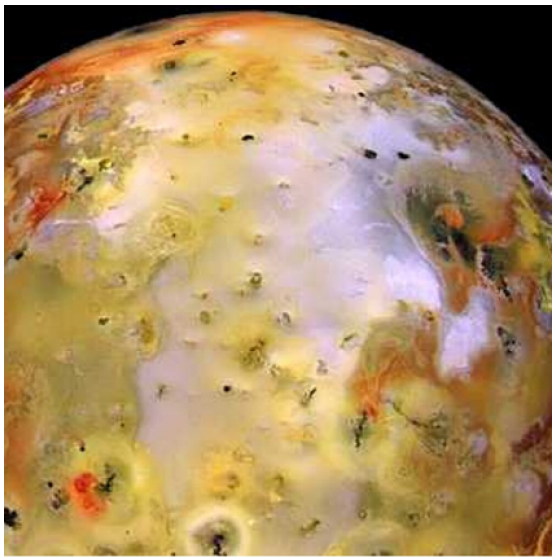
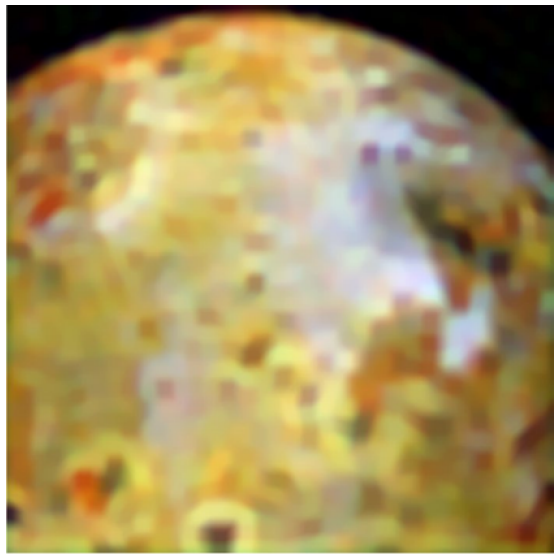
What is Image Deblurring?

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*, depending on the circumstances.

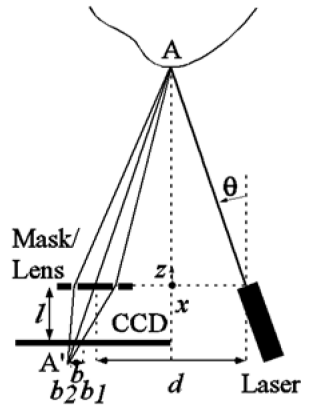
Image deblurring is the task of processing the image to make it a better representation of the scene – sharper and more useful.

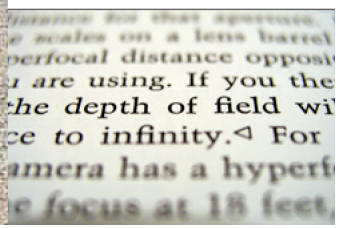




Deblurring is *model based*, in contrast to image enhancement techniques used, e.g., in PhotoShop and movie restoration.

Sources of Blurry Images





Case: Motion Blur

Motion blur arises frequently in several applications.

The two images are examples of spatially variant blur, where the blurring depends on the location in the image.

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.

Some blurring arises *within the camera:*

- The optical system in a camera lens may be out of focus, so that the incoming light is smeared out.
- The lens is not perfect, and light rays with different wavelengths follow slightly different paths (aberration).

Other kinds of blurring come from *outside the camera:*

- The camera or the object moved during the exposure.
- In astronomical imaging the incoming light in the telescope is slightly bent by turbulence in the atmosphere.

Errors in Digital Images

In addition to the unavoidable blur, there are *errors* in our data.

1. *False light*, background radiation, etc.
2. *Defects* in the recording process, e.g., slight variations in the film or slight differences in the digital recording device.
3. *Approximation/truncation errors* due to the resolution level of the pixels, i.e., recording an integer approximation to a continuous quantity (typically uniformly distributed noise).

   8-bit image with $2^8 = 256$ graylevels: ± 0.5 error $\rightarrow 0.3\%$.
4. *Noise* generated during the conversion of the light to an electrical signal (typically Gaussian noise).
5. *Photon noise* due to the nature of the light itself, in the form of photons (typically Poisson noise).

Why is Image Deblurring Important?

- 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, etc.
- It has important applications in our daily life: for example, bar-code readers used in stores and by shipping companies must be able to compensate for imperfections in the scanner optics.
- Also important applications in biometrics:
  - iris or retina scanning,
  - fingerprint identification, etc.

How Can Blur be Reduced or Eliminated?

Use photo editing software (e.g., PhotoShop) to perform "cosmetic" improvements: sharpen, filter, etc.

Image deblurring is the scientific approach, where we seek to recover the original, sharp image by using a **mathematical model** of the blurring process.

Key issue 1: Some information on the lost details is indeed still present in the blurred image – but this information is *"hidden"* and can only be recovered if we know the details of the blurring process.

Key issue 2: Unfortunately there is no hope that we can recover the original image exactly due to the *error* in our data.

Our Challenge:

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

This process involves several key ingredients:

1. A study of the mathematical model.
2. Derivation of the corresponding matrix structure.
3. Efficient implementation (large-scale problem).
4. Understanding the conditioning of the problem (ill-posed).
5. The choice of stabilization parameter.
6. The treatment of more general deblurring problems by iterative methods and other deblurring algorithms.

Images Are Arrays of Numbers = Pixels

Images are typically recorded by a *CCD* (charge-coupled device), an array of tiny detectors arranged in a rectangular grid, able to record the amount, or intensity, of the light that hits each detector.

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 binary (black&white image), an integer (grayscale image) or a vector of integers (color/multispectral image).

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

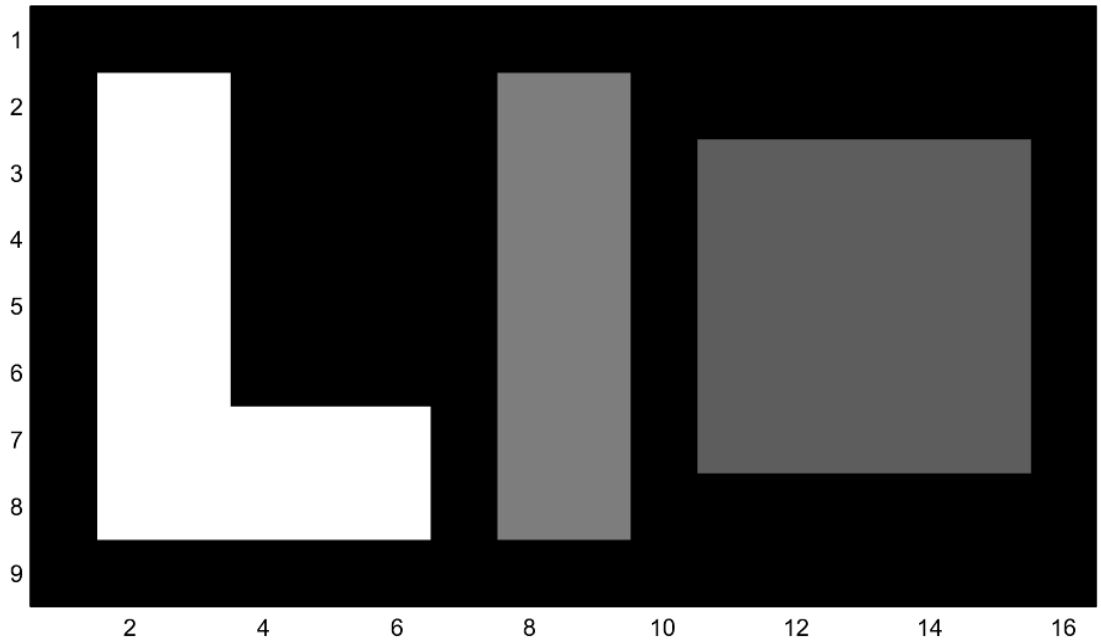
We need to represent images as arrays of numbers in order to use mathematical techniques for deblurring.

Grayscale Images

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:

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

If we enter `X` into Matlab 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.

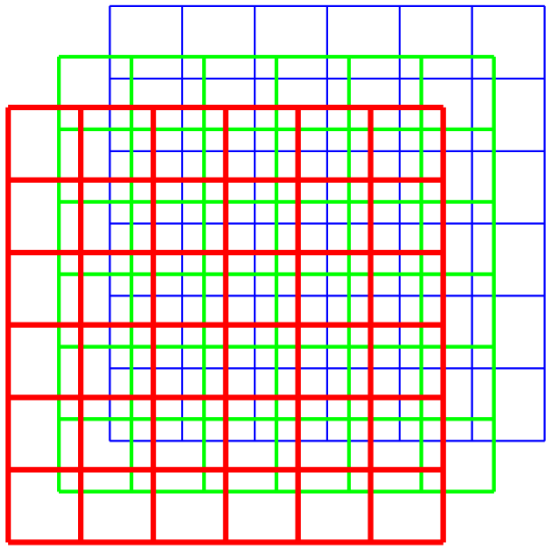
Color Images

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

(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.

For such *RGB images*, we need three arrays (of the same size) or, more practically, a single 3D-array, to represent a color image.



An Example of a Color Image

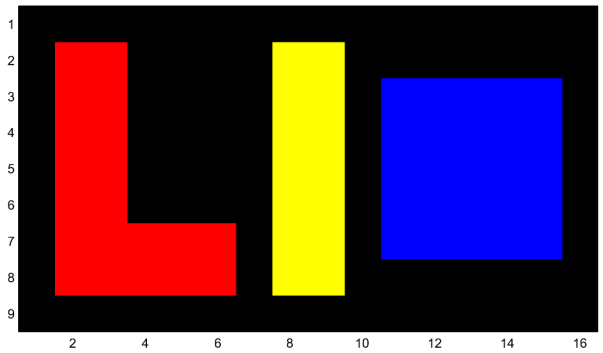
Let `X` be a three-dimensional Matlab array of dimensions $9 \times 16 \times 3$:

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

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

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

The command `imagesc(X)` gives the picture



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, $(\mathbf{A}_c \, \mathbf{X}) \, \mathbf{A}_r^T = \mathbf{A}_c \, (\mathbf{X} \, \mathbf{A}_r^T)$, it does not matter in which order we perform the two blurrings.

What Makes Deblurring Hard?

A First Attempt at Deblurring.

Now, this looks simple! If

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

then

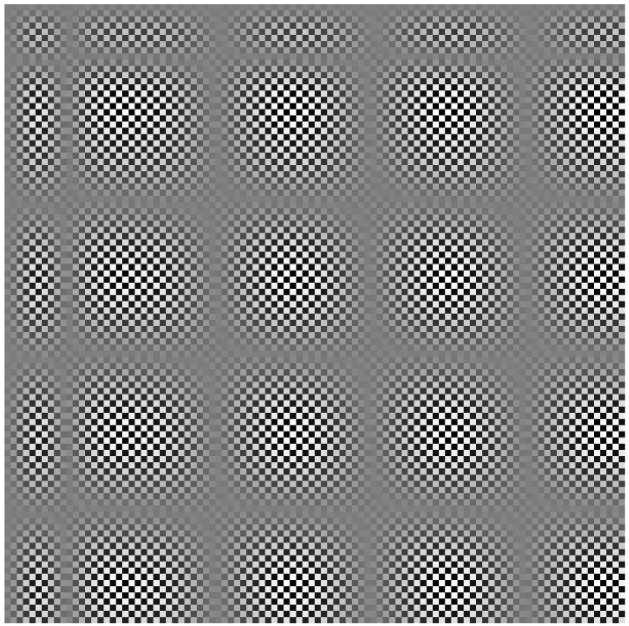
$$\mathbf{X}_{\text{naive}} = \mathbf{A}_c^{-1} \mathbf{B} \mathbf{A}_r^{-T}.$$

(We use the notation $\mathbf{A}_r^{-T} = (\mathbf{A}_r^T)^{-1} = (\mathbf{A}_r^{-1})^T$.)

So do we already have an algorithm for deblurring . . . ?

The Results of Our First Algorithm

The "naive" reconstruction of the pumpkin image, obtained by computing $\mathbf{X}_{\text{naive}} = \mathbf{A}_c^{-1}\mathbf{B}\,\mathbf{A}_r^{-T}$ via Gaussian elimination on both $\mathbf{A}_c$ and $\mathbf{A}_r$ (in Matlab: `Ac\B/Ar'`).



$\mathbf{X}_{\text{naive}}$ is completely dominated by the influence of the noise.

What Went Wrong?

To understand why this *naive* approach fails, we take a closer look.

- Exact (unknown) image $= \mathbf{X}$
- Noise-free blurred version of image $= \mathbf{B}_{\text{exact}} = \mathbf{A}_c \mathbf{X} \mathbf{A}_r^T$

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

Small random errors (noise) are present in the recorded data.

Even the representation of the image as an integer (typically with 8–10 bits) introduces small errors.

Let us assume that this noise is additive and statistically uncorrelated with the image. Then the recorded blurred image $\mathbf{B}$ is really given by

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

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

Why Did the Naive Reconstruction Fail?

The naive reconstruction computes

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

and

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

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.

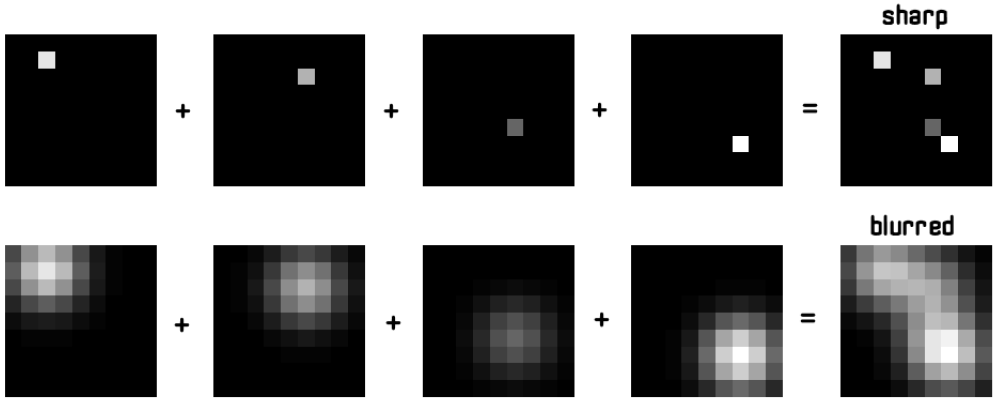
This inverted noise will dominate the solution if $\mathbf{A}_c^{-1}\mathbf{E}\,\mathbf{A}_r^{-T}$ has larger elements than $\mathbf{X}$. *Unfortunately, in many situations, the inverted noise indeed dominates.*

Apparently, image deblurring is not as simple as it first appears.

We will spend this course developing deblurring methods that are able to correctly handle (or rather suppress) the inverted noise.

How do We Model More General Blurring?

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



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

This one assumption opens a wide choice of methods!

But our first linear model $\mathbf{A}_c \mathbf{X} \mathbf{A}_r^T = \mathbf{B}$ is too limited.

A General Linear Model

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:"

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

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 fundamental linear model of image deblurring:

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

For now, assume that $\mathbf{A}$ is known; we'll give more details in Chapter 3.

Recap – What Does Linearity Mean?

Assume that $\mathbf{B}_1$ and $\mathbf{B}_2$ are the blurred images of the exact images $\mathbf{X}_1$ and $\mathbf{X}_2$. Then *linearity* means that

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

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 $\mathbf{b} = \text{vec}(\mathbf{B})$ and $\mathbf{x} = \text{vec}(\mathbf{X})$ 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 Problem?

The linear model

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

means that

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

but this is just the *naive* 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 Naive Approach Fails

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

$$\mathbf{b}_{\text{exact}} = \text{vec}(\mathbf{B}_{\text{exact}}) = \mathbf{A}\,\mathbf{x}, \qquad \mathbf{e} = \text{vec}(\mathbf{E}).$$

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

$$\mathbf{b} = \mathbf{b}_{\text{exact}} + \mathbf{e}, \qquad \mathbf{b}_{\text{exact}} = \mathbf{A}\,\mathbf{x}.$$

Consequently (ignoring rounding errors) the naive reconstruction is given by

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

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

Reflection

Again, the important observation is that the deblurred image $\mathbf{x}_{\text{naive}} = \mathbf{x} + \mathbf{A}^{-1}\mathbf{e}$ consists of two components:

- The first component $\mathbf{x}$ is the exact image.
- The second component $\mathbf{A}^{-1}\mathbf{e}$ is the inverted noise.

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

Moreover, until now we have ignored rounding errors.

But it is well known that rounding errors in the solution process manifest themselves as errors in the solutions/reconstructions.

From now on, think of these errors as part of the term $\mathbf{A}^{-1}\mathbf{e}$.

Towards an Improved Method – SVD Analysis

Important tool for insight: the *singular value decomposition* (SVD).

The SVD of a square matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ is essentially unique:

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

- $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices, $\mathbf{U}^T\mathbf{U} = \mathbf{I}_N$ 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*.
- If $i \neq j$ then $\mathbf{u}_i^T\mathbf{u}_j = 0$ and $\mathbf{v}_i^T\mathbf{v}_j = 0$.
- $\mathbf{\Sigma} = \mathrm{diag}(\sigma_i)$ 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.

Matlab and the SVD

To compute the SVD in Matlab, use:

```
[U,S,V] = svd(A);
```

To compute the k largest singular values and the corresponding left and right singular vectors, use:

```
[Uk,Sk,Vk] = svds(A,k);
```

How Matlab computes the "rank" of a matrix:

```
function r = rank(A,tol)
s = svd(A);
if nargin==1
   tol = max(size(A)) * eps(max(s));
end
r = sum(s > tol);
```

Writing $\mathbf{A}^{-1}$ in Terms of the SVD

First representation:

$$\mathbf{A} = \mathbf{U}\,\boldsymbol{\Sigma}\,\mathbf{V}^T \qquad \Leftrightarrow \qquad \mathbf{A}^{-1} = \mathbf{V}\,\boldsymbol{\Sigma}^{-1}\,\mathbf{U}^T.$$

Given the SVD, we can easily multiply a vector by $\mathbf{A}^{-1}$ since $\boldsymbol{\Sigma}$ is diagonal, so $\boldsymbol{\Sigma}^{-1}$ 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,

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

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 naive solution to our problem is

$$\mathbf{x}_{\text{naive}} = \mathbf{A}^{-1} \, \mathbf{b} = \mathbf{V} \, \mathbf{\Sigma}^{-1} \mathbf{U}^T \mathbf{b} = \sum_{i=1}^{N} \frac{\mathbf{u}_i^T \mathbf{b}}{\sigma_i} \, \mathbf{v}_i$$

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?

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

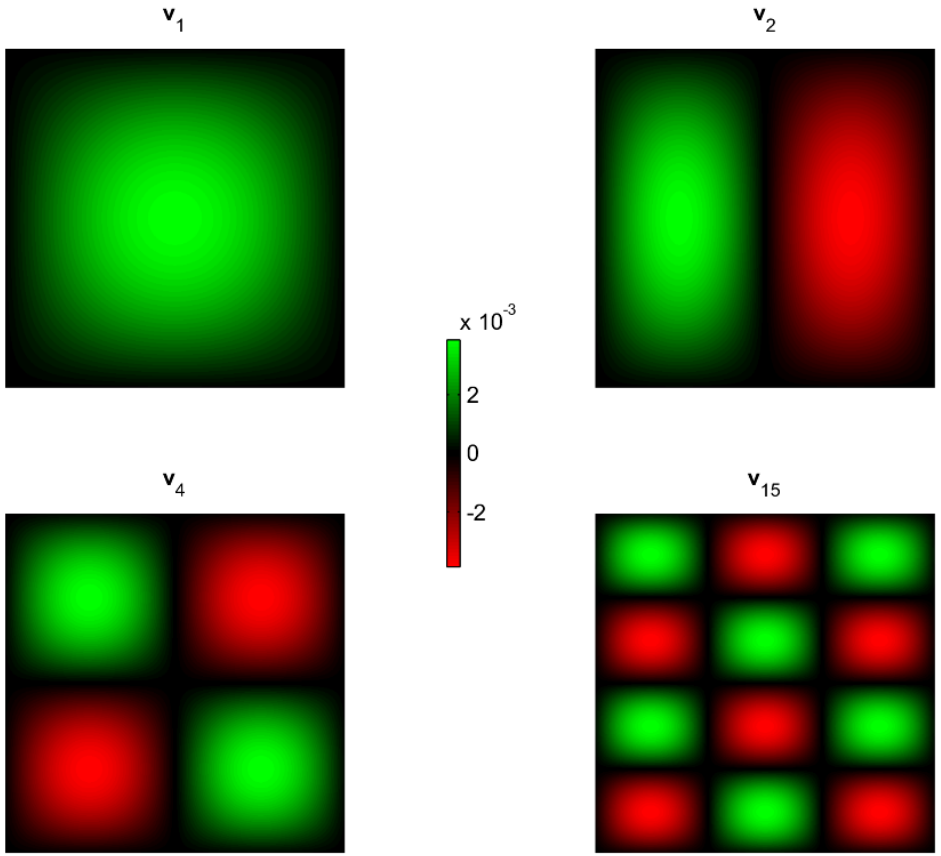
The error components $|\mathbf{u}_i^T\mathbf{e}|$ are *small* and typically of roughly the same order of magnitude for all i.

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. See next slide.

Reshaped Singular Vectors



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

Interpreting the Coefficients of the Solution

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

The quantities $\mathbf{u}_i^T\mathbf{b}/\sigma_i$ are 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 can be *large* due to the presence of the *noise* in $\mathbf{b} = \mathbf{b}_{\text{exact}} + \mathbf{e}$.

So in the presence of *error*, the naive reconstruction appears as a random image dominated by high frequencies.

An Improved Solution Through Truncation

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

We can replace

$$\mathbf{x}_{\text{naive}} = \mathbf{A}^{-1}\mathbf{b} = \mathbf{V}\,\boldsymbol{\Sigma}^{-1}\mathbf{U}^T\mathbf{b} = \sum_{i=1}^{N} \frac{\mathbf{u}_i^T\mathbf{b}}{\sigma_i}\,\mathbf{v}_i$$

by

$$\mathbf{x}_k = \sum_{i=1}^{k} \frac{\mathbf{u}_i^T\mathbf{b}}{\sigma_i}\,\mathbf{v}_i \equiv \mathbf{A}_k^{\dagger}\mathbf{b}$$

for some choice of $k < N$. Here, we introduce the pseudoinverse:

$$\mathbf{A}_k^{\dagger} = \begin{bmatrix} \mathbf{v}_1 & \cdots & \mathbf{u}_k \end{bmatrix} \begin{bmatrix} \sigma_1^{-1} & & \\ & \ddots & \\ & & \sigma_k^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{u}_1^T \\ \vdots \\ \mathbf{u}_k^T \end{bmatrix}.$$

Truncation

The reconstruction obtained for the blur of pumpkins by using $k = 800$ (instead of the full $k = N = 412 \cdot 412 = 169\,744$).

Notice that the computed reconstruction is *noticeably* better than the naive solution shown before.

Summary

1. A digital image is a 2- or 3-dimensional array of numbers representing intensities on a grayscale or color scale.
2. 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 $\mathbf{b}$.
3. 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 $\mathbf{A}$.
4. Algorithms for image deblurring need to avoid this pitfall.

NB: I will not cover Chapter 2 – if you need to recap image processing in Matlab then read it yourself and try Challenge 5.