Computational Vision
U. Minn. Psy 5036

Linear systems, Convolutions and Optical blur

Linear Systems

Linear system models are important to vision for modeling: optical, retinal sampling, and neural transformations of image data. The basic requirements for a system **T** to be linear are:

$$T(a+b) = T(a) + T(b)$$
$$T(\lambda a) = \lambda T(a)$$

For continuous models, the so-called *linear superposition operator* for an image transformation can be expressed as:

$$r(x,y) = \iint I(x',y')W(x,y;x',y')dx'\,dy'$$

Exercises

1. As an pencil and paper exercise, show that the above 2D linear superposition operator satisfies the above two criteria.

2. Use *Mathematica* to verify that matrix multiplication on vectors satisfies the two criteria for a linear sytem: superposition and homogeneity. The matrix **WW** is the transformation applied to vector inputs **aa** or **bb**.

```
aa = Table[a[i],{i,1,3}];
bb = Table[b[i],{i,1,3}];
WW = Table[W[i,j],{i,1,3},{j,1,3}];
```

Answer to 2--or at least one way of doing it

```
Simplify[WW.aa + WW.bb - WW.(aa + bb)]
Simplify[WW.(k aa) - k WW.aa]
```

{0, 0, 0}

{0, 0, 0}

Convolutions

Often we can assume that the transformation is space invariant. This is a reasonable assumption if we restrict ourselves to small optical patches (so-called isoplanatic patches), uniform retinal sampling, or subgroups of neural systems whose receptive fields subtend small regions of the visual field.

For a space invariant system, W(x,y; x',y') is equal to W(x-x',y-y') and the linear operation becomes a *continuous convolution*: from -infinity to +infinity :

$$r(x,y) = \iint I(x',y')W(x-x',y-y')dx'\,dy'$$

For computations, we usually model the system operation as a *discrete matrix* multiplication. Let **W** be the matrix, **r** and **l** are vectors:

r = WI

2D convolution is used to model:

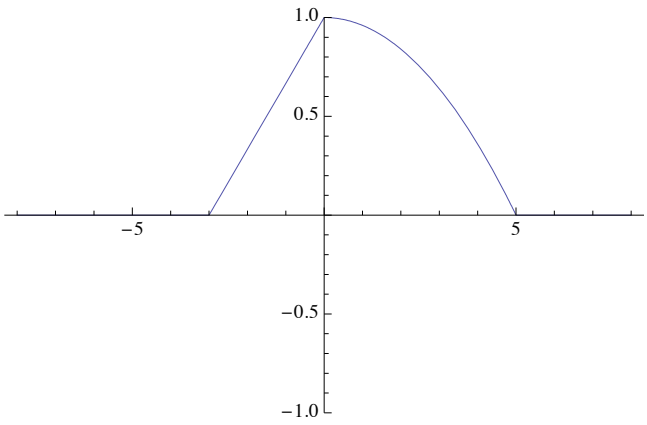
1) optical blur,
2) discrete retinal sampling by the photoreceptors, and
3) "neural image" transformations by ganglion cell arrays, and cortical cells.

Convolution is also used in computer vision as a pre-process for edge-detection, in models of color constancy, and motion detection. So it is worth spending time to understand it.

■ Continuous 1-D convolution

```
a = 3; b = 5;

Clear[filter];  (* When you play with building up definitions,  you should clear the function to make sure
filter[x_]:= N[(1-Abs[x]/a)] /; x>-a && x <0
filter[x_]:= N[(1-x^2/b^2)] /; x < b && x>=0
filter[x_]:= 0.0 /; x<=-a || x >= b
```

```
Plot[filter[x], {x, -8, 8}, PlotRange → {-1, 1}]
```



/; means "such that". Mathematica uses notation borrowed from the C programming language for logical operations such as:

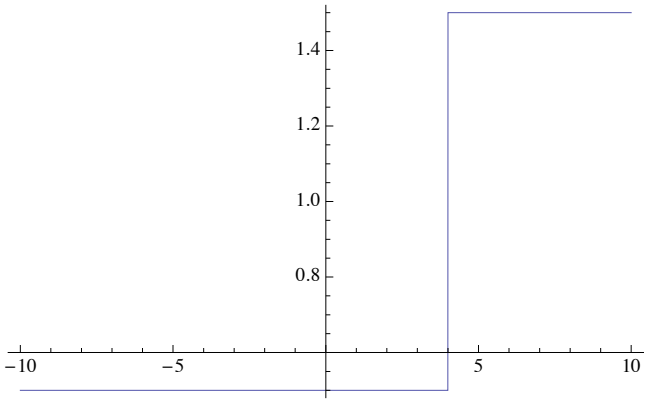
OR => **||**

and

AND => **&&**

```
edge[x_] := N[If[x<4,1/2,1.5]];
```

```
Plot[edge[x], {x, -10, 10}]
```



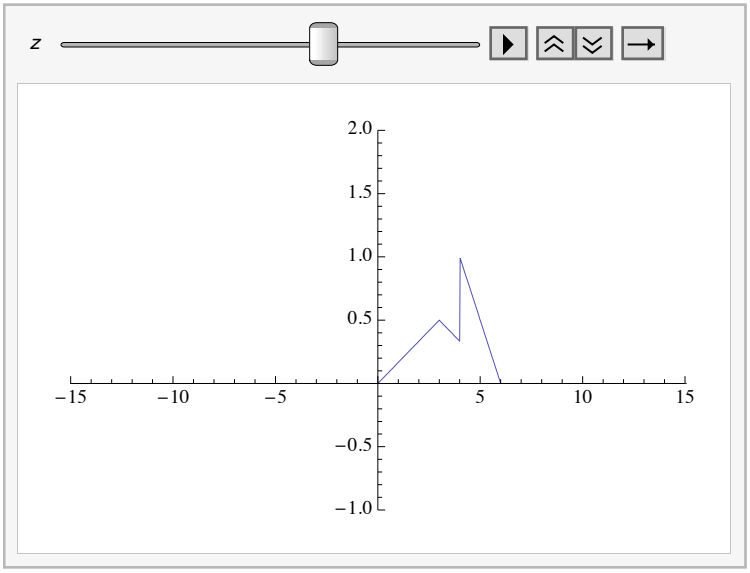
A 1-D convolution operation is written:

$$r(x) = \int_{-\infty}^{+\infty} l(x')w(x - x')dx'$$

$$r(x) = \int_{-\infty}^{+\infty} edge(z)\,filter\,(x - z)dz$$

It is useful to visualize this as the area underneath the curve formed by the product of edge and the left-right reversed filter as it slides along the x-axis.

```
Animate[
 Plot[edge[x] filter[z - x], {x, -30, 30}, PlotRange → {{-15, 15}, {-1, 2}}], {z, -10, 10, 0.5}]
```



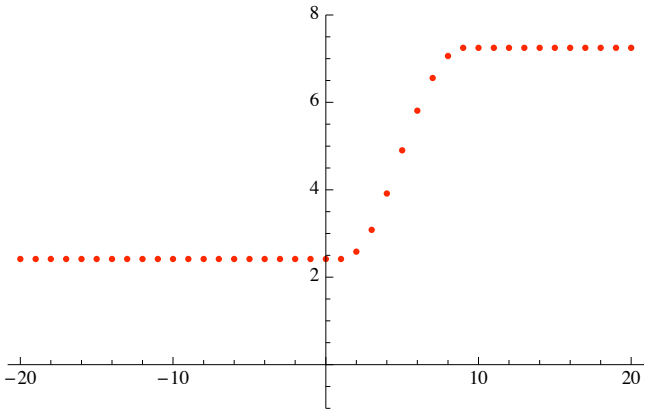
We can find the area under each of the above curves by using *Mathematica*'s built-in numerical integration capability, **NIntegrate[]**. It helps if we specify the range of necessary integration as precisely as possible. Inspection of the above curves shows that **{z-b,z+a}** is an appropriate range for **x**.

```
r1 =
Table[{z,NIntegrate[edge[x] filter[z - x],
                {x,z-b,z+a}]}, {z,-20,20,1}];
```

NIntegrate::slwcon :
Numerical integration converging too slowly; suspect one of the following: singularity, value of the integration is 0, highly oscillatory integrand, or WorkingPrecision too small. ≫

NIntegrate::slwcon :
Numerical integration converging too slowly; suspect one of the following: singularity, value of the integration is 0, highly oscillatory integrand, or WorkingPrecision too small. ≫

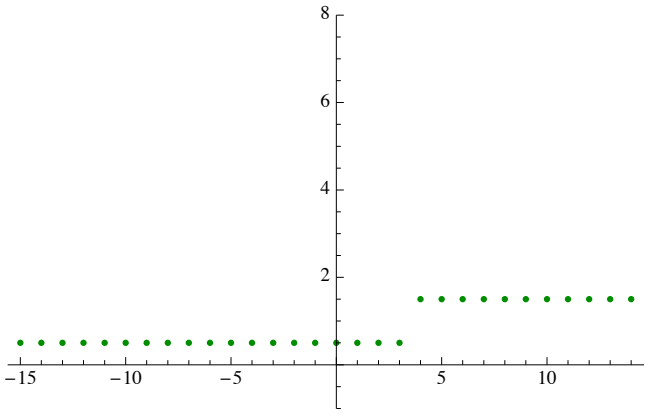
```
continousconvg = ListPlot[r1, PlotRange → {-1, 8}, PlotStyle → {RGBColor[1, 0, 0]}]
```



■ Discrete 1-D convolution

In this section, we will pass the 1-D "edge" through the filter again, but this time we will set it up as a discrete matrix operation. We'll apply a convolution matrix to a vector that represents a sharp edge, **l=edge**, at x = size/2.

```
size = 30; edge[x_] := N[If[x < 4, 1/2, 1.5`]];
edgelist = Table[{x, edge[x]}, {x, -size/2, size/2 - 1}]; edgevector = Transpose[edgelist][[2]];
edgelistg = ListPlot[edgelist, PlotRange → {-1, 8}, PlotStyle → {RGBColor[0, 0.5`, 0]}]
```



Because the integration is finite, we have to make some decision about what to do at the boundaries. One choice is to assume that

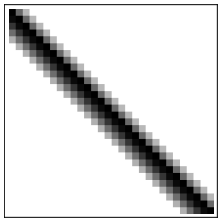
w and l are zero beyond the boundaries. Another is to assume that they are periodic with a shared common period, and we are just representing one period of the signal.

Now we construct a **size x size** matrix that will perform a convolution operation.

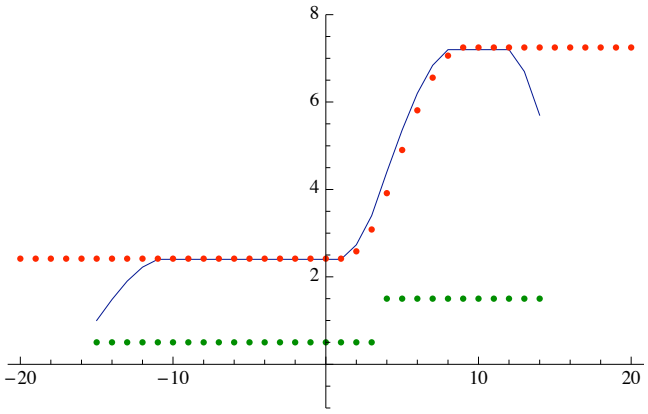
```
w   = Table[filter[i-j], {i,size},{j,size}];
```

A convenient way to view the matrix is to use **ArrayPlot[]**, where you can see how each row is a shifted version of the preceding row.

```
ArrayPlot[w, ImageSize → Tiny]
```



```
rvect = w.edgevector; r = Table[{i - size/2, rvect[[i + 1]]}, {i, 0, size - 1}]; matrixconvolg =
 ListPlot[r, Joined → True, PlotRange → {-1, 8}, PlotStyle → {RGBColor[0, 0, 0.5`]}];
Show[{matrixconvolg, edgelistg, continousconvg}]
```



The convolution matrix "blurs" out the edge and is in reasonable agreement with the approximation to the continous convolution. Note that the left and right boundary edges also got "blurred" by the matrix convolution.

■ Exercise: Blur the above edge with a line-spread function (LSF) that models diffraction-limited optics

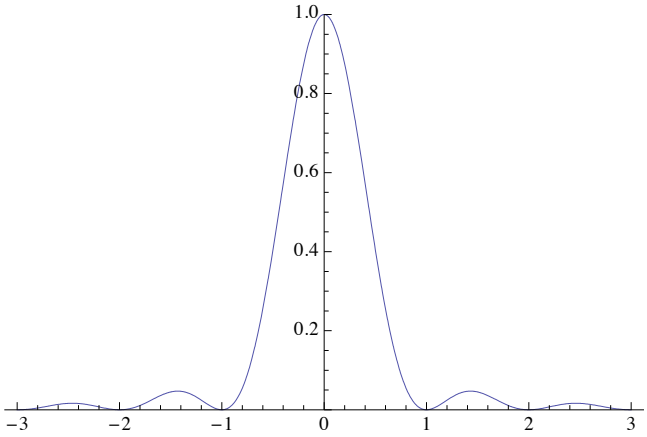
■ Introduction to diffraction limited blur

Pin-hole optics are limited by diffraction. The image of a point though a circular aperture is an Airy pattern. We have to take care to define the function at x and y =0.

```
(*Cell inactive*)
Airy2D[x_,y_] :=
If[x==0 && y==0,1,
(2 BesselJ[1,Pi Sqrt[x^2+y^2]]/(Pi Sqrt[x^2+y^2]))^2]
Plot[Airy2D[x,0],{x,-3,3}];
```

For a single slit aperture, the "line-spread-function" for a diffraction limited system is a "sync" function squared:

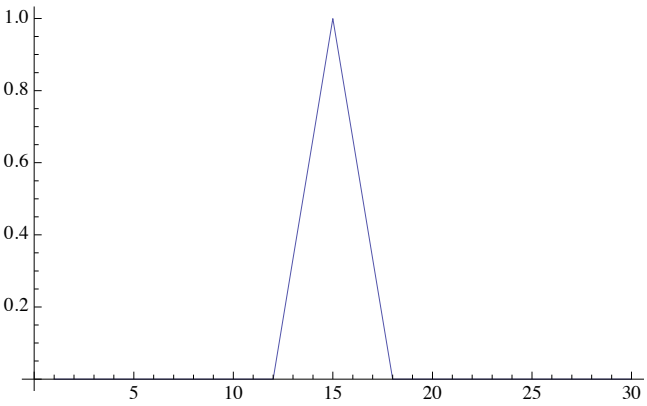
```
SincSq[x_]:= If[x==0,1.0,N[(Sin[Pi x]/(Pi x))^2]];

Plot[SincSq[x], {x, -3, 3}, PlotRange → {0, 1}]
```



- **Now as an exercise, construct a weight or filter matrix with the SincSq[] function, do a discrete convolution with the above edge, and plot up the results.**

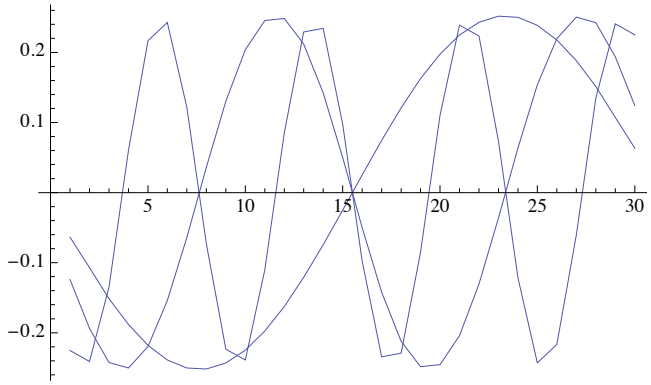
Convolutions and eigenvectors

```
size=30;
a = 3; b = 3;

Clear[filter];  (* When you play with building up definitions,  you should clear the function to make sure
filter[x_]:= N[(1-Abs[x]/a)] /; x>-a && x <0
filter[x_]:= N[(1-Abs[x]/a)] /; x < b && x>=0
filter[x_]:= 0.0 /; x<=-a || x >= b


ww  = Table[filter[N[i-j]], {i,size},{j,size}];
ee = Eigenvectors[ww];

ListPlot[ww[[15]], Joined → True]
```



Let's plot a few of the eigenvectors of w:

```
g1 = ListPlot[ee[[2]], Joined → True];
g2 = ListPlot[ee[[4]], Joined → True]; g3 = ListPlot[ee[[8]], Joined → True]; Show[{g1, g2, g3}]
```

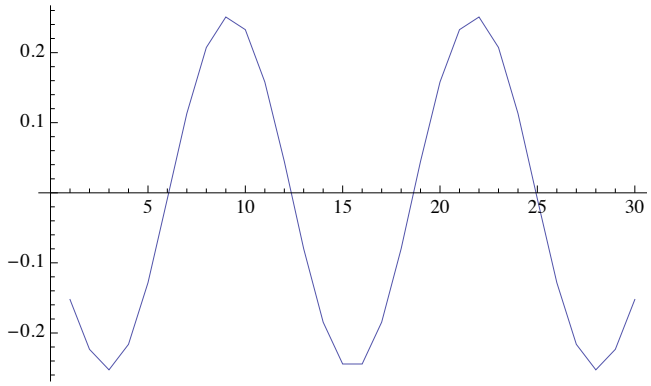


They sure look a lot like sinewaves, and in fact, they are pretty close. This is related to the fact that the sine-wave gratings keep the same form when imaged by the optics--even in the presence of aberrations. Why? Because the linear, shift-invariant model is a reasonable model for an optical system.

■ Exercise

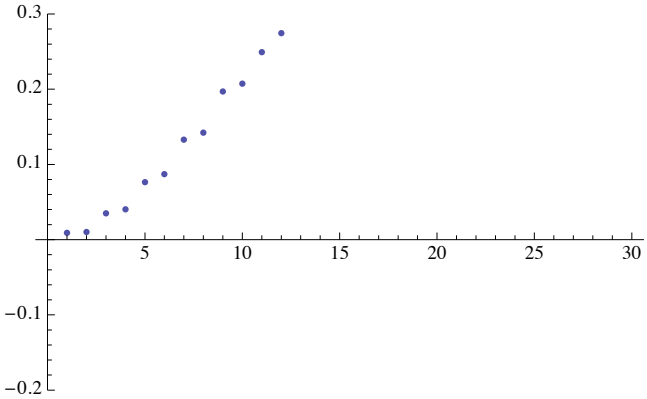
Verify that **ee[[5]]** is indeed an eigenvector of **ww** by showing that ww.ee[[5]] points in the same direction as e[[5]].

```
ListPlot[ee[[5]], Joined → True]
```



Here is a plot of the sorted eigenvalues for **ww**:

```
lambda = Sort[Eigenvalues[ww]]; ListPlot[N[lambda], PlotRange → {-0.2`, 0.3`}]
```



Later on we will use eigenvectors and eigenvalues in a completely different context when we discuss efficient ways of representing image information.

■ 2D Convolution computation and the FFT (Fast Fourier Transform)

In this notebook, we've taken a preliminary look at linear filtering with simple examples in 1-D. Later, we will extend our techniques to efficiently handle 2-D images. Convolution can either be done in the space domain (as above), or in the Fourier domain. If we multiply the fourier transform of the image with the fourier transform of the filter (e.g. point spread function), and then take the inverse fourier transform of this product, we have the convolution of the image with the filter. Why bother? The main reason is that there are fast digital techniques, e.g. the Fast Fourier Transform or FFT, which make doing convolutions by multiplying in the fourier domain much more efficient.

Later, we will have a notebook that illustrates blurring and general filtering of 2-D images using the FFT and inverse FFT, which are a built-in operations in *Mathematica*.

Contrast sensitivity function (CSF)

This demonstration will let you see: 1) the effect of convolution of your own optics on high spatial frequency gratings; 2) the effect of your own neural processing on low-spatial frequencies. You will see that your visual system behaves as a band-pass filter of spatial frequencies. This combintion of optical and neural processing can be modeled as a convolution, but with a filter that has a center-surround organization.

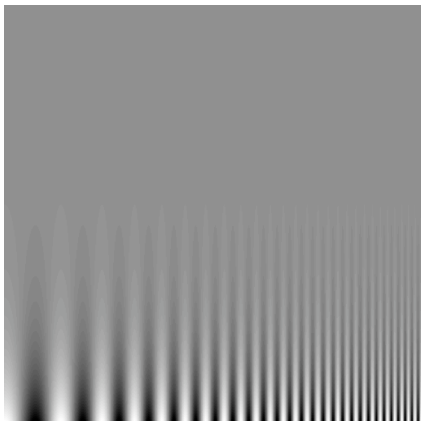
In the 1960's John Robson and Fergus Campbell at the Physiological Laboratories in Cambridge, England devised a clever way of enabling people to "see" their own constrast sensitivity functions. The idea was to generate a pattern whose spatial frequency increased exponentially to the right, and whose contrast decreased exponentially vertically.

For the middle to high spatial frequency range (right half of the figure), a normal observer sees a drop off in contrast sensitivity with increasing spatial frequency. Much of this drop-off can be accounted for by the modulation transfer function of the optics (MTF). For low spatial frequencies, there is also a fall-off in sensitivity--the optical transfer function can NOT account for this. To understand this low spatial frequency fall-off, we need to take a look at how light information gets transformed by the neural circuitry of the eye itself by means of lateral inhibition.

The *Mathematica* Edit:Preferences:Graphics option should be set to render the Postscript for the best monitor. Then set the monitor gray-levels with 256 levels by going to the , and then to Control Panels: Monitors.

Robson and Campbell made this pattern using analog circuitry. We'll use the computer. But be careful! It takes a REALLY long time to generate. I made the cell below "inactive", so that you have to think twice before executing the production of a large picture.

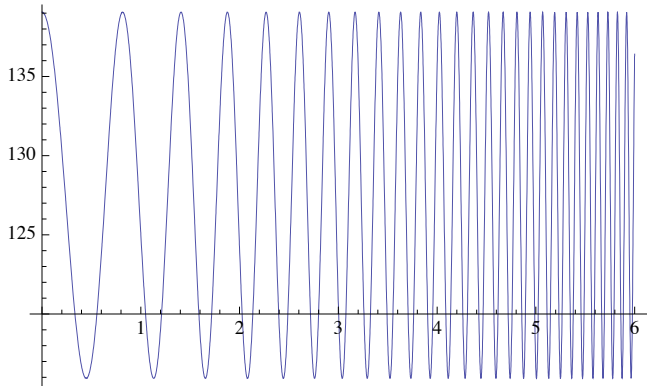
```
CSF[x_, y_] := N[127.5` e^(-y/0.125) Cos[2 π e^(x/4) x] + 127.5`];

DensityPlot[CSF[x, y], {x, 0, 6}, {y, 0, 1}, PlotPoints → {512, 1024}, PlotRange → {1, 254},
 Mesh → False, Axes → False, Frame → False, ColorFunction → "GrayTones"]
```



You may have to squint a little when you look at this pattern to reduce the effect of coarse gray-level quantization near the top. What you should see is a upside-down "U" shaped boundary--above which you can't see the grating pattern, and below which you do see it.

You are experiencing your own contrast senstivity function. A critical point to realize is that the *objective* contrast across a horizontal line is fixed, but the *subjective* contrast near threshold varies.

```
CSF[x_, y_] := N[127.5 e^(-y/0.125) Cos[2 π e^(x/4) x] + 127.5`]; Plot[CSF[x, 0.3], {x, 0, 6}, PlotPoints → 100]
```



In fact, if you move back and forth in *depth*, the peak of the upside-down U should move left or right. This is because the peak is between 3 and 5 cycles/degree for a normally sighted person. By changing your viewing distance, you can change the place on the screeen corresponding to 3-5 cycles/degree.

Caveat: To see the true shape of your CSF as **Log contrast senstivity vs. Log spatial frequency** you would have to allow for two factors. 1) The computer screen does not have a uniform effect on contrast as a function of spatial frequency. In fact, a CRT has its own MTF. This becomes particularly important when we sample near the pixel rate. As long as you are not too close to the right end of the figure, the contrast loss due to the screen is negligible. 2) The luminance of the screen is not linearly related to the gray-level. This correction can be made by "gamma" correction. The log of CRT intensity is proportional to gamma times the log of the voltage input, where a typical value of gamma is 2. This correction is made by loading the computer display board's look-up-table (LUT) with the inverse of this power function.