# L14 Signal Decomposition Using EVD of Hankel Matrices

Ivan Slapničar

May 23, 2018

## 1 Signal Decomposition Using EVD of Hankel Matrices

Suppose we are given a data signal which consists of several nearly mono-components.

*Can we recover the mono-components?*

The answer is *YES, with an efficient algorithm using EVDs of Hankel matrices.*

Mono-component recovery can be successfully applied to audio signals.

### 1.1 Prerequisites

The reader should be familiar to elementary concepts about signals, and with linear algebra concepts, particularly EVD and its properties and algorithms.

### 1.2 Competences

The reader should be able to decompose given signal into mono-components using EVD methods.

### 1.3 References

For more details see P. Jain and R. B. Pachori, An iterative approach for decomposition of multi-component non-stationary signals based on eigenvalue decomposition of the Hankel matrix.

**Credits**: The first Julia implementation was derived in Section **??**.

### 1.4 Extraction of stationary mono-components

#### 1.4.1 Definitions

Let $x \in \mathbb{R}^m$, denote a **signal** with $N$ samples.

Assume $x$ is composed of $L$ **stationary mono-components**:

$$x = \sum_{k=1}^{L} x^{(k)},$$

where

$$x_i^{(k)} = A_k \cos(2\pi f_k i + \theta_k) \quad i = 1, 2, \ldots, m.$$

Here:

$f_k = \dfrac{F_k}{F}$ is the **normalized frequency** of $x^{(k)}$,

$F$ is the **sampling frequency** of $x$ in Hz,

$F_k$ is the sampling frequency of $x^{(k)}$,

$A_k$ is the **amplitude** of $x^{(k)}$, and

$\theta_k$ is the **phase** of $x^{(k)}$.

We assume that $F_k < F_{k+1}$ for $k = 1, 2, \ldots, n-1$, and $F > 2F_n$.

A **Hankel matrix** is a (real) square matrix with constant values along the skew-diagonals. More precisely, let $a \in \mathbb{R}^{2n-1}$. An $n \times n$ matrix $H \equiv H(a)$ for which $H_{ij} = A_{i+1,j-1} = a_{i+j-1}$ is a Hankel matrix.

### 1.4.2 Facts

Let $x$ be a signal with $2n-1$ samples composed of $L$ stationary mono-components.

Let $H$ be an $n \times n$ Hankel matrix corresponding to $x$ and let $H = U\Lambda U^T$ be its EVD (Hankel matrix is symmetric) with $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$.

Smilarly, let $H_k$ be the $n \times n$ Hankel matrix corresponding to the $k$-th component $x^{(k)}$ and let $H_k = U_k \Lambda_k U_k^T$ be its EVD.

1. $H = \sum\limits_{k=1}^{L} H_k$.
2. $H_k = \lambda_k U_{:,k} U_{:,k}^T + \lambda_{n-k+1} U_{:,n-k+1} U_{:,n-k+1}^T$.

### 1.4.3 Example - Signal with three mono-components

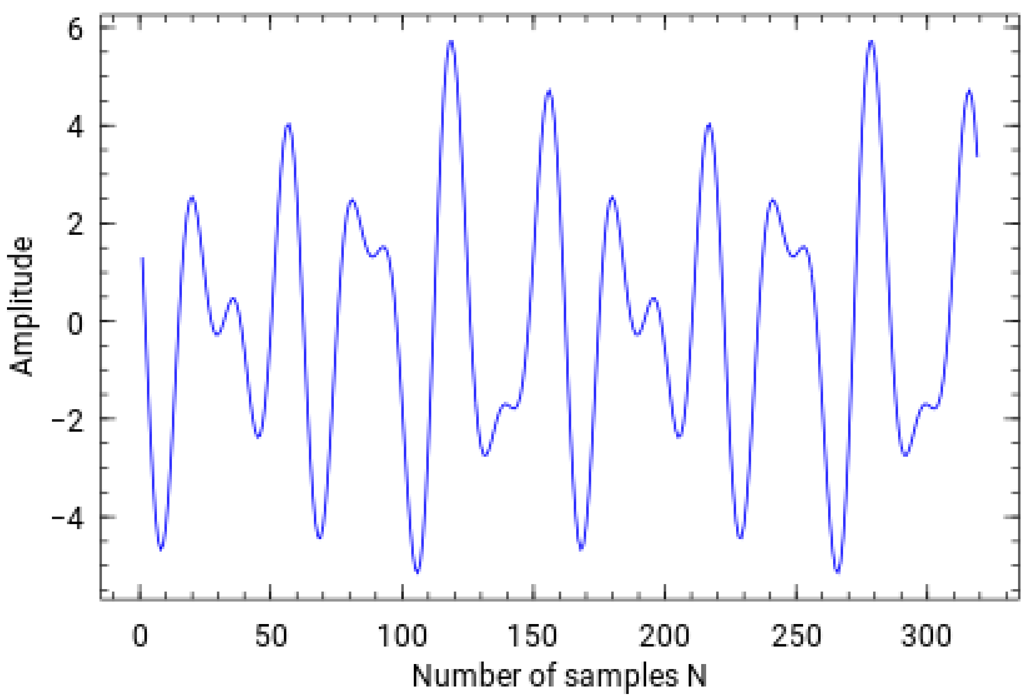
In [1]:
```
using Winston
using SpecialMatrices
```

In [2]:
```
# Small Hankel matrix
a=collect(1:11)
Hankel(a)
```

Out[2]:
```
6×6 SpecialMatrices.Hankel{Int64}:
 1  2  3  4   5   6
 2  3  4  5   6   7
 3  4  5  6   7   8
 4  5  6  7   8   9
 5  6  7  8   9  10
 6  7  8  9  10  11
```

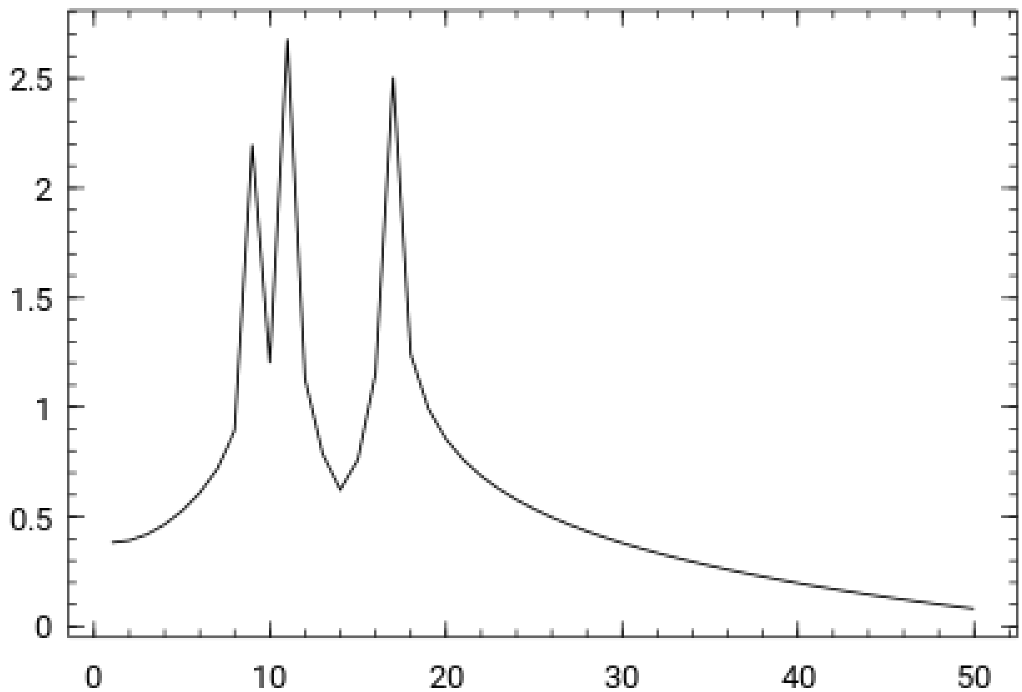
```
In [3]: # Create the signal
        n=160
        N=2*n-1
        F = 6400
        L = 3
        A = [3, 2, 1]
        Fk= [200, 320, 160]
        θ = [pi/2, pi/4, 0]
        x = zeros(N)
        for k=1:L
            for i=1:N
                x[i]+=A[k]*cos(2*pi*Fk[k]*i/F+θ[k])
            end
        end
        plot(x,"b",xlabel="Number of samples N", ylabel="Amplitude")
```

Out[3]:




```
In [4]: # FFT indicates that there are three components
        y=fft(x)
        plot(log10.(abs.(y[1:50])))
```

Out[4]:




```
In [5]: # Let us decompose the signal
        H=Hankel(x)
```

```
Out[5]: 160×160 SpecialMatrices.Hankel{Float64}:
  1.3104     0.115875  -1.08857    …   4.07448    3.35497    2.41421
  0.115875  -1.08857   -2.22028        3.35497    2.41421    1.3104
 -1.08857   -2.22028   -3.20152        2.41421    1.3104     0.115875
 -2.22028   -3.20152   -3.96587        1.3104     0.115875  -1.08857
 -3.20152   -3.96587   -4.46374        0.115875  -1.08857   -2.22028
 -3.96587   -4.46374   -4.66636    …  -1.08857   -2.22028   -3.20152
 -4.46374   -4.66636   -4.56793       -2.22028   -3.20152   -3.96587
 -4.66636   -4.56793   -4.18585       -3.20152   -3.96587   -4.46374
 -4.56793   -4.18585   -3.55882       -3.96587   -4.46374   -4.66636
 -4.18585   -3.55882   -2.74321       -4.46374   -4.66636   -4.56793
 -3.55882   -2.74321   -1.80783    …  -4.66636   -4.56793   -4.18585
 -2.74321   -1.80783   -0.827855      -4.56793   -4.18585   -3.55882
 -1.80783   -0.827855   0.121836      -4.18585   -3.55882   -2.74321
  ⋮                                ⋱
  0.555961   1.35743    2.19081       -1.22175   -0.762656  -0.163073
  1.35743    2.19081    2.99615       -0.762656  -0.163073   0.555961
  2.19081    2.99615    3.70922    …  -0.163073   0.555961   1.35743
  2.99615    3.70922    4.2674         0.555961   1.35743    2.19081
```

```
 3.70922    4.2674     4.61573       1.35743    2.19081    2.99615
 4.2674     4.61573    4.71235       2.19081    2.99615    3.70922
 4.61573    4.71235    4.53309       2.99615    3.70922    4.2674
 4.71235    4.53309    4.07448   ...  3.70922    4.2674     4.61573
 4.53309    4.07448    3.35497       4.2674     4.61573    4.71235
 4.07448    3.35497    2.41421       4.61573    4.71235    4.53309
 3.35497    2.41421    1.3104        4.71235    4.53309    4.07448
 2.41421    1.3104     0.115875      4.53309    4.07448    3.35497
```

In [6]:
```
λ,U=eig(full(H))
λ
```

Out[6]:
```
160-element Array{Float64,1}:
 -240.0
 -160.0
  -80.0
   -6.71857e-13
   -4.44089e-14
   -3.78389e-14
   -3.44169e-14
   -2.3885e-14
   -2.35646e-14
   -2.22281e-14
   -2.1441e-14
   -2.10717e-14
   -2.07968e-14
    ⋮
    2.30926e-14
    2.39808e-14
    2.4647e-14
    2.54311e-14
    2.79693e-14
    3.01408e-14
    3.16842e-14
    3.81917e-14
    7.9198e-14
   80.0
  160.0
  240.0
```

We see that the three smallest and the three largest eigenvalues come in pairs and define the three mono-components.

The ratios of the moduli of the eigenvalues correspond to the ratios of the amplitudes of the mono-components.

In [7]:
```
# Form the three matrices
Hcomp=Array{Any}(3)
```

```
for k=1:L
    Hcomp[k]=λ[k]*U[:,k]*U[:,k]' + λ[end-k+1]*U[:,end-k+1]*U[:,end-k+1]'
end
```

In [8]:
```
# Compare the first matrix with the Hankel matrix of the first mono-component
x1 = zeros(N)
l=1
for i=1:N
    x1[i]+=A[l]*cos(2*pi*Fk[l]*i/F+θ[l])
end
```

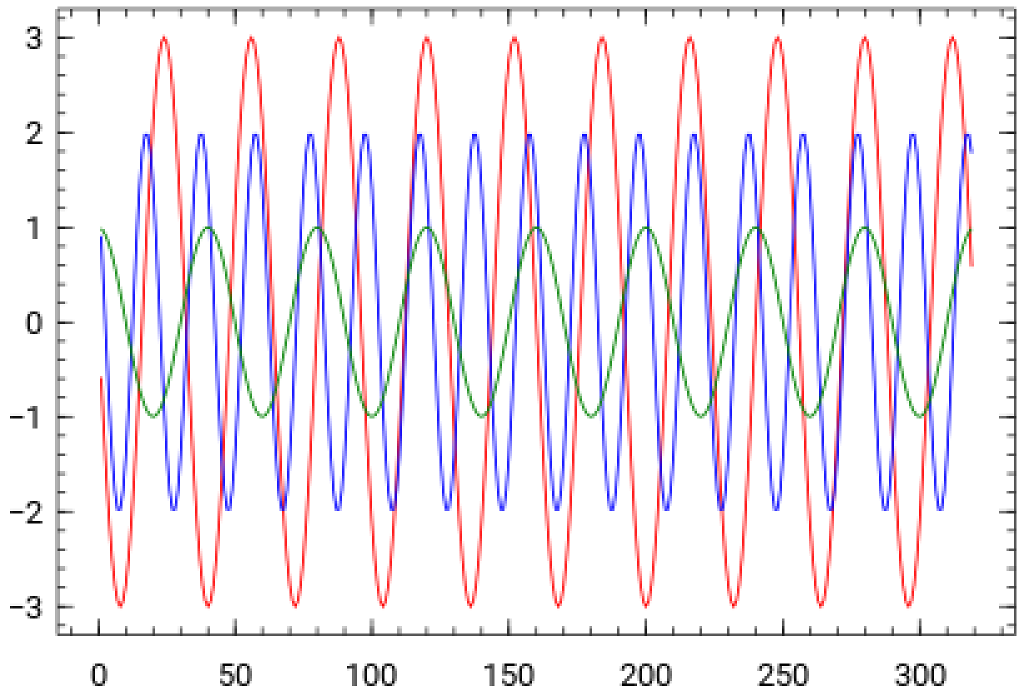
In [9]:
```
H1=Hankel(x1)
eigvals(full(H1)), norm(Hcomp[1]-H1)
```

Out[9]:
```
([-240.0, -1.86035e-13, -1.82753e-13, -1.72388e-13, -1.63843e-13, -1.60189e-13, -1.41385
```

In [10]:
```
# Now we reconstruct the mono-components from the skew-diagonal elements of Hcomp
xcomp=Array{Array{Float64}}(L)
z=Array{Float64}(N)
for k=1:L
    z[1:2:N]=diag(Hcomp[k])
    z[2:2:N]=diag(Hcomp[k],1)
    xcomp[k]=copy(z)
end
```

In [11]:
```
xaxis=collect(1:N)
plot(xaxis,xcomp[1],"r", xaxis,xcomp[2],"b", xaxis,xcomp[3],"g")
```

Out[11]:




### 1.5 Fast EVD of Hankel matrices

Several outer eigenvalues pairs of Hankel matrices can be computed using Lanczos method. If the multiplication $Hx$ is performed using Fast Fourier Transform, this EVD computation is very fast.

#### 1.5.1 Definitions

A **Toeplitz matrix** is a (real) square matrix with constant values along the diagonals. More precisely, let

$$a = (a_{-(n-1)}, a_{-(n-2)}, \ldots, a_{-1}, a_0, a_1, \ldots, a_{n-1}) \in \mathbb{R}^{2n-1}.$$

An $n \times n$ matrix $T \equiv T(a)$ for which $T_{ij} = T_{i+1,j+1} = a_{i-j}$ is a Toeplitz matrix.

A **circulant matrix** is a Toeplitz matrix where each column is rotated one element downwards relative to preceeding column.

More precisely, let $a \in \mathbb{R}^n$. An $n \times n$ matrix $C \equiv C(a) = T(a, a_{1:n-1})$ is a Circulant matrix.

A **rotation matrix** is an identity matrix rotated 90 degrees to the right (or left).

A **Fourier matrix** is Vandermonde matrix

$$F_n = V(1, \omega_n, \omega_n^2, \ldots, \omega_n^{n-1}),$$

where $\omega_n = exp(2\pi i/n)$ is the $n$-th root of unity (see the Eigenvalue Decomposition - Definitions and Facts notebook).

### 1.5.2 Example

Notice different meaning of vector $a$: in `C=Circulant(a)`, $a$ is the first column, in `T=Toeplitz(a)`, $a_i$ is the diagonal element of the $i$-th diagonal starting from $T_{1n}$, and in `H=Hankel(a)`, $a_i$ is the element of the $i$-th skew-diagonal starting from $H_{11}$.

```
In [12]: C=Circulant([1,2,3,4,5])
```

```
Out[12]: 5×5 SpecialMatrices.Circulant{Int64}:
          1  5  4  3  2
          2  1  5  4  3
          3  2  1  5  4
          4  3  2  1  5
          5  4  3  2  1
```

```
In [13]: TC=Toeplitz([2,3,4,5,1,2,3,4,5])
```

```
Out[13]: 5×5 SpecialMatrices.Toeplitz{Int64}:
          1  5  4  3  2
          2  1  5  4  3
          3  2  1  5  4
          4  3  2  1  5
          5  4  3  2  1
```

```
In [14]: T=Toeplitz([1,2,3,4,5,6,7,8,9])
```

```
Out[14]: 5×5 SpecialMatrices.Toeplitz{Int64}:
          5  4  3  2  1
          6  5  4  3  2
          7  6  5  4  3
          8  7  6  5  4
          9  8  7  6  5
```

```
In [15]: H1=Hankel([1,2,3,4,5,6,7,8,9])
```

```
Out[15]: 5×5 SpecialMatrices.Hankel{Int64}:
          1  2  3  4  5
          2  3  4  5  6
          3  4  5  6  7
          4  5  6  7  8
          5  6  7  8  9
```

### 1.5.3 Facts

For more details see G. H. Golub and C. F. Van Loan, Matrix Computations, p. 202, and the references therein

1. Hankel matrix is the product of a Toeplitz matrix and the rotation matrix.
2. Circulant matrix is normal and, thus, unitarily diagonalizable, with the eigenvalue decomposition
$$C(a) = U \operatorname{diag}(F_n^* a) U^*,$$
where $U = \dfrac{1}{\sqrt{n}} F_n$. The product $F_n^* a$ can be computed by the *Fast Fourier Transform*(FFT).
3. Given $a, x \in \mathbb{R}^n$, the product $y = C(a)x$ can be computed using FFT as follows:
$$\begin{aligned}\tilde{x} &= F_n^* x\\ \tilde{a} &= F_n^* a\\ \tilde{y} &= \tilde{x}. * \tilde{a}\\ y &= F_n^{-*} \tilde{y}.\end{aligned}$$
4. Toeplitz matrix of order $n$ can be embedded in a circulant matrix of order $2n-1$: if $a \in \mathbb{R}^{2n-1}$, then
$$T(a) = [C([a_{n+1:2n-1}; a_{1:n}])]_{1:n,1:n}.$$
5. Further, let $x \in \mathbb{R}^n$ and let $\bar{x} \in \mathbb{R}^{2n-1}$ be equal to $x$ padded with $n-1$ zeros.Then
$$T(a)x = [C([a_{n+1:2n-1}; a_{1:n}])\bar{x}]_{1:n}.$$
6. Fact 1 implies $H(a)x = (T(a)J)x = T(a)(Jx)$.

### 1.5.4 Examples

In [16]:
```
# Fact 1
J=rotl90(eye(Int64,5))
T*J
```

Out[16]:
```
5×5 Array{Int64,2}:
 1  2  3  4  5
 2  3  4  5  6
 3  4  5  6  7
 4  5  6  7  8
 5  6  7  8  9
```

In [17]:
```
# Fact 1
full(T)*J
```

```
Out[17]: 5×5 Array{Int64,2}:
          1  2  3  4  5
          2  3  4  5  6
          3  4  5  6  7
          4  5  6  7  8
          5  6  7  8  9
```

```
In [18]: # Fact 2
         a=rand(-8:8,6)
         n=length(a)
         C=Circulant(a)
         ω=exp(2*pi*im/n)
         v=map(Complex,[ω^k for k=0:n-1])
         F=Vandermonde(v)
         U=F/sqrt(n)
         λ=full(F)'*a
```

```
Out[18]: 6-element Array{Complex{Float64},1}:
          -4.0+0.0im
          -7.5-11.2583im
           3.5-4.33013im
           6.0+1.62093e-14im
           3.5+4.33013im
          -7.5+11.2583im
```

```
In [19]: # Residual
         norm(full(C)*U-U*diagm(λ))
```

```
Out[19]: 4.4393619099953567e-14
```

```
In [20]: ?fft;
```

```
search: fft fft! FFTW fftshift rfft ifft bfft ifft! bfft! ifftshift irfft brfft
```

```
In [21]: # Check fft
         norm(λ-fft(a))
```

```
Out[21]: 4.978802550048858e-14
```

Fact 3 - Circulant() x vector, as implemented in the package `SpecialMatrices.jl`

```
function *{T}(C::Circulant{T},x::Vector{T})
    xt=fft(x)
    vt=fft(C.c)
    yt=vt.*xt
    real(ifft(yt))
end
```

Similarly, `A_mul_B!()`

```
function A_mul_B!{T}(y::StridedVector{T},C::Circulant{T},x::StridedVector{T})
    xt=fft(x)
    vt=fft(C.c)
    yt=ifft(vt.*xt)
    if T<: Int
        map!(round,y,yt)
    elseif T<: Real
        map!(real,y,yt)
    else
        copy!(y,yt)
    end
    return y
end
```

```
In [22]: x=rand(-9:9,n)
```

```
Out[22]: 6-element Array{Int64,1}:
           8
           1
           9
          -7
           2
           7
```

```
In [23]: [full(C)*x C*x A_mul_B!(similar(x),C,x)]
```

```
Out[23]: 6×3 Array{Int64,2}:
          -26  -26  -26
          -33  -33  -33
           84   84   84
           12   12   12
          -44  -44  -44
          -73  -73  -73
```

```
In [24]: # Fact 4 - Embedding Toeplitz() into Circulant()
         n=5
         a=rand(-6:6,2*n-1)
         T=Toeplitz(a)
```

```
Out[24]: 5×5 SpecialMatrices.Toeplitz{Int64}:
          -1  -4   1  -4   3
           4  -1  -4   1  -4
           3   4  -1  -4   1
           4   3   4  -1  -4
          -1   4   3   4  -1
```

```
In [25]: C=Circulant([a[n:2*n-1];a[1:n-1]])
```

```
Out[25]: 9×9 SpecialMatrices.Circulant{Int64}:
          -1  -4   1  -4   3  -1   4   3   4
           4  -1  -4   1  -4   3  -1   4   3
           3   4  -1  -4   1  -4   3  -1   4
           4   3   4  -1  -4   1  -4   3  -1
          -1   4   3   4  -1  -4   1  -4   3
           3  -1   4   3   4  -1  -4   1  -4
          -4   3  -1   4   3   4  -1  -4   1
           1  -4   3  -1   4   3   4  -1  -4
          -4   1  -4   3  -1   4   3   4  -1
```

```
In [26]: # Fact 5 - Toeplitz() x vector
         x=rand(-6:6,n)
```

```
Out[26]: 5-element Array{Int64,1}:
           2
           0
           1
           2
          -4
```

```
In [27]: [full(T)*x T*x A_mul_B!(similar(x),T,x)]
```

```
Out[27]: 5×3 Array{Int64,2}:
          -21  -21  -21
           22   22   22
           -7   -7   -7
           26   26   26
           13   13   13
```

```
In [28]: # Fact 6 - Hankel() x vector
         H1=Hankel(a)
```

```
Out[28]: 5×5 SpecialMatrices.Hankel{Int64}:
           3  -4   1  -4  -1
          -4   1  -4  -1   4
           1  -4  -1   4   3
          -4  -1   4   3   4
          -1   4   3   4  -1
```

```
In [29]: [full(H1)*x H1*x A_mul_B!(similar(x),H1,x)]
```

```
Out[29]: 5×3 Array{Int64,2}:
            3    3    3
```

```
-30  -30  -30
 -3   -3   -3
-14  -14  -14
 13   13   13
```

### 1.5.5 Example - Fast EVD of a Hankel matrix

Given a Hankel matrix $H$, the Lanczos method can be applied by defining a function (linear map) which returns the product $Hx$ for any vector $x$. This approach uses the package LinearMaps.jl and is described in the Symmetric Eigenvalue Decomposition - Lanczos Method notebook.

*The computation is very fast and allocates little extra space.*

**IMPORTANT** For package SpecialMatrices.jl to work with very large Hankel matrices, we need to modify the corresponding lines in the file `hankel.jl` to

```
getindex(H::Hankel, i, j) = H.c[i+j-1]
isassigned(H::Hankel, i, j) = isassigned(H.c, i+j-1)
```

```
In [30]: using LinearMaps
```

```
In [31]: n=size(H,1)
         f(x)=A_mul_B!(similar(x),H,x)
```

```
Out[31]: f (generic function with 1 method)
```

```
In [32]: H
```

```
Out[32]: 160×160 SpecialMatrices.Hankel{Float64}:
  1.3104      0.115875  -1.08857   …   4.07448    3.35497    2.41421
  0.115875   -1.08857   -2.22028       3.35497    2.41421    1.3104
 -1.08857    -2.22028   -3.20152       2.41421    1.3104     0.115875
 -2.22028    -3.20152   -3.96587       1.3104     0.115875  -1.08857
 -3.20152    -3.96587   -4.46374       0.115875  -1.08857   -2.22028
 -3.96587    -4.46374   -4.66636   …  -1.08857   -2.22028   -3.20152
 -4.46374    -4.66636   -4.56793      -2.22028   -3.20152   -3.96587
 -4.66636    -4.56793   -4.18585      -3.20152   -3.96587   -4.46374
 -4.56793    -4.18585   -3.55882      -3.96587   -4.46374   -4.66636
 -4.18585    -3.55882   -2.74321      -4.46374   -4.66636   -4.56793
 -3.55882    -2.74321   -1.80783   …  -4.66636   -4.56793   -4.18585
 -2.74321    -1.80783   -0.827855     -4.56793   -4.18585   -3.55882
 -1.80783    -0.827855   0.121836     -4.18585   -3.55882   -2.74321
  ⋮                                ⋱
  0.555961    1.35743    2.19081      -1.22175   -0.762656  -0.163073
  1.35743     2.19081    2.99615      -0.762656  -0.163073   0.555961
  2.19081     2.99615    3.70922   …  -0.163073   0.555961   1.35743
  2.99615     3.70922    4.2674        0.555961   1.35743    2.19081
```

```
3.70922    4.2674     4.61573       1.35743    2.19081    2.99615
4.2674     4.61573    4.71235       2.19081    2.99615    3.70922
4.61573    4.71235    4.53309       2.99615    3.70922    4.2674
4.71235    4.53309    4.07448   ...   3.70922    4.2674     4.61573
4.53309    4.07448    3.35497       4.2674     4.61573    4.71235
4.07448    3.35497    2.41421       4.61573    4.71235    4.53309
3.35497    2.41421    1.3104        4.71235    4.53309    4.07448
2.41421    1.3104     0.115875      4.53309    4.07448    3.35497
```

```
In [33]: # Need Pkg.add("LinearMaps"); Pkg.checkout("LinearMaps")
         A=LinearMap(f,n,issymmetric=true)

Out[33]: LinearMaps.FunctionMap{Float64}(f, nothing, 160, 160; ismutating=false, issymmetric=tru

In [34]: size(A)

Out[34]: (160, 160)

In [35]: @time eigs(full(H));

  3.184200 seconds (1.57 M allocations: 79.998 MiB, 2.98% gc time)


In [37]: # Run twice
         @time λA,UA=eigs(A, nev=6, which=:LM)

  0.004240 seconds (3.87 k allocations: 1.126 MiB)


Out[37]: ([-240.0, 240.0, 160.0, -160.0, -80.0, 80.0], [0.0864252 -0.0709273 ... 0.00877199 0.11
```

## 1.6 Extraction of non-stationary mono-components

### 1.6.1 Definitions

Let $x \in \mathbb{R}^m$, denote a **signal** with $N$ samples.

Assume $x$ is composed of $L$ **non-stationary mono-components**:

$$x = \sum_{k=1}^{L} x^{(k)},$$

where

$$x_i^{(k)} = A_k \cos(2\pi f_k i + \theta_k), \quad i = 1, 2, \ldots, m.$$

Assume that the normalized frequencies $f_k = \frac{F_k}{F}$, the sampling frequencies $F_k$, the amplitudes $A_k$, and the phases $\theta_k$, all *sightly* change in time.

Let $H \equiv H(x)$ be the Hankel matrix of $x$. The eigenpair of $(\lambda, u)$ of $H$ is **significant** if $|\lambda| > \tau \cdot \sigma(H)$. Here $\sigma(H)$ is the spectral radius of $H$, and $\tau$ is the **significant threshold percentage** chosen by the user depending on the type of the problem.

### 1.6.2 Fact

The following algorithm decomposes the signal $x$: 1. Choose $\tau$ and form the Hankel matrix $H$ 2. Compute the EVD of $H$ 3. Choose the significant eigenpairs of $H$ 4. For each significant eigenpair $(\lambda, u)$ 1. Form the rank one matrix $M = \lambda u u^T$ 2. Define a new signal $y$ consisting of averages of the skew-diagonals of $M$ 3. Form the Hankel matrix $H(y)$ 3. Compute the EVD of $H(y)$ 4. Choose the significant eigenpairs of $H(y)$ 5. **If** $H(y)$ has only two significant eigenpairs, declare $y$ a mono-component, **else** go to step 4.

### 1.6.3 Example - Note A

Each tone has its fundamental frequency (mono-component). However, musical instruments produce different overtones (harmonics) which are near integer multiples of the fundamental frequency. Due to construction of resonant boxes, these frequencies slightly vary in time, and their amplitudes are contained in a time varying envelope.

Tones produces by musical instruments are nice examples of non-stationary signals. We shall decompose the note A4 played on piano.

For manipulation of recordings, we are using package WAV.jl. Another package with similar functionality is the package AudioIO.jl.

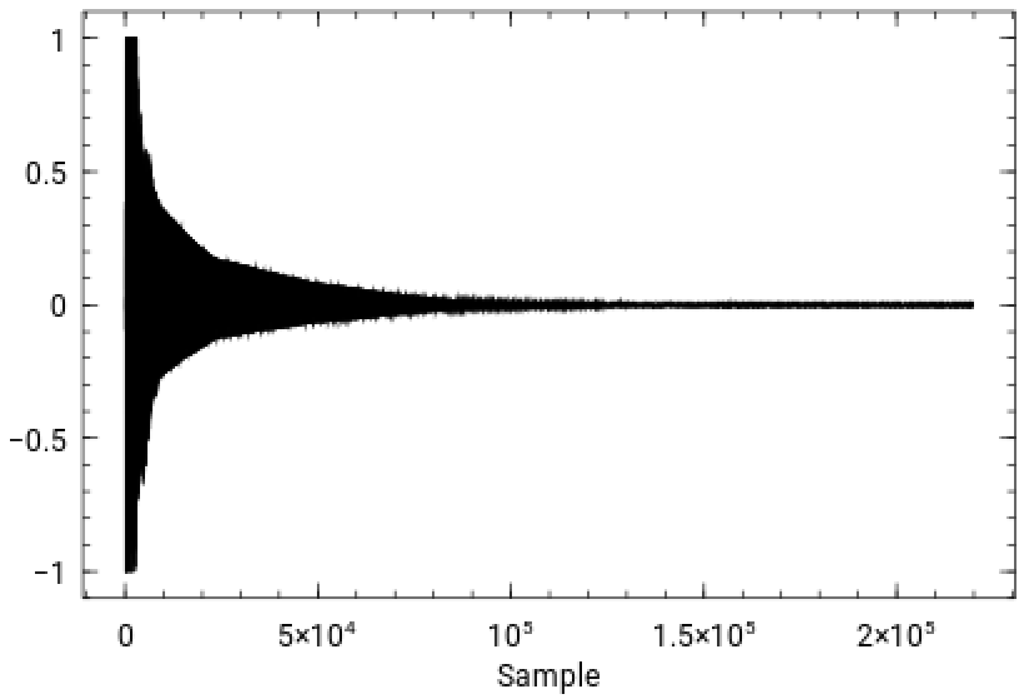
```
In [38]: # Pkg.checkout("WAV")
         using WAV
```

```
In [39]: whos(WAV)
```

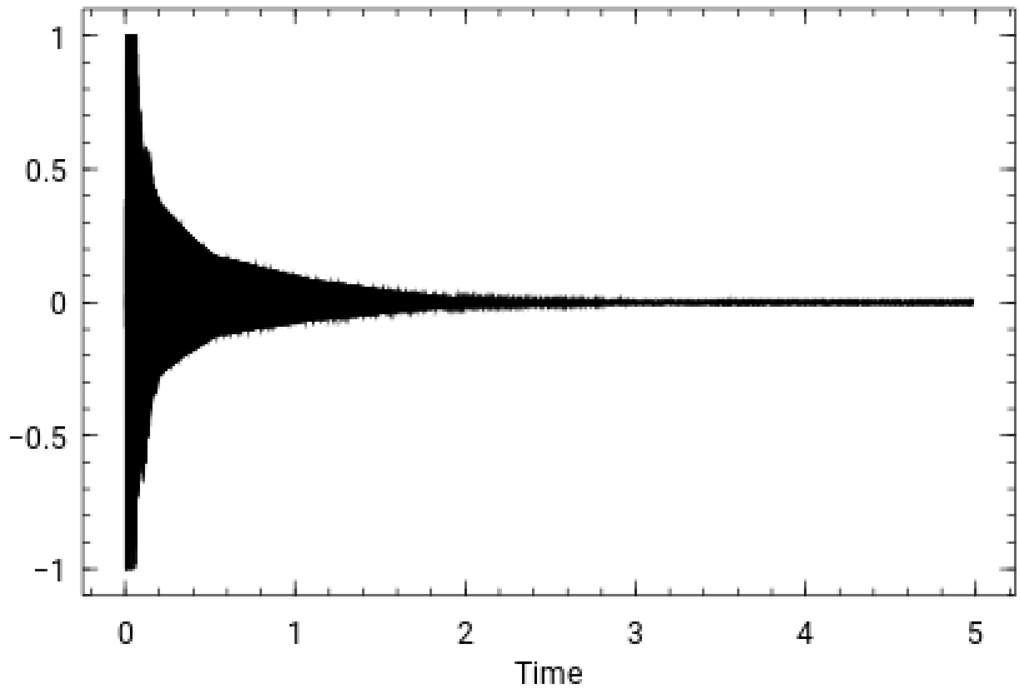
```
                    WAV  29719 KB     Module
               WAVArray     80 bytes  UnionAll
      WAVE_FORMAT_ALAW       2 bytes  UInt16
WAVE_FORMAT_IEEE_FLOAT       2 bytes  UInt16
     WAVE_FORMAT_MULAW       2 bytes  UInt16
       WAVE_FORMAT_PCM       2 bytes  UInt16
             WAVFormat     184 bytes  DataType
    WAVFormatExtension     136 bytes  DataType
       bits_per_sample       0 bytes  WAV.#bits_per_sample
          isextensible       0 bytes  WAV.#isextensible
              isformat       0 bytes  WAV.#isformat
             wavappend       0 bytes  WAV.#wavappend
               wavplay       0 bytes  WAV.#wavplay
               wavread       0 bytes  WAV.#wavread
```

```
wavwrite       0 bytes  WAV.#wavwrite
```

```
In [40]: # Load a signal
         s, Fs = wavread("files/piano_A41.wav")

Out[40]: ([-0.0101321; -0.0102542; ... ; 0.0; 0.0], 44100.0f0, 0x0010, Dict{Symbol,Any}(Pair{Sym

In [41]: # Play the signal
         wavplay(s,Fs)

In [42]: # Plot the signal
         plot(s,xlabel="Sample")

Out[42]:
```

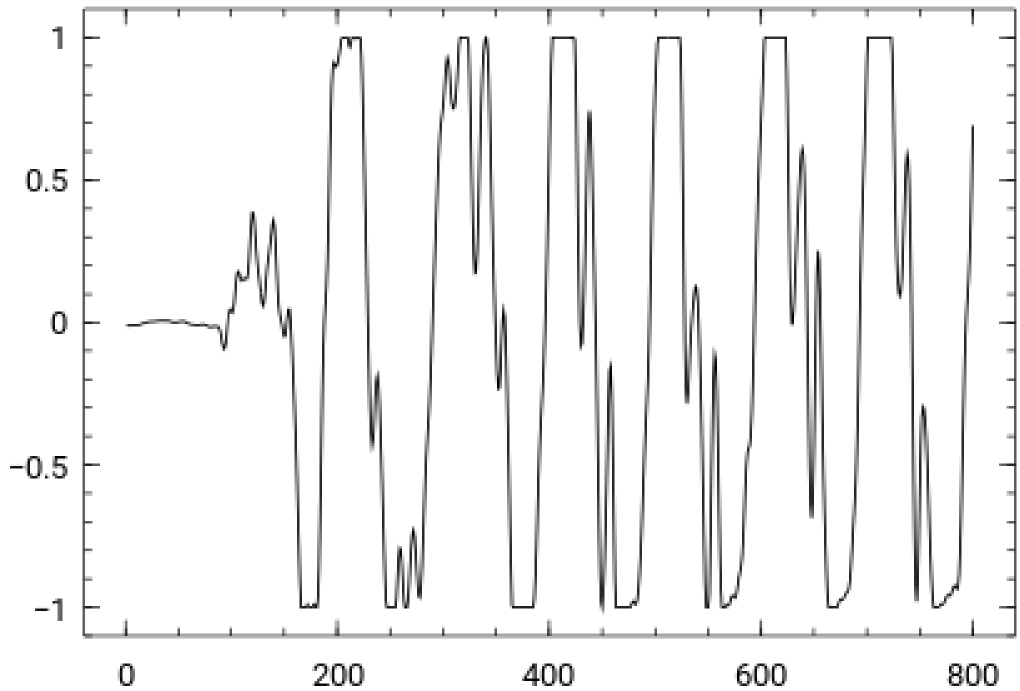



```
In [43]: # Plot the signal in time scale
         xx=linspace(0,length(s)/Fs,length(s))
         plot(xx,s,xlabel="Time")

Out[43]:
```




In [44]:
```
# Detail
plot(s[1:800])
```

Out[44]:




Let us visualize the signal in detail using the approach from the Julia is Fast notebook.

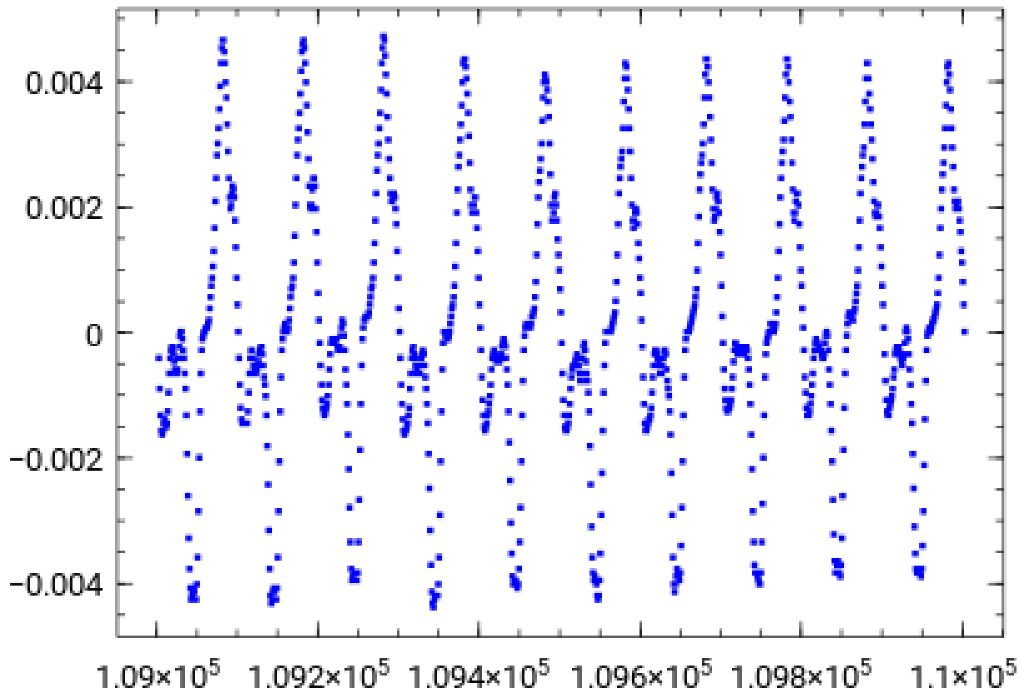
```
In [45]: using Interact
```

```
INFO: Interact.jl: using new nbwidgetsextension protocol
```

```
In [46]: @manipulate for k=1:1000:size(s,1)
             plot(collect(k:k+1000),s[k:k+1000],"b.")
         end
```

```
Interact.Options{:SelectionSlider,Int64}(1: "input" = 109001 Int64 , "k", 109001, "109001", 110,
```

```
Out[46]:
```




```
In [47]: # Last part of the signal is just noise, so we read a
         # shorter signal. N must be odd.
         sig = wavread("files/piano_A41.wav",100001)

Out[47]: ([-0.0101321; -0.0102542; ... ; -0.00286874; -0.00305185], 44100.0f0, 0x0010, Dict{Symb

In [48]: typeof(sig)

Out[48]: Tuple{Array{Float64,2},Float32,UInt16,Dict{Symbol,Any}}

In [49]: s=sig[1]

Out[49]: 100001×1 Array{Float64,2}:
          -0.0101321
          -0.0102542
          -0.0102542
          -0.0101321
          -0.00994903
          -0.00964385
          -0.00933866
          -0.00909452
          -0.00878933
```

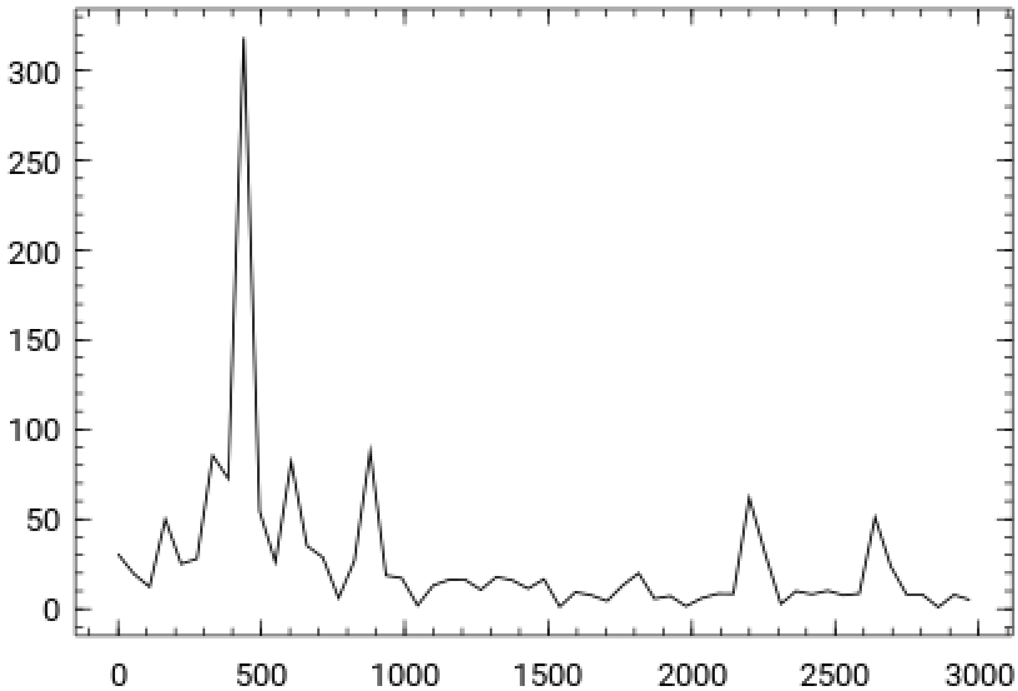
```
 -0.00860622
 -0.00842311
 -0.00799585
 -0.00720237
   ⋮
  0.00335704
  0.00317392
  0.00286874
  0.00231941
  0.00152593
  0.000549333
 -0.000427259
 -0.00134281
 -0.0021363
 -0.00262459
 -0.00286874
 -0.00305185
```

```
In [50]: wavplay(s,Fs)
```

```
In [51]: # File to play on Windows
         wavwrite(sig[1],"files/piano_A41_short.wav",Fs=sig[2])
```

```
In [52]: # Check the signal with FFT
         Fd=110
         N=convert(Int32,ceil(Fs/Fd))
         xx=collect(0:Fs/(2N):3000)
         nn=length(xx)
         plot(xx,abs.(fft(s[1:800]))[1:nn])
```

```
Out[52]:
```




```
In [53]: # Form the Hankel matrix
         # IMPORTANT - Do not try to display H - it is a 50001 x 50001 matrix.
         x=vec(s)
         H=Hankel(x);
```

```
In [54]: size(H), H[100,200]
```

```
Out[54]: ((50001, 50001), 0.727927488021485)
```

```
In [55]: @time fft(x);
```

```
  0.041465 seconds (60 allocations: 3.055 MiB)
```

```
In [56]: # We are looking for 20 eigenvalue pairs
         n=size(H,1)
         f(x)=A_mul_B!(similar(x),H,x)
         A=LinearMap(f,n,issymmetric=true)
         size(A)
```

```
Out[56]: (50001, 50001)
```

```
In [57]: @time λ,U=eigs(A, nev=40, which=:LM)

 13.074343 seconds (407.19 k allocations: 1.670 GiB, 3.66% gc time)


Out[57]: ([1802.91, -1799.5, 1297.45, -1296.27, -537.888, 537.759, 416.906, -415.32, 410.179, -4
```

```
In [58]: # Count the eigenvalue pairs (+-) larger than the 10% of the maximum
         τ=0.1
         L=round(Int,(sum(abs.(λ).>(τ*maximum(abs,λ)))/2))

Out[58]: 11
```

At this point, the implementation using full matrices is rather obvious. However, we cannot do that, due to large dimension. Recall, the task is to define Hankel matrices $H_k$ for $k = 1, \ldots, L$, from the signal obtained by averaging the skew-diagonals of the matrices

$$H_k = \lambda_k U_{:,k} U_{:,k}^T + \lambda_{n-k+1} U_{:,n-k+1} U_{:,n-k+1}^T,$$

*without actually forming the matrices.*

This is a nice programming excercise which can be solved using $\cdot$ products.

```
In [59]: function myaverages{T}(λ::T, u::Vector{T})
             n=length(u)
             x=Array{Float64}(2*n-1)
             # Average lower diagonals
             for i=1:n
                 x[i]=dot(u[1:i],reverse(u[1:i]))/i
             end
             for i=2:n
                 x[n+i-1]=dot(u[i:n],reverse(u[i:n]))/(n-i+1)
             end
             λ*x
         end

Out[59]: myaverages (generic function with 1 method)
```

```
In [60]: # A small test
         u=[1,2,3,4,5]
         u*u'

Out[60]: 5×5 Array{Int64,2}:
          1   2   3   4   5
          2   4   6   8  10
          3   6   9  12  15
          4   8  12  16  20
          5  10  15  20  25
```

```
In [61]: myaverages(1,u)

Out[61]: 9-element Array{Float64,1}:
           1.0
           2.0
           3.33333
           5.0
           7.0
          11.0
          15.3333
          20.0
          25.0
```

We now execute the first step of the algorithm from the above Fact.

Notice that `eigs()` returns the eigenvalues arranged by the absoulte value, so the consecutive pairs define the $i$-th signal. The computation of averages is long - it requires $O(n^2)$ operations and takes several minutes.

```
In [62]: # This step takes 7 minutes, so we skip it

         # xcomp=Array(Array{Float64},L)
         # for k=1:L
         #     xcomp[k]=myaverages(λ[2*k-1],U[:,2*k-1])+myaverages(λ[2*k],U[:,2*k])
         # end
```

**Can we do without averaging?**

The function `myaverages()` is very slow - 7 minutes, compared to 5 seconds for the eigenvalue computation.

The simplest option is to disregard the averages, and use the first column and the last row of the underlying matrix, as in definition of Hankel matrices, which we do next. Smarter approach might be to use small random samples to compute the averages.

Let us try the simple approach for the fundamental frequency. (See also the notebook Examples in Signal Decomposition.ipynb.)

```
In [63]: xcomp=Array{Array{Float64}}(L)
         for k=1:L
             k1=2*k-1
             k2=2*k
             xsimple=[(λ[k1]*U[1,k1])*U[:,k1]; (λ[k1]*U[n,k1])*U[2:n,k1]]
             xsimple+=[(λ[k2]*U[1,k2])*U[:,k2]; (λ[k2]*U[n,k2])*U[2:n,k2]]
             xcomp[k]=xsimple
         end
```

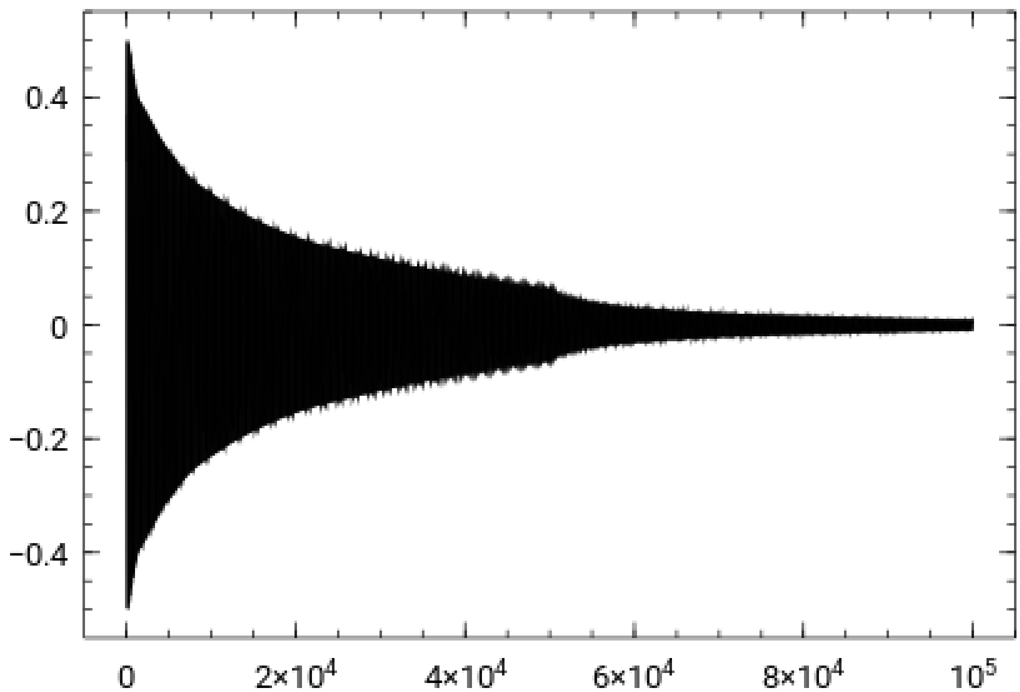
Let us look and listen to what we got:

```
In [64]: typeof(xcomp[1])
```

Out[64]: Array{Float64,1}

In [65]:
```
k=1
plot(xcomp[k])
```

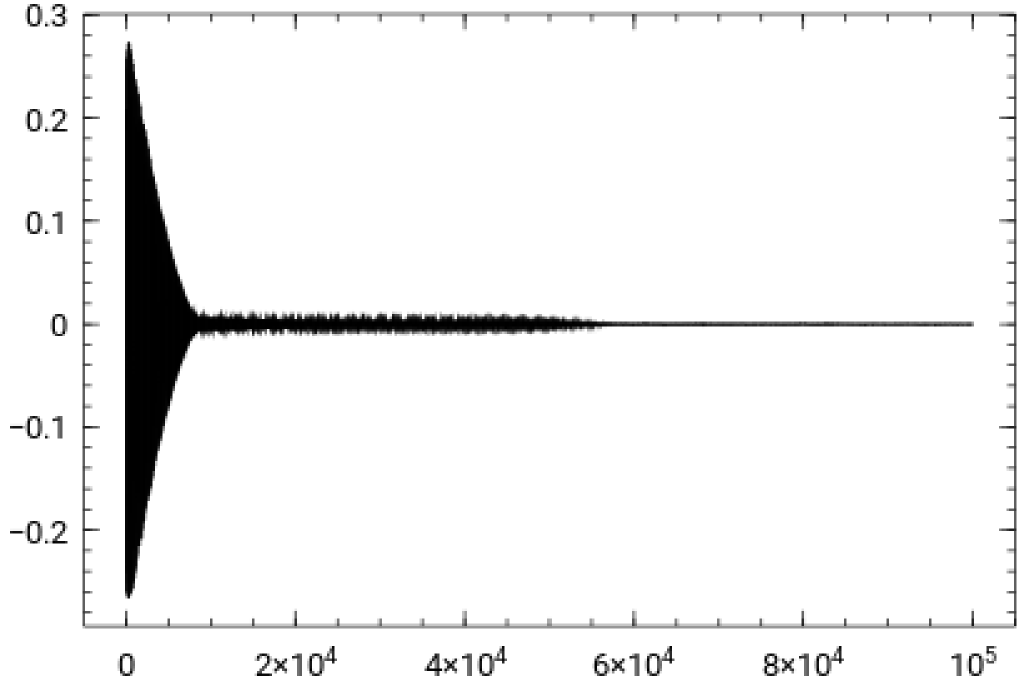
Out[65]:




In [66]:
```
# Entire components
@manipulate for k=1:L
    Winston.plot(xcomp[k])
end
```

Interact.Options{:SelectionSlider,Int64}(5: "input-2" = 6 Int64 , "k", 6, "6", 6, Interact.Optio

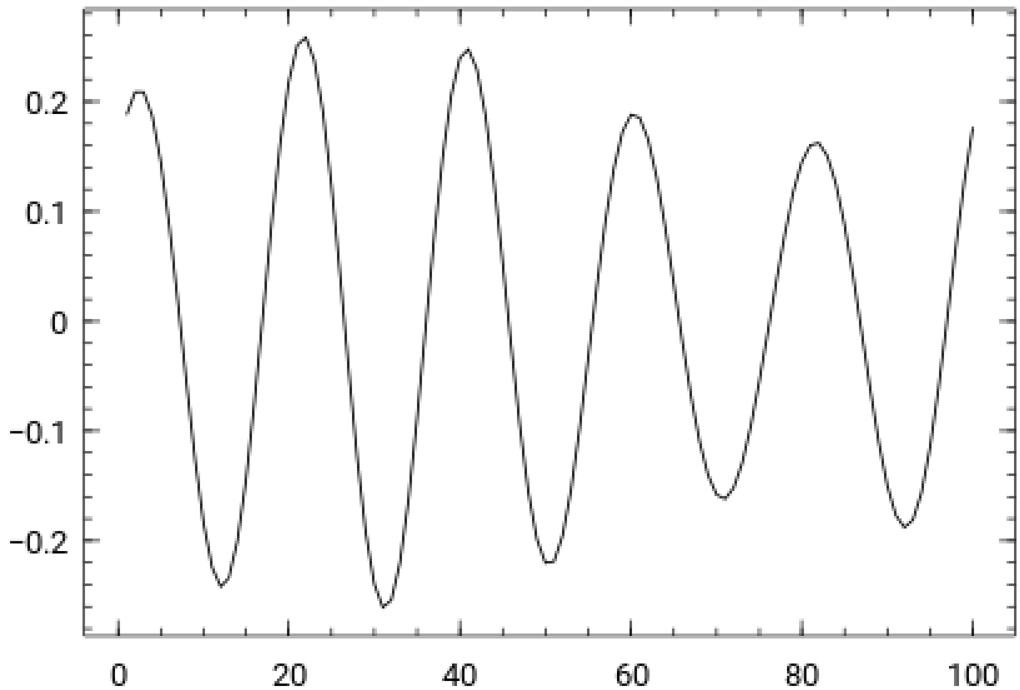
Out[66]:




```
In [67]: # Short parts of components
         @manipulate for k=1:L
             plot(xcomp[k][1:100])
         end
```

```
Interact.Options{:SelectionSlider,Int64}(9: "input-3" = 6 Int64 , "k", 6, "6", 6, Interact.Optio
```

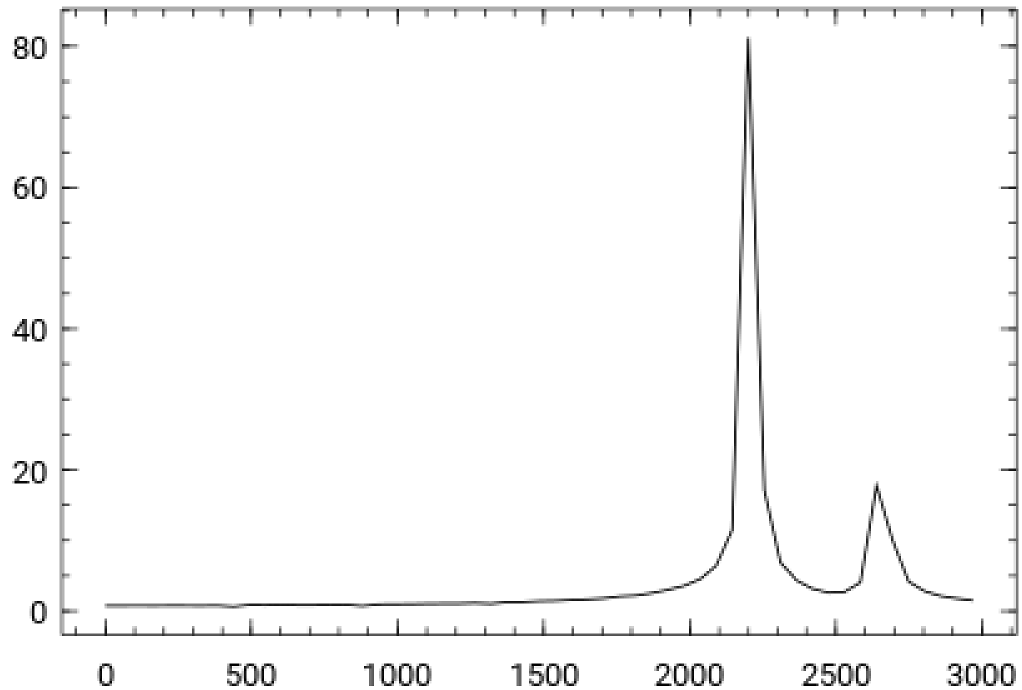
Out[67]:




In [68]:
```
# FFTs of short parts
Fd=110
N=convert(Int32,ceil(Fs/Fd))
xx=collect(0:Fs/(2N):3000)
nn=length(xx)
@manipulate for k=1:L
    plot(xx,abs.(fft(xcomp[k][1:800]))[1:nn])
end
```

```
Interact.Options{:SelectionSlider,Int64}(13: "input-4" = 6 Int64 , "k", 6, "6", 6, Interact.Opti
```

Out[68]:




We see that all `xcomp[k]` are clean mono-components - see Physics of Music - Notes:

```
1 = 440 Hz (A4)
2 = 880 Hz (2*440,+octave,A5)
3 = 1320 Hz (3*440,+quint,E6)
4 = 440 Hz
5 = 880 Hz
6 = 2200 Hz (5*440,++major terza, C#7)
7 = 2640 Hz (6*440,++quint,E7)
8 = 440 Hz
9 = 2200 Hz
10 = 1760 Hz (4*440,++octave,A6)
11 = 2640 Hz
```

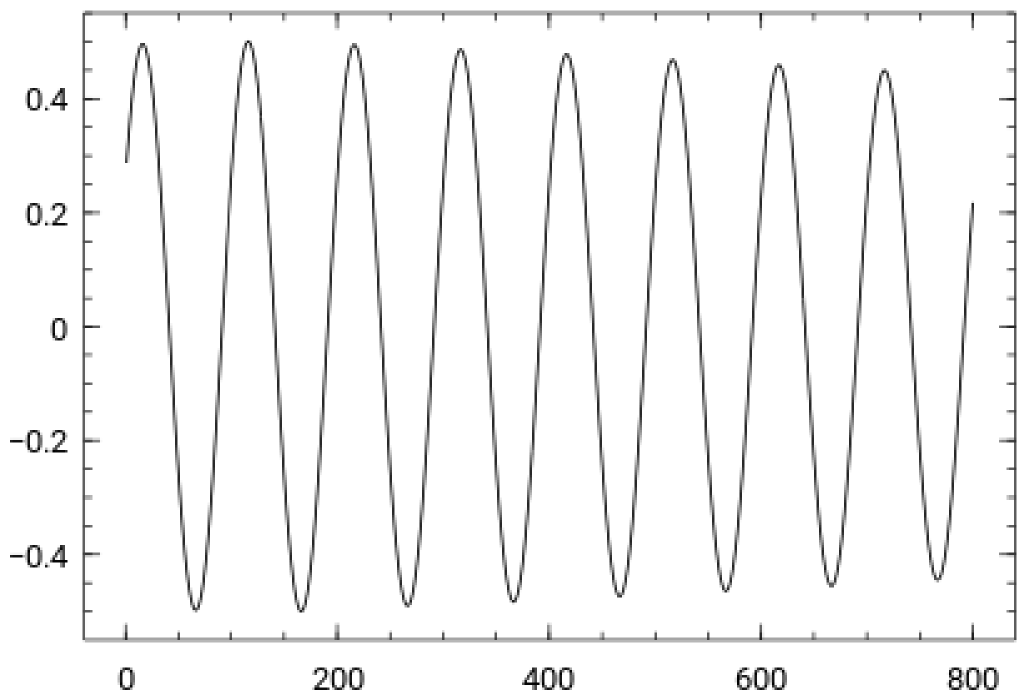
**N.B.** Some mono-components are repeated, and they should be grouped by adding components with absolute weighted correlation larger than some prescribed threshold.

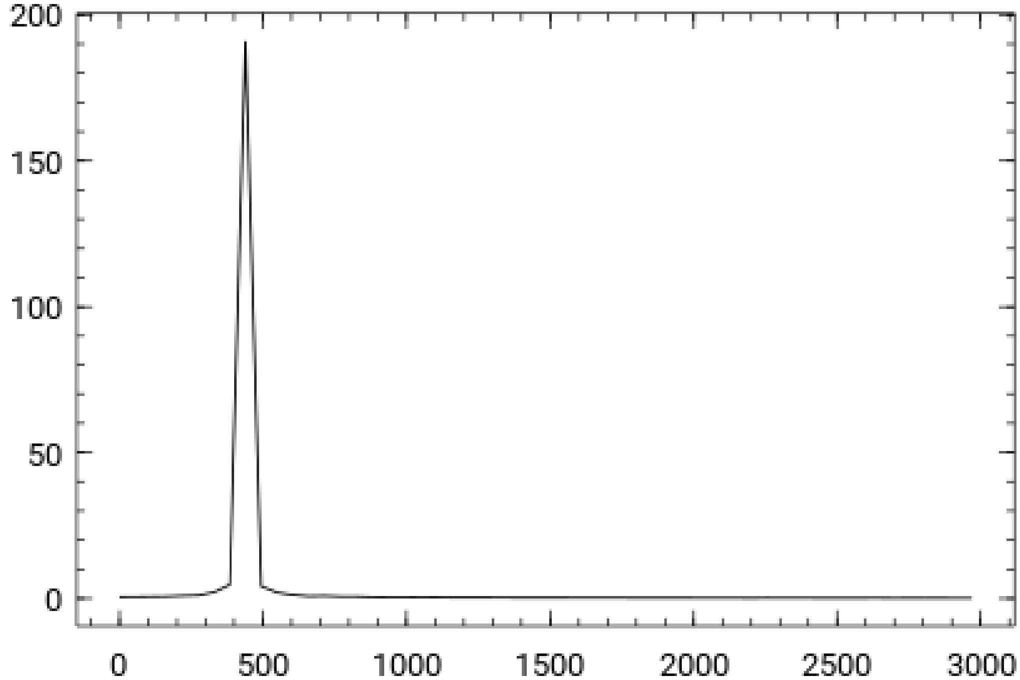
```
In [69]: # Listen to components - wait between k's
         # @manipulate for k in slider(1:L, value=1)
         wavplay(xcomp[1],Fs)
         # end
```

```
In [70]: wavplay(sum([xcomp[i] for i=1:11]),Fs)
```

```
In [71]: # On Windows, store the mono-components
         for i=1:11
             wavwrite(xcomp[i],"files/comp$i.wav",Fs=sig[2])
         end
```

```
In [72]: p=Array{Any}(3)
         xsimple=[(λ[1]*U[1,1])*U[:,1]; (λ[1]*U[n,1])*U[2:n,1]]
         xsimple+=[(λ[2]*U[1,2])*U[:,2]; (λ[2]*U[n,2])*U[2:n,2]]
         p[1]=plot(xsimple[1:800])
         p[2]=plot(xx,abs.(fft(xsimple[1:800]))[1:nn])
         @show norm(xcomp[1]-xsimple)/norm(xcomp[1])
         display(p[1]), display(p[2])
```






```
norm(xcomp[1] - xsimple) / norm(xcomp[1]) = 0.0
```

```
Out[72]: (nothing, nothing)
```