Linear Filters

Thomas M. Breuel

Linear Filters

Linear Filters

One of the most important concepts in image processing is that of a *linear filter*. Almost every signal and image processing application uses a linear filter somewhere.

We will look at...

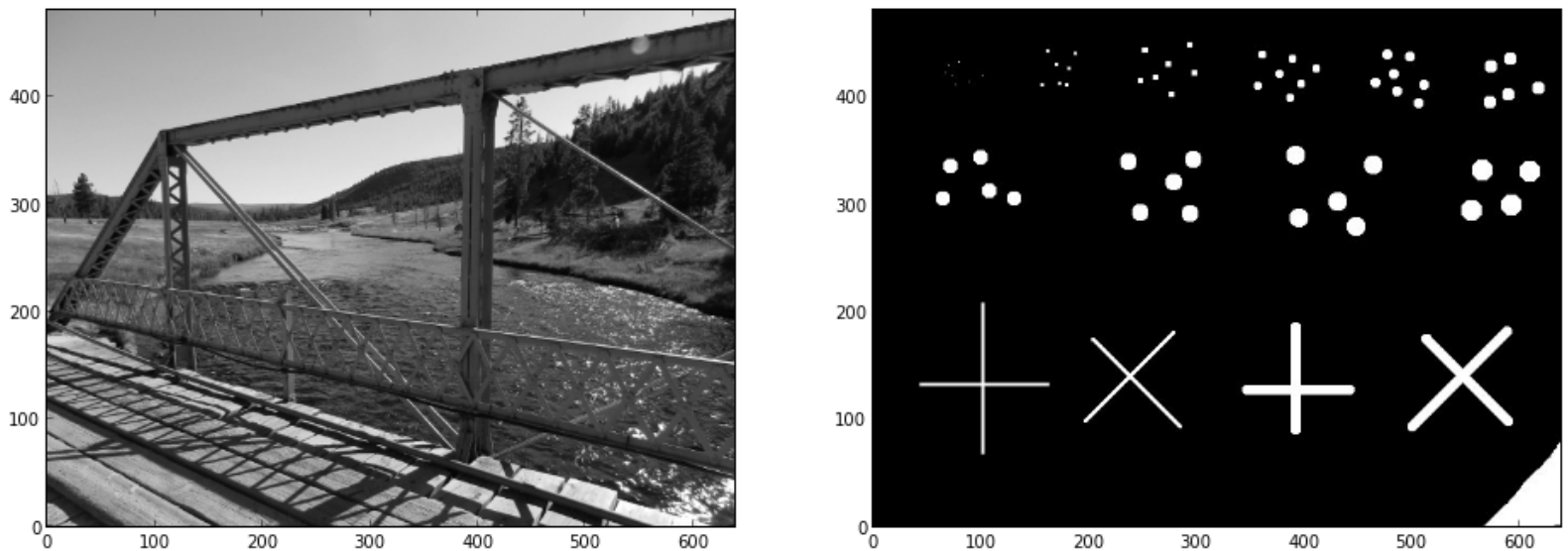
- what linear filters do to images
- statistical justifications for using linear filters
- mathematical properties of linear filters
- transform methods related to linear filters

Test Images

Here are two test images. The first is a natural image. The second is an image with lines at various orientations and dots of different sizes. Observing what happens to its elements helps you understand the action of various filters.

test image examples

```
1 image = mean(imread("bridge.jpg")/255.0,axis=2)[::-1]
2 test = mean(imread("testimage.jpg")/255.0,axis=2)[::-1]
3 imrow(image,test)
```

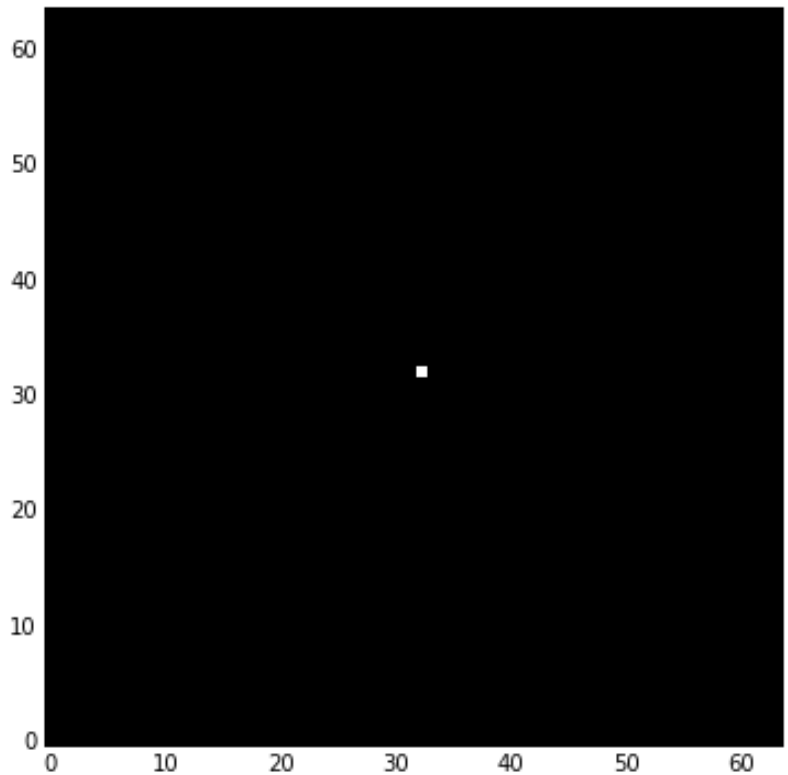


Impulse

An impulse is just an image consisting of all zeros, except for a single pixel that has the value 1.0, at the center. Impulse responses are like our test image above, but they actually fully characterize linear filters, as we shall see below.

2D array representing an impulse

```
1 impulse = zeros((64,64))
2 impulse[32,32] = 1.0
3 imp(impulse)
```

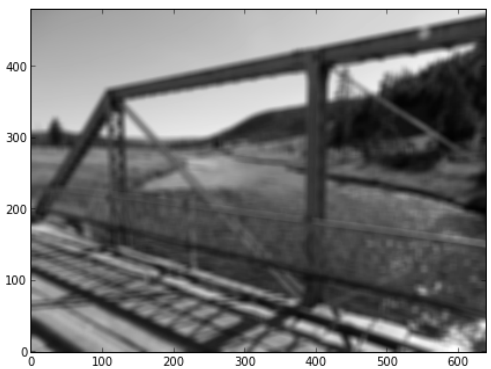


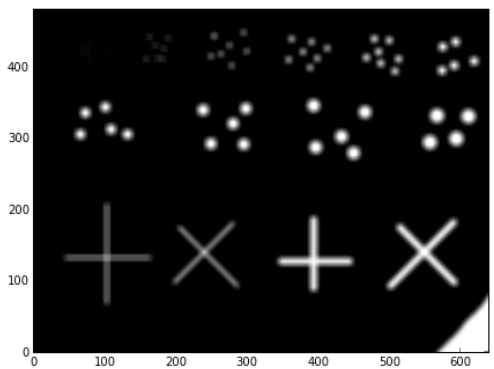
Box Filter and Gaussian Filter

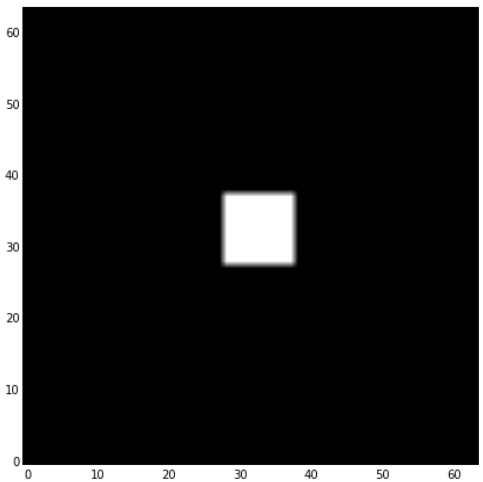
Let's start with a box filter (or uniform filter). You can think of this as "local averaging" of the pixels of an image. That is, you replace each pixel with an average of the pixels around it.

filtering with a box filter, aka uniform filter

```
1 from scipy.ndimage.filters import uniform_filter as bf
2 imrow(bf(image,10.0),bf(test,10.0),bf(impulse,10.0))
```



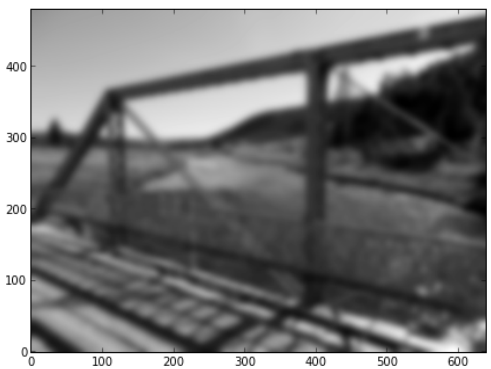


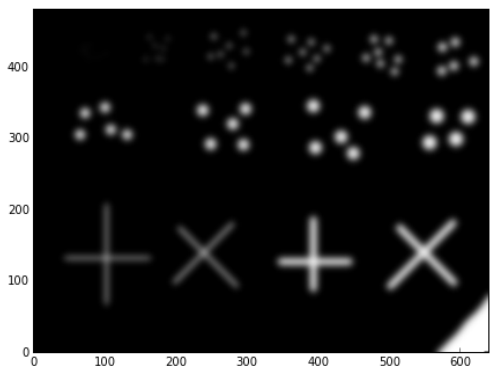


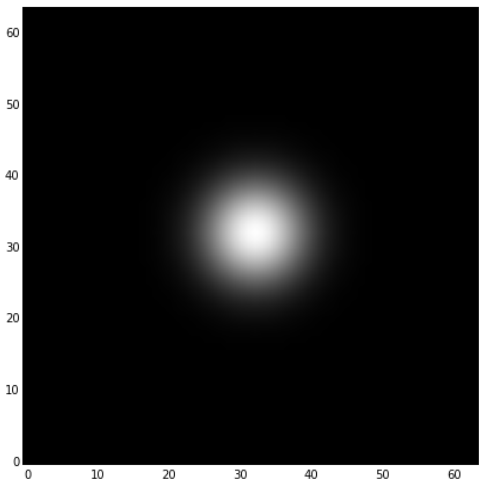
A Gaussian filter is similar to a box filter, but instead of taking an average over a neighborhood, you take a weighted average. The further a pixel is away from the center pixel, the less it contributes to the average. The weight of the averages is given by the Gaussian function (see below). Using it has numerous mathematical and practical advantages. This is one of the most common filters used in practice. You can "read off" the weights in the impulse response.

gaussian filter

```
1 from scipy.ndimage.filters import gaussian_filter as gf
2 imrow(gf(image,5.0),gf(test,5.0),gf(impulse,5.0))
```







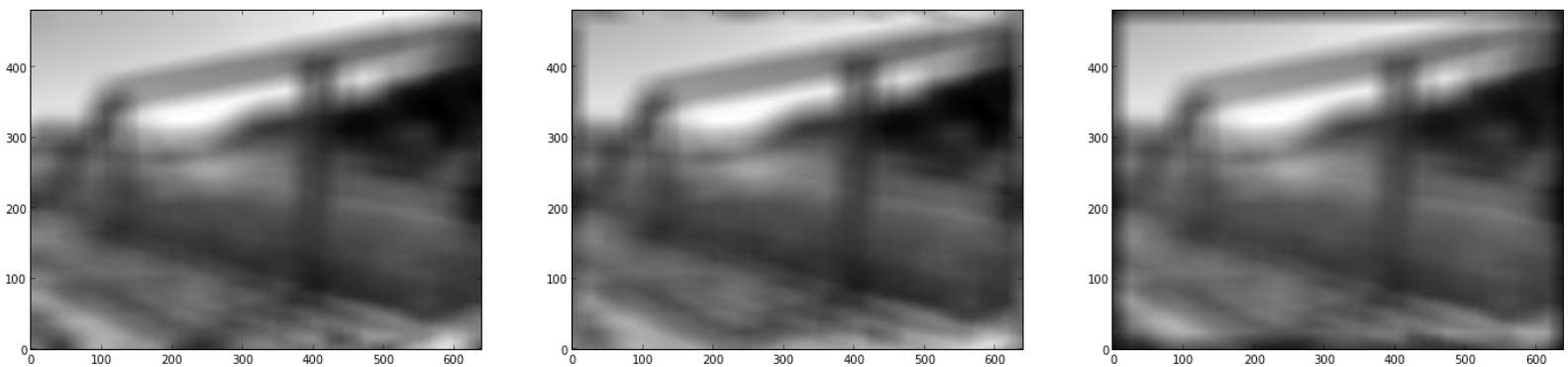
Boundary Conditions

When computing local averages like this, you need to figure out what to do at the boundaries. The most obvious choice would be to compute the average only over the pixels within the image, but that turns out not to be so useful in practice. Instead, averages are always computed over the same number of pixels, but there are different choices for how we fill in the missing pixels. Three common choices are 'reflect', 'wrap', and 'constant'.

These are called *boundary conditions*.

reflect, wrap, constant boundary conditions

```
1 from scipy.ndimage.filters import uniform_filter as bf
2 imrow(bf(image,50.0,mode='reflect'),
3        bf(image,50.0,mode='wrap'),
4        rescale(bf(image,50.0,mode='constant',cval=0.0)))
```



Noise Reduction

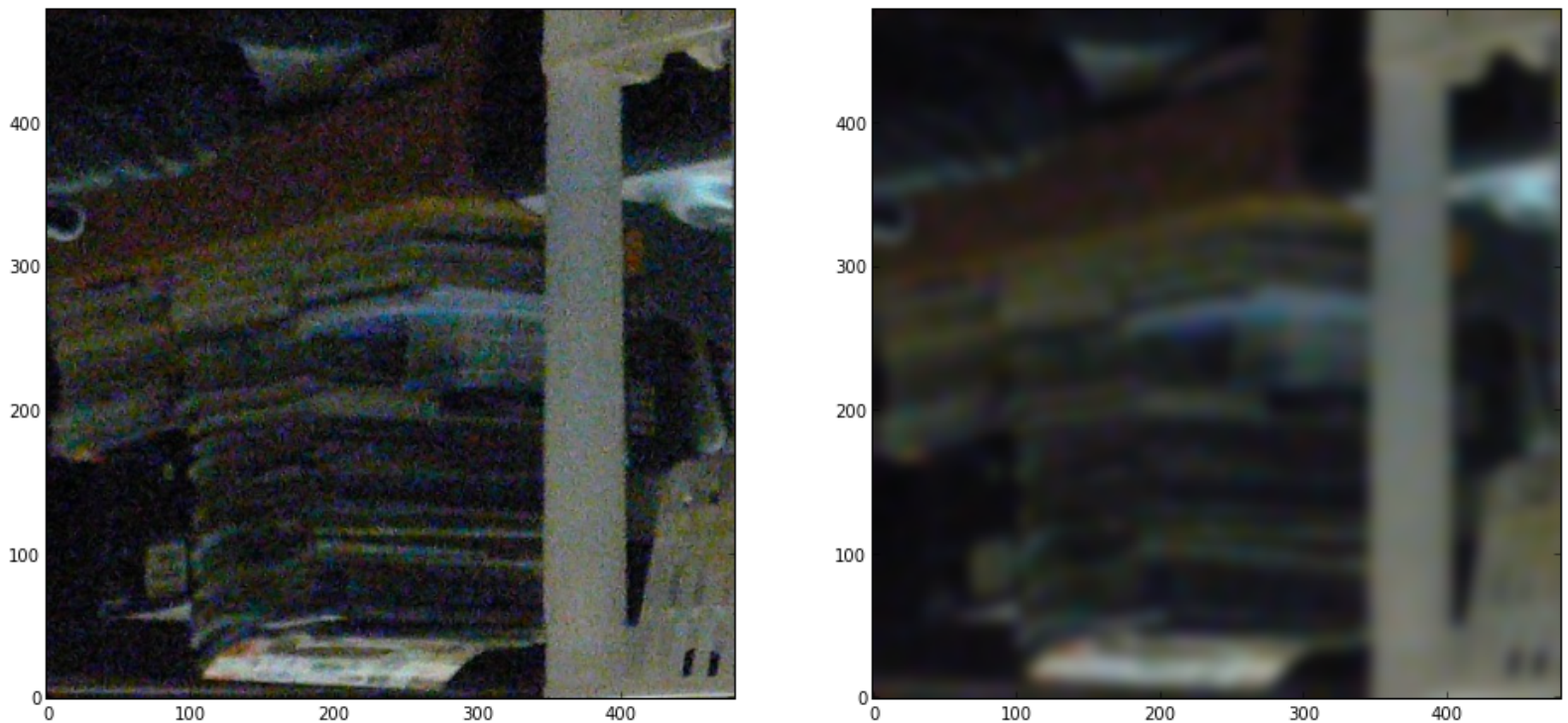
Why do we make our pictures blurry like this?

Usually, it is to *reduce noise* or *reduce image detail*.

Below is an example of a noisy image taken with a digital camera. Next to it is the same image smoothed with a Gaussian. As you can see, there is a lot less noise in the second image. But in return, the image has become blurry.

noise reduction by blurring

```
noise = imread("imagenoise.jpg")
imrow(noise,gf(noise,(5.0,5.0,0.0)))
```

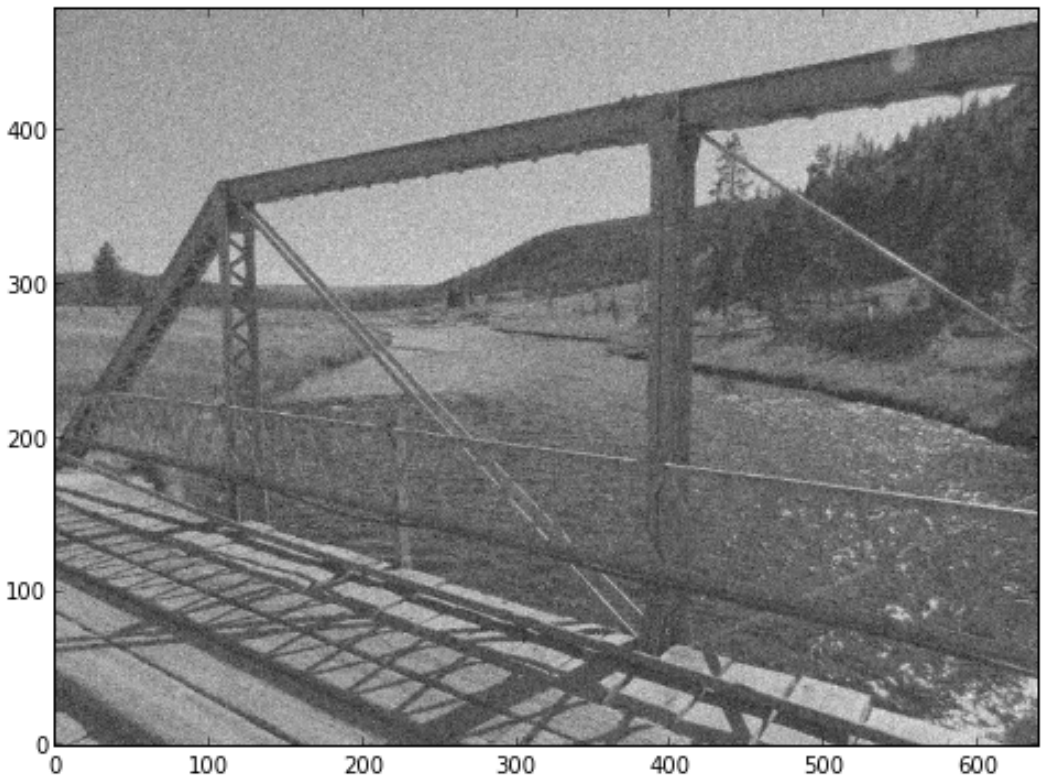


Stack of Noisy Images

It's easy to understand why smoothing reduces noise. Assume that instead of a single image, we have a stack of noisy images. (We actually get this kind of problem in astrophotography.)

stack of noisy images

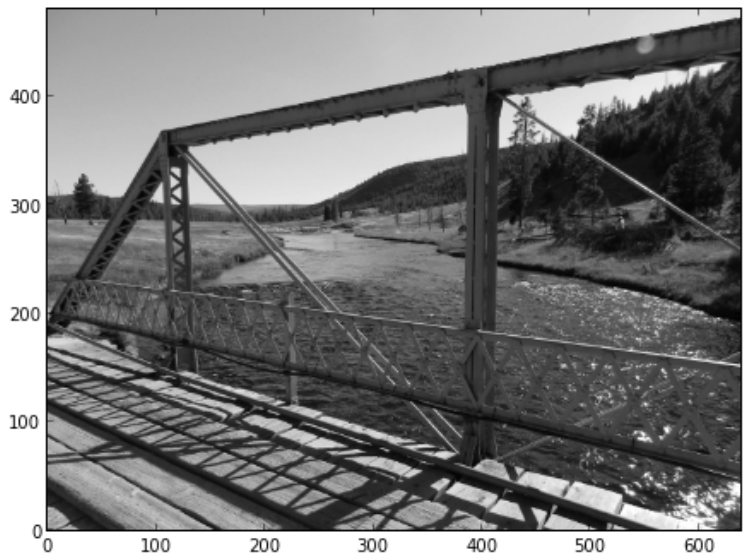
```
noisy = array([image+0.1*randn(*image.shape)
                    for i in range(50)])
ims(noisy[10])
```

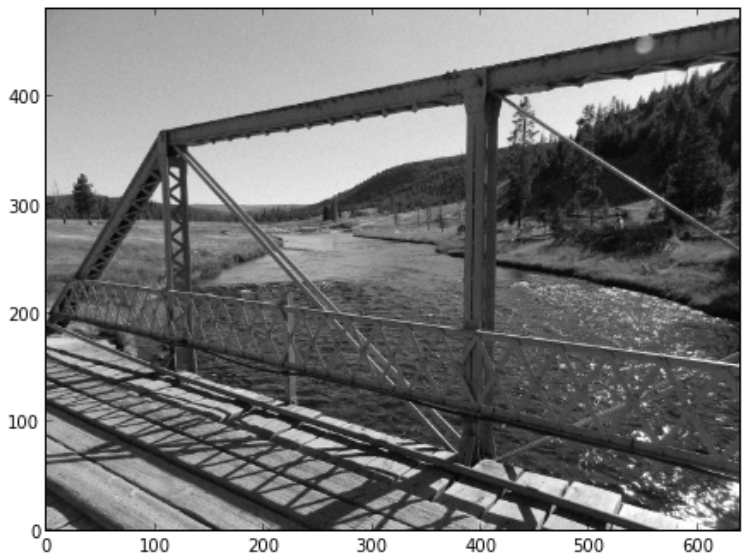


That is, we are now getting multiple measurements per pixel. Each measurement is noisy. But the noise starts to cancel out as we average all the measurements.

noise reduction by averaging a stack of noisy images

```
avg = mean(noisy,axis=0)
imrow(image,clip(avg,0.0,1.0))
```





averaging of images

Mathematically, we have a signal s and noise ν . Each measurement x_i is the sum of the signal and the noise, $x_i = s + \nu$. (Here, ν is a random variable.) When we average a lot of measurements we get:

$$\bar{x} = \frac{1}{N} \sum_{i}^{N} x_i \tag{1}$$

averaging of images

By linearity

$$\bar{x} = \frac{1}{N}\sum_{i}^{N} x_i \tag{2}$$

$$= \frac{1}{N}\sum_{i}^{N}(s + \nu_i) \tag{3}$$

$$= \frac{1}{N}Ns + \frac{1}{N}\sum_{i}^{N}\nu_i \tag{4}$$

$$= s + \frac{1}{N}\sum_{i}^{N}\nu_i \tag{5}$$

limit

As $N \to \infty$, we obtain:

$$E[x] = E[s] + E[\nu] = s + E[\nu] \tag{6}$$

If the additive noise has expectation value 0 , then

$$E[x] = s \tag{7}$$

Linearity

definition of linear filters

This entire worksheet is about *linear filters*. We haven't yet said what that means.

Linearity for a filter F is defined as follows.

Given any two signals A and B , and a real number α , a filter is linear if the following two identities hold:

$$F[A + B] = F[A] + F[B] \tag{8}$$

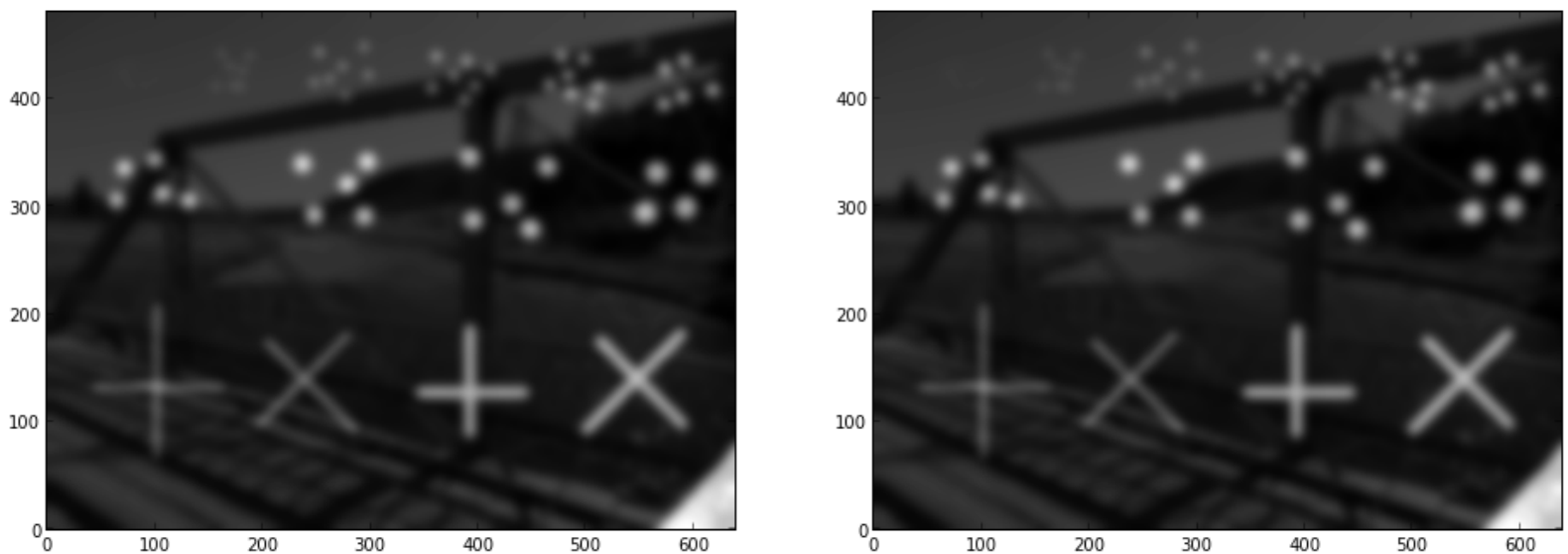
$$F[\alpha A] = \alpha F[A] \tag{9}$$

(Here, we write application of the filter F to a signal A as $F[A]$, to indicate that the argument is a signal. We might also write it as $F(A)$.)

demonstration of linearity

```
1 def F(image): return gf(image,5.0)
2 result1 = F(image)+2.0*F(test)
3 result2 = F(image+2.0*test)
4 imrow(result1/3.0,result2/3.0)
5 print amax(abs(result1-result2))
```

```
8.881784197e-16
```



Linearity has important consequences, as we will see below.

Shift Invariance

We call a filter *shift invariant* if applying it to a shifted image is the same as shifting the result of applying the filter.

Define the *shift filter* S as:

$$S_{(u,v)}[I]\,(x,y) = I(x+u, y+v)$$

Then a filter is shift invariant if:

$$S[F[I]] = F[S[I]]$$

or equivalently:

$$F[I] = S^{-1}[F[S[I]]]$$

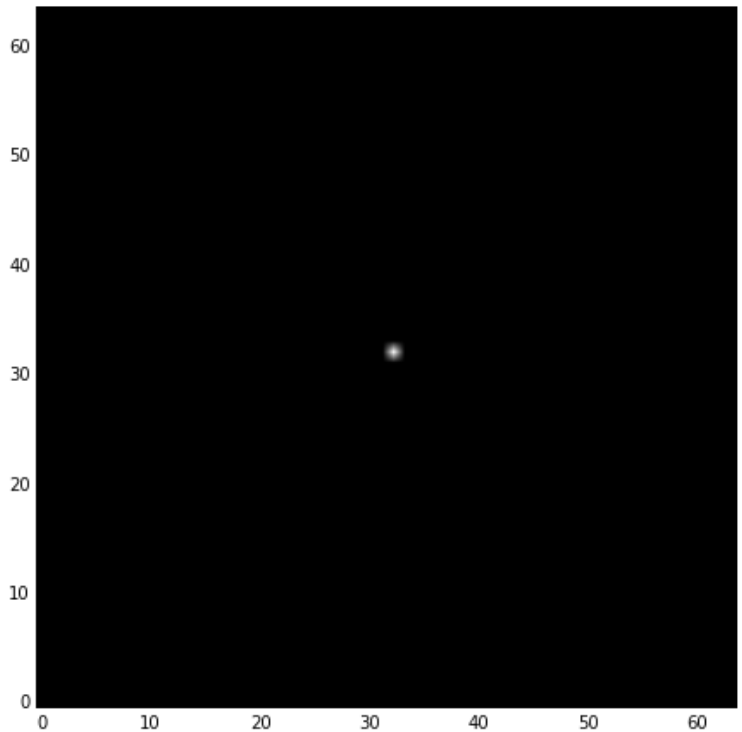
Note that S itself is both linear and shift invariant.

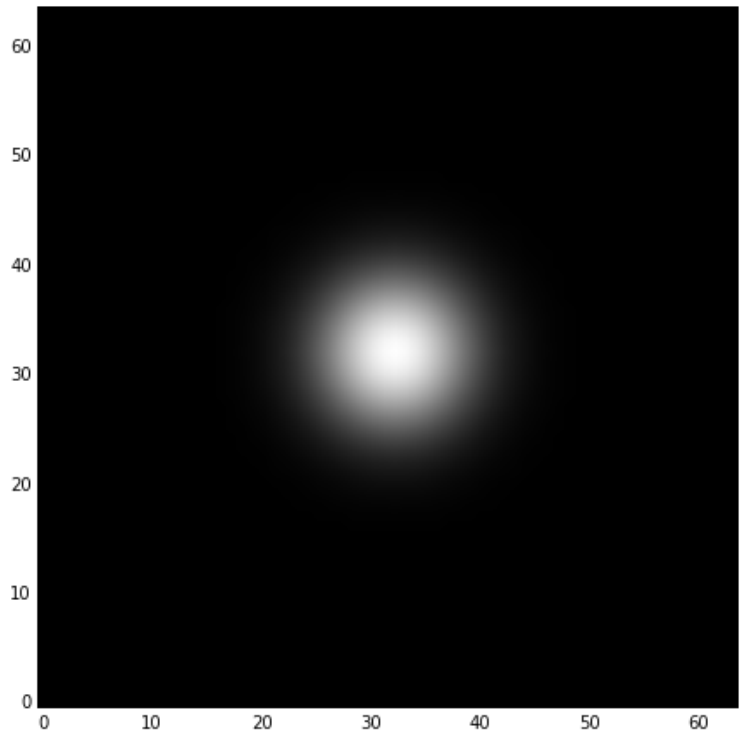
Impulse Response and Convolution

Recall that the impulse response of a filter is the output of filtering an image consisting of all zeros except a single value of 1.0 at the center.

impulse response of Gaussian filter

```
1 mask = gf(impulse,5.0)
2 imrow(impulse,mask)
```





The impulse response is important because it completely characterizes a linear shift invariant filter. Why? Because we can decompose an image into impulses, then apply the impulse response to each filter, and then add it all up again.

definitions

Let the input image be I and pixel (i, j) given by I_{ij} .

Write the impulse signal as δ .

Also, let S_{ij} be a filter that shifts the image such that the pixel at the origin ends up at position i, j . It's easy to see that S is also a linear filter.

Now,

$$I = \sum S_{ij}[I_{ij} \cdot \delta] \tag{10}$$

impulse response and filters

$$F[I] = F[\sum S_{ij}[I_{ij}\delta]] \tag{11}$$

By linearity:

$$= \sum F[S_{i,j}[I_{ij}\delta]] \tag{12}$$

By shift invariance:

$$= \sum S_{i,j}[F[I_{ij}\delta]] \tag{13}$$

By linearity, since I_{ij} is just a number:

$$= \sum S_{i,j}[I_{ij}F[\delta]] \tag{14}$$

Here, $F[\delta]$ is the impulse response.

equivalence

Therefore, applying a linear filter is equivalent to just summing up its impulse response.

convolution

The operation of combining an image with the impulse response of a filter is called the *convolution*.

Ignoring boundary conditions, for infinitely large 1D signals, it is written as:

$$(x * y)_i = \sum_{j=-\infty}^{\infty} x_j y_{i-j} \tag{15}$$

properties of convolutions

Convolution has the following properties:

- commutativity: $x * y = y * x$
- associativity: $x * (y * z) = (x * y) * z$
- distributivity: $x * (y + z) = x * y + x * z$
- scalar multiplication: $\alpha(x * y) = (\alpha x) * y = x * (\alpha y)$

2D convolution

In 2D, convolution is defined analogously.

These properties carry over to linear shift invariant filters, since convolutions and linear shift invariant filters are equivalent.

cross-correlation

Note that there is a closely related operation where the minus sign has been replaced with a plus, the *cross-correlation*:

$$(x \star y)_i = \sum_{j=-\infty}^{\infty} x_j y_{i+j} \tag{16}$$

The minus sign has the effect of "flipping" the mask around. This is also a useful operation, but the algebraic identities above don't quite hold anymore.

convolution in the library

Convolution is implemented by the `convolve` function.

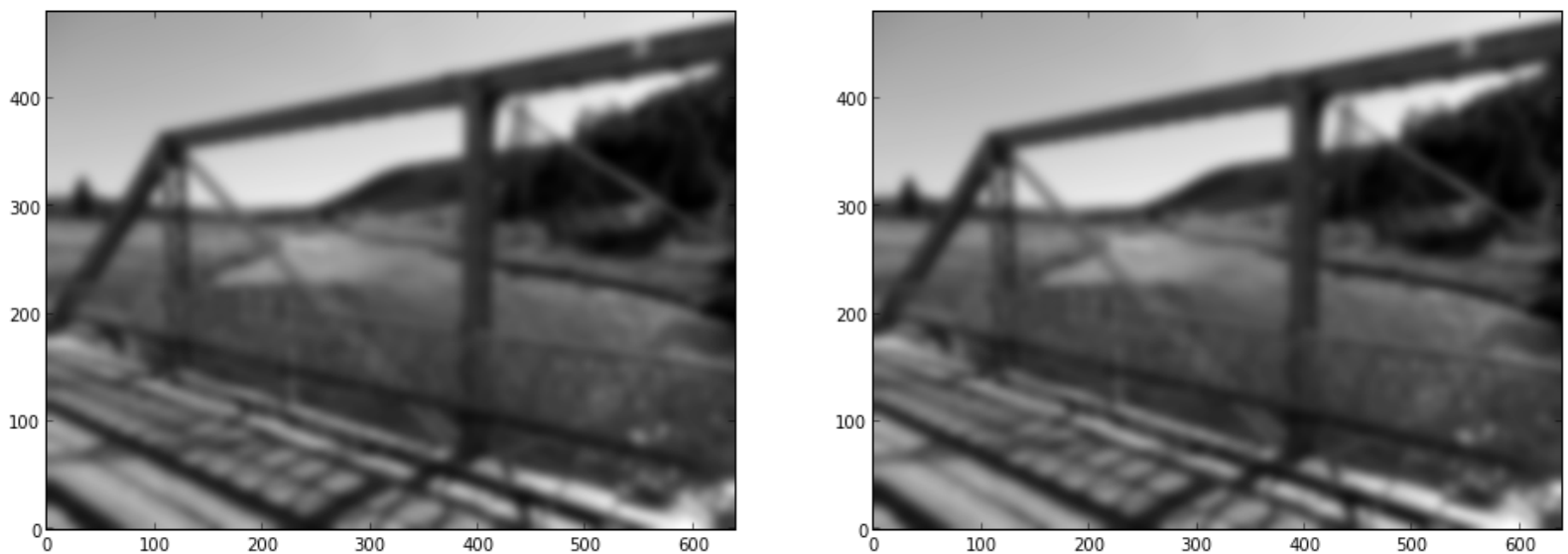
Let us check that applying linear filter $g(\cdot, 5)$ is the same as convolving with its impulse response.

That is, $g(x, 5) = x * g(\delta, 5)$

convolving with the impulse response

```
result1 = gf(image,5.0)
mask = gf(impulse,5.0)
result2 = filters.convolve(image,mask)
imrow(result1,result2)
print amax(abs(result1-result2))
```

```
5.10702591328e-15
```



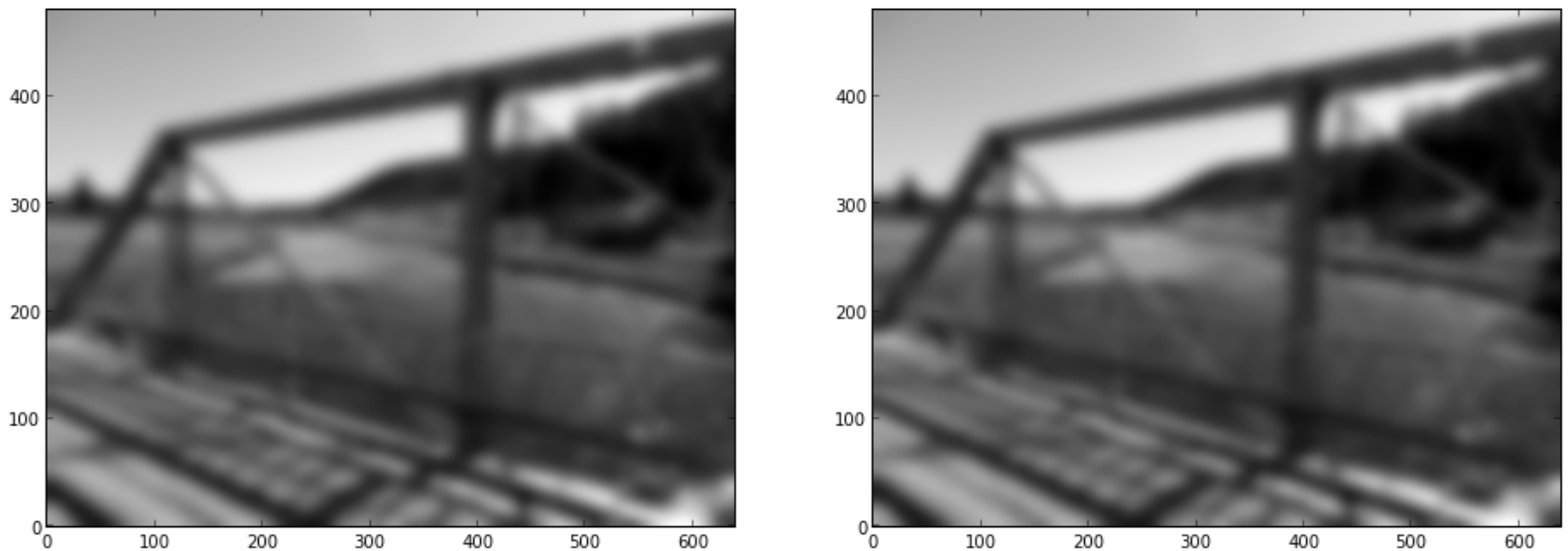
Composition of Filters

Let us check associativity of filters. This is a very useful property of linear filters because instead of applying a sequence of tilers, we can pre-compose the filters and then just apply the filter once.

composition of linear filters

```
1 result1 = filters.convolve(filters.convolve(image,mask),mask)
2 result2 = filters.convolve(image,filters.convolve(mask,mask))
3 imrow(result1,result2)
4 print amax(abs(result1-result2))
```

```
2.21133870582e-06
```



composition of Gaussian filters

Does this work for Gaussian filters?

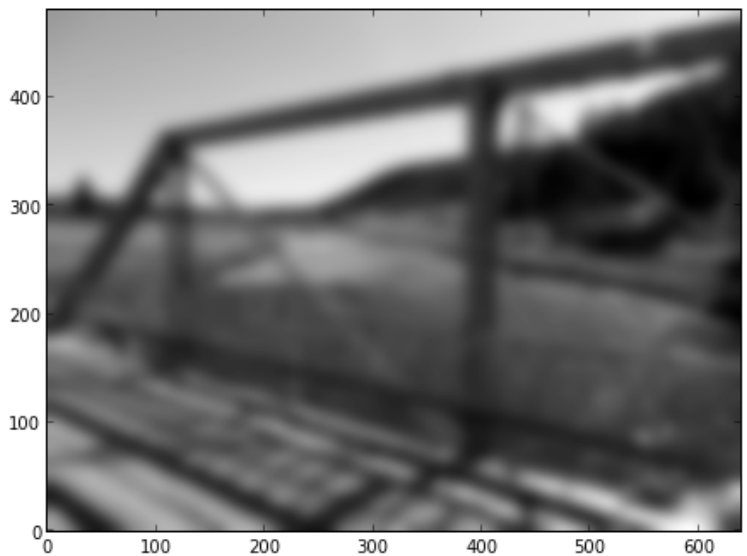
Yes, but there is one small thing to watch out for: the parameter that we give to the Gaussian filter does not quite combine in the way we expect. Applying two Gaussian filters of width 5 is equivalent to applying on one of width $\sqrt{5^2 + 5^2}$.

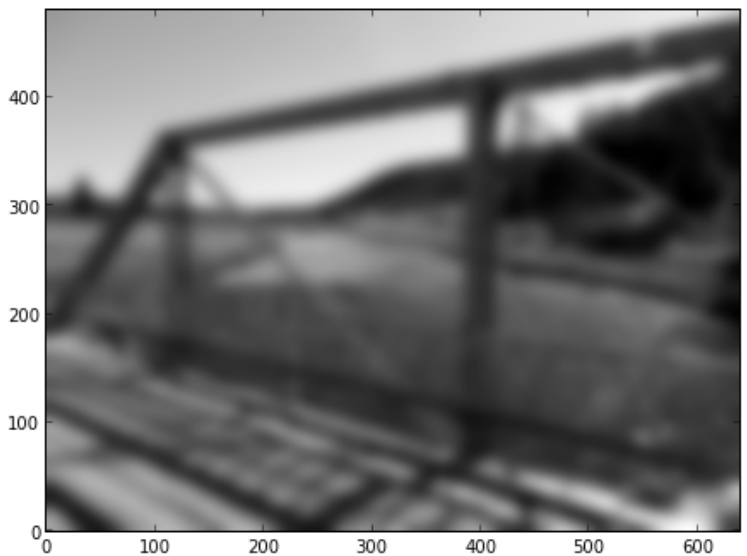
(Looking at the definition of the Gaussian and Gaussian filter below, work out for yourself why these parameters behave like that.)

composition of Gaussian filters

```
1 result1 = gf(gf(image,5.0),5.0)
2 result2 = gf(image,sqrt(5**2+5**2))
3 imrow(result1,result2)
4 print amax(abs(result1-result2))
```

```
4.05664875084e-05
```





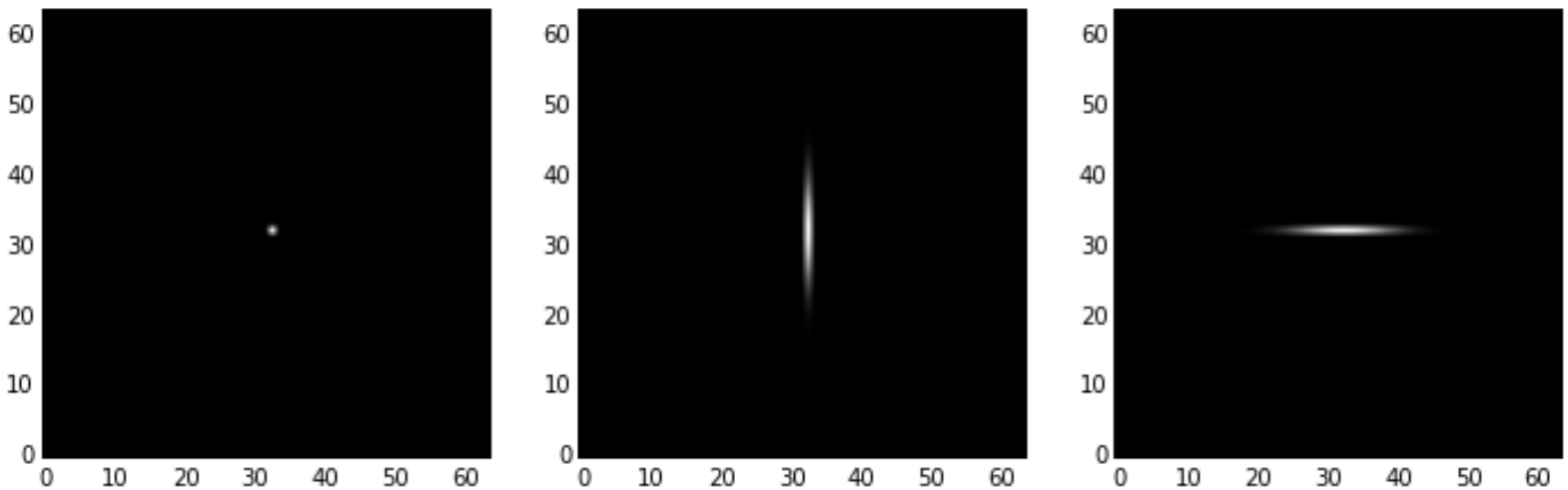
Separability

Many filtes are separable. That is, instead of convolving with a 2D mask, we can convolve with two 1D masks.

For Gaussian filters, we can actually specify the width and the height of the filter separately. Applying two filters of shape (5,0) and (0,5) is equivalent to applying one filter of shape (5,5).

horizontal and vertical Gaussians

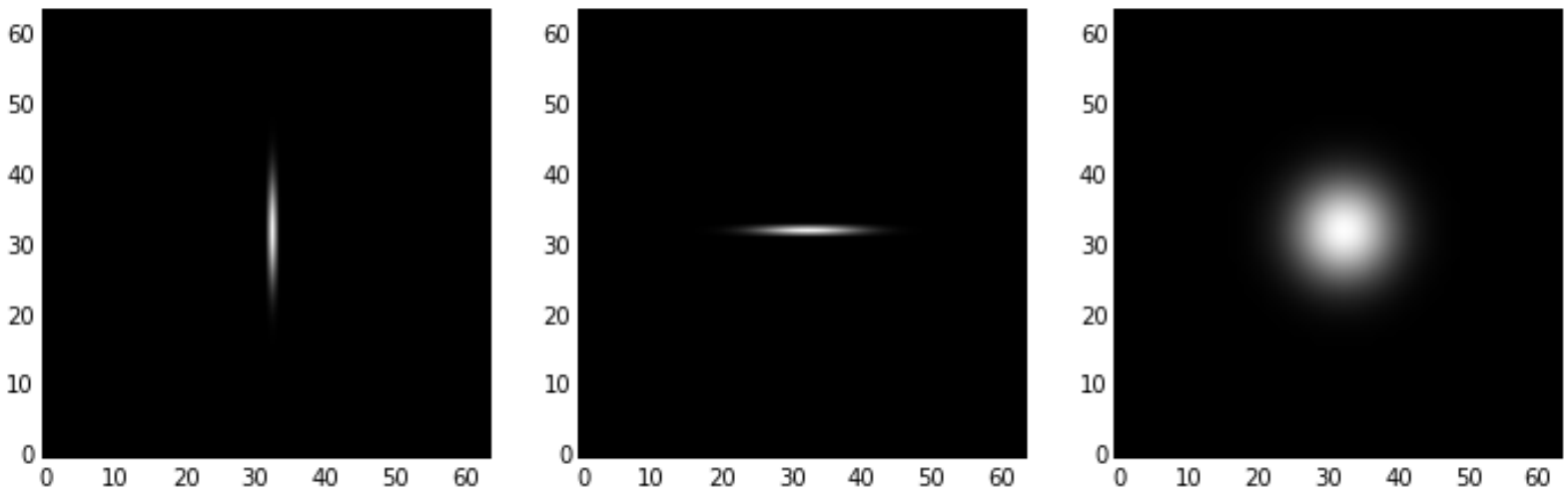
```
1 vf = gf(impulse,(5.0,0.0))
2 hf = gf(impulse,(0.0,5.0))
3 imrow(impulse,vf,hf,s=4)
```



The convolution of these two filters is the same as a Gaussian.

convolution of horizontal and vertical Gaussians

```
1 imrow(vf,hf,filters.convolve(vf,hf),s=4)
```

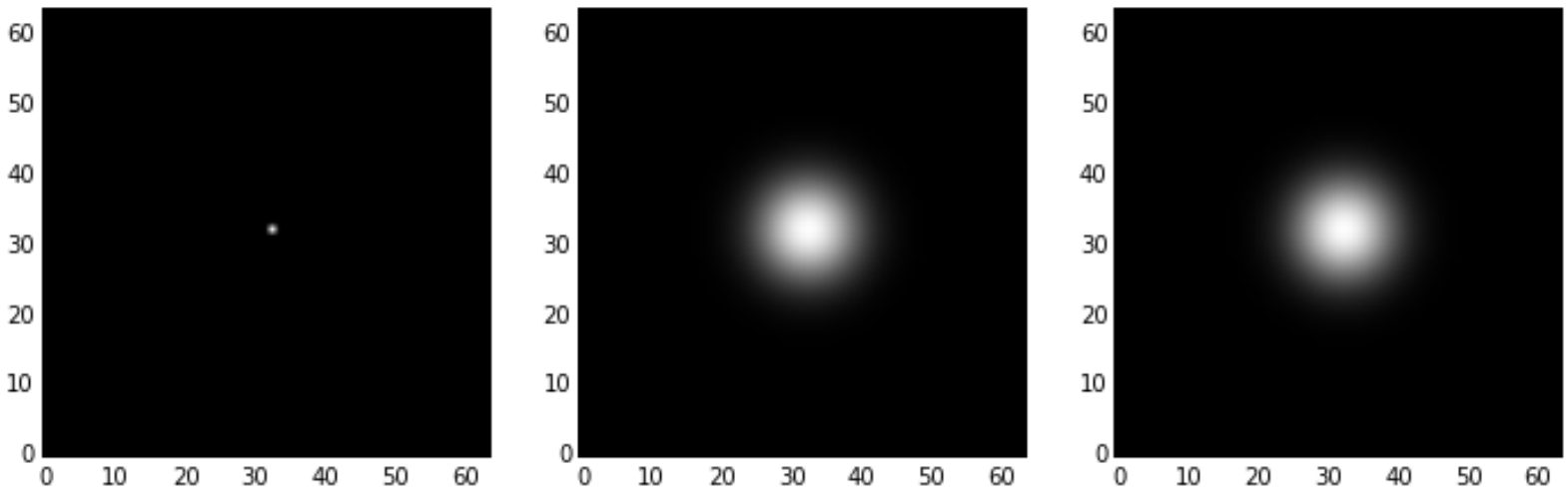


These properties now carry over to the filters themselves.

Note that $g(\cdot, (5, 0))$ is implemented more efficiently than $g(\cdot, (5, 5))$.

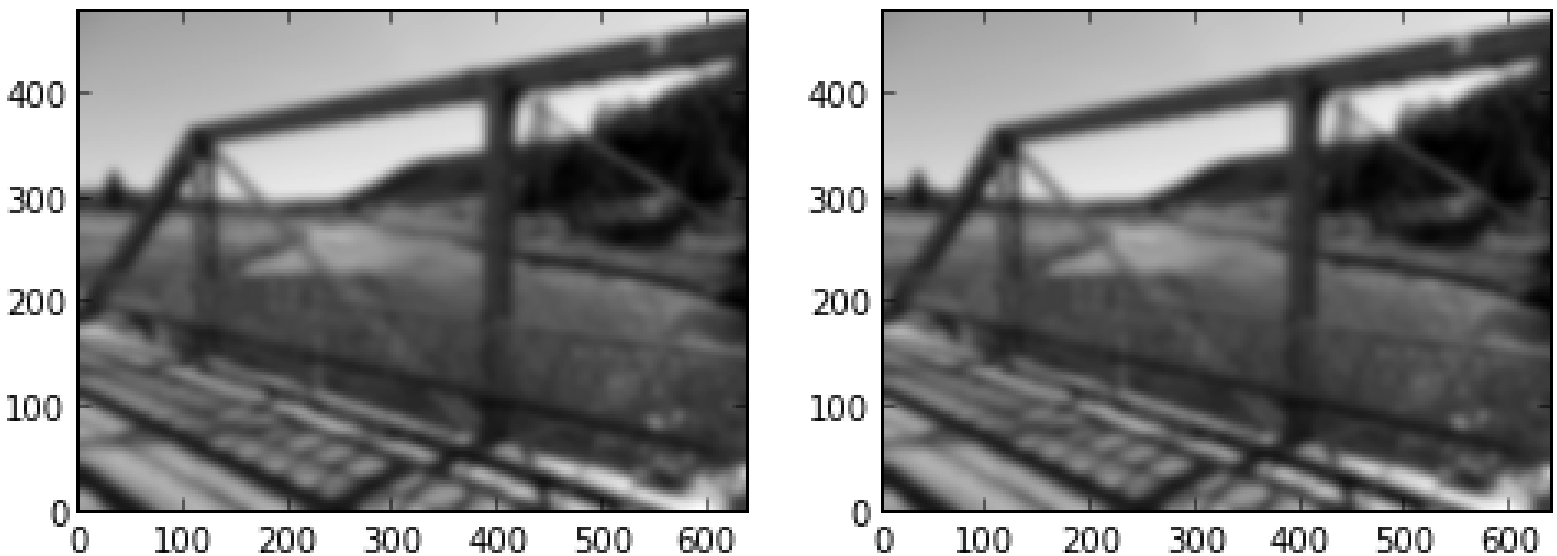
separable convolution

```
1 imrow(impulse,gf(impulse,(5.0,5.0)),gf(gf(impulse,(5.0,0.0))
      ,(0.0,5.0)),s=4)
```



separable convolution

```
1 imrow(gf(image,(5.0,5.0)),gf(gf(image,(5.0,0.0)),(0.0,5.0)),s=4)
```



speed of separable vs full 2D convolution

```
1 import timeit
2 print "2D convolution",
3 print timeit.timeit(lambda: filters.convolve(image,mask),number=1)
4 print "separable convolution",
5 print timeit.timeit(lambda: gf(gf(image,(5.0,0.0)),(0.0,5.0)),
      number=100)/100.0
6 print "Gaussian library",
7 print timeit.timeit(lambda: gf(image,(5.0,5.0)),number=100)/100.0
```

```
2D convolution 1.08510494232
separable convolution 0.024929189682
Gaussian library 0.0241062402725
```

speed advantage

2D convolution is 50$\times$ slower than separable convolution.

Python libraries obviously use separability for Gaussian convolution.

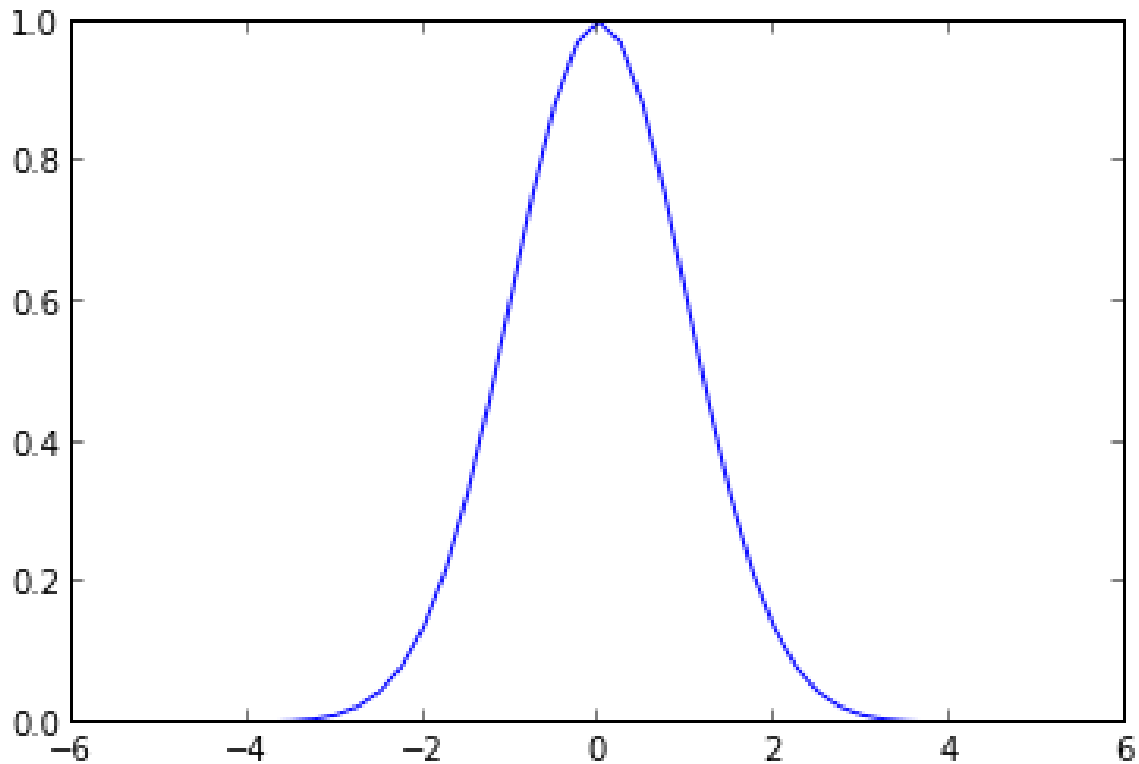
Gaussians

We keep talking about Gaussian filters, but haven't really looked at what those weights are.

They are given by the following simple function.

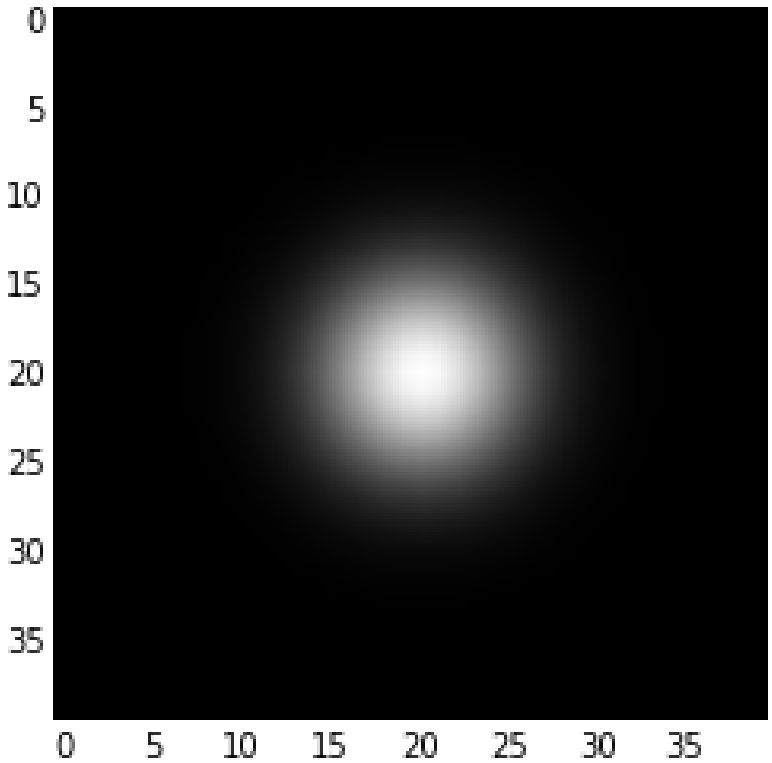
Gaussians

```
1 xs = arange(-5.0,5.0,0.25)
2 ys = exp(-xs**2/2.0)
3 plot(xs,ys)
```



For 2D filtering, we compute the outer product of these 1D masks. This is analogous to the convolution of the horizontal and vertical masks above, where we looked at separability.

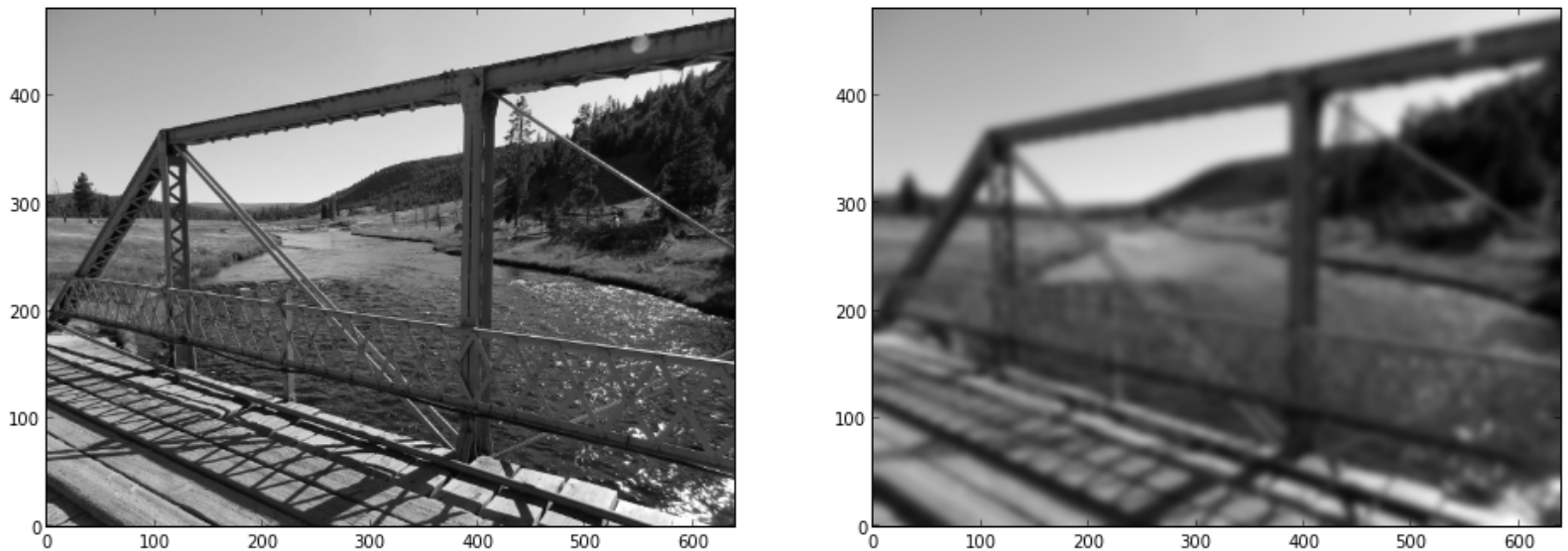
```
mask = outer(ys,ys)
mask /= sum(mask)
imshow(mask)
```



Now let's just look at how this filter looks.

convolution with an explicitly constructed Gaussian mask

```
1 imrow(image,filters.convolve(image,mask))
```



Data Parallel Convolution

Above, we had already talked about linear filters as kinds of local averaging operations.

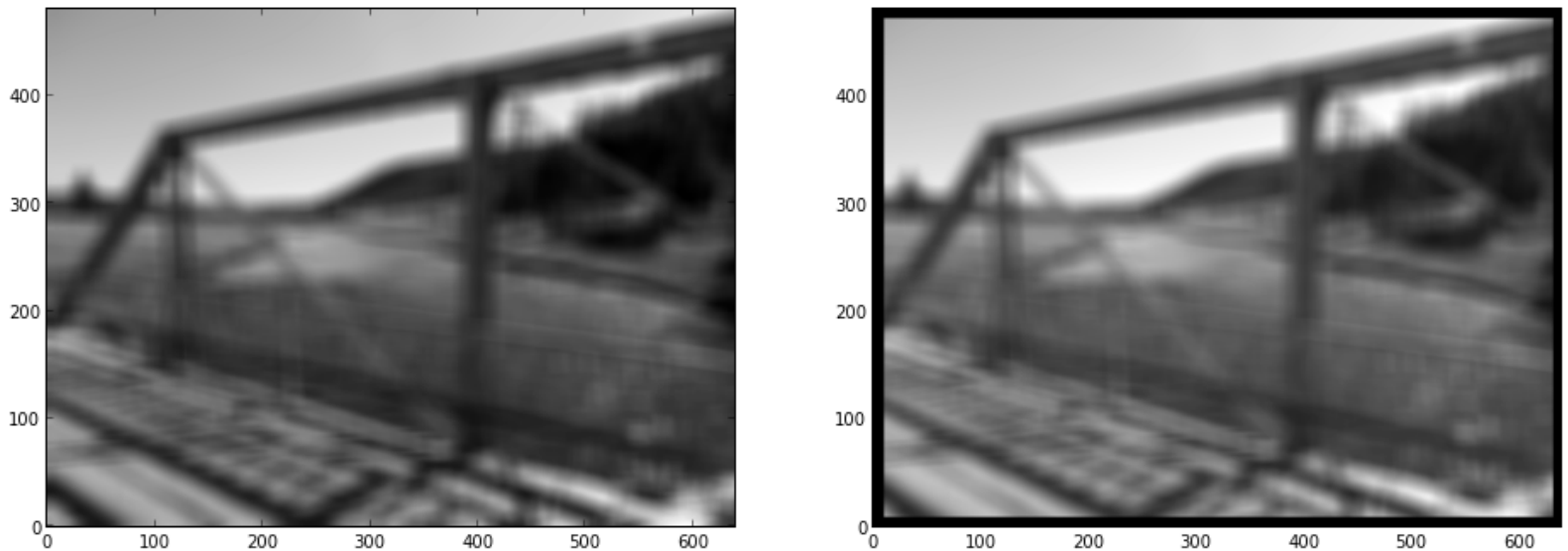
A simple implementation of this is to actually compute local averages. Here is Python code that does this.

nested loop implementation of a box filter

```
1 def dumb_boxfilter(image,r):
2     r /= 2
3     w,h = image.shape
4     out = zeros(image.shape)
5     for i in range(r,w-r):
6         for j in range(r,h-r):
7             out[i,j] = mean(image[i-r:i+r,j-r:j+r].ravel())
8     return out
```

library vs nested loop for box filter

```
1 imrow(filters.uniform_filter(image,20),dumb_boxfilter(image,20))
```



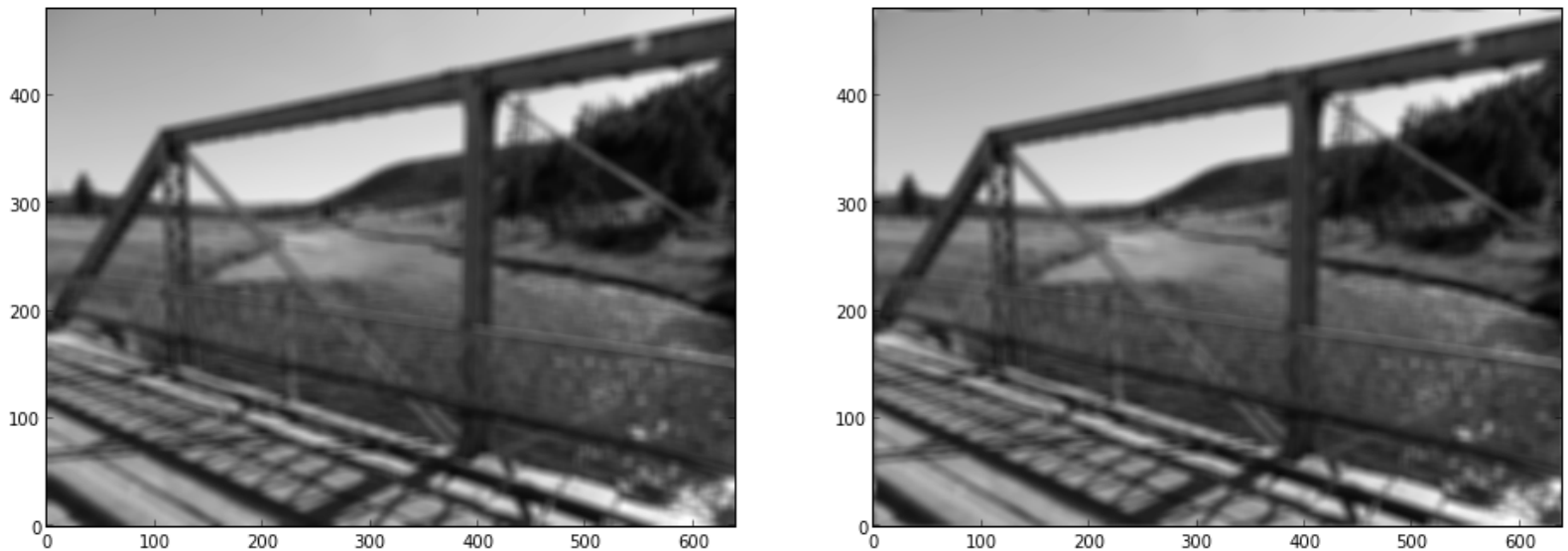
This turns out not to be a very efficient way of implementing these kinds of filters in array languages, because there are lots of loop steps and local operations.

It is much more efficient to loop over the r values and leave the addition of the big arrays to fast array operations within Python.

data parallel implementation of box filter

```
1 def myaverage(image,r):
2     out = zeros(image.shape)
3     for i in range(-r/2,r/2):
4         for j in range(-r/2,r/2):
5             out += roll(roll(image,i,axis=0),j,axis=1)
6     return out*1.0/(r*r)
```

```
1 imrow(filters.uniform_filter(image,10),myaverage(image,10))
```



We can also implement general convolutions with masks by computing a weighted average/sum.

(Think about: which boundary conditions does this implement?)

data parallel implementation of convolution

```
1 def myconvolve(image,mask):
2     mw,mh = mask.shape
3     out = zeros(image.shape)
4     for i in range(mw):
5         for j in range(mh):
6             out += mask[i,j]*roll(roll(image,mw/2-i,axis=0),mh/2-j,
                  axis=1)
7     return out
```

note the differences in boundary conditions

```
1 imrow(filters.convolve(image,mask),myconvolve(image,mask))
```