KARLSRUHE INSTITUTE OF TECHNOLOGY
Communications Engineering Lab
Prof. Dr.rer.nat. Friedrich K. Jondral

Implementing Thomson's Multitaper Method in GNU Radio

Bachelorarbeit

Gerald N. Baier

Hauptreferent	:	Prof. Dr.rer.nat. Friedrich K. Jondral
Betreuer	:	Jens P. Elsner, M.Sc.

Beginn	:	15.04.2010
Abgabe	:	09.09.2010

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig und ohne unzulässige fremde Hilfe angefertigt habe. Die verwendeten Literaturquellen sind im Literaturverzeichnis vollständig zitiert.

Karlsruhe, den 09.09.2010

Gerald N. Baier

Abstract

This thesis covers the fundamentals of the multitaper method. It is shown why Discrete Prolate Spheroidal Sequences (DPSS) are so appealing as data windows. An algorithm to compute DPSS of arbitrary length can be found in section 3.1. Two test cases to verify the correctness of the spectrum estimators results are introduced.

We will start with the fundamentals of multitapering. The basic idea of multitapering is using multiple spectrum estimates, called eigenspectra and weighting them in order to reduce the variance of the total spectrum estimate. The eigenspectra are computed by applying a data taper to the input data and they are a smoothed version of the true spectrum. We use DPSS as data tapers due to the high energy concentrations of the corresponding spectral windows. The DPSS are the eigenvectors of the $N \times N$ matrix $\mathbf{A}$ the elements of which are given by

$$\mathbf{A_{n,m}} = \frac{\sin(2\pi W(m-n))}{\pi(m-n)}.$$

In addition to their good energy concentration the DPSS produce uncorrelated eigenspectra if the true spectrum $S(f)$ is slowly varying compared to the bandwidth of their spectral windows. However, not every DPSS has an optimal energy concentration λ_k

$$\lambda_k = \frac{\int_{-W}^{W} |U_k(f)|^2 \, df}{\int_{-\frac{1}{2}}^{\frac{1}{2}} |U_k(f)|^2 \, df}$$

where $2W$ is the bandwidth of the spectral windows. Figure 0.1 shows the energy concentration of the DPSS for the sample size $N = 128$ and $NW = 4$.

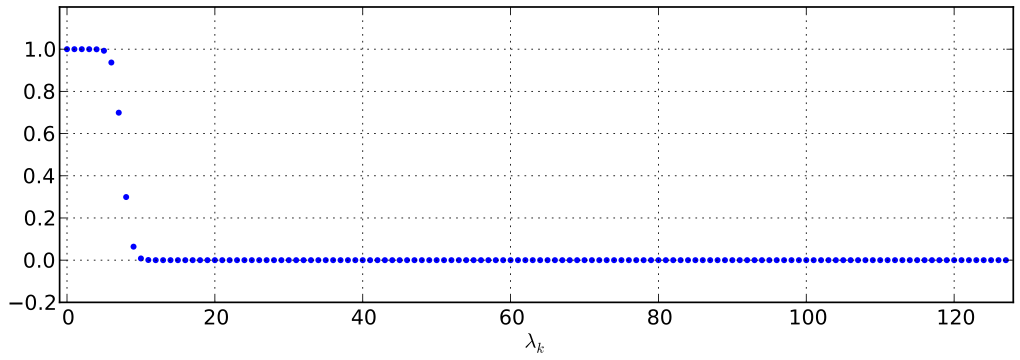
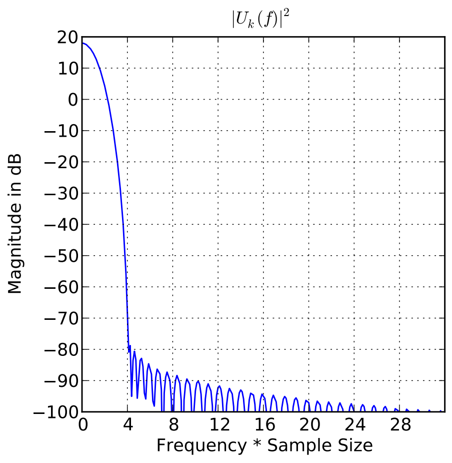
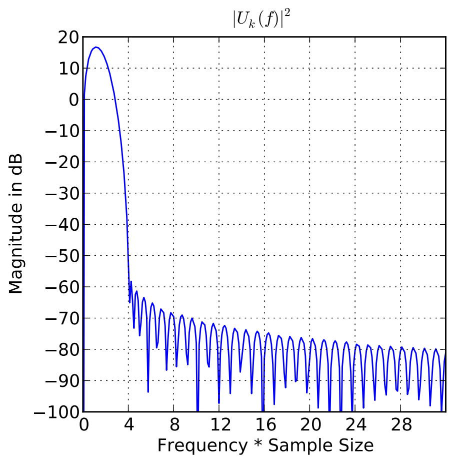


Figure 0.1.: Energy Concentration of the DPSS for $N = 128$ and $NW = 4$

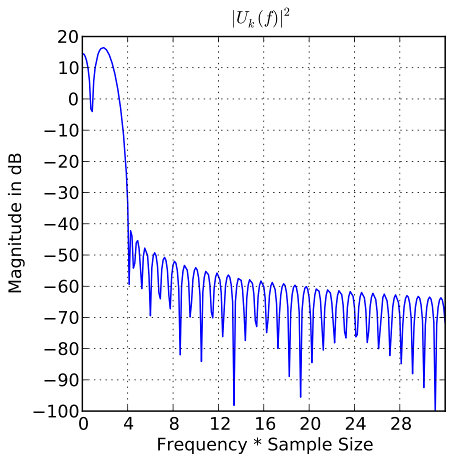
Obviously only the DPSS with an energy concentration close to one produce good eigenspectra. The first $(2NW - 1)$ DPSS fulfill this requirement. The quality of the spectral windows and with them the quality of the eigenspectra deteriorates with increasing k. Figure 0.2 shows the first four spectral windows for $N = 128$ and $NW = 4$. The x-axis is in units of frequency times sample size. The deterioration of the spectral windows can be seen clearly by the increasing side lobe levels of the higher order spectral windows.



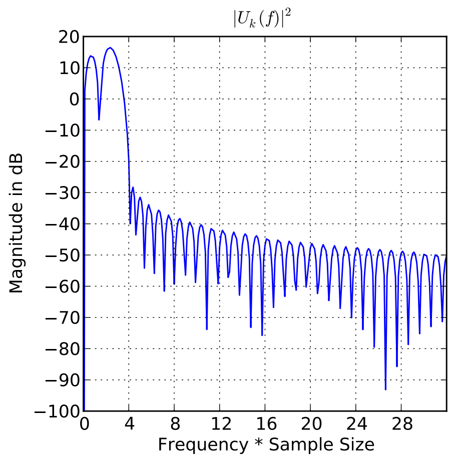
(a) $k = 0$



(b) $k = 1$



(c) $k = 2$



(d) $k = 3$

Figure 0.2.: The first four spectral windows for $N = 128$ and $NW = 4$

The varying quality of the eigenspectra motivates different weighting methods:

- Unity weighting
- Eigenvalue weighting
- Adaptive weighting

Unity weighting takes the average of all eigenspectra. The difference to eigenvalue weighting, where the eigenspectra are weighted according to their energy concentrations, is marginal. Adaptive weighting utilizes frequency dependant weights.

Two test cases will be used to highlight the performance of the multitaper method. In Figure 0.3 the true spectrum of Marple's test case [5] can be seen. It consists of four complex sinusoids and complex coloured noise. For $N = 256$, $NW = 3$ and five tapers spectrum

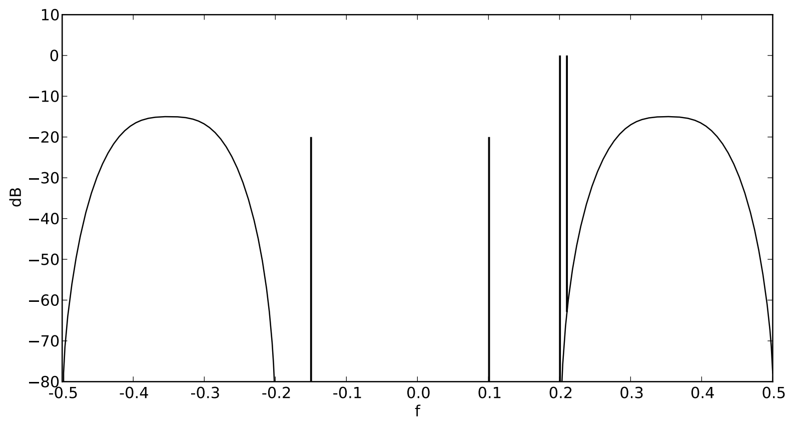


Figure 0.3.: The true spectrum of Marple's test case

estimates gained by averaging 1000 runs can be seen in Figure 0.4. They show the superior performance of adaptive weighting compared to unity weighting. The reduction in

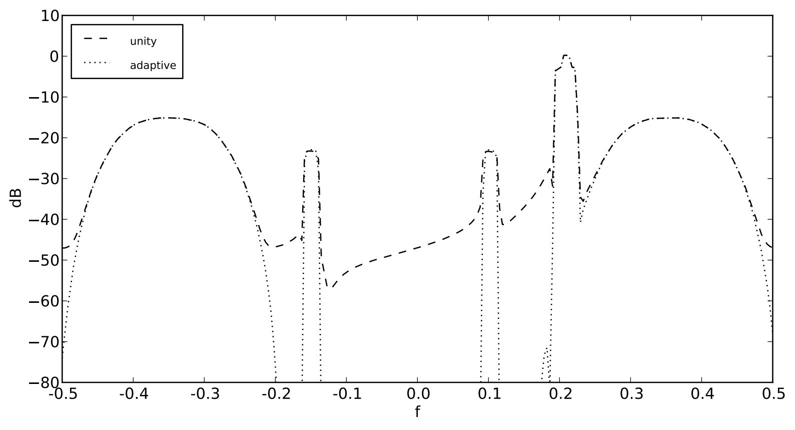
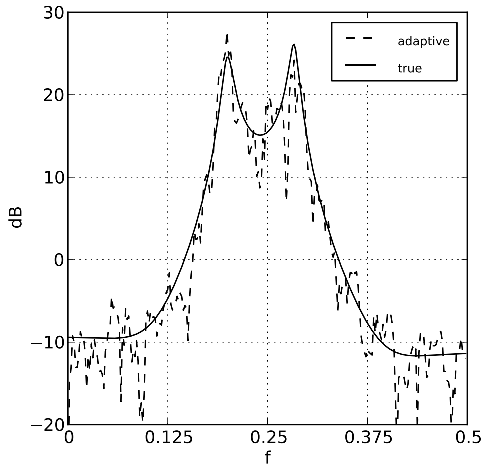
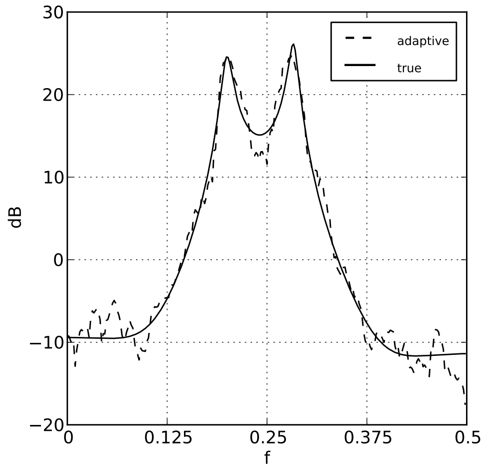


Figure 0.4.: Comparison of unity and adaptive weighting $N = 256, NW = 3, K = 5$

variance is demonstrated by Stoica's and Sundin's test case [6]. The two plots in Figure 0.5 show the true spectrum and the spectrum estimate produced by the multitaper method with adaptive weighting. The parameters are identical for both spectrum estimates, except for the numbers of tapers K used. In the first plot only two tapers are used to compute the spectrum estimate, while in the second plot the first seven eigenspectra make up the spectrum estimate. Clearly the spectrum estimate in the second plot is much smoother compared to the one in the first plot.



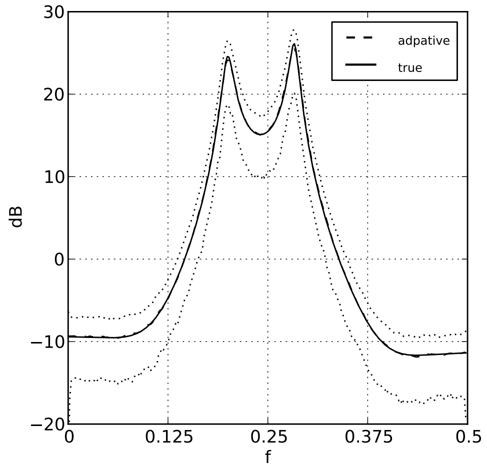
(a) $N = 512, NW = 4, K = 2$



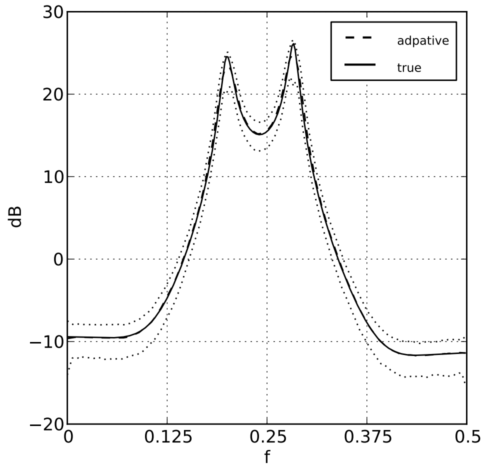
(b) $N = 512, NW = 4, K = 7$

Figure 0.5.: Spectrum estimates from Stoica's and Sundin's test case

In Figure 0.6 the two plots display the average of 1000 trials. The dotted lines represent the true spectrum plus/minus the standard derivation, which is less if more tapers are used. However, since the number of usable tapers is limited by $K \leq 2NW - 1$ we have to increase the sample size or the bandwidth if we wish to use more tapers, which results in an increase of computation time or a reduction of the resolution.



(a) $N = 512, NW = 4, K = 2$



(b) $N = 512, NW = 4, K = 7$

Figure 0.6.: Average of 1000 spectrum estimates and their standard derivation

The GNU Radio implementation of the multitaper spectrum estimator presented in this thesis is capable of computing DPSS of arbitrary length, can use a freely chosen number of tapers and can make use of the three weighting methods discussed earlier. A signal flow diagram of the multitaper spectrum estimator when using adaptive weighting can be seen in Figure 0.7

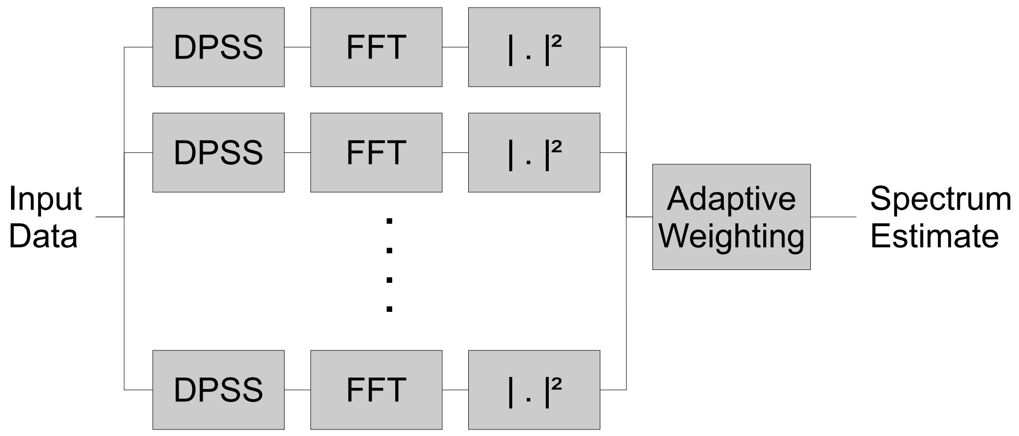


Figure 0.7.: Signal Flow Diagram of the multitaper spectrum estimator

The code written during this thesis is available from https://www.cgran.org/wiki/SpecEst. An example which uses Thomson's multitaper method is located in *apps/specest_noisy_sinusoid_mtm.py*.

Contents

1. A Short Introduction to Spectral Analysis

The goal of spectral analysis is to determine the energy distribution of a discretely sampled signal over all frequencies. This incomplete list shows some of the numerous applications of spectral analysis:

- analyzing an audio signal in order to compress it,
- finding hidden periodicities in a series of measurements,
- measuring the vibrations of a building, structure or vehicle,
- finding a free channel in a radio network.

There are essentially two approaches to spectral analysis: the parametric and the non-parametric approach. The parametric approach assumes that the received signal satisfies certain criteria which can lead to greater accuracy and a simpler implementation of the actual spectral estimator. However, if the assumed model for the data is false the parametric method may produce erroneous results.

If no assumptions can be made for the received signal, the non-parametric approach yields the best results. Probably the most intuitive description of the non-parametric approach is a narrow bandpass filter which sweeps over the spectrum of the input signal and records the output power.

The following chapters will deal with a non-parametric method called multitapering which was introduced by David J. Thomson in [2].

But first some very basic information about spectral analysis. Most of the time the signals we wish to analyze have infinite energy but finite average power, which makes it impossible to compute the signal's energy distribution. However, we may measure the signal's power spectral density. One of the definition of power spectral density (PSD) is the following ([1, p 7])

$$\boxed{\phi(f) = \lim_{N\to\infty} \mathrm{E}\left\{ \frac{1}{N} \left| \sum_{n=1}^{N} x(n) e^{-j2\pi f n} \right|^2 \right\}} \tag{1.1}$$

where $s(t)$ is the signal we wish to analyze and N is the length of the sample. Since we cannot sample an infinite amount of data, we can neither calculate the expected value nor the limit operation. This leads to the definition of the periodogram ([1, p 24]) as an estimate of the PSD

$$\boxed{\hat{\phi}(f) = \frac{1}{N}\left|\sum_{n=0}^{N-1} s(t)e^{-j2\pi fn}\right|^2}. \tag{1.2}$$

Unfortunately, the stochastic properties of the periodogram are rather poor ([1, p 39]). However, they can be greatly improved by applying a data window, also called data taper, to the input data. There exist many different windows: Bartlett, Hamming, Blackman, etc., which all have different advantages and drawbacks. The multitaper method relies on Discrete Prolate Spheroidal Sequences (DPSS) as data tapers to create a spectrum estimate. The next chapter will make it clear why the DPSS are so appealing to use for a spectrum estimate.

2. The Multitaper Method

2.1. The Motivation behind Multitapering

As the name multitapering implies, multiple spectrum estimates are used to estimate the true spectrum. A very simple implementation would be to take the average of all spectrum estimates

$$\hat{S}(f) = \frac{1}{K}\sum_{k=0}^{K-1} \hat{S}_k(f)\ . \tag{2.1}$$

Thomson refers to the spectrum estimates $\hat{S}_k(f)$ as eigenspectra, which is the name we will use from now on. The eigenspectra should closely resemble the true spectrum $S(f)$ and be uncorrelated to reduce the variance of $\hat{S}(f)$ (see A.1.1). The data taper u_k is applied to the input data in order to reach those properties. If the data taper is of unit length

$$\sum_{n=0}^{N-1} |u_k(n)|^2 = 1 \tag{2.2}$$

it can be expected to resemble $[1, 1, \dots, 1]^T/\sqrt{N}$ for large values of N. In this case the representation for the eigenspectra becomes

$$\boxed{\hat{S}_k(f) = \left|\sum_{n=0}^{N-1} u_k(n)x(n)e^{-j2\pi f n}\right|^2} \tag{2.3}$$

which has approximately the same energy as the spectrum estimate produced by the periodogram defined in (1.2) and is consistent with the definition Thomson uses in [2, 1055].

Let $U_k(f)$ denote the Discrete-Time Fourier Transform of the data tapers

$$U_k(f) = \sum_{n=0}^{N-1} u_k(n)e^{-j2\pi f n}\ , \tag{2.4}$$

then the expected value of the eigenspectra are given by (a proof can be found in the appendix (A.1.2))

$$\mathrm{E}\{\hat{S}_k(f)\} = \int_{-\frac{1}{2}}^{\frac{1}{2}} \left|U_k(f - f')\right|^2 S(f')df'\ . \tag{2.5}$$

which is the convolution of the spectral window $|U_k(f)|^2$ and the true spectrum $S(f)$. For the eigenspectra to be largely free of leakage, the spectral windows should have a high energy concentration, meaning almost all energy lies in a narrow frequency band $(-W, W)$ with $W < \frac{1}{2}$ where W is the number of cycles per sampling interval

$$\int_{-W}^{W} |U_k(f)|^2 \, df \approx \int_{-\frac{1}{2}}^{\frac{1}{2}} |U_k(f)|^2 \, df \, . \tag{2.6}$$

If we want the eigenspectra to be uncorrelated the data tapers must be orthogonal

$$\sum_{n=0}^{N-1} u_k(n) u_p(n) = 0 \quad \text{for } k \neq p \, . \tag{2.7}$$

2.2. Finding a Data-Taper

The following derivation is identical with the one from Stoica and Moses in [1]. We are trying to find a data taper with the following properties:

- Passing the frequencies between $-W$ and W as undistorted as possible while more or less suppressing all frequencies outside of $-W$ and W.
- The eigenspectra should be uncorrelated.

$U_k(f)$ can also be written as a multiplication of two vectors

$$\begin{aligned} U_k(f) &= \sum_{n=0}^{N-1} u_k(n) e^{-j2\pi f n} \\ &= \mathbf{u_k^* a} \, , \end{aligned} \tag{2.8}$$

with $\mathbf{u_k} = \left[u_k^*(0), u_k^*(1), \ldots, u_k^*(N-1)\right]^T$ and $\mathbf{a} = \left[1, e^{-i\omega}, e^{-2i\omega}, \ldots, e^{i(N-1)\omega}\right]^T$.

The total output power of the system if the input is white noise of unit variance is given by

$$\begin{aligned} \int_{-\frac{1}{2}}^{\frac{1}{2}} |U_k(f)|^2 df &= \int_{-\frac{1}{2}}^{\frac{1}{2}} \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} u_k(n) u_k^*(m) e^{-j2\pi f n} e^{j2\pi f m} df \\ &= \int_{-\frac{1}{2}}^{\frac{1}{2}} \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} u_k(n) u_k^*(m) e^{j2\pi f(m-n)} df \\ &= \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} u_k(n) u_k^*(m) \left[\int_{-\frac{1}{2}}^{\frac{1}{2}} e^{j2\pi f(m-n)} df \right] \\ &= \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} u_k(n) u_k^*(m) \delta_{n,m} \\ &= \mathbf{u_k^* u_k} \, . \end{aligned} \tag{2.9}$$

For the output power between $-W$ and W we get

$$\begin{aligned}\int_{-W}^{W} |U_k(f)|^2 df &= \sum_{n=0}^{N-1}\sum_{m=0}^{N-1} u_k(n)u_k^*(m)\left[\int_{-W}^{W} e^{j2\pi f(m-n)}df\right] \\ &= \mathbf{u_k^*}\left\{\int_{-W}^{W} e^{j2\pi f(m-n)}df\right\}\mathbf{u_k} \\ &= \mathbf{u_k^* A u_k}\ ,\end{aligned} \tag{2.10}$$

with $\mathbf{A}$ being a $N \times N$ matrix whose elements are

$$\mathbf{A_{n,m}} = \int_{-W}^{W} e^{-i2\pi f(m-n)}df = \boxed{\frac{\sin(2\pi W(m-n))}{\pi(m-n)}}\ , \tag{2.11}$$

which leads to the ratio of the output power between $-W$ and W and the total output power

$$\lambda_k = \frac{\int_{-W}^{W}|U_k(f)|^2 df}{\int_{-\frac{1}{2}}^{\frac{1}{2}}|U_k(f)|^2 df} = \frac{\mathbf{u_k^* A u_k}}{\mathbf{u_k^* u_k}} \tag{2.12}$$

Naturally we want λ_k to be as close to one as possible. If u_k is of unit length we can rewrite (2.12)

$$\boxed{\max_{\mathbf{u_k}} \mathbf{u_k^* A u_k} \quad \text{subject to } \mathbf{u_k^* u_k} = 1}\ . \tag{2.13}$$

It can be shown that every eigenvector of $\mathbf{A}$ fulfills this requirement. A proof can be found in the appendix (A.1.4). These eigenvectors are called Discrete Prolate Spheroidal Sequences (DPSS) with λ_k being their corresponding eigenvalue. Since $\mathbf{u_k}$ is an eigenvector of $\mathbf{A}$ it follows that $\mathbf{u_k}^T \cdot \mathbf{u_j} = 0$ for $k \neq j$ and the eigenspectra are uncorrelated. However, this is only the case if the true spectrum $S(f)$ is slowly varying in the interval $(-W, W)$ since only then the orthogonal properties of the DPSS hold.

2.3. On Discrete Prolate Spheroidal Sequences

This section provides a short overview about some of the useful properties of DPSS. For further information the reader is referred to Slepian's original paper [3].

We have already learned that the DPSS are the normalized eigenvectors of the matrix $\mathbf{A}$ the elements of which are given by

$$\boxed{\mathbf{A_{n,m}} = \frac{\sin(2\pi W(m-n))}{\pi(m-n)}}\ . \tag{2.14}$$

The DPSS are a function of N and W. For convenience we change the notation from $u_k(n, N, W)$ to $u_k(n)$. The eigenvalue of a DPSS represents the energy concentration. Every eigenvalue is real, since $\mathbf{A}$ is symmetric and lies in the interval from 0 to 1 (compare 2.12)

$$\lambda_k \in (0, 1) . \tag{2.15}$$

We order the DPSS by their eigenvalues

$$1 > \lambda_0 > \lambda_1 > \cdots > \lambda_{N-1} > 0 \tag{2.16}$$

so u_0 is the DPSS with the highest energy concentration.

2.3.1. Properties of Discrete Prolate Spheroidal Sequences

Orthogonality

The DPSS are orthogonal [3, p 1378]

$$\sum_{n=0}^{N-1} u_j(n) u_k(n) = \delta_{j,k} . \tag{2.17}$$

Relationship to Discrete Prolate Spheroidal Wave Functions

The DFT of a DPSS is called Discrete Prolate Spheroidal Wave Function (DPSWF) $U_k(f, N, W)$. As with the DPSS we change the notation to $U_k(f)$

$$U_k(f) = \varepsilon_k \sum_{t=0}^{N-1} u_k(n) e^{j2\pi f [n-(N-1)/2]} \tag{2.18}$$

where

$$\varepsilon_k = \begin{cases} 1, & k \text{ even} \\ 0, & k \text{ odd.} \end{cases} \tag{2.19}$$

2.3.2. Properties of Discrete Prolate Spheroidal Wave Functions

Orthogonality

The DPSWF are doubly orthogonal [3, p 1374]

$$\begin{aligned} \int_{-W}^{W} U_j(f) U_k(f) df &= \lambda_k \delta_{j,k} , \\ \int_{-\frac{1}{2}}^{\frac{1}{2}} U_j(f) U_k(f) df &= \delta_{j,k} , \end{aligned} \tag{2.20}$$

with λ_k being the eigenvalue of the corresponding DPSS.

Relationship to Discrete Prolate Spheroidal Sequences

A DPSS is the Fourier Transform of its corresponding DPSWF [3, p 1380]

$$
\begin{aligned}
u_k(n) &= \frac{1}{\varepsilon_k} \int_{-\frac{1}{2}}^{\frac{1}{2}} U_k(f) e^{-j2\pi[n-(N-1)/2]f} \\
&= \frac{1}{\lambda_k \varepsilon_k} \int_{-W}^{W} U_k(f) e^{-j2\pi[n-(N-1)/2]f} \qquad (2.21) \\
\varepsilon_k &= \begin{cases} 1, & k \text{ even} \\ 0, & k \text{ odd.} \end{cases}
\end{aligned}
$$

Symmetry

$U_k(f)$ is odd or even if k is odd or even [3, p 1375]

$$U_k(f) = (-1)^k U_k(-f) \,. \qquad (2.22)$$

Eigenfunctions

The DPSWF are eigenfunctions of the Dirichlet kernel [3, p 1374]

$$\int_{-W}^{W} \frac{\sin\left(N\pi(f-f')\right)}{\sin\left(\pi(f-f')\right)} U_k(N,W;f')df' = \lambda_k(N,W) \cdot U_k(N,W;f) \,. \qquad (2.23)$$

This will be of importance later, when we deal with adaptive weighting of the eigenspectra.

2.3.3. About the Eigenvalues λ_k

As mentioned earlier the eigenvalues represent the energy concentration of the spectral windows

$$\lambda_k = \frac{\int_{-W}^{W} |U_k(f)|^2}{\int_{-\frac{1}{2}}^{\frac{1}{2}} |U_k(f)|^2} \,. \qquad (2.24)$$

The first $2NW - 1$ eigenvalues are close to 1 [2, p 1059] and their DPSS are therefore especially useful as data tapers. Figure 2.1 shows the eigenvalues of the DPSS for $N = 128$ and a time bandwidth product of $NW = 4$.

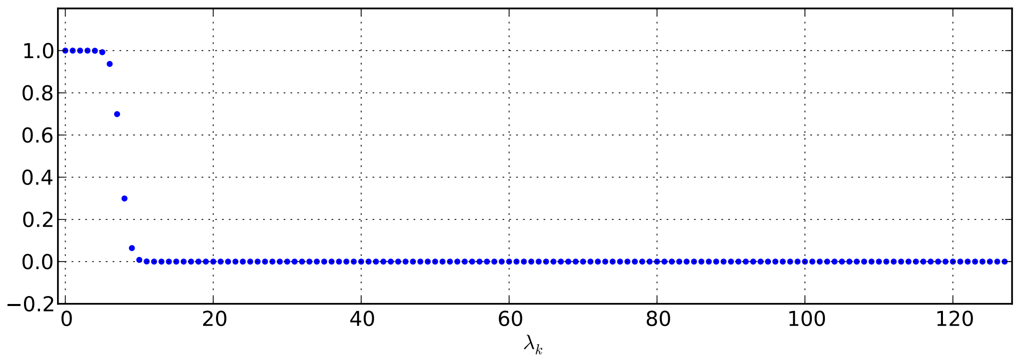


Figure 2.1.: Eigenvalues of the DPSS for $N = 128$ and $NW = 4$

In Figure 2.2 the first 20 eigenvalues can be seen in logarithmic scale.

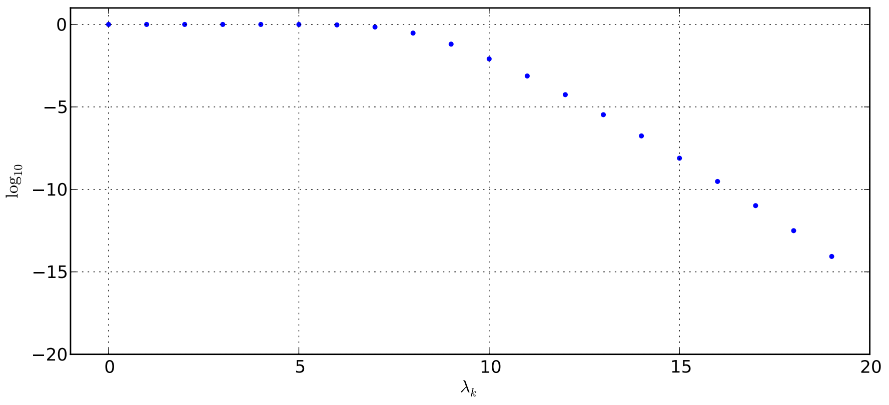
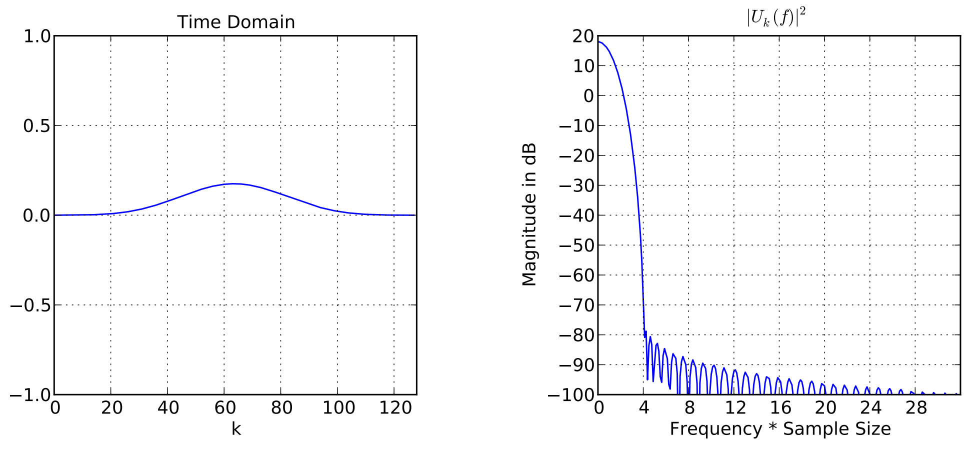


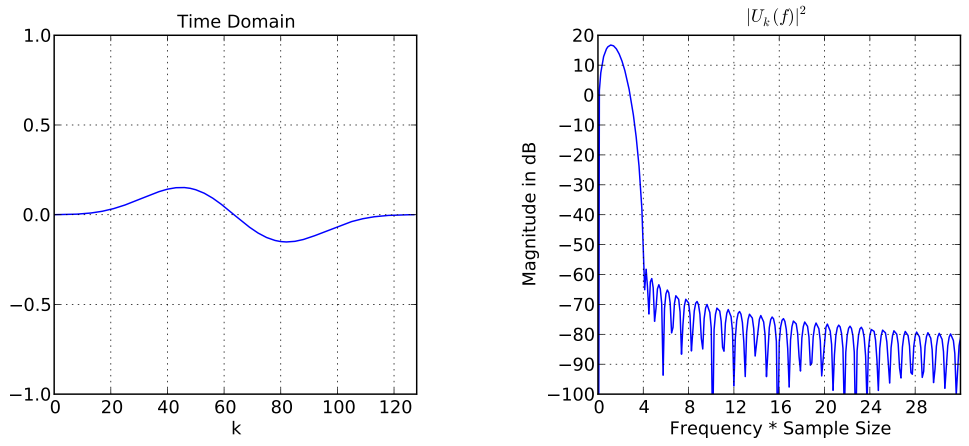
Figure 2.2.: The first 20 eigenvalues of the DPSS for $N = 128$ and $NW = 4$ in logarithmic scale

2.3.4. Some Discrete Prolate Spheroidal Sequences and their corresponding Spectral Windows

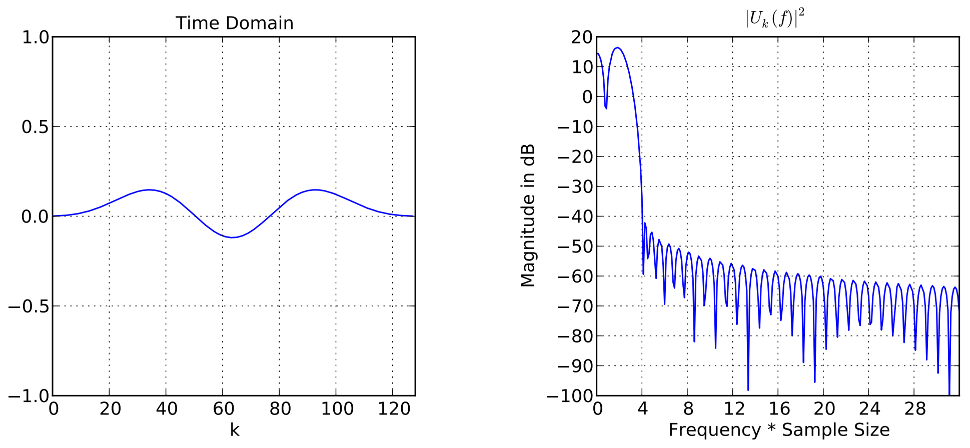
The Figures 2.3, 2.4 and 2.5 display the first 9 DPSS and their corresponding spectral windows for $N = 128$ and $NW = 4$. Note that the kth DPSS has k roots and that with increasing k the spectral windows $|U_k(f)|^2$ deteriorate. This can clearly be seen by the higher side lobe levels outside $NW = 4$. In table 2.1 the energy concentrations of the spectral windows can be found. The x-axes of the spectral window plots are in units of frequency times sample size.



(a) $k = 0, N = 128, NW = 4$

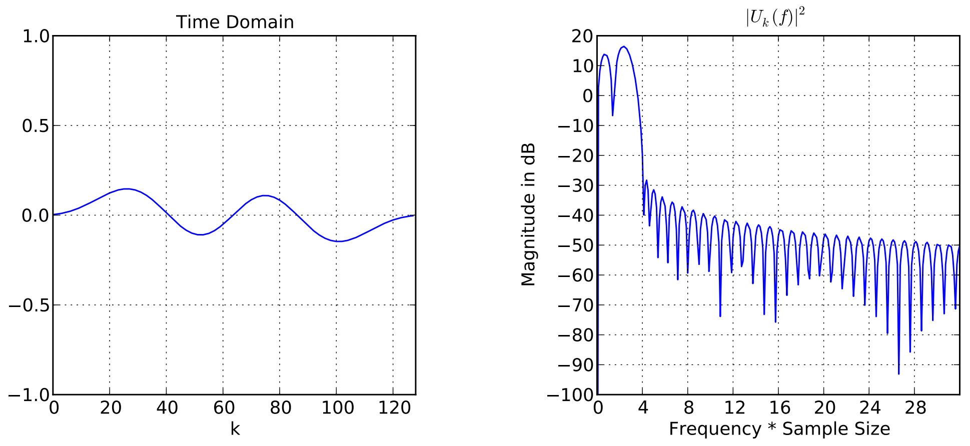


(b) $k = 1, N = 128, NW = 4$

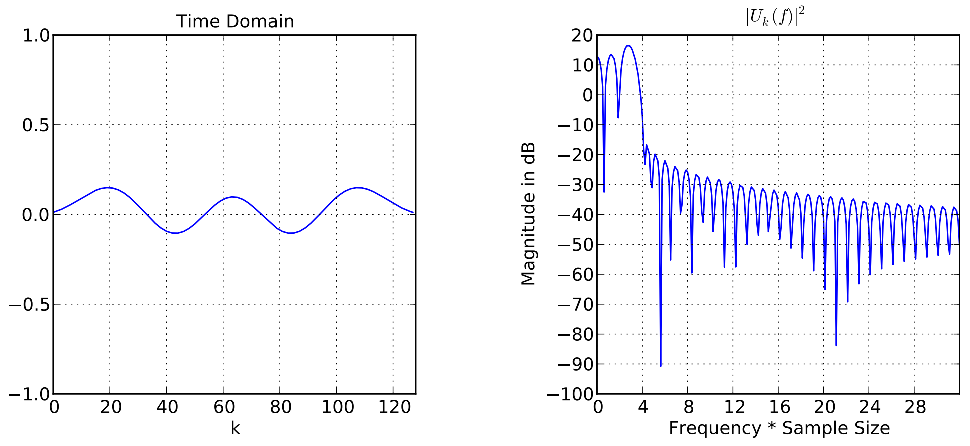


(c) $k = 2, N = 128, NW = 4$

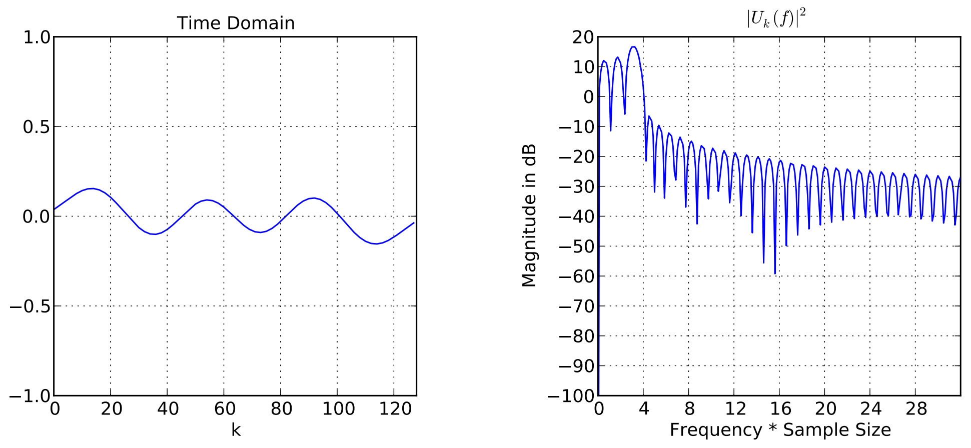
Figure 2.3.: DPSS and their spectral windows for $N = 128, NW = 4$ and $k = 0, 1, 2$



(a) $k = 3, N = 128, NW = 4$

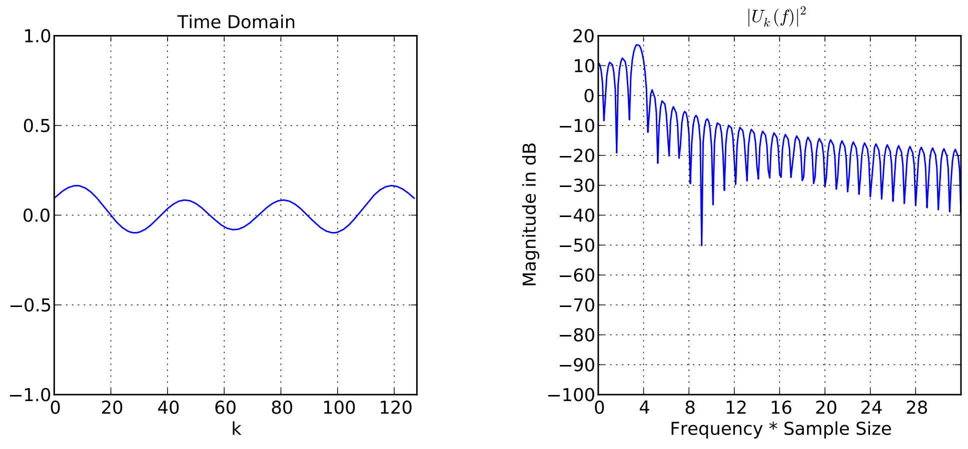


(b) $k = 4, N = 128, NW = 4$

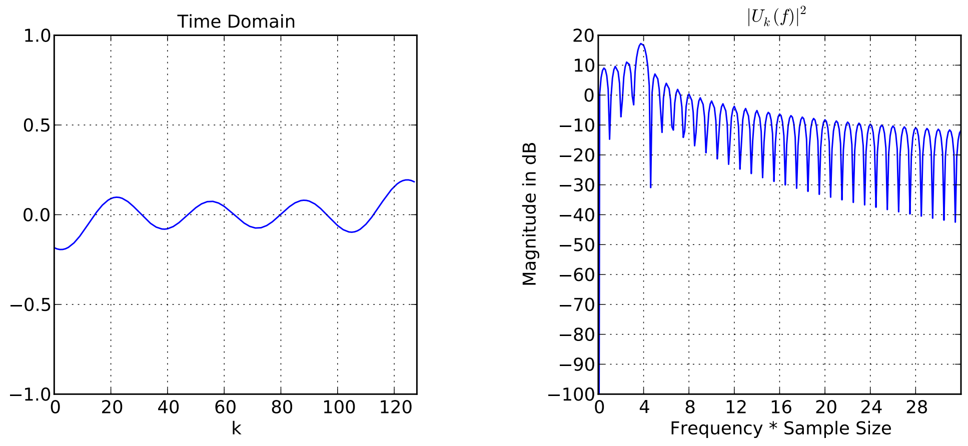


(c) $k = 5, N = 128, NW = 4$

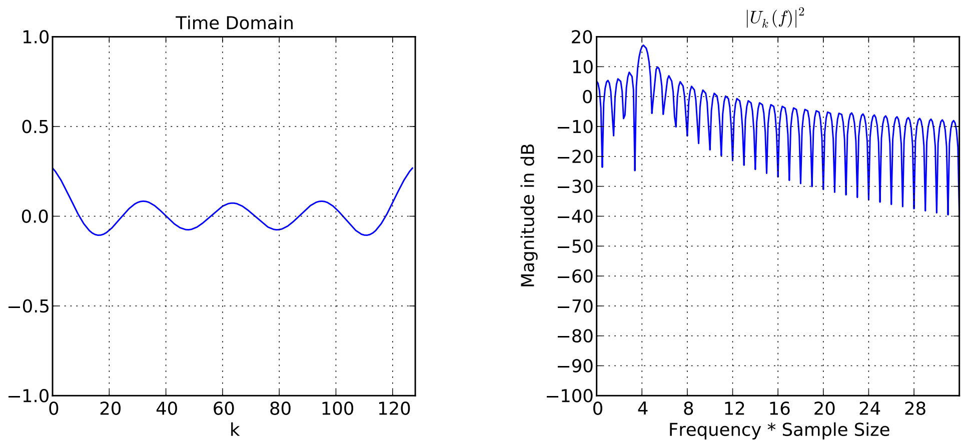
Figure 2.4.: DPSS and their spectral windows for $N = 128, NW = 4$ and $k = 3, 4, 5$



(a) $k = 6, N = 128, NW = 4$



(b) $k = 7, N = 128, NW = 4$



(c) $k = 8, N = 128, NW = 4$

Figure 2.5.: DPSS and their spectral windows for $N = 128, NW = 4$ and $k = 6, 7, 8$

$\|U_k(f)\|^2$	λ_k
0	0.99999999971599296
1	0.99999997311466315
2	0.99999881686676351
3	0.99996808906854018
4	0.9994167543397644
5	0.9925560207018389
6	0.93685566684291355
7	0.69904653266699568
8	0.29918687150623902

Table 2.1.: Energy concentration of the spectral windows in Figures 2.3, 2.4 and 2.5

2.3.5. Some Multitaper Spectral Windows

Figures 2.6 and 2.7 show the multitaper spectral windows which take the average of the first K spectral windows for $N = 128$ and $NW = 4$. As the number of tapers increases the multitaper spectral windows approach rectangular shape for frequencies below $NW = 4$, however, the energy ratio main lobe to side lobes decreases.

2.4. Weighting of the Eigenspectra

In section 2.3.3 we have learned that due to their varying energy concentrations not every spectral window is suitable as a spectral estimator and the quality of the eigenspectra produced by them deteriorates. The reduction of quality motivates weighting the eigenspectra. On the next pages the following weighting methods will be explained:

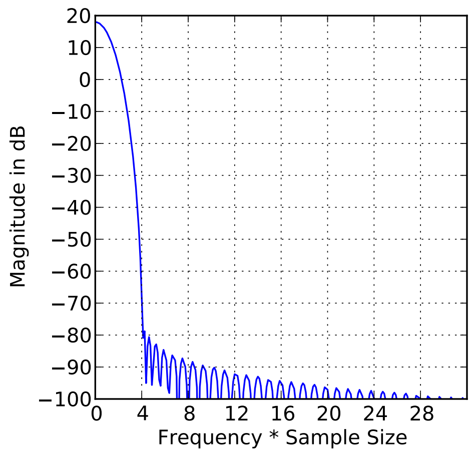
- Unity weighting
- Eigenvalue weighting
- Adaptive weighting

2.4.1. Unity Weighting

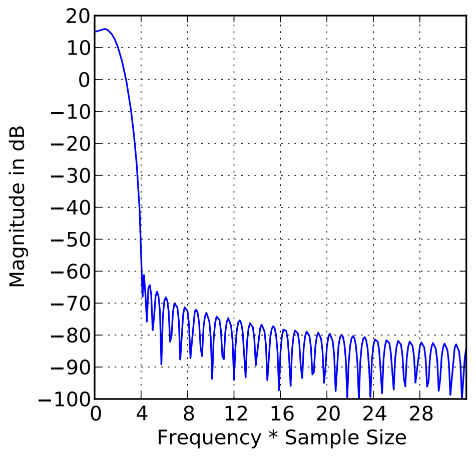
The average of all eigenspectra is computed regardless of their energy concentration

$$\boxed{\hat{S}(f) = \frac{1}{K}\sum_{k=0}^{K-1}\hat{S}_k(f)}\ . \tag{2.25}$$

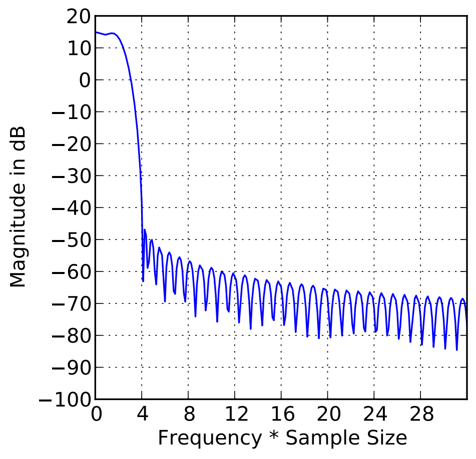
As long as only the first $2NW - 1$ eigenspectra are used, this weighting procedure gives good results since the eigenspectra have all roughly the same quality.



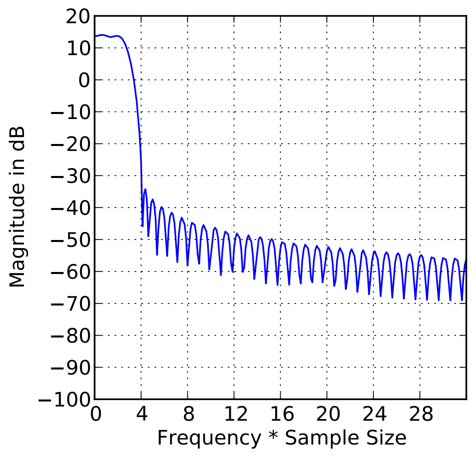
(a) $K = 1, N = 128, NW = 4$



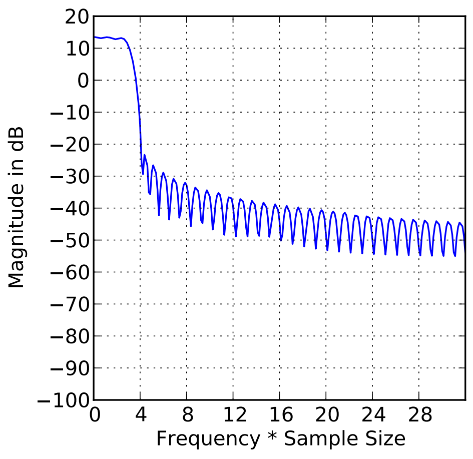
(b) $K = 2, N = 128, NW = 4$



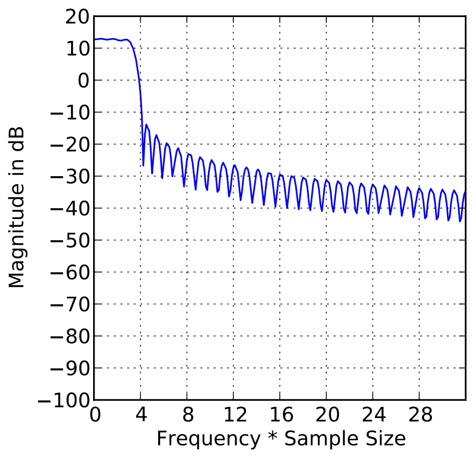
(c) $K = 3, N = 128, NW = 4$



(d) $K = 4, N = 128, NW = 4$

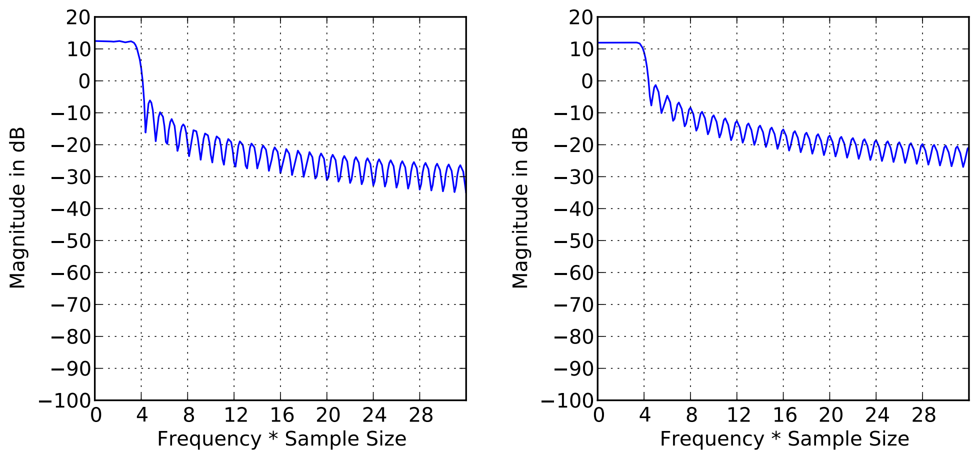


(e) $K = 5, N = 128, NW = 4$



(f) $K = 6, N = 128, NW = 4$

Figure 2.6.: Multitaper spectral window of the first K spectral windows



(a) $K = 7, N = 128, NW = 4$ (b) $K = 8, N = 128, NW = 4$

Figure 2.7.: Multitaper spectral window of the first K spectral windows

2.4.2. Eigenvalue Weighting

The eigenspectra are weighted according to their energy concentration λ_k

$$\boxed{\hat{S}(f) = \frac{\sum\limits_{k=0}^{K-1} \lambda_k \hat{S}_k(f)}{\sum\limits_{k=0}^{K-1} \lambda_k}}. \tag{2.26}$$

This weighting procedure is very similar to unity weighting and the advantages are marginal if only the first $2NW - 1$ eigenspectra are used. Higher order eigenspectra have almost no effect on the spectrum estimate since the eigenvalues rapidly converge to zero.

2.4.3. Adaptive Weighting

This weighting method requires some mathematical background. The derivation here is similar to the one in [4, p 361] and [2]. We will start with the Cramér spectral representation for stationary random processes.

Cramér Spectral Representation

Every stationary random process $x(t)$ has a Cramér spectral representation

$$\boxed{x(t) = \int_{-\frac{1}{2}}^{\frac{1}{2}} e^{j2\pi f t} dZ(f)}. \tag{2.27}$$

The increments of $Z(f)$ have some interesting properties

$$\boxed{\mathrm{E}\left\{dZ(f)\right\} = 0} \tag{2.28}$$

and

$$\boxed{\mathrm{E}\left\{|dZ(f)|^2\right\} = S(f)df}\,, \tag{2.29}$$

so the problem is the estimation of $dZ(f)$. Changing the definition of $dZ(f)$ by a phase factor yields

$$x(t) = \int_{-\frac{1}{2}}^{\frac{1}{2}} e^{j2\pi\nu[t-(N-1)/2]} dZ(\nu)\ . \tag{2.30}$$

The Fourier transform in time-centered form of the input data is given by

$$y(f) = \sum_{t=0}^{N-1} e^{-j2\pi f[t-(N-1)/2]} x(t)\ . \tag{2.31}$$

Conversely we can obtain the input data from its Fourier transform

$$x(t) = \int_{-\frac{1}{2}}^{\frac{1}{2}} e^{j2\pi f[t-(N-1)/2]} y(f)df\ . \tag{2.32}$$

Substituting (2.27) in (2.31) gives

$$y(f) = \int_{-\frac{1}{2}}^{\frac{1}{2}} \sum_{t=0}^{N-1} e^{j2\pi(\nu-f)[t-(N-1)/2]} dZ(\nu)\ . \tag{2.33}$$

We recognize the Dirichlet kernel

$$\sum_{t=0}^{N-1} e^{j2\pi(\nu-f)[t-(N-1)/2]} = \frac{\sin\left(N\pi(f-\nu)\right)}{\sin\left(\pi(f-\nu)\right)} \tag{2.34}$$

which leads to

$$\boxed{y(f) = \int_{-\frac{1}{2}}^{\frac{1}{2}} \frac{\sin\left(N\pi(f-\nu)\right)}{\sin\left(\pi(f-\nu)\right)} dZ(\nu)}\ . \tag{2.35}$$

Solving the integral in equation 2.35

We are trying to find an approximate solution $\hat{z}$ for $dZ(f)$. If we regard (2.35) as a linear Fredholm integral equation of the first kind $\hat{z}$ is given by ([2, p 1059])

$$\hat{z} = \sum_k \frac{1}{\lambda_k} y_k \psi_k \tag{2.36}$$

where ψ_k are a set of orthogonal eigenfunctions and y_k are the expansion coefficients given by

$$y_k = \int_{-\frac{1}{2}}^{\frac{1}{2}} y(f)\psi_k(f)df \,. \tag{2.37}$$

The DPSWF are eigenfunctions of the Dirichlet kernel

$$\int_{-W}^{W} \frac{\sin\left(N\pi(f-f')\right)}{\sin\left(\pi(f-f')\right)} U_k(N,W;f')df' = \lambda_k(N,W)\cdot U_k(N,W;f) \,. \tag{2.38}$$

The notation is changed as discussed earlier (2.3.1)

$$\begin{aligned} U_k(f) &= U_k(N,W,f) \\ \lambda_k &= \lambda_k(N,W) \,. \end{aligned} \tag{2.39}$$

We define the eigencoefficients $y_k(f)$ which are a projection of the phase-shifted Fourier Transform $y(f+\nu)$ on the interval $[-W,W]$

$$y_k(f) = \frac{1}{\lambda_k}\int_{-W}^{W} U_k(\nu)y(f+\nu)d\nu \,. \tag{2.40}$$

Theoretically we should divide by $\sqrt{\lambda_k}$ (see 2.20) but dividing by λ_k leads to prettier results. We will get rid of this error later.

The eigencoefficients can also be expressed as a convolution of $U_k(f)$ and $dZ(f)$

$$\begin{aligned} y_k(f) &= \frac{1}{\lambda_k}\int_{-W}^{W} U_k(\nu)y(f+\nu)d\nu \\ &= \frac{1}{\lambda_k}\int_{-W}^{W} U_k(\nu)\int_{\frac{1}{2}}^{\frac{1}{2}} \frac{\sin\left(N\pi(f+\nu-\xi)\right)}{\sin\left(\pi(f+\nu-\xi)\right)}dZ(\xi)d\nu \\ &= \frac{1}{\lambda_k}\int_{-W}^{W} U_k(\nu)\int_{\frac{1}{2}}^{\frac{1}{2}} \frac{\sin\left(N\pi(\xi-f-\nu)\right)}{\sin\left(\pi(\xi-f-\nu)\right)}dZ(\xi)d\nu \\ &= \frac{1}{\lambda_k}\int_{-\frac{1}{2}}^{\frac{1}{2}}\int_{-W}^{W} U_k(\nu)\frac{\sin\left(N\pi(\xi-f-\nu)\right)}{\sin\left(\pi(\xi-f-\nu)\right)}d\nu dZ(\xi) \,. \end{aligned} \tag{2.41}$$

Using the orthogonality properties of the DPSWF gives

$$y_k(f) = \frac{1}{\lambda_k}\int_{\frac{1}{2}}^{\frac{1}{2}} \lambda_k U_k(\xi-f)dZ(\xi) \tag{2.42}$$

and finally substituting $\nu = \xi - f$ leads to

$$\boxed{y_k(f) = \int\limits_{\frac{1}{2}}^{\frac{1}{2}} U_k(\nu) dZ(f+\nu)} \,. \tag{2.43}$$

The expansion coefficients of $dZ(f)$ are given by

$$\boxed{Z_k(f) = \frac{1}{\sqrt{\lambda_k}} \int\limits_{-W}^{W} U_k(\nu) dZ(f+\nu)} \,. \tag{2.44}$$

Comparing (2.44) to (2.43) we recognize that the only differences are the integration bounds and a factor. An alternative representation of the eigencoefficients can be derived by using (2.40) which is a convolution of $U_k(f)$ and $y(f)$

$$y_k(f) = \frac{1}{\lambda_k} \int\limits_{-W}^{W} U_k(\nu) y(f+\nu) d\nu \,. \tag{2.45}$$

Making use of the convolution theorem (A.1.3) and (2.21) this is equal to the Fourier transform of the input data multiplied by a DPSS.

$$\boxed{y_k(f) = \frac{1}{\varepsilon_k} \sum_{n=0}^{N-1} u_k(n) x(n) e^{-i2\pi f[n-(N-1)/2]}} \tag{2.46}$$

The eigencoefficients can be computed easily by multiplying the input data with a DPSS and perform a FFT on the result. Now that we have the eigencoefficients $y_k(f)$ we may compute the spectrum estimate. To estimate $dZ(f)$ we will use the first $K = 2NW$ eigencoefficients due to the high energy concentration of their spectral windows. Recognizing that the eigencoefficients are $\frac{1}{\sqrt{\lambda_k}}$ too small, as mentioned earlier (2.40), (2.36) becomes

$$d\hat{Z}(f; f_0) = \sum_{k=0}^{K-1} \frac{1}{\sqrt{\lambda_k}} y_k(f_0) U_k(f - f_0) \,. \tag{2.47}$$

We may then compute, what Thomson called the *high-resolution spectrum estimate*

$$\hat{S}_h(f; f_0) = \frac{1}{N} \left| \sum_{k=0}^{K-1} \frac{1}{\sqrt{\lambda_k}} y_k(f_0) U_k(f - f_0) \right|^2 \,. \tag{2.48}$$

Taking the average over the interval $(f_0 - W, f_0 + W)$

$$\hat{S}(f_0) = \frac{1}{2W} \int_{f_0 - W}^{f_0 + W} \hat{S}_h(f; f_0) df \tag{2.49}$$

and using the orthogonal properties of the DPSWF we arrive at

$$\begin{aligned} \hat{S}(f_0) &= \frac{1}{2NW} \sum_{k=0}^{K-1} |y_k(f_0)|^2 \\ &= \frac{1}{K} \sum_{k=0}^{K-1} |y_k(f_0)|^2 \ , \end{aligned} \tag{2.50}$$

which we recognize as being identical to the equation for unity weighting (2.25).

The $Z_k(f)$ and $y_k(f)$ have other interesting properties. We can rewrite (2.44), where we substitute $\nu = f' - f$ and use the symmetry properties (2.22) of the DPSWF

$$Z_k(f) = \frac{(-1)^k}{\sqrt{\lambda_k}} \int_{f-W}^{f+W} U_k(f - f')dZ(f') \ . \tag{2.51}$$

Using (2.29) leads to the expected values of the $|Z_k(f)|^2$.

$$\boxed{\mathrm{E}\left\{|Z_k(f)|^2\right\} = \frac{1}{\lambda_k} \int_{f-W}^{f+W} \left|U_k(f - f')\right|^2 S(f')df'} \tag{2.52}$$

We can also rearrange the equation for the eigencoefficients

$$y_k(f) = \int_{-\frac{1}{2}}^{\frac{1}{2}} U_k(\nu)dZ(f + \nu) \ . \tag{2.53}$$

The expected value of the $|y_k(f)|^2$ is given by

$$\boxed{\mathrm{E}\left\{|y_k(f)|^2\right\} = \int_{-\frac{1}{2}}^{\frac{1}{2}} \left|U_k(f - f')\right|^2 S(f')df'} \tag{2.54}$$

which is identical to (2.5). So Thomson's and the approach from Stoica and Moses of deriving the multitaper method lead to the same result. Comparing (2.52) to (2.54) we recognize that $Z_k(f)$ as well as $y_k(f)$ produce a smoothed version of the true spectrum. However the eigencoefficient spectrum estimates suffer from leakage while the $Z_K(f)$ estimate does not, due to their different integration bounds. Unfortunately we don't have the $Z_k(f)$ since $dZ(f)$ is unknown. Hence we have to use the estimates $y_k(f)$.

Reducing the error

We can minimize the error by utilizing the least square method, multiplying $y_k(f)$ by a factor $d_k(f)$

$$\begin{aligned} e_k(f) &= Z_k(f) - d_k(f) y_k(f) \\ &= \frac{1}{\sqrt{\lambda_k}} \int_{-W}^{W} U_k(\nu) dZ(f+\nu) - d_k(f) \int_{-\frac{1}{2}}^{\frac{1}{2}} U_k(\nu) dZ(f+\nu) \\ &= \left(\frac{1}{\sqrt{\lambda_k}} - d_k(f) \right) \int_{-W}^{W} U_k(\nu) dZ(f+\nu) - d_k(f) \fint U_k(\nu) dZ(f+\nu) \,, \end{aligned} \tag{2.55}$$

with the cut integral being defined as

$$\fint = \int_{-\frac{1}{2}}^{\frac{1}{2}} - \int_{-W}^{W} \,. \tag{2.56}$$

The expected value of the squared error is

$$\begin{aligned} \mathrm{E}\left\{ e_k^2(f) \right\} = \left(\frac{1}{\sqrt{\lambda_k}} - d_k(f) \right)^2 \mathrm{E}\left\{ \left| \int_{-W}^{W} U_k(\nu) dZ(f+\nu) \right|^2 \right\} \\ + d_k^2(f)\, \mathrm{E}\left\{ \left| \fint U_k(\nu) dZ(f+\nu) \right|^2 \right\} \,, \end{aligned} \tag{2.57}$$

where we used that $\mathrm{E}\{|dZ(f)|\} = 0$.

We will examine both integrals separately. The first integral is given by

$$\mathrm{E}\left\{ \left| \int_{-W}^{W} U_k(\nu) dZ(f+\nu) \right|^2 \right\} = \int_{-W}^{W} \int_{-W}^{W} U_k(\nu) U_k^*(\tilde{\nu})\, \mathrm{E}\left\{ dZ^*(f+\nu) dZ(f+\tilde{\nu}) \right\} \,. \tag{2.58}$$

Using that the $U_k(f)$ and $dZ(f)$ are orthogonal and that $\mathrm{E}\left\{ |dZ(f)|^2 \right\} = S(f)df$ yields

$$\begin{aligned} &= \int_{-W}^{W} \left[U_k(\nu) \right]^2 \mathrm{E}\left\{ dZ^*(f+\nu) dZ(f+\nu) \right\} \\ &= \int_{-W}^{W} \left[U_k(\nu) \right]^2 S(f+\nu) d\nu \,. \end{aligned} \tag{2.59}$$

If we make the assumption that $S(f)$ is slowly varying we can treat it as a constant

$$\begin{aligned} &\approx S(f) \int_{-W}^{W} \left[U_k(\nu) \right]^2 d\nu \\ &= \lambda_k S(f) \,. \end{aligned} \tag{2.60}$$

For the second Integral we take the average over all frequencies

$$\begin{aligned}\int_{-\frac{1}{2}}^{\frac{1}{2}} \mathrm{E}\left\{\left|\not\int U_k(\nu)dZ(f+\nu)\right|^2\right\} df &= \int_{-\frac{1}{2}}^{\frac{1}{2}} \not\int \left[U_k(\nu)\right]^2 S(f+\nu)d\nu df \\ &= \not\int \int_{-\frac{1}{2}}^{\frac{1}{2}} \left[U_k(\nu)\right]^2 S(f+\nu)dfd\nu \\ &= \not\int \left[U_k(\nu)\right]^2 \int_{-\frac{1}{2}}^{\frac{1}{2}} S(f+\nu)dfd\nu\end{aligned} \tag{2.61}$$

with

$$\int_{-\frac{1}{2}}^{\frac{1}{2}} S(f)df = \sigma^2 \tag{2.62}$$

and

$$\not\int \left[U_k(f)\right]^2 df = \int_{-\frac{1}{2}}^{\frac{1}{2}} \left[U_k(f)\right]^2 df - \int_{-W}^{W} \left[U_k(f)\right]^2 df = 1-\lambda_k\,, \tag{2.63}$$

the second integral becomes

$$\int_{-\frac{1}{2}}^{\frac{1}{2}} \mathrm{E}\left\{\left|\not\int U_k(\nu)dZ(f+\nu)\right|^2\right\} df = (1-\lambda_k)\sigma^2\,, \tag{2.64}$$

which leads to the expected value of the squared error

$$\mathrm{E}\left\{e_k^2(f)\right\} = \left(\frac{1}{\sqrt{\lambda_k}} - d_k(f)\right)^2 \lambda_k S(f) + d_k^2(f)(1-\lambda_k)\sigma^2\,. \tag{2.65}$$

Since we are interested in the minimum of the squared error we differentiate with respect to $d_k(f)$ and set the result equal to zero

$$\frac{1}{2}\frac{\partial}{\partial d_k(f)} \mathrm{E}\left\{e_k^2(f)\right\} = -\left(\frac{1}{\sqrt{\lambda_k}} - d_k(f)\right)\lambda_k S(f) + d_k(f)(1-\lambda_k)\sigma^2 \overset{!}{=} 0 \tag{2.66}$$

Rearranging for $d_k(f)$ yields

$$\boxed{d_k(f) = \frac{\sqrt{\lambda_k}S(f)}{\lambda_k S(f) + (1-\lambda_k)\sigma^2}}\,. \tag{2.67}$$

We may now use $d_k(f)\cdot y_k(f)$ to estimate $Z_k(f)$

$$d_k^2(f)\left|y_k(f)\right|^2 \approx \left|Z_k(f)\right|^2\,, \tag{2.68}$$

which leads to the following equation for the multitaper spectrum estimator

$$\boxed{\hat{S}(f) = \frac{\sum\limits_{k=0}^{K-1} d_k^2(f)\left|y_k(f)\right|^2}{\sum\limits_{k=0}^{K-1} d_k^2(f)}}\,. \tag{2.69}$$

The $y_k(f)$ are given by (2.46)

$$y_k(f) = \frac{1}{\varepsilon_k} \sum_{n=0}^{N-1} x(n) u_k(n) e^{-i2\pi f[n-(N-1)/2]} \,. \tag{2.70}$$

This leads to the following equation for the eigenspectra

$$\begin{aligned} |y_k(f)|^2 &= \left| \frac{1}{\varepsilon_k} \sum_{n=0}^{N-1} x(n) u_k(n) e^{-j2\pi f[n-(N-1)/2]} \right|^2 \\ &= \left| \sum_{n=0}^{N-1} x(n) u_k(n) e^{-j2\pi f n} \right|^2 \,, \end{aligned} \tag{2.71}$$

which is identical to 2.3. Unfortunately, to calculate the $d_k(f)$ we need the true spectrum $S(f)$, which we obviously do not know. So we have to use the estimated spectrum $\hat{S}(f)$. For the initial spectrum estimate we take the average of $\hat{S}_0(f)$ and $\hat{S}_1(f)$,

$$\hat{S}_{init}(f) = \frac{1}{2} \left[S_0(f) + S_1(f) \right] \tag{2.72}$$

and then solve iteratively until we are satisfied with the spectrum estimate. Usually three iterations suffice to get a reasonable accurate spectrum estimate.

It is possible to obtain a criterion for stopping the iteration process. We start with the equations for $d_k(f)$ and $\hat{S}(f)$:

$$\begin{aligned} d_k(f) &= \frac{\sqrt{\lambda_k} \hat{S}(f)}{\lambda_k \hat{S}(f) + (1-\lambda_k)\sigma^2} \\ \hat{S}(f) &= \sum_{k=0}^{K-1} \frac{d_k^2(f) \hat{S}_k(f)}{d_k^2(f)} \,. \end{aligned} \tag{2.73}$$

Moving everything on one side

$$\begin{aligned} \hat{S}(f) \sum_{k=0}^{K-1} d_k^2(f) &= \sum_{k=0}^{K-1} d_k^2(f) \hat{S}_k(f) \\ \sum_{k=0}^{K-1} d_k^2(f) \left[\hat{S}(f) - \hat{S}_k(f) \right] &= 0 \end{aligned} \tag{2.74}$$

and plugging in $d_k(f)$ results in

$$\boxed{\sum_{k=0}^{K-1} \frac{\lambda_k \left[\hat{S}(f) - \hat{S}_k(f) \right]}{\left[\lambda_k \hat{S}(f) + (1-\lambda_k)\sigma^2 \right]^2} = 0} \,. \tag{2.75}$$

If (2.75) is reasonably close to zero the loop can be terminated.

2.5. Comparison of different Multitaper Estimators

This section illustrates how a change of the following parameters affects the spectrum estimate produced by the multitaper method:

- the length of the DPSS,
- the time bandwidth product,
- the number of tapers,
- the weighting method.

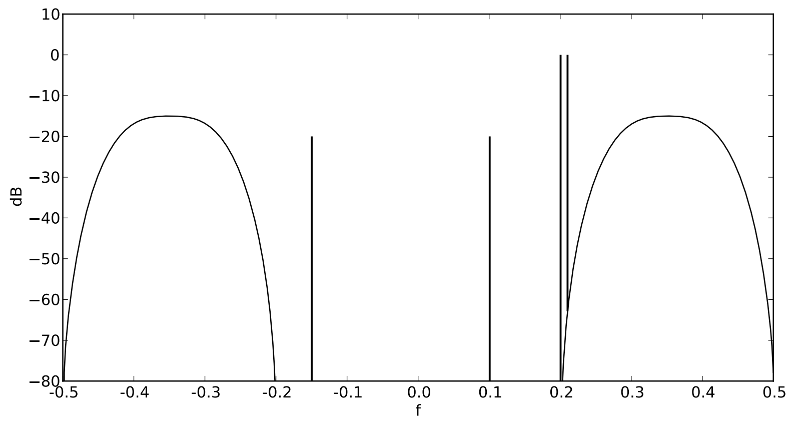
To judge the results test cases introduced in [5, p 17] and [6, p 178] are used. The first test case consists of data which is generated by four complex sinusoids and complex coloured noise. The coloured noise is produced by Gaussian white noise which passes a moving average filter. Two of the sinusoids have a frequency of 0.2 and 0.21 in order to check the resolution of the spectrum estimate. The other two sinusoids with a frequency of 0.1 and -0.15 have a lower amplitude enabling us to test the capability of the spectrum estimator to pick out weaker signals among stronger ones. Figure 2.8(a) shows the true analytically determined spectrum. In the appendix (A.3) a table can be found which includes 128 samples of this test case. This sample can be used to check any spectrum estimation routine.

The other test case, devised by Stoica and Sundin, uses an IIR-filter to generate a fast-varying spectrum with two distinguished peaks. The IIR-filters transfer function is given by

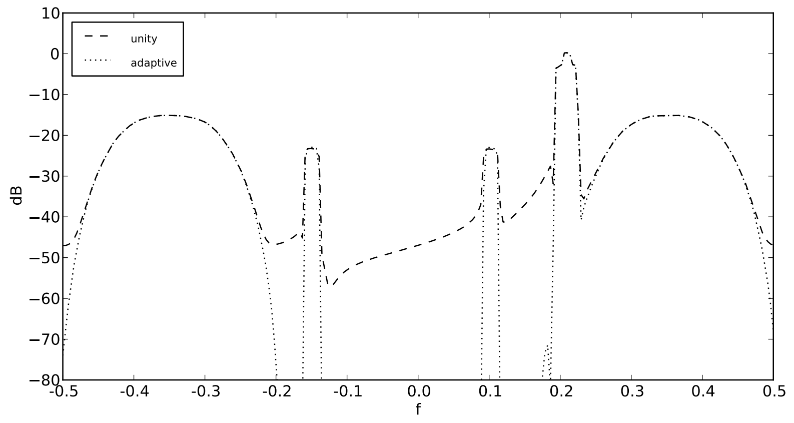
$$\boxed{H(z) = \frac{1 - 0.21z^{-2} + 0.25z^{-4}}{1 - 0.2z^{-1} + 1.61z^{-2} - 0.19z^{-3} + 0.8556z^{-4}}}\,. \tag{2.76}$$

We start with Marple's test case. Figure 2.8 shows the true spectrum and two spectrum plots for $N = 256$ and $N = 1024$ in both cases the time bandwidth product is 3, 5 tapers are used and the results of 1000 runs are averaged. It does not come as a surprise that the resolution is better if the lengths of the DPSS increase. With a sample size of only 256 the spectrum estimator is unable to distinguish between the two peaks at 0.20 and 0.21. These two plots also show that adaptive weighting is superior to unity weighting, or eigenvalue weighting for that matter. In low-power spectrum regions (e.g. $f \in (-0.15, 0.1)$) unity weighting proved to be incapable of estimating the true spectrum very closely.

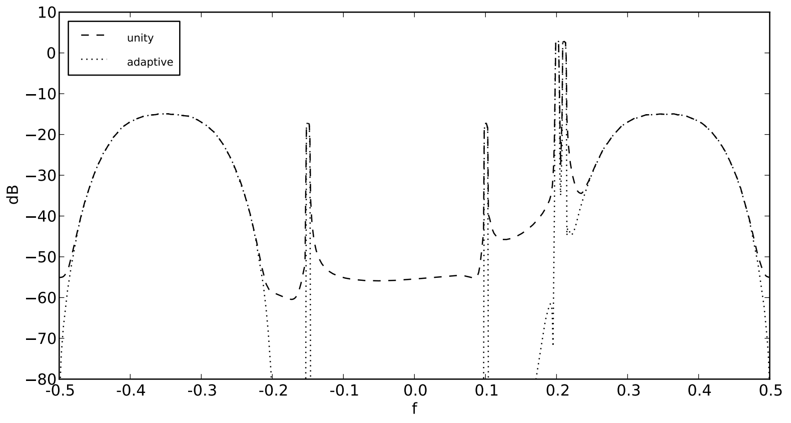
The second test case demonstrates the effect of the number of tapers. In Figure 2.9 different plots for $N = 512$ and $NW = 4$ can be seen. The first two plots show spectrum estimates of a single sample for $K = 2$ and $K = 7$. In both cases adaptive weighting is used. These two plots show the reduced variance in the second spectrum estimate due to the increased numbers of tapers. The last two plots have the same parameters as the aforementioned plots but the results of 1000 trials are averaged. The dashed line which is the spectrum estimate produced by the multitaper method is almost identical to the true spectrum. The



(a) The true spectrum



(b) $N = 256, NW = 3, K = 5$



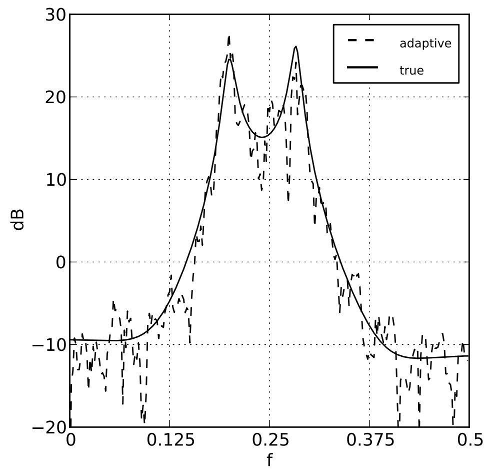
(c) $N = 1024, NW = 3, K = 5$

Figure 2.8.: Marple's test case

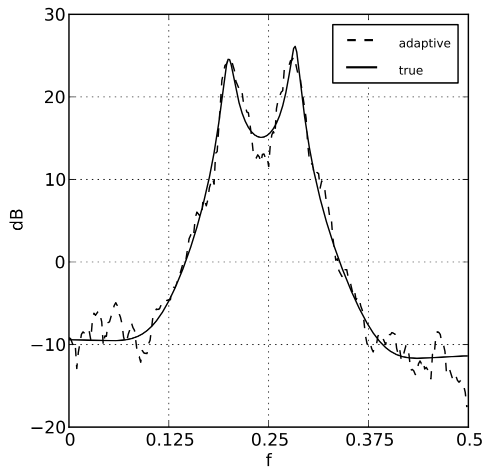
dotted lines indicate one standard deviation from the spectrum estimate which show the lower standard deviation in case more tapers are used. However, since the number of usable tapers K is limited by the time bandwidth product NW through

$$\boxed{K \leq 2NW - 1}, \tag{2.77}$$

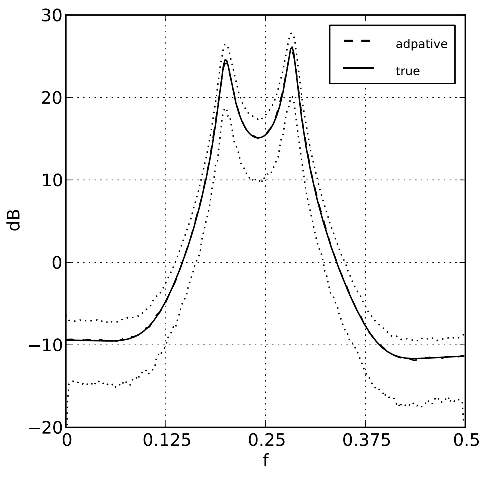
the time bandwidth product or the sample size have to increase if more tapers are to be used, which decreases the resolution of the spectrum estimate or is computationally more expensive. An additional comment on Stoica's and Sundin's test case: It was essentially



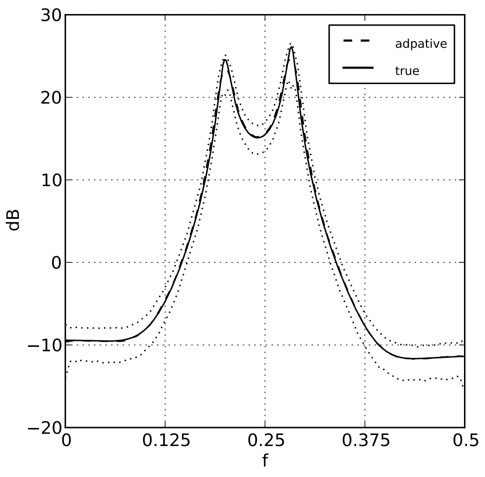
(a) $N = 512, NW = 4, K = 2$



(b) $N = 512, NW = 4, K = 7$



(c) $N = 512, NW = 4, K = 2$



(d) $N = 512, NW = 4, K = 7$

Figure 2.9.: Stoica's and Sundin's test case

unimportant which weighting method was used. So in some cases the additional computing costs for adaptive weighting, compared to unity weighting might be unnecessary.

3. Implementing the Multitaper Spectral Estimator in GNU Radio

3.1. Generating the Discrete Prolate Spheroidal Sequences

So far we know that we need the eigenvectors of the matrix $\mathbf{A}$ to design the spectrum estimator. Unfortunately, for large values of N the matrix $\mathbf{A}$ becomes ill-conditioned and we must therefore find another way to generate the DPSS. The algorithm used here was published in [7]. A Python implementation of this algorithm can be found in the appendix (A.2). This algorithm utilises the fact that if two matrices commute ($\mathbf{AB} = \mathbf{BA}$) they have the same eigenvectors (see A.1.5 for a proof). The elements of $\mathbf{A}$ are

$$\boxed{\mathbf{A_{n,m}} = \frac{\sin(2\pi W(m-n))}{\pi(m-n)}}\,. \tag{3.1}$$

In [8] proof is given that a matrix which is defined by

$$\mathbf{T_{n,m}} = \begin{cases} \frac{1}{2}n(N-n) & \text{for } m = n-1 \\ \left(\frac{N-1}{2}-n\right)^2\cos(2\pi W) & \text{for } m = n \\ \frac{1}{2}(n+1)(N-n-m) & \text{for } m = n+1 \\ 0 & \text{for } |m-n| > 1 \end{cases} \tag{3.2}$$

commutes to $\mathbf{A}$. The eigenvalues are computed using the method of bisection (see [9, p 437]). We will use the following naming convention for the elements of $\mathbf{T}$

$$\mathbf{T} = \begin{bmatrix} a_1 & b_1 & 0 & \dots & 0 \\ b_1 & a_2 & b_2 & \ddots & \vdots \\ 0 & b_2 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & b_{n-1} \\ 0 & \dots & 0 & b_{n-1} & a_n \end{bmatrix} \tag{3.3}$$

The characteristic polynomials $p_r(x) = \det(\mathbf{T_r} - x\mathbf{I})$ can be used to find the eigenvalues, where $\mathbf{T_r}$ is defined as the first r rows and columns of the matrix $\mathbf{T}$.

For example:

$$\mathbf{T_0} = 0, \mathbf{T_1} = [a_1], \mathbf{T_2} = \begin{bmatrix} a_1 & b_1 \\ b_1 & a_2 \end{bmatrix}, \mathbf{T_3} = \begin{bmatrix} a_1 & b_1 & 0 \\ b_1 & a_2 & b_2 \\ 0 & b_2 & a_3 \end{bmatrix}, \dots \tag{3.4}$$

There exists a recursive relationship between the characteristic polynomials

$$p_r(x) = (a_n - x)p_{r-1}(x) - b_{r-1}^2 p_{r-2}(x) \tag{3.5}$$

Using $p_0(x) = 1$ and $p_1(x) = a_1 - x$ as the seed values we are able to easily compute all characteristic polynomials. The eigenvalues are the roots of $p_n(x)$. To obtain a certain eigenvalue we make use of the Sturm sequence property ([9, p 438]) which states the following:

The number of sign changes $\alpha(\lambda)$ in the sequence

$$\{p_0(\lambda), p_1(\lambda), \ldots, p_n(\lambda)\} \tag{3.6}$$

equals the number of eigenvalues which are less than λ. The kth largest eigenvalue may then be computed through bisection,

```
while $|\lambda_{max} - \lambda_{min}| \geq \varepsilon$ do
  $\lambda_k = (\lambda_{max} + \lambda_{min})/2$
  if $\alpha(k) \geq k$ then
    $\lambda_{max} = \lambda_k$
  else
    $\lambda_{min} = \lambda_k$
  end if
end while
```

where ε is the accuracy with which the eigenvalue is to be computed. The interval that contains all eigenvalues of $\mathbf{T}$ is given by the Gershgorin circle theorem [7, p 3277] and [9, p 341].

$$\begin{aligned} \lambda_{max,init} &= \frac{(N-1)\cdot(N+2) + N\cdot(N+1)}{8} + 0.25 \\ \lambda_{min,init} &= -\frac{(N-1)\cdot(N+2) + N\cdot(N+1)}{8} - 0.25 \end{aligned} \tag{3.7}$$

The only problem encountered during the implementation of this algorithm was the recursive relationship between the characteristic polynomials. For a certain r the numbers of $p_r(x)$ grew so large that python was no longer able to handle them. But since we are only interested in the sign changes and not the value of $p_r(x)$ this was solved by dividing $p_r(x)$ and $p_{r-1}(x)$ by the absolute value of $p_{r-1}(x)$ as soon as $p_r(x)$ got too large. This operation will only change the value of $p_{r+1}(x)$ but will leave its sign as it is.

```
if $p_r(x) \geq MAX$ then
  $p_r(x) = \frac{p_r(x)}{|p_{r-1}(x)|}$
  $p_{r-1}(x) = \frac{p_{r-1}(x)}{|p_{r-1}(x)|}$
end if
```

Once the eigenvalue has been computed, inverse iteration is used to calculate the corresponding DPSS [10, p 356]. The principle of inverse iteration is rather simple. For a system of linear equations given by

$$\boxed{(\mathbf{T} - \lambda\mathbf{I})\mathbf{u_{new}} = \mathbf{u_{old}}}\,, \tag{3.8}$$

where the initial vector $\mathbf{u_0}$ is a random vector and λ an eigenvalue of $\mathbf{T}$. The solution $\mathbf{u_1}$ resembles the true eigenvector which belongs to λ. If this step is repeated the accuracy of the eigenvector estimate increases. The loop can be stopped after the estimates no longer change drastically or after a fixed number of iterations. The tridiagonal matrix algorithm [11, p 153] was used to solve (3.8). The following pseudo code outlines the whole procedure

```
$\mathbf{u_{new}} = [1, 1, \ldots, 1]^T$ {our initial guess}
$\mathbf{u_{old}} = [0, 0, \ldots, 0]^T$
while $|\mathbf{u_{new}} - \mathbf{u_{old}}| \geq \Delta$ do
    $\mathbf{u_{old}} = \mathbf{u_{new}}$
    Solve $(T - \lambda I)\mathbf{u_{new}} = \mathbf{u_{old}}$
    Normalize $\mathbf{u_{new}}$
end while
```

Here Δ sets the accuracy with which we wish to compute the DPSS. Of course it is possible to define other conditions to terminate the loop. It proved to be sufficient to compare just the first elements of $\mathbf{u_{new}}$ and $\mathbf{u_{old}}$.

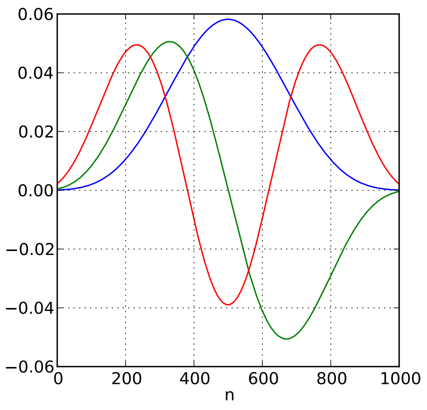
What we have so far is the eigenvalue of the matrix $\mathbf{T}$ and the corresponding DPSS. To compute the eigenvalue of the matrix $\mathbf{A}$ which belongs to the DPSS $\mathbf{u}$ the following equation can be solved easily

$$\begin{aligned} \mathbf{Au} &= \lambda\mathbf{u} \\ \frac{\mathbf{Au}}{\mathbf{u}} &= \lambda\,. \end{aligned} \tag{3.9}$$

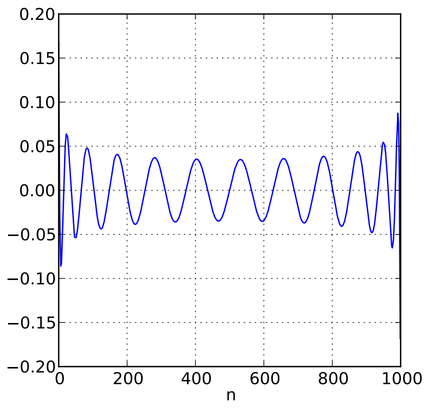
Since we are dealing with $N \times N$ matrices we get N unique solutions due to the fact that the estimated DPSS we use is not 100 percent accurate. It is possible to obtain a solution through the method of least squares, however the division in (3.9) may give erroneous results if an element of the DPSS is extremely close to zero. By picking just the equation where $\mathbf{u}$ has its largest element this problem is overcome. Additionally this is the numerical most stable solution. If k is the index where $\mathbf{u}$ has its largest element λ is given by

$$\boxed{\lambda = \frac{\sum_{n=0}^{N-1} A_k(n)u(n)}{u(k)}}\,. \tag{3.10}$$

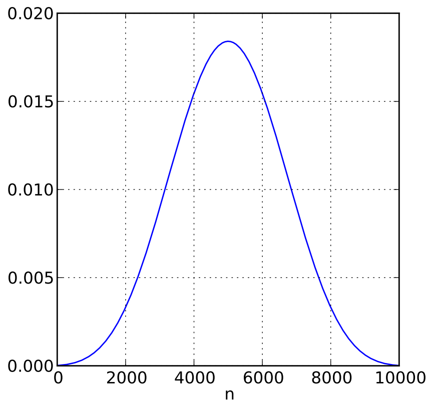
In Figure 3.1 some plots of DPSS can be seen which show the versatility of this algorithm.



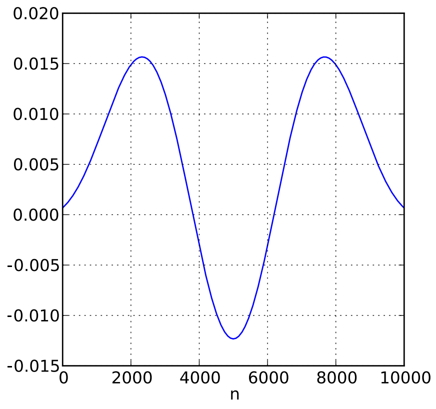
(a) $k = 0, 1, 2, N = 1000, NW = 3$



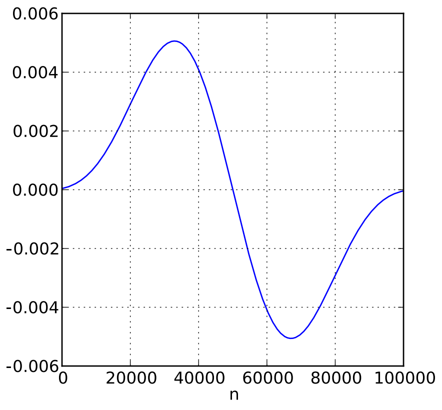
(b) $k = 23, N = 1000, NW = 3$



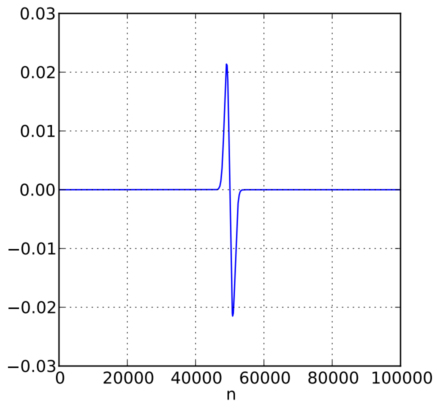
(c) $k = 0, N = 10000, NW = 3$



(d) $k = 2, N = 10000, NW = 3$



(e) $k = 1, N = 100000, NW = 3$



(f) $k = 1, N = 100000, NW = 1000$

Figure 3.1.: Some DPSS for various values of N,NW and k

3.2. The new GNU Radio Blocks

The eigenspectra are computed by a GNU Radio Hierarchical Block the Signal Flow Diagram of which can be seen in Figure 3.2. The only parameter is the DPSS which is used to compute the eigenspectrum. Figure 3.3 shows the actual spectral estimator which is

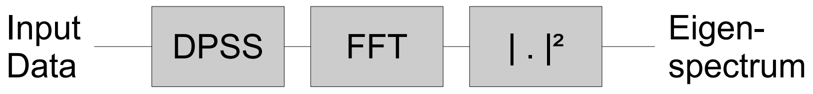


Figure 3.2.: Signal Flow Diagram of the GNU Radio Eigenspectrum calculator

also implemented as a GNU Radio Hierarchical Block. It needs the sample size N, the time bandwidth product NW, the number of tapers K and the type of weighting to use as parameters. K must not exceed N and should be smaller or equal to $2NW - 1$.

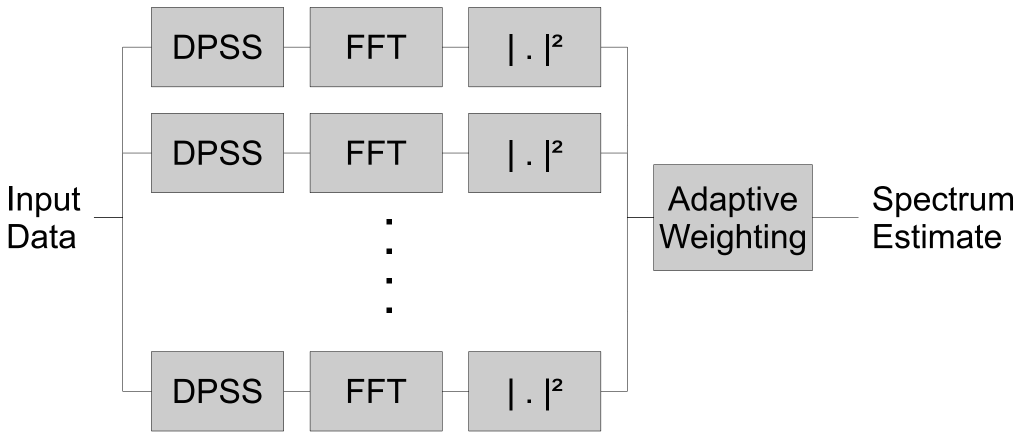


Figure 3.3.: Signal Flow Diagram of the GNU Radio Multitaper Spectrum Estimator

3.3. API - Documentation

The code written during this bachelor thesis is available from https://www.cgran.org/wiki/SpecEst. An example which uses Thomson's multitaper method is located in *apps/specest_noisy_sinusoid_mtm.py*. The multitaper spectrum estimator consists of the following files

- python/specest_mtm.py
- python/specest_gendpss.py
- lib/specest_adaptiveweighting_vff.h

- lib/specest_adaptiveweighting_vff.cc

We start with importing the spectral estimation toolbox

```
import specest
```

The multitaper spectrum estimator is made up of two classes: specest.mtm and specest.gendpss.

specest.mtm

The class specest.mtm is derived from the class gr.hier_block2 and can therefore be used in a GNU Radio flow graph. Specest.mtm calls specest.gendpss to generate the DPSS. So the DPSS do not have to be passed as an extra parameter.

Parameters

N: Length of the DPSS

NW: Time bandwidth product. Usually is of value 2, 2.5, 3.0, 3.5, or 4.

K: Numbers of tapers to use. K ‘should be smaller than $2NW - 1$.

weighting: Which type of weighting to use for the eigenspectra. Choices can be 'unity', 'eigenvalue' or 'adaptive'.

Example

```
self.source = gr.sig_source_c(...)
self.mtm = specest.mtm( N = 128, NW = 3, K = 5, weighting = 'adaptive')
self.sink specest.spec_sink_f(...)
self.connect( self.source, self.mtm, self.sink)
```

specest.gendpss

The class gendpss generates the DPSS and their eigenvalues. They are stored in the data attributes dpssarray and lambdas.

Parameters

N: Length of the DPSS

NW: Time bandwidth product. Usually is of value 2, 2.5, 3.0, 3.5, or 4.

K: Numbers of Tapers to use. K should be smaller than $2NW - 1$.

EPS: sets the accuracy with which the eigenvalue of the **T**-matrix are computed. The default value is 1E-6. If EPS is set to ϵ it follows for the accuracy of the estimated eigenvalue $\lambda_e \in [\lambda - \epsilon, \lambda + \epsilon]$ with λ being the true eigenvalue.

DELTA: sets the accuracy of the DPSS computation. The default value is 1E-6. If DELTA is set to Δ the computation of the eigenvector stops as soon as the first component of the eigenvector changes less than Δ between two iterations.

Example

```
#generates the first 5 DPSS and their eigenvalues
mydpss = specest.gendpss(N = 256, NW = 3, K = 5)
#prints the third dpss
print mydpss.dpssarray[2]
#prints all computed eigenvalues, starting with the largest.
print mydpss.lambdas
```

specest.adaptiveweighting_vff

If adaptive weighting is selected this GNU Radio Block is automatically used by specest.mtm, weights the different eigenspectra and returns a spectrum estimate.

Parameters

vlen: length of the eigenspectra which are connected by GNU Radio

lambdas: the different eigenvalues belonging to the eigenspectra

4. Summary and Conclusion

The reader is now familiar with the fundamentals of the multitaper method:

- The use of multiple spectrum estimates to reduce the variance,
- the benefit of DPSS due to their orthogonality and their optimal energy concentration,
- the different weighting methods,
- which parameters of the multitaper spectrum estimator can be changed and their effect on the spectrum estimate.

The two test cases introduced here, with a known, analytically determined spectrum can be used to evaluate the accuracy of any spectrum estimation routine.

The code written during the work on this bachelor thesis is now part of the GNU Radio Spectral Estimation Toolbox, which enables the user to easily implement the multitaper method in any preexisting GNU Radio program.

However, there still remains some work to be done, which would have gone beyond the scope of a bachelor thesis. In his article [12] Thomson introduced a new method to compute spectrum estimates called quadratic inverse theory which takes into account that the spectrum may vary within the interval $[-W, W]$ of the spectral windows. In [13] the performance of quadratic inverse theory compared to original multitapering is analyzed. These two papers should be a good starting point for people who wish to further improve the multitaper method in the GNU Radio Spectral Estimation Toolbox.

A. Appendix

A.1. Proofs

A.1.1. Variance of a sum of independent random variables

Let X and Y be two independent random variables. The variance of their sum is then given by

$$\begin{aligned}
\operatorname{Var}\{X+Y\} &= \mathrm{E}\left\{\left[(X-\mu_X)+(Y-\mu_y)\right]^2\right\} \\
&= \mathrm{E}\left\{(X-\mu_X)^2+2(X-\mu_X)(Y-\mu_y)+(Y-\mu_y)^2\right\} \\
&= \operatorname{Var}\{X\}+\operatorname{Var}\{Y\}+2\,\mathrm{E}\left\{(X-\mu_X)(Y-\mu_Y)\right\} \\
&= \operatorname{Var}\{X\}+\operatorname{Var}\{Y\}+2\,\mathrm{E}\left\{XY-X\mu_Y-\mu_X Y+\mu_X\mu_Y)\right\} \\
&= \operatorname{Var}\{X\}+\operatorname{Var}\{Y\}\ .
\end{aligned}$$

Conversely for a sum of independent random variables

$$\operatorname{Var}\left\{\sum_{k=0}^{K-1} X_k\right\} = \sum_{k=0}^{K-1} \operatorname{Var}\{X_k\}\ .$$

If all random variables are identically distributed

$$\operatorname{Var}\left\{\sum_{k=0}^{K-1} X\right\} = K\cdot\operatorname{Var}\{X\}$$

and for the variance of the average value

$$\operatorname{Var}\left\{\frac{1}{K}\sum_{k=0}^{K-1} X\right\} = \frac{1}{K}\cdot\operatorname{Var}\{X\}\ .$$

A.1.2. Expected Value of the Eigenspectra

We start with the definition of the eigenspectra:

$$\hat{S}_k(f) = \left|\sum_{n=0}^{N-1} \mathbf{u_k}(\mathbf{n})x(n)e^{-j2\pi fn}\right|^2\ .$$

which is identical to

$$\hat{S}_k(f) = \sum_{n=0}^{N-1}\sum_{m=0}^{N-1} \mathbf{u_k(n)u_k^*(m)}x(n)x^*(m)e^{-j2\pi f(n-m)} \; .$$

The estimated value of the eigenspectra is given by

$$\mathrm{E}\{\hat{S}_k(f)\} = \sum_{n=0}^{N-1}\sum_{m=0}^{N-1} \mathbf{u_k(n)u_k^*(m)}r(n-m)e^{-j2\pi f(n-m)} \; ,$$

where $r(n-m)$ is the autocovariance sequence. Plugging in

$$r(n-m) = \int_{-\frac{1}{2}}^{\frac{1}{2}} S(f)e^{j\frac{2\pi}{N}f(n-m)}df$$

gives

$$\begin{aligned}
\mathrm{E}\{\hat{S}_k(f)\} &= \int_{-\frac{1}{2}}^{\frac{1}{2}} S(f')\sum_{n=0}^{N-1}\sum_{m=0}^{N-1} \mathbf{u_k(n)u_k^*(m)}e^{-j2\pi(f-f')(n-m)}df' \\
&= \int_{-\frac{1}{2}}^{\frac{1}{2}} S(f')\left|\sum_{n=0}^{N-1} \mathbf{u_k(n)}e^{-j2\pi(f-f')}\right|^2 df' \\
&= \int_{-\frac{1}{2}}^{\frac{1}{2}} S(f')\left|U_k(f-f')\right|^2 df' \; .
\end{aligned}$$

A.1.3. Convolution Theorem of the Fourier Transform

Let $X(f)$ and $Y(f)$ denote the Fourier Transform of $x(n)$ and $y(n)$. We may than show that the convolution of $X(f)$ and $Y(f)$ is equal to the DFT of $x(n) \cdot y(n)$:

$$\begin{aligned}
X * Y &= \int_{-\frac{1}{2}}^{\frac{1}{2}} X(f')\cdot Y(f-f')df' \\
&= \int_{-\frac{1}{2}}^{\frac{1}{2}} \sum_{n=0}^{N-1} x_n e^{-j2\pi nf'} \cdot \sum_{m=0}^{N-1} y_m e^{-j2\pi m(f-f')} \\
&= \int_{-\frac{1}{2}}^{\frac{1}{2}} \sum_{n=0}^{N-1}\sum_{m=0}^{N-1} x_n y_m e^{-j2\pi mf} e^{-j2\pi f'(n-m)} \\
&= \sum_{n=0}^{N-1}\sum_{m=0}^{N-1} x_n \cdot y_m e^{-j2\pi fm} \int_{-\frac{1}{2}}^{\frac{1}{2}} e^{-j2\pi f'(n-m)} \\
&= \sum_{n=0}^{N-1}\sum_{m=0}^{N-1} x_n \cdot y_m e^{-j2\pi fm}\delta_n, m \\
&= \sum_{n=0}^{N-1} x_n \cdot y_n e^{-j2\pi fn}
\end{aligned} \tag{A.1}$$

A.1.4. Vectors with maximum Energy Concentration

Let **A** be the sinc-matrix whose elements are given by

$$A_{n,m} = \frac{\sin(2\pi W(m-n))}{\pi(m-n)} \,. \tag{A.2}$$

The eigenvalues of **A** are ordered by their value

$$\lambda_0 \geq \lambda_1 \geq \cdots \geq \lambda_{N-1} \tag{A.3}$$

and the eigenvalue decomposition of **A** is given by

$$A = Q\Lambda Q^* \,. \tag{A.4}$$

We are trying to prove that the energyconcentration is limited by the eigenvalues, which is equivalent with the claim, that the energy concentration is maximum if $\mathbf{u_k}$ is an eigenvector of **A**

$$\lambda_{N-1} \leq \frac{\mathbf{u_k^*} A \mathbf{u_k}}{\mathbf{u_k^*}\mathbf{u_k}} \leq \lambda_0 \,, \tag{A.5}$$

or

$$\lambda_{N-1} \leq \mathbf{u}_k^* A \mathbf{u}_k \leq \lambda_0 \tag{A.6}$$

if u_k is of unit length. We define a vector **w**

$$\mathbf{w} = Q^* \mathbf{u_k} \tag{A.7}$$

which leads to

$$\lambda_{N-1} \leq \mathbf{w}^* \Lambda \mathbf{w} = \sum_{k=0}^{N-1} \lambda_k \left| w_k \right|^2 \leq \lambda_0 \tag{A.8}$$

if **w** is of unit length. This is easily proved by

$$\lambda_0 - \sum_{k=0}^{N-1} \lambda_k \left| w_k \right|^2 = \sum_{k=0}^{N-1} (\lambda_0 - \lambda_k) \left| w_k \right|^2 \geq 0 \tag{A.9}$$

and

$$\sum_{k=0}^{N-1} \lambda_k \left| w_k \right|^2 - \lambda_{N-1} = \sum_{k=0}^{N-1} (\lambda_k - \lambda_{N-1}) \left| w_k \right|^2 \geq 0 \,. \tag{A.10}$$

We have shown that for any vector we can find an eigenvector with an higher or at least equal energy concentration.

A.1.5. Commuting Matrices and their Eigenvectors

Two matrices commute if $\mathbf{AB} = \mathbf{BA}$. Let $\mathbf{v}$ be an eigenvector of $\mathbf{A}$ and λ its corresponding eigenvalue. Then it follows that:

$$\mathbf{BAv} = \mathbf{B}\lambda\mathbf{v} = \lambda\mathbf{Bv} = \mathbf{ABv}\,. \tag{A.11}$$

From $\lambda\mathbf{Bv} = \mathbf{ABv}$ we deduce that $\mathbf{Bv}$ is also an eigenvector of $\mathbf{A}$. Therefore $\mathbf{Bv} = \tilde{\lambda}\mathbf{v}$. So $\mathbf{v}$ is an eigenvector of $\mathbf{B}$ as well. But for $\mathbf{B}$ the corresponding eigenvalue $\tilde{\lambda}$ is usually not equal to the eigenvalue λ of $\mathbf{A}$.

A.2. Algorithm to generate DPSS implemented in Python

```python
#!/usr/bin/python
# Generates DPSS utilizing the algorithm given in
# "A Simple Algorithm for Generating Discrete Prolate Spheroidal Sequences"
    D.M. Gruenenbacher, D.R. Hummels IEEE Vol 42 11. November 1994 \n

import math

## generates the first K slepian sequences
# @param N: Length of the FFT
# @param NW: Time Bandwidth Product usually is of value 2, 2.5, 3.0, 3.5, or
     4
# @param K: Numbers of Tapers to use. K should be smaller than 2*NW
# @param EPS: sets the accuracy with which the eigenvalue of the T-Matrix
    are computed. Default value is 1E-6.
#              If EPS is set to \f$ \epsilon \f$ it follows for the
    estimated eigenvalue \f$ \lambda_e \in [ \lambda - \epsilon, \lambda + \
    epsilon ]\f$ with \f$ \lambda \f$ being the true eigenvalue \n
# @param DELTA: sets the accuracy of the DPSS computation. Default value is
    1E-6
#              If DELTA is set to \f$ \Delta \f$ the computation of the
    eigenvector stops as soon as the first component of the eigenvector
    changes less than \f$ \Delta \f$ after an iteration.
class gendpss():
        def __init__(self, N = 512 , NW = 3 , K = 5, weighting = 'unity',
            EPS=1E-6, DELTA=1E-6):
                self.N = N
                self.NW = NW
                self.K = K
                self.weighting = weighting
                self.EPS = EPS
                self.DELTA = DELTA
                self.W = float(self.NW)/N
                # cosfac and nfac save some computation time because they do
                     not change during the execution of the following while-
                    loop
                cosfac = math.cos( 2 * math.pi * self.W )
        nfac = (N-1)/2.
        # generates the t-matrix
        self.diag = [1.0] * N
        self.subdiag = [0.0] * (N-1)
        self.diag[0] = (nfac**2)*cosfac
        i = 1
        while i < N:
                self.diag[i] = ( (nfac-i)**2 )*cosfac
                self.subdiag[i-1] = (1./2.)*i*(N-i)
                i = i + 1
        # space to save the eigenvalues and eigenvectors
        self.lambdas = []
        eigenvector = [1.1]*N
        self.dpssarray = []
        i = 0
        while i < K:
                self.lambdas.append(self.func_get_eigenvalue( k=N-(i
                    +1)))
                self.dpssarray.append(eigenvector[:])
                while True:
                        oldeigenvector = self.dpssarray[i][0]
                        self.dpssarray[i] = self.func_eigval2eigvec(
                            eigenvalue = self.lambdas[i],d=self.
                            dpssarray[i])
                        if DELTA > math.fabs( math.fabs(
                            oldeigenvector) - math.fabs(self.
                            dpssarray[i][0]) ):
                                break
                self.lambdas[i] = self.func_get_SINC_eigval(self.
                    dpssarray[i])
                i = i + 1

## Returns a sequence which contains all the sign changes of the
    characteristic polynoms of the T-Matrix at x
# the number of sign changes is equal to the number of eigenvalues
    which are smaller than x
# @param x: the value for which the characteristic polynoms are
    computed
# @return: returns a sequence which contains all the sign changes of
     the characteristic polynoms
def func_charpol(self,x):
        charpol = [0.0]*(self.N+1) # a NxN-matrix has N+1 "sub"
            characteristic polynoms
        charpol[0] = 1
        charpol[1] = self.diag[0] - x
        i = 2
        # See Golub and Van Loan, "Matrix Computations" 1989 2nd ed.
             page 437 for details about recursive relationship
        while i <= self.N:
                #we can calculate our subcharacteristic polynoms
                    recursively
                charpol[i] = ( self.diag[i-1] - x )*charpol[i-1] - (
                     (self.subdiag[i-2])**2 )*charpol[i-2]
                # this is necessary because if N is large enough
                    charpol[i] becomes extremely large and at one
                    point python can no longer cope with these large
                     numbers.
                # changing of charpol[i] is possible since we are
                    only interested in the sign of the results and
                    not the absolute value of them
                if math.fabs(charpol[i]) > 1E10:
                        charpol[i] /= math.fabs(charpol[i-1])
                        charpol[i-1] /= math.fabs(charpol[i-1])
                charpol[i-2] /= math.fabs(charpol[i-2])
                i = i + 1
        return charpol


## Gives back the number of sign changes in a sequence
# @param seq:
# @return: the number of sign-changes in seq
def func_signchanges(self,seq):
        count = 0
        i = 1
        while i < len(seq):
                p = seq[i-1]
                pp = seq[i]
                if p*pp <= 0:
                        count = count + 1
                i = i + 1
        return count

## Calculates the kth eigenvalue of the T-Matrix
# @return: the kth eigenvalue
def func_get_eigenvalue(self,k):
        # the interval in which the eigenvalues can be found is
            limited by the Gershgorin circle theorem. See Golub and
            Van Loan, "Matrix Computations" 1989 2nd ed. p341
        maxrange = 0.125*( (self.N-1)*(self.N+2) + self.N*(self.N+1)
             ) + 0.25
        lmax = maxrange
        lmin = -maxrange
        seq = [0.0]*(self.N+1)
        while math.fabs(lmax - lmin) > self.EPS: # EPS sets the
            accuracy with which the eigenvalues are computed
                lambda_k = (lmax + lmin)/2
                seq=self.func_charpol(x=lambda_k)
                if self.func_signchanges(seq) >= k+1:
                        lmax = lambda_k
                else:
                        lmin = lambda_k
        return lambda_k
```

```
## Calculates the eigenvector to a corresponding eigenvalue
# this is done by using inverse iteration (see F. S. Acton "
    Numerical Methods that work" 2nd ed 1990 p356)
# This function is basically an implementation of the Thomas-
    Algorithm (http://en.wikipedia.org/wiki/
    Tridiagonal_matrix_algorithm) to solve the System of linear
    Equations
# @param eigenvalue: the eigenvalue whose eigenvector is to be found
# @param d: the estimate of the eigenvector used in the inverse
    iteration method
def func_eigval2eigvec(self,eigenvalue,d):
        a = [0.0] * (self.N-1)
        b = [0.0] * self.N
        c = [0.0] * (self.N-1)
        k = 0
        j = 0
        #the matrix needs to be reseted after each iteration
        #it isnt really necessary to reset a because it doesnt
            change at all
        #b on the other hand needs a reset after change of the
            eigenvalue but not after each iteration
        #this leaves room for optimazition
        #setting up the Matrix (T - eigenvalue*I)
        while j < self.N-1:
                a[j] = self.subdiag[j]
                b[j] = self.diag[j] - eigenvalue
                c[j] = self.subdiag[j]
                j = j + 1
        b[j] = self.diag[j] - eigenvalue # a and c are shorter than
            b so b[N-1] must be assigned extra
        j = 0

        c[0] = c[0]/b[0]
        d[0] = d[0]/b[0]
        j = 1
        while j < self.N-1:
                id = 1/(b[j] - c[j-1]*a[j-1] ) #saves a little
                    computation time
                c[j] = c[j] * id
                d[j] = ( d[j] - d[j-1]*a[j-1] ) * id
                j = j  + 1
        d[j] = ( d[j] - d[j-1]*a[j-1] ) / (b[j] - c[j-1]*a[j-1] )
        j = j - 1
        while j >= 0:
                d[j] = d[j] - c[j]*d[j+1]
                j = j - 1
        #normalize d
        d = self.func_normalizevector(d)
        return d

## Normalizes a vector
# @param vector: the vector to be normalized
# @return: the normalized vector
def func_normalizevector(self,vector):
        j = 0
        sum = 0
        while j < len(vector):
                sum = sum + vector[j]**2
                j = j + 1
        sum = sum**0.5
        j = 0
        while j < len(vector):
                vector[j] = vector[j]/sum
                j = j + 1
        return vector

## Returns the eigenvalue of the sinc-matrix for a given eigenvector
# @param eigenvector: the eigenvector whose eigenvalue is to be
    calculated
# @return: the eigenvalue belonging to the given eigenvector
def func_get_SINC_eigval(self,eigenvector):
# get the index of the biggest element in our eigenvector. This is
    used because later we devide through a vector element and some
    dpss-values are close to zero, which could be critical.
# speed can propably increased if you search for the biggest element
     and its index at the same time.
        k = eigenvector.index(max(eigenvector))
# the eigenvalues lambda is calculated  by: A * v = lambda * v where
     A is the Sinc-Matrix. We don't need to do the full
    matrixmultiplication. One line suffices.
        linesinc = [0.0] * self.N
# this code generates the kth line of the Sinc Matrix and multiplies
     it with the eigenvector
# extra care must be taken since sin(x)/x can't be calculated by
    python for x = 0
        sum = 0
        i = 0
        while i < k:
                linesinc[i] = (math.sin(2*math.pi*self.W*(i-k)))/(
                    math.pi*(i-k))
                sum += linesinc[i]*eigenvector[i]
                i = i + 1
        # this is the critical element since python can't compute
            sin(0)/0
        linesinc[k] = (2*self.W)
        sum += linesinc[0+k]*eigenvector[0+k]
        i = k + 1
        while i < self.N:
                linesinc[i] = (math.sin(2*math.pi*self.W*(i-k)))/(
                    math.pi*(i-k))
                sum += linesinc[i]*eigenvector[i]
                i = i + 1
        eigenvalue = sum/eigenvector[k]
        return eigenvalue
```

A.3. Sample values of Marple's Test Case

The following table consists of 128 samples of a coloured noise process and four complex sinusoidals. Its analytically determined spectrum can be seen in figure 2.8(a).

	real part	imaginary part
X(0)	-0.078231882361905	0.046554012414623
X(1)	-0.080416603749162	-0.047495648597067
X(2)	0.020071352460406	-0.070697109577191
X(3)	0.081379978469209	0.001580423683994
X(4)	0.036481334048571	0.060315076101007
X(5)	-0.031931475719939	0.026992958767949
X(6)	-0.038408246243542	-0.036958895490966
X(7)	0.004297009608049	-0.028212308231944
X(8)	0.023254471293366	0.019518159320225
X(9)	-0.002492122558194	0.043461036747208
X(10)	-0.028816011568728	-0.008251238015715
X(11)	-0.016154980886163	-0.013942591030466
X(12)	0.043106903441201	-0.009391902232339
X(13)	-0.001048069870274	-0.067276381013745
X(14)	-0.043049527173190	0.045319263938455
X(15)	0.073382825926230	0.032616487375655
X(16)	-0.008161112024579	-0.030231355573440
X(17)	-0.058336298680298	0.061787509169784
X(18)	0.058187337271817	-0.007904807909308
X(19)	-0.055560205720559	-0.062239486800446
X(20)	0.018406989553324	-0.002529242010529
X(21)	0.056976885528990	0.036561105628250
X(22)	-0.119581277704021	0.026302427963765
X(23)	0.015601832794292	-0.020891684316827
X(24)	0.084274688407855	-0.091884778939501
X(25)	0.009389323417731	0.025267679062016
X(26)	0.029409622769714	0.126470685732616
X(27)	-0.053163011094968	-0.042214865121878
X(28)	-0.130068194883371	-0.026040690410182
X(29)	0.087615658287917	-0.053545782224909
X(30)	0.112564546011280	0.022089069144501
X(31)	-0.053323370027976	0.120151934492860
X(32)	-0.080093515409295	-0.070866926785253
X(33)	-0.062889914037207	-0.059838837237057
X(34)	0.140869032550362	-0.077394493013854
X(35)	0.080365172278010	0.053495902067414
X(36)	-0.085362758357155	0.197531072327800

X(37)	-0.062561237560336	-0.046155418384187
X(38)	-0.054028637398065	-0.143766728982005
X(39)	0.109601522808971	-0.044491501024780
X(40)	0.112344498301441	0.081184572607757
X(41)	-0.151368154194385	0.140477157575667
X(42)	-0.096189412200679	-0.066616307779700
X(43)	0.037776212086813	-0.167143519117835
X(44)	0.106137308133242	-0.000699886222843
X(45)	0.174989443634662	0.122740582445881
X(46)	-0.173558695399012	0.115093898995781
X(47)	-0.184584719824726	-0.066852651298921
X(48)	0.081897616984043	-0.169641544665229
X(49)	0.141938258298747	0.010515621473589
X(50)	0.068098862947880	0.174130564993833
X(51)	-0.171806914307641	0.085014141332306
X(52)	-0.113884106893988	-0.163658784557529
X(53)	0.079557204843315	-0.184418903733923
X(54)	0.198256223351677	0.098259784218946
X(55)	0.027082686565476	0.160693930351387
X(56)	-0.241552374764569	0.070472239129052
X(57)	-0.038479915167983	-0.087690711830870
X(58)	0.132450812884924	-0.210218762528369
X(59)	0.094767887727156	0.047713628682519
X(60)	-0.013469645736850	0.253642296896555
X(61)	-0.165603469700234	0.012079875560167
X(62)	-0.101128920215741	-0.264428422077575
X(63)	0.129381807293577	-0.050061022580270
X(64)	0.262647892989285	0.142885097274828
X(65)	-0.083869456040939	0.114132276532347
X(66)	-0.269647773186558	-0.001529532975395
X(67)	0.008732010606494	-0.132581805673703
X(68)	0.185262364586126	-0.124481030492487
X(69)	0.052627636892917	0.147967562091346
X(70)	-0.070711415385877	0.215910413890873
X(71)	-0.109526709166035	-0.157763141017061
X(72)	-0.124622068822747	-0.146742412965451
X(73)	0.273768742004888	-0.056418495478786
X(74)	0.144637153444810	0.111423527104707
X(75)	-0.272714025885377	0.232496402647931
X(76)	-0.069773528071616	-0.112476103320278
X(77)	0.104186536872793	-0.191988648146408
X(78)	0.078477766266915	0.021923181529223
X(79)	0.020299100042407	0.148408404760470

X(80)	-0.038750587387117	0.079037191148215
X(81)	-0.212828263667641	-0.132172738033391
X(82)	0.038303786882256	-0.098857702386044
X(83)	0.276902709233054	-0.012765823025079
X(84)	-0.022579036708743	0.145612805172806
X(85)	-0.128271047780878	0.128881648959647
X(86)	-0.110655201514654	-0.208445442696606
X(87)	0.080112890628170	-0.068199651011773
X(88)	0.138759150763054	0.115441334273267
X(89)	-0.043550339374042	0.041948593303200
X(90)	-0.116203129440673	0.071697610336570
X(91)	-0.066886675270656	-0.093606648661959
X(92)	0.139308938305927	-0.208424651842921
X(93)	0.080362838064696	0.138995437921763
X(94)	-0.070806378662012	0.150786899004353
X(95)	-0.055512785784534	-0.096231875777708
X(96)	-0.027482213916251	0.011797107737354
X(97)	0.090509730113523	-0.068370261723655
X(98)	0.018363559167642	0.011946403952815
X(99)	-0.016771414187295	0.136710060249749
X(100)	-0.068939618279489	-0.073248448415675
X(101)	-0.094685143583486	-0.092264522918512
X(102)	0.154989805053050	0.023350698536878
X(103)	0.058100564576560	0.065431240873456
X(104)	-0.096307457719590	-0.043763673525896
X(105)	0.023643807911135	0.055030537687307
X(106)	0.008038520605320	-0.020792226147762
X(107)	-0.060839503781481	-0.101406039374830
X(108)	0.068272484255268	0.145049749889963
X(109)	0.004233864342059	0.000097205099103
X(110)	-0.100570494743344	-0.077791150553911
X(111)	0.053760643308638	0.017040151732299
X(112)	0.062335647788010	-0.012379083301337
X(113)	-0.061345300302955	-0.026159961464935
X(114)	-0.015664902933044	0.013482810629614
X(115)	0.089054256275813	0.051445867292075
X(116)	-0.026896830481728	0.015338065199978
X(117)	-0.084045510754216	-0.034313055922456
X(118)	0.062849320381398	0.017729928619674
X(119)	0.018164922590693	-0.059933850076363
X(120)	-0.045362923370175	-0.001782604365117
X(121)	-0.002785117711907	0.155054662053910
X(122)	-0.009449810738490	-0.175400217698032

X(123)	-0.013770487373540	0.010134124589825
X(124)	0.088350229943394	0.072149547364144
X(125)	0.097252655426681	-0.100874214599954
X(126)	-0.203635941422627	0.193693151138165
X(127)	-0.014152875875475	-0.080737709464336

Bibliography

[1] P. Stoica and R. Moses, *Spectral Analysis of Signals*. Pearson Education, 2005.

[2] D. J. Thomson, "Spectrum Estimation and Harmonic Analysis," *Proceedings of the IEEE*, vol. 70, no. 9, pp. 1055 – 1096, 1982.

[3] D. Slepian, "Prolate Spheroidal Wave Functions, Fourier Analysis, and Uncertainty - V: The Discrete Case," *The Bell System Technical Journal*, vol. 57, no. 5, pp. 1371 – 1430, 1978.

[4] D. B. Percival and A. T. Walden, *Spectral analysis for physical applications: multitaper and conventional univariate techniques*. Cambridge Univ. Pr., 1993.

[5] S. L. Marple, *Digital spectral analysis : with applications*. Prentice-Hall, 1987.

[6] P. Stoica and T. Sundin, "On nonparametric Spectral Estimation," *Circuits, Systems and Signal Processing*, vol. 18, pp. 169–181, 1999.

[7] D. M. Gruenbacher and D. R. Hummels, "A Simple Algorithm for Generating Discrete Prolate Spheroidal Sequences," *IEEE Transactions on Singal Processing*, vol. 42, no. 11, pp. 3276 – 3278, 1994.

[8] F. A. Grünbaum, "Eigenvectors of a Toeplitz Matrix: Discrete Version of the Prolate Spheroidal Wave Functions," *SIAM. J. on Algebraic and Discrete Methods*, vol. 2, no. 2, pp. 136 – 141, 1981.

[9] G. H. Golub and C. F. V. Loan, *Matrix Computations*. The Johns Hopkins University Press, 1989.

[10] F. S. Acton, *Numerical Methods That Work*. The Mathematical Association of America, 1990.

[11] C. Conte, Samuel D. ; De Boor, *Elementary numerical analysis : an algorithmic approach*. McGraw-Hill, 1980.

[12] D. J. Thomson, "Quadratic-inverse spectrum estimates:applications to palaeoclimatology," *Philosophical Transactions of the Royal Society of London A*, vol. 332, pp. 539–597, 1990.

[13] G. Prieto, R. Parker, D. Thomson, F. Vernon, and R. Graham, "Reducing the bias of multitaper spectrum estimates," *Geophysical Journal International*, vol. 171, pp. 1269–1281, 2007.