Digital Signal Processing 2/ Advanced Digital Signal Processing Lecture 12, Wiener and Matched Filter, Prediction

Gerald Schuller, TU Ilmenau

Filters to reduce the influence of **noise or distortions.**
Assume: x(n) is the original signal, y(n) is the distorted signal.
Example: y(n)=x(n)+v(n)
where v(n) is is assumed to be independent white noise.

-Wiener filter $h_W(n)$: $y(n) * h_W(n) \rightarrow x(n)$
(**signal fidelity**, the reconstruction is close to the original, for instance for de-noising an image or audio signal, where the audio signal does not need to be deterministic)

-Matched filter $h_M(n)$:
$y(n) * h_M(n) \rightarrow x(n) * h_M(n)$ (no signal fidelity, just **high SNR for detection**, in communication applications, where you would like to detect a 0 or 1, or any given **known** signal, usually a **deterministic signal**; object recognition in images, face recognition).

Wiener Filter

The goal here is an approximation of the original signal x(n) in the least mean squared sense, meaning we would like to **minimize the mean quadratic error** between the filtered and the original signal.

We have a filter system with Wiener Filter $h_W(n)$

$$x(n) = y(n) * h_w(n)$$

meaning we filter our distorted signal y(n) with our still unknown filter $h_W(n)$.

The convolution of $h_W(n)$ (with filter length L) with y(n) can be written as a matrix multiplication:

$$x(n) = \sum_{m=0}^{L-1} y(n-m) \cdot h_W(m)$$

Let's define 2 vectors.

The first is a vector of the the **past L samples of our noisy signal** y, up to the present sample at time n, (bold face font to indicate that it it a vector)

$$\boldsymbol{y}(n) = [y(n-L+1), \dots, y(n)]$$

The next vector contains the **time reversed impulse response**,

$$\boldsymbol{h}_W = [h_W(L-1), \dots, h_W(0)]$$

Using those 2 vectors, we can rewrite our convolution equation above as a vector multiplication,

$$x(n)=\boldsymbol{y}(n)\cdot\boldsymbol{h}_W^T$$

Observe that $\boldsymbol{h}_W$ has no time index because it already contains all the samples of the time-reversed impulse response, and is constant.
We can now also put the output signal x(n) into the column vector,

$$\boldsymbol{x}=[x(0),x(1),\ldots]^T$$

To obtain this column vector, we simply assemble all the row vectors of our noisy signal $\boldsymbol{y}(n)$ into a matrix $\boldsymbol{A}$,

$$\boldsymbol{A}=\begin{bmatrix}\boldsymbol{y}(0)\\ \boldsymbol{y}(1)\\ \vdots\end{bmatrix}$$

With this matrix, we obtain the result of our convolution at all time steps n to

$$\boldsymbol{A}\cdot\boldsymbol{h}_W=\boldsymbol{x}$$

For the example of a filter length of h_W of L=2 hence we get,

$$\begin{bmatrix}y(1) & y(2)\\ y(2) & y(3)\\ y(3) & y(4)\\ \vdots & \vdots\end{bmatrix}\cdot\begin{bmatrix}h_W(2)\\ h_W(1)\end{bmatrix}=\begin{bmatrix}x(1)\\ x(2)\\ x(3)\\ \vdots\end{bmatrix}$$

Observe again that the vector $\boldsymbol{h}_W$ in this equation is the **time reversed** impulse response of our filter. This is now the **matrix multiplication** formulation of our **convolution**.

We can now obtain the minimum mean squared error **solution** of this matrix multiplication using the so-called Moore-Penrose **Pseudo Inverse** (http://en.wikipedia.org/wiki/Moore%E2%80%93Penrose_pseudoinverse)
This pseudo-inverse finds the column vector $\boldsymbol{h}$ which minimizes the distance to a given $\boldsymbol{x}$ with the matrix $\boldsymbol{A}$ (which contains our signal y to be filtered):

$$\boldsymbol{A} \cdot \boldsymbol{h}_W \rightarrow \boldsymbol{x}$$

Matrix $\boldsymbol{A}$ and vector $\boldsymbol{x}$ are known, and vector $\boldsymbol{h}_W$ is unknown.
This problem can be solved exactly if the matrix $\boldsymbol{A}$ is **square and invertible**. Just multiplying the equation with A^{-1} from the left would give us the solution

$$\boldsymbol{h}_W = \boldsymbol{A}^{-1} \cdot \boldsymbol{x}$$

This cannot be done, if $\boldsymbol{A}$ is **non-square,** for instance if it has many more rows than columns. In this case we don't have an exact solution, but many solutions that come close to $\boldsymbol{x}$. We would like to obtain the solution which comes **closest** to $\boldsymbol{x}$ in a mean squared error distance sense (also called **Euclidean Distance**).

This solution is derived using the Pseudo-Inverse:

$$A^T \cdot A \cdot h_W = A^T \cdot x$$

Here, $A^T \cdot A$ is now a square matrix, and usually invertible, such that we obtain our solution

$$h_W = \left(A^T \cdot A\right)^{-1} A^T \cdot x$$

This h_W is now the **solution** we where looking for. This solution has the minimum mean squared distance to the un-noisy version of all solutions.

Octave/Matlab Example for denoising speech:

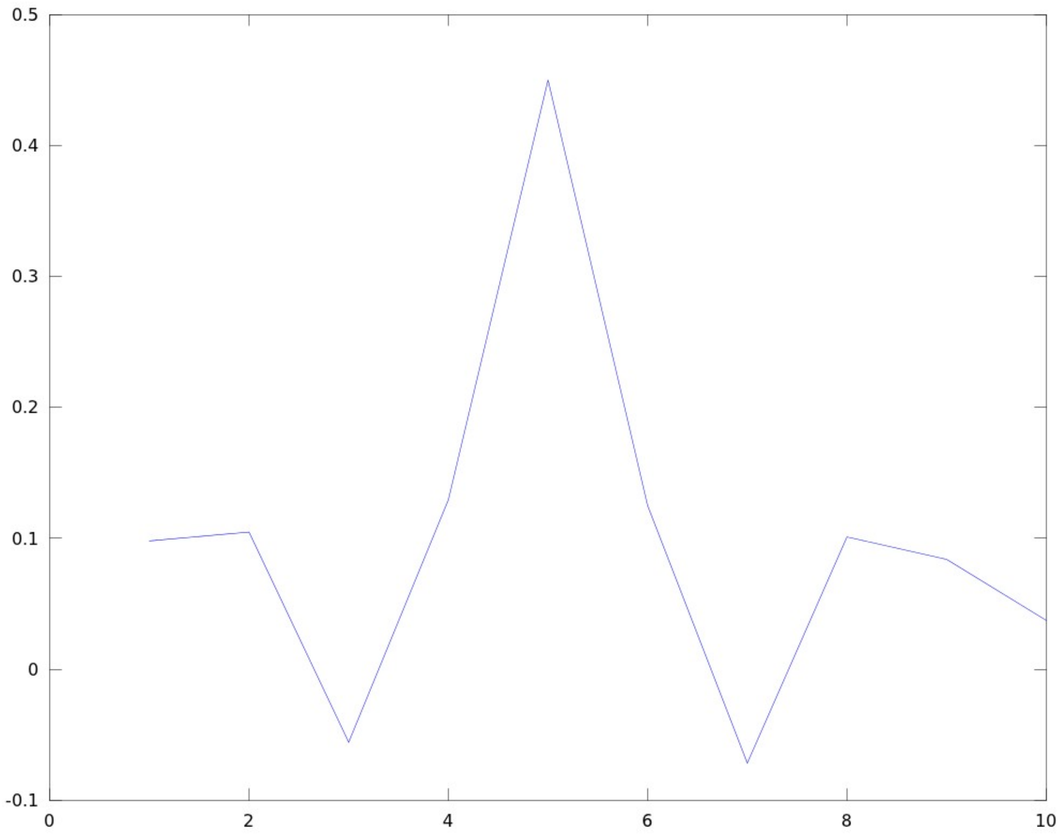
```
x=wavread('fspeech.wav');
sound(x,32000);
%additive zero mean white noise (for
-1<x<1):
y=x+0.1*(rand(size(x))-0.5); %column vector
sound(y,32000);
%we assume 10 coefficients for our Wiener filter.
%10 to 12 is a good number for speech signals.
A=zeros(100000,10);
for m=1:100000,
  A(m,:)=y(m-1+(1:10))';
end;

%Our matrix has 100000 rows and 10 colums:
size(A)
```

```
%ans =
%100000 10
```

%Compute Wiener Filter:

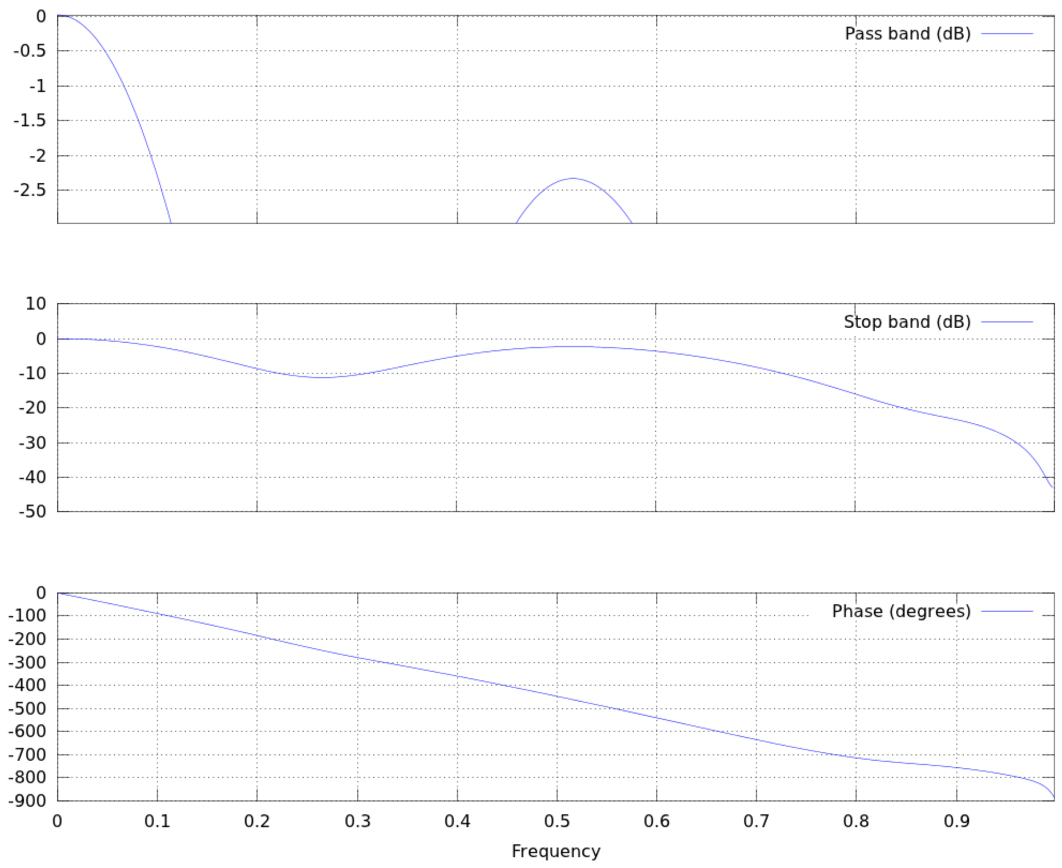
```
%Trick: allow (flipped) filter delay of 5
%samples to get better working denoising.
%The desired signal hence is x(6:100005):
h=inv(A'*A) * A' * x(6:100005)
plot(flipud(h))
```



Observe that for this non-flipped impulse response we see a delay of 4 samples, since the first sample corresponds to a delay of zero, and the location of biggest sample, which corresponds to the signal delay, is at delay 5-

1=4.

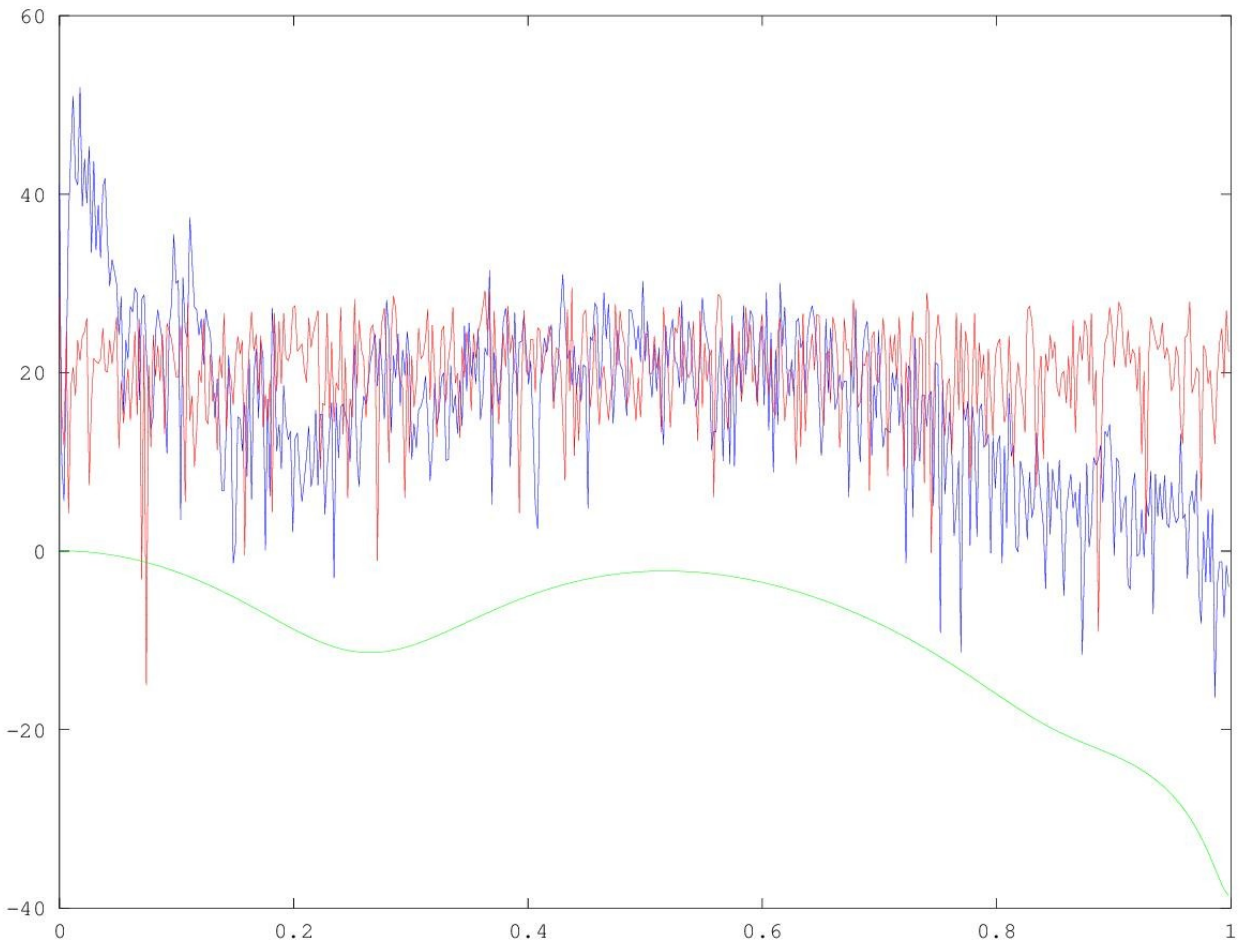
Its frequency response is
freqz(flipud(h))



Here we can see that the resulting filter has a somewhat **low pass characteristic**, because our speech signal has energy mostly at low frequencies. At high frequencies we have mostly noise, hence it makes sense to have more attenuation there! This attenuation curve of this Wiener filter also has some similarity to the speech spectrum on can observe with *freqz(x)*. If we look at the spectrum of the white noise with *freqz(0.1*(rand(size(x))-0.5))*, then

we see that at low frequencies speech is domianting, and at high frequencies, noise is dominating, and we need to remove or attenuate that latter part of the spectrum.
We can plot the spectra of the speech and the noise together with:

```
Hspeech=freqz(x);
Hnoise=freqz(0.1*(rand(size(x))-0.5));
Hw=freqz(flipud(h));
plot((0:511)/512,20*log10(abs(Hspeech)));
hold
plot((0:511)/512,20*log10(abs(Hnoise)),'r');
plot((0:511)/512,20*log10(abs(Hw)),'g');
```



The speech spectrum is blue, the noise

spectrum is red, and as a comparison the Wiener filter transfer function is green. Here we see that **speech dominates the spectrum only at low and middle frequencies**, noise at the other frequencies, hence it makes sense to suppress those noisy frequencies.

Now we can filter and listen to it:

```
xw=filter(flipud(h),1,y);
sound(xw,32000)
```

We can hear that the signal now sounds more “muffled”, the higher frequencies are indeed attenuated, which reduces the influence of the noise. But it is still a question if it actually “sounds” better to the human ear.

Let's compare the mean quadratic error. For the noisy signal it is

```
size(x)
%ans =
%207612 1
sum((y(1:200000)-x(1:200000)).^ 2)/200000
%ans = 8.3499e-04
```

For the Wiener filtered signal it is (taking into account 4 samples delay from our filter (5 from the end from flipping), beginning to peak).

```
sum((xw(4+(1:200000))-x(1:200000)).^
2)/200000
```

```
ans = 3.4732e-04
```

We can see that the mean quadratic error is indeed less than half as much as for the noisy version y(n)!

Let's take a look at the matrix $A^T \cdot A$ which we used in the computation,

```
>>> A'*A
ans =
Columns 1 through 8:
1011.69 859.44 806.14 842.55 837.94 797.05 774.33
745.8
1
859.44 1011.69 859.44 806.14 842.55 837.94 797.05
774.3
3
806.14 859.44 1011.69 859.44 806.14 842.55 837.94
797.0
5
842.55 806.14 859.44 1011.69 859.44 806.14 842.55
837.9
4
837.94 842.55 806.14 859.44 1011.69 859.44 806.14
842.5
5
797.05 837.94 842.55 806.14 859.44 1011.69 859.44
806.1
4
774.33 797.05 837.94 842.55 806.14 859.44 1011.69
859.4
4
745.81 774.33 797.05 837.94 842.55 806.14 859.44
1011.6
9
713.58 745.81 774.33 797.05 837.94 842.55 806.14
859.4
4
693.25 713.58 745.81 774.33 797.05 837.94 842.55
```

```
806.1
4
Columns 9 and 10:
713.58 693.25
745.81 713.58
774.33 745.81
797.05 774.33
837.94 797.05
842.55 837.94
806.14 842.55
859.44 806.14
1011.69 859.44
859.44 1011.69
```

We can see that it is a 10x10 matrix in our example for a Wiener filter with 10 filter taps. In this matrix, the next row looks almost like the previous line, but (circularly) shifted by 1 sample to the right.

Observe that in general this matrix $\boldsymbol{A}^T \cdot \boldsymbol{A}$ **converges** to the **autocorrelation matrix** of signal $y(n)$ if the length of the signal in the matrix goes to infinity!

$$\boldsymbol{A}^T \cdot \boldsymbol{A} \rightarrow \boldsymbol{R}_{yy} = \begin{bmatrix} r_{yy}(0) & r_{yy}(1) & r_{yy}(2) & \ldots \\ r_{yy}(1) & r_{yy}(0) & r_{yy}(1) & \ldots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

(The autocorrelation of signal y is $r_{yy}(m) = \sum_{n=-\infty}^{\infty} y(n) \cdot y(n+m)$)

Since one row of this matrix is the shifted by one sample version of the one above, it is called a “**Toeplitz Matrix**”.

The expression $\boldsymbol{A}^T \cdot \boldsymbol{x}$ in our formulation of the

Wiener filter becomes the cross correlation vector

$$\boldsymbol{A}^T \cdot \boldsymbol{x} \rightarrow \boldsymbol{r}_{xy} = \begin{bmatrix} r_{xy}(0) \\ r_{xy}(1) \\ \vdots \end{bmatrix}$$

(where the cross correlation function is

$r_{xy}(m) = \sum_{n=-\infty}^{\infty} y(n) \cdot x(n+m)$)

Observe: In the receiver we usually don't have the un-noisy signal x(n), but we can **estimate** the above **cross correlation** function.

Hence our expression for the Wiener filter $\boldsymbol{h} = (\boldsymbol{A}^T \cdot \boldsymbol{A})^{-1} \boldsymbol{A}^T \cdot \boldsymbol{x}$ becomes

$$\boldsymbol{h} = (\boldsymbol{R}_{yy})^{-1} \boldsymbol{r}_{xy}$$

This matrix form is also called the "**Yule-Walker** equation", and the general statistical formulation is called the "**Wiener-Hopf** equation".

*See also: *M.H. Hayes, "Statistical Signal Processing and Modelling", Wiley.*

This general statistical formulation now also has the advantage, that we can design a Wiener Filter by just **knowing the statistics**, the **auto-correlation function** of our noisy signal, and the **cross-correlation function** of

our noisy and original signal. Observe that this auto-correlation and cross-correlation can also be obtained from the **power-spectra** (cross-power spectra, the product of the 2 spectra) of the respective signals.
The power spectrum of the noisy signal can usually be measured, since it is the signal to filter, and the **spectrum of the original signal** x(n) usually has to be **estimated** (using assumptions about it).
For instance, we know typical speech spectra. If we want to adapt a Wiener filter in a receiver, we take this typical speech spectrum, and measure the noise level at the receiver. That is sufficient to compute the Wiener filter coefficients, using above formulations.