COEF - Convert coefficient from native format

#### Usage

```
coef=framenative2coef(F,coef);
```

**Description**

`framenative2coef(F, coef)` converts the frame coefficients from the native format of the transform into the common column format.

**7.5.3 FRAMECOEF2TF - Convert coefficients to time-frequency plane****Usage**

```
coef=framecoef2tf(F,coef);
```

**Description**

`framecoef2tf(F, coef)` converts the frame coefficients *coef* into the time-frequency plane layout. The frame object *F* must have been created using `frame`.

The time-frequency plane layout is a matrix, where the first dimension indexes frequency and the second dimension time. This is similar to the output format from `dgt` and `wmdct`.

Not all frame types support this coefficient layout. The supported frame types are: '`dgt`', '`dgtreal`', '`dwilt`', '`wmdct`', '`ufilterbank`', '`ufwt`', '`uwfbt`' and '`uwpfbt`'.

**7.5.4 FRAMETF2COEF - Convert coefficients from TF-plane format****Usage**

```
coef=frametf2coef(F,coef);
```

**Description**

`frametf2coef(F, coef)` converts the frame coefficients from the time-frequency plane layout into the common column format.

**7.5.5 FRAMECOEF2TFPLOT - Convert coefficients to time-frequency plane matrix****Usage**

```
cout=framecoef2tfplot(F,cin);
```

**Description**

`framecoef2tfplot(F, coef)` converts the frame coefficients *coef* into the time-frequency plane layout matrix. The frame object *F* must have been created using `frame`. The function acts exactly as `framecoef2tf` for frames which admit regular (rectangular) sampling of a time-frequency plane and converts irregularly sampled coefficients to a rectangular matrix. This is useful for custom plotting.

**7.6 Non-linear analysis and synthesis****7.6.1 FRANABP - Frame Analysis Basis Pursuit****Usage**

```
c = franabp(F,f)
c = franabp(F,f,lambda,C,tol,maxit)
[c,relres,iter,frec,cd] = franabp(...)
```

**Input parameters**

<b>F</b>	Frame definition
<b>f</b>	Input signal
<b>lambda</b>	Regularisation parameter.
<b>C</b>	Step size of the algorithm.
<b>tol</b>	Reative error tolerance.
<b>maxit</b>	Maximum number of iterations.

**Output parameters**

<b>c</b>	Sparse coefficients.
<b>relres</b>	Last relative error.
<b>iter</b>	Number of iterations done.
<b>frec</b>	Reconstructed signal such that $frec = frsyn(F,c)$
<b>cd</b>	The min $\ c\ _2$ solution using the canonical dual frame.

**Description**

`c = franabp (F, f)` solves the basis pursuit problem

$$\operatorname{argmin}_c \|c\|_1$$

$$\text{subject to } Fc = f$$

for a general frame  $F$  using SALSA (Split Augmented Lagrangian Srinkage algorithm) which is an appication of ADMM (Alternating Direction Method of Multipliers) to the basis pursuit problem.

The algorithm given  $F$  and  $f$  and parameters  $C > 0$ ,  $\lambda > 0$  (see below) acts as follows:

```

Initialize c,d
repeat
    v <- soft (c+d, lambda/C) - d
    d <- F*(FF*)^(-1) (f - Fv)
    c <- d + v
end

```

When compared to other algorithms,  $Fc = f$  holds exactly (up to a num. prec) in each iteration.

For a quick execution, the function requires analysis operator of the canonical dual frame  $F^*(FF^*)^{(-1)}$ . By default, the function attempts to call framedual to create the canonical dual frame explicitly. If it is not available, the conjugate gradient method algorithm is used for inverting the frame operator in each iteration of the algorithm. Optionally, the canonical dual frame object or an anonymous function acting as the analysis operator of the canonical dual frame can be passed as a key-value pair 'Fd', Fd see below.

**Optional positional parameters (**lambda,C,tol,maxit**)**

**lambda** A parameter for weighting coefficients in the objective function. For  $\lambda \approx 1$  the basis pursuit problem changes to

$$\arg \min_c \|\lambda c\|_1$$

subject to  $Fc = f$

`lambda` can either be a scalar or a vector of the same length as `c` (in such case the product is carried out elementwise). One can obtain length of `c` from length of `f` by `frameclength`. `framecoef2native` and `framenative2coef` will help with defining weights specific to some regions of coefficients (e.g. channel-specific weighting can be achieved this way). The default value of `lambda` is 1.

**C** A step parameter of the SALSA algorithm. The default value of `C` is the upper frame bound of `F`. Depending on the structure of the frame, this can be an expensive operation.

**tol** Defines tolerance of `relres` which is a norm or a relative difference of coefficients obtained in two consecutive iterations of the algorithm. The default value 1e-2.

**maxit** Maximum number of iterations to do. The default value is 100.

### Other optional parameters

Key-value pairs:

**'Fd'**, `Fd` A canonical dual frame object or an anonymous function acting as the analysis operator of the canonical dual frame.

**'printstep'**, `printstep` Print current status every `printstep` iteration.

Flag groups (first one listed is the default):

**'print'**, **'quiet'** Enables/disables printing of notifications.

**'zeros'**, **'frana'** Starting point of the algorithm. With '`zeros`' enabled, the algorithm starts from coefficients set to zero, with '`frana`' the algorithm starts from `c=frana(F, f)`.

### Returned arguments:

`[c, relres, iter] = franabp(...)` returns the residuals `relres` in a vector and the number of iteration steps done `iter`.

`[c, relres, iter, freq, cd] = franabp(...)` returns the reconstructed signal from the coefficients, `freq` (this requires additional computations) and a coefficients `cd` minimising the  $\|c\|_2$  norm (this is a byproduct of the algorithm).

The relationship between the output coefficients `freq` and `c` is given by

```
freq = frsyn(F, c);
```

And `cd` and `f` by

```
cd = frana(framedual(F), f);
```

### Examples:

The following example shows how `franabp` produces a sparse representation of a test signal `greasy` still maintaining a perfect reconstruction:

```
f = greasy;
% Gabor frame with redundancy 8
F = frame('dgtreal','gauss',64,512);
% Solve the basis pursuit problem
[c,~,~,freq,cd] = franabp(F,f);
% Plot sparse coefficients
```

```

figure(1);
plotframe(F,c,'dynrange',50);

% Plot coefficients obtained by applying an analysis operator of a
% dual Gabor system to *f*
figure(2);
plotframe(F,cd,'dynrange',50);

% Check the reconstruction error (should be close to zero).
% frec is obtained by applying the synthesis operator of frame *F*
% to sparse coefficients *c*.
norm(f-frec)

% Compare decay of coefficients sorted by absolute values
% (compressibility of coefficients)
figure(3);
semilogx([sort(abs(c),'descend')/max(abs(c)),...
sort(abs(cd),'descend')/max(abs(cd))]);
legend({'sparsified coefficients','dual system coefficients'});

```

*This code produces the following output:*

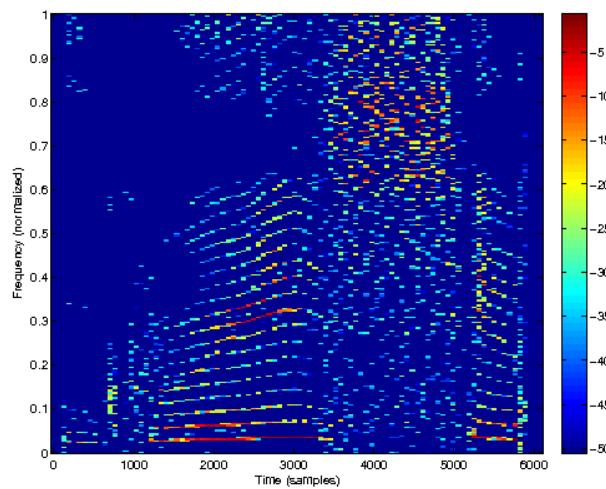
```

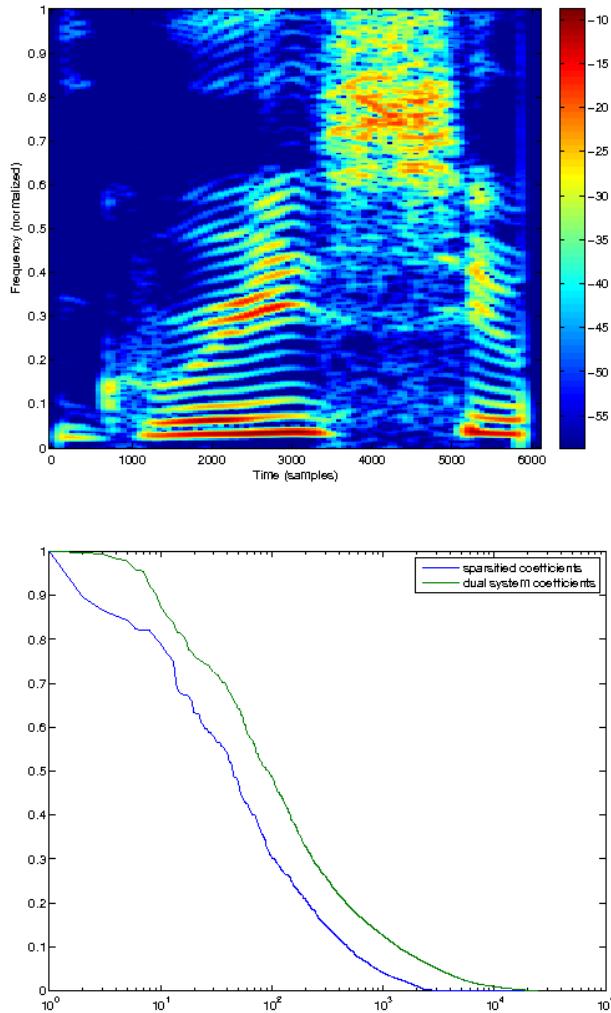
Iteration 10: relative error = 0.068385
Iteration 20: relative error = 0.034517
Iteration 30: relative error = 0.022865
Iteration 40: relative error = 0.016747
Iteration 50: relative error = 0.013124
Iteration 60: relative error = 0.010508

```

```
ans =
```

```
2.2797e-14
```





**References:** [75], [16]

### 7.6.2 FRANALASSO - Frame LASSO regression

#### Usage

```
tc = franalasso(F,f,lambda)
tc = franalasso(F,f,lambda,C,tol,maxit)
[tc,relres,iter,frec,cd] = franalasso(....)
```

#### Input parameters

<b>F</b>	Frame definition
<b>f</b>	Input signal
<b>lambda</b>	Regularisation parameter, controls sparsity of the solution
<b>C</b>	Step size of the algorithm.
<b>tol</b>	Reative error tolerance.
<b>maxit</b>	Maximum number of iterations.

### Output parameters

<b>tc</b>	Thresholded coefficients
<b>relres</b>	Vector of residuals.
<b>iter</b>	Number of iterations done.
<b>frec</b>	Reconstructed signal
<b>cd</b>	The min $\ c\ _2$ solution using the canonical dual frame.

### Description

`franalasso(F, f, lambda)` solves the LASSO (or basis pursuit denoising) regression problem for a general frame: minimize a functional of the synthesis coefficients defined as the sum of half the  $l^2$  norm of the approximation error and the  $l^1$  norm of the coefficient sequence, with a penalization coefficient `lambda` such that

$$\operatorname{argmin}_c \lambda \|c\|_1 + \frac{1}{2} \|Fc - f\|_2^2$$

The solution is obtained via an iterative procedure, called Landweber iteration, involving iterative soft thresholdings.

The following flags determining an algorithm to be used are recognized at the end of the argument list:

**'ista'** The basic (Iterative Soft Thresholding) algorithm given `F` and `f` and parameters  $C > 0$ , `lambda > 0` acts as follows:

```
Initialize c
repeat until stopping criterion is met
    c <- soft(c + F*(f - Fc)/C, lambda/C)
end
```

**'fista'** The fast version of the previous. This is the default option.

```
Initialize c(0), z, tau(0)=1, n=1
repeat until stopping criterion is met
    c(n) <- soft(z + F*(f - Fz)/C, lambda/C)
    tau(n) <- (1+sqrt(1+4*tau(n-1)^2))/2
    z     <- c(n) + (c(n)-c(n-1))(tau(n-1)-1)/tau(n)
    n     <- n + 1
end
```

`[tc, relres, iter] = franalasso(...)` returns the residuals `relres` in a vector and the number of iteration steps done `iter`.

`[tc, relres, iter, frec, cd] = franalasso(...)` returns the reconstructed signal from the coefficients, `frec` and coefficients `cd` obtained by analysing using the canonical dual system. Note that this requires additional computations.

The relationship between the output coefficients is given by

```
frec = frsyn(F, tc);
```

The function takes the following optional parameters at the end of the line of input arguments:

**'C',eval** Landweber iteration parameter: must be larger than square of upper frame bound. Default value is the upper frame bound.

**'tol',tol** Stopping criterion: minimum relative difference between norms in two consecutive iterations. Default value is 1e-2.

**'maxit',maxit** Stopping criterion: maximal number of iterations to do. Default value is 100.

**'print'** Display the progress.

**'quiet'** Don't print anything, this is the default.

**'printstep',*p*** If 'print' is specified, then print every *p*'th iteration. Default value is 10;

The parameters *C*, *itermax* and *tol* may also be specified on the command line in that order: `franalasso(F, x, lambda, C, itermax, tol)`

**Note:** If you do not specify *C*, it will be obtained as the upper framebound. Depending on the structure of the frame, this can be an expensive operation.

### Examples:

The following example shows how `franalasso` produces a sparse representation of a test signal *greasy*:

```
f = greasy;
% Gabor frame with redundancy 8
F = frame('dgtreal','gauss',64,512);
% Choosing lambda (weight of the sparse regularization param.)
lambda = 0.1;
% Solve the basis pursuit problem
[c,~,~,frec,cd] = franalasso(F,f,lambda);
% Plot sparse coefficients
figure(1);
plotframe(F,c,'dynrange',50);

% Plot coefficients obtained by applying an analysis operator of a
% dual Gabor system to f
figure(2);
plotframe(F,cd,'dynrange',50);

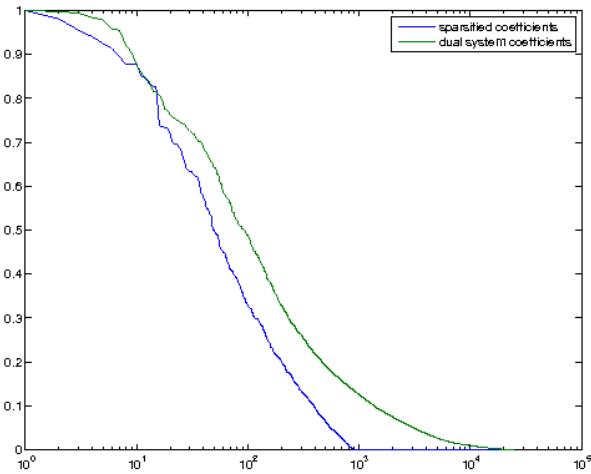
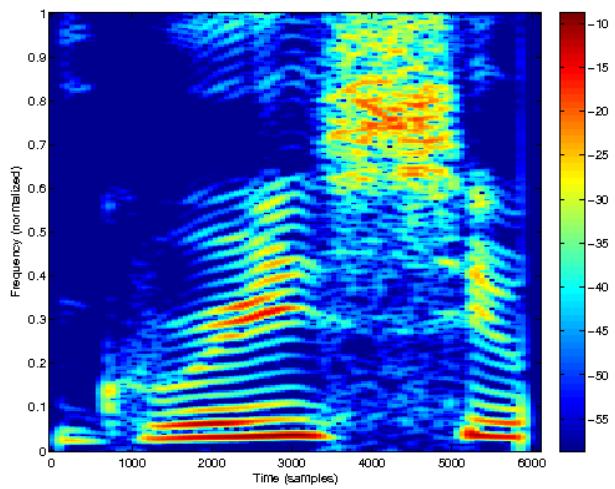
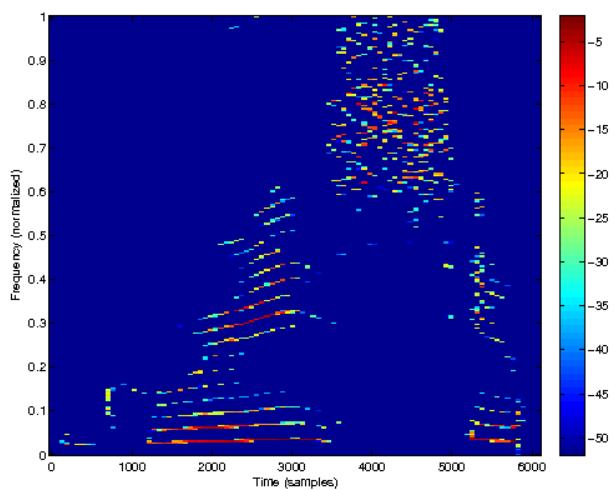
% Check the (NON-ZERO) reconstruction error .
% frec is obtained by applying the synthesis operator of frame F
% to sparse coefficients c.
norm(f-frec)

% Compare decay of coefficients sorted by absolute values
% (compressibility of coefficients)
figure(3);
semilogx([sort(abs(c),'descend')/max(abs(c)),...
sort(abs(cd),'descend')/max(abs(cd))]);
legend({'sparsified coefficients','dual system coefficients'});
```

This code produces the following output:

```
Iteration 10: relative error = 0.049585
Iteration 20: relative error = 0.032723
Iteration 30: relative error = 0.023291
Iteration 40: relative error = 0.015511
Iteration 50: relative error = 0.010411

ans =
    3.1636
```



**References:** [23], [11]

### 7.6.3 FRANAGROUPASSO - Group LASSO regression in the TF-domain

#### Usage

```
tc = franagroupasso(F,f,lambda)
tc = franagroupasso(F,f,lambda,C,tol,maxit)
[tc,relres,iter,frec] = franagroupasso(...)
```

#### Input parameters

<b>F</b>	Frame definition
<b>f</b>	Input signal
<b>lambda</b>	Regularisation parameter, controls sparsity of the solution
<b>C</b>	Step size of the algorithm.
<b>tol</b>	Reative error tolerance.
<b>maxit</b>	Maximum number of iterations.

#### Output parameters

<b>tc</b>	Thresholded coefficients
<b>relres</b>	Vector of residuals.
<b>iter</b>	Number of iterations done.
<b>frec</b>	Reconstructed signal

#### Description

`franagroupasso(F, f, lambda)` solves the group LASSO regression problem in the time-frequency domain: minimize a functional of the synthesis coefficients defined as the sum of half the  $l^2$  norm of the approximation error and the mixed  $l^1 / l^2$  norm of the coefficient sequence, with a penalization coefficient `lambda`.

The matrix of time-frequency coefficients is labelled in terms of groups and members. By default, the obtained expansion is sparse in terms of groups, no sparsity being imposed to the members of a given group. This is achieved by a regularization term composed of  $l^2$  norm within a group, and  $l^1$  norm with respect to groups. See the help on `groupthresh` for more information.

**Note** the involved frame `F` must support regular time-frequency layout of coefficients.

`[tc, relres, iter] = franagroupasso(...)` returns the residuals `relres` in a vector and the number of iteration steps done, `maxit`.

`[tc, relres, iter, frec] = franagroupasso(...)` returns the reconstructed signal from the coefficients, `frec`. Note that this requires additional computations.

The function takes the following optional parameters at the end of the line of input arguments:

<b>'freq'</b>	Group in frequency (search for tonal components). This is the default.
<b>'time'</b>	Group in time (search for transient components).
<b>'C',eval</b>	Landweber iteration parameter: must be larger than square of upper frame bound. Default value is the upper frame bound.
<b>'maxit',maxit</b>	Stopping criterion: maximal number of iterations. Default value is 100.
<b>'tol',tol</b>	Stopping criterion: minimum relative difference between norms in two consecutive iterations. Default value is 1e-2.
<b>'print'</b>	Display the progress.

<b>'quiet'</b>	Don't print anything, this is the default.
<b>'printstep',p</b>	If 'print' is specified, then print every p'th iteration. Default value is 10;

In addition to these parameters, this function accepts all flags from the groupthresh and thresh functions. This makes it possible to switch the grouping mechanism or inner thresholding type.

The parameters  $C$ ,  $maxit$  and  $tol$  may also be specified on the command line in that order: `franagrouplasso(F, x, lambda, C, maxit, tol)`.

The solution is obtained via an iterative procedure, called Landweber iteration, involving iterative group thresholdings.

The relationship between the output coefficients is given by

```
frec = frsyn(F, tc);
```

**References:** [48], [47]

#### 7.6.4 FRSYNABS - Reconstruction from magnitude of coefficients

##### Usage

```
f=frsynabs(F, s);
f=frsynabs(F, s, Ls);
[f, relres, iter]=frsynabs(...);
```

##### Input parameters

<b>F</b>	Frame
<b>s</b>	Array of coefficients.
<b>Ls</b>	length of signal.

##### Output parameters

<b>f</b>	Signal.
<b>relres</b>	Vector of residuals.
<b>iter</b>	Number of iterations done.

##### Description

`frsynabs(F, s)` attempts to find a signal which has  $s$  as the absolute value of its frame coefficients

```
s = abs(fрана(F, f));
```

using an iterative method.

`frsynabs(F, s, Ls)` does as above but cuts or extends  $f$  to length  $Ls$ .

If the phase of the coefficients  $s$  is known, it is much better to use `frsyn`.

`[f, relres, iter]=frsynabs(...)` additionally returns the residuals in a vector  $relres$  and the number of iteration steps  $iter$ . The residuals are computed as:

$$relres = \frac{\| |c_n| - s \|_{fro}}{\| s \|_{fro}},$$

where  $c_n$  is the Gabor coefficients of the signal in iteration  $n$ .

Generally, if the absolute value of the frame coefficients has not been modified, the iterative algorithm will converge slowly to the correct result. If the coefficients have been modified, the algorithm is not guaranteed to converge at all.

`frsynabs` takes the following parameters at the end of the line of input arguments:

<b>'input'</b>	Choose the starting phase as the phase of the input $s$ . This is the default
----------------	---

<b>'zero'</b>	Choose a starting phase of zero.
<b>'rand'</b>	Choose a random starting phase.
<b>'griflim'</b>	Use the Griffin-Lim iterative method. This is the default.
<b>'fgriflim'</b>	Use the Fast Griffin-Lim iterative method.
<b>'bfgs'</b>	Use the limited-memory Broyden Fletcher Goldfarb Shanno (BFGS) method.
<b>'alpha',a</b>	Parameter of the Fast Griffin-Lim algorithm. It is ignored if not used together with 'fgriflim' flag.
<b>'tol',t</b>	Stop if relative residual error is less than the specified tolerance.
<b>'maxit',n</b>	Do at most n iterations.
<b>'print'</b>	Display the progress.
<b>'quiet'</b>	Don't print anything, this is the default.
<b>'printstep',p</b>	If 'print' is specified, then print every p'th iteration. Default value is p=10;

**References:** [35], [64]

# Chapter 8

## LTFAT - Signal processing tools

### 8.1 General

#### 8.1.1 RMS - RMS value of signal

##### Usage

```
y = rms(f);  
y = rms(f, ...);
```

##### Description

`RMS(f)` computes the RMS (Root Mean Square) value of a finite sampled signal sampled at a uniform sampling rate. This is a vector norm equal to the  $l^2$  averaged by the length of the signal.

If the input is a matrix or ND-array, the RMS is computed along the first (non-singleton) dimension, and a vector of values is returned.

The RMS value of a signal  $x$  of length  $N$  is computed by

$$rms(f) = \frac{1}{\sqrt{N}} \left( \sum_{n=1}^N |f(n)|^2 \right)^{\frac{1}{2}}$$

RMS takes the following flags at the end of the line of input parameters:

<code>'ac'</code>	Consider only the AC component of the signal (i.e. the mean is removed).
<code>'dim',d</code>	Work along specified dimension. The default value of <code>[]</code> means to work along the first non-singleton one.

#### 8.1.2 NORMALIZE - Normalize input signal by specified norm

##### Usage

```
h=normalize(f, ...);
```

##### Description

`normalize(f, ...)` will normalize the signal  $f$  by the specified norm.

The norm is specified as a string and may be one of:

<code>'1'</code>	Normalize the $l^1$ norm to be 1.
<code>'area'</code>	Normalize the area of the signal to be 1. This is exactly the same as ' <code>1</code> '.

'2'	Normalize the $l^2$ norm to be 1.
'energy'	Normalize the energy of the signal to be 1. This is exactly the same as '2'.
'inf'	Normalize the $l^\infty$ norm to be 1.
'peak'	Normalize the peak value of the signal to be 1. This is exactly the same as 'inf'.
'rms'	Normalize the Root Mean Square (RMS) norm of the signal to be 1.
's0'	Normalize the S0-norm to be 1.
'wav'	Normalize to the $l^\infty$ norm to be 0.99 to avoid possible clipping introduced by the quantization procedure when saving as a wav file. This only works with floating point data types.
'null'	Do NOT normalize, output is identical to input.

It is possible to specify the dimension:

**'dim',d** Work along specified dimension. The default value of [] means to work along the first non-singleton one.

### 8.1.3 GAINDB - Increase/decrease level of signal

#### Usage

```
outsig = gaindb(insig,gn);
```

#### Description

`gaindb(insig,gn)` increases the energy level of the signal by  $gn$  dB.

If  $gn$  is a scalar, the whole input signal is scaled.

If  $gn$  is a vector, each column is scaled by the entries in  $gn$ . The length of  $gn$  must match the number of columns.

`gaindb(insig,gn,dim)` scales the signal along dimension  $dim$ .

### 8.1.4 CRESTFACTOR - Crest factor of input signal in dB

#### Usage

```
c=crestfactor(insig);
```

#### Description

`crestfactor(insig)` computes the crest factor of the input signal *insig*. The output is measured in dB.

### 8.1.5 UQUANT - Simulate uniform quantization

#### Usage

```
x=uquant(x);
x=uquant(x,nbits,xmax,...);
```

**Description**

`uquant (x, nbits, xmax)` simulates the effect of uniform quantization of  $x$  using  $nbits$  bits. The output is simply  $x$  rounded to  $2^{nbits}$  different values. The  $xmax$  parameters specify the maximal value that should be quantifiable.

`uquant (x, nbits)` assumes a maximal quantifiable value of 1.

`uquant (x)` additionally assumes 8 bit quantization.

`uquant` takes the following flags at the end of the input arguments:

<b>'nbits'</b>	Number of bits to use in the quantization. Default is 8.
<b>'xmax'</b>	Maximal quantifiable value. Default is 1.
<b>'s'</b>	Use signed quantization. This assumes that the signal has a both positive and negative part. Useful for sound signals. This is the default.
<b>'u'</b>	Use unsigned quantization. Assumes the signal is positive. Negative values are silently rounded to zero. Useful for images.

If this function is applied to a complex signal, it will be applied to the real and imaginary part separately.

## 8.2 Ramping

### 8.2.1 RAMPUP - Rising ramp function

**Usage**

```
outsig=rampup (L);
```

**Description**

`rampup (L)` will return a rising ramp function of length  $L$ . The ramp is a sinusoid starting from zero and ending at one. The ramp is centered such that the first element is always 0 and the last element is not quite 1, such that the ramp fits with following ones.

`rampup (L, wintype)` will use another window for ramping. This may be any of the window types from firwin. Please see the help on firwin for more information. The default is to use a piece of the Hann window.

### 8.2.2 RAMPDOWN - Falling ramp function

**Usage**

```
outsig=rampdown (siglen);
```

**Description**

`rampdown (siglen)` will return a falling ramp function of length  $siglen$ . The ramp is a sinusoid starting from one and ending at zero. The ramp is centered such that the first element is always one and the last element is not quite zero, such that the ramp fits with following zeros.

`rampdown (L, wintype)` will use another window for ramping. This may be any of the window types from firwin. Please see the help on firwin for more information. The default is to use a piece of the Hann window.

### 8.2.3 RAMPSIGNAL - Ramp signal

**Usage**

```
outsig=rampsignal (insig, L);
```

### Description

`rampsignal(insig,L)` applies a ramp function of length  $L$  to the beginning and the end of the input signal. The default ramp is a sinusoide starting from zero and ending at one (also known as a cosine squared ramp).

If  $L$  is scalar, the starting and ending ramps will be of the same length. If  $L$  is a vector of length 2, the first entry will be used for the rising ramp, and the second for the falling.

If the input is a matrix or an N-D array, the ramp will be applied along the first non-singleton dimension. `rampsignal(insig)` will use a ramp length of half the signal.

`rampsignal(insig,L,wintype)` will use another window for ramping. This may be any of the window types from `firwin`. Please see the help on `firwin` for more information. The default is to use a piece of the Hann window.

`rampsignal` accepts the following optional parameters:

<b>'dim',d</b>	Apply the ramp along dimension d. The default value of [] means to use the first non-singleton dimension.
----------------	---

## 8.3 Thresholding methods

### 8.3.1 THRESH - Coefficient thresholding

#### Usage

```
x=thresh(x,lambda,...);
[x,N]=thresh(x,lambda,...);
```

#### Description

`thresh(x,lambda)` will perform hard thresholding on  $x$ , i.e. all elements with absolute value less than scalar  $lambda$  will be set to zero.

`thresh(x,lambda,'soft')` will perform soft thresholding on  $x$ , i.e.  $lambda$  will be subtracted from the absolute value of every element of  $x$ .

The  $lambda$  parameter can also be a vector with number of elements equal to `numel(xi)` or it can be a numeric array of the same shape as  $xi$ .  $lambda$  is then applied element-wise and in a column major order if  $lambda$  is a vector.

`[x,N]=thresh(x,lambda)` additionally returns a number  $N$  specifying how many numbers where kept.

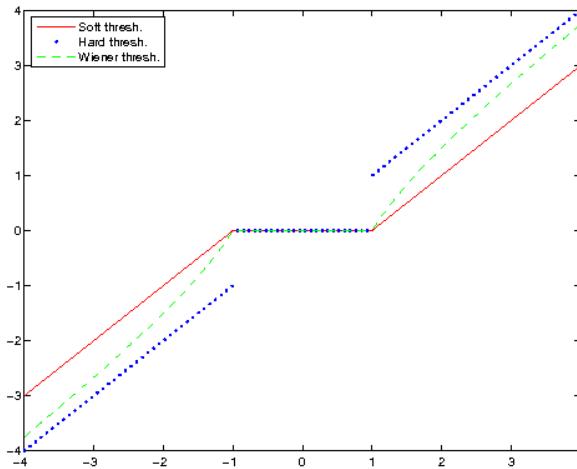
`thresh` takes the following flags at the end of the line of input arguments:

<b>'hard'</b>	Perform hard thresholding. This is the default.
<b>'wiener'</b>	Perform empirical Wiener shrinkage. This is in between soft and hard thresholding.
<b>'soft'</b>	Perform soft thresholding.
<b>'full'</b>	Returns the output as a full matrix. This is the default.
<b>'sparse'</b>	Returns the output as a sparse matrix.

The function `wthresh` in the Matlab Wavelet toolbox implements some of the same functionality.

The following code produces a plot to demonstrate the difference between hard and soft thresholding for a simple linear input:

```
t=linspace(-4,4,100);
plot(t,thresh(t,1,'soft'),'r',...
      t,thresh(t,1,'hard'),'b',...
      t,thresh(t,1,'wiener'),'-g');
legend('Soft thresh.','Hard thresh.','Wiener thresh.','Location','NorthWest');
```



**References:** [50], [32]

### 8.3.2 LARGESTR - Keep fixed ratio of largest coefficients

#### Usage

```
xo=largestr(x,p);
xo=largestr(x,p,mtype);
[xo,N]=largestr(...);
```

#### Description

`largestr(x,p)` returns an array of the same size as  $x$  keeping the fraction  $p$  of the coefficients. The coefficients with the largest magnitude are kept.

`[xo,n]=largestr(xi,p)` additionally returns the number of coefficients kept.

**Note:** If the function is used on coefficients coming from a redundant transform or from a transform where the input signal was padded, the coefficient array will be larger than the original input signal. Therefore, the number of coefficients kept might be higher than expected.

`largestr` takes the following flags at the end of the line of input arguments:

'hard'	Perform hard thresholding. This is the default.
'wiener'	Perform empirical Wiener shrinkage. This is in between soft and hard thresholding.
'soft'	Perform soft thresholding.
'full'	Returns the output as a full matrix. This is the default.
'sparse'	Returns the output as a sparse matrix.

**Note:** If soft- or Wiener thresholding is selected, one less coefficient will actually be returned. This is caused by that coefficient being set to zero.

**References:** [54]

### 8.3.3 LARGESTN - Keep N largest coefficients

#### Usage

```
xo=largestn(x,N);
xo=largestn(x,N,mtype);
```

**Description**

`largestn(x, N)` returns an array of the same size as  $x$  keeping the  $N$  largest coefficients.

`largestn` takes the following flags at the end of the line of input arguments:

'hard'	Perform hard thresholding. This is the default.
'wiener'	Perform empirical Wiener shrinkage. This is in between soft and hard thresholding.
'soft'	Perform soft thresholding.
'full'	Returns the output as a full matrix. This is the default.
'sparse'	Returns the output as a sparse matrix.

If the coefficients represents a signal expanded in an orthonormal basis then this will be the best  $N$ -term approximation.

**Note:** If soft- or Wiener thresholding is selected, only  $N - 1$  coefficients will actually be returned. This is caused by the  $N$ 'th coefficient being set to zero.

**References:** [54]

**8.3.4 DYNLIMIT - Limit the dynamical range of the input****Usage**

```
xo=dynlimit(xi,dynrange);
```

**Description**

`dynlimit(xi, dynrange)` will threshold the input such that the difference between the maximum and minimum value of  $xi$  is exactly  $dynrange$ .

**8.3.5 GROUPTHRESH - Group thresholding****Usage**

```
xo=groupthresh(xi,lambda);
```

**Description**

`groupthresh(x, lambda)` performs group thresholding on  $x$ , with threshold  $lambda$ .  $x$  must be a two-dimensional array, the first dimension labelling groups, and the second one labelling members. This means that the groups are the row vectors of the input (the vectors along the 2nd dimension).

Several types of grouping behaviour are available:

- `groupthresh(x, lambda, 'group')` shrinks all coefficients within a given group according to the value of the  $l^2$  norm of the group in comparison to the threshold  $lambda$ . This is the default.
- `groupthresh(x, lambda, 'elite')` shrinks all coefficients within a given group according to the value of the  $l^1$  norm of the group in comparison to the threshold value  $lambda$ .

`groupthresh(x, lambda, dim)` chooses groups along dimension  $dim$ . The default value is  $dim = 2$ .

`groupthresh` accepts all the flags of `thresh` to choose the thresholding type within each group and the output type (full / sparse matrix). Please see the help of `thresh` for the available options. Default is to use soft thresholding and full matrix output.

**References:** [48], [47], [89]

## 8.4 Image processing

### 8.4.1 RGB2JPEG - Converts from RGB format to the YCbCr format used by JPEG

#### Usage

```
YCbCr = rgb2jpeg(RGB);
```

#### Input parameters

<b>RGB</b>	3d data-cube, containing RGB information of the image
------------	---

#### Output parameters

<b>YCbCr</b>	3d data-cube, containing the YCbCr information of the image
--------------	---

#### Description

'rgb2jpeg(RGB)' performs a transformation of the 3d data-cube *RGB* with dimensions  $N \times M \times 3$ , which contains information of the colours "red", "green" and "blue". The output variable *YCbCr* is a 3d data-cube of the same size containing information about "luminance", "chrominance blue" and "chrominance red". The output will be of the *uint8* type.

See <http://en.wikipedia.org/wiki/YCbCr> and <http://de.wikipedia.org/wiki/JPEG>.

#### Examples:

In the following example, the Lichtenstein test image is split into its three components. The very first subplot is the original image:

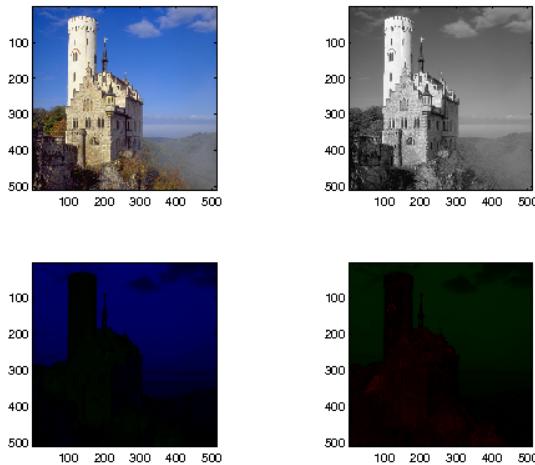
```
f=lichtenstein;
f_jpeg=rgb2jpeg(f);

subplot(2,2,1);
image(f);axis('image');

Ymono=zeros(512,512,3,'uint8');
Ymono(:,:,1)=f_jpeg(:,:,1);
Ymono(:,:,2:3)=128;
fmono=jpeg2rgb(Ymono);
subplot(2,2,2);
image(fmono);axis('image');

Cbmono=zeros(512,512,3,'uint8');
Cbmono(:,:,2)=f_jpeg(:,:,2);
Cbmono(:,:,3)=128;
fmono=jpeg2rgb(Cbmono);
subplot(2,2,3);
image(fmono);axis('image');

Crmono=zeros(512,512,3,'uint8');
Crmono(:,:,3)=f_jpeg(:,:,3);
Crmono(:,:,2)=128;
fmono=jpeg2rgb(Crmono);
subplot(2,2,4);
image(fmono);axis('image');
```



### 8.4.2 JPEG2RGB - Converts from RGB format to YCbCr format

#### Usage

```
RGB = jpeg2rgb(YCbCr);
```

#### Input parameters

<b>YCbCr</b>	3d data-cube, containing the YCbCr information of the image
--------------	---

#### Output parameters

<b>RGB</b>	3d data-cube, containing RGB information of the image
------------	---

#### Description

'jpeg2rgb(YCbCr)' performs a transformation of the 3d data-cube  $YCbCr$  with dimensions  $N \times M \times 3$ , which contains information of "luminance", "chrominance blue" and "chrominance red". The output variable  $RGB$  is a 3d data-cube of the same size containing information about the colours "red", "green" and "blue". The output will be of the `uint8` type.

For more information, see <http://en.wikipedia.org/wiki/YCbCr> and <http://de.wikipedia.org/wiki/JPEG>.

## 8.5 Tools for OFDM

### 8.5.1 QAM4 - Quadrature amplitude modulation of order 4

#### Usage

```
xo=qam4(xi);
```

#### Description

`qam4(xi)` converts a vector of 0's and 1's into the complex roots of unity (QAM4 modulation). Every 2 input coefficients are mapped into 1 output coefficient.

### 8.5.2 IQAM4 - Inverse QAM of order 4

`iqam4(xi)` demodulates a signal mapping the input coefficients to the closest complex root of unity, and returning the associated bit pattern. This is the inverse operation of `qam4`.



# Chapter 9

## LTFAT - Simple auditory processing

### 9.1 Plots

#### 9.1.1 SEMIAUDPLOT - 2D plot on auditory scale

##### Usage

```
h=semiaudplot(x,y);
```

##### Description

semiaudplot ( $x, y$ ) plots the data ( $x, y$ ) on an auditory scale. By default the values of the x-axis will be shown on the Erb-scale.

semiaudplot takes the following parameters at the end of the line of input arguments:

'x'	Make the x-axis use the auditory scale. This is the default.
'y'	Make the y-axis use the auditory scale.
'opts',c	Pass options stored in a cell array onto the plot function.

In addition to these parameters, the auditory scale can be specified. All scales supported by freqtoaud are supported. The default is to use the erb-scale.

### 9.2 Auditory scales

#### 9.2.1 AUDTOFREQ - Converts auditory units to frequency (Hz)

##### Usage

```
freq = audtofreq(aud);
```

##### Description

audtofreq ( $aud, scale$ ) converts values on the selected auditory scale to frequencies measured in Hz.

See the help on freqtoaud to get a list of the supported values of the  $scale$  parameter. If no scale is given, the erb-scale will be selected by default.

#### 9.2.2 FREQTOAUD - Converts frequencies (Hz) to auditory scale units

##### Usage

```
aud = freqtoaud(freq,scale);
```

**Description**

`freqtoaud(freq, scale)` converts values on the frequency scale (measured in Hz) to values on the selected auditory scale. The value of the parameter *scale* determines the auditory scale:

<b>'erb'</b>	A distance of 1 erb is equal to the equivalent rectangular bandwidth of the auditory filters at that point on the frequency scale. The scale is normalized such that 0 erbs corresponds to 0 Hz. The width of the auditory filters were determined by a notched-noise experiment. The erb scale is defined in Glasberg and Moore (1990). This is the default.
<b>'mel'</b>	The mel scale is a perceptual scale of pitches judged by listeners to be equal in distance from one another. The reference point between this scale and normal frequency measurement is defined by equating a 1000 Hz tone, 40 dB above the listener's threshold, with a pitch of 1000 mels. The mel-scale is defined in Stevens et. al (1937).
<b>'mel1000'</b>	Alternative definition of the mel scale using a break frequency of 1000 Hz. This scale was reported in Fant (1968).
<b>'bark'</b>	The bark-scale is originally defined in Zwicker (1961). A distance of 1 on the bark scale is known as a critical band. The implementation provided in this function is described in Traunmuller (1990).
<b>'erb83'</b>	This is the original defintion of the erb scale given in Moore. et al. (1983).
<b>'freq'</b>	Return the frequency in Hz.

If no flag is given, the erb-scale will be selected.

**References:** [80], [91], [27], [33], [83], [59]

**9.2.3 AUDSPACE - Equidistantly spaced points on auditory scale****Usage**

```
y=audspace(flow,fhigh,n,scale);
```

**Description**

`audspace(flow, fhigh, n, scale)` computes a vector of length *n* containing values equidistantly scaled on the selected auditory scale between the frequencies *flow* and *fhigh*. All frequencies are specified in Hz.

See the help on `freqtoaud` to get a list of the supported values of the *scale* parameter.

`[y, bw]=audspace( . . . )` does the same but outputs the bandwidth between each sample measured on the selected scale.

**9.2.4 AUDSPACEBW - Auditory scale points specified by bandwidth****Usage**

```
y=audspacebw(flow,fhigh,bw,hitme);
y=audspacebw(flow,fhigh,bw);
y=audspacebw(flow,fhigh);
[y,n]=audspacebw( . . . );
```

**Description**

`audspacebw(flow, fhigh, bw, scale)` computes a vector containing values equistantly scaled between frequencies `flow` and `fhigh` on the selected auditory scale. All frequencies are specified in Hz. The distance between two consecutive values is `bw` on the selected scale, and the points will be centered on the scale between `flow` and `fhigh`.

See the help on `freqtoaud` to get a list of the supported values of the `scale` parameter.

`audspacebw(flow, fhigh, bw, hitme, scale)` will do as above, but one of the points is guaranteed to be the frequency `hitme`.

`[y, n] = audspacebw(...)` additionally returns the number of points `n` in the output vector `y`.

**9.2.5 ERBTOFREQ - Converts erb units to frequency (Hz)****Usage**

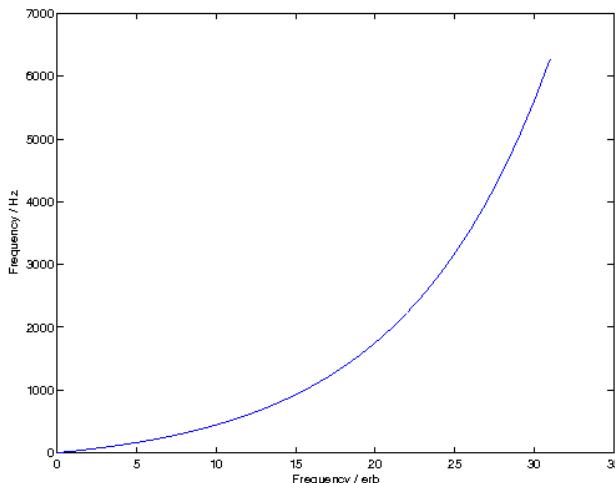
```
freq = erbtofreq(erb);
```

**Description**

This is a wrapper around `autofreq` that selects the erb-scale. Please see the help on `autofreq` for more information.

The following figure shows the corresponding frequencies for erb values up to 31:

```
erbs=0:31;
freqs=erbtofreq(erbs);
plot(erbs,freqs);
xlabel('Frequency / erb');
ylabel('Frequency / Hz');
```

**9.2.6 FREQTOERB - Converts frequencies (Hz) to erbs****Usage**

```
erb = freqtoerb(freq);
```

**Description**

This is a wrapper around `freqtoaud` that selects the erb-scale. Please see the help on `freqtoaud` for more information.

### 9.2.7 ERBSPACE - Equidistantly spaced points on erbscale

#### Usage

```
y=erbspace(flow,fhigh,n);
```

#### Description

This is a wrapper around audspace that selects the erb-scale. Please see the help on audspace for more information.

### 9.2.8 ERBSPACEBW - Erbscale points specified by bandwidth

#### Usage

```
y=erbspacebw(flow,fhigh,bw,hitme);
y=erbspacebw(flow,fhigh,bw);
y=erbspacebw(flow,fhigh);
```

#### Description

This is a wrapper around audspacebw that selects the erb-scale. Please see the help on audspacebw for more information.

### 9.2.9 AUDFILTBW - Bandwidth of auditory filter

#### Usage

```
bw = audfiltbw(fc)
```

#### Description

`audfiltbw(fc)` returns the equivalent rectangular bandwidth of the auditory filter at center frequency  $fc$ . The function uses the relation

$$bw = 24.7 + \frac{fc}{9.265}$$

as estimated in Glasberg and Moore (1990)

**References:** [33]

## 9.3 Range compression

### 9.3.1 RANGECOMPRESS - Compress the dynamic range of a signal

#### Usage

```
[outsig, sigweight] = rangecompress(insig,mu);
```

#### Description

`[outsig, sigweight]=rangecompress(insig,mu)` range-compresses the input signal `insig` using  $\mu$ -law range-compression with parameter `mu`.

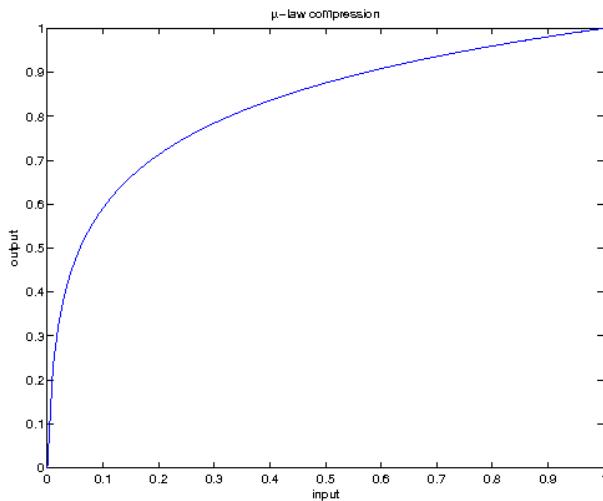
`rangecompress` takes the following optional arguments:

'mulaw'	Do mu-law compression, this is the default.
'alaw'	Do A-law compression.
'mu',mu	$\mu$ -law parameter. Default value is 255.

**'A',A** A-law parameter. Default value is 87.7.

The following plot shows how the output range is compressed for input values between 0 and 1:

```
x=linspace(0,1,100);
xc=rangecompress(x);
plot(x,xc);
xlabel('input');
ylabel('output');
title('\mu-law compression');
```



**References:** [43]

### 9.3.2 RANGEEXPAND - Expand the dynamic range of a signal

#### Usage

```
sig = rangeexpand(insig,mu,sigweight);
```

#### Description

`rangeexpand(insig,mu,sigweight)` inverts a previously applied  $\mu$ -law companding to the signal `insig`. The parameters `mu` and `sigweight` must match those from the call to `rangecompress`

`rangeexpand` takes the following optional arguments:

- 'mulaw'** Do mu-law compression, this is the default.
- 'alaw'** Do A-law compression.
- 'mu',mu**  $\mu$ -law parameter. Default value is 255.

**References:** [43]

## 9.4 Auditory filters

### 9.4.1 GAMMATONEFIR - Gammatone filter coefficients

#### Usage

```
b = gammatonefir(fc,fs,n,betamul);
b = gammatonefir(fc,fs,n);
b = gammatonefir(fc,fs);
```

**Input parameters**

<b>fc</b>	center frequency in Hz.
<b>fs</b>	sampling rate in Hz.
<b>n</b>	filter order.
<b>beta</b>	bandwidth of the filter.

**Output parameters**

<b>b</b>	FIR filters as an cell-array of structs.
----------	--

**Description**

`gammatonefir(fc, fs, n, betamul)` computes the filter coefficients of a digital FIR gammatone filter of length  $n$  with center frequency  $fc$ , 4th order rising slope, sampling rate  $fs$  and bandwidth determined by  $betamul$ . The bandwidth  $\beta$  of each filter is determined as  $betamul$  times `audfiltbw` of the center frequency of corresponding filter.

`gammatonefir(fc, fs, n)` will do the same but choose a filter bandwidth according to Glasberg and Moore (1990).  $betamul$  is chosen to be 1.0183.

`gammatonefir(fc, fs)` will do as above and choose a sufficiently long filter to accurately represent the lowest subband channel.

If  $fc$  is a vector, each entry of  $fc$  is considered as one center frequency, and the corresponding coefficients are returned as column vectors in the output.

The impulse response of the gammatone filter is given by

$$g(t) = at^{n-1} \cos(2\pi \cdot fc \cdot t) e^{-2\pi\beta \cdot t}$$

The gammatone filters as implemented by this function generate complex valued output, because the filters are modulated by the exponential function. Using `real` on the output will give the coefficients of the corresponding cosine modulated filters.

To create the filter coefficients of a 1-erb spaced filter bank using gammatone filters use the following construction:

```
g = gammatonefir(erbspacebw(flow, fhigh), fs);
```

**References:** [5], [33]

# Chapter 10

## LTFAT - Signals

### 10.1 Signal generators

#### 10.1.1 CTESTFUN - Complex 1-D test function

##### Usage

```
ftest=ctestfun(L);
```

##### Description

`ctestfun(L)` returns a test signal consisting of a superposition of a chirp and an indicator function.

#### 10.1.2 NOISE - Stochastic noise generator

##### Usage

```
outsig = noise(siglen,nsigs,type);
```

##### Input parameters

<b>siglen</b>	Length of the noise (samples)
<b>nsigs</b>	Number of signals (default is 1)
<b>type</b>	type of noise. See below.

##### Output parameters

<b>outsig</b>	$\text{siglen} \times \text{nsigs}$ signal vector
---------------	---

##### Description

`noise(siglen,nsigs)` generates  $nsigs$  channels containing white noise of the given type with the length of `siglen`. The signals are arranged as columns in the output. If only `siglen` is given, a column vector is returned.

`noise` takes the following optional parameters:

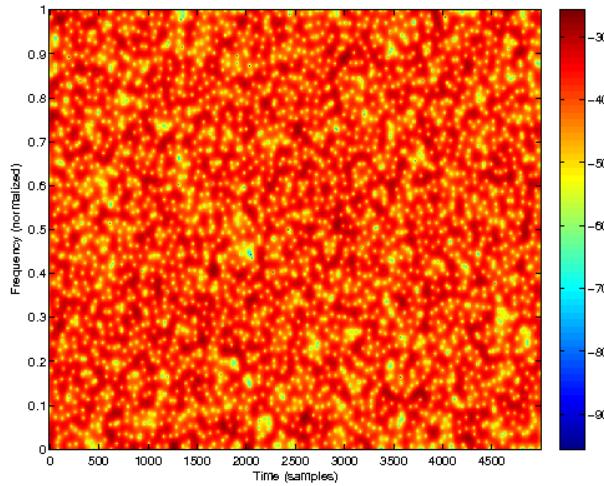
<b>'white'</b>	Generate white (gaussian) noise. This is the default.
<b>'pink'</b>	Generate pink noise.
<b>'brown'</b>	Generate brown noise.
<b>'red'</b>	This is the same as brown noise.

By default, the noise is normalized to have a unit energy, but this can be changed by passing a flag to normalize.

**Examples:**

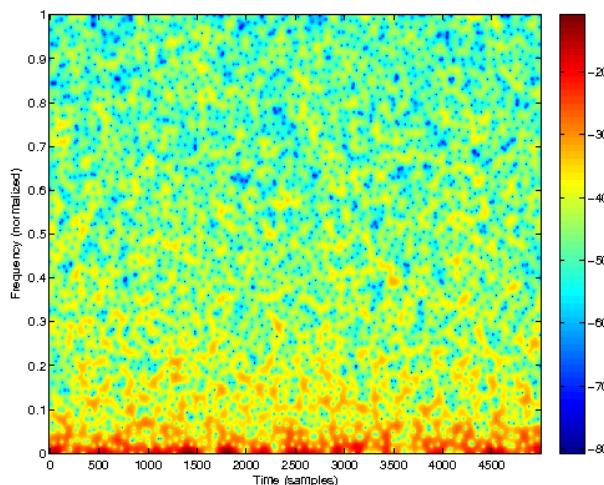
White noise in the time-frequency domain:

```
sgram(noise(5000,'white'),'dynrange',70);
```



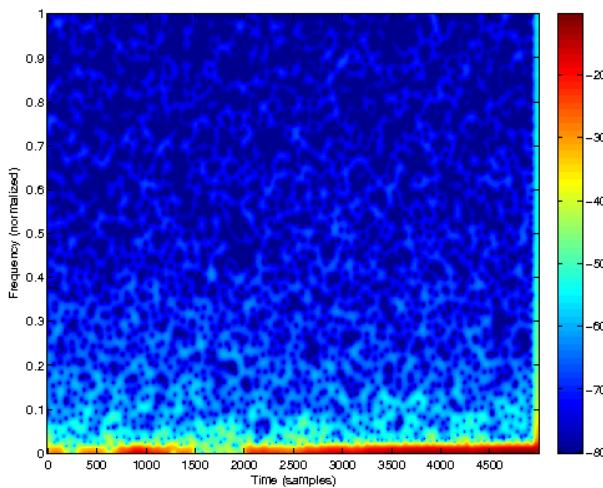
Pink noise in the time-frequency domain:

```
sgram(noise(5000,'pink'),'dynrange',70);
```



Brown/red noise in the time-frequency domain:

```
sgram(noise(5000,'brown'),'dynrange',70);
```



### 10.1.3 PINKNOISE - Generates a pink noise signal

#### Usage

```
outsig = pinknoise(siglen,nsigs);
```

#### Input parameters

<b>siglen</b>	Length of the noise (samples)
<b>nsigs</b>	Number of signals (default is 1)

#### Output parameters

<b>outsig</b>	$siglen \times nsigs$ signal vector
---------------	-------------------------------------

#### Description

`pinknoise(siglen,nsigs)` generates  $nsigs$  channels containing pink noise ( $1/f$  spectrum) with the length of `siglen`. The signals are arranged as columns in the output.

`pinknoise` is just a wrapper around `noise(...,'pink')`;

### 10.1.4 EXPCHIRP - Exponential chirp

#### Usage

```
outsig=expchirp(L,fstart,fend)
```

#### Description

`expchirp(L,fstart,fend)` computes an exponential chirp of length  $L$  starting at normalized frequency `fstart` and ending at frequency `fend`.

`expchirp` takes the following parameters at the end of the line of input arguments:

<b>'fs',fs</b>	Use a sampling frequency of $fs$ Hz. If this option is specified, <code>fstart</code> and <code>fend</code> will be measured in Hz.
<b>'phi',phi</b>	Starting phase of the chirp. Default value is 0.

## 10.2 Sound signals.

### 10.2.1 BAT - Load the 'bat' test signal

#### Usage

```
s=bat;
```

#### Description

`bat` loads the 'bat' signal. It is a 400 samples long recording of a bat chirp sampled with a sampling period of 7 microseconds. This gives a sampling rate of 143 kHz.

`[sig, fs]=bat` additionally returns the sampling frequency  $fs$ .

The signal can be obtained from <http://dsp.rice.edu/software/bat-echolocation-chirp>

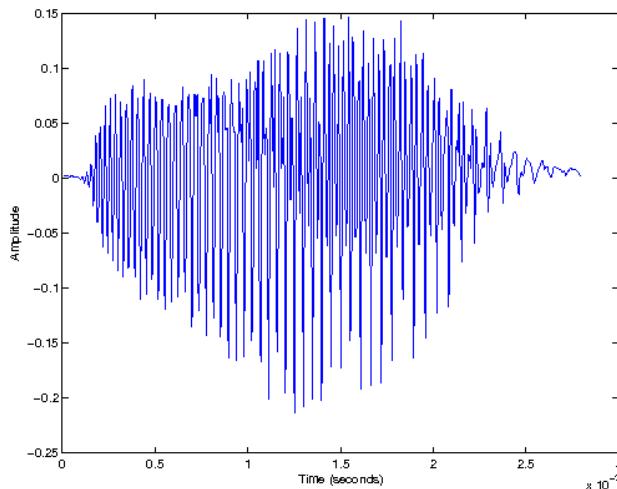
Please acknowledge use of this data in publications as follows:

The author wishes to thank Curtis Condon, Ken White, and Al Feng of the Beckman Institute of the University of Illinois for the bat data and for permission to use it in this paper.

#### Examples:

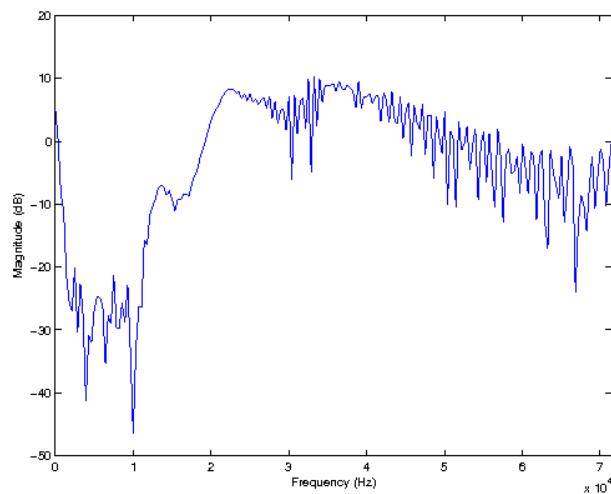
Plot of 'bat' in the time-domain:

```
plot((1:400)/143000,bat);
xlabel('Time (seconds)');
ylabel('Amplitude');
```



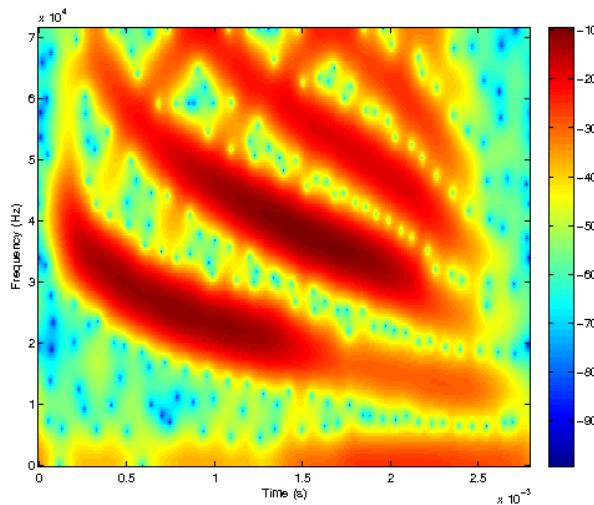
Plot of 'bat' in the frequency-domain:

```
plotfftreal(fftreal(bat),143000,90);
```



Plot of 'bat' in the time-frequency-domain:

```
sgram(bat, 143000, 90);
```



### 10.2.2 BATMASK - Load a Gabor multiplier symbol for the 'bat' test signal

#### Usage

```
c=batmask;
```

#### Description

batmask loads a Gabor multiplier with a 0/1 symbol that masks out the main contents of the 'bat' signal. The symbol fits a Gabor multiplier with lattice given by  $a = 10$  and  $M = 40$ .

The mask was created manually using a image processing program. The mask is symmetric, such that the result will be real valued if the multiplier is applied to a real valued signal using a real valued window.

### 10.2.3 GREASY - Load the 'greasy' test signal

#### Usage

```
s=greasy;
```

### Description

`greasy` loads the 'greasy' signal. It is a recording of a woman pronouncing the word "greasy".

The signal is 5880 samples long and recorded at 16 kHz with around 11 bits of effective quantization.

`[sig, fs]=greasy` additionally returns the sampling frequency  $fs$ .

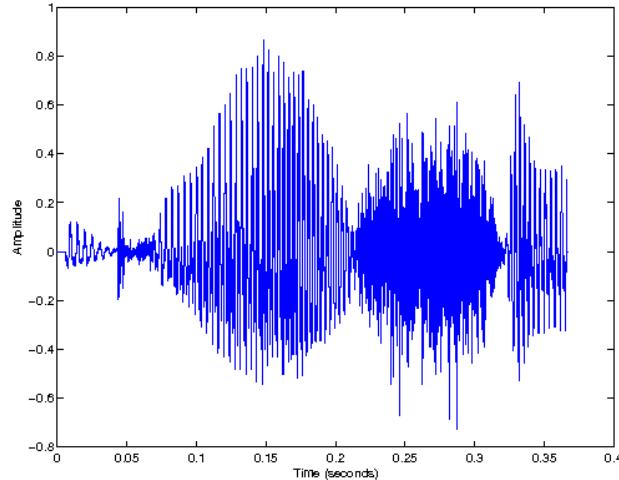
The signal has been scaled to not produce any clipping when played. To get integer values use `round(greasy*2048)`.

The signal was obtained from Wavelab: <http://www-stat.stanford.edu/~wavelab/>, it is a part of the first sentence of the TIMIT speech corpus "She had your dark suit in greasy wash water all year": <http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC93S1>.

### Examples:

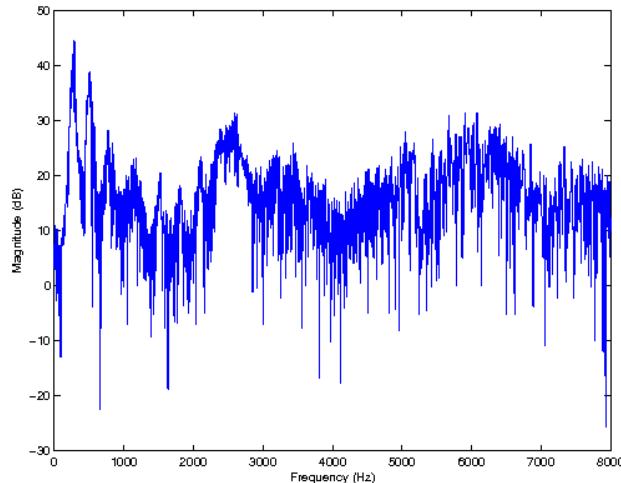
Plot of 'greasy' in the time-domain:

```
plot ((1:5880)/16000,greasy);
xlabel('Time (seconds)');
ylabel('Amplitude');
```



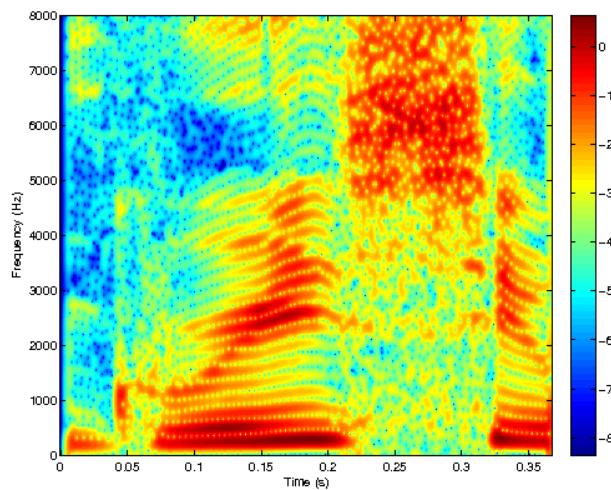
Plot of 'greasy' in the frequency-domain:

```
plotfftreal(fftreal(greasy),16000,90);
```



Plot of 'greasy' in the time-frequency-domain:

```
sgram(greasy,16000,90);
```



**References:** [56]

#### 10.2.4 COCKTAILPARTY - Load the 'cocktailparty' test signal

##### Usage

```
s=cocktailparty;
```

##### Description

`cocktailparty` loads the 'cocktailparty' signal. It is a recording of a male native English speaker pronouncing the sentence "The cocktail party effect refers to the ability to focus on a single talker among a mixture of conversations in background noises".

`[sig, fs]=cocktailparty` additionally returns the sampling frequency  $fs$ .

The signal is 363200 samples long and recorded at 44.1 kHz in an anechoic environment.

#### 10.2.5 GSPI - Load the 'glockenspiel' test signal

`gspi` loads the 'glockenspiel' signal. This is a recording of a simple tune played on a glockenspiel. It is 262144 samples long, mono, recorded at 44100 Hz using 16 bit quantization.

`[sig, fs]=gspi` additionally returns the sampling frequency  $fs$ .

This signal, and other similar audio tests signals, can be found on the EBU SQAM test signal CD <http://tech.ebu.ch/publications/sqamcd>.

#### 10.2.6 LINUS - Load the 'linus' test signal

##### Usage

```
s=linus;
```

##### Description

`linus` loads the 'linus' signal. It is a recording of Linus Thorvalds pronouncing the words "Hello. My name is Linus Thorvalds, and I pronounce Linux as Linux".

The signal is 41461 samples long and is sampled at 8 kHz.

`[sig, fs]=linus` additionally returns the sampling frequency  $fs$ .

See <http://www.paul.sladen.org/pronunciation/>.

### 10.2.7 LTFATLOGO - Load the 'ltfatlogo' test signal

#### Usage

```
s=ltfatlogo;
```

#### Description

`ltfatlogo` loads the 'ltfatlogo' signal. This is a sound synthesised from an artificial spectrogram of the word 'LTFAT'. See the help of `lftattext`.

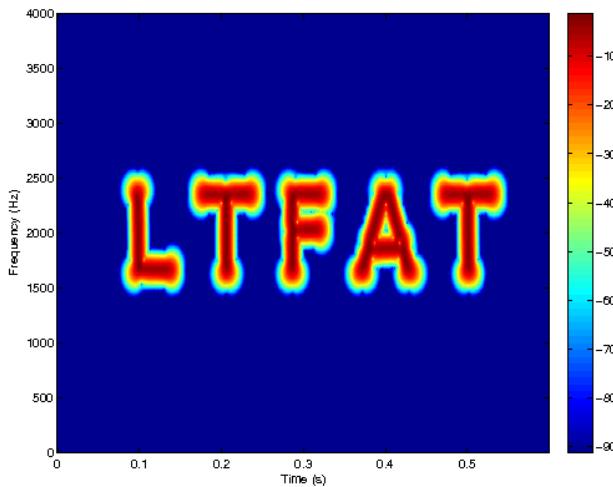
`[sig, fs]=ltfatlogo` additionally returns the sampling frequency  $fs$ .

The signal is 7200 samples long and recorded at 8 kHz. It has been scaled to not produce any clipping.

#### Examples:

To produce a spectrogram of the logo, use:

```
sgram(ltfatlogo, 8000, 90);
```



### 10.2.8 OTOCLICK - Load the 'otoclick' test signal

#### Usage

```
s=otoclick;
```

#### Description

`otoclick` loads the 'otoclick' signal. The signal is a click-evoked otoacoustic emission. It consists of two clear clicks followed by a ringing. The ringing is the actual otoacoustic emission.

`[sig, fs]=otoclick` additionally returns the sampling frequency  $fs$ .

It was measured by Sarah Verhulst at CAHR (Centre of Applied Hearing Research) at Department of Electrical Engineering, Technical University of Denmark

The signal is 2210 samples long and sampled at 44.1 kHz.

### 10.2.9 TRAINDOPPLER - Load the 'traindoppler' test signal

#### Usage

```
s=traindoppler;
```

**Description**

`traindoppler` loads the 'traindoppler' signal. It is a recording of a train passing close by with a clearly audible doppler shift of the train whistle sound.

`[sig, fs]=traindoppler` additionally returns the sampling frequency  $fs$ .

The signal is 157058 samples long and sampled at 8 kHz.

The signal was obtained from <http://www.fourmilab.ch/cship/doppler.html>

## 10.3 Images.

### 10.3.1 CAMERAMAN - Load the 'cameraman' test image

**Usage**

```
s=cameraman;
```

**Description**

`cameraman` loads a  $256 \times 256$  greyscale image of a cameraman.

The returned matrix `s` consists of integers between 0 and 255, which have been converted to double precision.

To display the image, use `imagesc` with a gray colormap:

```
imagesc(cameraman); colormap(gray); axis('image');
```



See <ftp://nic.funet.fi/pub/graphics/misc/test-images/> or <http://sipi.usc.edu/database/database.cgi?volume=misc>.

### 10.3.2 LICHTENSTEIN - Load the 'lichtenstein' test image

**Usage**

```
s=lichtenstein;
```

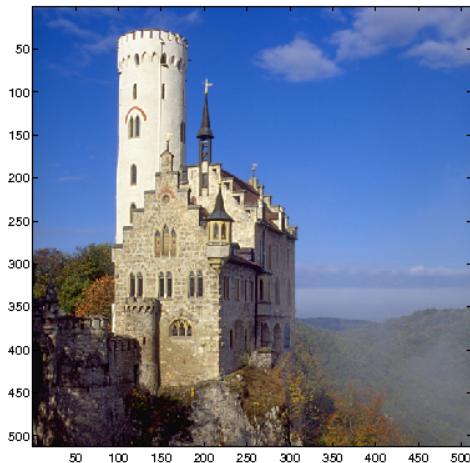
**Description**

`lichtenstein` loads a  $512 \times 512$  color image of a castle Lichtenstein, [http://en.wikipedia.org/wiki/Lichtenstein\\_Castle](http://en.wikipedia.org/wiki/Lichtenstein_Castle).

The returned matrix `s` consists of integers between 0 and 255.

To display the image, simply use `image`:

```
image(lichtenstein); axis('image');
```



See [http://commons.wikimedia.org/wiki/File:Lichtenstein\\_img\\_processing\\_test.png](http://commons.wikimedia.org/wiki/File:Lichtenstein_img_processing_test.png).

### 10.3.3 LTFATTEXT - Load the 'ltfattext' test image

#### Usage

```
s=ltfattext;
```

#### Description

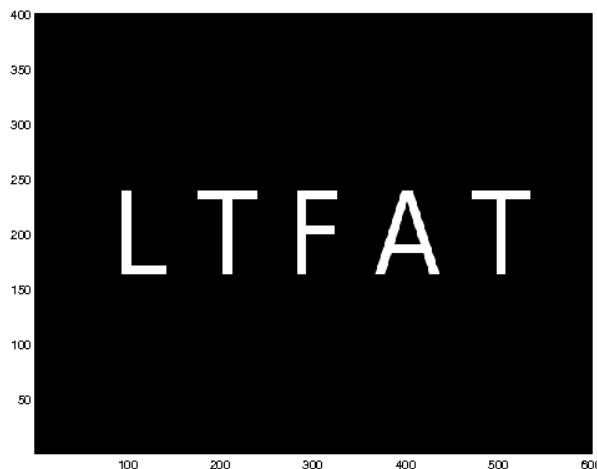
`ltfattext` loads a  $401 \times 600$  black and white image of the word 'LTFAT'.

The image is assumed to be used as a spectrogram with 800 channels as produced by `dgtreal`.

The returned matrix `s` consists of the integers 0 and 1, which have been converted to double precision.

To display the image, use `imagesc` with a gray colormap:

```
imagesc(ltfattext);
colormap(gray);
axis('xy');
```



# Chapter 11

## LTFAT - Demos

### 11.1 Basic demos

#### 11.1.1 DEMO\_DGT - Basic introduction to DGT analysis/synthesis

This demo shows how to compute Gabor coefficients of a signal.

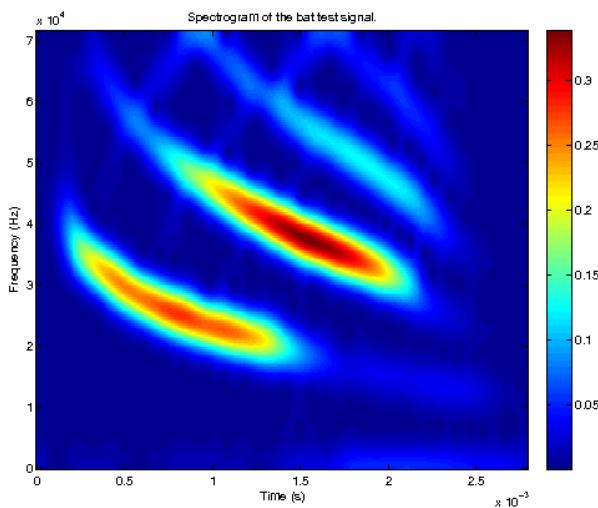


Figure 11.1: Spectrogram of the 'bat' signal.

The figure shows a spectrogram of the 'bat' signal. The coefficients are shown on a linear scale.

#### Output

Type "help demo\_dgt" to see a description of how this demo works.

----- Spectrogram analysis -----

The spectrogram is highly redundant.

No. of coefficients in the signal: 400

No. of coefficients in the spectrogram: 80400

Redundancy of the spectrogram: 201.000000

----- Simple Gabor analysis using a standard Gaussian window. -----

Setup parameters for a Discrete Gabor Transform.

Time shift:

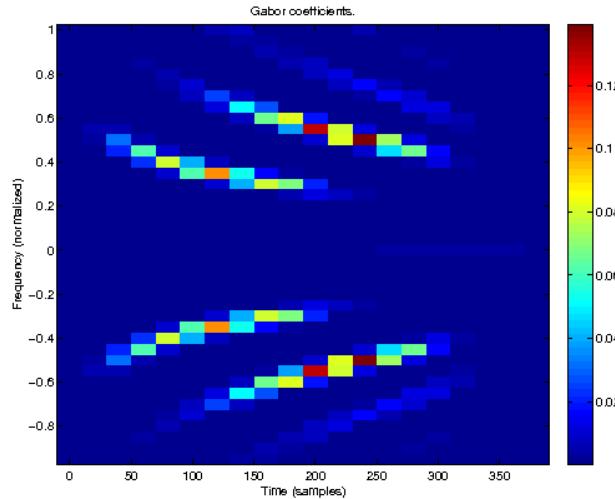


Figure 11.2: Gabor coefficients of the 'bat' signal.

The figure show a set of Gabor coefficients for the 'bat' signal, computed using a DGT with a Gaussian window. The coefficients contains all the information to reconstruct the signal, even though there are far fewer coefficients than the spectrogram contains.

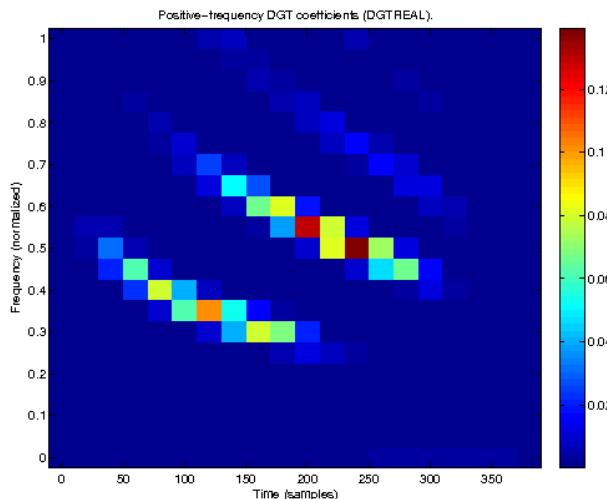


Figure 11.3: Real-valued Gabor analysis

This figure shows only the coefficients for the positive frequencies. As the signal is real-value, these coefficients contain all the necessary information. Compare to the shape of the spectrogram shown on Figure 1.

$a =$

20

Number of frequency channels.

$M =$

40

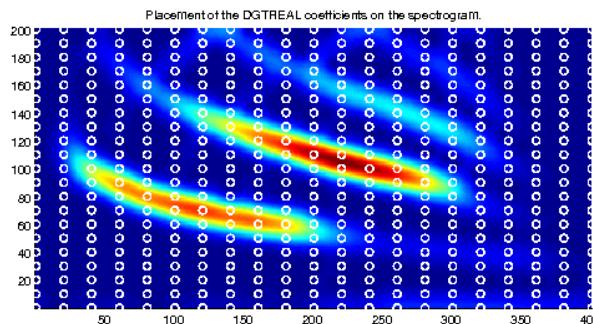


Figure 11.4: DGT coefficients on a spectrogram

This figure shows how the coefficients from DGTREAL can be picked from the coefficients computed by a full Short-time Fourier transform, as visualized by a spectrogram.

Note that it must hold that  $L = M \cdot b = N \cdot a$  for some integers  $b$ ,  $N$  and  $L$ , and that  $a < M$ .  $L$  is the transform length, and the DGT will choose the smallest possible value of  $L$  that is larger or equal to the length of the signal. Choosing  $a < M$  makes the transform redundant, otherwise the transform will be lossy, and reconstruction will not be possible.

Number of time shifts in transform:

Length of transform:

```
L =
```

```
400
```

The redundancy of the Gabor transform can be reduced without loosing information.

No. of coefficients in the signal: 400

No. of output coefficients from the DGT: 800

Redundacy of the DGT (in this case) 2.000000

----- Real valued Gabor analysis. -----

----- Perfect reconstruction. -----

Reconstruction error using IDGT: 5.384302e-16

Reconstruction error using IDGTREAL: 5.664802e-16

### 11.1.2 DEMO\_GABFIR - Working with FIR windows

This demo demonstrates how to work with FIR windows in Gabor systems.

FIR windows are the windows traditionally used in signal processing. They are short, much shorter than the signal, and this is used to make effecient algorithms. They are also the only choice for applications involving streaming data.

It is very easy to compute a spectrogram or Gabor coefficients using a FIR window. The hard part is

reconstruction, because both the window and the dual window used for reconstruction must be FIR, and this is hard to obtain, if the window is longer than the number of channels.

This demo demonstrates two methods:

- 1) Using a Gabor frame with a simple structure, for which dual/tight FIR windows are easy to construct. This is a very common technique in traditional signal processing, but it limits the choice of windows and lattice parameters.
- 2) Cutting a canonical dual/tight window. We compute the canonical dual window of the analysis window, and cut away the parts that are close to zero. This will work for any analysis window and any lattice constant, but the reconstruction obtained is not perfect.

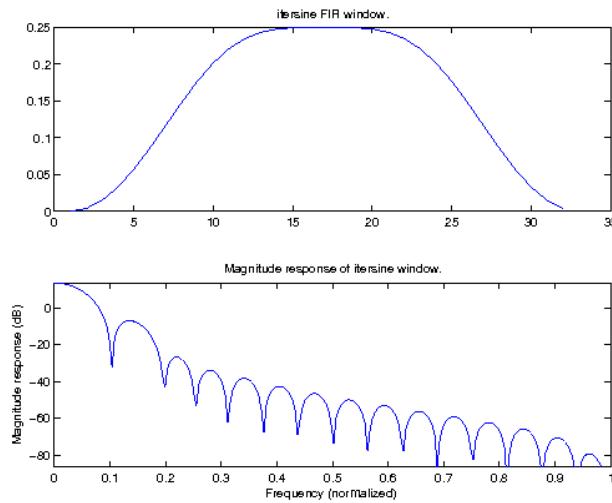


Figure 11.5: Hanning FIR window

This figure shows the a Hanning window in the time domain and its magnitude response.

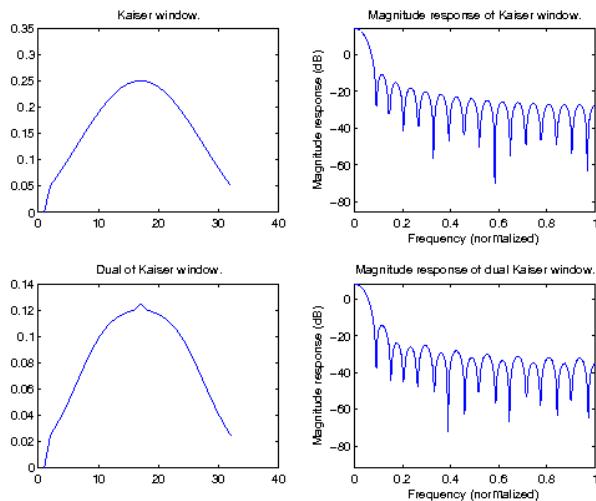


Figure 11.6: Kaiser-Bessel FIR window

This figure shows a Kaiser Bessel window and its magnitude response, and the same two plots for the canonical dual of the window.

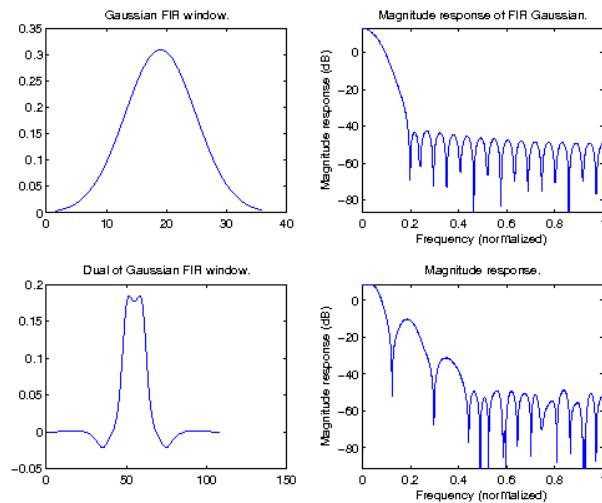


Figure 11.7: Gaussian FIR window for low redundancy

This figure shows a truncated Gaussian window and its magnitude response. The same two plots are shown for the truncated canonical dual window.

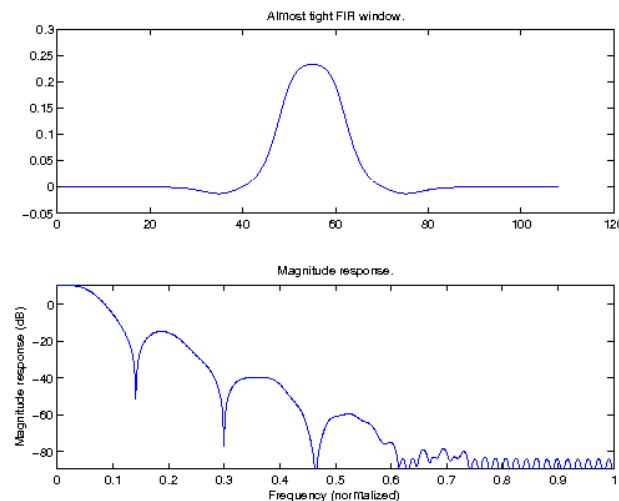


Figure 11.8: Almost tight Gaussian FIR window

This figure shows a tight Gaussian window that has been truncated and its magnitude response.

## Output

Reconstruction error using itersine window, should be close to zero:

```
ans =
```

```
0.5000
```

Reconstruction error canonical dual of Kaiser window, should be close to zero:

```
ans =
```

```
6.4020e-17
```

Reconstruction error using cutted dual window.

```
recerr =
0.0018
or expressed in dB:
ans =
-27.5387
```

Reconstruction error using cutted tight window.

```
recerr =
9.8583e-04
or expressed in dB:
ans =
-30.0620
```

### 11.1.3 DEMO\_WAVELETS - Wavelet filterbanks

This demo exemplifies the use of the wavelet filterbank trees. All representations use "least asymmetric" Daubechies wavelet orthonormal filters 'sym8' (8-regular, length 16).

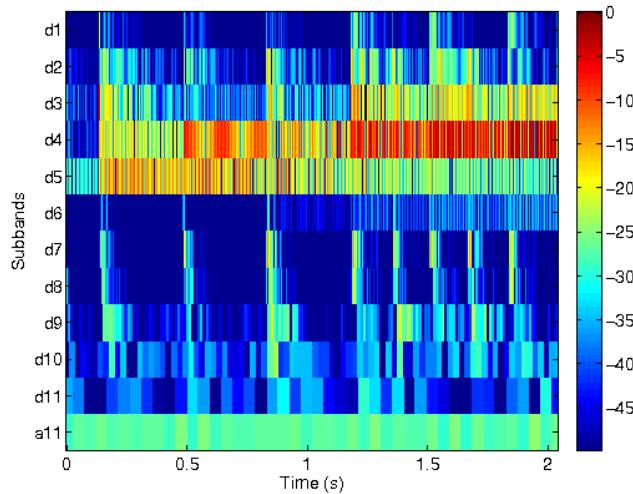


Figure 11.9: DWT representation

The filterbank tree consists of 11 levels of iterated 2-band basic wavelet filterbank, where only the low-pass output is further decomposed. This results in 12 bands with octave resolution.

### Output

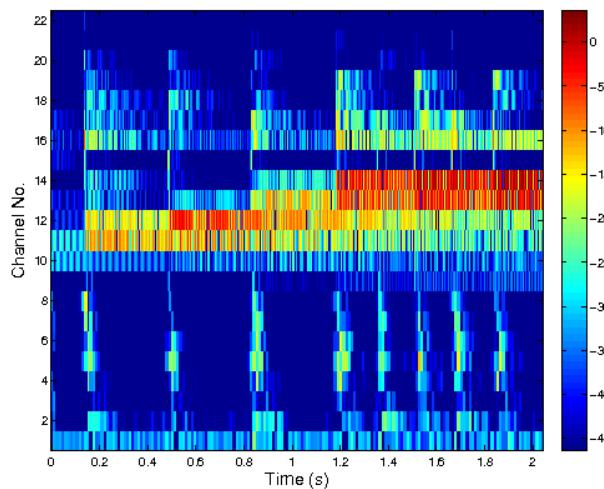


Figure 11.10: 8-band DWT representation

The filterbank tree (effectively) consists of 3 levels of iterated 8-band basic wavelet filterbank resulting in 22 bands. Only the low-pass output is decomposed at each level.

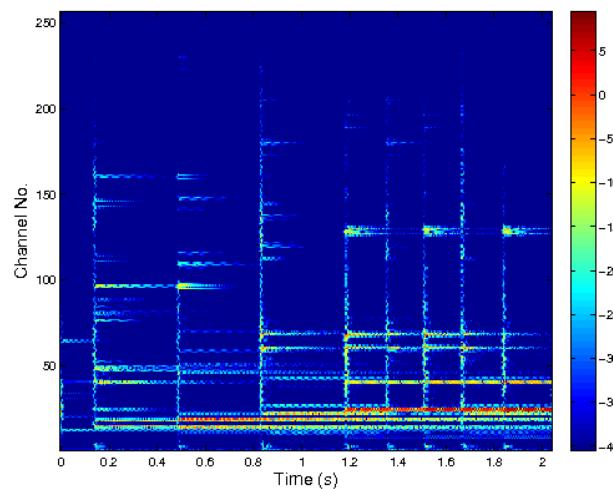


Figure 11.11: Full Wavelet filterbank tree representation

The filterbank tree depth is 8 and it is fully decomposed meaning both outputs (low-pass and high-pass) of the basic filterbank is plotted further. This results in 256 bands linearly covering the frequency axis.

## 11.2 Compression

### 11.2.1 DEMO\_IMAGECOMPRESSION - Image compression using N-term approximation

This demo shows how to perform a simple imagecompression using either a Wilson basis or a Wavelet. The compression step is done by retaining only 5% of the coefficients.

#### Output

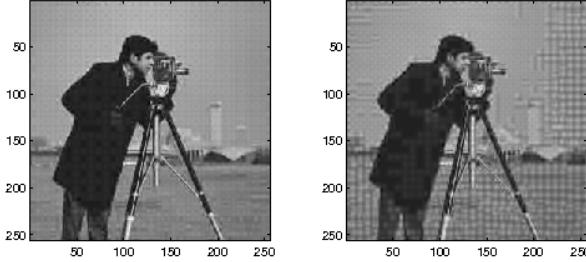


Figure 11.12: Wilson and WMDCT basis

This right figure shows the image compressed using a dwilt basis with 8 channels. This corresponds quite closely to JPEG compression, except that the borders between neighbouring blocs are smoother, since the dwilt uses a windowing function.

The left figure shows the same, now using a MDCT basis. The MDCT produces more visible artifacts, as the slowest changing frequency in each block has a half-wave modulation. This is visible on otherwise smooth backgrounds.

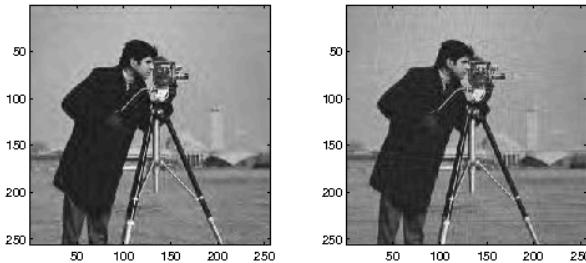


Figure 11.13: Wavelet

The Wavelet used is the DB6 with  $J = 5$  levels. On the right figure the standard layout has been used, on the left the tensor layout was used.

### 11.2.2 DEMO\_AUDIOCOMPRESSION - Audio compression using N-term approx

This demos shows how to do audio compression using best N-term approximation of an wmdct transform.

The signal is transformed using an orthonormal wmdct transform. Then approximations with a fixed number  $N$  of coefficients are obtained by:

- Linear approximation: The  $N$  coefficients with lowest frequency index are kept.
- Non-linear approximation: The  $N$  largest coefficients (in magnitude) are kept.

The corresponding approximated signal can be computed using iwm dct.

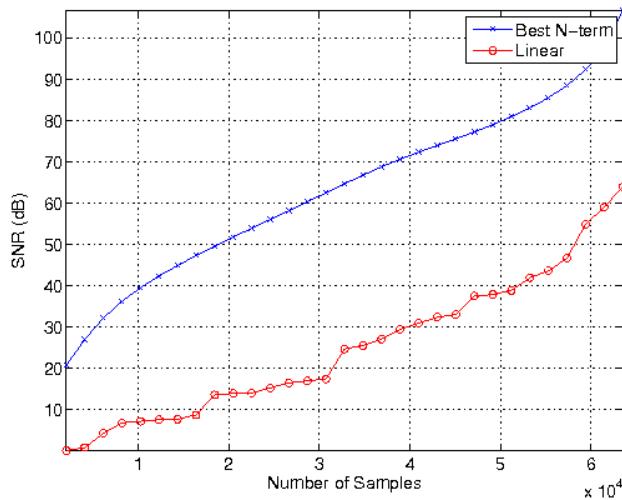


Figure 11.14: Rate-distortion plot

The figure shows the output Signal to Noise Ratio (SNR) as a function of the number of retained coefficients.

Note: The inverse WMDCT is not needed for computing computing SNRs. Instead Parseval theorem states that the norm of a signal equals the norm of the sequence of its wmdct coefficients.

### Output

## 11.3 Denoising

### 11.3.1 DEMO\_AUDIO\_DENOISE - Audio denoising using thresholding

This demos shows how to do audio denoising using thresholding of WMDCT transform.

The signal is transformed using an orthonormal WMDCT transform followed by a thresholding. Then the signal is reconstructed and compared with the original.

### Output

#### RESULTS:

```

Input SNR: 6.044694 dB.
Output SNR (hard): 9.091411 dB.
Output SNR (soft): 14.283695 dB.
Signals are stored in variables sig, nsig, hrec, srec

```

## 11.4 Applications

### 11.4.1 DEMO\_OFDM - Demo of Gabor systems used for OFDM

This demo shows how to use a Gabor Riesz basis for OFDM.

We want to transmit a signal consisting of 0's and 1's through a noisy communication channel. This is accomplished in the following steps in the demo:

- 1) Convert this digital signal into complex valued coefficients by QAM modulation.
- 2) Construct the signal to be transmitted by an inverse Gabor transform of the complex coefficients

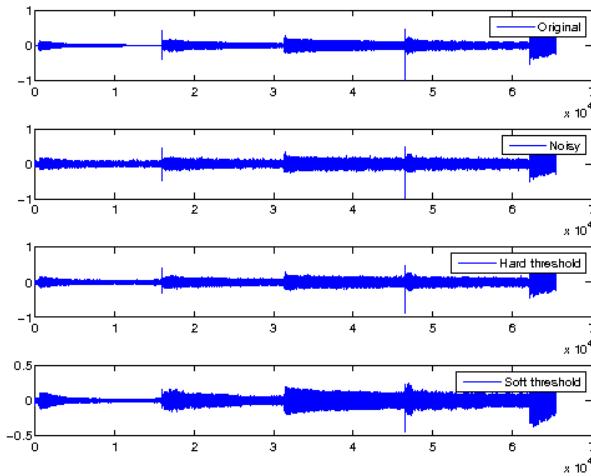


Figure 11.15: Denoising

The figure shows the original signal, the noisy signal and denoised signals using hard and soft thresholding applied to the WMDCT of the noise signal.

- 3) "Transmit" the signal by applying a spreading operator to the signal and adding white noise
- 4) Convert the received signal into noisy coefficients by a Gabor transform
- 5) Convert the noisy coefficients into bits by inverse QAM.

Some simplifications used to make this demo simple:

- We assume that the whole spectrum is available for transmission.
- The window and its dual have full length support. This is not practical, because all data would have to be processed at once. Instead, an FIR should be used, with both the window and its dual having a short length.
- The window is periodic. The data at the very end interferes with the data at the very beginning. A simple way to solve this is to transmit zeros at the beginning and at the end, to flush the system properly.

## Output

Type "help demo\_ofdm" to see a description of how this demo works.

Number of faulty bits:

```
faulty =
```

```
0
```

Error rate:

```
ans =
```

```
0
```

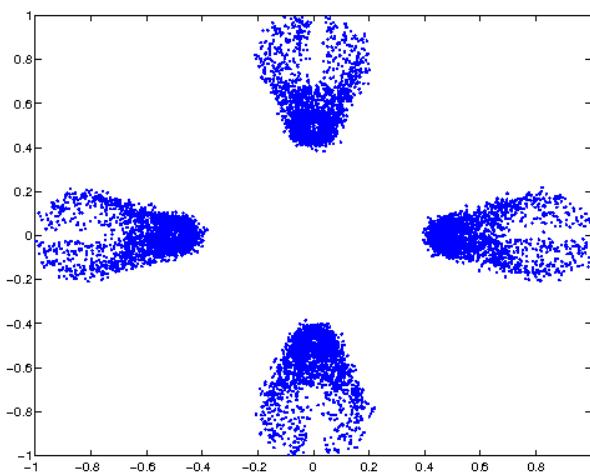


Figure 11.16: Received coefficients.

This figure shows the distribution in the complex plane of the received coefficients. If the channel was perfect, all the points should appear at the complex roots of unity ( $1, i, -1$  and  $-i$ ). This demo is random, so everytime it is run it produces a new plot, and the error rate may vary.

#### 11.4.2 DEMO\_AUDIO SHRINK - Decomposition into tonal and transient parts

This demos shows how to do audio coding and "tonal + transient" decomposition using group lasso shrinkage of two wmdct transforms with different time-frequency resolutions.

The signal is transformed using two orthonormal wmdct bases. Then group lasso shrinkage is applied to the two transforms in order to:

- select fixed frequency lines of large wmdct coefficients on the wide window wmdct transform
- select fixed time lines of large wmdct coefficients on the narrow window wmdct transform

The corresponding approximated signals are computed with the corresponding inverse, iwmddct.

Corresponding reconstructed tonal and transient sounds may be listened from arrays rec1 and rec2 (sampling rate: 44.1 kHz)

##### Output

```
Percentage of retained coefficients: 10.156250 + 12.988281 = 23.144531
To play the original, type "soundsc(sig,fs)"
To play the tonal part, type "soundsc(rec1,fs)"
To play the transient part, type "soundsc(rec2,fs)"
```

#### 11.4.3 DEMO\_GABMULAPPR - Approximate a slowly time variant system by a Gabor multiplier

This script construct a slowly time variant system and performs the best approximation by a Gabor multiplier with specified parameters (a and L see below). Then it shows the action of the slowly time variant system (A) as well as of the best approximation of (A) by a Gabor multiplier (B) on a sinusoids and an exponential sweep.

##### Output

Type "help demo\_gabmulappr" to see a description of how this demo works.

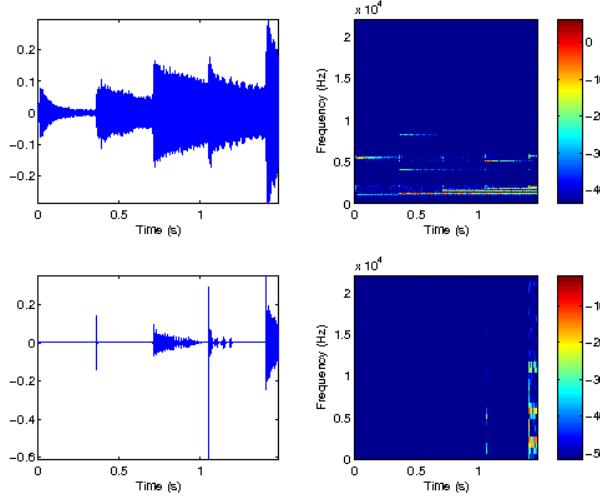


Figure 11.17: Plots and time-frequency images

The upper plots in the figure show the tonal parts of the signal, the lower plots show the transients. The TF-plots on the left are the corresponding wmdct coefficients found by appropriate group lasso shrinkage

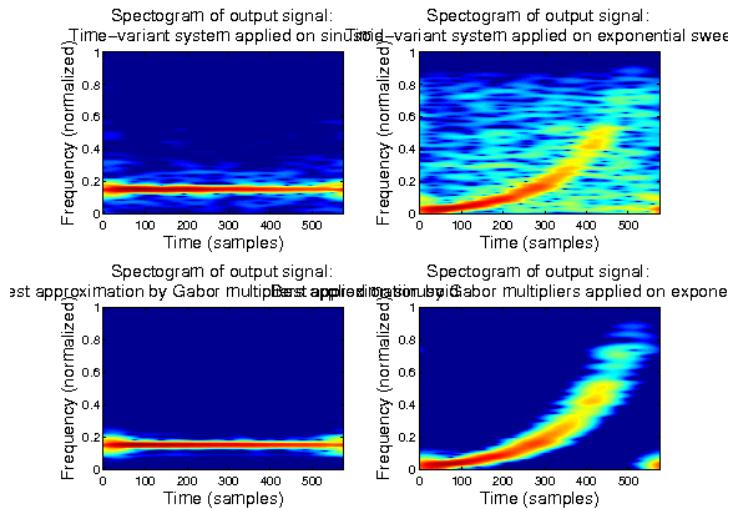


Figure 11.18: Spectrogram of signals

The figure shows the spectrogram of the output of the two systems applied on a sinusoid (left) and an exponential sweep.

#### 11.4.4 DEMO\_BPFRAMEMUL - Frame multiplier acting as a time-varying band-pass filter

This demo demonstrates creation and effect of a Gabor multiplier. The multiplier performs a time-varying bandpass filtering. The band-pass filter with center frequency changing over time is explicitly created but it is treated as a "black box" system. The symbol is identified by "probing" the system with a white noise and dividing DGT of the output by DGT of the input element-wise. The symbol is smoothed out by a 5x5 median filter.

##### Output

The original signal can be played by typing: `sound(forig, 44100);`

The signal obtained by a direct filtering can be played by (this is what is approximat

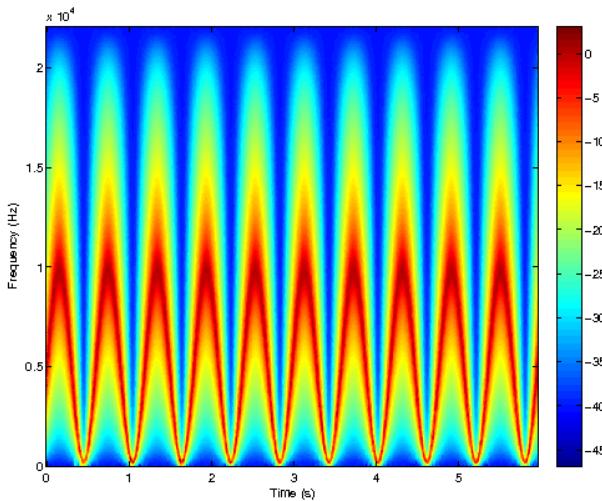


Figure 11.19: The symbol of the multiplier.  
This figure shows a symbol used in the Gabor multiplier.

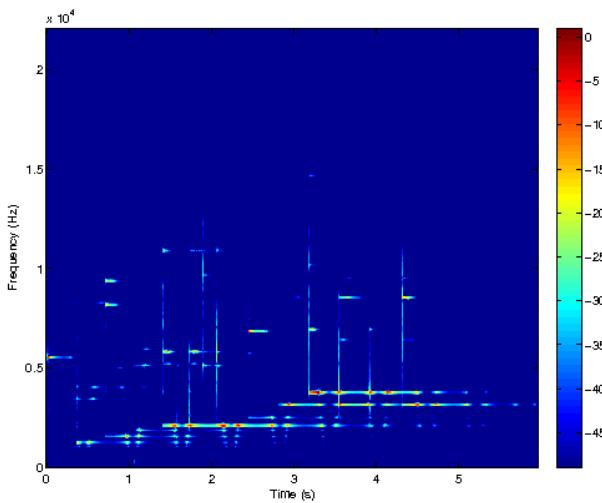


Figure 11.20: Spectroram obtained by re-analysis of the test signal after applying the multiplier  
This figure shows a spectrogram of the test signal after applying the estimated Gabor multiplier.

The signal obtained by applying the Gabor multiplier: `sound(fmul, 44100);`

### 11.4.5 DEMO\_FRSYNABS - Construction of a signal with a given spectrogram

This demo demonstrates iterative reconstruction of a spectrogram.

#### Output

Iteration	FunEvals	Step Length	Function Val	Opt Cond
1	4	3.15913e-03	3.90230e+03	2.87505e+01
2	8	2.77377e-01	3.41137e+03	5.90590e+01
3	9	1.00000e+00	3.17168e+03	3.70145e+01
4	12	4.33149e-01	2.82986e+03	8.98805e+01
5	13	1.00000e+00	2.52558e+03	6.31840e+01

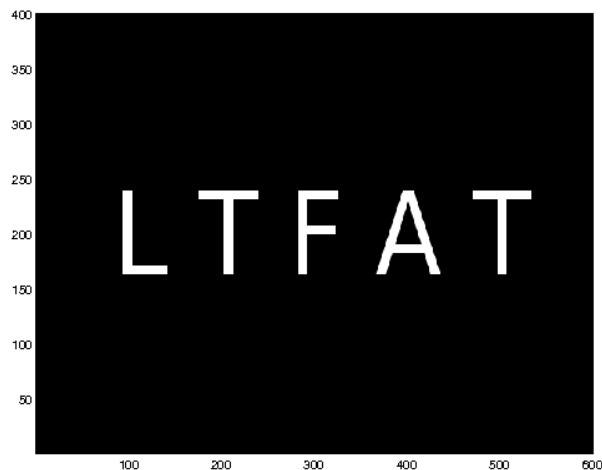


Figure 11.21: Original spectrogram

This figure shows the target spectrogram

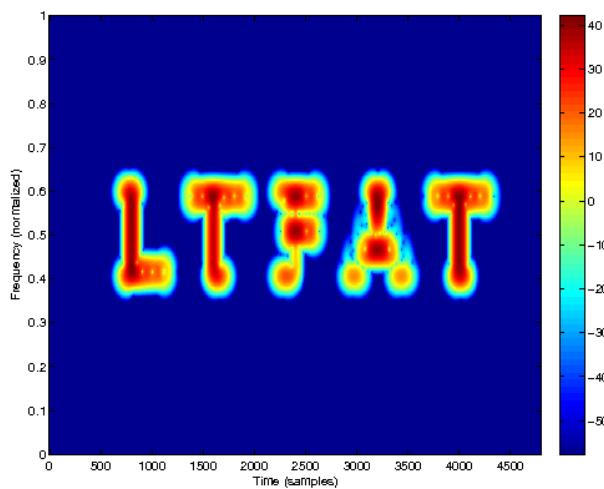


Figure 11.22: Linear reconstruction

This figure shows a spectrogram of a linear reconstruction of the target spectrogram.

6	14	1.00000e+00	2.27079e+03	4.06172e+01
7	16	6.20623e-01	2.03221e+03	3.89615e+01
8	17	1.00000e+00	1.92900e+03	4.53433e+01
9	18	1.00000e+00	1.80834e+03	2.17270e+01
10	19	1.00000e+00	1.72020e+03	2.75247e+01
11	20	1.00000e+00	1.66585e+03	1.32635e+01
12	21	1.00000e+00	1.62779e+03	1.30775e+01
13	22	1.00000e+00	1.56124e+03	3.40932e+01
14	23	1.00000e+00	1.48769e+03	3.82270e+01
15	24	1.00000e+00	1.39065e+03	2.84914e+01
16	25	1.00000e+00	1.33769e+03	2.37581e+01
17	27	4.67626e-01	1.32161e+03	1.31561e+01
18	28	1.00000e+00	1.28823e+03	1.50018e+01

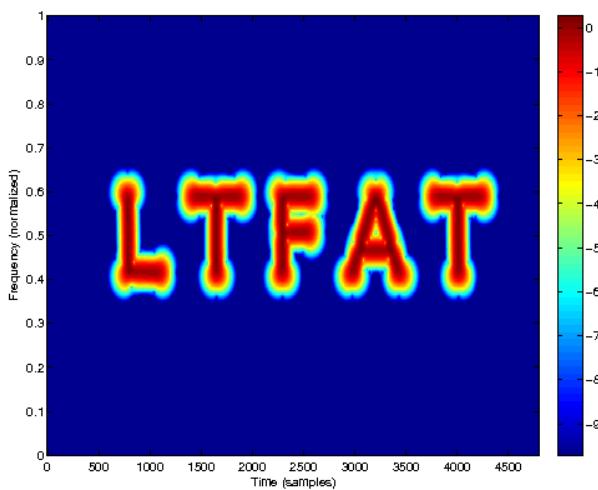


Figure 11.23: Iterative reconstruction using the Griffin-Lim method.

This figure shows a spectrogram of an iterative reconstruction of the target spectrogram using the Griffin-Lim projection method.

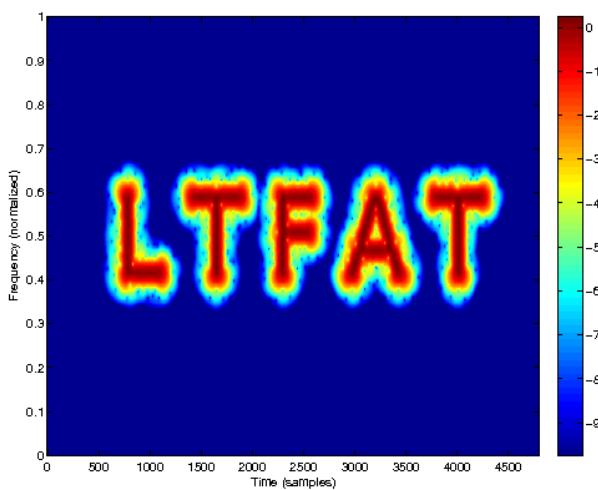


Figure 11.24: Iterative reconstruction using the BFGS method

This figure shows a spectrogram of an iterative reconstruction of the target spectrogram using the BFGS method.

19	29	1.00000e+00	1.24806e+03	1.62263e+01
20	30	1.00000e+00	1.22975e+03	1.91357e+01
21	31	1.00000e+00	1.20536e+03	1.50017e+01
22	32	1.00000e+00	1.20338e+03	6.02111e+00
23	33	1.00000e+00	1.19950e+03	7.83330e+00
24	34	1.00000e+00	1.18896e+03	1.06102e+01
25	35	1.00000e+00	1.17668e+03	1.89324e+01
26	36	1.00000e+00	1.14813e+03	1.91084e+01
27	37	1.00000e+00	1.12769e+03	4.72956e+00
28	38	1.00000e+00	1.12311e+03	7.44280e+00
29	39	1.00000e+00	1.10920e+03	1.66219e+01
30	40	1.00000e+00	1.10237e+03	6.37538e+00
31	41	1.00000e+00	1.09882e+03	5.16906e+00

32	42	1.00000e+00	1.09498e+03	6.75999e+00
33	43	1.00000e+00	1.08635e+03	7.16026e+00
34	45	3.81108e-01	1.08478e+03	8.15267e+00
35	46	1.00000e+00	1.07380e+03	5.22764e+00
36	49	1.46171e-01	1.07339e+03	6.04082e+00
37	50	1.00000e+00	1.07060e+03	8.15677e+00
38	51	1.00000e+00	1.06573e+03	4.74986e+00
39	52	1.00000e+00	1.06108e+03	8.02286e+00
40	53	1.00000e+00	1.05414e+03	8.62442e+00
41	69	6.92120e-02	1.05411e+03	7.99077e+00
42	72	1.54984e-01	1.05390e+03	5.64380e+00
43	75	1.04496e-01	1.05358e+03	5.52928e+00
44	77	3.66244e-01	1.05221e+03	4.40492e+00
45	78	1.00000e+00	1.05019e+03	8.77455e+00
46	79	1.00000e+00	1.04476e+03	5.78831e+00
47	80	1.00000e+00	1.04360e+03	9.11101e+00
48	81	1.00000e+00	1.03491e+03	4.47858e+00
49	82	1.00000e+00	1.03264e+03	2.31842e+00
50	84	2.86103e-01	1.03223e+03	2.85258e+00
51	86	3.52863e-01	1.03069e+03	3.67991e+00
52	87	1.00000e+00	1.03042e+03	7.61050e+00
53	88	1.00000e+00	1.02462e+03	3.79045e+00
54	89	1.00000e+00	1.02384e+03	2.88559e+00
55	90	1.00000e+00	1.02277e+03	3.26059e+00
56	91	1.00000e+00	1.02198e+03	3.66535e+00
57	92	1.00000e+00	1.01888e+03	6.98288e+00
58	104	0.00000e+00	1.01888e+03	6.98288e+00

Step Size below progTol

## 11.5 Aspects of particular functions

### 11.5.1 DEMO\_NSDGT - Nonsationary Gabor transform demo

This script sets up a nonstationary Gabor frame with the specified parameters, computes windows and corresponding canonical dual windows and a test signal, and plots the windows and the energy of the coefficients.

#### Output

Type "help demo\_nsdgt" to see a description of how this example works.  
 Relative error of reconstruction (should be close to zero.): 3.227678e-16

### 11.5.2 DEMO\_PGAUSS - How to use PGAUSS

This script illustrates various properties of the Gaussian function.

#### Output

Type "help demo\_pgauss" to see a description of how this demo works.  
 Test of DFT invariance: Should be close to zero.

ans =

2.4815e-15

The function is WP even. The following should be 1.

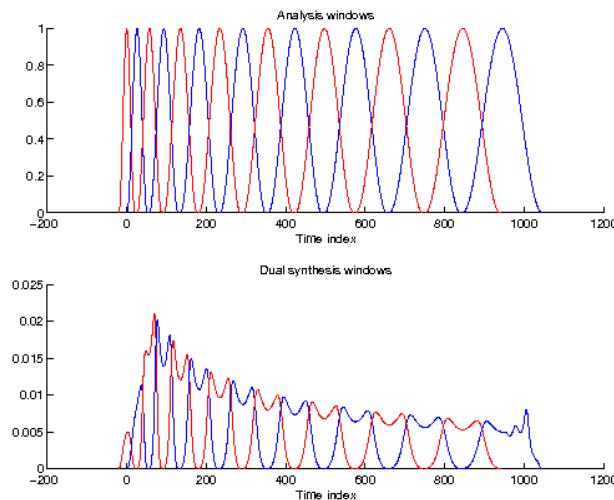


Figure 11.25: Windows + dual windows

This figure shows the window functions used and the corresponding canonical dual windows.

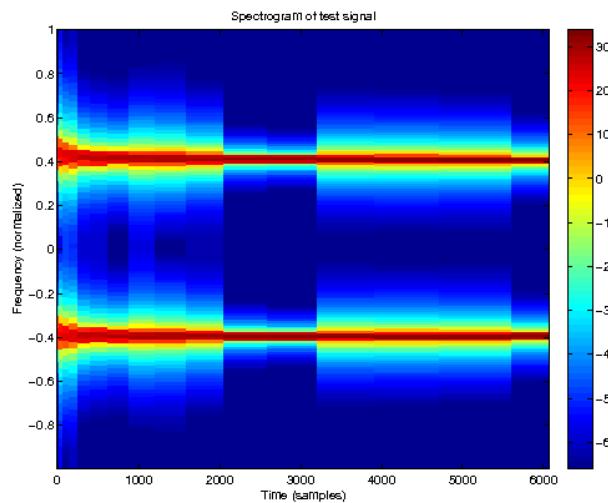


Figure 11.26: Spectrogram (absolute value of coefficients in dB)

This figure shows a (color coded) image of the nsdgt coefficient modulus.

```
ans =
```

```
1
```

Therefore, its DFT is real.

The norm of the imaginary part should be close to zero.

```
ans =
```

```
1.5013e-15
```

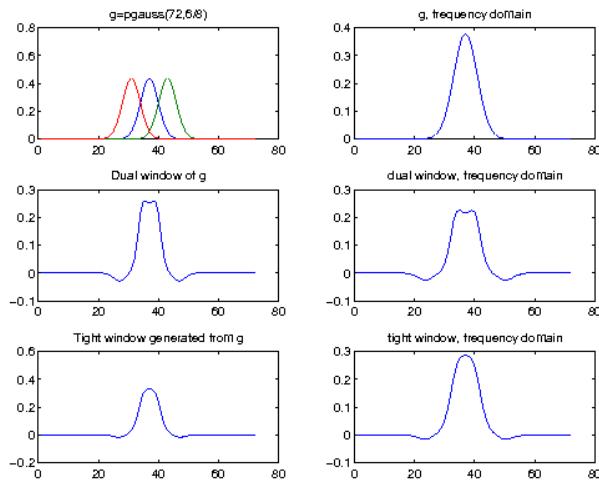


Figure 11.27: Window+Dual+Tight

This figure shows an optimally centered Gaussian for a given Gabor system, its canonical dual and tight windows and the DFTs of these windows.

### 11.5.3 DEMO\_PBSPLINE - How to use PBSPLINE

This script illustrates various properties of the pbspline function.

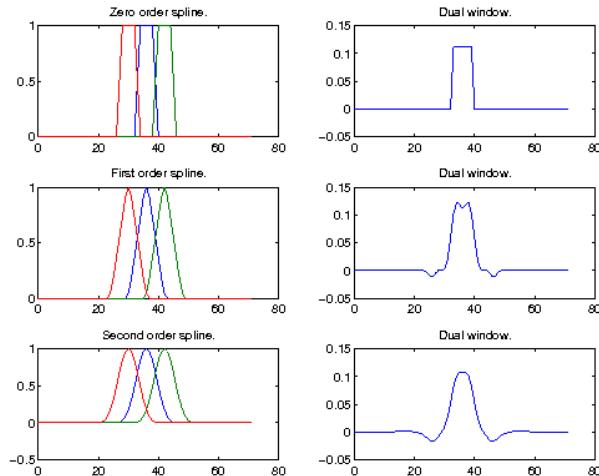


Figure 11.28: Three first splines

This figure shows the three first splines (order 0, 1 and 2) and their dual windows.

Note that they are calculated for an even number of the parameter  $a$ , meaning that they are not exactly splines, but a slightly smoother construction, that still form a partition of unity.

### Output

Type "help demo\_pbspline" to see a description of how this demo works.  
Length of the generated window:

```
ntaps =
```

```
19
```

Norm of imaginary part. Should be close to zero.

```
ans =
```

```
3.4251e-16
```

Window is whole point even. Should be 1.

```
ans =
```

```
1
```

Length of g after cutting.

```
ans =
```

```
19
```

difference between original g, and gextend.  
Should be close to zero.

```
ans =
```

```
4.6075e-16
```

#### 11.5.4 DEMO\_GABMIXDUAL - How to use GABMIXDUAL

This script illustrates how one can produce dual windows using GABMIXDUAL

The demo constructs a dual window that is more concentrated in the time domain by mixing the original Gabor window by one that is extremely well concentrated. The result is somewhat in the middle of these two.

The lower framebound of the mixing Gabor system is horrible, but this does not carry over to the gabmixdual.

#### Output

Type "help demo\_gabmixdual" to see a description of how this demo works.  
Framebounds of initial Gabor system:

```
A1 =
```

```
0.6058
```

```
B1 =
```

```
1.6902
```

Framebounds of mixing Gabor system:

```
A2 =
```

```
2.0241e-05
```

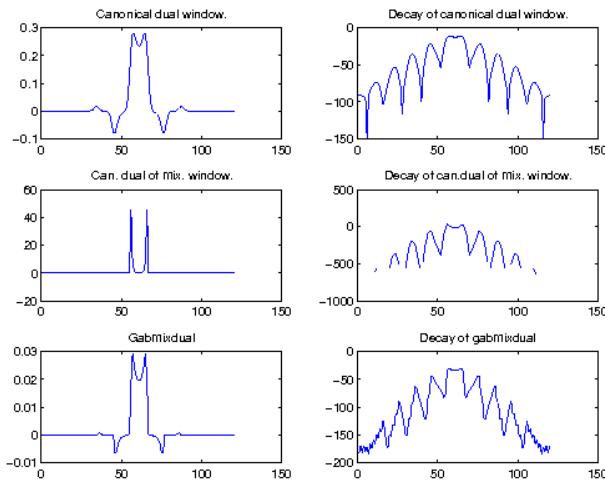


Figure 11.29: Gabmixdual of two Gaussians.

The first row of the figure shows the canonical dual window of the input window, which is a Gaussian function perfectly localized in the time and frequency domains.

The second row shows the canonical dual window of the window we will be mixing with: This is a Gaussian that is 10 times more concentrated in the time domain than in the frequency domain. The resulting canonical dual window has rapid decay in the time domain.

The last row shows the gabmixdual of these two. This is a non-canonical dual window of the first Gaussian, with decay resembling that of the second.

B2 =

4.8990

Framebounds of gabmixdual Gabor system:

Am =

0.0041

Bm =

0.0115

### 11.5.5 DEMO\_FRAMEMUL - Time-frequency localization by a Gabor multiplier

This script creates several different time-frequency symbols and demonstrate their effect on a random, real input signal.

#### Output

Type "help demo\_framemul" to see a description of how this demo works.

### 11.5.6 DEMO\_PHASEPLOT - Give demos of nice phaseplots

This script creates a synthetic signal and then uses phaseplot on it, using several of the possible options.

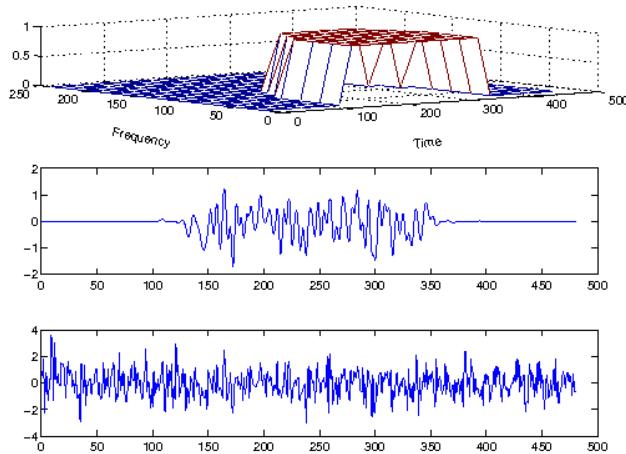


Figure 11.30: Cut a circle in the TF-plane

This figure shows the symbol (top plot, only the positive frequencies are displayed), the input random signal (bottom) and the output signal (middle).

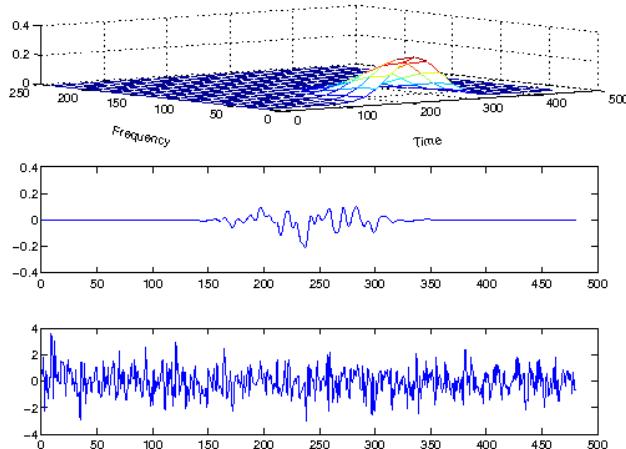


Figure 11.31: Keep low frequencies (low-pass)

This figure shows the symbol (top plot, only the positive frequencies are displayed), the input random signal (bottom) and the output signal (middle).

For real-life signal only small parts should be analyzed. In the chosen demo the fundamental frequency of the speaker can be nicely seen.

**References:** [19], [18], [37]

### Output

Type "help demo\_phaseplot" to see a description of how this demo works.

#### 11.5.7 DEMO\_PHASERET - Phase retrieval and phase difference

This demo demonstrates iterative reconstruction of a spectrogram and the phase difference.

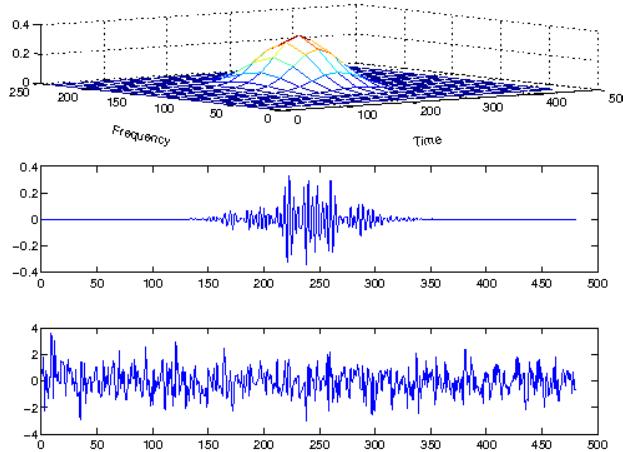


Figure 11.32: Keep middle frequencies (band-pass)

This figure shows the symbol (top plot, only the positive frequencies are displayed), the input random signal (bottom) and the output signal (middle).

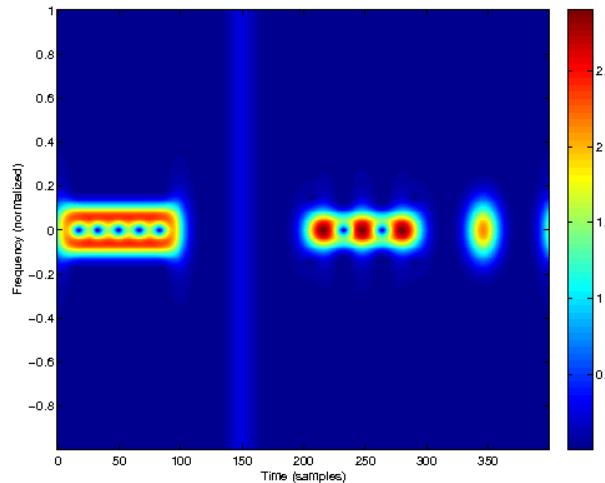


Figure 11.33: Synthetic signal

Compare this to the pictures in reference 2 and 3. In the first two figures a synthetic signal is analyzed. It consists of a sinusoid, a small Delta peak, a periodic triangular function and a Gaussian. In the time-invariant version in the first part the periodicity of the sinusoid can be nicely seen also in the phase coefficients. Also the points of discontinuities can be seen as asymptotic lines approached by parabolic shapes. In the third part both properties, periodicity and discontinuities can be nicely seen. A comparison to the spectrogram shows that the rectangular part in the middle of the signal can be seen by the phase plot, but not by the spectrogram.

In the frequency-invariant version, the fundamental frequency of the sinusoid can still be guessed as the position of an horizontal asymptotic line.

## Output

### 11.5.8 DEMO\_NEXTFASTFFT - Next fast FFT number

This demo shows the behaviour of the nextfastfft function.

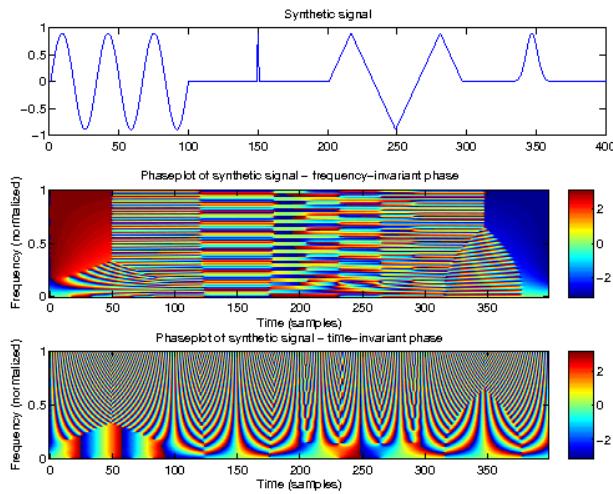


Figure 11.34: Synthetic signal, thresholded.

This figure shows the same as Figure 1, except that values with low magnitude has been removed.

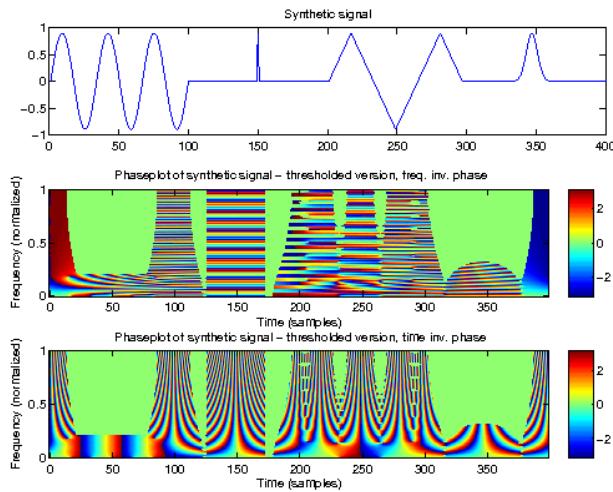


Figure 11.35: Speech signal.

The figure shows a part of the 'linus' signal. The fundamental frequency of the speaker can be nicely seen.

## Output

```
ans =
1.0079
```

### 11.5.9 DEMO\_FILTERBANKS - CQT and ERBLELT filterbanks

This demo shows CQT (Constant Quality Transform) and ERBLELT (Equivalent Rectangular Bandwidth -let transform) representations acting as filterbanks with high and low redundancies. Filterbanks are build such that the painless condition is always satisfied. Real input signal and filters covering only the positive frequency range are used. The redundancy is calculated as a ratio of the number of (complex) coefficients and the input length times two to account for the storage requirements of complex numbers.

- The high redundancy representation uses 'uniform' subsampling i.e. all channels are sub-

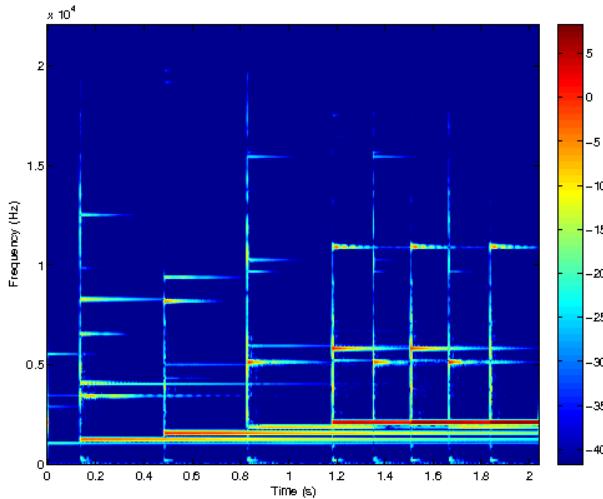


Figure 11.36: Original spectrogram

This figure shows the target spectrogram of an excerpt of the gspi signal

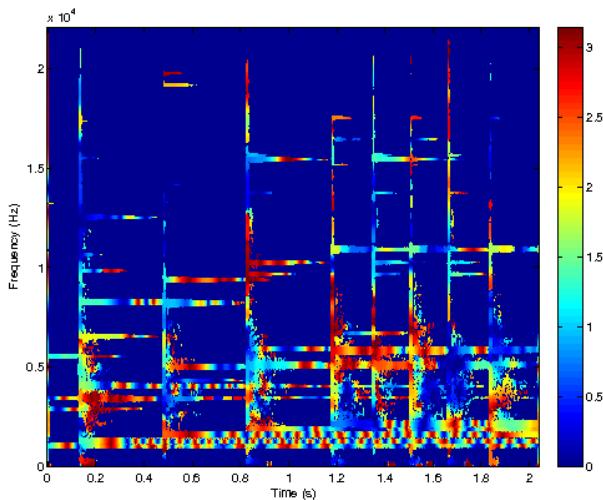


Figure 11.37: Phase difference

This figure shows a difference between the original phase and the reconstructed using 100 iterations of a Fast Griffin-Lim algorithm. Note: The figure in the LTFAT 2.0 paper differs slightly because it is generated using 1000 iterations.

sampled with the same subsampling factor which is the lowest from the filters according to the painless condition rounded towards zero.

- The low redundancy representation uses 'fractional' subsampling which results in the least redundant representation still satisfying the painless condition. Actual time positions of atoms can be non-integer, hence the word fractional.

## Output

ERBLET high redundancy 133.33, low redundancy 2.85.  
CQT high redundancy 50.91, low redundancy 2.02.

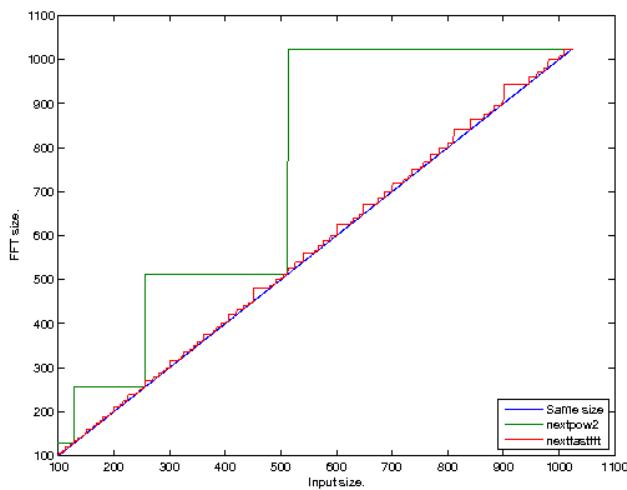


Figure 11.38: Benchmark of the FFT routine

The figure shows the sizes returned by the `nextfastfft` function compared to using `nextpow2`. As can be seen, the `nextfastfft` approach gives FFT sizes that are much closer to the input size.

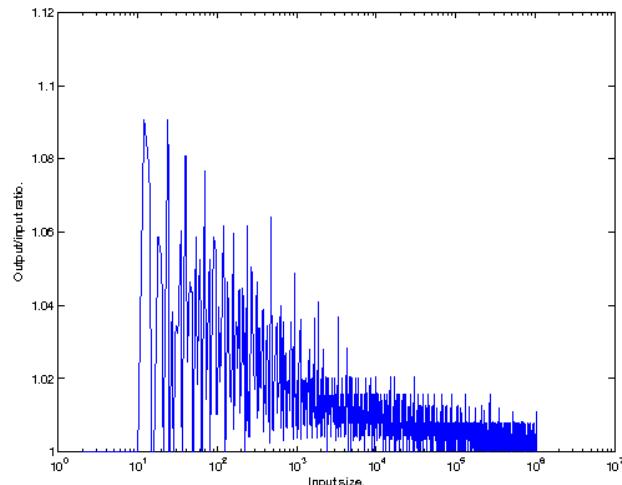


Figure 11.39: Efficiency of the table

The figure show the highest output/input ratio for varying input sizes. As can be seen, the efficiency is better for larger input values, where the output size is at most a few percent larger than the input size.

## 11.6 Auditory scales and filters

### 11.6.1 DEMO\_AUDSCALES - Plot of the different auditory scales

This demos generates a simple figure that shows the behaviour of the different audiology scales in the frequency range from 0 to 8000 Hz.

#### Output

Type `"help demo_audscales"` to see a description of how this demo works.

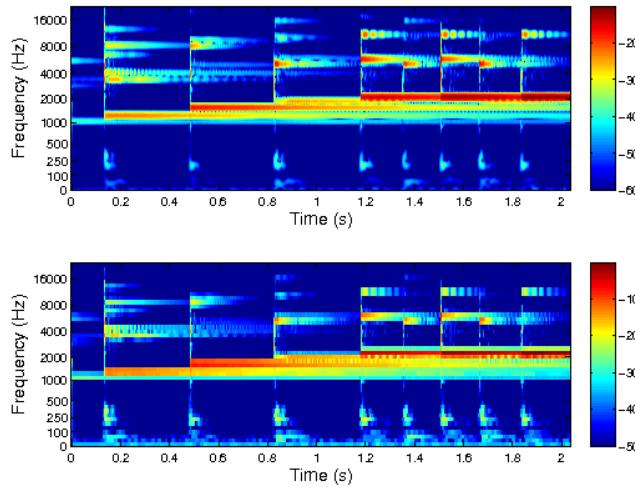


Figure 11.40: ERBLET representations

The high-redundancy plot (top) consists of 400 channels (~9 filters per ERB) and low-redundancy plot (bottom) consists of 44 channels (1 filter per ERB).

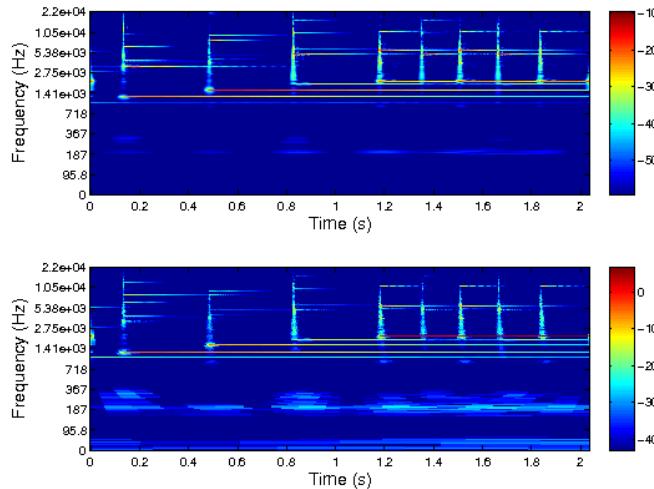


Figure 11.41: CQT representations

Both representations consist of 280 channels (32 channels per octave, frequency range 50Hz-20kHz). The high-redundancy representation is on the top and the low-redundancy repr. is on the bottom.

## 11.6.2 DEMO\_AUDITORYFILTERBANK - Construct an auditory filterbank

In this file we construct a uniform filterbank using a the impulse response of a 4th order gammatone for each channel. The center frequencies are equidistantly spaced on an ERB-scale, and the width of the filter are choosen to match the auditory filter bandwidth as determined by Moore.

Each channel is subsampled by a factor of 8 ( $a=8$ ), and to generate a nice plot, 4 channels per Erb has been used.

The filterbank covers only the positive frequencies, so we must use `filterbankreal dual` and `filterbankreabounds`.

**References:** [33]

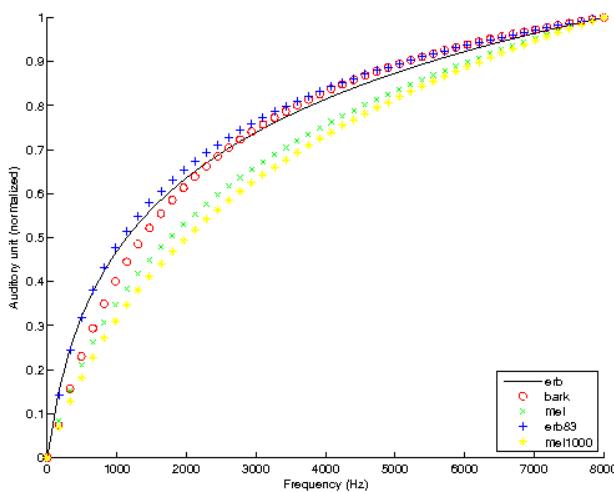


Figure 11.42: Auditory scales

The figure shows the behaviour of the auditory scales on a normalized frequency plot.

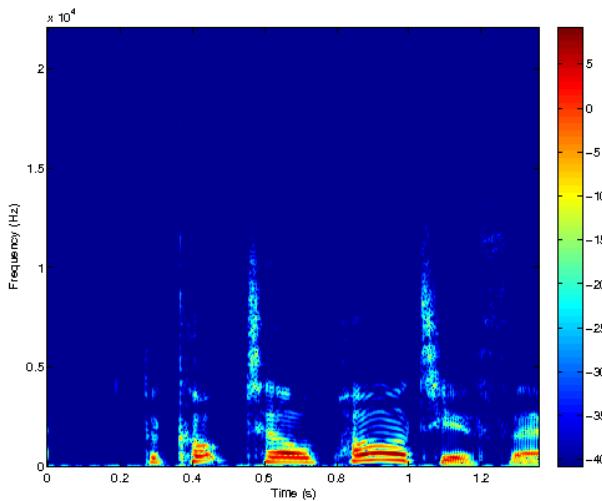


Figure 11.43: Classic spectrogram

A classic spectrogram of the spoken sentence. The dynamic range has been set to 50 dB, to highlight the most important features.

### Output

Frame bound ratio for gammatone filterbank, should be close to 1 if the filters are ch

```
ans =
```

```
1.5989
```

Relative error in reconstruction, should be close to zero.

```
ans =
```

```
6.3517e-05
```

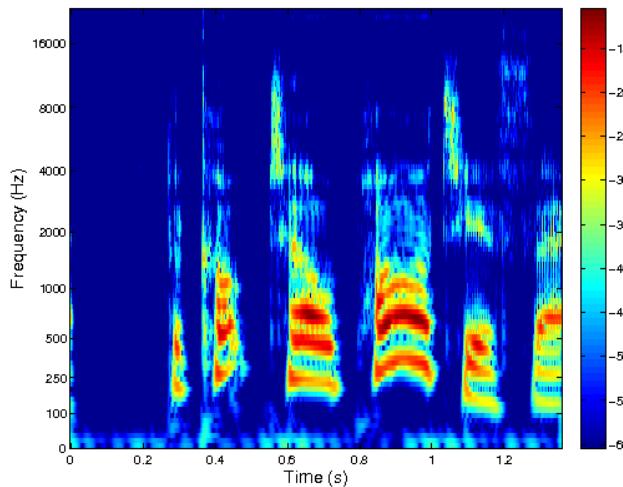


Figure 11.44: Auditory filterbank representation

Auditory filterbank representation of the spoken sentence using gammatone filters on an Erb scale. The dynamic range has been set to 50 dB, to highlight the most important features.

Relative error in iterative reconstruction, should be close to zero.

```
ans =
1.8834e-10
```

### 11.6.3 DEMO\_WFBT - Auditory filterbanks built using filterbank tree structures

This demo shows two specific constructions of wavelet filterbank trees using several M-band filterbanks with possibly different M (making the tree non-homogenous) in order to approximate auditory frequency bands and a musical scale respectively. Both wavelet trees produce perfectly reconstructable non-redundant representations.

The constructions are the following:

- 1) Auditory filterbank, approximately dividing the frequency band into intervals reminiscent of the bark scale.
- 2) Musical filterbank, approximately dividing the freq. band into intervals reminiscent of the well-tempered musical scale.

Shapes of the trees were taken from fig. 8 and fig. 9 from the reference. Sampling frequency of the test signal is 48kHz as used in the article.

**References:** [49]

#### Output

The reconstruction should be close to zero:

```
ans =
4.7845e-14
```

The reconstruction should be close to zero:

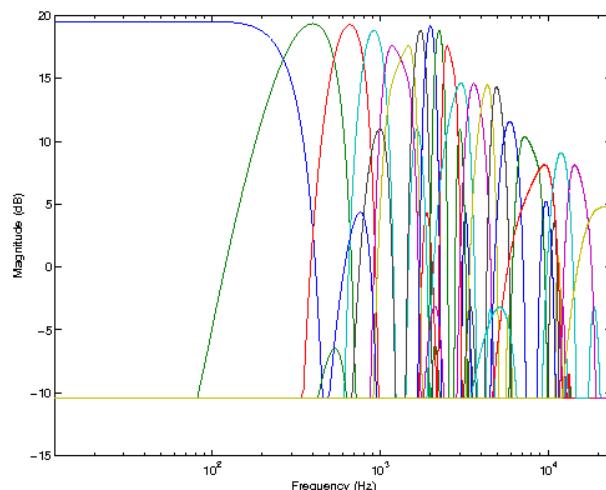


Figure 11.45: Frequency responses of the auditory filterbank.  
Both axes are in logarithmic scale.

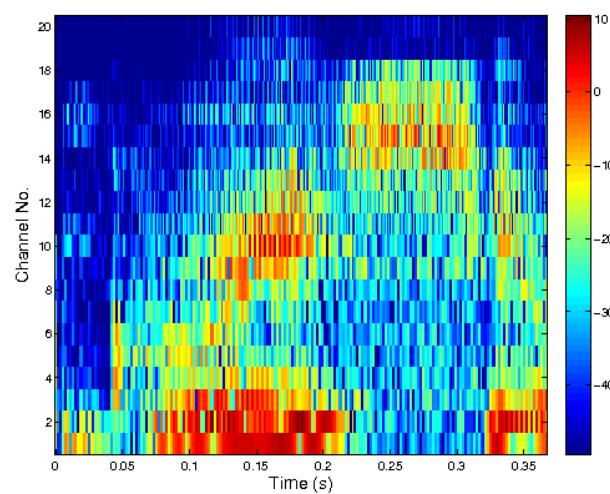


Figure 11.46: TF plot of the test signal using the auditory filterbank.

```
ans =
```

```
8.9202e-14
```

## 11.7 Block-processing demos

### 11.7.1 DEMO\_BLOCKPROC\_BASICLOOP - Basic real-time audio manipulation

#### Usage

```
demo_blockproc_basicloop('gspi.wav')
```

#### Description

For additional help call `demo_blockproc_basicloop` without arguments.

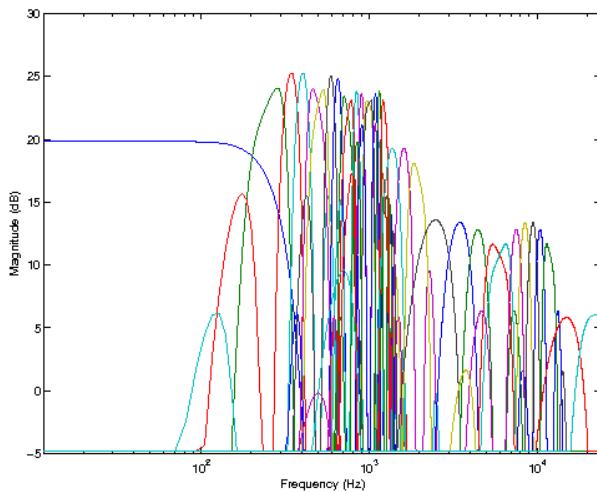


Figure 11.47: Frequency responses of the musical filterbank.  
Both axes are in logarithmic scale.

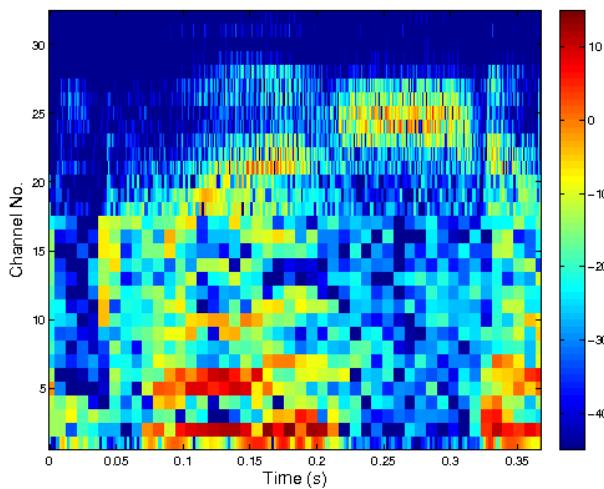


Figure 11.48: TF plot of the test signal using the musical filterbank.

The demo runs simple playback loop allowing to set gain in dB.

## Output

DEMO\_BLOCKPROC\_BASICLOOP:

To run the demo, use one of the following:

```
demo_blockproc_basicloop('gspi.wav') to play gspi.wav (any wav file will do).
demo_blockproc_basicloop('dialog') to choose the wav file via file chooser dialog GUI
demo_blockproc_basicloop(f,'fs',fs) to play from a column vector f using sampling freq fs
demo_blockproc_basicloop('playrec') to record from a mic and play simultaneously.
```

Available input and output devices can be listed by `|blockdevices|`.

Particular device can be chosen by passing additional key-value pair `'devid'`, `devid`.  
Output channels of the device can be selected by additional key-value pair `'playch'`, [c]

Input channels of the device can be selected by additional key-value pair 'recch', [ch1, ch2, ...].

### 11.7.2 DEMO\_BLOCKPROC\_PARAMEQUALIZER - Real-time equalizer demonstration

#### Usage

```
demo_blockproc_paramequalizer('gspi.wav')
```

#### Description

For additional help call demo\_blockproc\_paramequalizer without arguments.

This demonstration shows an example of a octave parametric equalizer. See chapter 5.2 in the book by Zolzer.

**References:** [90]

#### Output

DEMO\_BLOCKPROC\_PARAMEQUALIZER:

To run the demo, use one of the following:

demo\_blockproc\_paramequalizer('gspi.wav') to play gspi.wav (any wav file will do).  
 demo\_blockproc\_paramequalizer('dialog') to choose the wav file via file chooser dialog.  
 demo\_blockproc\_paramequalizer(f,'fs',fs) to play from a column vector f using sampling frequency fs.  
 demo\_blockproc\_paramequalizer('playrec') to record from a mic and play simultaneously.

Available input and output devices can be listed by |blockdevices|.

Particular device can be chosen by passing additional key-value pair 'devid', devid.

Output channels of the device can be selected by additional key-value pair 'playch', [ch1, ch2, ...].

Input channels of the device can be selected by additional key-value pair 'recch', [ch1, ch2, ...].

### 11.7.3 DEMO\_BLOCKPROC\_DENOISING - Variable coefficients thresholding

#### Usage

```
demo_blockproc_denoising('gspi.wav')
```

#### Description

For additional help call demo\_blockproc\_denoising without arguments.

The present demo allows you to set the coefficient threshold during the playback using the control panel.

#### Output

DEMO\_BLOCKPROC\_DENOISING:

To run the demo, use one of the following:

demo\_blockproc\_denoising('gspi.wav') to play gspi.wav (any wav file will do).  
 demo\_blockproc\_denoising('dialog') to choose the wav file via file chooser dialog GUI.  
 demo\_blockproc\_denoising(f,'fs',fs) to play from a column vector f using sampling frequency fs.  
 demo\_blockproc\_denoising('playrec') to record from a mic and play simultaneously.

Available input and output devices can be listed by |blockdevices|.

Particular device can be chosen by passing additional key-value pair 'devid', devid.

Output channels of the device can be selected by additional key-value pair 'playch', [ch1, ch2, ...].

Input channels of the device can be selected by additional key-value pair 'recch', [ch1, ch2, ...].

### 11.7.4 DEMO\_BLOCKPROC\_SLIDINGSGRAM - Basic real-time rolling spectrogram visualization

#### Usage

```
demo_blockproc_slidingsgram('gspi.wav')
```

#### Description

For additional help call `demo_blockproc_slidingsgram` without arguments.

This demo shows a simple rolling spectrogram of whatever is specified in source.

#### Output

`DEMO_BLOCKPROC_SLIDINGSGRAM`:

To run the demo, use one of the following:

```
demo_blockproc_slidingsgram('gspi.wav') to play gspi.wav (any wav file will do).
demo_blockproc_slidingsgram('dialog') to choose the wav file via file chooser dialog
demo_blockproc_slidingsgram(f,'fs',fs) to play from a column vector f using sampling f
demo_blockproc_slidingsgram('playrec') to record from a mic and play simultaneously.
```

Available input and output devices can be listed by `|blockdevices|`.

Particular device can be chosen by passing additional key-value pair `'devid'`, `devid`.  
 Output channels of the device can be selected by additional key-value pair `'playch'`, [`ch`].  
 Input channels of the device can be selected by additional key-value pair `'recch'`, [`ch`].

### 11.7.5 DEMO\_BLOCKPROC\_SLIDINGCQT - Basic real-time rolling CQT-spectrogram visualization

#### Usage

```
demo_blockproc_slidingcqt('gspi.wav')
```

#### Description

For additional help call `demo_blockproc_slidingcqt` without arguments.

This demo shows a simple rolling CQT-spectrogram of whatever is specified in source.

#### Output

`DEMO_BLOCKPROC_SLIDINGCQT`:

To run the demo, use one of the following:

```
demo_blockproc_slidingcqt('gspi.wav') to play gspi.wav (any wav file will do).
demo_blockproc_slidingcqt('dialog') to choose the wav file via file chooser dialog GU
demo_blockproc_slidingcqt(f,'fs',fs) to play from a column vector f using sampling fre
demo_blockproc_slidingcqt('playrec') to record from a mic and play simultaneously.
```

Available input and output devices can be listed by `|blockdevices|`.

Particular device can be chosen by passing additional key-value pair `'devid'`, `devid`.  
 Output channels of the device can be selected by additional key-value pair `'playch'`, [`ch`].  
 Input channels of the device can be selected by additional key-value pair `'recch'`, [`ch`].

### 11.7.6 DEMO\_BLOCKPROC\_SLIDINGERBLETS - Basic real-time rolling erblet-spectrogram visualization

#### Usage

```
demo_blockproc_slidingerblets('gspi.wav')
```

#### Description

For additional help call demo\_blockproc\_slidingerblets without arguments.

This demo shows a simple rolling erblet-spectrogram of whatever is specified in source.

#### Output

DEMO\_BLOCKPROC\_SLIDINGERBLETS:

To run the demo, use one of the following:

```
demo_blockproc_slidingerblets('gspi.wav') to play gspi.wav (any wav file will do).
demo_blockproc_slidingerblets('dialog') to choose the wav file via file chooser dialog
demo_blockproc_slidingerblets(f,'fs',fs) to play from a column vector f using sampling fs
demo_blockproc_slidingerblets('playrec') to record from a mic and play simultaneously
```

Available input and output devices can be listed by |blockdevices|.

Particular device can be chosen by passing additional key-value pair 'devid', devid.

Output channels of the device can be selected by additional key-value pair 'playch', [ch1, ch2, ...].

Input channels of the device can be selected by additional key-value pair 'recch', [ch1, ch2, ...].

### 11.7.7 DEMO\_BLOCKPROC\_DGTEQUALIZER - Real-time audio manipulation in the transform domain

#### Usage

```
demo_blockproc_dgtequalizer('gspi.wav')
```

#### Description

For additional help call demo\_blockproc\_dgtequalizer without arguments.

This script demonstrates a real-time Gabor coefficient manipulation. Frequency bands of Gabor coefficients are multiplied (weighted) by values taken from sliders having a similar effect as a octave equalizer. The shown spectrogram is a result of a re-analysis of the synthesized block to show a frequency content of what is actually played.

#### Output

DEMO\_BLOCKPROC\_DGTEQUALIZER:

To run the demo, use one of the following:

```
demo_blockproc_dgtequalizer('gspi.wav') to play gspi.wav (any wav file will do).
demo_blockproc_dgtequalizer('dialog') to choose the wav file via file chooser dialog
demo_blockproc_dgtequalizer(f,'fs',fs) to play from a column vector f using sampling fs
demo_blockproc_dgtequalizer('playrec') to record from a mic and play simultaneously
```

Available input and output devices can be listed by |blockdevices|.

Particular device can be chosen by passing additional key-value pair 'devid', devid.

Output channels of the device can be selected by additional key-value pair 'playch', [ch1, ch2, ...].

Input channels of the device can be selected by additional key-value pair 'recch', [ch1, ch2, ...].

## 11.7.8 DEMO\_BLOCKPROC\_EFFECTS - Various vocoder effects using DGT

### Usage

```
demo_blockproc_effects('gspi.wav')
```

### Description

For additional help call `demo_blockproc_effects` without arguments. This demo works correctly only with the sampling rate equal to 44100 Hz.

This script demonstrates several real-time vocoder effects. Namely:

- 1) Robotization effect: Achieved by doing DGT reconstruction using absolute values of coefficients only.
- 2) Whisperization effect: Achieved by randomizing phase of DGT coefficients.
- 3) Pitch shifting effect: Achieved by stretching/compressing coefficients along frequency axis.
- 4) Audio morphing: Input is morphed with a background sound such that the phase of DGT coefficients is substituted by phase of DGT coefficients of the background signal. File `beat.wav` (at 44,1kHz) (any sound will do) is expected to be in the search path, otherwise the effect will fail.

This demo was created for the Lange Nacht der Forschung 4.4.2014 event.

### Output

`DEMO_BLOCKPROC_EFFECTS`:

To run the demo, use one of the following:

```
demo_blockproc_effects('gspi.wav') to play gspi.wav (any wav file will do).
demo_blockproc_effects('dialog') to choose the wav file via file chooser dialog GUI.
demo_blockproc_effects(f,'fs',fs) to play from a column vector f using sampling frequency fs.
demo_blockproc_effects('playrec') to record from a mic and play simultaneously.
```

Available input and output devices can be listed by `|blockdevices|`. Particular device can be chosen by passing additional key-value pair `'devid'`, `devid`. Output channels of the device can be selected by additional key-value pair `'playch'`, `[ch]`. Input channels of the device can be selected by additional key-value pair `'recch'`, `[ch]`.

# Bibliography

- [1] A. Abdelnour. Dense grid framelets with symmetric lowpass and bandpass filters. In *Signal Processing and Its Applications, 2007. ISSPA 2007. 9th International Symposium on*, volume 172, pages 1–4, 2007.
- [2] A. F. Abdelnour and I. W. Selesnick. Symmetric nearly shift-invariant tight frame wavelets. *IEEE Transactions on Signal Processing*, 53(1):231–239, 2005.
- [3] F. Abdelnour. Symmetric wavelets dyadic sibling and dual frames. *Signal Processing*, 92(5):1216 – 1229, 2012.
- [4] F. Abdelnour and I. W. Selesnick. Symmetric nearly orthogonal and orthogonal nearly symmetric wavelets. *The Arabian Journal for Science and Engineering*, 29(2C):3 – 16, 2004.
- [5] A. Aertsen and P. Johannesma. Spectro-temporal receptive fields of auditory neurons in the grassfrog. I. Characterization of tonal and natural stimuli. *Biol. Cybern.*, 38:223–234, 1980.
- [6] S. Akkarakaran and P. Vaidyanathan. Nonuniform filter banks: New results and open problems. In P. M. C.K. Chui and L. Wuytack, editors, *Studies in Computational Mathematics: Beyond Wavelets*, volume 10, pages 259 –301. Elsevier B.V., 2003.
- [7] O. Alkin and H. Caglar. Design of efficient M-band coders with linear-phase and perfect-reconstruction properties. *Signal Processing, IEEE Transactions on*, 43(7):1579 –1590, jul 1995.
- [8] F. Auger and P. Flandrin. Improving the readability of time-frequency and time-scale representations by the reassignment method. *IEEE Trans. Signal Process.*, 43(5):1068–1089, 1995.
- [9] P. Balazs, M. Dörfler, F. Jaitlet, N. Holighaus, and G. A. Velasco. Theory, implementation and applications of nonstationary Gabor frames. *J. Comput. Appl. Math.*, 236(6):1481–1496, 2011.
- [10] I. Bayram and I. Selesnick. On the dual-tree complex wavelet packet and  $m$ -band transforms. *Signal Processing, IEEE Transactions on*, 56(6):2298–2310, June 2008.
- [11] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. Img. Sci.*, 2(1):183–202, Mar. 2009.
- [12] H. Bölcskei, H. G. Feichtinger, K. Gröchenig, and F. Hlawatsch. Discrete-time Wilson expansions. In *Proc. IEEE-SP 1996 Int. Sympos. Time-Frequency Time-Scale Analysis*, june 1996.
- [13] H. Bölcskei and F. Hlawatsch. Oversampled Wilson-type cosine modulated filter banks with linear phase. In *Asilomar Conf. on Signals, Systems, and Computers*, pages 998–1002, nov 1996.
- [14] H. Bölcskei and F. Hlawatsch. Discrete Zak transforms, polyphase transforms, and applications. *IEEE Trans. Signal Process.*, 45(4):851–866, april 1997.
- [15] H. Bölcskei, F. Hlawatsch, and H. G. Feichtinger. Frame-theoretic analysis of oversampled filter banks. *Signal Processing, IEEE Transactions on*, 46(12):3256–3268, 2002.
- [16] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends Mach. Learn.*, 3(1):1–122, Jan. 2011.

- [17] A. Bultheel and S. Martínez. Computation of the Fractional Fourier Transform. *Appl. Comput. Harmon. Anal.*, 16(3):182–202, 2004.
- [18] R. Carmona, W. Hwang, and B. Torrésani. *Practical Time-Frequency Analysis: continuous wavelet and Gabor transforms, with an implementation in S*, volume 9 of *Wavelet Analysis and its Applications*. Academic Press, San Diego, 1998.
- [19] R. Carmona, W. Hwang, and B. Torrésani. Multiridge detection and time-frequency reconstruction. *IEEE Trans. Signal Process.*, 47:480–492, 1999.
- [20] E. Chassande-Mottin, I. Daubechies, F. Auger, and P. Flandrin. Differential reassignment. *Signal Processing Letters, IEEE*, 4(10):293–294, 1997.
- [21] J. Cooley and J. Tukey. An algorithm for the machine calculation of complex Fourier series. *Math. Comput.*, 19(90):297–301, 1965.
- [22] I. Daubechies. *Ten Lectures on Wavelets*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992.
- [23] I. Daubechies, M. Defrise, and C. De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications in Pure and Applied Mathematics*, 57:1413–1457, 2004.
- [24] I. Daubechies, S. Jaffard, and J. Journé. A simple Wilson orthonormal basis with exponential decay. *SIAM J. Math. Anal.*, 22:554–573, 1991.
- [25] R. Decorsière and P. L. Søndergaard. Modulation filtering using an optimization approach to spectrogram reconstruction. In *Proceedings of the Forum Acousticum*, 2011.
- [26] B. Dumitrescu, I. Bayram, and I. W. Selesnick. Optimization of symmetric self-hilbertian filters for the dual-tree complex wavelet transform. *IEEE Signal Process. Lett.*, 15:146–149, 2008.
- [27] G. Fant. Analysis and synthesis of speech processes. In B. Malmberg, editor, *Manual of phonetics*. North-Holland, 1968.
- [28] H. G. Feichtinger, M. Hazewinkel, N. Kaiblinger, E. Matusiak, and M. Neuhauser. Metaplectic operators on  $c^n$ . *The Quarterly Journal of Mathematics*, 59(1):15–28, 2008.
- [29] H. G. Feichtinger and T. Strohmer, editors. *Gabor Analysis and Algorithms*. Birkhäuser, Boston, 1998.
- [30] J. Flanagan, D. Meinhart, R. Golden, and M. Sondhi. Phase Vocoder. *The Journal of the Acoustical Society of America*, 38:939, 1965.
- [31] M. Frigo and S. G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".
- [32] S. Ghosh, A. Sayeed, and R. Baraniuk. Improved wavelet denoising via empirical Wiener filtering. In *Proceedings of SPIE*, volume 3169, pages 389–399. San Diego, CA, 1997.
- [33] B. R. Glasberg and B. Moore. Derivation of auditory filter shapes from notched-noise data. *Hearing Research*, 47(1-2):103, 1990.
- [34] R. Gopinath and C. Burrus. On cosine-modulated wavelet orthonormal bases. *Image Processing, IEEE Transactions on*, 4(2):162–176, Feb 1995.
- [35] D. Griffin and J. Lim. Signal estimation from modified short-time Fourier transform. *IEEE Trans. Acoust. Speech Signal Process.*, 32(2):236–243, 1984.
- [36] K. Gröchenig. *Foundations of Time-Frequency Analysis*. Birkhäuser, 2001.
- [37] A. Grossmann, M. Holschneider, R. Kronland-Martinet, and J. Morlet. Detection of abrupt changes in sound signals with the help of wavelet transforms. *Inverse Problem*, pages 281–306, 1987.

- [38] F. Harris. On the use of windows for harmonic analysis with the discrete Fourier transform. *Proceedings of the IEEE*, 66(1):51 – 83, jan 1978.
- [39] N. Holighaus, M. Dörfler, G. A. Velasco, and T. Grill. A framework for invertible, real-time constant-Q transforms. *IEEE Transactions on Audio, Speech and Language Processing*, 21(4):775 – 785, 2013.
- [40] M. Holschneider, R. Kronland-Martinet, J. Morlet, and P. Tchamitchian. A real-time algorithm for signal analysis with the help of the wavelet transform. In *Wavelets. Time-Frequency Methods and Phase Space*, volume 1, page 286, 1989.
- [41] A. J. E. M. Janssen. Duality and biorthogonality for discrete-time Weyl-Heisenberg frames. Unclassified report, Philips Electronics, 002/94.
- [42] A. J. E. M. Janssen and T. Strohmer. Hyperbolic secants yield Gabor frames. *Appl. Comput. Harmon. Anal.*, 12(2):259–267, 2002.
- [43] S. Jayant and P. Noll. *Digital Coding of Waveforms: Principles and Applications to Speech and Video*. Prentice Hall, 1990.
- [44] N. Kingsbury. Complex wavelets for shift invariant analysis and filtering of signals. *Applied and Computational Harmonic Analysis*, 10(3):234 – 253, 2001.
- [45] N. Kingsbury. Design of q-shift complex wavelets for image processing using frequency domain energy minimization. In *Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on*, volume 1, pages I–1013–16 vol.1, Sept 2003.
- [46] N. G. Kingsbury. A dual-tree complex wavelet transform with improved orthogonality and symmetry properties. In *ICIP*, pages 375–378, 2000.
- [47] M. Kowalski. Sparse regression using mixed norms. *Appl. Comput. Harmon. Anal.*, 27(3):303–324, 2009.
- [48] M. Kowalski and B. Torrésani. Sparsity and persistence: mixed norms provide simple signal models with dependent coefficients. *Signal, Image and Video Processing*, 3(3):251–264, 2009.
- [49] F. Kurth and M. Clausen. Filter bank tree and M-band wavelet packet algorithms in audio signal processing. *Signal Processing, IEEE Transactions on*, 47(2):549–554, Feb 1999.
- [50] J. Lim and A. Oppenheim. Enhancement and bandwidth compression of noisy speech. *Proceedings of the IEEE*, 67(12):1586–1604, 1979.
- [51] T. Lin, S. Xu, Q. Shi, and P. Hao. An algebraic construction of orthonormal M-band wavelets with perfect reconstruction. *Applied mathematics and computation*, 172(2):717–730, 2006.
- [52] Y.-P. Lin and P. Vaidyanathan. Linear phase cosine modulated maximally decimated filter banks with perfectreconstruction. *IEEE Trans. Signal Process.*, 43(11):2525–2539, 1995.
- [53] D. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.
- [54] S. Mallat. *A wavelet tour of signal processing*. Academic Press, San Diego, CA, 1998.
- [55] S. Mallat. *A Wavelet Tour of Signal Processing, Third Edition: The Sparse Way*. Academic Press, 3rd edition, 2008.
- [56] S. Mallat and Z. Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Trans. Signal Process.*, 41(12):3397–3415, 1993.
- [57] S. G. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 11(7):674–693, July 1989.
- [58] H. S. Malvar. *Signal Processing with Lapped Transforms*. Artech House Publishers, 1992.

- [59] B. Moore and B. Glasberg. Suggested formulae for calculating auditory-filter bandwidths and excitation patterns. *J. Acoust. Soc. Am.*, 74:750, 1983.
- [60] T. Necciari, P. Balazs, N. Holighaus, and P. L. Søndergaard. The ERBlet transform: An auditory-based time-frequency representation with perfect reconstruction. In *Proceedings of the 38th International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2013)*, pages 498–502, Vancouver, Canada, May 2013. IEEE.
- [61] A. Nuttall. Some windows with very good sidelobe behavior. *IEEE Trans. Acoust. Speech Signal Process.*, 29(1):84–91, 1981.
- [62] A. V. Oppenheim and R. W. Schafer. *Discrete-time signal processing*. Prentice Hall, Englewood Cliffs, NJ, 1989.
- [63] H. M. Ozaktas, Z. Zalevsky, and M. A. Kutay. *The Fractional Fourier Transform*. John Wiley and Sons, 2001.
- [64] N. Perraudin, P. Balazs, and P. L. Søndergaard. A fast Griffin-Lim algorithm. In *Applications of Signal Processing to Audio and Acoustics (WASPAA), 2013 IEEE Workshop on*, pages 1–4, Oct 2013.
- [65] A. A. Petrosian and F. G. Meyer, editors. *Wavelets in Signal and Image Analysis: From Theory to Practice*, chapter The Double Density DWT, pages 39–66. Kluwer, 1 edition, 2001.
- [66] J. P. Princen and A. B. Bradley. Analysis/synthesis filter bank design based on time domain aliasing cancellation. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-34(5):1153–1161, 1986.
- [67] J. P. Princen, A. W. Johnson, and A. B. Bradley. Subband/transform coding using filter bank designs based on time domain aliasing cancellation. *Proceedings - ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 2161–2164, 1987.
- [68] M. Puckette. Phase-locked vocoder. *Applications of Signal Processing to Audio and Acoustics, 1995., IEEE ASSP Workshop on*, pages 222 –225, 1995.
- [69] L. Rabiner, R. Schafer, and C. Rader. The chirp Z-transform algorithm. *Audio and Electroacoustics, IEEE Transactions on*, 17(2):86–92, 1969.
- [70] K. Rao and P. Yip. *Discrete Cosine Transform, Algorithms, Advantages, Applications*. Academic Press, 1990.
- [71] O. Rioul and P. Duhamel. A remez exchange algorithm for orthonormal wavelets. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 41(8):550 –560, aug 1994.
- [72] C. Schörkhuber, A. Klapuri, N. Holighaus, and M. Dörfler. A Matlab Toolbox for Efficient Perfect Reconstruction Time-Frequency Transforms with Log-Frequency Resolution. In *Audio Engineering Society Conference: 53rd International Conference: Semantic Audio*. Audio Engineering Society, 2014.
- [73] I. Selesnick. The double-density dual-tree DWT. *Signal Processing, IEEE Transactions on*, 52(5):1304–1314, May 2004.
- [74] I. Selesnick. A higher density discrete wavelet transform. *IEEE Transactions on Signal Processing*, 54(8):3039–3048, 2006.
- [75] I. Selesnick. L1-Norm Penalized Least Squares with SALSA. *OpenStax\_CNX*, Jan. 2014.
- [76] I. Selesnick and A. Abdelnour. Symmetric wavelet tight frames with two generators. *Appl. Comput. Harmon. Anal.*, 17(2):211–225, 2004.
- [77] I. Selesnick, R. Baraniuk, and N. Kingsbury. The dual-tree complex wavelet transform. *Signal Processing Magazine, IEEE*, 22(6):123 – 151, nov. 2005.
- [78] P. L. Søndergaard. Symmetric, discrete fractional splines and Gabor systems. *preprint*, 2008.

- [79] P. L. Søndergaard. LTFAT-note 17: Next fast FFT size. Technical report, Technical University of Denmark, 2011.
- [80] S. Stevens, J. Volkmann, and E. Newman. A scale for the measurement of the psychological magnitude pitch. *J. Acoust. Soc. Am.*, 8:185, 1937.
- [81] P. Sysel and P. Rajmic. Goertzel algorithm generalized to non-integer multiples of fundamental frequency. *EURASIP Journal on Advances in Signal Processing*, 2012(1):56, 2012.
- [82] C. Taswell. Near-best basis selection algorithms with non-additive information cost functions. In *Proceedings of the IEEE International Symposium on Time-Frequency and Time-Scale Analysis*, pages 13–16. IEEE Press, 1994.
- [83] H. Traunmüller. Analytical expressions for the tonotopic sensory scale. *J. Acoust. Soc. Am.*, 88:97, 1990.
- [84] G. A. Velasco, N. Holighaus, M. Dörfler, and T. Grill. Constructing an invertible constant-Q transform with non-stationary Gabor frames. *Proceedings of DAFX11*, 2011.
- [85] T. Werther, Y. Eldar, and N. Subbana. Dual Gabor Frames: Theory and Computational Aspects. *IEEE Trans. Signal Process.*, 53(11), 2005.
- [86] E. Wesfreid and M. Wickerhauser. Adapted local trigonometric transforms and speech processing. *IEEE Trans. Signal Process.*, 41(12):3596–3600, 1993.
- [87] M. V. Wickerhauser. Lectures on wavelet packet algorithms. In *INRIA Lecture notes*. Citeseer, 1991.
- [88] M. V. Wickerhauser. *Adapted wavelet analysis from theory to software*. Wellesley-Cambridge Press, Wellesley, MA, 1994.
- [89] G. Yu, S. Mallat, and E. Bacry. Audio Denoising by Time-Frequency Block Thresholding. *IEEE Trans. Signal Process.*, 56(5):1830–1839, 2008.
- [90] U. Zolzer. *Digital Audio Signal Processing*. John Wiley and Sons Ltd, 2 edition, 2008.
- [91] E. Zwicker. Subdivision of the audible frequency range into critical bands (frequenzgruppen). *J. Acoust. Soc. Am.*, 33(2):248–248, 1961.