ElectricMind Spectral Analysis Series

This documentation and associated software accompanies ElectricMind's blog series on spectral analysis of EEG signals. The purpose of the software is to provide a straight-forward command-line interface for the purpose of spectral analysis of EEG signals using the *discrete Fourier transform* (DFT). The software provides a set of basic methods to interface with a *fast Fourier transform* (FFT) algorithm, utilising either MATLAB or Octave base-libraries.

The available methods in the class provided minimal parametrisation, so that one may focus on the impact of parameters and applied tapers on the estimated periodogram and associated band-power measures. This is not intended to be flashy, turn-key software, it is presented with scientific aims in mind: to grasp the DFT and periodogram routines.

This software and documentation are covered by ElectricMind's terms of usage, third-party licenses where applicable, as well as the GNU General Public License (GNU GPL) for the source code. The software is distributed as free and open-source software for teaching and research purposes.

If you have any useful contributions to the software, you can email us with your suggestions: matthew@electricmind.me. Please note, this is NOT a technical support email, it's a feedback channel to better the software. We can't guarantee an rapid response, but we will attempt to respond to your queries and feedback.

Requirements

To use the class, ensure that:

1. You have a working, properly licensed, installation of MATLAB; alternatively, a working installation of Octave. The latter is an open-source platform, distributed under the GNU GPL license.

2. That the class definition and supporting datafile(s) has been downloaded and have been placed in the MATLAB/Octave current directory.

We have tested the software on MATLAB 2017a, 2015a, as well as Octave 5.1.0, all running on a Windows 10, 64-bit operating system. It has worked successfully in our testing environment. While we hope the software is useful to you, we cannot offer any guarantees, and we do not offer additional support over-and-above this documentation.

Methods

.electricMindSpectral()

Constructor method: creates a new spectral analysis object. Characteristically, this object will be assigned to an object variable in the MATLAB/Octave workspace. A newly created object contains a number of different properties related to the definitions for spectral analysis. The object properties may be accessed from the workspace in MATLAB. The theory behind the class is provided in our two-part series on spectral analysis of EEG signals. We encourage you to read our series in order to understand how we have built the class—the property labels within the class definition correspond to spectral estimation theory in Oppenheimer and Schaffer (2010) *Discrete-time Signal Processing*. International Edition. Usage:

```
>>spec = electricMindSpectral;
```

.synthSignal()

This method synthesises a random noise signal with a 100Hz sinusoidal component. This needs to be run prior to computing the periodogram. For the engineer among us: while this method is useful for demonstrating certain properties of the DFT and periodogram, it is not strictly speaking a random signal, arising from a random process. Sinusoidals in noise have a legacy in the signal-processing literature and are useful in the context of understanding DFT parametrisation. Usage:

```
>>spec.synthSignal;
```

.checkSignal()

The method prints the signal parameters to screen. It's a convenient way, especially in Octave, to check the signal has been loaded. Usage:

```
>>spec.checkSignal;
```

.loadSignal()

This method loads a single channel of EEG data into the object. The data is an occipital midline channel from an eyes-open, steady-state recording. Each time this method is called, the object fields will be re-initialised. This needs to be run prior to computing the periodogram. Usage:

```
>>spec.loadSignal
```

.periodogram(L,R,window)

This method computes the Welch's periodogram spectral-estimate and plots the results. There are three parameters:

L *– window length in sample points*

R *– the shift of the window in sample points*

Window *– 'hamming', 'reactangular'*

Usage: >>spec.periodogram(1024,1024,'hamming')

.getParameters()

Prints to screen the parameters used in the periodogram analysis. These include, relevant features of the signal and the parameters of the periodogram routine.

>>spec.getParameters

.computeBandPower(lower,upper,scale)

This method extracts total band-power and average band-power from the estimated periodogram. There are three parameters:

Lower *– the lower bound of the band of interest*

Upper *– the upper bound of the band of interest*

Scale *– whether to use 'linear' scaled measurement (absolute power), or a 'log' scaled measurement (relative power).*

Usage: >> spec.computeBandPower(8,12,'linear')

Note: when parameterised, such that *Lower* = *Upper*, a single frequency bin estimate is extracted from the estimated periodogram. Usage:

>> spec.computeBandPower(4,4,'linear')

Workflow Examples

These are the workflow examples in Part 2 of the spectral analysis series. The user can change the window-length, shift parameters, change the taper, and explore the changes in the bias and variance of the estimated periodogram. This is useful if one wants to really gain an understanding of spectral analysis of an EEG signal.

A noisy signal

This workflow generates a pseudo-random, white-noise process with a sinusoidal at 100Hz. Welch's periodogram is computed using a 2-second, non-overlapping window (1024 datum-points, Fs = 512Hz).

You can play with the parameters to see what happens to the spectral estimate. Lengthening the window creates a less biased, more variable (noisy) estimate. Reducing the window length and reducing how many datum-points the window is slid by, leads to a less variable (less noisy), but a more bias estimate. On can also change the taper and observe the influence on the estimated spectrum.

All the parameters of the periodogram routine are stored and may be queried from the command line at any time.

```
>> spec  = electricMindSpectral(); % create a new spectral analysis object
>> spec.synthSignal % synthesise a synthetic signal
>> spec.periodogram(1024,1024,'hamming'); % compute a periodogram estimate
>> spec.periodogram(1024,1024,'rectangular'); % observe the change in frequency resolution
>> spec.periodogram(2048,2048,'hamming'); % a less biased, more variable estimate
>> spec.periodogram(512,512,'hamming'); % a more biased, less variable estimate
>> spec.getParameters % view the properties of the periodogram routine
```

EEG signal analysis

This workflow loads an EEG signal from the occipital midline electrode. The recording was done while the participant was in an eyes-open state. Welch's periodogram is computed using different parameters to demonstrate the estimation bias-variance trade-offs in signal parametrisation. Band power is also extracted for each modified periodogram to demonstrate the changes in band power and single-point power estimates.

```
>> spec  = electricMindSpectral(); % create a new spectral analysis object
>> spec.loadSignal % load an EEG signal
>> spec.periodogram(400,400,'hamming'); % compute a periodogram estimate
>> spec.computeBandPower(8,10,'log'); % extract narrow band power
>> spec.getParameters % display parameters for the periodogram estimate
>> spec.periodogram(800,800,'hamming'); % a less biased, more variable estimate
>> spec.computeBandPower(8,10,'log'); % extract narrow band power
>> spec.computeBandPower(9,9,'log'); % extract single frequency bin estimate
>> spec.getParameters % display parameters for the periodogram estimate
>> spec.periodogram(200,200); % a more biased, less variable estimate
>> spec.computeBandPower(8,10,'linear'); % extract narrow band
>> specAnalysis.getParameters % display parameters for the periodogram estimate
```