Image Deblurring
Spring 2012
Notes on Chapter 1
Dianne P. O'Leary

Overview of Course

- Textbook: Deblurring Images: Matrices, Spectra, and Filtering, SIAM Press, 2006.
- Organization:
  - Lectures: Please ask questions!
  - Challenges: Work to be done alone or in small group collaborations, as you prefer. Some in class, some on your own.
  - Course syllabus and schedule.

The plan for this lecture:

- What is image deblurring?
- How do images become arrays of numbers?
- How do we model the blurring process?
- What makes blurring hard?
- How do we model more general blurring?

Reference: Chapter 1 of Deblurring Images.

What is image deblurring?

What is image deblurring?

When we use a camera, we want the recorded image to be a faithful representation of the scene that we see – but every image is more or less blurry.

Thus, image deblurring, the process of processing the image to make it a better representation of the scene, is fundamental in making pictures sharp and useful.

Pixels

A digital image is composed of picture elements called pixels.

Each pixel is assigned an intensity, meant to characterize the color of a small rectangular segment of the scene. The intensity can be an integer or a vector of integers. (More on this later.)

A small image typically has around $256^2 = 65{,}536$ pixels while a high-resolution image often has 5 to 10 million pixels.

Why are images blurry?

Some blurring always arises in the recording of a digital image, because it is unavoidable that scene information "spills over" to neighboring pixels.

- The optical system in a camera lens may be out of focus, so that the incoming light is smeared out.
- In astronomical imaging the incoming light in the telescope is slightly bent by turbulence in the atmosphere.

Lenses are not perfect, so blurring always occurs, but in most images we ignore it.

How can blur be reduced or eliminated?

In image deblurring, we seek to recover the original, sharp image by using a mathematical model of the blurring process.

Key issue: some information on the lost details is indeed present in the blurred image – but this information is "hidden" and can only be recovered if we know the details of the blurring process.

Unfortunately there is no hope that we can recover the original image exactly: there is error in our data.

- defects in the recording process: e.g., slight variations in the film or slight differences in the digital hardware that records each pixel.
- approximation errors due to the resolution level of the pixels.
- truncation errors, due to recording an integer approximation to a continuous quantity.

Our challenge:

Devise efficient and reliable algorithms for recovering as much information as possible from the given (imperfect) data.

Why is image deblurring important?

- Yes, it is a useful tool for our vacation pictures.
- More importantly, it enables us to extract maximal information in cases where it is expensive or even impossible to obtain an image without blur:
  - astronomical images
  - medical images
- It has important applications in our economy: for example, barcode readers used in stores and by shipping companies must be able to compensate for imperfections in the scanner optics.

How do images become arrays of numbers?

We need to represent images as arrays of numbers in order to use mathematical techniques for deblurring.

Grayscale Images

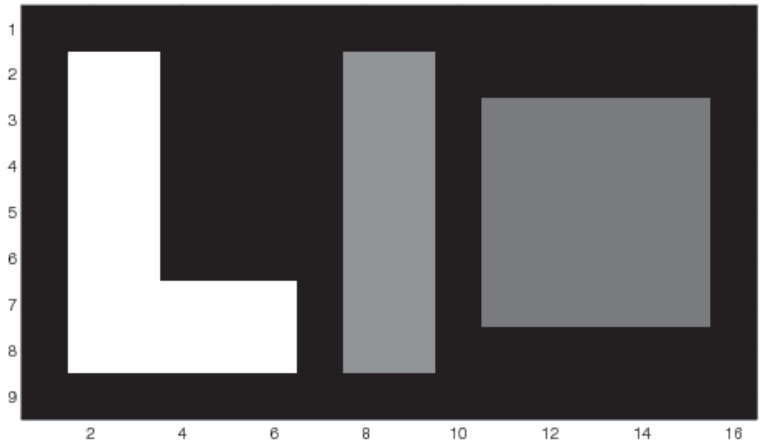
Grayscale images are typically recorded by CCDs (charge-coupled devices), arrays of tiny detectors, arranged in a rectangular grid, able to record the amount, or intensity, of the light that hits each detector.

Thus, we can think of a grayscale digital image as a rectangular $m \times n$ array, whose entries represent light intensities captured by the detectors.

Consider the following 9×16 array:

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 8 & 8 & 0 & 0 & 0 & 0 & 4 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 8 & 8 & 0 & 0 & 0 & 0 & 4 & 4 & 0 & 3 & 3 & 3 & 3 & 3 & 0 \\
0 & 8 & 8 & 0 & 0 & 0 & 0 & 4 & 4 & 0 & 3 & 3 & 3 & 3 & 3 & 0 \\
0 & 8 & 8 & 0 & 0 & 0 & 0 & 4 & 4 & 0 & 3 & 3 & 3 & 3 & 3 & 0 \\
0 & 8 & 8 & 0 & 0 & 0 & 0 & 4 & 4 & 0 & 3 & 3 & 3 & 3 & 3 & 0 \\
0 & 8 & 8 & 8 & 8 & 8 & 0 & 4 & 4 & 0 & 3 & 3 & 3 & 3 & 3 & 0 \\
0 & 8 & 8 & 8 & 8 & 8 & 0 & 4 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

If we enter this into a MATLAB variable `X` and display the array with the commands `imagesc(X), axis image, colormap(gray)`, then we obtain



Notice:

- 8 is displayed as white
- 0 is displayed as black.
- Values in between are shades of gray.

Color images

Color images are stored as three components, which represent their intensities on the red, green, and blue scales.

- $(1, 0, 0)$ is red.
- $(0, 0, 1)$ is blue.
- $(1, 1, 0)$ is yellow.

Other colors can be obtained with different choices of intensities.

Hence, we need three arrays (of the same size) to represent a color image.

An example of a color image

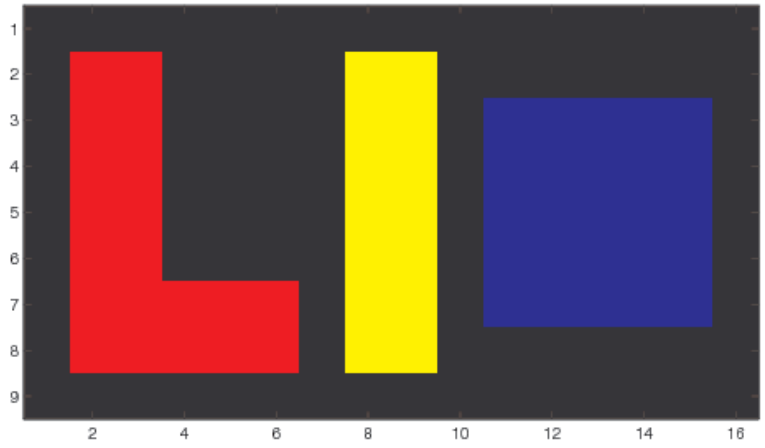
Let `X` be a multidimensional MATLAB array of dimensions $9 \times 16 \times 3$ defined as

$$\mathtt{X}(:,:,1) = \begin{bmatrix} 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&1&1&0&0&0&0&1&1&0&0&0&0&0&0&0 \\ 0&1&1&0&0&0&0&1&1&0&0&0&0&0&0&0 \\ 0&1&1&0&0&0&0&1&1&0&0&0&0&0&0&0 \\ 0&1&1&0&0&0&0&1&1&0&0&0&0&0&0&0 \\ 0&1&1&0&0&0&0&1&1&0&0&0&0&0&0&0 \\ 0&1&1&1&1&1&0&1&1&0&0&0&0&0&0&0 \\ 0&1&1&1&1&1&0&1&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \end{bmatrix},$$

$$\mathtt{X}(:,:,2) = \begin{bmatrix} 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \end{bmatrix},$$

$$\mathtt{X}(:,:,3) = \begin{bmatrix} 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&0 \\ 0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&0 \\ 0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&0 \\ 0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&0 \\ 0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \end{bmatrix}.$$

The command `imagesc(X)`, gives the picture



- We will focus mostly on grayscale images.
- However, the techniques carry over to color images, and we will discuss them later in the course.

How do we model the blurring process?

We must devise a mathematical model that relates the given blurred image to the unknown true image.

To fix notation:

- $\mathbf{X} \in \mathbb{R}^{m \times n}$ represents the desired sharp image,
- $\mathbf{B} \in \mathbb{R}^{m \times n}$ denotes the recorded blurred image.

An important special case

Assume that the blurring of the columns in the image is independent of the blurring of the rows.

When this is the case, then there exist two matrices $\mathbf{A}_c \in \mathbb{R}^{m \times m}$ and $\mathbf{A}_r \in \mathbb{R}^{n \times n}$, such that

$$\mathbf{A}_c \, \mathbf{X} \, {\mathbf{A}_r}^T = \mathbf{B}.$$

- Left multiplication with the matrix $\mathbf{A}_c$ applies the same vertical blurring operation to all the n columns $\mathbf{x}_j$ of $\mathbf{X}$, because

$$\mathbf{A}_c \, \mathbf{X} = \mathbf{A}_c \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_n \end{bmatrix} = \begin{bmatrix} \mathbf{A}_c\mathbf{x}_1 & \mathbf{A}_c\mathbf{x}_2 & \cdots & \mathbf{A}_c\mathbf{x}_n \end{bmatrix}.$$

- Right multiplication with ${\mathbf{A}_r}^T$ applies the same horizontal blurring to all the m rows of $\mathbf{X}$.
- Since matrix multiplication is associative, i.e., $(\mathbf{A}_c \, \mathbf{X}) \, {\mathbf{A}_r}^T = \mathbf{A}_c \, (\mathbf{X} \, {\mathbf{A}_r}^T)$, it does not matter in which order we perform the two blurring operations.

What makes blurring hard?

A First Attempt at Deblurring

This looks simple! If

$$\mathbf{A}_c \mathbf{X} \mathbf{A}_r{}^T = \mathbf{B},$$

then

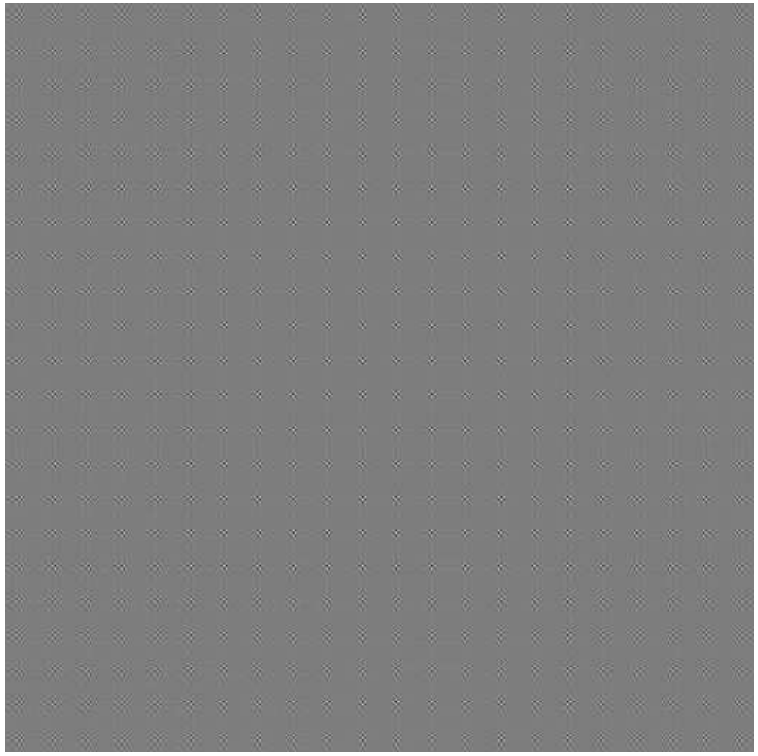
$$\mathbf{X} = \mathbf{A}_c{}^{-1} \mathbf{B} \mathbf{A}_r{}^{-T}$$

($\mathbf{A}_r{}^{-T} = (\mathbf{A}_r{}^T)^{-1} = (\mathbf{A}_r{}^{-1})^T$.)

So we have an algorithm for deblurring.

OK. I guess we're finished, and can spend the rest of the course playing Angry Birds.

The results of our algorithm



The "naïve" reconstruction of the pumpkin image, obtained by computing $\mathbf{X} = {\mathbf{A}_c}^{-1}\mathbf{B}\,{\mathbf{A}_r}^{-T}$ via Gaussian elimination on both $\mathbf{A}_c$ and $\mathbf{A}_r$. The image $\mathbf{X}$ is completely dominated by the influence of the noise.

What went wrong?

To understand why this naïve approach fails, we must take a closer look.

Notation:

- exact (unknown) image $= \mathbf{X}_{exact}$
- noise-free blurred version of the image $= \mathbf{B}_{exact} = \mathbf{A}_c \, \mathbf{X}_{exact} \, \mathbf{A}_r{}^T$.

Unfortunately, we don't know $\mathbf{B}_{exact}$!

The blurred image is collected by a mechanical device, so inevitably small random errors (noise) will be present in the recorded data.

Let us assume that this noise is additive and that it is statistically uncorrelated with the image.

Then the recorded blurred image $\mathbf{B}$ is really given by

$$\mathbf{B} = \mathbf{B}_{exact} + \mathbf{E} = \mathbf{A}_c \, \mathbf{X}_{exact} \, \mathbf{A}_r{}^T + \mathbf{E},$$

where the matrix $\mathbf{E}$ (of the same dimensions as $\mathbf{B}$) represents the noise in the recorded image.

Why did the naïve reconstruction fail?

The naïve reconstruction computed

$$\mathbf{X}_{naive} = \mathbf{A}_c^{-1}\mathbf{B}\,\mathbf{A}_r^{-T} = \mathbf{A}_c^{-1}\mathbf{B}_{exact}\,\mathbf{A}_r^{-T} + \mathbf{A}_c^{-1}\mathbf{E}\,\mathbf{A}_r^{-T}$$

and therefore

$$\mathbf{X}_{naive} = \mathbf{X}_{exact} + \mathbf{A}_c^{-1}\mathbf{E}\,\mathbf{A}_r^{-T},$$

where the term $\mathbf{A}_c^{-1}\mathbf{E}\,\mathbf{A}_r^{-T}$, which we can informally call inverted noise, represents the contribution to the reconstruction from the additive noise.

This inverted noise will dominate the solution if $\mathbf{A}_c^{-1}\mathbf{E}\,\mathbf{A}_r^{-T}$ has larger elements than $\mathbf{X}_{exact}$.

Unfortunately, in many situations, the inverted noise indeed dominates.

Apparently, image deblurring is not as simple as it first appears, which will limit our time playing Angry Birds.

We will spend most of the course developing deblurring methods that are able to correctly handle the inverted noise.

How do we model more general blurring?

Linear models

We assume throughout this course that the blurring, i.e., the operation of going from the sharp image to the blurred image, is linear.

- This assumption is (usually) a good approximation to reality.
- This assumption makes our life much easier!
- This assumption is almost always made in the literature and in practice.

This one assumption opens a wide choice of methods!

A general linear model

Our first model

$$\mathbf{A}_c \, \mathbf{X} \, \mathbf{A}_r{}^T = \mathbf{B},$$

requires that the same horizontal blur and the same vertical blur be applied to every pixel.

To form a more general model, we must rearrange the elements of the images $\mathbf{X}$ and $\mathbf{B}$ into column vectors by stacking the columns of these images into two long vectors $\mathbf{x}$ and $\mathbf{b}$, both of length $N = m\,n$. The notation for this operator is vec:

$$\mathbf{x} = \mathrm{vec}(\mathbf{X}) = \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} \in \mathbb{R}^N, \qquad \mathbf{b} = \mathrm{vec}(\mathbf{B}) = \begin{bmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_n \end{bmatrix} \in \mathbb{R}^N.$$

Since the blurring is assumed to be a linear operation, there must exist a large matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ such that $\mathbf{x}$ and $\mathbf{b}$ are related by the linear model

$$\mathbf{A}\,\mathbf{x} = \mathbf{b}$$

and this is our fundamental image blurring model.

For now, assume that $\mathbf{A}$ is known; we'll give more details on this later.

What does linearity mean?

If $\mathbf{B}_1$ and $\mathbf{B}_2$ are the blurred images of the exact images $\mathbf{X}_1$ and $\mathbf{X}_2$, then

$$\mathbf{B} = \alpha\,\mathbf{B}_1 + \beta\,\mathbf{B}_2$$

is the image of

$$\mathbf{X} = \alpha\,\mathbf{X}_1 + \beta\,\mathbf{X}_2$$

.

When this is the case, then there exists a large matrix $\mathbf{A}$ such that $\mathbf{b} = \mathrm{vec}(\mathbf{B})$ and $\mathbf{x} = \mathrm{vec}(\mathbf{X})$ are related by the equation

$$\mathbf{A}\,\mathbf{x} = \mathbf{b}.$$

The matrix $\mathbf{A}$ represents the blurring that is taking place in the process of going from the exact to the blurred image.

How can we solve the linear model?

$$\mathbf{A}\,\mathbf{x} = \mathbf{b}.$$

means

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b},$$

but this is just the naïve approach again, and we can expect failure due to the effects of inverted noise.

Let's develop the machinery to understand why this fails and to cure the failure.

Understanding why the naïve approach fails

Again let $\mathbf{X}_{exact}$ and $\mathbf{B}_{exact}$ be, respectively, the exact image and the noise-free blurred image, and define

$$\mathbf{x}_{exact} = \mathrm{vec}(\mathbf{X}_{exact}), \qquad \mathbf{b}_{exact} = \mathrm{vec}(\mathbf{B}_{exact}) = \mathbf{A}\,\mathbf{x}_{exact}.$$

Then the noisy recorded image $\mathbf{B}$ is

$$\mathbf{b} = \mathbf{b}_{exact} + \mathbf{e},$$

where the vector $\mathbf{e} = \mathrm{vec}(\mathbf{E})$ represents the image noise.

Consequently (again ignoring rounding errors) the naïve reconstruction is given by

$$\mathbf{x}_{naive} = \mathbf{A}^{-1}\mathbf{b} = \mathbf{A}^{-1}\mathbf{b}_{exact} + \mathbf{A}^{-1}\mathbf{e} = \mathbf{x}_{exact} + \mathbf{A}^{-1}\mathbf{e},$$

where the term $\mathbf{A}^{-1}\mathbf{e}$ is the inverted noise.

Again, the important observation is that the deblurred image consists of two components:

- The first component is the exact image.
- The second component is the inverted noise.

If the deblurred image looks unacceptable, it is because the inverted noise term contaminates the reconstructed image.

Designing an improved method

Important tool for insight: the singular value decomposition (SVD) of $\mathbf{A}$.

The SVD of a square matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ is essentially unique, and is defined as the decomposition

$$\mathbf{A} = \mathbf{U}\,\boldsymbol{\Sigma}\,\mathbf{V}^T,$$

where

- $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices, satisfying $\mathbf{U}^T\mathbf{U} = \mathbf{I}_N$ and $\mathbf{V}^T\mathbf{V} = \mathbf{I}_N$. The columns $\mathbf{u}_i$ of $\mathbf{U}$ are called the left singular vectors, while the columns $\mathbf{v}_i$ of $\mathbf{V}$ are the right singular vectors.
- $\boldsymbol{\Sigma} = \mathrm{diag}(\sigma_i)$ is a diagonal matrix whose elements σ_i appear in non-increasing order,

  $$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_N \geq 0.$$

  The quantities σ_i are called the singular values, and the rank of $\mathbf{A}$ is equal to the number of positive singular values.

Important property: if $i \neq j$,

$$\mathbf{u}_i^T \mathbf{u}_j = 0$$

and

$$\mathbf{v}_i^T \mathbf{v}_j = 0.$$

A^{-1} using the SVD

Assume for the moment that all singular values are strictly positive.

First representation:

$$\begin{aligned} \mathbf{A} &= \mathbf{U}\,\Sigma\mathbf{V}^T \\ \mathbf{A}^{-1} &= \mathbf{V}\,\Sigma^{-1}\mathbf{U}^T \end{aligned}$$

Given the SVD, we can easily multiply a vector by $\mathbf{A}^{-1}$ since Σ is a diagonal matrix, so Σ^{-1} is also diagonal, with entries $1/\sigma_i$ for $i = 1, \ldots, N$.

Second representation:

$$
\begin{aligned}
\mathbf{A} &= \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T \\
&= \begin{bmatrix} \mathbf{u}_1 & \cdots & \mathbf{u}_N \end{bmatrix} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_N \end{bmatrix} \begin{bmatrix} \mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_N^T \end{bmatrix} \\
&= \mathbf{u}_1\sigma_1\mathbf{v}_1^T + \cdots + \mathbf{u}_N\sigma_N\mathbf{v}_N^T \\
&= \sum_{i=1}^{N} \sigma_i \mathbf{u}_i \, \mathbf{v}_i^T .
\end{aligned}
$$

Similarly,

$$
\mathbf{A}^{-1} = \sum_{i=1}^{N} \frac{1}{\sigma_i} \mathbf{v}_i \, \mathbf{u}_i^T .
$$

Finally: how the inverted noise gets magnified

Using our second representation,

$$\mathbf{A}^{-1} = \sum_{i=1}^{N} \frac{1}{\sigma_i} \mathbf{v}_i \, \mathbf{u}_i^T \, .$$

the solution to our problem is

$$\mathbf{A}^{-1}\mathbf{b} = \sum_{i=1}^{N} \frac{1}{\sigma_i} \mathbf{v}_i \, \mathbf{u}_i^T \mathbf{b} \, .$$

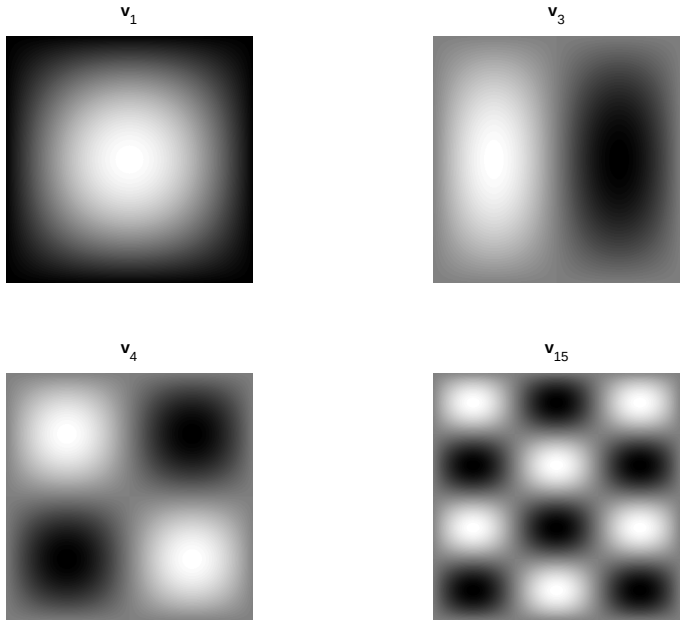
and the inverted noise contribution to the solution is

$$\mathbf{A}^{-1}\mathbf{e} = \mathbf{V}\,\mathbf{\Sigma}^{-1}\mathbf{U}^T\mathbf{e} = \sum_{i=1}^{n} \frac{\mathbf{u}_i^T \mathbf{e}}{\sigma_i} \mathbf{v}_i \, .$$

Why does the error term dominate?

$$\mathbf{A}^{-1}\mathbf{e} = \mathbf{V}\,\boldsymbol{\Sigma}^{-1}\mathbf{U}^T\mathbf{e} = \sum_{i=1}^{n} \frac{\mathbf{u}_i^T\mathbf{e}}{\sigma_i}\,\mathbf{v}_i\,.$$

- The error components $|\mathbf{u}_i^T\mathbf{e}|$ are small and typically of roughly the same order of magnitude for all i.
- The singular values decay to a value very close to zero.
- When we divide by a small singular value such as σ_N, we greatly magnify the corresponding error component, $\mathbf{u}_N^T\mathbf{e}$, which in turn contributes a large multiple of the high frequency information contained in $\mathbf{v}_N$ to the computed solution.
- The singular vectors corresponding to the smaller singular values typically represent higher frequency information. That is, as i increases, the vectors $\mathbf{u}_i$ and $\mathbf{v}_i$ tend to have more sign changes.



A few of the singular vectors for the blur of the pumpkin image. The "images" shown in this figure were obtained by reshaping the $n^2 \times 1$ singular vectors $\mathbf{v}_i$ into $n \times n$ arrays.

Interpreting the coefficients of the solution

$$\mathbf{A}^{-1}\mathbf{b} = \mathbf{V}\,\Sigma^{-1}\mathbf{U}^T\mathbf{b} = \sum_{i=1}^{n} \frac{\mathbf{u}_i^T\mathbf{b}}{\sigma_i}\,\mathbf{v}_i\,.$$

The quantities $\mathbf{u}_i^T\mathbf{b}/\sigma_i$ are the expansion coefficients for the basis vectors $\mathbf{v}_i$.

- When these quantities are small in magnitude, the solution has very little contribution from $\mathbf{v}_i$
- But when σ_i is very small, these quantities are large.

And when this happens in the presence of error, the naïve reconstruction appears as a random image dominated by high frequencies.

An improved solution through filtering

Because of the contamination due to the error components, we might be better off leaving the high frequency components out altogether.

We can replace

$$\mathbf{A}^{-1}\mathbf{b} = \sum_{i=1}^{N} \frac{\mathbf{u}_i^T \mathbf{b}}{\sigma_i} \mathbf{v}_i$$

by

$$\sum_{i=1}^{k} \frac{\mathbf{u}_i^T \mathbf{b}}{\sigma_i} \mathbf{v}_i \, .$$

for some choice of $k < N$.

The reconstruction obtained for the blur of pumpkins by using $k = 800$ (instead of the full $k = N = 169,744$)

Notice that the computed reconstruction is noticably better than the naïve solution shown before.

Next question:

- Will a different value for k produce a better reconstruction?
- If so, how can we choose a good value?

Important questions, but first, in the next lecture, we will discuss manipulating images in MATLAB.

Summary

- A digital image is a 2- or 3-dimensional array of numbers representing intensities on a grayscale or color scale.
- We model the blurring of images as a linear process characterized by a blurring matrix $\mathbf{A}$ and an observed image $\mathbf{B}$, which, in vector form, is $\mathbf{b}$.
- The reason $\mathbf{A}^{-1}\mathbf{b}$ cannot be used to deblur images is the amplification of high-frequency components of the noise in the data, caused by the inversion of very small singular values of $\mathbf{A}$. Practical methods for image deblurring need to avoid this pitfall.