Frequency estimation using chromatic derivatives

Aleksandar Ignjatovic 2010 ©
Version 2.0, July 8, 2010.

Changes in version 2: (1) now no need to specify the SNR in the detection part of the algorithm - it is determined automatically (by a rough but reasonably accurate hack which will be improved later) from the noisy signal alone, assuming that there are at least a few legitimate fragments; (2) counters in the previous version were off and showed more errors than actually present; (3) much faster implementation with much less memory used.
To do: (1) to clean up and include a new part of the code which increases noise robustness of the algorithm in a way explained further in the tutorial; (2) to implement colored noise environment; (3) to implement all of this and morre in Matlab, hopefully in August after I come back from a trip home.

Introduction

This is a **tutorial,** which, besides implementing the frequency estimation algorithm from the paper [1] "***Frequency estimation using time domain methods based on robust differential operators***", available at http://www.cse.unsw.edu.au/~ignjat/diff/ is designed to illustrate along the way some useful and important properties of chromatic derivatives. Thus, there are lots of things calculated which are unnecessary for the frequency estimation algorithm (and slow down everything a lot), but whose values and plots highlight various features of chromatic derivatives. After the first reading, you might want to **comment out the parts of the code needed for plots, to have the program run much faster**. Soon (hopefully in August) we will provide at at http://www.cse.unsw.edu.au/~ignjat/diff/ a much faster *Matlab* implementation, as well as several improvements of the basic algorithm implemented in this file, most notably one that makes the algorithm much more noise robust and can handle colored noise, see later in this file.

Before each run you must quit the *Mathematica* kernel to clear all the variables; the last command *Quit[ ]* in this file does this automatically after each run, but you lose the values computed but not printed out or plotted; if you comment it out, remember to quit the kernel manually before each new run!

The paper we mentioned above, together with all other papers mentioned in this file and listed at the bottom of this tutorial, is available at http://www.cse.unsw.edu.au/~ignjat/diff/.

The theory of chromatic derivatives is described in most detail in [2] "***Chromatic Derivatives, Chromatic Expansions and Associated Spaces***", East Journal on Approximations, Volume 15, Number 3 (2009), 263-302, or, in a more condensed form, in [3] "***Chromatic derivatives and local approximations",*** IEEE Transactions on Signal Processing, Volume 57, Issue 8, 2009.
A good way to start is to read sections I and II of that paper and then go through this tutorial up to the "Generating the Input Signal" section, to see how the theory of chromatic derivatives works in practice (simulations). Then one can read [4] "***Signal interpolation using numerically robust differential operators***", to understand how we generate the input signal and then read/execute this tutorial up to "Frequency Estimation Section". Then one can read [1] "***Frequency estimation using time domain methods based on robust differential operators***" and finally finish going through this tutorial.

At the moment, only the white noise environment is implemented. Thus, since the power spectrum density function of the noise is constant, the corresponding family of chromatic derivatives are the (normalized and rescaled) Legendre polynomials; see [1] for the details. We will soon post a version for colored noise which first generates the right family of orthogonal polynomials from the power spectrum density of the noise, as explained in [1]. This version also involves adapting the Remez exchange algorithm to generate the FIR approximations of the corresponding differentiation filters.

* * * * * * * * * * * * * * * * * * * *

IMPLEMENTATION

****

We provide here some examples which we have used in testing. Choose which one you want to run by setting the value of "case"; see below for the range provided in this file.Each parameter is explained just before its use. **NOTE:** the parameters used for the detection and estimation algorithm for a given minimal duration of segments *minLength* , such as *INT[i]* have NOT been optimized at all, due to lack of time.

```
In[1]:= case = 5;  (* 1 – 6, see below *)
        minimum = 1;  (* minimal number of sinusoidal
         components in fragments to be generated in a randomized way *)
        maximum = detect; (* maximal number of sinusoidal components in fragments
         to be generated; "detect" (≤ 4) is the maximal number of components
         which the algorithm can detect (at the moment); fragments containing
         more than "detect" many sinusoids will be classified as transients *)
        number = 30;     (* total number of segments of signals to be detected *)
```

******************* A CHOICE OF SOME EXAMPLES ****************************

Example: detecting pieces of length of only 2 Nyquist rate intervals which are single sinusoids, in SNR of 35db.

Note: Since the signal interpolation generates a 15 times continuously differentiable function, it produces on each side of a signal support additional intervals of length 3-4 Nyquist rate intervals where the interpolated signal is still quite an accurate extrapolation of the original signal as defined over its support; see [4] for details and the plots at the very end of this file. Thus, in case 1, the signal support of 2 Nyquist rate intervals produces in effect approximately 8 - 10 Nyquist rate intervals long section of single sinusoids. In fact, one can even set minLength to **0**, which gives a support consisting of a single point; however, the (smoothnes of) interpolation still produces an interval of lenth 6 to 8 (3-4 on each side of the single point of support) over which the signal is a good approximation of a sine wave. If you set minLength to 0, set treshold to 3.5 and see what happens; however, the Fourier Transform plots might be off in this extreme case, showing incorect measured frequency due to a small bug I have no time to correct now...

```
In[3]:= If[case == 1, {
          detect = 1;        (* maximal number of sinusoids to be detected;
          it can range from 1 to four, at the moment;
          higher values of detect and more noise naturally need segments of longer duration *)
          minLength = 0;     (* minimal length of the support
           of each segment to be detected  *)
          maxLength = minLength;  (* maximal length of supports of fragments to be generated;
          does not impact detection.  *)
          minFreq = .3;     (* minimal frequency of sinusoidal components in all fragments;
          at the moment we cannot handle DC but this can be fixed with
           more complexity by looking at operators of odd order as well *)
          maxFreq = 3;      (* maximal frequency of sinusoidal components in all fragments  *)
          NDB = 25;          (* Signal to Noise; noise is white Gaussian;
          since the signal is twice oversampled and the noise is WGA,
          the bandwidth of the noise is twice the signal bandwidth;
          thus, signal to in-band noise is 3db higher *)
          INT[1] = 4;       (* the number of points over which the signal has to
            satisfy a second order differential equation is equal to 2INT[1]+1;
          if we are detecting more sinusoids we need higher order differential operators
           and they need different, longer supports so we will have bellow INT[2], INT[3],...*)
          treshold = 3.5;    (* the constant giving the multiple of the RMS of the
           noise which bounds the square root of the minimal eigenvalue of the
           normalized correlation matrix; to be explained later and in the paper [1] *)
          guard = 0;       (* number of points to throw away on the edges of detected
           intervals to reduce the error and the false positives in detection stage *)
          ampModify = 1;      (* decrease  of the amplitudes of the sinusoidal components;
          if equal to 1, all components have the same amplitude  *)
         }];
```

Example: detecting pieces of length of only 4 Nyquist rate interval which are single sinusoids with SNR = 25db (thus, together with a subsection of the interpolation between the fragments, in total about 10 - 12 Nyquist rate intervals long)

```
In[4]:= If[case == 2, {
          detect = 1;
          minLength = 4;
          maxLength = minLength;
          minFreq = .3;
          maxFreq = 3;
          NDB = 25;
          treshold = 1.6;
          INT[1] = 6;
          guard = 0;
          ampModify = 1;
           }];
```

****

Example: detecting pieces of length only 8 Nyquist rate intervals which are linear combinations of at most two sinusoids; (thus, together with a part of interpolation, in total about 14 - 16 Nyquist rate intervals long)

```
In[5]:= If[case == 3, {detect = 2;
          minLength = 8;
          maxLength = minLength;
          minFreq = .3;
          maxFreq = 3;
          NDB = 25;
          treshold = 1.6;
          guard = 4;
          INT[1] = 6;
          INT[2] = 10;
          ampModify = 1; }]
```

****

Example: detecting pieces of length 16 Nyquist rate intervals which are linear combinations of at most three sinusoids

```
In[6]:= If[case == 4, {detect = 3;
           minLength = 16;
           maxLength = minLength;
           minFreq = .3;
           maxFreq = 3;
           NDB = 30;
           treshold = 1.8;
           guard = 5;
           INT[1] = 8;
           INT[2] = 12;
           INT[3] = 16;
           ampModify = 1; }];
```

****

Example: detecting pieces of length 24 Nyquist rate intervals which are linear combinations of at most three sinusoids with attenuation of amplitude

```
In[7]:= If[case == 5, {detect = 3;
           minLength = 24;
           maxLength = minLength;
           minFreq = .3;
           maxFreq = 3;
           NDB = 30;
           treshold = 1.8;
           guard = 5;
           ampModify = .5;
           INT[1] = 12;    (* 6,8,12 *)
           INT[2] = 16;
           INT[3] = 20;
          }];
```

****

Example: detecting pieces of length 32 Nyquist rate intervals which are linear combinations of at most four sinusoids

In[8]:=

```
If[case == 6, {detect = 4;
   minLength = 32;
   maxLength = minLength;
   minFreq = .3;
   maxFreq = 3;
   NDB = 35;
   treshold = 2;
   guard = 10;
   ampModify = 1;
   INT[1] = 8;
   INT[2] = 16;
   INT[3] = 20;
   INT[4] = 24;
  }];
```

```
**********************************************************************
```

Loading Filters:

We start by specifying which filterbanks of chromatic derivatives we are going to use.

Note:

1) **Filters are applied to twice oversampled signals;**

2) **The Legendre polynomials correspond to the white Gaussian noise; later we will extend the simulation to colored noise.**

We define the corresponding recursion coefficients, the corresponding orthogonal polynomials and the corresponding chromatic derivatives; **please reffer to [3] or [2]**.

First we give the three term recursion coefficients for two families; one (*cfc[n]*) for the "clean" signals with bandwidth π (produced by a double precission evaluation of analytic expressions), and another (*cfn[n]*) for the "real world" sampled noisy signals with an extended bandwidth up to *extend/100* π = 1.1π, which allow accurate differentiation of signals with a slight out of band content, and in the presence of noise (with out of band content as well).

In[9]:=

```
name = "legen"; (* specifies the Legendre polynomials;
                     we need it as a variable to be able to hanle in the future the case
                          of colored noise *)
```

In[10]:=

```
extend = 110;
(* specifies how much the bandwidth of the filters for the noisy inputs has been
                         extended - such bandwidth is extend/100 π = 1.1 π; thus, it is a
                              ten percent
 extension for the present implementation filters *)
```

In[11]:=

$$\mathtt{If}\Big[\mathtt{name} == \texttt{"legen"},\ \Big\{\mathtt{cfn[n\_]} = \frac{\mathtt{extend\ /\ 100}\ \pi\ \mathtt{(n+1)}}{\sqrt{\mathtt{4\ (n+1)}^{2}-\mathtt{1}}};\ \mathtt{cfc[n\_]} = \frac{\pi\ \mathtt{(n+1)}}{\sqrt{\mathtt{4\ (n+1)}^{2}-\mathtt{1}}};\ \Big\},$$
$$\{\mathtt{Print[\texttt{"ERROR"}];\ Quit[];}\}\Big];$$

Orthogonal polynomials for the bandwidth π:

In[12]:=

$$\mathtt{TC[0,\ w\_]} = \mathtt{1};\ \mathtt{TC[1,\ w\_]} = \frac{\mathtt{w}}{\mathtt{cfc[0]}};$$
$$\mathtt{TC[deg\_,\ w\_]} := \mathtt{TC[deg,\ w]} = \frac{\mathtt{w}}{\mathtt{cfc[deg-1]}}\ \mathtt{TC[deg-1,\ w]} - \frac{\mathtt{cfc[deg-2]}}{\mathtt{cfc[deg-1]}}\ \mathtt{TC[deg-2,\ w]};$$

Orthogonal polynomials for the bandwidth *extend/100 π = 1.1 π*: (thus, *cfn[i]* in place of *cfc[i]*)

In[14]:=

$$\mathtt{TN[0,\ w\_]} = \mathtt{1};\ \mathtt{TN[1,\ w\_]} = \frac{\mathtt{w}}{\mathtt{cfn[0]}};$$
$$\mathtt{TN[deg\_,\ w\_]} := \mathtt{TN[deg,\ w]} = \frac{\mathtt{w}}{\mathtt{cfn[deg-1]}}\ \mathtt{TN[deg-1,\ w]} - \frac{\mathtt{cfn[deg-2]}}{\mathtt{cfn[deg-1]}}\ \mathtt{TN[deg-2,\ w]};$$

Chromatic derivatives, for functions with one free parameter (variable) *m* and for both families: (I) with the bandwidth π:

In[16]:=

$$\mathtt{KC[f\_,\ 0,\ m\_,\ t\_]} := \mathtt{f[m,\ t]};$$
$$\mathtt{KC[f\_,\ 1,\ m\_,\ t\_]} := \frac{\mathtt{1}}{\mathtt{cfc[0]}}\ \partial_{\mathtt{t}}\ \mathtt{f[m,\ t]};$$
$$\mathtt{KC[f\_,\ n\_,\ m\_,\ t\_]} :=$$
$$\mathtt{KC[f,\ n,\ m,\ t]} = \mathtt{Expand}\Big[\Big(\frac{\mathtt{1}}{\mathtt{cfc[n-1]}}\ \partial_{\mathtt{t}}\mathtt{KC[f,\ n-1,\ m,\ t]} + \frac{\mathtt{cfc[n-2]}}{\mathtt{cfc[n-1]}}\ \mathtt{KC[f,\ n-2,\ m,\ t]}\Big)\Big];$$

and (II) with the bandwidth *extend/100 π = 1.1 π*:

In[19]:=

$$\mathtt{KN[f\_,\ 0,\ m\_,\ t\_]} := \mathtt{f[m,\ t]};$$
$$\mathtt{KN[f\_,\ 1,\ m\_,\ t\_]} := \frac{\mathtt{1}}{\mathtt{cfn[0]}}\ \partial_{\mathtt{t}}\ \mathtt{f[m,\ t]};$$
$$\mathtt{KN[f\_,\ n\_,\ m\_,\ t\_]} :=$$
$$\mathtt{KN[f,\ n,\ m,\ t]} = \mathtt{Expand}\Big[\Big(\frac{\mathtt{1}}{\mathtt{cfn[n-1]}}\ \partial_{\mathtt{t}}\mathtt{KN[f,\ n-1,\ m,\ t]} + \frac{\mathtt{cfn[n-2]}}{\mathtt{cfn[n-1]}}\ \mathtt{KN[f,\ n-2,\ m,\ t]}\Big)\Big];$$

The transfer functions of these ***ideal operators,*** see again [3] or [2]:

In[22]:=

```
Do[P[k, w_] = Piecewise[
    {{TN[k, 2 w], -1.1 π / 2 ≤  w ≤ 1.1 π / 2}, {0, w > 1.1 π / 2}, {0, w < -1.1 π / 2 }}], {k, 0, 8}];
Do[PP[k, w_] = Piecewise[{{TC[k, 2 w], -π / 2 ≤  w ≤ π / 2}, {0, w > π / 2}, {0, w < π / 2 }}],
  {k, 0, 15}];
```

We use normalized frequencies (in radians); thus, for the bandwidth π, **sampling at integers is a Nyquist rate sampling.** We will sample signals at half integers, thus at **twice the Nyquist rate**.

We use two types of filterbanks for evaluation of chromatic derivatives: filters U for the noisy inputs with bandwidth *extend/100* π and filters W for clean signals with bandwidth π, both designed using the Remez exchange method for producing equiripple filters. (On the mentioned website one can find also filters of higher orders than those we provide here and with different cut-off and transition frequencies; as well as the ijmplementation of the corresponding Remez exchange algorithm; see one of the *Mathematica* file on the website.)

The pass band of the filters is given by the value of the "pass" variable as the percentage of the "refernce" bandwidth π; For filters U we take pass = 105 (in % of π). The width of the transition band for filters in U is 10% of the bandwidth, thus from 1.05π to 1.15π; this is chosen by the value of the variable tran = 10 (also in % of π). The filters approximate transfer functions of chromatic derivatives for a bandwidth of *extend/100* $\pi = 1.1\pi$. Thus, with such filters we will be able to differentiate accurately signals with a banwidth of up to 1.05π and yet reject the noise component above $1.15\ \pi$. The behavior over the transition region is such that it will not amplify the component of the noise with frequencies within the transition region, as we will demonstrate this later. We have provided here such filters of orders up to 8, but the Remez exchange implementation on the website can be used all the way up to 30 (and probably above, with an appropriate number of taps chosen).

Filters W for differentiation of the "clean" signals approximate the transfer function of the chromatic derivatived for the bandwidth π, have passband $[-\pi, \pi]$ and a transition region of width 20% π (thus between π and $1.2\ \pi$). We have provided such filters for orders up to 15.

We set the directories appropriately and read the files, assembling them into a matrix U for numerical differentiation of noisy signals and a matrix W for differentiation of the "clean" signals. Coefficients of the filters **are stored as integers** obtained by rounding off 10^{15} times the real coefficients, so they are re-normalized by dividing them with 10^{15}. If you have decompressed the zip file straight to your "C" drive, you do not have to change the directory paths.

```
In[24]:= (* filters U for the noisy inputs up to the order 8 *)

In[25]:= pass = 105;
         tran = 10;

In[27]:= string = StringJoin[name, "_", IntegerString[extend]];
         SetDirectory[StringJoin["c:/ChromaticDerivatives/filterbanks/", string]];

         U = {}; Do[{filename = StringJoin[string, "_pass_", IntegerString[pass],
              "_tran_", IntegerString[tran], "_imp_", IntegerString[dg], ".txt"];
         U = Join[U, {Flatten[Import[filename, "Table"]]}]}, {dg, 0, 8}];
         U = N[U / 10^15];

In[31]:=
         (* filters W for the "clean signals"
           (obtained by a double precission evaluation of analytic expressions)
          up to the order 15 *)

In[32]:= SetDirectory[StringJoin["c:/ChromaticDerivatives/filterbanks/", name, "Q"]];

In[33]:=  W = {}; Do[{filename = StringJoin[name, "_imp_", IntegerString[dg], ".txt"];
         W = Join[W, {Flatten[Import[filename, "Table"]]}]}, {dg, 0, 15}];
         W = N[W / 10^15];
```

We set *L* to be such that $2L + 1$ is the number of taps of the U filters used; *LL* is such that $2LL + 1$ is the number of taps of the W filters used;
in our case $LL = L = (129 - 1) / 2 = 64$.

```
In[35]:= L = (Dimensions[U][[2]] - 1) / 2; LL = (Dimensions[W][[2]] - 1) / 2;
```

We now compare the transfer function of the ideal filter of order 8 and of the filter from the filterbanks U of the same order.
We also compare the transfer function of the ideal filter of order 15 and of the filter from the filterbank W of the same order.
One can compare filters of other odres by changing the values *kk* and *mm* below.

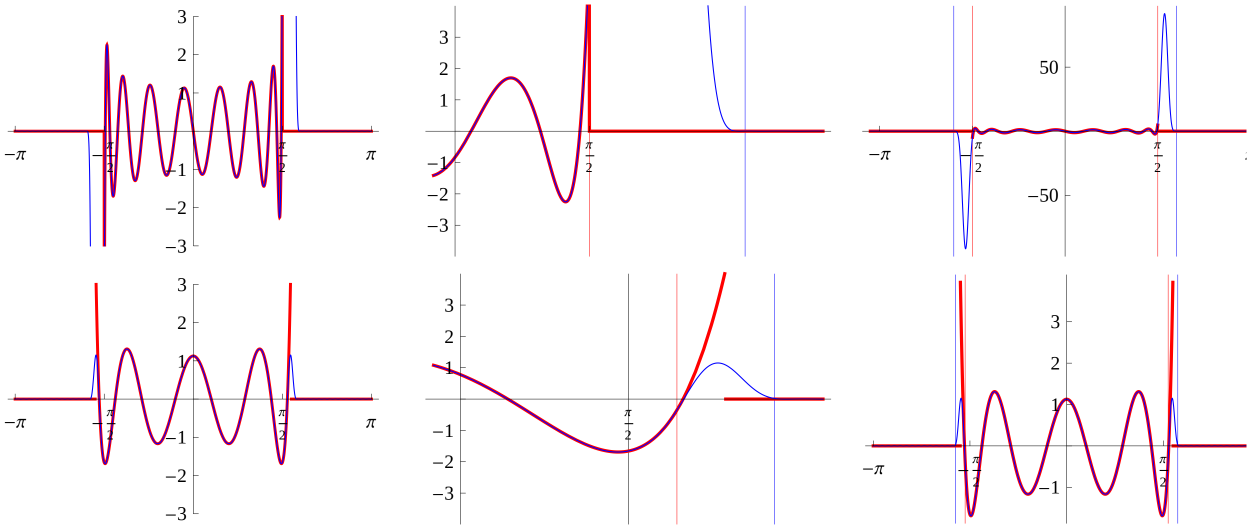
```
In[36]:= mm = 8;
         kk = 15;

         (* transfer function of the filter from U of order p *)
                                       LL
         TransFcnClean[p_, w_] := Re[   ∑   (-I)^p W[[p + 1, LL + 1 + k]] E^(I k w)];
                                      k=-LL

         (* transfer function of the filter from W of order p *)
                                       L
         TransFcnNoisy[p_, w_] := Re[   ∑   (-I)^p U[[p + 1, L + 1 + k]] E^(I k w)];
                                      k=-L
```

The following plotting is very slow; you might want to comment it out after the first reading:

In[40]:=

```
PLC1 = Plot[{PP[kk, w], TransFcnClean[kk, w]}, {w, -1.05 π, 1.05 π},
   PlotRange → All, PlotStyle → {{Red, Thickness[.008]}, Blue},
   GridLines → {{{-π / 2, Red}, {-1.2 π / 2, Blue}, {π / 2, Red}, {1.2 π / 2, Blue}}, {}},
   Ticks → {{-π, -π / 2, π / 2, π}, {-50, 50}}];

PLC2 = Plot[{PP[kk, w], TransFcnClean[kk, w]}, {w, -π, π}, PlotRange → {-3, 3},
   PlotStyle → {{Red, Thickness[.008]}, Blue}, GridLines → {{π / 2, Black}, {}},
   Ticks → {{-π, -π / 2, π / 2, π}, {-3, -2, -1, 0, 1, 2, 3}}];

PLC3 = Plot[{PP[kk, w], TransFcnClean[kk, w]}, {w, .8 π / 2, 1.3 π / 2},
   PlotRange → {-4, 4}, PlotStyle → {{Red, Thickness[.008]}, Blue},
   GridLines → {{{π / 2, Red}, {1.2 π / 2, Blue}}, {}},
   Ticks → {{-π, -π / 2, π / 2, π}, {-3, -2, -1, 0, 1, 2, 3}}];

PLN1 = Plot[{P[mm, w], TransFcnNoisy[mm, w]}, {w, -π, π},
   PlotRange → All, PlotStyle → {{Red, Thickness[.008]}, Blue}, GridLines →
    {{{-1.05 π / 2, Red}, {-1.15 π / 2, Blue}, {1.05 π / 2, Red}, {1.15 π / 2, Blue}}, {}},
   Ticks → {{-π, -π / 2, π / 2, π}, {-3, -2, -1, 0, 1, 2, 3}}];

PLN2 = Plot[{P[mm, w], TransFcnNoisy[mm, w]}, {w, -π, π}, PlotRange → {-3, 3},
   PlotStyle → {{Red, Thickness[.008]}, Blue}, GridLines → {{π / 2, Black}, {}},
   Ticks → {{-π, -π / 2, π / 2, π}, {-3, -2, -1, 0, 1, 2, 3}}];

PLN3 = Plot[{P[mm, w], TransFcnNoisy[mm, w]}, {w, .8 π / 2, 1.2 π / 2},
   PlotRange → {-4, 4}, PlotStyle → {{Red, Thickness[.008]}, Blue},
   GridLines → {{{π / 2, Black}, {1.05 π / 2, Red}, {1.15 π / 2, Blue}}, {}},
   Ticks → {{-π, -π / 2, π / 2, π}, {-3, -2, -1, 0, 1, 2, 3}}];

GraphicsGrid[{{PLC2, PLC3, PLC1}, {PLN2, PLN3, PLN1}}, ImageSize → 600]
```

Out[46]=



Compare the transfer functions of the filterbank W for the clean signals and the filterbank U for the noisy signals with the transfer functions of their corresponding ideal filters. Remember that both filters operate on twice oversampled signals; thus the bandwidth $[-\pi, \pi]$ corresponds to $[-\pi/2, \pi/2]$ on the oversampled scale:

The filterbank W (the top row of plots) is nearly perfectly accurate for the entire bandwidth $[-\pi/2, \pi/2]$, even for the operator of order 15 (left plot); the mid plot above shows the behavior in a narrow interval $[.4\,\pi, .65\,\pi]$ around $\pi/2$, with the red gridline set at $\omega = \pi/2$ and the blue gridline set at $1.2\,\pi/2 = .6\,\pi$. The right plot shows that over the transition region $[\pi/2, .6\,\pi]$ the transfer function attains a very large peak very accurately approximated by the following calculation on a grid with spacing of only .001:

```
In[47]:= Max[Table[Abs[TransFcnClean[15, π / 2 + π / 2 w / 1000]], {w, 0, 200}]]

Out[47]= 91.7551
```

Thus, such filter greatly ampifies the noise within the transition region $[\pi/2, 1.2\,\pi/2\,]$ and thus cannot be used to evaluate the chromatic derivatives of noisy signals. However, for signals obtained by evaluating analytic expressions with a double precission this is clearly not an issue.

Filter of order 8 from the filterbank U for the noisy signals is shown on the second row of plots. It is accurate with very high precission over the bandwidth $[-1.05\pi/2, 1.05\pi/2]$ as it can be seen from the left and the central plots. The black gridline on the central plot is at $\pi/2$; the red is the bandpass of the filter at $1.05\pi/2$ and the blue gridline is at the edge $1.15\pi/2$ of the transition region. As one can see, in the transition region the maximal amplitude of the transfer function is only

```
In[48]:= Max[Table[Abs[TransFcnNoisy[8, 1.1 π / 2 + π / 2 w / 1000]], {w, 0, 100}]]

Out[48]= 1.08737
```

Consequently, such filter **does not amplify the noise** from the transition region and removes it entirely above it; thus, it is extremely noise robust. This means that we can evaluate chromatic derivatives of signals with slight out of band content (up to 1.05π) **both very accurately and in a very noise robust way**. We will numerically verify these conclusions below.

****

GENERATING THE INPUT SIGNAL

****

We first generate an input signal by joining sections of band limited signals in a way that ensures that the resulting signal will have very low out of band energy. The method is described in detail in the paper [4] "***Signal interpolation using numerically robust differential operators***".

The segments of the signals *f[i,t]* will be linear combinations of *numsin[i]* many exponentially modified sinusoids, where
minimum ≤ *numsin[i]* ≤ *maximum* ,
with 1 ≤ *minimum* and *maximum* ≤ *detect*≤ *4,* plus separating fragments which are linear combinations of *max = detect + 12* fragments (12 is entirely arbitrary, just to insure *max* >> *detect.)*

The program will detect those segments which are linear combinations of up to *detect* ≤ *4* sinusoids; those with more than *detect* sinusoids will be classified as "transients".
However, the actual number of sinusoidal components in the signal generated can be chosen to be any between *minimum* and *maximum* , where 1 ≤ *minimum* ≤ *maximum* ≤ *detect .*
This is handy for testing of the detection algorithm

There are *number* many such "legitimate" segments of functions interleaved with the same number of "spacing" signals; thus, altogether, there will be
num = 2 number fragments in total, plus "patches" between them, which keep the interpolated signal extremely smooth, i.e., 15 times continuously differntiable.
This guarantees extremely low out of band content; see [4] for details and proofs.

The i-*th* function will have as a support an interval of length *size[i]*, centered at *centerf[i]*, *1* ≤ *i* ≤ *num.* The sizes of the "legitimate fragments", to be detected, can vary randomly between integer lengths *minLength* and *maxLength;* however, for easy comparison with the FFT, we chose *size[i]* to be of even integer length. In this implementation the separating segments are of lenght chosen randomly between 16 and 64, but this is not at all essential; they can be either longer or ommitted, but in this case if two consecutive fragmets come from the same function they can be misclassified as a single segment. To prevet this we include separating fragments which have higher number of components then what the system can detect.

The i-*th* function is a linear combination of *numsin[i]* many exponentially modified sine waves of frequencies *freq[j, i]*, $1 \le j \le$ *numsin[i]*, either damped by a factor $e^{damp[j,i]\,(t-center[i])}$ (if *damp[j,i]* < 0), or expanding (if *damp[j,i]* > 0). The value *damp[i,j]* is in the range - *damping* < *damp[i,j]* < *damping.*

Such maximal value must be chosen so that the signal *f[i,t]* cannot totally vanish or "explode" over the length of its support. The multiplier $e^{damp[j,i]\,(t-center[i])}$ has been centered at the mid point *centerf[i]* of the support interval; for that reason we chose $damping = Min\left[\frac{2\,Log[2]}{maxLength+8}, 0.05\right]$ which ensures that over the half of the length of the longest support (which is approximately one half of maxLength + 8), in either direction from the central point *centerf[i]* the signal at most halves or doubles its max amplitude. This can be altered into trippling by choosing $damping = Min\left[\frac{2\,Log[3]}{maxLength+8}, 0.05\right]$; limit 0.05 comes into play only if maxLength is quite small, to avoid extreme situations.

The frequencies are also chosen randomly from the range minFreq and maxFreq. At the moment the program cannot handle DC, because we do not look at differential operators of odd degrees, necessary to detect fragments which are of the form $e^{damp[j,i]\,(t-center[i])}$ corresponding to an exponentially modified DC. Thus, we limit minFreq to .3 in this implementation. If the segment consists of several frequencies, we order them in a decreasing order for easy book keeping.

The amplitude of *j-th* component of the *i-th* function *f[i,t]* is *A[j, i]*. The j-*th* amplitude is set to *A[j, i]* = *a[j]* = $ampModify^{j-1}$. As a default, *ampModify* = *1*; thus, all components are of the same amplitude, but setting ampModify = *.5* makes the amplitudes decrease by halving as in one of the above examples; setting it to ampModify = 2 cause the amplitudes to double. The fragments are then re-normalized so that all of them have the RMS over their supports equal to 1.

```
In[49]:= (* maximal range of the damping (expanding) factor; chosen so that the signal
          can at most halve/double in amplitude within the support of the signal *)

                          2 Log[2]
         damping = Min[ ─────────────── , 0.05];
                          maxLength + 8
```

```
In[50]:= max = detect + 12;
         (* This is kind of arbitrary, just to make the separating segments look
          different from segments which are sums of up to "detect" many sinusoids. *)

         num = 2 number;
         (* The total number of segments, including the separating ones. *)
```

```
In[52]:= size[0] = 0; size[num + 1] = 0;
         (* For technical reasons we add trivial 0-th and (n+1)-st functions,
         both identically 0. *)

         Do[If[EvenQ[i], size[i] = 2 RandomInteger[{Round[minLength / 2], Round[maxLength / 2]}],
            size[i] = RandomInteger[{16, 64}] ], {i, 1, num}]; (* 4  *)

         (* Thus,
         the "legitimate segments to be detected (with an even index) are between "minLength" and
         "maxLength" long; separating segments (odd
            indices) are 16 - 64 unit (thus Nyquist rate intervals) long.   *)

         Do[
           If[EvenQ[i], numsin[i] = RandomInteger[{minimum, maximum}], numsin[i] = max], {i, 1, num}] ;
         Do[damp[j, i] = If[EvenQ[i], RandomReal[{-damping, damping}], 0], {i, 1, num}, {j, 1, max}];

         (* The "legitimate" fragmenst are linear combinations of
          "minimum" to "maximum" many sinusoides, with  "maximum ≤ detect",
         and with the number of sinusoids chosen randomly for each segment;
         the damping factor for each sinusoid in the legitimate
          fragments is chosen randomly from the interval [-damping, damping];
         the sinewaves in the separating fragments are not damped.  *)

         a[j_] := ampModify^(j-1);
         Do[A[j, 2 i] = a[j], {i, 1, number}, {j, 1, numsin[2 i]}];
         Do[A[j, 2 i + 1] = RandomReal[{.1, 1}], {i, 0, number - 1}, {j, 1, numsin[2 i + 1]}];

         (* the values of amplitudes A[j,i] are chosen so that they decrease
           or increase by the corresponding jth power of the factor "ampModify". If
           ampModify = .5 then the amplitude of the (j+1)th  component
           is one half of the amplitude of the jth  component;
         if ampModify = 2 then the amplitude of the (j+1)th  component
           is twice the amplitude of the jth  component

           The amplitudes can also be randomized, but then one has to worry what
          the "local" signal to noise ratio is for each particular component. *)

         (* We initialize the values of frequencies to zero *)

         Do[freq[j, i] = 0, {j, 1, max}, {i, 1, num}];
```

The particular values of the frequencies are chosen depending on what we want to test. For general purpose tests of the accuracy of frequency estimation we split the range [*minFreq*, *maxFreq*] into *2 numsin[i] - 1* bins, thus depending on the number of components *numsin[i]* to be chosen. We then chose frequencies from the highest to the lowest, by picking a frequency from a bin and then skipping a bin.

Clearly, to test how well the method separates close frequencies we must do that in a different way (as we do in the next file), but the above is good for initial tests.

```
In[60]:= Do[{delta[i] = (maxFreq - minFreq) / (2 numsin[i] - 1);
            Do[freq[j, i] = RandomReal[{minFreq + 2 (numsin[i] - j) delta[i],
                minFreq + (2 (numsin[i] - j) + 1) delta[i]}], {j, 1, numsin[i]}]}, {i, 1, num}];
```

In[61]:=
```
Do[shift[i, j] = RandomReal[{-π, π}], {i, 1, num}, {j, 1, numsin[i]}];
(* We chose randomly the phase shifts of the sinusoidal components. *)
```

We now set spacing between fragments to distance = 16. Note that segments which are linear combinations of max many sine waves will be rejected. Thus the spacing between the "legitimate" fragments is in total 16 + 8 + 16 = 40 Nyquist rate intervals.

In[62]:=
```
distance = 16; NN = distance;
(* This cannot be changed at the moment,
because each value requires a different syththesis filterbank for
   producing the patches between segments of signals. The patching over
   an interval of length "distance = NN = 16" ensures that the resulting
   signal will have continuous derivatives of orders up to NN - 1 = 15.  *)
```

We now calculate the positions of supports of f[i,t]. We start with an interval of length L/2= 32 where the signal will be zero, just to be able to evaluate the chromatic derivatives over the entire length of the signal support using filters with half support L = 64 taps = 32 Nyquist rate intervals.

A spacing of *distance = NN* points precedes the first signal support. The the supports of the signal fragments f[i,t] alternate with a spacing equal to *distance.* At the end of the signal is a final spacing and another interval of length 32 where the signal is set to zero.

Thus, the positions of the centers *centerf[i]* of segments are given by:

In[63]:=

$$\texttt{centerf[0] = 0; centerf[num + 1] = 0;}$$

$$\texttt{Do}\left[\texttt{centerf[i] = L / 2 + distance + }\sum_{\texttt{j=1}}^{\texttt{i-1}}\texttt{(size[j] + distance) + size[i] / 2, \{i, 1, num\}}\right]\texttt{;}$$

We now define segments of band limited signals *f[i, t];* first we calculate some normalization factors:

In[65]:=

$$\texttt{Do}\Big[\texttt{mx[i] =}$$

$$\texttt{RootMeanSquare}\Big[\texttt{Table}\Big[\sum_{\texttt{j=1}}^{\texttt{numsin[i]}}\texttt{A[j, i] E}^{\texttt{damp[j,i] (t-centerf[i])}}\texttt{ Sin[freq[j, i] t + shift[i, j]] /.}$$

$$\texttt{t} \to \texttt{(centerf[i] + u / 2), \{u, -size[i] - 8, size[i] + 8\}}\Big]\Big]\texttt{, \{i, 1, num\}}\Big]\texttt{;}$$

Thus, *mx[i]* is the RMS of the function $\sum_{j=1}^{numsin[i]} A[j,\ i]\ E^{damp[j,i]\ (t-centerf[i])}\ Sin[freq[j,\ i]\ t\ +\ shift[i,\ j]]$ over the support interval *[centerf[i] - size[i]/2, centerf[i] + size[i]/2];* we now use its reciprocal value to normalize *f[i,t]* so that the SNR is equal across all segments.

In[66]:=

$$\texttt{f[0, t\_] = 0; f[num + 1, t\_] = 0; Do}\Big[\texttt{ f[i, t\_] =}$$

$$\frac{\texttt{1}}{\texttt{mx[i]}}\sum_{\texttt{j=1}}^{\texttt{numsin[i]}}\texttt{A[j, i] E}^{\texttt{damp[j,i] (t-centerf[i])}}\texttt{ Sin[freq[j, i] t + shift[i, j]], \{i, 1, num\}}\Big]\texttt{;}$$

These signals will be truncated to their supports of *size[i]*, centered at *centerf[i];*
thus their supports are intervals *[centerf[i] - size[i]/2, centerf[i] + size[i]/2]*. We verify that they are properly normalized:

```
In[67]:= RootMeanSquare[Flatten[Table[f[i, centerf[i] + t / 2], {i, 1, num}, {t, -size[i], size[i]}]]]

Out[67]= 0.986585
```

We also check their maximal amplitude:

```
In[68]:= Max[Flatten[Table[f[i, centerf[i] + t / 2], {i, 1, num}, {t, -size[i], size[i]}]]]

Out[68]= 3.00575
```

We have to interpolate the signal over the interval of length *distance = NN = 16* between two consecutive fragments. This is done so that the values of derivatives of orders up to *NN - 1 = distance -1 = 15* of the interpolation at the right end point of the support of *f[i,t]*, i.e., at the point *centerf[i]+size[i]/2* match the corresponding values of the derivatives of *f[i,t]* at that point, and that the derivatives of the interpolation at the left end point of the support of *f[i+1,t]* i.e., at the point *center[i+1] - size[i+1]/2* match the values of the corresponding derivatives of *f[i+1,t]* at that point. In this way we ensure that the resulting interpolated signal will have continuous derivatives of orders 0 to 15, and this in turn ensures very low energy outside the bandwidth [-π, π], see [4] .

Such interpolation is obtained by evaluating the chromatic derivatives of the signals *f[i,t]* using the filterbank W for differentiation of clean signals for orders 0 - 15. The vector of values of the chromatic derivatives of *f[i,t]* at the left end point *centerf[i] - size[i]/2* of the support of *f[i,t]* is denoted by *SigL[i]* and the vector of values of the chromatic derivatives of *f[i,t]* at the right end point *centerf[i] + size[i]/2* of the support of *f[i,t]* is denoted by *SigR[i];* thus *SigL[i]* and *SigR[i]* are vectors of length 16 (orders 0 - 15). To obtain the values of these derivatives we do not need any analytic differentiation; instead we find the appropriate samples of *f[i,t]* and apply the filters *W*:

```
In[69]:= SAMPL = Table[f[i, centerf[i] - size[i] / 2 + j / 2], {j, -LL, LL}, {i, 0, num + 1}];
         SAMPR = Table[f[i, centerf[i] + size[i] / 2 + j / 2], {j, -LL, LL}, {i, 0, num + 1}];
         SigL = Transpose[W.SAMPL]; SigR = Transpose[W.SAMPR];
```

We can now compare the values *SigL[i]* obtained using numerical differentiation via our FIR filterbank *W* with the analytically computed values *SigLA[i]:*

```
In[71]:=  SigLA = Table[KC[f, j, i, t] /. t → (centerf[i] - size[i] / 2) , {i, 0, num + 1}, {j, 0, NN - 1}];
         SigRA = Table[KC[f, j, i, t] /. t → (centerf[i] + size[i] / 2) , {i, 0, num + 1}, {j, 0, NN - 1}];
         (* The values of the chromatic derivatives obtained by analytic differentiation *)

In[72]:= ED = Transpose[Join[Abs[SigL - SigLA], Abs[SigR - SigRA]]];
```

Below is the table of the **RMS values of the chromatic derivatives of the signal, RMS values of the error of the chromatic derivatives, SNR in db,** as well as the **mean, median** and **maximal** values of the errors of the derivatives of orders 0 - 15, showing that our filters are extremely accurate:

```
In[73]:= Grid[Join[Transpose[{{"degree"}, {"RMS of CD"}, {"RMS error"},
             {"SNR db"}, {"Mean error"}, {"Median error"}, {"Max error"}}],
           Table[{j, RootMeanSquare[Transpose[SigLA][[j + 1]]], RootMeanSquare[ED[[j + 1]]],
                          RootMeanSquare[Transpose[SigLA][[j + 1]]]
              20 Log[10, ─────────────────────────────────────────], Mean[ED[[j + 1]]],
                             RootMeanSquare[ED[[j + 1]]]
              Median[ED[[j + 1]]], Max[ED[[j + 1]]]}, {j, 0, NN - 1}]], Frame → All]
```

Out[73]=

degree	RMS of CD	RMS error	SNR db	Mean error	Median error	Max error
0	0.894518	5.9492×10^{-6}	103.543	4.47032×10^{-6}	3.10615×10^{-6}	0.0000193719
1	1.12534	0.000011238	100.012	8.15741×10^{-6}	5.79369×10^{-6}	0.0000423444
2	1.00353	0.0000173372	95.251	0.0000130165	9.06913×10^{-6}	0.0000559452
3	1.01959	0.000026207	91.8001	0.0000190002	0.0000134186	0.0000982268
4	0.864756	0.0000387115	86.9811	0.0000289895	0.0000208017	0.000122529
5	0.926997	0.0000599836	83.7809	0.0000434088	0.000030668	0.000223296
6	0.940109	0.0000869737	80.6758	0.0000647696	0.0000486428	0.000269819
7	0.95365	0.000138001	76.7901	0.0000996136	0.0000708931	0.000509538
8	1.05567	0.000194589	74.6882	0.000144013	0.000108307	0.000601609
9	1.06035	0.000314545	70.5554	0.000226345	0.000156132	0.0011491
10	0.862971	0.000428978	66.0712	0.000316017	0.000242214	0.00133197
11	0.923778	0.000548208	64.5324	0.0004225	0.000378164	0.00194787
12	0.915158	0.000943864	59.7317	0.00070811	0.000507942	0.00291552
13	1.14805	0.00108272	60.5088	0.000831803	0.000730093	0.00369425
14	1.0489	0.00191329	54.7791	0.00143347	0.00103631	0.00605803
15	0.971584	0.00205306	53.5016	0.00157323	0.00141556	0.00697388

We now download synthesis filters for the interpolation ("patching"); the coefficients are again stored as integers so we re-normalize them by dividing them with 10^{15}:

```
In[74]:= SetDirectory["c:/ChromaticDerivatives/filterbanks/patch"];
```

```
In[75]:= Patch = {}; Do[{filename = StringJoin["patch_", IntegerString[dg], ".txt"];
         Patch = Join[Patch, {Flatten[Import[filename, "Table"]]}]}, {dg, 0, 15}];
         Patch = N[Patch / 10^15];
```

We use the values from the *Patch* file to generate the values of the interpolation functions for left side and right side of the interpolation "patch", see [4]:

```
In[77]:= PatchL = Take[Transpose[Patch], {2, 32}];
         PatchR = Table[(-1)^m Patch[[m + 1, 33 - t]], {t, 1, 31}, {m, 0, NN - 1}];
```

Interpolation between functions *f[i,t]* and *f[i+1,t]* is obtained at every half integer (because we are producing twice oversampled signal) using the synthesis filters, as
PatchL **.** *SigRA[i - 1] + PatchR* **.** *SigL[i]*; see [4] for explanation. Essentially, we represent the interpolation over the interval $[t_L, t_R]$ between the supports of f_i and f_{i+1} as
$\sum_{j=0}^{15} K^j[f_i](t_L) B_j(t-t_L) + \sum_{j=0}^{15} K^j[f_j](t_R)(-1)^j B_j(t_R-t)$, where B_j(t) satisfy $K^j[B_m](0) = \delta\,(j-m)$ and $K^j[B_m](distance) = 0.$

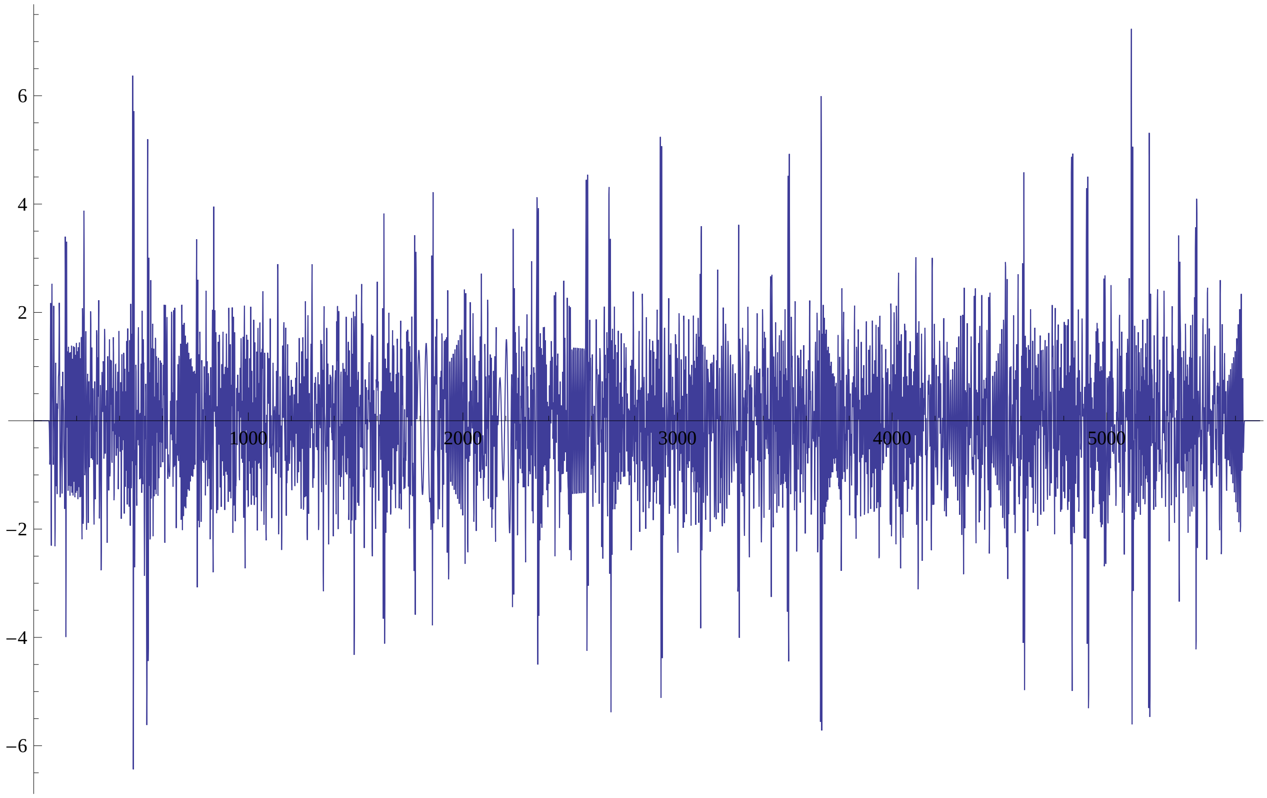
Finally, we can create our test signal:

```
In[79]:= Signal =
           Join[Table[0, {k, 1, L + 1}], Flatten[Table[Join[PatchL.SigR[[i]] + PatchR.SigL[[i + 1]],
               Table[f[i, centerf[i] + t / 2], {t, -size[i], size[i]}]], {i, 1, num}] ],
            PatchL.SigR[[num + 1]] + PatchR.SigL[[num + 2]], Table[0, {k, 1, L}]];
```

We now check the max value of the interpolated signal and plot its values:

```
In[80]:= Max[Signal]

Out[80]= 7.23493

In[81]:= length = Dimensions[Signal][[1]];

In[82]:= ListPlot[Signal, Joined → True, PlotRange → All, ImageSize → 600]
```

Out[82]=



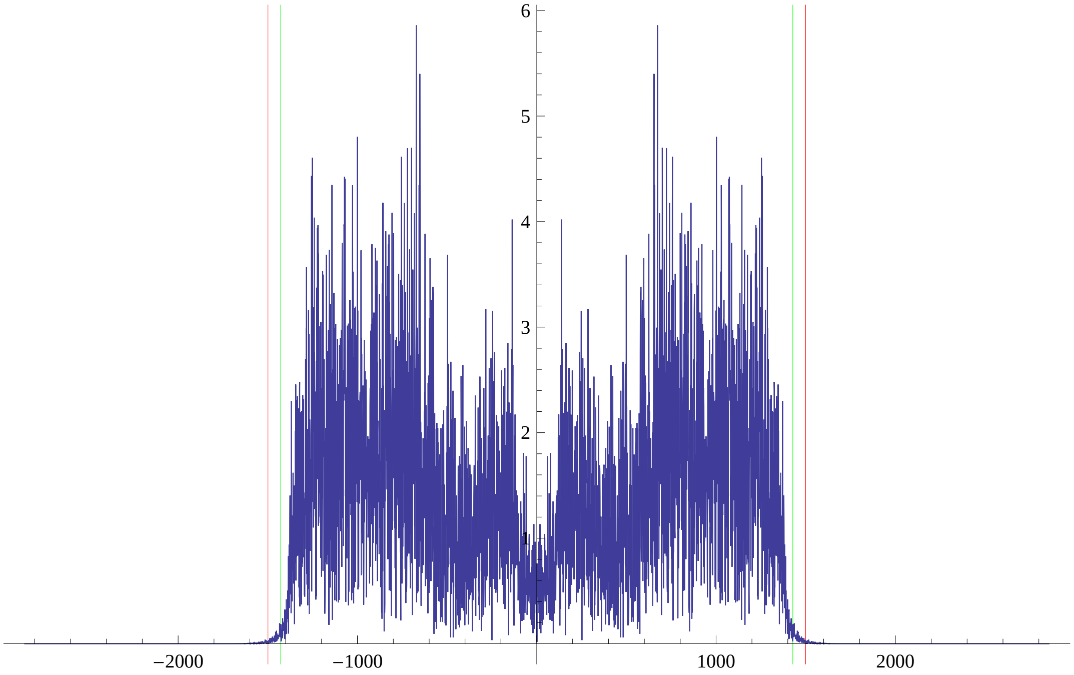
To check its out of band content, make sure you ran the program with *number* = 100 or larger so that the FFT has a sufficient resolution to represent extremely small out of band content accurately; the green grid lines are set at the ±π band limit and the red gridlines at ±1.05π:

```
In[83]:= FT1 = Abs[Fourier[Signal]];
         df = Dimensions[FT1][[1]];
         FT = Join[Table[{i - Round[df / 2], FT1[[df / 2 + i]]}, {i, 1, Round[df / 2]}],
             Table[{i, FT1[[i]]}, {i, 1, Round[df / 2]}]];
```

In[86]:=

```
ListPlot[FT, Joined → True, PlotRange → All,
 GridLines → {{{Round[-1.05 df / 4], Red}, {Round[-df / 4], Green},
    {Round[df / 4], Green}, {Round[1.05 df / 4], Red}}, {}}, ImageSize → 500]
```

Out[86]=



We now compute the fraction of energy of the signal outside the bandwidth [-π, π], and the fraction of energy outside [-1.05π, 1.05π]

In[87]:=
```
Grid[Join[{{"out of [-π,π] energy/total signal energy", "in db"}},
  {{Sum[FT1[[i]]^2, {i, Round[df / 4], Round[3 df / 4]}] / Sum[FT1[[i]]^2, {i, 1, df}],
    10 Log[10 , Sum[FT1[[i]]^2, {i, Round[df / 4], Round[3 df / 4]}] / Sum[FT1[[i]]^2, {i, 1, df}]]},
   {"*************************************************", "*******"}},
  {{"out of [-1.05π, 1.05π] energy/total signal energy", "in db"}},
  {{Sum[FT1[[i]]^2, {i, Round[1.05 df / 4], Round[.95 × 3 df / 4]}] / Sum[FT1[[i]]^2, {i, 1, df}],
    10 Log[10 , Sum[FT1[[i]]^2, {i, Round[1.05 df / 4], Round[.95 × 3 df / 4]}] / Sum[FT1[[i]]^2, {i, 1, df}]]}}], Frame → All]
```

Out[87]=
```
  out of [-π,π] energy/total signal energy            | in db
                0.0000823258                         | -40.8446
************************************************* | *******
  out of [-1.05π, 1.05π] energy/total signal energy | in db
                1.775 × 10^-6                        | -57.508
```

Thus, the energy out of [-1.05 π, 1.05 π] bandwidth is about or less than 1/10 of the energy which is out of [-π, π] bandwidth, showing that out of band energy decays extremely fast; for all practical purposes, our interpolated signal is [-1.05 π , 1.05 π] band limited. For this reason, as we will see below, our chromatic derivative filters with a pass band [-1.05 π , 1.05 π] are extremely accurate when applied to signals produced by our "smooth patching" of pieces of unrelated band limited signals.

We now corrupt signal with the white Gaussian noise; NDB sets signal to noise ratio in db; we compute the RMS of the signal and of the noise generated to get the multiplicative scaling constant noise which sets the signal to noise ratio to exactly the prescribed *NDB* value in db.

In[88]:=
```
NOISE = Table[RandomReal[NormalDistribution[0, 1]], {k, 1, length}];
```

In[89]:=
```
noise = RootMeanSquare[Signal] / RootMeanSquare[NOISE] 10^(-NDB/20)
```

Out[89]=
```
0.03904
```

We compare SNR with the signal to noise ratio for the [-π,π] signal component and [-π,π] noise component; the later is **the relevant signal to noise ratio** for our experiments.

In[90]:=
```
FTN = Abs[Fourier[noise NOISE]];
```

In[91]:=
```
SN = Sum[FT1[[i]]^2, {i, 1, df}] / Sum[FTN[[i]]^2, {i, 1, df}];

InbandSN = (Sum[FT1[[i]]^2, {i, 1, Round[df / 4]}] + Sum[FT1[[i]]^2, {i, Round[3 df / 4], df}]) /
   (Sum[FTN[[i]]^2, {i, 1, Round[df / 4]}] + Sum[FTN[[i]]^2, {i, Round[3 df / 4], df}]);
```

```
In[92]:= Grid[Join[{{"SNR in db", "[-π,π] signal/[-π,π] noise in db"}},
            {{10 Log[10 , SN], 10 Log[10 , InbandSN]}}], Frame → All]
```

Out[92]=

SNR in db	[-π,π] signal/[-π,π] noise in db
30.	32.8573

We now apply the filterbank to the signal + noise samples:

```
In[93]:= SAMP = Table[Signal[[p + j]] + noise NOISE[[p + j]], {j, -L, L}, {p, L + 1, length - L}] ;

In[94]:= DER1 = Transpose[U.SAMP];
```

We now check the accuracy of differentiation using filters for noisy inputs. Thus, we also apply the filterbanks to the signal and the noise separately.

```
In[95]:= SAMPS = Table[Signal[[p + j]], {j, -L, L}, {p, L + 1, length - L}] ;
         DERS = U.SAMPS;
         SAMPN = Table[noise NOISE[[p + j]], {j, -L, L}, {p, L + 1, length - L}] ; DERN = U.SAMPN;
```

We now collect the values of the chromatic derivatives of orders 0 - 8 of the signal **without any noise** over the support intervals we will patch together:

```
In[98]:= Do[DSig[j] = Flatten[Table[Table[DERS[[1 + j, 1 + 2 centerf[i] - L + p]],
               {p, -size[i], size[i] - 1}], {i, 1, num}]], {j, 0, 8}];
```

We compare these values with the analytically obtained values of the chromatic derivatives of the signal pieces *f[i, t]:*

```
In[99]:= Do[DSigA[j] =
            Flatten[Table[Table[KN[f, j, i, t] /. t → (centerf[i] + p / 2),
               {p, -size[i], size[i] - 1}], {i, 1, num}]], {j, 0, 8}];
```

The table below gives the RMS value of each chromatic derivative of the signal over these supports, the RMS value of the error of the filterbanks compared to the true values as obtained from the analytic expressions for *f[i, t]*, the corresponding SNR in db, as well as the mean error, median error and the maximal error. Note how extraordinarily accurate are the values obtained using our filterbank U.

In[100]:=

```
Grid[Join[Transpose[{{"degree"}, {"RMS"}, {"RMS error"},
    {"SNR db"}, {"Mean error"}, {"Median error"}, {"Max error"}}],
  Table[{j, RootMeanSquare[DSigA[j]], RootMeanSquare[DSigA[j] - DSig[j]],
    20 Log[10, RootMeanSquare[DSigA[j]] / RootMeanSquare[DSigA[j] - DSig[j]]], Mean[Abs[DSig[j] - DSigA[j]]],
    Median[Abs[DSig[j] - DSigA[j]]], Max[Abs[DSig[j] - DSigA[j]]]}, {j, 0, 8}]], Frame → All]
```

Out[100]=

degree	RMS	RMS error	SNR db	Mean error	Median error	Max error
0	0.983596	0.000758227	62.2603	0.000585635	0.00048	0.00411871
1	0.971168	0.00133028	57.267	0.0010257	0.000814942	0.00695157
2	0.788108	0.0017128	53.2576	0.0013225	0.00106847	0.00939617
3	0.787179	0.00207539	51.5795	0.00159841	0.00125986	0.0111335
4	0.854329	0.00234488	51.2301	0.00181075	0.00145043	0.0130665
5	0.972509	0.00268381	51.1828	0.00206022	0.00163537	0.0149841
6	0.941073	0.00289039	50.2533	0.00222952	0.00179186	0.0159978
7	0.881443	0.00324834	48.6707	0.00248107	0.00195836	0.0187386
8	0.835506	0.00342896	47.7357	0.0026182	0.00211175	0.0226674

In[101]:=

```
(*  Note that the supports of the signals are only minLenght intervals long:  *)
minLength
```

Out[101]=

```
24
```

while the supports of the differentiation filters are 64 intervals long (129 taps, spaced two per unit interval), **Thus, filters "see" values far outside of the support of the function being differentiated. However, the smoothness of the interpolated signal ensures that even the high order derivatives are remarkably accurate!!**

We now look at the impact of the noise on the accuracy of differentiation. We fist find the SNR of the in-band signal versus in-band noise component:

In[102]:=

```
(* in-band signal to in-
 band noise ratio compared to the entire signal to the entire noise ratio  *)
Print[Grid[{{"signal/noise", "in-band signal/in-band noise",
     "in-band signal/in-band noise as obtained by filters"},
    {20 Log[10, RootMeanSquare[Signal] / RootMeanSquare[noise NOISE]], 10 Log[10, InbandSN],
     20 Log[10, RootMeanSquare[DERS[[1]]] / RootMeanSquare[DERN[[1]]]]}}, Frame → All]];
```

signal/noise	in-band signal/in-band noise	in-band signal/in-band noise as obtained by filters
30.	32.8573	32.602

This is consistent with the fact that the signal has very little out of band energy, while half the energy of the noise is out of band; thus in-band signal to in-band noise ratio should be about 3 db higher than total signal to total noise ratio.

Thus, below is a very good estimate of the signal to noise ratios for all derivatives of order up to eight:

```
In[103]:= Print[Grid[Transpose[Join[{{" degree", "in-band SNR db"}},
            Table[{j, 20 Log[10, RootMeanSquare[DERS[[1 + j]]] / RootMeanSquare[DERN[[1 + j]]]]}, {j, 0, 8}]]], Frame → All]];
```

```
degree         | 0      | 1       | 2       | 3       | 4       | 5       | 6       | 7       | 8
in-band SNR db | 32.602 | 32.4313 | 29.6974 | 31.4902 | 32.3246 | 32.2574 | 32.5751 | 31.7862 | 31.8915
```

Thus, the SNR of all chromatic derivatives are essentially the same and equal to the *in-band signal* to *in-band noise* ratio! This empirically demonstrates extraordinary accuracy and noise robustness of evaluation of chromatic derivatives using a FIR filterbank.

****

****

```
In[104]:= Signal1 = Take[Signal, {L + 1, Dimensions[Signal][[1]] - L}];

In[105]:= length1 = Dimensions[Signal1][[1]]; dist1 = L;
```

FREQENCY ESTIMATION

The algorithm below uses only the noisy signal samples, and has no access to any other signal parameters, not even the signal to noise ratio, which is estimated from the signal samples.

```
In[106]:=
      Do[{Prod1[k, m] = Table[DER1[[i]][[k + 1]] DER1[[i]][[m + 1]], {i, 1, length1}];
         Prod1[m, k] = Prod1[k, m];}, {m, 0, 2 detect}, {k, 0, m}];
      Do[{Quiet[ Tble = RecurrenceTable[{CO[i] == CO[i - 1] -
                  Prod1[k, m][[INT[detect] + i - 1 - INT[q] ]] + Prod1[k, m][[INT[detect] + i + INT[q]]],
                CO[1] == Sum[Prod1[k, m][[INT[detect] + 1 + j]], {j, -INT[q], INT[q]}]}, CO,
              {i, 1, length1 - 2 INT[detect]}];]; crl2[q, k, m] = Tble;
         crl2[q, m, k] = Tble; Clear[Tble];} , {q, 1, detect}, {m, 0, 2 q}, {k, 0, m}];

In[108]:= Signal2 = Take[Signal1, {INT[detect] + 1, Dimensions[Signal1][[1]] - INT[detect]}];
          length2 = Dimensions[Signal2][[1]]; dist2 = dist1 + INT[detect];
```

We now for each point *i* in the domain of the input signal evaluate the smallest eigenvalues λ_j of the matrices [crl[i, k, m]: 0 ≤ k ≤2 j, 0 ≤ m ≤ 2 j].

```
In[110]:= Do[eigenV2[j] =
             Table[√(1 / (2 INT[j] + 1) Chop[Eigenvalues[Table[crl2[j, k, m][[i]], {k, 0, 2 j},
                      {m, 0, 2 j}], -1][[1]]]), {i, 1, length2}], {j, 1, detect}];
```

We filter out some noise by replacing the calculated value eigV[j, i] with the median of these values over the interval [i-cl, i+cl]. (This is not essential but it appears to improve accuracy).

```
In[111]:= cl = 8;

In[112]:= Do[eigenVF3[j] = Table[Median[Table[eigenV2[j][[i + k]], {k, -cl, cl}]],
             {i, cl + 1, length2 - cl}], {j, 1, detect}];
```

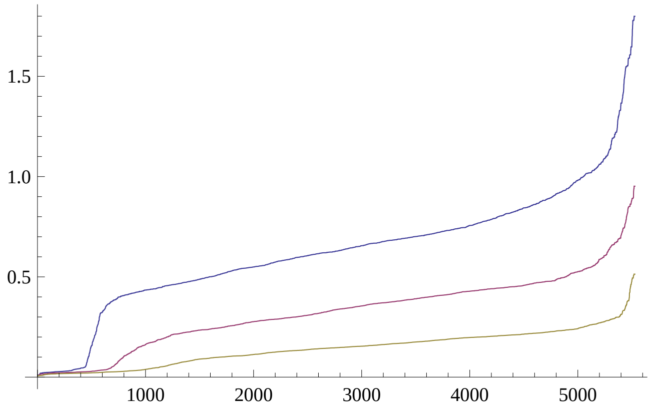
We shorten the signal again:

```
In[113]:= Signal3 = Take[Signal2, {cl + 1, length2 - cl}];

In[114]:= length3 = Dimensions[Signal3][[1]]; dist3 = dist2 + cl;
```

We now approximately determine the signal to noise (SNR). The plot below shows the distribution of the smallest eigenvalues; the RMS of the noise is the RMS of the flat part of the low end of the graph, approximated bt the variable *cut:* ww[[1]] is the minimal number of sinusoidal components present

```
In[115]:= Do[mini[j] = Min[eigenVF3[j]], {j, 1, detect}];

In[116]:= Do[selm[j] = Median[Select[eigenVF3[j], # < 2 mini[j] &]], {j, 1, detect}];

In[117]:= Do[sorted3[j] = Sort[eigenVF3[j]], {j, 1, detect}];

In[118]:= s = Table[selm[j], {j, 1, detect}];

In[119]:= ww = SortBy[Select[Table[{j, selm[j]}, {j, 1, detect}], #[[2]] < 2 Min[s] &], #[[1]]][[1]];

In[120]:= Table[mini[j], {j, detect}]

Out[120]= {0.0105144, 0.008215, 0.00731686}

In[121]:= Do[sorted[j] = Sort[eigenVF3[j]], {j, 1, detect}];

In[122]:= ListPlot[Table[sorted3[j], {j, 1, detect}], Joined → True]
```

Out[122]=



```
In[123]:= cut  = RootMeanSquare[Select[sorted3[ww[[1]]], # < 2 ww[[2]]  &]]

Out[123]= 0.0252497

In[124]:= ww[[1]]

Out[124]= 1
```

comparing the RMS of the inband comonent of the noise `noise` $/\sqrt{2}$ and the approximated value *cut* as established by the algorithm:

```
In[125]:= {cut , noise/√2}

Out[125]= {0.0252497, 0.0276054}

In[126]:= Nums = Join[Table[0, {i, 1, L + 1}],
             Flatten[Table[Join[Table[0, {i, 1, 31}], Table[Sign[max - numsin[i]] numsin[i],
                 {t, -size[i], size[i]}]], {i, 1, num}]], Table[0, {i, 1, 31}], Table[0, {k, 1, L}]];

In[127]:= Nums3 = Take[Nums, {dist3 + 1, length - dist3}];

In[128]:= nf3 = Table[0, {i, 1, length3}];

In[129]:= Do[{p = 1; nf3[[i]] = p; While[p < detect + 1 && eigenVF3[p][[i]] ≥ treshold cut,
              {nf3[[i]] = Sign[detect - p] (p + 1); p++;}]}, {i, 1, length3}];
```

We now for each point *i* in the domain of the input signal evaluate the smallest eigenvalues λ_j of the matrices [COR[i, k, m]: $0 \le k \le 2 j$, $0 \le m \le 2 j$].
The code below is wasteful evaluating some unnecessary things, but it provides plots of various quantities which are helpful to understand the behavior of the algorithm.

nf3[[i]] is the number of components at a point *i* : We now for each point *i* in the domain of the input signal evaluate the smallest eigenvalues λ_j of the matrices [COR[i, k, m]: $0 \le k \le 2 j$, $0 \le m \le 2 j$].

For each point *i* we look for the smallest value of $j \le detect$ such that $eigV[j, i] = \sqrt{\lambda_j / (2\, INT[j] + 1)} < treshold\ noise$; variable nf[i] gives the number of sinusoidal components present over the interval [i-INT, i+INT].

NOTE: This is really not the right way of doing it; instead of comparing the eigenvalues with a fixed treshold value, the decision should be made by comparing the eigenvalues of matrices of several orders. I'll post such algorithm after I return from my trip, sometimes in early August.

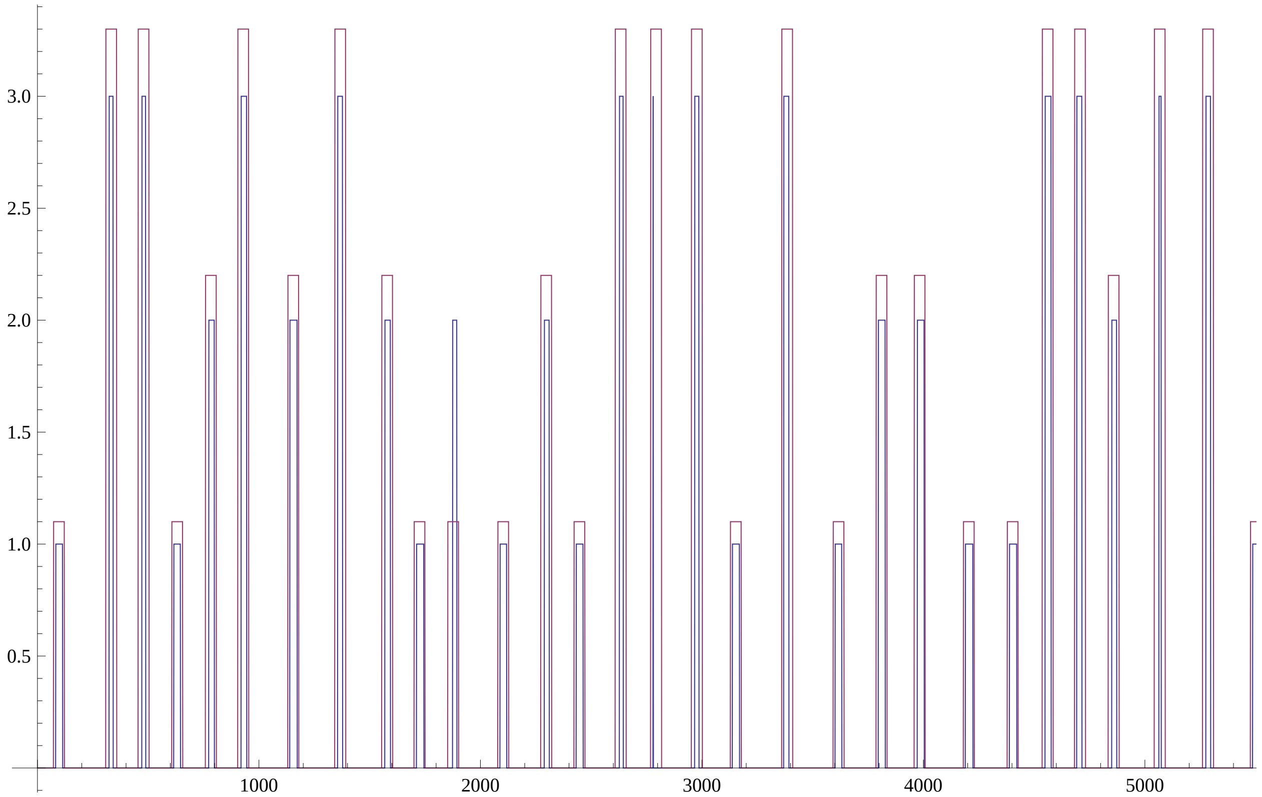
To avoid errors present at the ends of the supports of the functions we take only those i which lie inside detected intervals at least guard many sampling points:

```
In[130]:= nfc3 = Table[0, {i, 1, length3}];

In[131]:= jj = 1; kk = jj; left = {}; right = {}; nc = {};
          While[kk < length3, { While[nf3[[kk]] == 0 && kk < length3, kk++]; xx = kk;
            yy = kk; While[nf3[[yy]] ≠ 0 && yy < length3, yy++]; zz = Round[1 / 2 (xx + yy)];
            pp = 0; qq = 0; While[zz - pp > xx && nf3[[zz - (pp + 1)]] == nf3[[zz]], pp++];
            While[zz + qq < yy && nf3[[zz + 1]] == nf3[[zz]], qq++];
            If[pp ≥ guard && qq ≥ guard, {Do[nfc3[[zz - r]] = nf3[[zz]], {r, 0, pp - guard}];
              Do[nfc3[[zz + r]] = nf3[[zz]], {r, 1, qq - guard}]; left = Append[left, zz - (pp - guard)];
              right = Append[right, zz + (qq - guard)]; nc = Append[nc, nf3[[zz]]];}]; kk = yy;}];

In[132]:= Do[LIMIT[j] = Table[treshold cut, {i, 1, length3}], {j, 1, detect}]
```

The plot below compares the number of components as established by the algorithm (nfc3) and the true number *Nums* obtained from the definition of the signal:

In[133]:=
```
ListPlot[{nfc3, 1.1 Nums3}, Joined → True, PlotRange → All, ImageSize → 600]
```

Out[133]=

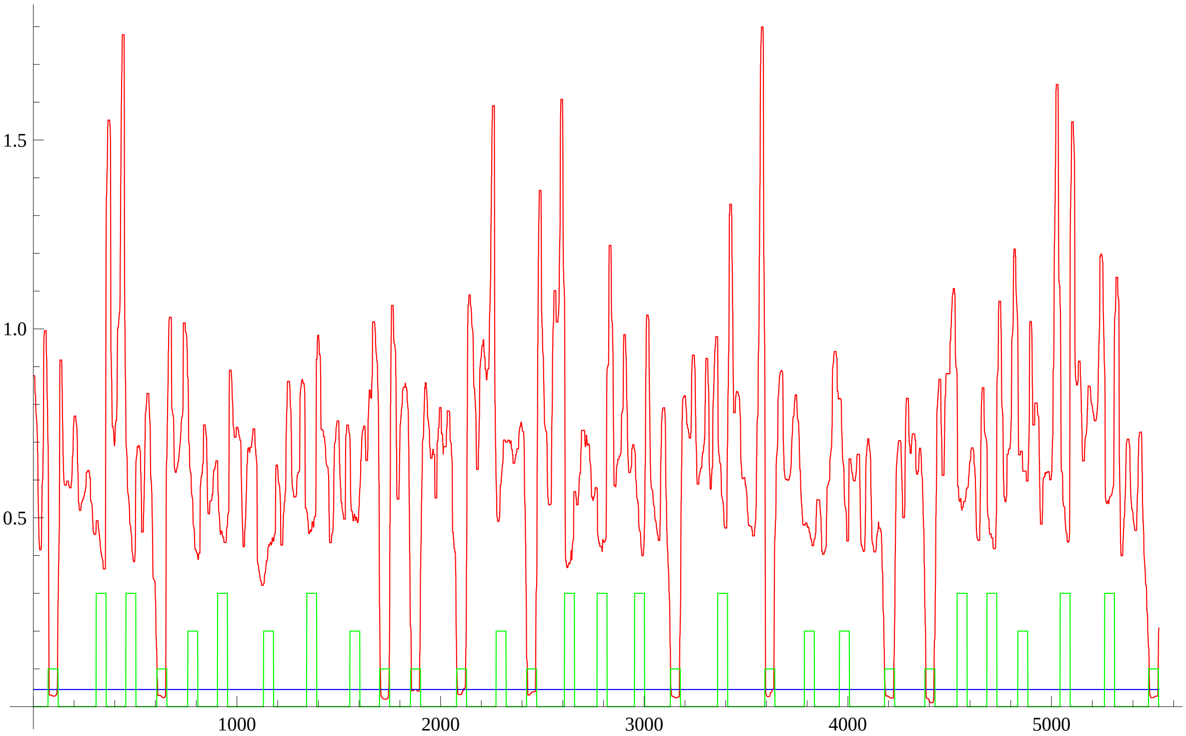


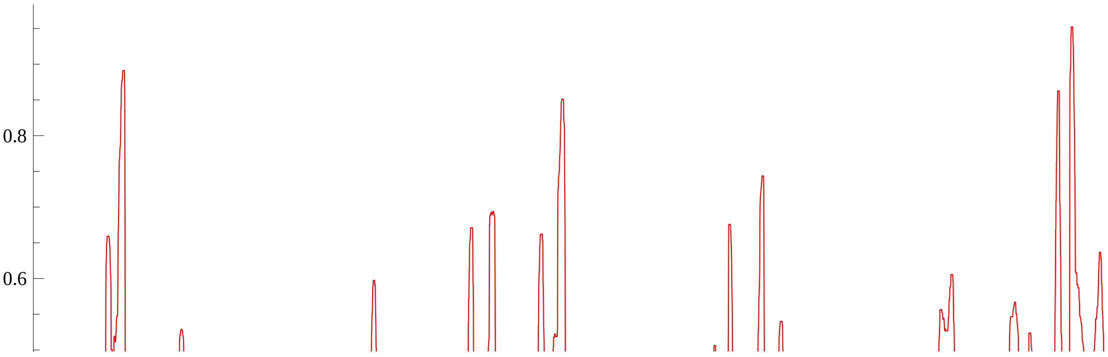
Plotting the smallest eigenvalues of the matrices [COR[i, k, m]: 0 ≤ k ≤ 2 j, 0 ≤ m ≤ 2 j] and comparing them with the treshold value *treshold cut;* regions where the minimal eigenvalues dip bellow the value *treshold cut* correspond to intervals detected by the algorithm, and the smallest rank of the matrix [COR[i, k, m]: 0 ≤ k ≤ 2 j, 0 ≤ m ≤ 2 j] for which this happens determines the number of components, because it is equal to

In[134]:=
```
NC = Table[numsin[2 j], {j, 1, number}];
LEFT[add_] := Table[-add + L + 1 + 31 + 2 size[1] + 1 +
                        k-1
        31 + 1 +  ∑  (2 size[2 j] + 1 + 31 + 2 size[2 j + 1] + 1 + 31), {k, 1, number}];
                        j=1
RIGHT[add_] := Table[add + L + 1 + 31 + 2 size[1] + 1 + 31 + 2 size[2 k] + 1 +
        k-1
         ∑  (2 size[2 j] + 1 + 31 + 2 size[2 j + 1] + 1 + 31), {k, 1, number}];
        j=1
FREQ = Table[Table[freq[j, 2 k], {j, 1, numsin[2 k]}], {k, 1, number}];
DAMP = Table[Table[damp[j, 2 k], {j, 1, numsin[2 k]}], {k, 1, number}];
```

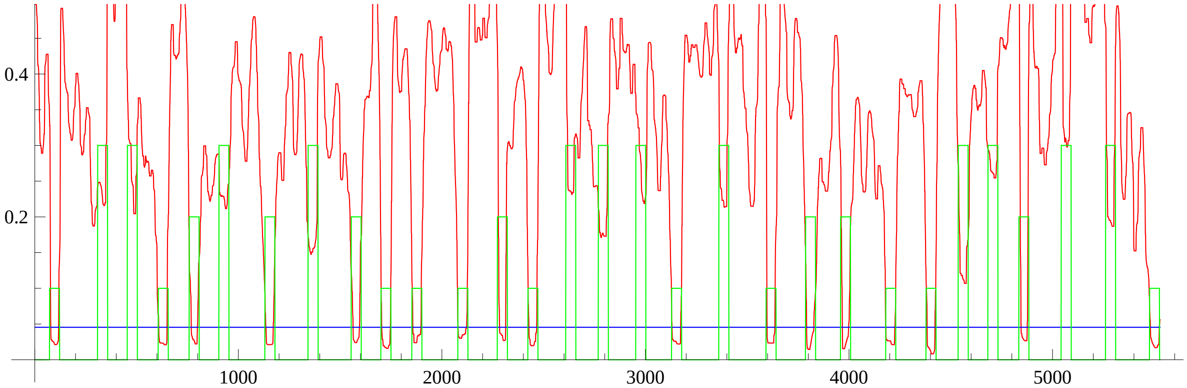
In[139]:=

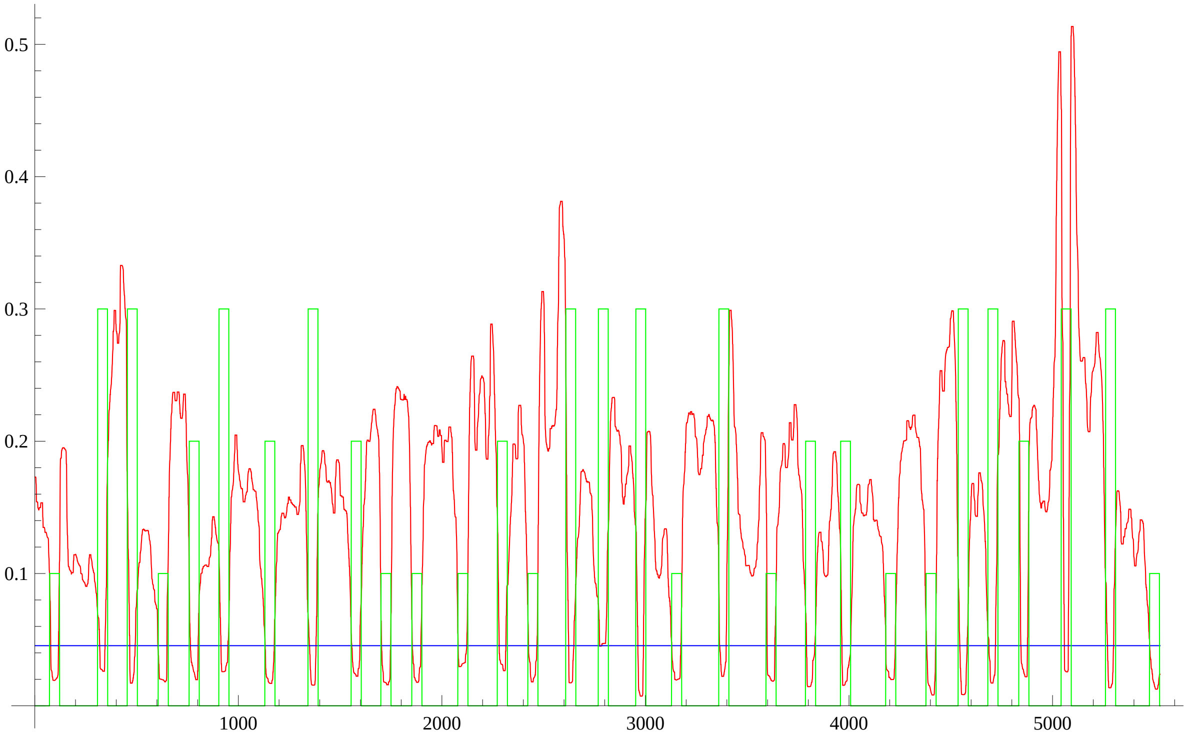
In[140]:=
```
Do[ PLT1[j] = ListPlot[{eigenVF3[j], LIMIT[j], .1 Nums3},
   PlotStyle → {Red, Blue, Green, Orange}, Joined → True], {j, 1, detect}];
GraphicsColumn[Table[PLT1[j], {j, 1, detect}], ImageSize → 600]
```


1.5
1.0
0.5
1000
2000
3000
4000
5000


0.8
0.6

Out[140]=





We now find the eigenvector which corresponds to the appropriate minimal eigenvalue, see paper [1] for the details

```
In[141]:= MR3 = Table[2 detect + 1, {i, 1, length3}];

In[142]:= Do[MR3[[i]] = 2 nfc3[[i]] + (2 detect + 1) (1 - Sign[nfc3[[i]]]), {i, 1, length3}];

In[143]:= P3 = Table[0, {i, 1, length3}];

In[144]:= Do[P3[[i]] = If[MR3[[i]] < 2 detect + 1, Table[crl2[nfc3[[i]], k, m][[i + cl]],
            {k, 0, Min[MR3[[i]], 2 detect]}, {m, 0, Min[MR3[[i]], 2 detect]}]], {i, 1, length3}];

In[145]:= F3 = G3 = SolZ3 = solB3 = TZB3 = TZD3 = Table[0, {i, 1, length3}];

In[146]:= Do[If[MR3[[i]] < 2 detect + 1,
           F3[[i]] = Chop[Eigenvectors[P3[[i]], -1]][[1]]], {i, 1, length3}];

In[147]:= Do[If[MR3[[i]] < 2 detect + 1, G3[[i]] =
            Table[X[k] → F3[[i]][[k + 1]], {k, 0, Min[{MR3[[i]], 2 detect}]}]], {i, 1, length3}];
```

```
In[148]:= im[t_] = Im[t];
```

Finally, to retrieve the frequencies we solve the corresponding algebraic equation and isolate the imaginary parts (frequencies) and the real parts (damping factors):

```
In[149]:= Do[If[MR3[[i]] < 2 detect + 1,
                            MR3[[i]]
             {SolZ3[[i]] = Solve[   ∑   X[k] N[(-I)^k TN[k, -I z]] == 0 /. G3[[i]], z];
                                   k=0
              solB3[[i]] = Table[z /. SolZ3[[i]][[j]], {j, 1, Dimensions[SolZ3[[i]]][[1]]}];
              TZB3[[i]] = Table[-Im[SortBy[solB3[[i]], im]][[j]],
                 {j, 1, Round[Dimensions[solB3[[i]]][[1]] / 2]}];
              TZD3[[i]] = Table[Re[SortBy[solB3[[i]], im]][[j]],
                 {j, 1, Round[Dimensions[solB3[[i]]][[1]] / 2]}]; },
             { TZB3[[i]] = Table[{0}, {m, 1, detect}]; TZD3[[i]] = Table[{0}, {m, 1, detect}]}], {i,
             1, length3}];
```

```
In[150]:= NFC3 = Table[If[nfc3[[i]] - j ≥ 0, 1, 0], {j, 1, detect}, {i, 1, length3}];
```

```
In[151]:= Do[FREQQ3[j] = Flatten[Flatten[
               Table[If[nfc3[[i]] - j ≥ 0, TZB3[[i]][[j]], 0], {i, 1, length3}]]], {j, 1, detect}];
```

```
In[152]:= Do[
           DMP3[j] = Flatten[Flatten[Table[If[nfc3[[i]] - j ≥ 0, TZD3[[i]][[j]], 0], {i, 1, length3}]]],
           {j, 1, detect}];
```

```
In[153]:= Do[FREQQ3[j] = FREQQ3[j] NFC3[[j]], {j, 1, detect}];
          Do[DMP3[j] = DMP3[j] NFC3[[j]], {j, 1, detect}];
```

The rest of the code is just for counting the hits, misses of itervals sought, the number of itervals which got split or merged together and intervals over which a large error of frequency of estimation was made.

```
In[155]:= Do[Markers[w] = Join[Table[0, {i, 1, L + 1}], Flatten[Table[Join[Table[0, {i, 1, 31}],
                 Table[Sign[max - numsin[i]] freq[w, i], {t, -size[i], size[i]}]], {i, 1, num}]],
              Table[0, {i, 1, 31}], Table[0, {k, 1, L}]], {w, 1, max - 1}];
```

```
In[156]:= Do[Markers3[w] = Take[Markers[w], {dist3 + 1, length - dist3}], {w, 1, max - 1}];
```

```
In[157]:= Do[DMarkers[w] = Join[Table[0, {i, 1, L + 1}], Flatten[Table[Join[Table[0, {i, 1, 31}],
                 Table[Sign[max - numsin[i]] damp[w, i], {t, -size[i], size[i]}]], {i, 1, num}]],
              Table[0, {i, 1, 31}], Table[0, {k, 1, L}]], {w, 1, detect}];
```

```
In[158]:= Do[DMarkers3[w] = Take[DMarkers[w], {dist3 + 1, length - dist3}], {w, 1, detect}];
```

```
In[159]:= dmp3 = fr3 = Table[0, {k, 1, detect}, {i, 1, length3}];
```

```
In[160]:= Do[NFC3[[k, 1]] = 0, {k, 1, detect}];
```

```
In[161]:= Do[{x = {}; s = {};
             Do[{If[NFC3[[k, i]] == 1, {If[NFC3[[k, i - 1]] == 0, y = i]; x = Append[x, FREQQ3[k][[i]]];
                 s = Append[s, DMP3[k][[i]]]; If[NFC3[[k, i + 1]] == 0, {
                   m = Median[x]; n = Median[s];
                   Do[{fr3[[k, j]] = m; dmp3[[k, j]] = n}, {j, y, i}]; x = {}; s = {};}]},
                {fr3[[k, i]] = 0; dmp3[[k, i]] = 0;}]}, {i, 2, length3 - 1}]}, {k, 1, detect}];
```

```
In[162]:= left3 = Join[left, {∞}]; right3 = Join[right, {∞}]; LEFT3 = Join[LEFT[0] - dist3, {∞}];
          RIGHT3 = Join[RIGHT[0] - dist3, {∞}]; nc3 = Join[nc, {∞}]; NC3 = Join[NC, {∞}];
          FREQ3 = Join[FREQ, {∞}]; DAMP3 = Join[DAMP, {∞}];
          FoundF = {}; count = Table[i, {i, 1, number}];
          hits = spur = miss = split = fused = misclassified = 0; Index = {};
          (*   MatrixForm[{left3,right3,nc3}]
           MatrixForm[{LEFT3,RIGHT3,NC3,FREQ3}]
           Table[fr3[[k,left3[[1]]+1]],{k,1,nc3[[1]]}]
           {hits,miss,spur,misclassified, split,fused}
           Index   *)

In[165]:=

In[166]:= While[left3 ≠ {∞} || LEFT3 ≠ {∞},
            If[Max[left3[[1]], LEFT3[[1]]] ≤ Min[right3[[1]], RIGHT3[[1]]],
             {While[left3[[2]] ≤ RIGHT3[[1]],
               {left3 = Rest[left3]; right3 = Rest[right3]; nc3 = Rest[nc3]; split ++}];
              While[LEFT3[[2]] ≤ right3[[1]], {LEFT3 = Rest[LEFT3]; RIGHT3 = Rest[RIGHT3];
                NC3 = Rest[NC3]; FREQ3 = Rest[FREQ3];
                DAMP3 = Rest[DAMP3]; count = Rest[count]; fused ++}];
              If[nc3[[1]] == NC3[[1]] , {hits ++; Index = Append[Index, count[[1]]];
                FoundF = Append[FoundF, Table[fr3[[k, left3[[1]]]], {k, 1, nc3[[1]]}]];
                errF[hits] = Table[Abs[fr3[[k, left3[[1]]]] - FREQ3[[1, k]]], {k, 1, nc3[[1]]}];
                errD[hits] = Table[Abs[dmp3[[k, left3[[1]]]] + DAMP3[[1, k]]], {k, 1, nc3[[1]]}]},
               misclassified ++];
              left3 = Rest[left3]; right3 = Rest[right3]; LEFT3 = Rest[LEFT3];
              RIGHT3 = Rest[RIGHT3]; nc3 = Rest[nc3]; NC3 = Rest[NC3];
              FREQ3 = Rest[FREQ3]; DAMP3 = Rest[DAMP3]; count = Rest[count];}];
            If[right3[[1]] < LEFT3[[1]],
              {spur ++; right3 = Rest[right3]; left3 = Rest[left3]; nc3 = Rest[nc3];}]
             If[RIGHT3[[1]] < left3[[1]], {miss ++; RIGHT3 = Rest[RIGHT3]; LEFT3 = Rest[LEFT3];
               NC3 = Rest[NC3]; FREQ3 = Rest[FREQ3]; DAMP3 = Rest[DAMP3]; count = Rest[count];}]; ];
          (*  MatrixForm[{left3,right3,nc3}]
           MatrixForm[{LEFT3,RIGHT3,NC3,FREQ3}]
           Table[fr3[[k,left3[[1]]]],{k,1,nc3[[1]]}]
           {hits,miss,spur,misclassified, split,fused}
           Index
           count   *)

In[167]:= Do[errorF[k] = {}; ErrorD[k] = {};
            Do[If[Dimensions[errF[i]][[1]] ≥ k, {errorF[k] = Append[errorF[k], errF[i][[k]]];
               ErrorD[k] = Append[ErrorD[k], errD[i][[k]]];}], {i, 1, hits}], {k, 1, detect}];
          Do[ErrorFR[k] = Select[errorF[k], # < .3 &], {k, 1, detect}];
          Do[GrossErrorFR[k] = Select[ErrorFR[k], # ≥  .3 &], {k, 1, detect}];

In[170]:= adt = 8;
```

```
In[171]:= Do[SIG1add[k] = Take[Signal + noise NOISE, {LEFT[adt][[k]], RIGHT[adt][[k]] - 1}],
            {k, 1, number}];
          Do[SIG2add[k] = Take[Signal + noise NOISE, {LEFT[2 adt][[k]], RIGHT[2 adt][[k]] - 1}],
            {k, 1, number}];
          Do[F1add[k] = Table[f[2 k, centerf[2 k] + m / 2], {m, -size[2 k] - adt, size[2 k] + adt - 1}],
            {k, 1, number}];
          Do[F2add[k] = Table[f[2 k, centerf[2 k] + m / 2], {m, -size[2 k] - 2 adt, size[2 k] + 2 adt - 1}],
            {k, 1, number}];
          GR0 = Grid[{{"total number of segments"}, {number}}, Frame → All];
          GR2 = Grid[{{"hits", "misses", "spurious", "misclassified", "split", "fused"},
              {hits, miss, spur, misclassified, split, fused}}, Frame → All];
          GR1 = Grid[{Join[{"number of components"}, Table[i, {i, 1, detect}]],
              Join[{"number of segments"},
               Table[Dimensions[Select[Table[numsin[i], {i, 1, num}], (# == k) &]][[1]],
                {k, 1, detect}]]}, Frame → All];
          GR3 = Grid[Join[{{"error RMS", "error mean", "error median",
                "error maximum", "gross error maximum"}},
              Table[{RootMeanSquare[Join[ErrorFR[k], {0}]], Mean[Join[ErrorFR[k], {0}]],
                Median[Join[ErrorFR[k], {0}]], Max[Join[ErrorFR[k], {0}]],
                Max[Join[GrossErrorFR[k], {0}]] }, {k, 1, detect}]], Frame → All];
          Do[{
             LSP1[nn] = ListPlot[Table[{j, Take[Abs[Fourier[SIG1add[Index[[nn]]]]],
                    {1, Round[Dimensions[Fourier[SIG1add[Index[[nn]]]]][[1]] / 4] + 1}][[j + 1]]},
                 {j, 0, Round[Dimensions[Fourier[SIG1add[Index[[nn]]]]][[1]] / 4]}],
                ImageSize → 400, PlotRange → {0, Max[Abs[Fourier[SIG1add[Index[[nn]]]]]] + .5},
                GridLines → {Join[{{Pi Dimensions[SIG1add[Index[[1]]]][[1]] / (4 Pi), {Orange}}},
                   Table[{FREQ[[Index[[nn]], j]] Dimensions[SIG1add[Index[[nn]]]][[1]] / (4 Pi), {Red}},
                    {j, 1, numsin[2 Index[[nn]]]}],
                   Table[{FoundF[[nn, j]] Dimensions[SIG1add[Index[[nn]]]][[1]] / (4 Pi),
                     {Blue, Dashed}}, {j, 1, numsin[2 Index[[nn]]]}]], {}},
                PlotMarkers → {Automatic, Medium}, Axes → True, PlotLabel -> "DFT"] ;

             LSP2[nn] =
              ListPlot[{SIG2add[Index[[nn]]], F2add[Index[[nn]]]}, GridLines → {{{adt, {Blue, Thick}},
                  {2 adt, {Red, Thick}}, {Dimensions[SIG2add[Index[[nn]]]][[1]] - adt, {Blue, Thick}},
                  {Dimensions[SIG2add[nn]][[1]] - 2 adt, {Red, Thick}}}, {}},
                Joined → True, PlotLabel -> "Signal segment", ImageSize → 400];

             GR6[nn] = Grid[{{LSP1[nn]}, {LSP2[nn]}}, Spacings → {5}];

             GR4[nn] = Grid[Transpose[
                 Join[{{"component", "actual frequency", "measured frequency", "absolute error"}},
                  Table[{j, FREQ[[Index[[nn]], j]], FoundF[[nn, j]], Abs[FREQ[[Index[[nn]], j]] -
                        FoundF[[nn, j]]]}, {j, 1, numsin[2 Index[[nn]]]}]]], Frame → All];

             GR5[nn] = Grid[{{"index", "segment index", "# of components"},
                 {StringJoin[IntegerString[ nn], " (<", IntegerString[Dimensions[Index][[1]]], ")"],
                  Index[[nn]], numsin[2 Index[[nn]]]}}, Frame → All];}, {nn, 1, Dimensions[Index][[1]]
            }];
```

EXPLANATION OF TABLES AND PLOTS:

First table: **total number of functions (segments of) and their corresponding supports:**
Second table: **number of segments with the given number of components as CORRECTLY identified by the algorithm;**
Third table: **the number of correctly found segments (hits), the number of segments algorithm has missed (misses); the number of spurious segments, i.e, segments not originally inserted by the signal generation algorithm but which happen to look like ones; this can hapopen in the patches between legitimate segments.**

PLOTS: You can stroll through correctly identified segments using the 4 "arrow buttons" above the plots (the left corner of the "slide view"). The first plot gives the modulus of the DFT over the entire segment where the approximation is good, see the second plot. Thus: on the second plot the vertical red grid lines correspond to the support boundaries of each signal segment joined into the input signal; the blue gridlines correspond to the boundaries of the interpolation over which the signal remains a good approximation. For our chosen degree of smoothenes as ensured by interpolation (15) this is about 4 Nyquist rate unit intervals; since we oversample twice, this provides about 8 additional sampling pints on each side of the support, see [4] for details.

Inside the slide view the first table gives the ordinal number of each correctly identified interval, its index in the sequence of signals and its number of sinusoidal components; the second table gives actual frequency of each component, the corresponding measured frequency as obtained by the algorithm and the absolute error. the secon plot gives the waveform of the true signal component f[i,t] which is a sum of several sinewaves (red), the waveform of the noisy signal as obtained by patching these pieces together, corrupting them with noise and sampling them.

Second plot shows DFT part which is within the signal bandwidth (orange grid line), red gridlines shows the true frequncies of each component, the blue gridlines the established frequency estimates.

Second plot shows DFT part which is within the signal bandwidth (orange grid line), red grid-

lines shows the true frequncies of each component, the blue gridlines the established frequency estimates.

```
In[180]:= Grid[{{GR0}, {GR1}, {GR2}, {GR3}}]
```

Out[180]=

total number of segments
30

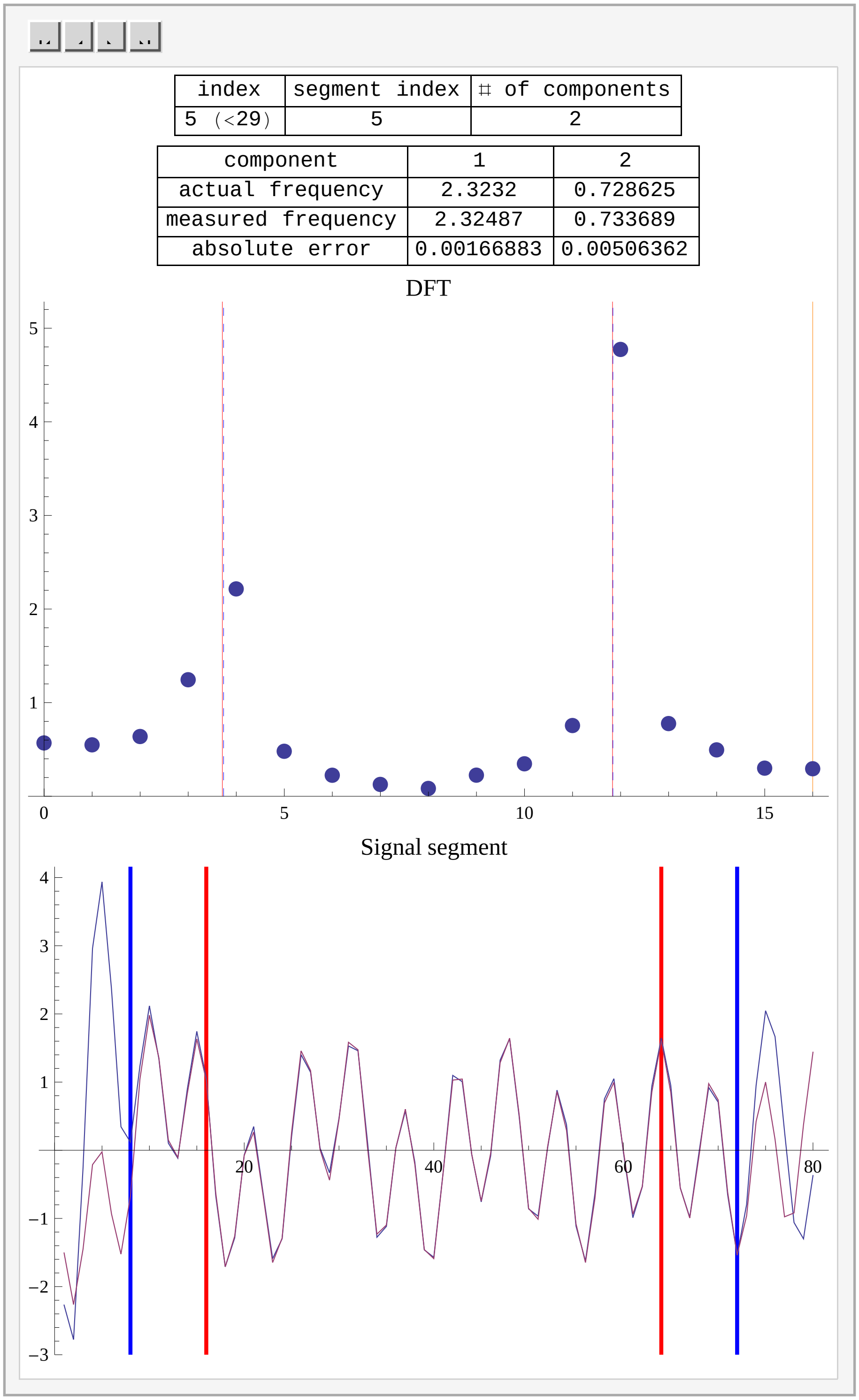
number of components	1	2	3
number of segments	11	7	12

hits	misses	spurious	misclassified	split	fused
29	0	0	1	0	0

error RMS	error mean	error median	error maximum	gross error maximum
0.00164239	0.000999475	0.000695868	0.00694469	0
0.0110914	0.00569946	0.00254822	0.0356536	0
0.0295155	0.0219122	0.0168158	0.0758692	0

```
In[181]:= SlideView[Table[Grid[{{GR5[ii]}, {GR4[ii]}, {GR6[ii]}}], {ii, 1, Dimensions[Index][[1]]}]]
```

Out[181]=



index	segment index	♯ of components
5 (<29)	5	2

component	1	2
actual frequency	2.3232	0.728625
measured frequency	2.32487	0.733689
absolute error	0.00166883	0.00506362

ENJOY playing with chromatic derivatives and **PLEASE do not hesitate to get in touch with me if you have ANY questions or comments**. I am leaving Sydney in two hours time going to Serbia and Greece; will be back at the end of July; I might have internet access whle at home in Serbia.

REFERENCES (all available at http://www.cse.unsw.edu.au/~ignjat/diff/)

[1] A. Ignjatovic: "***Frequency estimation using time domain methods based on robust differential operators***", preprint.
[2] -"- "***Chromatic Derivatives, Chromatic Expansions and Associated Spaces***", East Journal on Approximations, Volume 15, Number 3 (2009), 263-302.
[3] -"- "***Chromatic derivatives and local approximations",*** IEEE Transactions on Signal Processing, Volume 57, Issue 8, 2009.
[4] -"- "***Signal interpolation using numerically robust differential operators***", 14th WSEAS CSCC Multiconference, July 22-25, 2010, Corfu Island, Greece.