Taking Bad Pictures

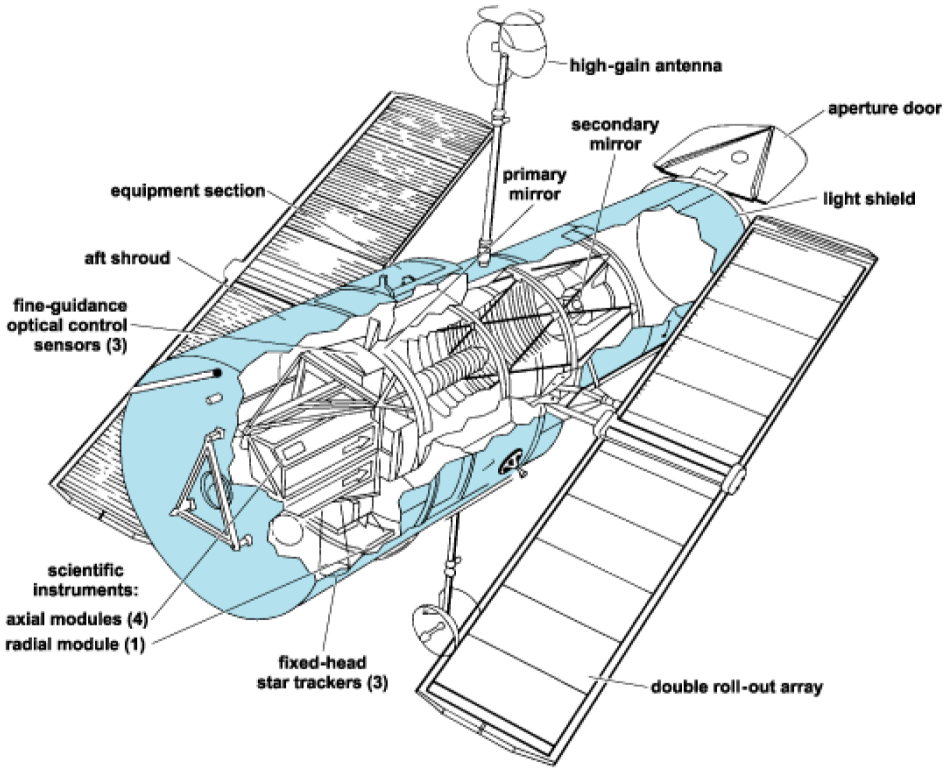
Sources of blurring:

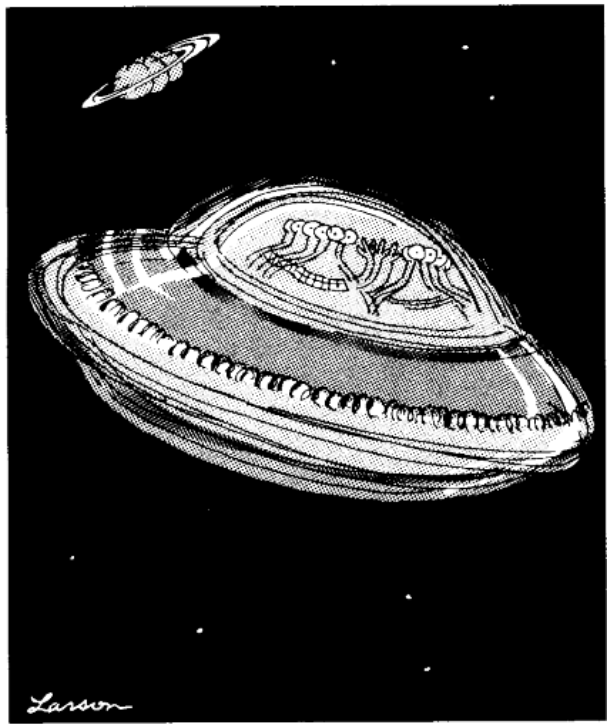
1. The lens is out of focus.
2. The camera is shaking.
3. The object is moving.
4. Defects in the lens or optical system.
5. Aberration – the optical path depends on the wavelength.
6. Statistical variations in the optical path (turbulence).

Our goal is to (try to) reconstruct the sharp image, using a mathematical model for the blurring.

An Example: the Hubble Space Telescope

For several years, the HST produced blurred images.





Another photograph from the Hubble telescope

Two Representations of Images

Our notation for images:

$$\mathbf{X} = \text{sharp image}, \qquad \mathbf{B} = \text{blurred image}$$

$$\text{all images are } m \times n.$$

The "vec" operation stacks the columns of the images:

$$\mathbf{x} = \text{vec}(\mathbf{X}), \qquad \mathbf{b} = \text{vec}(\mathbf{B})$$

$$\text{all vectors have length } N = m\,n.$$

Matlab commands to go back and forth:

```
x = X(:)
```

```
X = reshape(x,m,n)
```

Linearity of the Blurring Model

All our blurring models are **linear.**

There exists a large $N \times N$ matrix $\mathbf{A}$ such that we can write

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

Note that we distinguish between

- the $N \times N$ *blurring matrix* $\mathbf{A}$ and
- the $m \times n$ *image arrays* $\mathbf{B}$, $\mathbf{X}$.

In Matlab they are all just arrays/matrices.

The Linear Deblurring Problem

The relation $\mathbf{b} = \mathbf{A}\,\mathbf{x}$ is the *forward model*.

If we swap the "ingredients" then we obtain the *inverse problem*

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

associated with the reconstruction of $\mathbf{x}$.

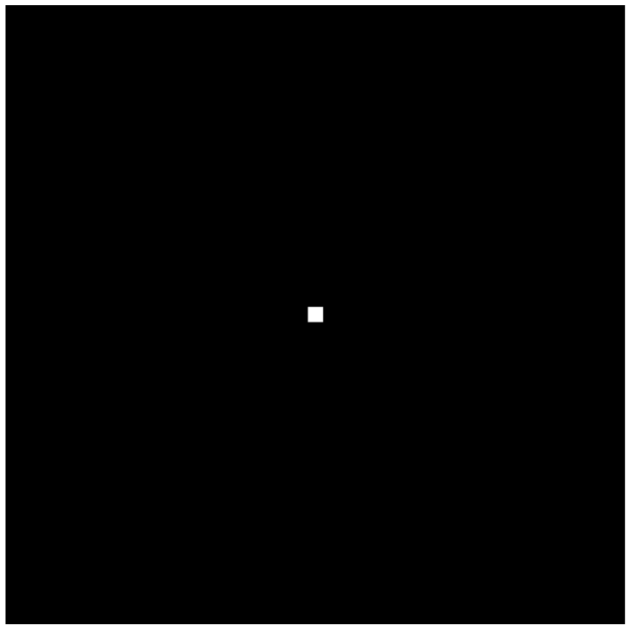
The computational problem in image deblurring is to solve this system. This is difficult, for several reasons:

1. The matrix $\mathbf{A}$ is large.
2. The matrix $\mathbf{A}$ is very ill conditioned.
3. The matrix $\mathbf{A}$ may be an imprecise model of the blurring.

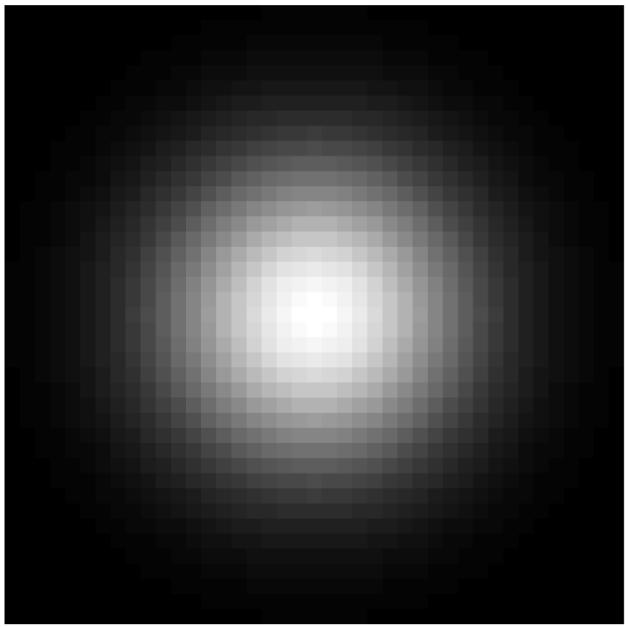
The Point Spread Function (PSF)

We need a precise description of the blurring matrix **A**.

Point source
A single white pixel



Point spread function
Image of point source



Measuring the PSF

Point sources and PSFs are often generated experimentally by taking a picture of a point source.

- *Astronomy:* point source = single bright star.
- *Microscopy:* point source = fluorescent microsphere.

The **PSF array** is the $m \times n$ image of the point source.

- The vector $\mathbf{e}_i$ (the ith unit vector) represents a point source.
- The image of the point source,

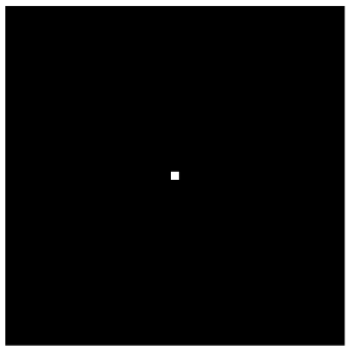
$$\mathbf{A}\,\mathbf{e}_i = \mathbf{a}_i = \text{column } i \text{ of } \mathbf{A},$$

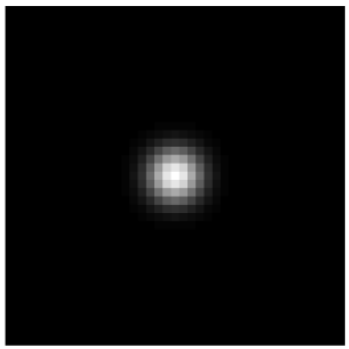
  represents the corresponding PSF.
- In this way we can fully assemble the blurring matrix $\mathbf{A}$ (in a rather cumbersome way).

The PSF is Local

Typically, the PSF is *local.*

This means that the extent of the region with (practically) nonzero elements of the PSF array is small.



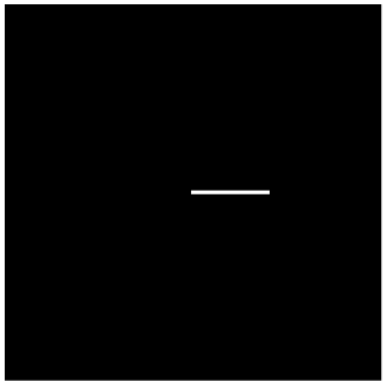


We can limit the storage and only save the (practically) nonzero part of the PSF as a much smaller $p \times q$ array $\mathbf{P}$.

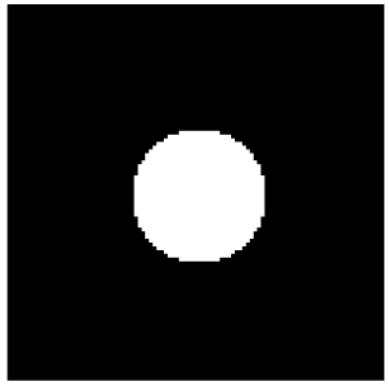
PSF Models

Sometimes we can give a specific formula for computing the PSF.

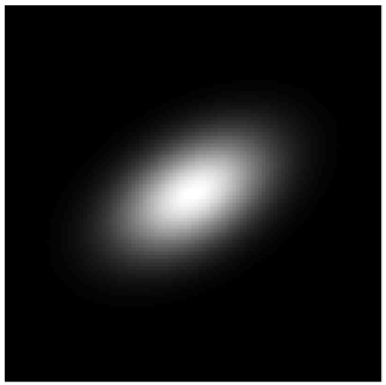
Horizontal motion blur



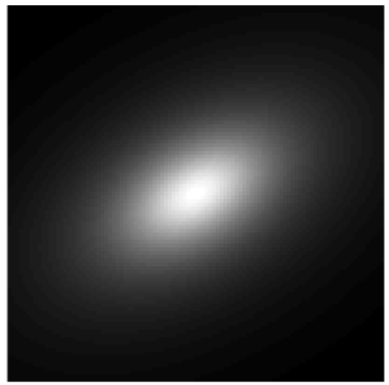
Out-of-focus blur



Atmospheric turbulence blur



Moffat blur



See next slide for simplified expressions for atmospheric turbulence (Gaussian) blur and Moffatt blur without a tilted axis.

Some PSF Models

Out-of-focus blur (radius $= r$):

$$p_{ij} = \begin{cases} 1/(\pi r^2), & \text{if } (i-k)^2 + (j-\ell)^2 \leq r^2 \\ 0, & \text{elsewhere.} \end{cases}$$

Atmospheric turbulence blur:

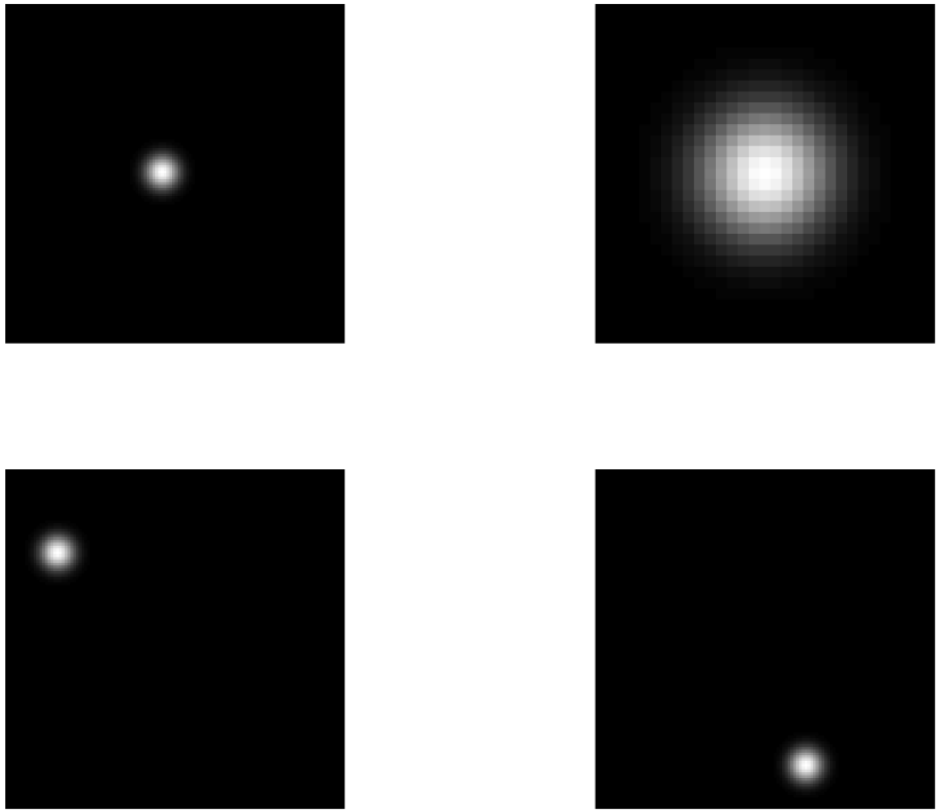
$$p_{ij} = \frac{1}{2\pi s_1 s_2} \exp\left(-\frac{1}{2}\left(\frac{i-k}{s_1}\right)^2 - \frac{1}{2}\left(\frac{j-\ell}{s_2}\right)^2 \right).$$

Moffat function for astronomical telescope blur:

$$p_{ij} = \left(1 + \left(\frac{i-k}{s_1}\right)^2 + \left(\frac{j-\ell}{s_2}\right)^2 \right)^{-\beta}.$$

The PSFs are centered at element (k, ℓ), and s_1 and s_2 determine the width of the PSF. – **Misprint p. 27:** no "exp" in equation.

Spatially Invariant Blur



Often the shape of the PSF is (almost) independent on the localization in the image = spatially invariant blur.

Leads to structure in the blurring matrix $\mathbf{A}$ ($\rightarrow$ Chapter 4).

Computing the Blurred Image

From the relation $\mathbf{b} = \mathbf{A}\,\mathbf{x}$ we see that the ith element in the blurred image is given by

$$b_i = \mathbf{e}_i^T \mathbf{b} = \mathbf{e}_i^T \mathbf{A}\,\mathbf{x} = \mathbf{A}(i, :)\,\mathbf{x}$$

where $\mathbf{A}(i, :)$ denotes the ith row of the matrix $\mathbf{A}$.

This is not compatible with the column-wise specification of $\mathbf{A}$.

(If the blurring is spatially invariant, then there is a simple relation between the rows and columns of $\mathbf{A}$.)

But instead of working out the specific relationship, we use the well-known concept of a *convolution*.

Two-Dimensional Convolution (Preview of Ch. 4)

In a *2D convolution*, the new image $\mathbf{B}$ is created by applying a filter $\mathbf{P}$ to the original image $\mathbf{X}$:

$$\mathbf{B} = \mathbf{P} * \mathbf{X}.$$

The convolution operation "$*$" can be specified as follows:

- Rotate $\mathbf{P}$ 180 degrees.
- For all i, j in $\mathbf{X}$:
  1. Center the rotated $\mathbf{P}$ at pixel i, j in $\mathbf{X}$.
  2. Multiply corresponding components in $\mathbf{X}$ and the rotated $\mathbf{P}$.
  3. Sum the results in pixel i, j of $\mathbf{B}$.

This is precisely the same as computing $\mathbf{b} = \mathbf{A}\,\mathbf{x}$ when $\mathbf{b} = \text{vec}(\mathbf{B})$, $\mathbf{x} = \text{vec}(\mathbf{X})$ and $\mathbf{P}$ is the PSF array for spatially invariant blur!

2D Convolution by Example

For example, if $\mathbf{P}$ has the form

$$\mathbf{P} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

and if b_{ij} and x_{ij} are the elements/pixels of $\mathbf{B}$ and $\mathbf{X}$, then

$$\begin{aligned} b_{ij} \quad = \quad & 9 \cdot x_{i-1,j-1} + 8 \cdot x_{i-1,j} + 7 \cdot x_{i-1,j+1} + \\ & 6 \cdot x_{i,j-1} \quad + 5 \cdot x_{i,j} \quad + 4 \cdot x_{i,j+1} \quad + \\ & 3 \cdot x_{i+1,j-1} + 2 \cdot x_{i+1,j} + 1 \cdot x_{i+1,j+1}. \end{aligned}$$

This illustrates a *fundamental principle* in image deblurring:

> The matrix-vector notation $\mathbf{A}\,\mathbf{x} = \mathbf{b}$ is useful for analysis, but working with the images and the PSF array $\mathbf{P}$ is more practical!

Convolution and Filtering

In the signal and image processing communities, convolutions are often used for filtering ($\rightarrow$ Challenge 7).

- **Low-pass filtering** (local averaging) for noise removal:

$$\mathbf{P} = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \text{or} \quad \mathbf{P} = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}.$$

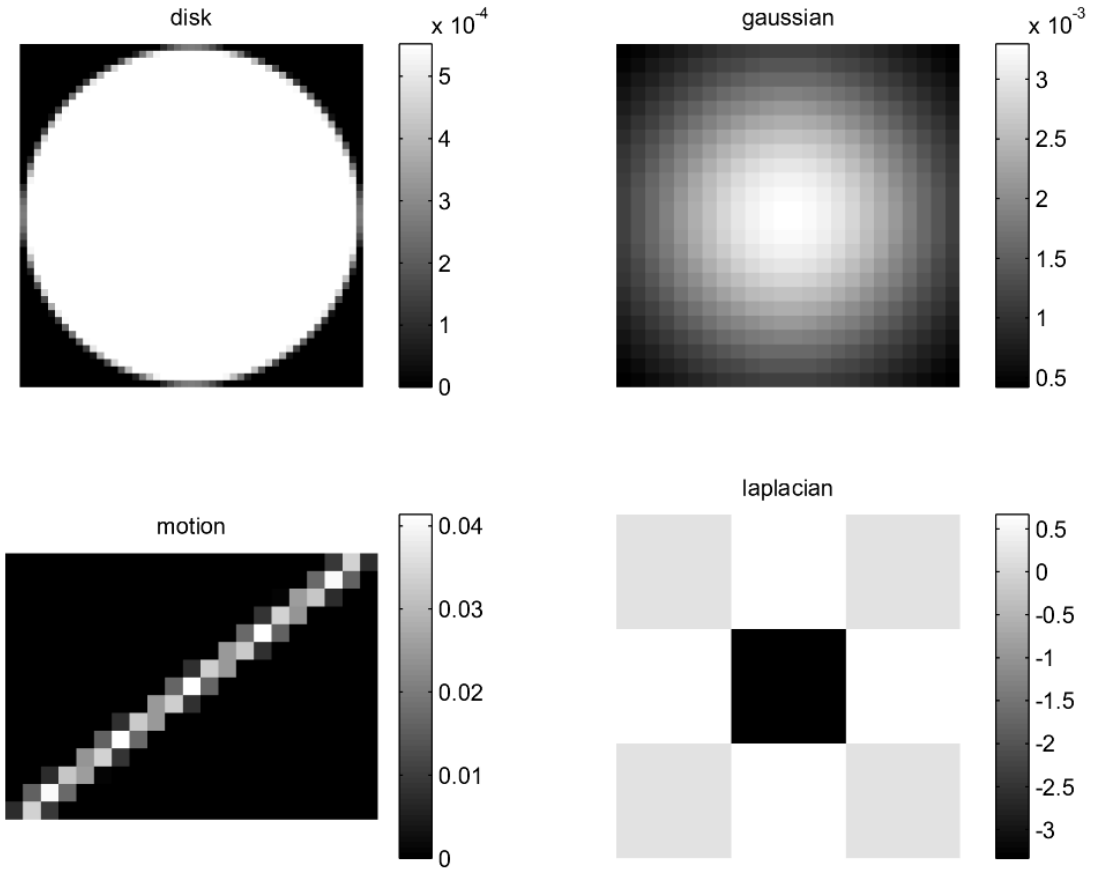
- **High-pass filtering** (computing derivatives) finds edges:

$$\mathbf{P} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad \text{or} \quad \mathbf{P} = \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}.$$

- **Edge enhancement:** a weighted average of the original image and the high-pass filtered image enhances the edges.

Spatial Filters in Matlab's Image Processing Toolbox (IPT)

```
P = fspecial('type',parameters)
```



2D Convolution in Matlab

```
CONV2 Two dimensional convolution.
   C = CONV2(A, B) performs the 2-D convolution of matrices
   A and B.   If [ma,na] = size(A) and [mb,nb] = size(B), then
   size(C) = [ma+mb-1,na+nb-1].
   C = CONV2(H1, H2, A) convolves A first with the vector H1
   along the rows and then with the vector H2 along the columns.

   C = CONV2( ... ,'shape') returns a subsection of the 2-D
   convolution with size specified by 'shape':
     'full'  - (default) returns the full 2-D convolution,
     'same'  - returns the central part of the convolution
               that is the same size as A.
     'valid' - returns only those parts of the convolution
               that are computed without the zero-padded
               edges. size(C) = [ma-mb+1,na-nb+1] when
               all(size(A) >= size(B)), otherwise C is empty.
```

Note that CONV2 assumes zero boundary conditions.

Other Choices in Matlab

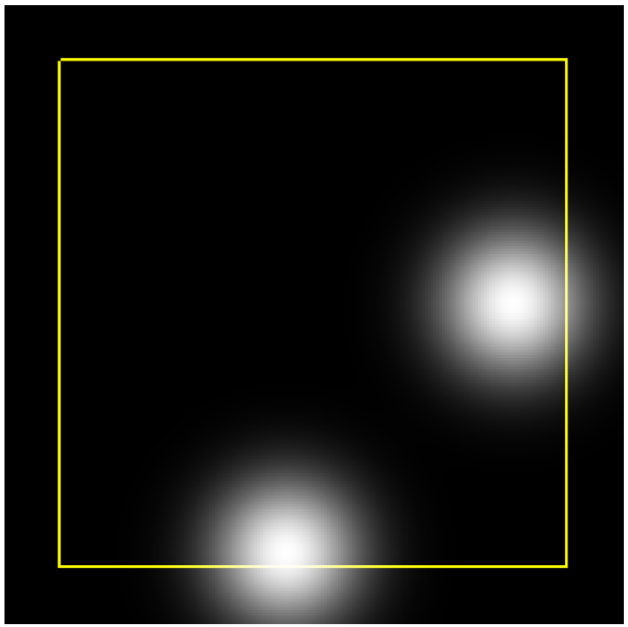
CONV2 is built-in; the IPT provides a more versatile function:

```
IMFILTER N-D filtering of multidimensional images.
   B = IMFILTER(A,H) filters the multidimensional array A with the
   multidimensional filter H.  A can be logical or it can be a
   nonsparse numeric array of any class and dimension.  The result,
   B, has the same size and class as A.

   B = IMFILTER(A,H,OPTION1,OPTION2,...) performs multidimensional
   filtering according to the specified options:

   - Boundary options: number, 'symmetric', 'replicate', 'circular'

   - Output size options: 'same', 'full'

   - Correlation and convolution: 'corr', 'conv'
```

Very Important Points

1. The PSF array **P** is the image of a single white pixel.
2. Its effective dimensions are usually much smaller than those of **B** and **X**.
3. It can often be directly measured or specified mathematically.
4. If the blurring is local and spatially invariant, then **P** contains all information about the blurring throughout the image.
5. In this case, the blurring is most conveniently computed by means of two-dimensional convolution.

Close To The Edge



The yellow line shows the boundary of the blurred image **B**.

The PSF "spills over the edge" at the image boundary, so that information from the exact image **X** is lost in the blurred image **B** that we record.

Also, we see that values of the exact scene outside the border of **B** affect what is actually recorded.

Hence we would be able to compute a better reconstruction if we knew what was outside the recorded image (but we don't).

A good image deblurring model must take account of these effects!

Boundary Conditions

- Most common approach: impose a mathematical expression for the behavior at the image boundary.
- Easy to formulate a "catalogue" of boundary conditions in our language of matrix computations.
- Conceptually, we can formulate this as embedding the image $\mathbf{X}$ in a larger image $\mathbf{X}^{\text{ext}}$.

Zero boundary conditions. Assume that the exact image is black outside the boundary:

$$\mathbf{X}^{\text{ext}} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{X} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}, \qquad \mathbf{0} = \text{zero matrix.}$$

More Boundary Conditions

Periodic. The image repeats itself (endlessly) in both directions:

$$\mathbf{X}^{\text{ext}} = \begin{bmatrix} \mathbf{X} & \mathbf{X} & \mathbf{X} \\ \mathbf{X} & \mathbf{X} & \mathbf{X} \\ \mathbf{X} & \mathbf{X} & \mathbf{X} \end{bmatrix}.$$

Reflexive. The scene outside the image boundary is a mirror image of the scene inside the boundary:

$$\mathbf{X}^{\text{ext}} = \begin{bmatrix} \mathbf{X}_{\times} & \mathbf{X}_{\text{ud}} & \mathbf{X}_{\times} \\ \mathbf{X}_{\text{lr}} & \mathbf{X} & \mathbf{X}_{\text{lr}} \\ \mathbf{X}_{\times} & \mathbf{X}_{\text{ud}} & \mathbf{X}_{\times} \end{bmatrix}, \qquad \begin{aligned} \mathbf{X}_{\text{lr}} &= \texttt{fliplr}(\mathbf{X}) \\ \mathbf{X}_{\text{ud}} &= \texttt{flipud}(\mathbf{X}) \\ \mathbf{X}_{\times} &= \texttt{fliplr}(\mathbf{X}_{\text{ud}}). \end{aligned}$$

How to incorporate these boundary conditions into the deblurring problem is explained in Chapters 4 and 5.

Noise!

We consider two types of additive noise.

- **Image noise** originates from the photons entering the CCD, and is modelled as Poisson noise.
- **Readout noise** originates from the CCD and the analog-to- digital conversion, and is modelled as Gaussian white noise.

Readout noise and background image noise is *additive*, and the noisy blurred image is therefore given by

$$\mathbf{B} = \mathbf{P} * \mathbf{X} + \mathbf{E},$$

where $\mathbf{E}$ is an $m \times n$ array whose elements are from a Gaussian or Poisson distribution (or the sum of both).

Image Photon Noise

The recorded light intensity is due to photons hitting the CCD, and therefore the intensity follows a Poisson distribution. How to introduce the corresponding *photon noise?*
The following two formulations for Gaussian noise are equivalent:

$$b_i = (b_{\text{exact}})_i + e_i, \quad e_i \sim \mathcal{N}\left(0, \eta^2\right)$$

$$b_i \sim \mathcal{N}\left((b_{\text{exact}})_i, \eta^2\right).$$

The latter is more suited for introducing Poisson photon noise:

$$b_i \sim \mathcal{P}\left((b_{\text{exact}})_i\right).$$

In both cases, the expected values are $\mathcal{E}(b_i) = (b_{\text{exact}})_i$. The variances are given by:

$$V(b_i) = \eta^2 \text{ (Gaussian)}, \qquad V(b_i) = (b_{\text{exact}})_i \text{ (Poisson)}.$$

Adding Noise in Matlab

Adding Gaussian noise is easy:

```
B = Bexact + eta*randn(m,n);
```

Adding Poission noise with the Statistics Toolbox;

```
for i=1:m, for j=1:n
    B(i,j) = poissrnd(Bexact(i,j));
end, end
```

Things are easier with the Image Processing Toolbox:

```
B = imnoise(Bexact,'gaussian',0,eta^2);
```

```
B = imnoise(Bexact,'poisson');
```

Other types of noise are also available.

More Very Important Points

1. The matrix in the deblurring model $\mathbf{A}\,\mathbf{x} = \mathbf{b}$ is determined from two ingredients:
   1. the point spread function (PSF), which defines how each pixel is blurred, and
   2. the boundary conditions, which specify our assumptions on the scene just outside our image.
2. Ignoring boundary conditions is equivalent to assuming zero boundary conditions.
3. Noise is always present in a recorded image, and is modelled by Poisson or Gaussian noise (or a sum of both).

Challenge 8 misprint: $\mathbf{B}_{\text{ext}}$ = `conv2(`$\mathbf{X}_{\text{ext}}$`, P, 'same')`.