Perfect reconstruction transforms in the

(Large Time Frequency Analysis Toolbox)

Zdeněk Průša 21.11.2014
LTFAT is

- ... Matlab/Octave toolbox for working with time-frequency analysis and synthesis.
- ... currently in version 2.0.0.
- ... baby of Peter L. Søndergaard since 2004.
- ... \( \sim 52k \) lines of code, 60% Matlab, 27% C, 8% C++, and bit of java JAVA etc., excluding thirdparty code and comments.
- ... an official GNU Octave package since 1.4.2.
  
  ```
octave:1> pkg install -forge ltfat
```
Currently also available as a Debian (Jessie) package.
  
  ```
$sudo apt-get install octave-ltfat$
```
- ... downloaded 80 times per week on average.
- ... free software (in both meanings).
- and is celebrating 10th anniversary this year!
LTFAT is unique because

- ... uses frame theory as the underlying mathematical abstraction
  ⇒ perfect reconstruction.
  ⇒ modification of coefficients (e.g. frame multiplier).
- ... contains a plethora of TF transforms: DGT, NSDGT, Discrete Wavelets, filterbanks, CQT, Erblets etc.
- ... uses fast algorithms whenever possible. Implementations in both Matlab and C.
- ... provides unified programming interface for working with different types of frames in $\mathbb{C}^L$ (NO explicit matrices).
- ... allows doing real-time audio processing directly from Matlab and Octave.
(Show something before everyone leaves....)

DEMO
Talk Overview

- Frames framework
  - Allows working with different types of frames in a uniform manner.
  - Exploits fast algorithms whenever possible.
  - Avoids explicit creation of matrices.
- Block processing in LTFAT
  - Makes real-time audio in Matlab and Octave possible.
  - Integrated with the frames framework.
  - Provides real-time visualization and controlling tools.
Frames framework – Basic usage

1. Create a frame object of the chosen type. Not fixed $L$.
2. Use analysis and synthesis operators: `frana`, `frsyn`.
3. (Optional) Investigate it’s properties: `framered`, `framebounds`, `frsynmatrix`.

Example:

```plaintext
F = frame('dgtreal','gauss',10,1000);
c = frana(F,greasy);
plotframe(F,c,'dynrange',60);
```

vs.

```plaintext
c = dgtreal(f,'gauss',10,1000);
plotdgt(c,10,'dynrange',60);
```
Frames framework – How to reconstruct

- Create a dual frame:

\[ \mathbf{F}_d = (\mathbf{F}\mathbf{F}^*)^{-1}\mathbf{F}, \quad \text{then} \quad \mathbf{F}\mathbf{F}^* = \mathbf{F}_d\mathbf{F}^* = \mathbf{I} \]

and use it’s synthesis operator \( \mathbf{F}_d = \text{framedual}(\mathbf{F}) \)

- Create a (Parseval) tight frame:

\[ \mathbf{F}_t = (\mathbf{F}\mathbf{F}^*)^{-\frac{1}{2}}\mathbf{F}, \quad \text{then} \quad \mathbf{F}_t\mathbf{F}_t^* = \mathbf{I} \]

and use it for both analysis and synthesis.
\( \mathbf{F}_t = \text{frametight}(\mathbf{F}). \)

- What if a dual frame with the same structure cannot be created? Iterative inversion of the frame operator:

\( \text{franaiter}, \text{frsyniter} \)
franabp – frame analysis as a basis pursuit problem

Basis pursuit problem:

\[ \arg \min_c \| \lambda c \|_1, \quad \text{subject to } Fc = f \]  \hspace{1cm} (1)

Split Augmented Lagrangian Shrinkage Algorithm – SALSA

1. Initialize \( c, d, \mu, \lambda \)
2. repeat
   - \( v \leftarrow \text{soft} \left( c + d, \frac{\lambda}{\mu} \right) - d \)
   - \( d \leftarrow F^*(FF^*)^{-1}(f - Fv) \)
   - \( c \leftarrow d + v \)
3. end

In LTFAT:
\[ c = \text{franabp}(F,f,\lambda); \]
franalasso – frame analysis as a LASSO problem

LASSO or Basis pursuit denoising problem:

\[
\arg\min_c ||\lambda c||_1 + \frac{1}{2}||F c - f||_2^2
\]  

(2)

Iterative Soft Thresholding Algorithm – ISTA

1. Initialize \( c, \mu, \lambda \)
2. repeat until stopping criterion is met
   - \( c \leftarrow \text{soft} \left( c + \frac{1}{\mu} F^* (f - F c), \frac{\lambda}{\mu} \right) \)

In LTFAT (actually doing the Fast ISTA):

\( c = \text{franalasso}(F,f,\text{lambda}); \)
frsynabs – synthesis without a phase

Problem

Find f knowing \( s = |F^*f| \).

Griffin-Lim algorithm

1. Initialize \( c = s \)
2. repeat until stopping criterion is met
   ▶ \( c \leftarrow s \cdot \exp(i \arg(F^*(FF^*)^{-1}Fc)) \)
3. \( f = (FF^*)^{-1}Fc \)
4. end

In LTFAT:

\( f = \text{frsynabs}(F,c) \);
Block processing routines

Build on top of:

- **Portaudio** – a free, open-source, audio I/O C/C++ library. An uniform way of working with audio I/O across operating systems. [http://www.portaudio.com](http://www.portaudio.com)

- **Playrec** – a MEX file providing an interface from Matlab to Portaudio. [http://github.com/PlayrecForMatlab](http://github.com/PlayrecForMatlab)

Simple use:

```matlab
block('playrec');
p = blockpanel({'GdB','Gain',-20,20,0,21});
while p.flag
    gain = blockpanelget(p,'GdB');
    f = blockread();
    blockplay(f*10^(gain/20));
end
blockdone(p);
```
source in block(source); may be:

- 'file.wav' name of a wav file
- 'dialog' shows the file dialog to choose a wav file.
- 'rec' input is taken from a microphone/auxillary input;
- 'playrec' loopbacks the input to the output
- data input data as columns of a matrix for each input channel

block(...,'outfile','outfile.wav'); together with blockwrite performs on-the-fly writing to a file.
block(...,'offline'); for offline processing.
Specifying system devices and channels.
Sources of delay (assuming 44.1 kHz):

- In the system driver itself; ranges 5ms–100ms.
- Introduced by the block length; \( \sim 23 \text{ ms} \) for 1024 samples.
- Processing delay, when doing e.g. DGT analysis and synthesis \( \sim \) length of the window.

Limitations

- Cannot use much shorter block lengths.
- Everything has to be done in less than \( \sim 23 \text{ ms} \) ...

We can choose to use lower (non-native) sampling rate, but that usually means higher delay in system.
Block processing and frames framework together

Conceptually:

```matlab
% Initialize block, F = frame(..), Fd = framedual(F)
while flag
    f = blockread();
    c = blockana(Fd,f);
    % Process c
    blockplot(c);
    fhat = blocksyn(F,c);
    blockplay(fhat);
end
```

Problems:

- Is it fast enough?
- How to handle blockwise analysis and synthesis?
Handling blockwise analysis and synthesis

- **Naive** approach – do analysis and synthesis with the current block only.
  + Works for any frame (provided it is fast enough).
  + No additional delay.
  - Coefficients are different.
  - Any coefficient modification will produce horrible blocking artifacts.

- **Slicing window** (metawindow) – another layer of weighted and overlapped blocks.
  + Works for any frame (provided it is fast enough).
  + Blocking artifacts reduced.
  - Modifying and plotting coefficients is not straightforward.

- **Overlap Add/Save** – known compact support of atoms can be exploited to do a proper overlapping.
  - Works only for (some) frames with compactly supported windows/filters.
  + Blocking artifacts completely avoided.
  + Straightforward modification and plotting of coefficients.