Computational Statistics with Application to Bioinformatics

Prof. William H. Press
Spring Term, 2008
The University of Texas at Austin

Unit 19: Wiener Filtering (and some Wavelets)

Unit 19: Wiener Filtering (and some Wavelets) (Summary)

- Wiener filtering is a general way of finding the best reconstruction of a noisy signal (in L^2 norm)
  - applies in any orthogonal function basis
  - different bases give different results
    - best when the basis most cleanly separates signal from noise
- The "universal" Wiener filter is to multiply components by $S^2/(S^2+N^2)$
  - smooth tapering of noisy components towards zero
- In Fourier basis, the Wiener filter is an optimal low-pass filter
  - learn how the frequencies of an FFT are arranged!
  - this is useful in many signal processing applications
  - but for images, it loses spatial resolution
- In spatial (pixel) basis, the Wiener filter is usually applied to the difference between an image and a smoothed image
- In a wavelet basis, components encode both spatial and frequency information
  - largest magnitude components often have the least noise
  - so Wiener filtering can preserve resolution
- There are two key ideas behind wavelets (such as the DAUB family)
  - quadature mirror filters
  - pyramidal algorithm

Wiener Filtering

Norbert Wiener
1894 - 1964

This general idea can be applied whenever you have a basis in function space that concentrates "mostly signal" in some components relative to "mostly noise" in others.

You *could* just set components with too much noise to zero.

Wiener filtering is better: it gives the optimal way of tapering off the noisy components, so as to give the best (L^2 norm) reconstruction of the original signal.

Can be applied in spatial basis (delta functions, or pixels), Fourier basis (frequency components), wavelet basis, etc.

Different bases are not equivalent, because, in particular problems, signal and noise distribute differently in them. A lot of signal processing is finding the right basis for particular problems – in which signal is most concentrated.

(For simplicity, I'm going to write out the equations as if in a finite-dimensional space, but think infinite dimensional.)

You measure components of signal plus noise

$$C_i = S_i + N_i$$

Let's look for a signal estimator that simply scales the individual components of what is measured

$$\widehat{S}_i = C_i \Phi_i$$

$$\text{find the } \Phi_i\text{s that minimize } \left\langle \, |\widehat{\mathbf{S}} - \mathbf{S}|^2 \, \right\rangle$$

Here is where we use the fact that we are in some orthogonal basis, so the L^2 norm is just the sum of squares of components:

expectations come inside and land on the only two stochastic things, signal and noise

$$\left\langle \, (\widehat{\mathbf{S}} - \mathbf{S}) \cdot (\widehat{\mathbf{S}} - \mathbf{S}) \, \right\rangle = \left\langle \sum_i \left[(S_i + N_i)\Phi_i - S_i \right]^2 \right\rangle$$

$$= \sum_i \left\{ \left\langle S_i^2 \right\rangle (1 - \Phi_i)^2 + \left\langle N_i^2 \right\rangle \Phi_i^2 \right\} - 2 \sum_i \Phi_i \left\langle N_i S_i \right\rangle$$

Differentiate w.r.t. Φ and set to zero, giving

this is zero!

$$\Phi_i = \frac{\langle S_i^2 \rangle}{\langle S_i^2 \rangle + \langle N_i^2 \rangle}$$

This is the Wiener filter. It requires estimates of the signal and noise power in each component.

Let's demonstrate in some different bases on this image:

```
IN = fopen('image-face.raw','r');
face = flipud(reshape(fread(IN),256,256)');
fclose(IN);
bwcolormap = [0:1/256:1; 0:1/256:1; 0:1/256:1]';
image(face)
colormap(bwcolormap);
axis('equal')
```

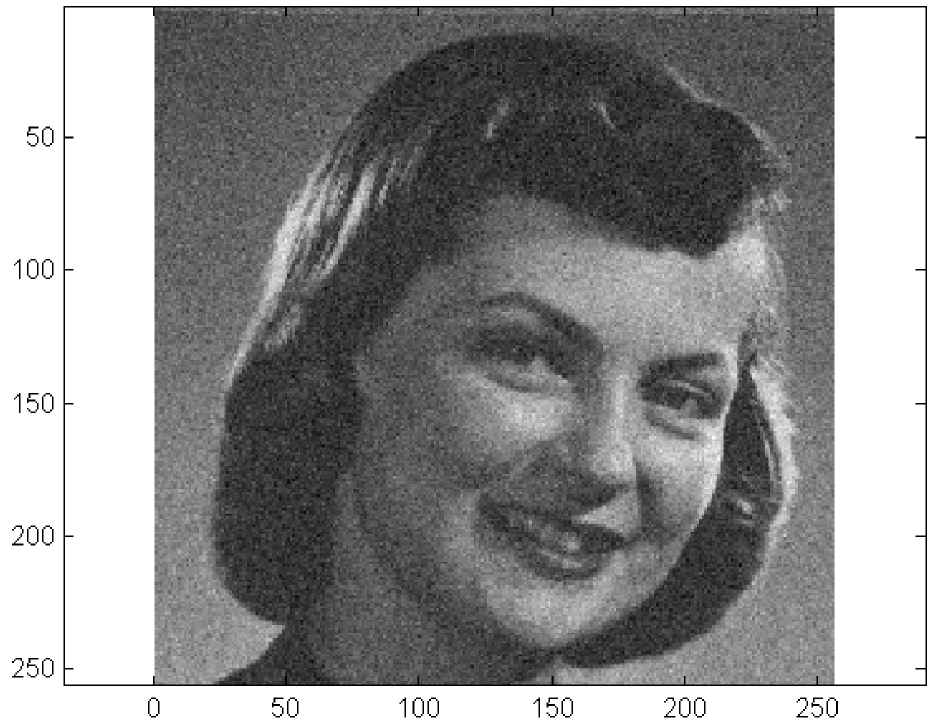
the file is unformatted bytes, so you read it with fopen and fread

(Favorite demo image in NR. Very retro, it's a Kodak test photo from the 1950s, shows film grain and other defects.)

Add noise (here, Gaussian white noise):

```
noisyface = face + 20*randn(256,256);
noisyface = 255 * (noisyface - min(noisyface(:)))/(max(noisyface(:))-min(noisyface(:)));
image(noisyface)
colormap(bwcolormap);
axis('equal')
```

Have to rescale, because noise takes it out of 0-255.
Note reduced contrast resulting.



First example: **Fourier basis**

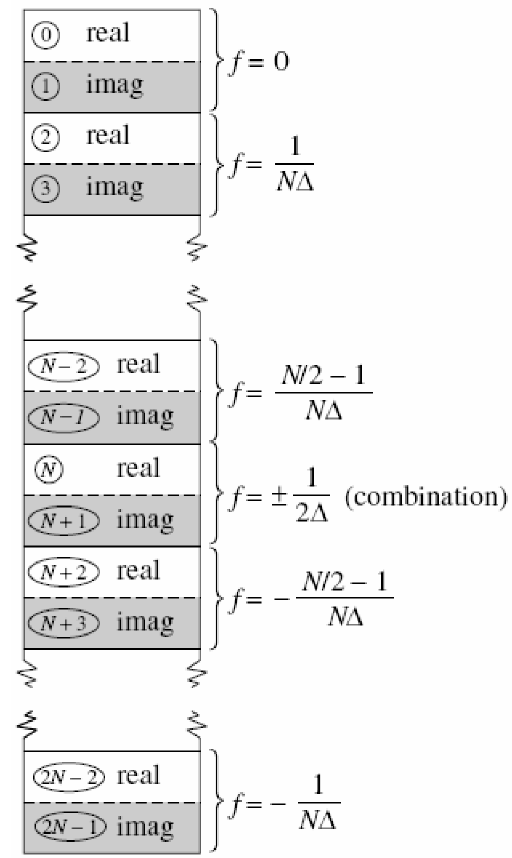
This will be a “low pass filter” using the fact that the signal is concentrated at low spatial frequencies, while the noise is white (flat).

Actually, Fourier is not a great basis for de-noising most images, since low-pass will reduce the resolution of the picture (blur it) along with de-noising.

1 1 1… 1 2 3…
2 2 2… 1 2 3…
3 3 3… 1 2 3…

```
fftface = fft2(noisyface);

[row col] = ndgrid(1:256);
ftable = [0:127,128:-1:1];
freqs = 0.5 * sqrt((ftable(row).^2+ftable(col).^2)/(2*128^2));
```

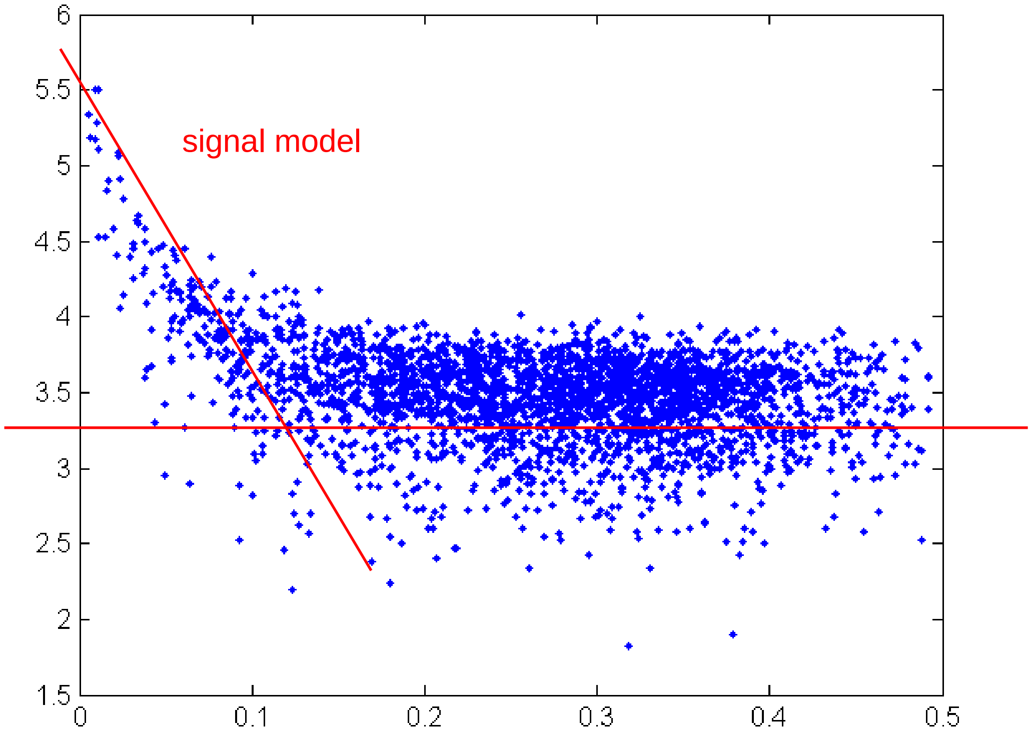
Nyquist frequency



Yes, we can see a separation between signal and noise:

```
samp = randsample(256*256,3000);
plot(freqs(samp),log10(abs(fftface(samp))),'.')
```

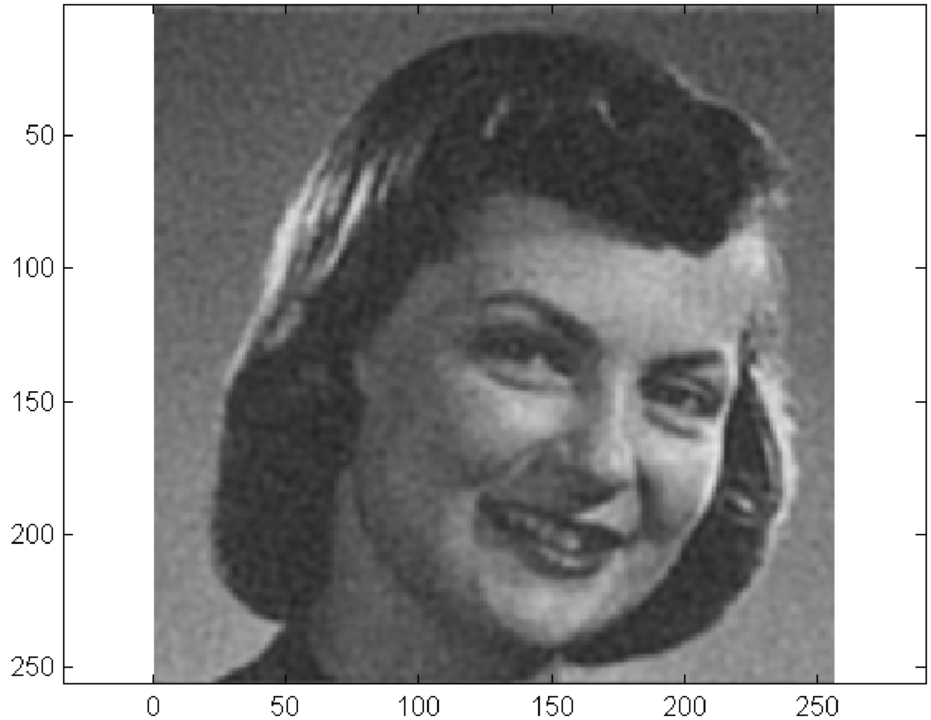
abs is here the complex modulus



Why the line a bit low?
I played around with it to make the picture look better!

So my reconstruction is not exactly best least squares reconstruction, but will be less blurred.

de-noised

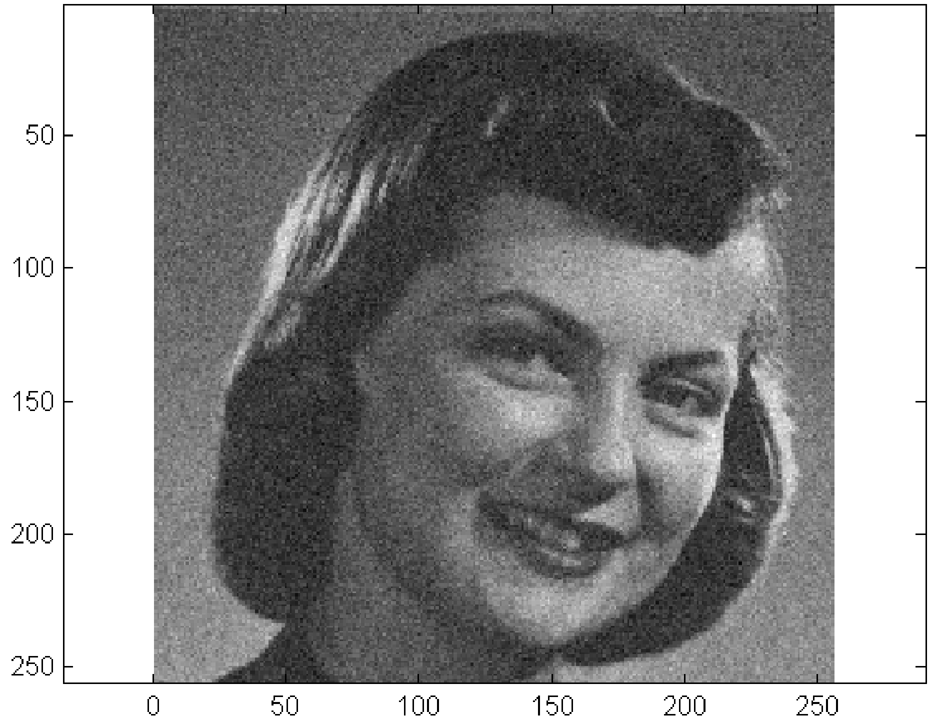


```
sig = 10.^(5.5 - 15 .* freqs);
noi = 10.^3.2;
fftfiltface = fftface .* (sig.^2 ./ (sig.^2 + noi.^2));
reface = ifft2(fftfiltface,'symmetric');
image(reface)
colormap(bwcolormap);
axis('equal')
```

I just read the constants off by eye from the previous chart

note square, to get power

this tells Matlab that you intend the inverse FFT to be real-valued noisy



Actually, you might prefer the noisy image, because your brain has good algorithms for adaptively smoothing! But it is a less accurate representation of the original photo in L^2 norm!

Second example: **spatial (pixel) basis**

It doesn't make sense to use the pure pixel basis, because there is no particular separation of signal and noise separately in each pixel.

But a closely related method is to decompose the image into a smoothed background image, and then to take deviations from this as estimating signal power + noise power:

$$X_{ij} = \frac{1}{N_{\text{hood}}} \sum_{\text{hood}} x_{ij}$$

"hood" might be a 5x5 neighborhood centered on each point

$$\left\langle S_{ij}^2 + N_{ij}^2 \right\rangle = \frac{1}{N_{\text{hood}}} \sum_{\text{hood}} (x_{ij} - X_{ij})^2$$

this is the Wiener part

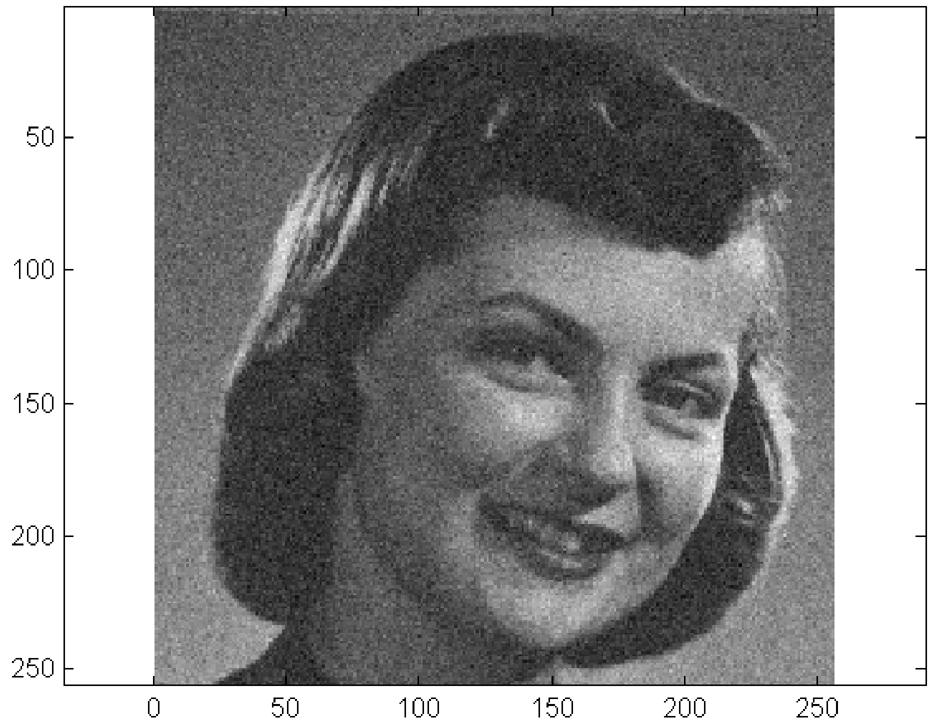
$$\widehat{x}_{ij} = X_{ij} + \frac{\left\langle S_{ij}^2 + N_{ij}^2 \right\rangle - \left\langle N_{ij}^2 \right\rangle}{\left\langle S_{ij}^2 + N_{ij}^2 \right\rangle} (x_{ij} - X_{ij})$$

and let the user adjust $\left\langle N_{ij}^2 \right\rangle$ as a parameter.

Matlab has a function for this called wiener2


```
wiener = wiener2(noisyface,[5,5]);
image(wiener)
colormap(bwcolormap);
axis('equal')
```

you can put your noise estimate as another argument, or you can let Matlab estimate it as some kind of heuristic minimum of values seen for S^2+N^2 over the image noisy



Most fun of all is the **wavelet basis**.
<u>You don't even have to know what it is</u>, except that it is an (orthogonal) rotation in function space, as is the Fourier transform.
(Its basis is localized both in space and in scale.)

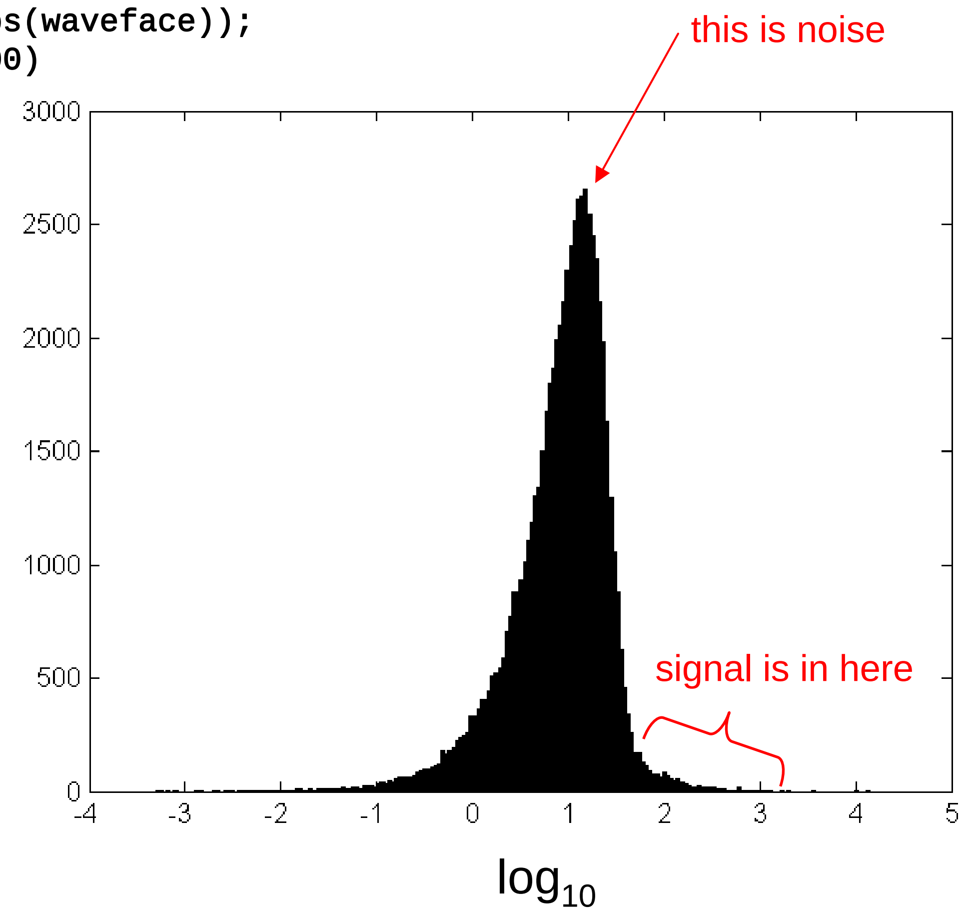
Matlab has a Wavelet Toolbox which I find completely incomprehensible! (I'm sure it's only me with this problem.) So, I'll do a mexfunction wrapper of the NR3 wavelet transform.

```
#include "nr3_matlab.h"
#include "wavelet.h“

/* Matlab usage:
          outmatrix = wavelet2(inmatrix,isign)
*/

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[]) {
          MatDoub ain(prhs[0]);
          VecInt dims(2);
          Int mm=(dims[0]=ain.nrows()),nn=(dims[1]=ain.ncols());
          Int isign = Int(mxScalar<Doub>(prhs[1]));
          Int i,mn = mm*nn;
          Daub4 daub4;
          if (nrhs != 2 || nlhs != 1) throw("wavelet2.cpp: bad number of args");
          if ((nn & (nn-1)) != 0 || (mm & (mm-1)) != 0)
                    throw("wavelet2.cpp: matrix sizes must be power of 2");
          VecDoub a(mn);
          for (i=0;i<mn;i++) a[i] = (&ain[0][0])[i];
          wtn(a,dims,isign,daub4);
          MatDoub aout(mm,nn,plhs[0]);
          for (i=0;i<mn;i++) (&aout[0][0])[i] = a[i];
          return;
}
```

this is the whole point,
the NR3 wavelet transform

Take the wavelet transform and look at the magnitude of the components on a log scale:

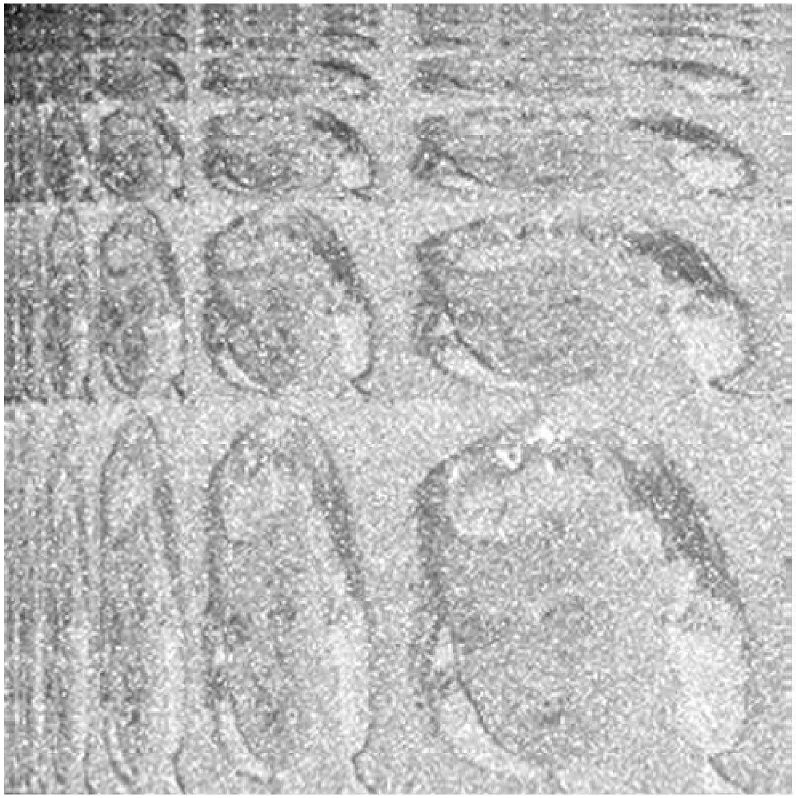
```
waveface = wavelet2(noisyface,1);
dist = log10(abs(waveface));
hist(dist(:),200)
```



Notice the difference in philosophy from Fourier: There we used frequency ("which component") to estimate S and N. Here we use the magnitude of the component directly , without regard to which component it is.

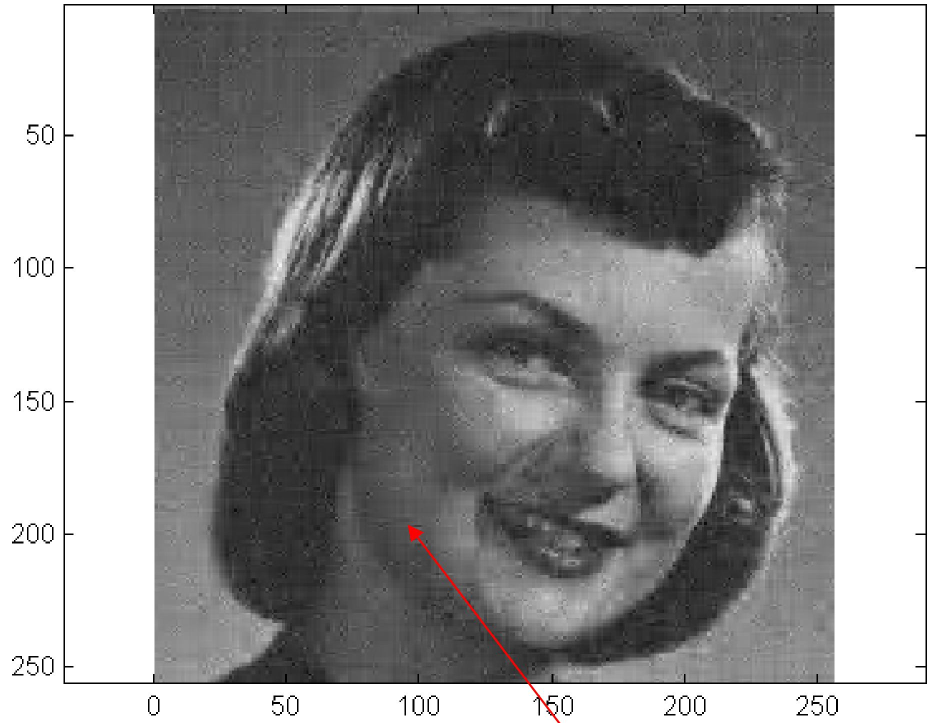
If you fiddle around with mapping the gray scale (zero point, contrast, etc.) of the matrix "`waveface`" you can see how the wavelet basis works

low resolution information is in this corner



high resolution information is in this corner

Truncate-to-zero components with magnitude less than 30.
This is not a true Wiener filter, because it doesn't roll off smoothly.




```
fwaveface = waveface;
fwaveface(abs(waveface)<30) = 0.;
werecface = wavelet2(fwaveface,-1);
image(werecface)
colormap(bwcolormap)
axis('equal')
```

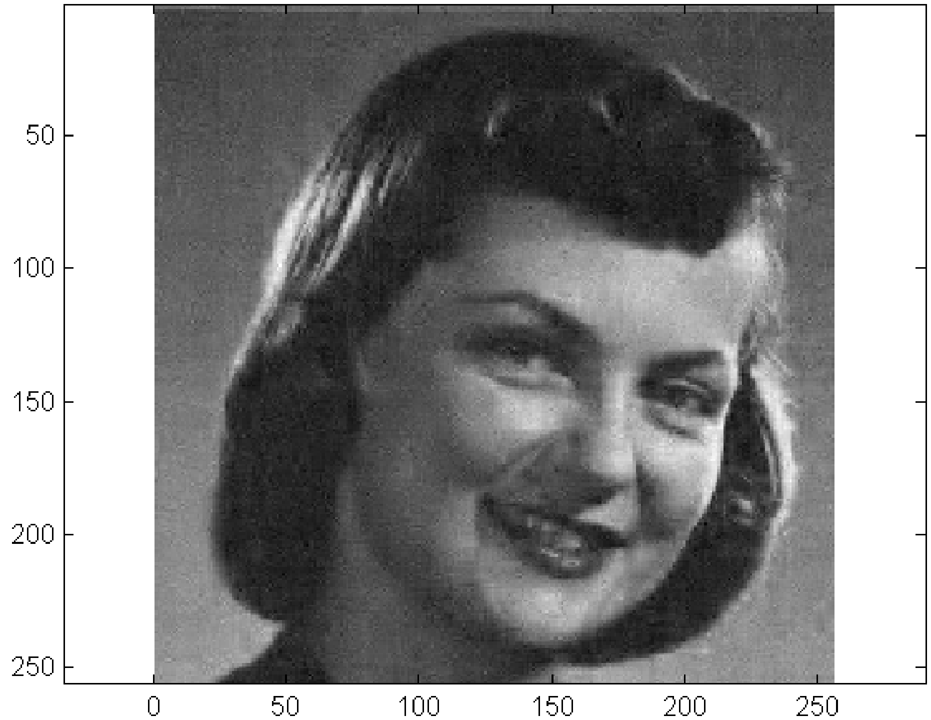
Notice the "wavelet plaid" in the image. You sometimes see this on digital TV, because MPEG4 uses wavelets for still texture coding.

Compare to Wiener filter (smooth roll-off)


```
fwaveface = waveface .* (waveface .^ 2 ./ (waveface.^2 + 900));
werecface = wavelet2(fwaveface,-1);
image(werecface)
colormap(bwcolormap)
axis('equal')
```

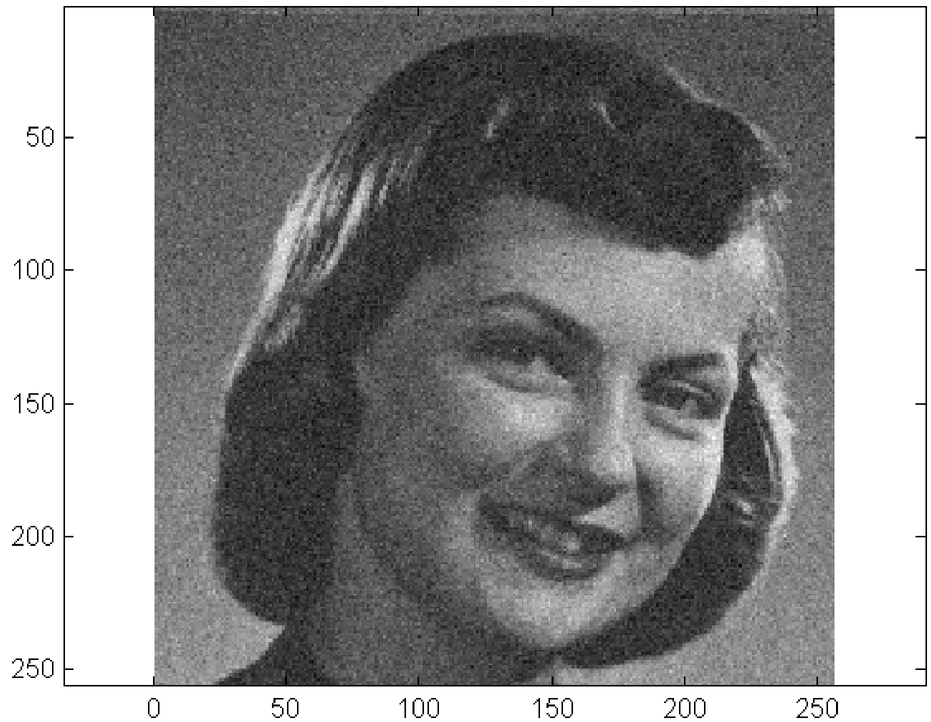
i.e., noise amplitude 30 in the previous histogram

Even better if we restore the contrast




```
werecface = 255*(werecface - min(werecface(:)))/(max(werecface(:))-min(werecface(:)));
image(werecface)
colormap(bwcolormap)
axis('equal')
```

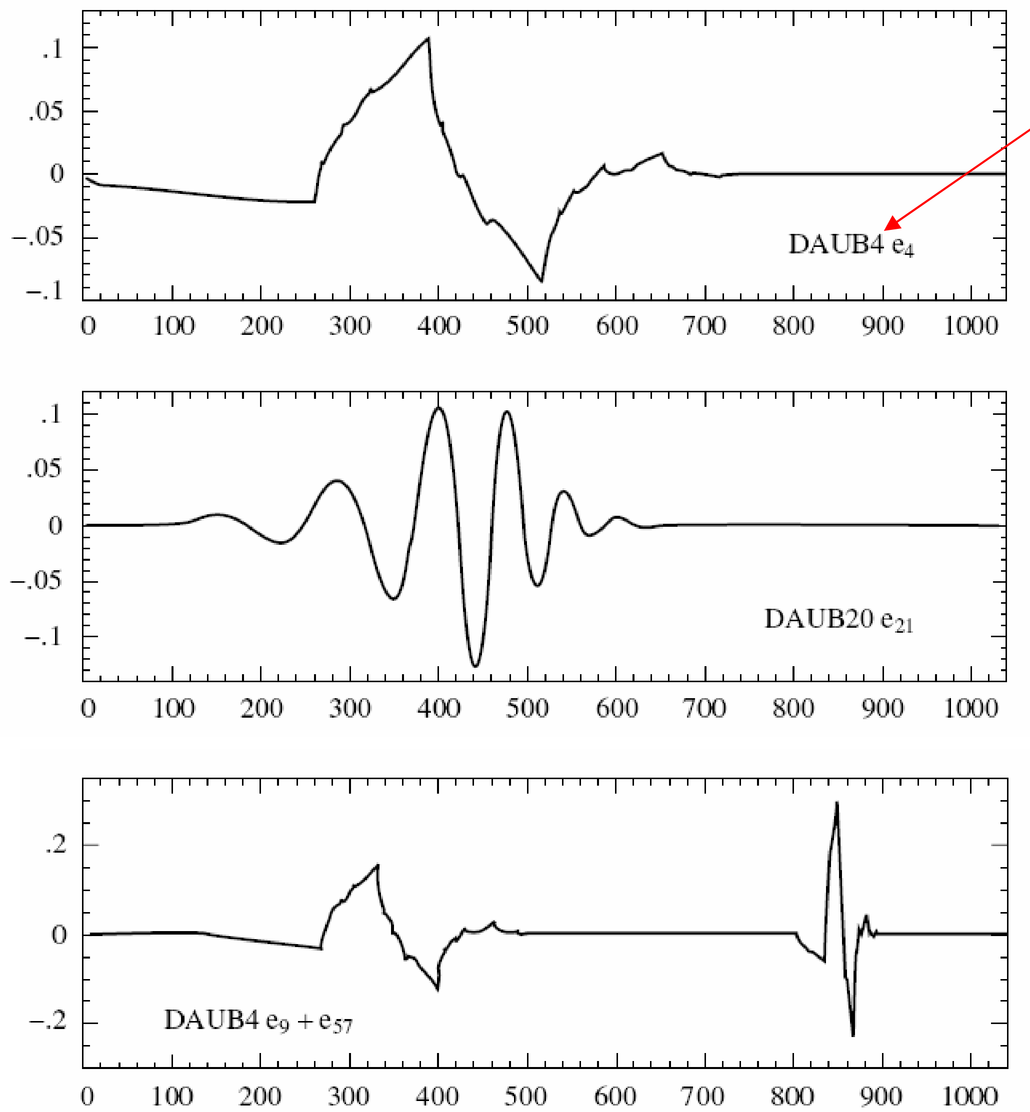
Compare to what we started with (noisy)



Of course, we can never get the original back:
information is truly lost in the presence of noise

The moral about Wiener filters is that they work in any basis, but are better in some than in others. That is what signal processing is all about!

Want to see some wavelets? Where do they come from?



The "DAUB" wavelets are named after Ingrid Daubechies, who discovered them.

(This is like getting the sine function named after you!)

So who is the sine function named after? it's the literal translation into Latin, ca. 1500s, of the corres-pondng mathematical concept in Arabic, in which language the works of Hipparchus (~150 BC) and Ptolemy (~100 AD) were preserved. The tangent function wasn't invented until the 9th Century, in Persia.

The first key idea in wavelets ("quadrature mirror filter") is to find an orthogonal transformation that separates "smooth" from "detail" information. We illustrate in the 1-D case.

$$
\begin{bmatrix}
c_0 & c_1 & c_2 & c_3 & & & & & & & \\
c_3 & -c_2 & c_1 & -c_0 & & & & & & & \\
 & & c_0 & c_1 & c_2 & c_3 & & & & & \\
 & & c_3 & -c_2 & c_1 & -c_0 & & & & & \\
\vdots & \vdots & & & & & \ddots & & & & \\
 & & & & & & & c_0 & c_1 & c_2 & c_3 \\
 & & & & & & & c_3 & -c_2 & c_1 & -c_0 \\
c_2 & c_3 & & & & & & & & c_0 & c_1 \\
c_1 & -c_0 & & & & & & & & c_3 & -c_2
\end{bmatrix}
$$

smooth average of 4 sequential components

not-smooth linear combination

transpose is

$$
\begin{bmatrix}
c_0 & c_3 & & & \cdots & & & & & c_2 & c_1 \\
c_1 & -c_2 & & & \cdots & & & & & c_3 & -c_0 \\
c_2 & c_1 & c_0 & c_3 & & & & & & & \\
c_3 & -c_0 & c_1 & -c_2 & & & & & & & \\
 & & & & \ddots & & & & & & \\
 & & & & & c_2 & c_1 & c_0 & c_3 & & \\
 & & & & & c_3 & -c_0 & c_1 & -c_2 & & \\
 & & & & & & & c_2 & c_1 & c_0 & c_3 \\
 & & & & & & & c_3 & -c_0 & c_1 & -c_2
\end{bmatrix}
$$

implying orthogonality conditions

$$
\begin{aligned}
c_0^2 + c_1^2 + c_2^2 + c_3^2 &= 1 \\
c_2 c_0 + c_3 c_1 &= 0
\end{aligned}
$$

these are two conditions on 4 unknowns, so we get to impose two more conditions

Choose the extra two conditions to make the not-smooth linear combination have zero response to smooth functions. That is, make its lowest moments vanish:

$$c_3 - c_2 + c_1 - c_0 = 0$$ no response to a constant function

$$0c_3 - 1c_2 + 2c_1 - 3c_0 = 0$$ no response to a linear function

The unique solution is now

$$c_0 = (1+\sqrt{3})/4\sqrt{2} \qquad c_1 = (3+\sqrt{3})/4\sqrt{2}$$
$$c_2 = (3-\sqrt{3})/4\sqrt{2} \qquad c_3 = (1-\sqrt{3})/4\sqrt{2}$$

"the DAUB4 wavelet coefficients"

If we had started with a wider-banded matrix we could have gotten higher order Daubechies wavelets (more zeroed moments), e.g., DAUB6:

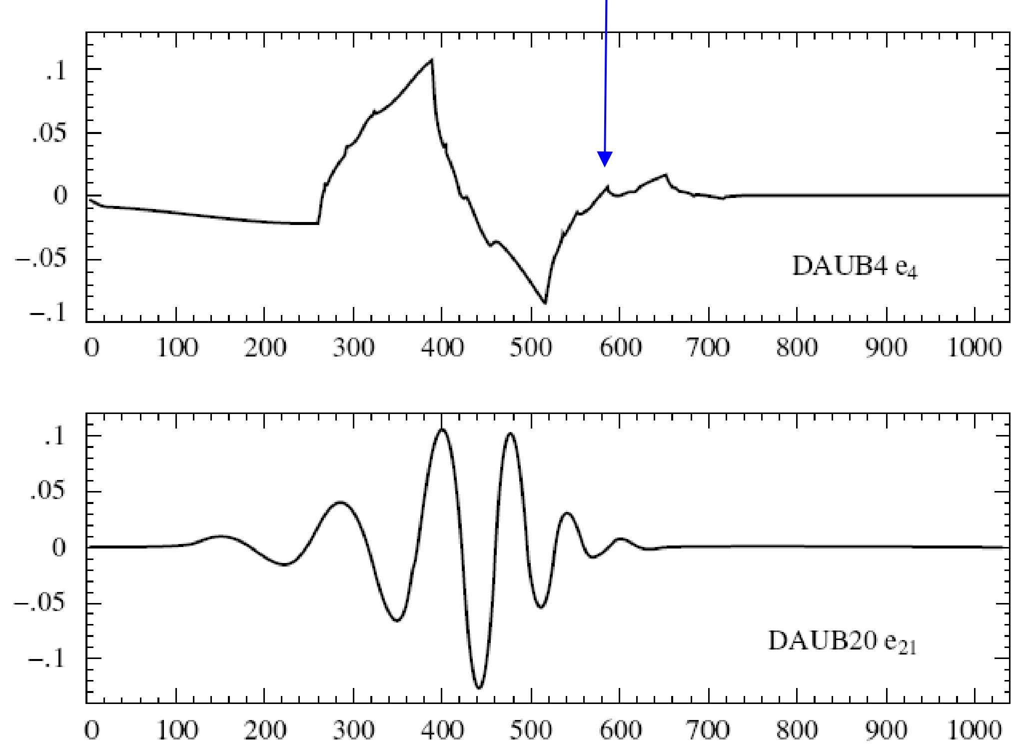
$$c_0 = \left(1+\sqrt{10}+\sqrt{5+2\sqrt{10}}\right)/16\sqrt{2} \qquad c_1 = \left(5+\sqrt{10}+3\sqrt{5+2\sqrt{10}}\right)/16\sqrt{2}$$
$$c_2 = \left(10-2\sqrt{10}+2\sqrt{5+2\sqrt{10}}\right)/16\sqrt{2} \qquad c_3 = \left(10-2\sqrt{10}-2\sqrt{5+2\sqrt{10}}\right)/16\sqrt{2}$$
$$c_4 = \left(5+\sqrt{10}-3\sqrt{5+2\sqrt{10}}\right)/16\sqrt{2} \qquad c_5 = \left(1+\sqrt{10}-\sqrt{5+2\sqrt{10}}\right)/16\sqrt{2}$$

The second key idea in wavelets is to apply the orthogonal matrix multiple times, hierarchically. This is called the pyramidal algorithm.

$$
\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \\ y_9 \\ y_{10} \\ y_{11} \\ y_{12} \\ y_{13} \\ y_{14} \\ y_{15} \end{bmatrix}
\xrightarrow{13.10.1}
\begin{bmatrix} s_0 \\ d_0 \\ s_1 \\ d_1 \\ s_2 \\ d_2 \\ s_3 \\ d_3 \\ s_4 \\ d_4 \\ s_5 \\ d_5 \\ s_6 \\ d_6 \\ s_7 \\ d_7 \end{bmatrix}
\xrightarrow{\text{permute}}
\left[\begin{array}{c} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \\ \hline d_0 \\ d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ d_7 \end{array}\right]
\xrightarrow{13.10.1}
\left[\begin{array}{c} S_0 \\ D_0 \\ S_1 \\ D_1 \\ S_2 \\ D_2 \\ S_3 \\ D_3 \\ \hline d_0 \\ d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ d_7 \end{array}\right]
\xrightarrow{\text{permute}}
\left[\begin{array}{c} S_0 \\ S_1 \\ S_2 \\ S_3 \\ \hline D_0 \\ D_1 \\ D_2 \\ D_3 \\ \hline d_0 \\ d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ d_7 \end{array}\right]
\xrightarrow{\text{etc.}}
\left[\begin{array}{c} \mathcal{S}_0 \\ \mathcal{S}_1 \\ \hline \mathcal{D}_0 \\ \mathcal{D}_1 \\ \hline D_0 \\ D_1 \\ D_2 \\ D_3 \\ \hline d_0 \\ d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ d_7 \end{array}\right]
$$

Since each step is an orthogonal rotation (either in the full space or in a subspace), the whole thing is still an orthogonal rotation in function space.

the cusps are really there: DAUB4 has no right-derivative at values $p/2^n$, for integer p and n



Higher DAUBs gain about half a degree of continuity per 2 more coefficients. But not exactly half. The actual orders of regularity are irrational!

Continuity of the wavelet is not the same as continuity of the representation. DAUB4 represents piecewise linear functions exactly, e.g. But the cusps do show up in <u>truncated</u> representations as “wavelet plaid”.