MATLAB EXPO 2016
Signal Processing Master Class

Graham Reith
Session Outline

- **Offline signal processing with MATLAB**
  - Functions, Capabilities

- **Streaming signal processing with MATLAB**
  - Structure, Performance

- **Deployed signal processing from MATLAB**
  - Prototyping, Implementation

MATLAB EXPO 2016
Session Outline

Offline signal processing with MATLAB
Functions, Capabilities

Streaming signal processing with MATLAB
Structure, Performance

Deployed signal processing from MATLAB
Prototyping, Implementation
Spectral Analysis

- Determining the Frequency Content of a Signal
  - Usually analysis based on the Fast Fourier Transform (FFT)
  - Trade-offs to manage: frequency resolution vs sidelobe attenuation
Introducing the Signal Analyzer App

Analyze signals in time and frequency and navigate through the signals using a panner.

- Spectral analysis of signals
- Time domain panning
- Enhanced command line interface allows specifying input signals and sample rates
Spectrum Controls

Control spectral leakage (Kaiser window shape factor)
Read resolution bandwidth values for the current spectral estimates
Spectrum Scope

- dsp.SpectrumAnalyzer
Scope Improvements

- **Spectral Mask in Spectrum Analyzer**
  - Compare a spectrum to a mask, e.g. to verify compliance with communication standards or design requirements
  - Use in MATLAB or Simulink

- **Now supported by MATLAB Compiler:**
  - `dsp.TimeScope`
  - `dsp.SpectrumAnalyzer`
  - `dsp.ArrayPlot`

- **Array Plot: Custom X-axis data**
  - Visualize arbitrarily-spaced data

- **Time Scope: Programmatic legend strings and autoscaling**
Channelizer and Filter-bank Spectral Analysis

Analyze and synthesize narrow subbands of a broadband signal using a polyphase FFT filter bank

- **Analysis**: `dsp.Channelizer`
  
  **Synthesis**: `dsp.ChannelSynthesizer`

- Efficient filter banks based on FFT and a polyphase structure for filtering

- Used for high-resolution spectral analysis:
  - Achieve arbitrary inter-channel separation

- Applications in 5G communications – beyond OFDM

- Use as method of `dsp.SpectrumEstimator`
Comparison: Channelizer for Spectral Analysis
Analysing Vibration Data

- Data from rotational machinery, captured using accelerometers
- Typically fundamental frequencies and harmonics vary with RPM
Order Spectrum and Order Tracking

Analyze the amplitude and spectrum of vibration signal orders

- Compute and display an order amplitude profile vs. RPM using \texttt{ordertrack}
- Compute and display an average spectrum using \texttt{orderspectrum}

\begin{align*}
\texttt{>> ordertrack(x,Fs,rpm,orderlist)} \\
\texttt{>> ordertrack(map,order,rpm,...)} \\
\texttt{>> orderspectrum(x,Fs,rpm)} \\
\texttt{>> orderspectrum(map,order)}
\end{align*}
Order Waveforms

Extract time-domain orders waveforms

- Extract decoupled time-domain waveforms for crossing (and non-crossing) orders
- Vold-Kalman Filter
  - Specify bandwidth and order
- Compute waveforms by segments to improve computation speed

>> orderlist=[0.052 0.066 0.264])
>> orderwaveform(x,fs,rpm,orderlist)
Synchrosqueezing Transforms

High resolution time-frequency spectral analysis and waveform extraction

- Compute and display a compact time-frequency representation of modes using \( \text{fsst} \)
- Extract time-frequency ridges (e.g. instantaneous frequency of each mode) using \( \text{tfridge} \)
- Reconstruct the entire plane, a band of frequencies, or a time-frequency ridge using \( \text{ifsst} \)

\[
\begin{align*}
\text{Magnitude of Fourier Synchrosqueezed Transform of Two Chirps} \\
\text{Instantaneous Frequency} \\
\text{Waveform extraction}
\end{align*}
\]

\[ \gg \ sst = \text{fsst}(x,\ldots) \]
\[ \gg \ [f,idx] = \text{tfridge}(sst,\ldots) \]
\[ \gg \ y = \text{ifsst}(sst,\ldots,\ldots,idx) \]
Finding a Signal via Similarity Search: findsignal

Locate the best matching data to your signal

- Compare real/complex vectors of arbitrary dimensions
- Popular distance metrics (Euclidean, Squared Euclidean, Symmetric Kullback-Liebler)
- Convenient normalizations (Zero-mean, Unit norm, average power) over an arbitrary sample length for both data and signal
- Time warping
- Outlier robustness

```matlab
>> data = exp(-(1:300)/100).^2).*cos(2*pi*(1:300)/100);
>> signal = sin(2*pi*(1:100)/100);
>> findsignal(data,signal)
```
Session Outline

**Offline signal processing with MATLAB**

- Functions, Capabilities

**Streaming signal processing with MATLAB**

- Structure, Performance

**Deployed signal processing from MATLAB**

- Prototyping, Implementation
Offline Processing vs Streaming Processing

- 2 Entry Points for describing Signal Processing operations in MATLAB:

  - Data analysis
  - Algorithm exploration
  - System simulation
  - Efficient implementation

```matlab
%Apply data to filter all at once
output = filter(b,a,data);

%Or, save and restore state each call
[output,state]=filter(b,a,data,state);

%Initialize filter with coefficients
lpf = dsp.FIRFilter('Numerator',b);

%Apply filter to block of data
output = lpf(data);
```
Example: RTL-SDR USB RF Receiver

```matlab
% other parameters. Change this center frequency to unveil a different band.
%
% Set initial parameters
fc = 102.5e6; % Center frequency (Hz)
FrontEndSampleRate = 1e6; % Samples per second
FrameLength = 256*20;

% Create receiver and spectrum analyzer System objects
hSDRRx = comm.SDRRTLReceiver(...
  'CenterFrequency', fc, ...
  'EnableTunerAGC', true, ...
  'SampleRate', FrontEndSampleRate, ...
  'SamplesPerFrame', FrameLength, ...
  'OutputDataType', 'double');

hSpectrum = dsp.SpectrumAnalyzer(...
  'Name', 'Actual frequency offset');
```

Sampling frequency (up to ~2.8MHz)
RF Centre Frequency (20MHz to ~1.8GHz)
+ Tuner gain parameters
Frequency correction parameters
ADS-B Receiver

Tracking Airplanes Using ADS-B Signals

Copyright 2015-2016 The MathWorks, Inc.

MATLAB EXPO 2016
Accelerating your Simulations

- Execution Engine from R2015b
- Efficient Coding Constructs
- Parallel Computing
  - Desktop
  - Computer Clusters
  - GPU Acceleration
- MATLAB to C
  - MATLAB Coder
Multithreaded Acceleration via Unfolding

Generate a multi-threaded MEX file from a MATLAB function that operates on streamed signals with dspunfold

- Acceleration workflow based on code generation
  - Requires MATLAB Coder

- Application to DSP
  - Distribute subsequent frames to different cores at the expense of latency

- Support for custom processing algorithms
Creating Parallelism by Unfolding

Normal: Execute one instance of one algorithm on the dataset

Unfolded: Replicate the algorithm to run multiple instances (copies)
- Each instance operates on a different frame of the dataset
- Each instance executes on a different thread (core)
Stateless Algorithms

F = 50, U=4

Step 1

<table>
<thead>
<tr>
<th>Frame</th>
<th>CPU1</th>
<th>CPU2</th>
<th>CPU3</th>
<th>CPU4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50</td>
<td>51</td>
<td>101</td>
<td>151</td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frame1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frame2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>150</td>
<td>201</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>Frame3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>151</td>
<td>200</td>
<td>251</td>
<td>300</td>
<td></td>
</tr>
<tr>
<td>Frame4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Step 2

<table>
<thead>
<tr>
<th>Frame</th>
<th>CPU1</th>
<th>CPU2</th>
<th>CPU3</th>
<th>CPU4</th>
</tr>
</thead>
<tbody>
<tr>
<td>201</td>
<td>250</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frame5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>251</td>
<td>300</td>
<td>351</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frame6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>301</td>
<td>350</td>
<td>400</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frame7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>351</td>
<td>400</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frame8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Speed improvement = \frac{\text{unique samples processed}}{\text{samples processed per thread}} = \frac{400}{100} = 4x

F = Frame size – minimum number of samples given to algorithm at once
U = Threads – assume each thread maps to a core
Algorithms with State

Step 1

<table>
<thead>
<tr>
<th>U1</th>
<th>U2</th>
<th>U3</th>
<th>U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>41</td>
<td>91</td>
<td>141</td>
</tr>
<tr>
<td>50</td>
<td>100</td>
<td>150</td>
<td>200</td>
</tr>
<tr>
<td>Frame1</td>
<td>Frame2</td>
<td>Frame3</td>
<td>Frame4</td>
</tr>
</tbody>
</table>

Step 2

<table>
<thead>
<tr>
<th>U1</th>
<th>U2</th>
<th>U3</th>
<th>U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>191</td>
<td>200</td>
<td>201</td>
<td>250</td>
</tr>
<tr>
<td>Frame5</td>
<td>Frame6</td>
<td>Frame7</td>
<td>Frame8</td>
</tr>
</tbody>
</table>

Efficiency = \( \frac{\text{unique samples processed}}{\text{total samples processed}} \) = \( \frac{400}{480} \) = 83%

Speedup = \( \frac{\text{unique samples processed}}{\text{samples processed per thread}} \) = \( \frac{400}{120} \) = 3.3x
Session Outline

**Offline signal processing with MATLAB**
Functions, Capabilities

**Streaming signal processing with MATLAB**
Structure, Performance

**Deployed signal processing from MATLAB**
Prototyping, Implementation
From MATLAB algorithms to real-time code

- Experiment with algorithm in MATLAB
- Architect/review/optimize MATLAB code
- **Generate real-time source C/C++ code**
- Verify/validate generated code
- Optimize generated code
Raspberry Pi with RTL-SDR

- From R2015b: Using the Raspberry Pi with the RTL-SDR receiver
- Deploy signal processing algorithms directly to the ARM processor on the Raspberry Pi, and stream in data from the RTL-SDR receiver
- Output audio, or send data to ThingSpeak, etc
Session Outline

**Offline signal processing with MATLAB**
- Functions, Capabilities

**Streaming signal processing with MATLAB**
- Structure, Performance

**Deployed signal processing from MATLAB**
- Prototyping, Implementation
Tools used in this Presentation

- MATLAB
- Signal Processing Toolbox
- DSP System Toolbox

- Communications System Toolbox
  - Support Package for RTL-SDR

- Simulink
  - Support Package for Raspberry Pi

- MATLAB Coder

MATLAB EXPO 2016
Other Sessions at MATLAB EXPO Today

Application Track 1:

- **15:45-16:15**

  **MATLAB Algorithm Development and Verification for Eurofighter Typhoon Praetorian**
  15:45–16:15
  The Praetorian Defensive Aids Sub-System (DASS) currently installed on the Eurofighter Typhoon provides protection against air-to-air and surface-to-air threats by monitoring and proactively responding to the operational environment. It contains Electronic Support Measures, missile warning, on-board electronic countermeasures, and towed radar decoys to detect, evaluate, and counter threats at maximum range.

  In this session, Neil provides an overview of the Praetorian system and explains how MATLAB® is being used as a simulation and modelling tool. He explains how automatic code generation can be configured to create loadable "apps" to provide more system flexibility and reduce the high cost associated with traditional development, verification, and maintenance of such systems.

- **16:15-17:00**

  **Modelling and Simulating RF Sensor Systems**
  16:15–17:00
  Wireless communications and radar systems are pervasive across many application fields, such as consumer electronics, aerospace and defence, and automotive. The need to improve performance while reducing the overall area and power imposes challenging system requirements. MATLAB® and Simulink® are powerful tools for the development of RF systems. In this session, Marc uses live demonstrations and examples to demonstrate how to:
  - Analyse and visualise RF sensor data
  - Develop sensor processing algorithms
  - Model RF front ends and antenna array systems
  - Stream real-world RF signals into MATLAB
Demo Stations at MATLAB EXPO Today

Real-Time Audio Processing with MATLAB: Automatic VST Plugin Generation

Using MATLAB to Explore the Internet of Things

Wireless Communication System Design Using MATLAB and Simulink

Designing, Prototyping, and Testing Video Algorithms for FPGA and SoC