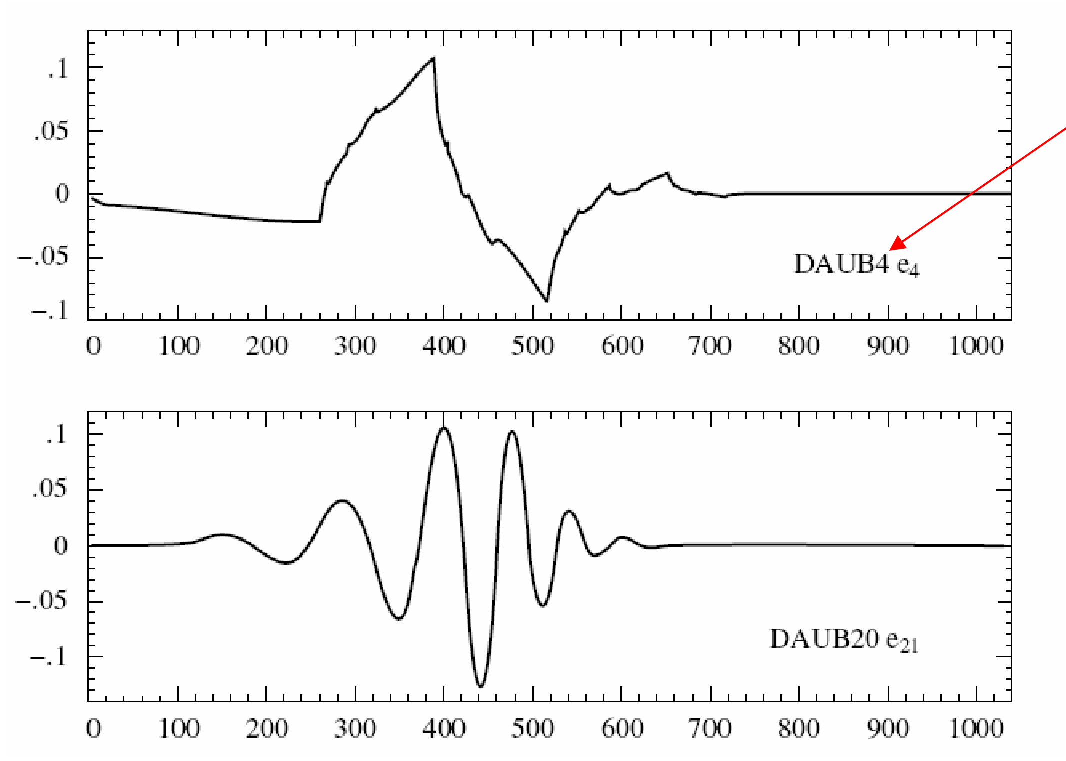
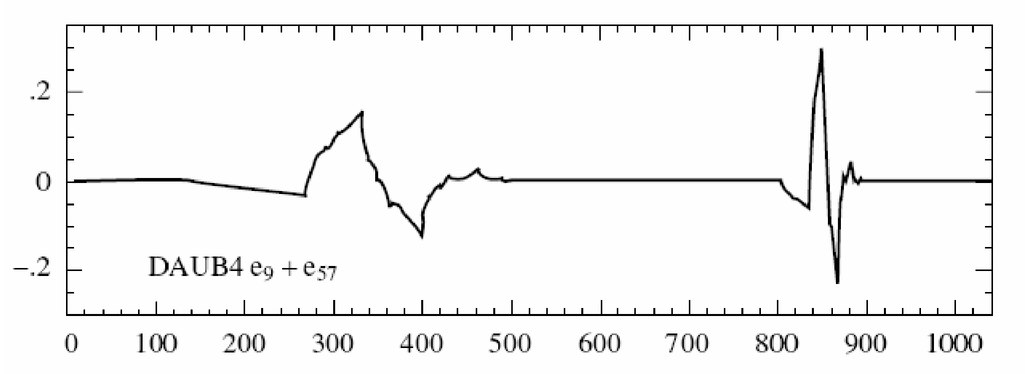
CS395T
Computational Statistics with Application to Bioinformatics

Prof. William H. Press
Spring Term, 2011
The University of Texas at Austin

Lecture 26

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 <u>is</u> the sine function named after? it's the literal translation into Latin, ca. 1500s, of the corres-ponding 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

$$c_0^2 + c_1^2 + c_2^2 + c_3^2 = 1$$
$$c_2 c_0 + c_3 c_1 = 0$$

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:

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

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

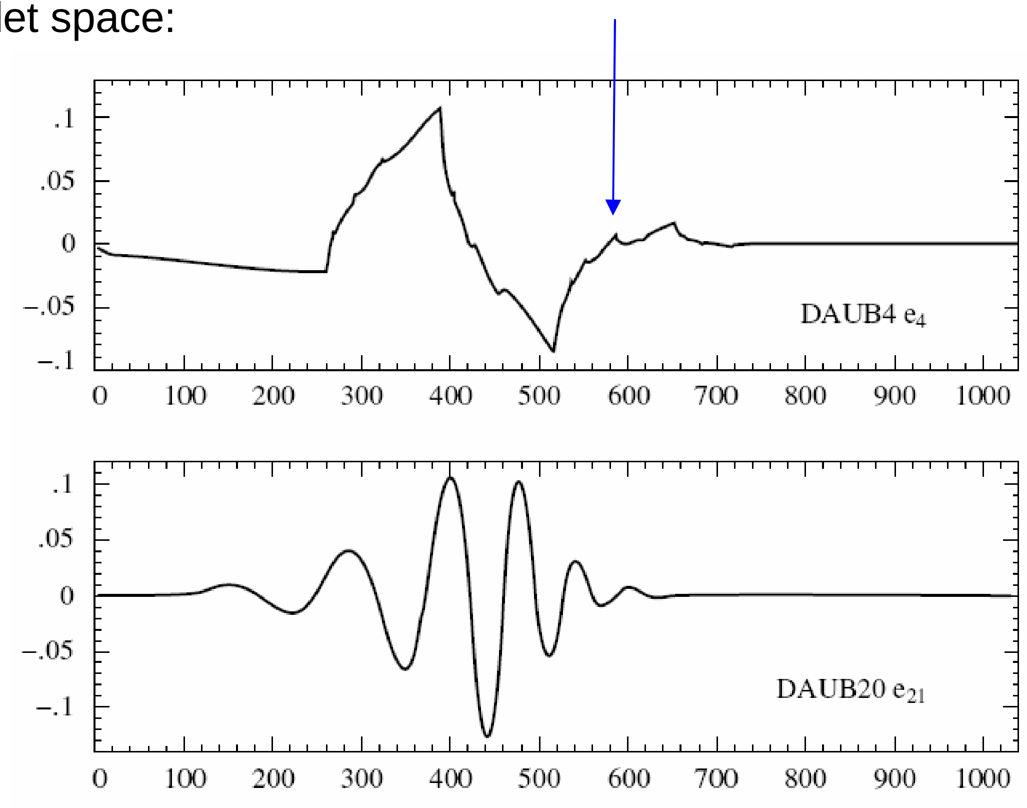
$$\begin{bmatrix} y_0\\ y_1\\ y_2\\ y_3\\ y_4\\ y_5\\ y_6\\ y_7\\ y_8\\ y_9\\ y_{10}\\ y_{11}\\ y_{12}\\ y_{13}\\ y_{14}\\ y_{15} \end{bmatrix} \xrightarrow{\text{QMF}} \begin{bmatrix} s_0\\ d_0\\ s_1\\ d_1\\ s_2\\ d_2\\ s_3\\ d_3\\ s_4\\ d_4\\ s_5\\ d_5\\ s_6\\ d_6\\ s_7\\ d_7 \end{bmatrix} \xrightarrow{\text{permute}}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

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.

For multi-dimensional wavelet transforms, you separately transform each dimension, in any order. (Same procedure as multi-dimensional Fourier transform.)

We can see individual wavelets by taking the inverse transform of unit vectors in wavelet 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”.

That’s all for wavelets!

Laplace Interpolation is a specialized interpolation method for restoring missing data on a grid. It's simple, but sometimes works astonishingly well.

Mean value theorem for solutions of Laplace's equation (harmonic functions):

If y satisfies $\nabla^2 y = 0$ in any number of dimensions, then for any sphere not intersecting a boundary condition,

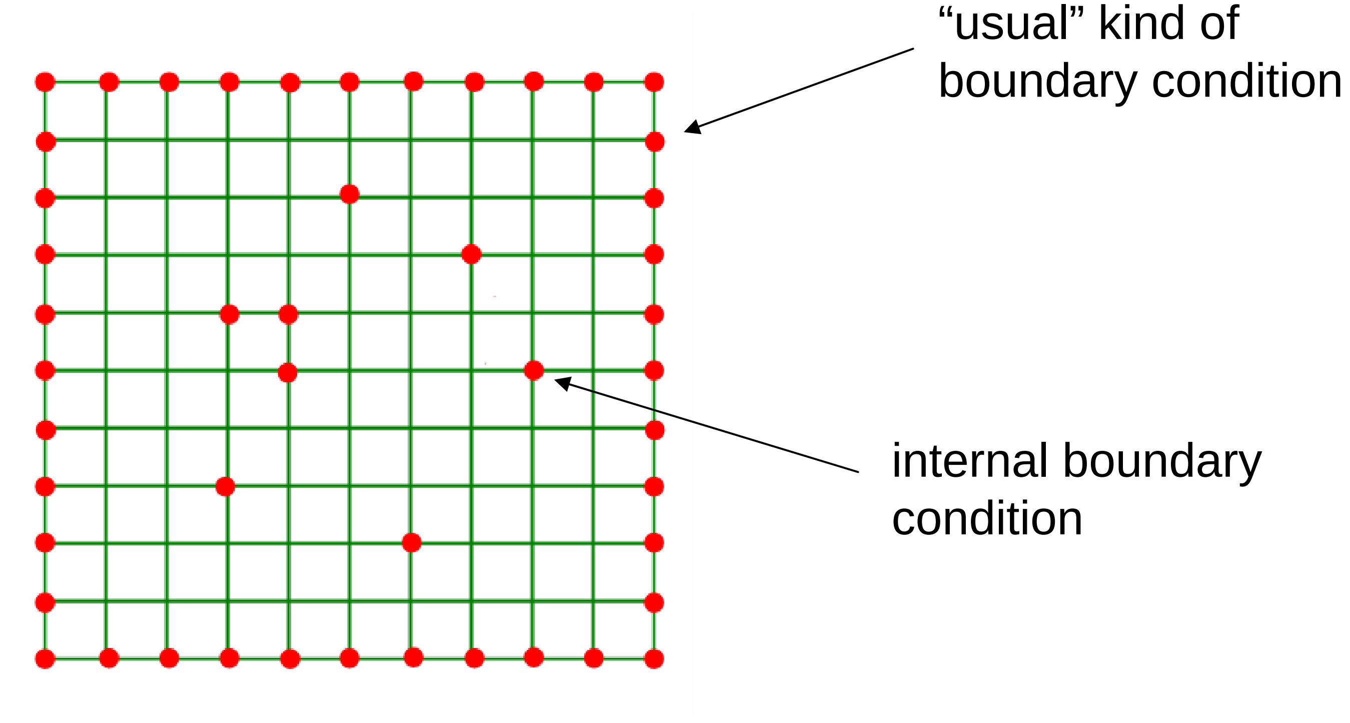
$$\frac{1}{\text{area}} \int_{\text{surface }\omega} y \, d\omega = y(\text{center})$$

So Laplace's equation is, in some sense, the perfect interpolator. It also turns out to be the one that minimizes the integrated square of the gradient,

$$\int_\Omega |\nabla y|^2 \, d\Omega$$

So the basic idea of Laplace interpolation is to set $y(\mathbf{x}_i) = y_i$ at every known data point, and solve $\nabla^2 y = 0$ at every unknown point.

You may not be used to thinking of Laplace's equation as allowing isolated internal boundary conditions. But it of course does!


"usual" kind of boundary condition
internal boundary condition

values are fixed on red dots

Lots of linear equations (one for each grid point)!

$$y_0 = y_{0(\text{measured})}$$

generic equation for a known point

$$y_0 - \frac{1}{4}y_u - \frac{1}{4}y_d - \frac{1}{4}y_l - \frac{1}{4}y_r = 0$$

generic equation for an unknown point
note that this is basically the mean value theorem

lots of special cases:

$$y_0 - \frac{1}{2}y_u - \frac{1}{2}y_d = 0 \qquad \text{(left and right boundaries)}$$

$$y_0 - \frac{1}{2}y_l - \frac{1}{2}y_r = 0 \qquad \text{(top and bottom boundaries)}$$

$$y_0 - \frac{1}{2}y_r - \frac{1}{2}y_d = 0 \qquad \text{(top-left corner)}$$

$$y_0 - \frac{1}{2}y_l - \frac{1}{2}y_d = 0 \qquad \text{(top-right corner)}$$

$$y_0 - \frac{1}{2}y_r - \frac{1}{2}y_u = 0 \qquad \text{(bottom-left corner)}$$

$$y_0 - \frac{1}{2}y_l - \frac{1}{2}y_u = 0 \qquad \text{(bottom-right corner)}$$

There is exactly one equation for each grid point, so we can solve this as a giant (sparse!) linear system, e.g., by the bi-conjugate gradient method.

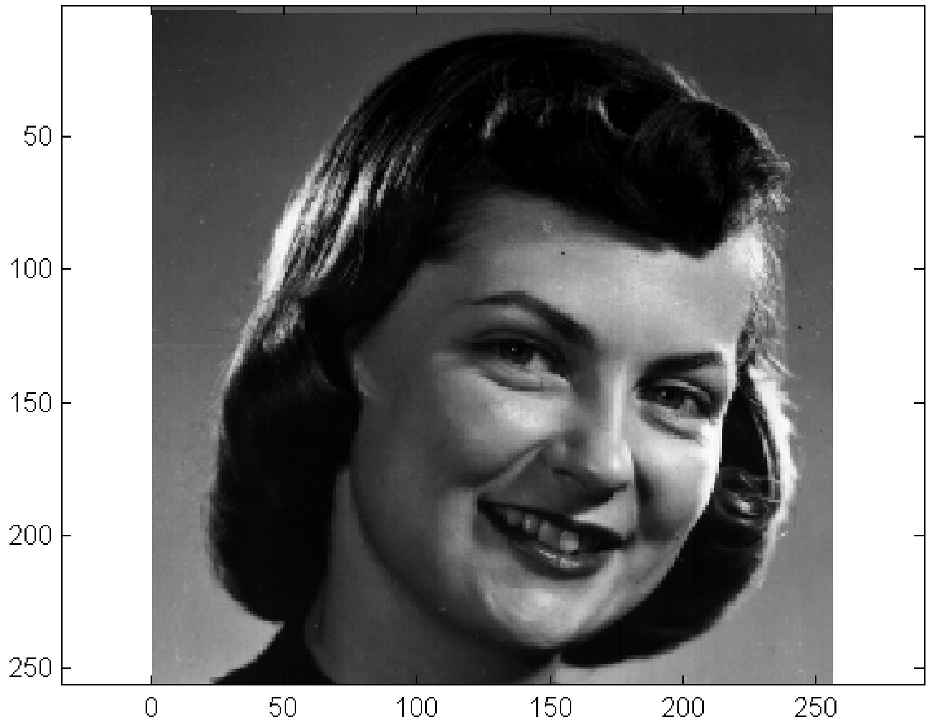
Surprise! It’s in NR3, as Laplace_interp, using Linbcg for the solution.

Easy to embed in a mex function for Matlab

```
#include "..\nr3_matlab.h"
#include "linbcg.h"
#include "interp_laplace.h“

/* Usage:
          outmatrix = laplaceinterp(inmatrix)
*/

Laplace_interp *mylap = NULL;

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[]) {
     if (nrhs != 1 || nlhs != 1) throw("laplaceinterp.cpp: bad number of args");
     MatDoub ain(prhs[0]);
     MatDoub aout(ain.nrows(),ain.ncols(),plhs[0]);
     aout = ain; // matrix op
     mylap = new Laplace_interp(aout);
     mylap->solve();
     delete mylap;
     return;
}
```

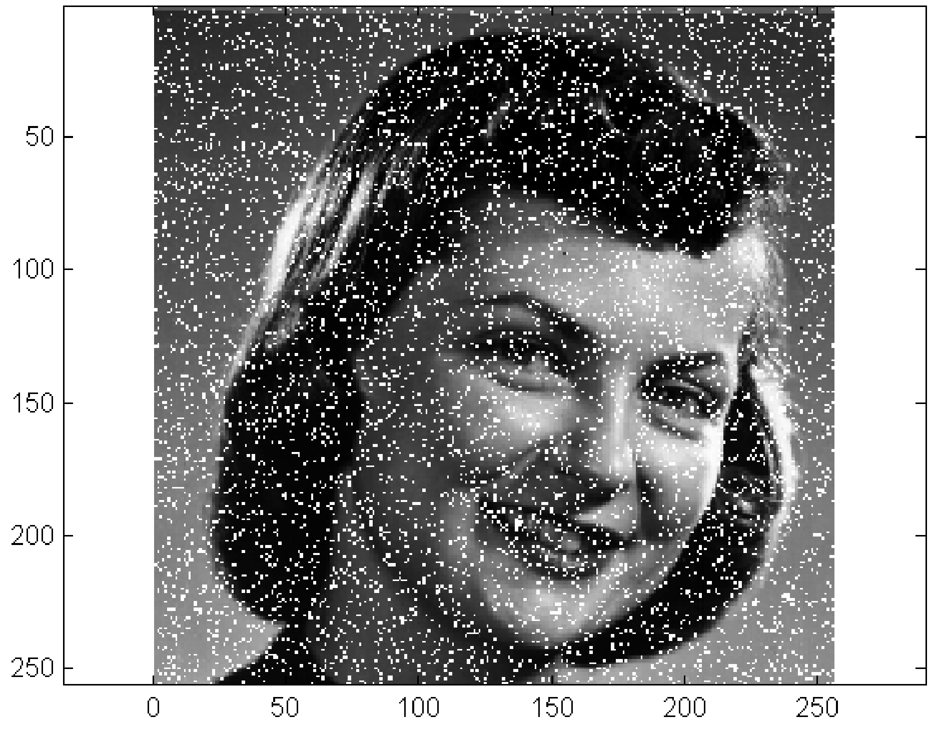
Let’s try it on our favorite face for filtering
(But this is interpolation, not filtering: there is no noise!)

```
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')
```



```
facemiss = face;
ranface = rand(size(face));
facemiss(ranface < 0.1) = 255;
image(facemiss)
colormap(bwcolormap)
axis('equal')
```

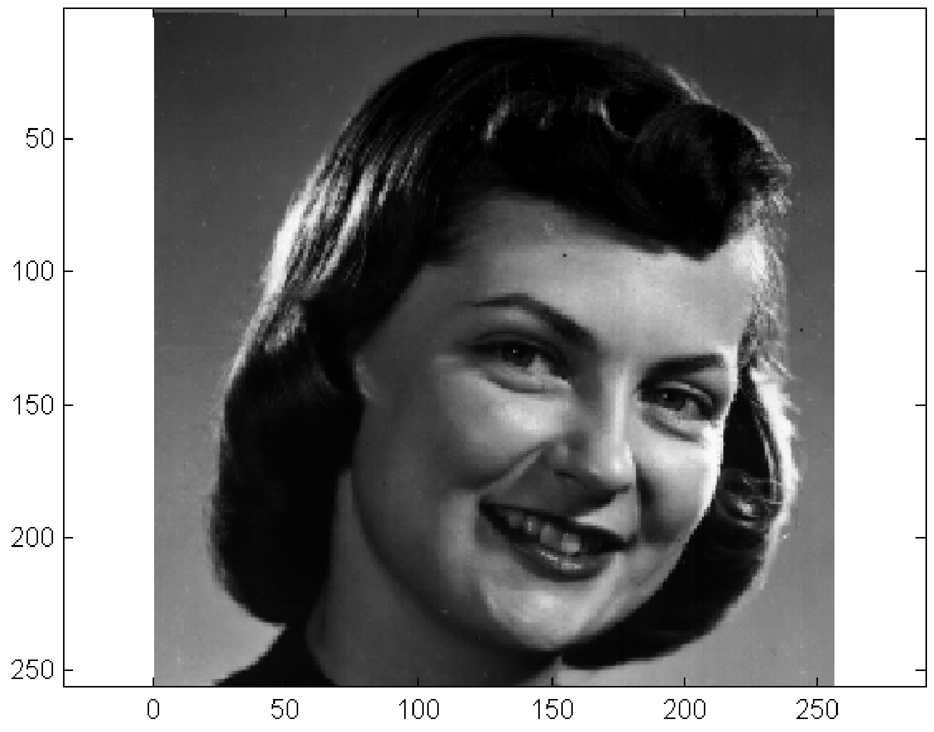
delete a random 10% of pixels



```
facemiss(facemiss > 254) = 9.e99;
newface = laplaceinterp(facemiss);
image(newface)
colormap(bwcolormap)
axis('equal')
```

restore them by Laplace interpolation

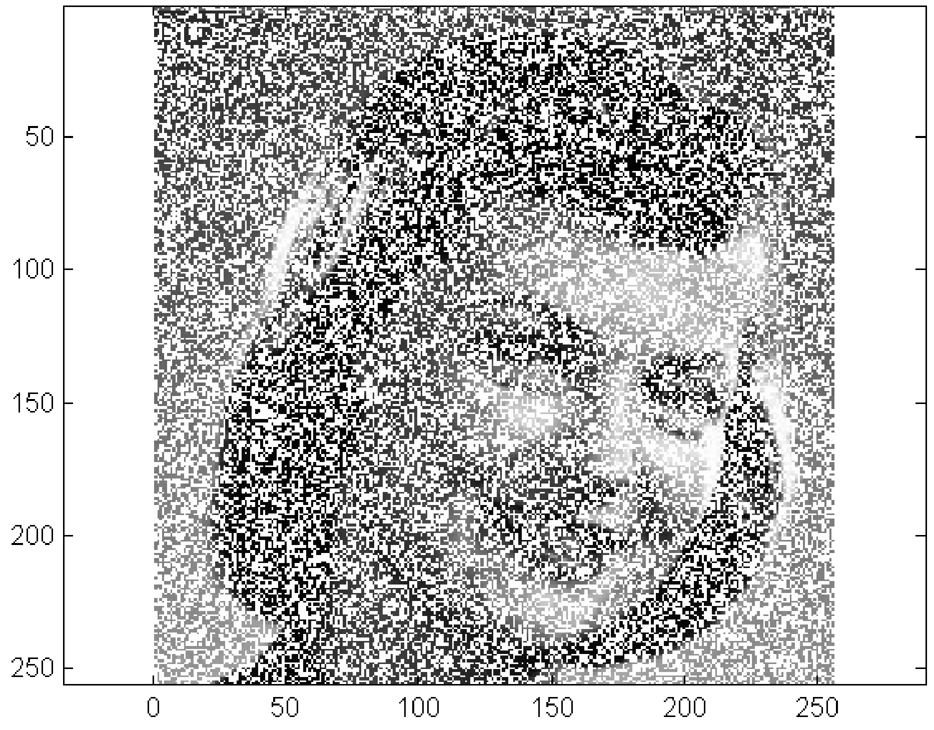
this is the convention expected by laplaceinterp for missing data



pretty amazing!

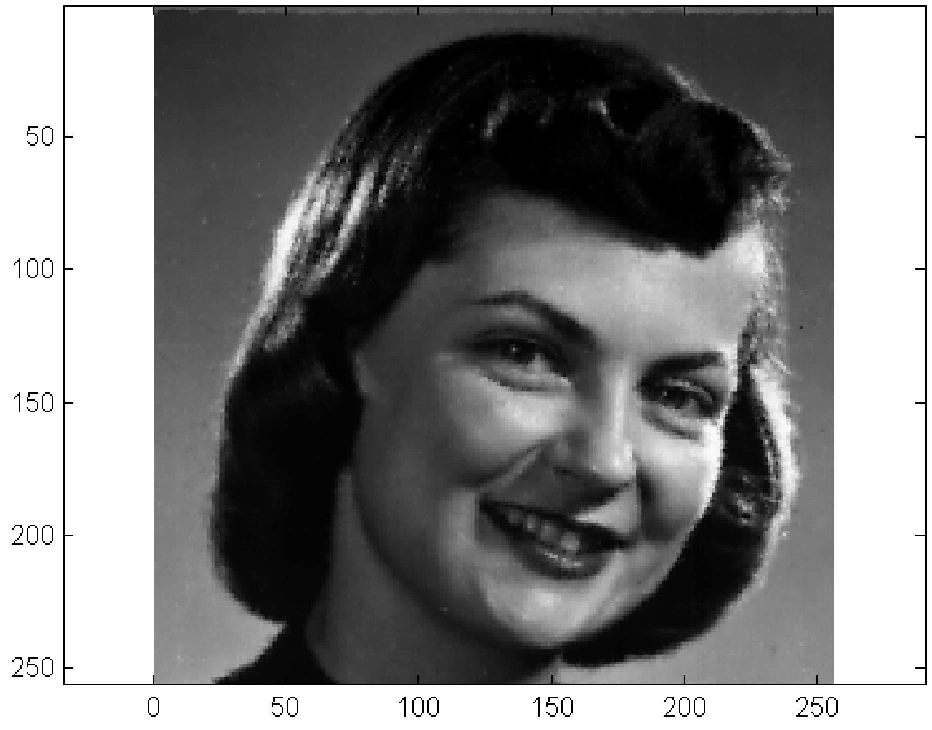
```
facemiss = face;
ranface = rand(size(face));
facemiss(ranface < 0.5) = 255;
image(facemiss)
colormap(bwcolormap)
axis('equal')
```

delete a random 50% of pixels



```
facemiss(facemiss > 254) = 9.e99;
newface = laplaceinterp(facemiss);
image(newface)
colormap(bwcolormap)
axis('equal')
```

restore them by Laplace interpolation

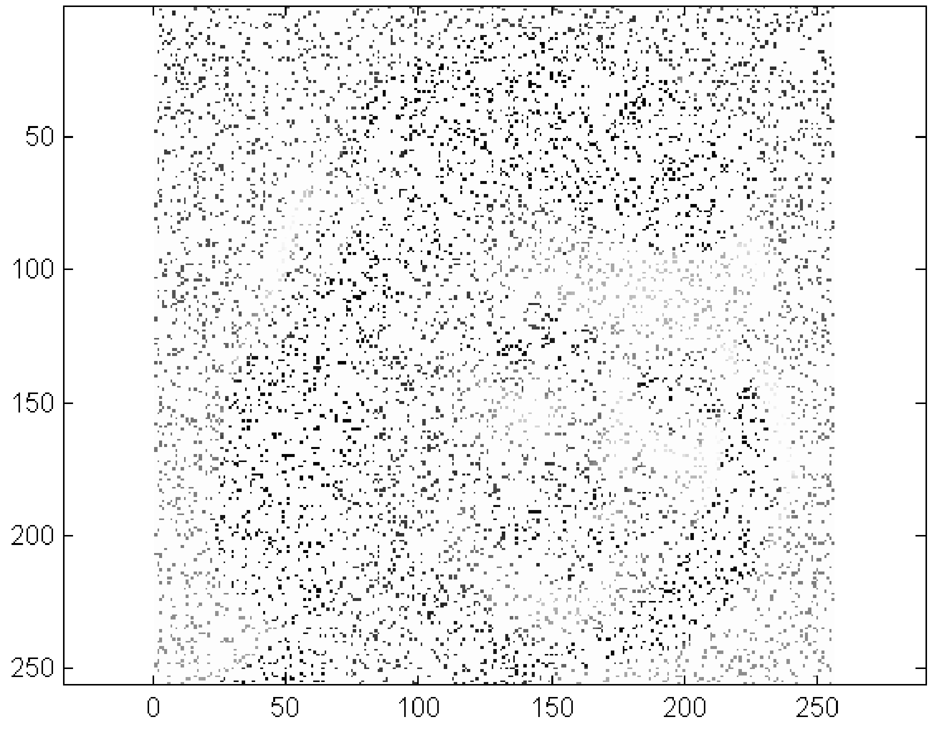


starting to see some degradation

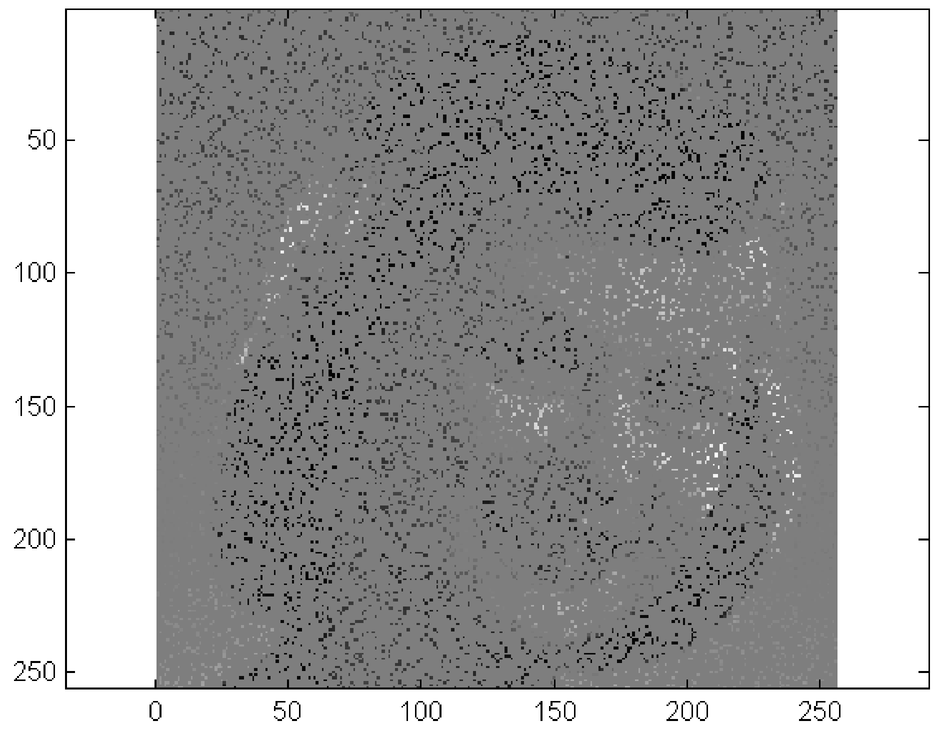
```
facemiss = face;
ranface = rand(size(face));
facemiss(ranface < 0.9) = 255;
image(facemiss)
colormap(bwcolormap)
axis('equal')
```

delete a random 90% of pixels

(well, it's cheating a bit, because your eye can't see the shades of grey in the glare of all that white)

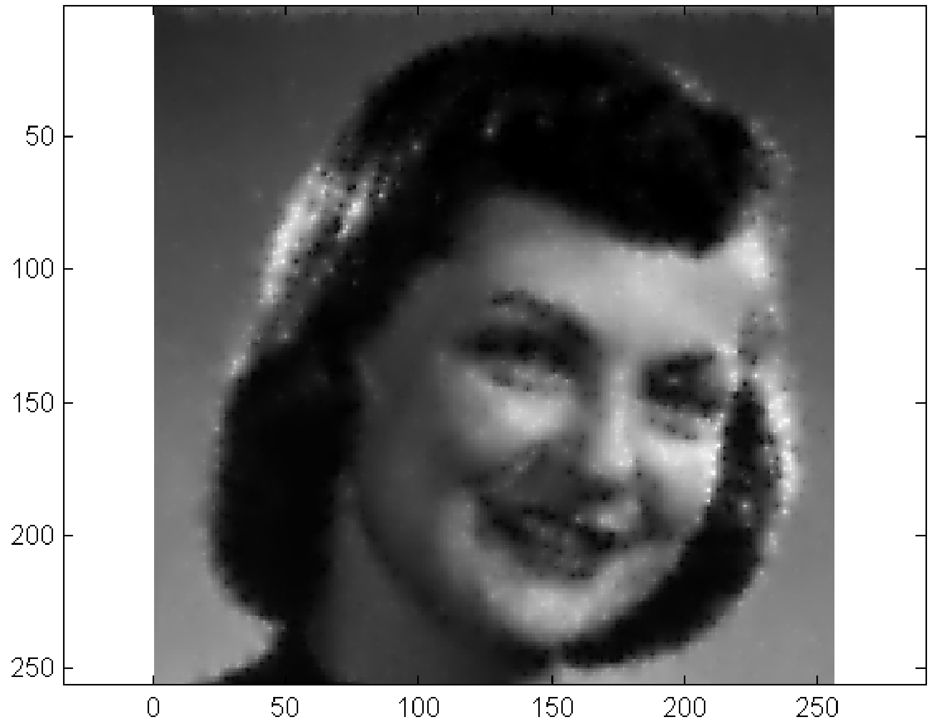


This is a bit more fair…



```
facemiss(facemiss > 254) = 9.e99;
newface = laplaceinterp(facemiss);
image(newface)
colormap(bwcolormap)
axis('equal')
```

restore by Laplace interpolation



still pretty amazing (e.g., would you have thought that the individual teeth were present in the sparse image?)

Interpolation on Scattered Data in Multidimensions

In dimensions >2, the explosion of volume makes things difficult.

Rarely enough data for any kind of mesh.

Lots of near-ties for nearest neighbor points (none very near).

The problem is more like a machine learning problem:

Given a training set $\mathbf{x}_i$ with "responses" y_i, $i = 1\ldots N$
Predict $y(\mathbf{x})$ for some new $\mathbf{x}$

Example: In a symmetrical multivariate normal distribution in large dimension D, everything is at almost the same distance from everything else:

$$\mathbf{x}: \quad x_i \sim \text{Normal}(0,1)$$
$$\mathbf{x}_1 - \mathbf{x}_2: \quad (x_{1i} - x_{2i}) \sim \text{Normal}(0, \sqrt{2})$$
$$\|\mathbf{x}_1 - \mathbf{x}_2\|^2 \sim 2 \ \text{Chisq}(D)$$
$$\approx 2(D \pm \sqrt{2D})$$