Computational Vision
U. Minn. Psy 5036W

Assignment #1: Ideal pattern detector

In this assignment, you will simulate an ideal observer called the signal-known-exactly ideal (SKE). This observer has a built-in template that matches the signal that it is looking for. The signal is embedded in "white gaussian noise". "white" means the pixels are not correlated with each other--intuitively this means that you can't reliably predict what any one pixel's value is from any of the others.

In the absence of any internal noise, this ideal detector behaves as one would expect a linear neuron to behave when a target signal pattern exactly matches its synaptic weight pattern.

There are some neurons in the the primary cortex of the visual system called "simple cells". These cells can be modeled as ideal detectors for the patterns that match their receptive fields. Given the right orientation and size, an image stimulus consisting of an sinusoidal grating "patch" (a "gabor patch") can be a good match for a given simple cell. In actual practice, simple cells, and neurons in general are not noise-free, and are not linear.

In a *Science* paper published in 1981, Burgess and colleagues showed that human observers were almost as good as the ideal pattern discriminator for gabor patches of the right spatial frequency and size (See Burgess, A. E., Wagner, R. F., Jennings, R. J., & Barlow, H. B. (1981). Efficiency of human visual signal discrimination. *Science*, 214 (4516), 93-94.)

■ Before you get started

When doing calculations on images, it is useful to be able to easily go back and forth between matrix and vector representations of images. So in the exercises below, you have a **signal** which is represented as a **size x size** matrix. But in some of the calculations you'd like to treat the signal as a vector. How do you get back and forth? The answer is with **Flatten[]** and **Partition[]**. Here is an example:

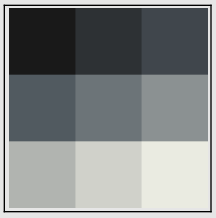
```
m = {{1,2,3},{4,5,6},{7,8,9}}
v=Flatten[m]
Partition[v,3]
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

{1, 2, 3, 4, 5, 6, 7, 8, 9}

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

```
ArrayPlot[m, Mesh → False, ColorFunction → "GrayTones"]
```



The ColorFunction → "GrayTones" option reverses **ArrayPlot[]**'s default where zero values are shown in white, and maximum positive or negative values in black.

■ The target

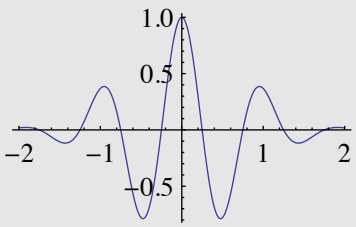
The target is given to you below. It is a low contrast sine-wave grating "patch" oriented at 45 deg and vignetted by a gaussian window. This patch is an example of what is called a *2D gabor function*.

The x-y dimensions are **size x size**. We will define both the target signal and the noise to have zero means so that when we add the noise as contrast noise, the mean doesn't change.

```
Grating[x_,y_,fx_,fy_] := Cos[2.0 Pi (fx x + fy y)];
GratingPatch[x_,y_,fx_,fy_] := Exp[-(fx x)^2 - (fx y)^2]*Grating[x,y,fx,fy]
```

We'll plot a cross-section to see what it should look like:

```
Plot[GratingPatch[x, 0, 1, 1], {x, -2, 2}]
```

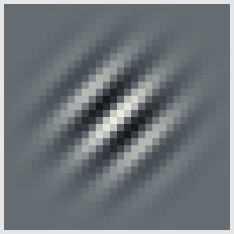


OK, now let's make the 2D target image.

```
size = 32; amplitude = 0.0625;
fx= 1.0; fy = 1.0;
x1=-2; x2=2;
signal = Table[amplitude*GratingPatch[x,y,fx,fy],
{x,x1,x2,(x2-x1)/(size-1)},{y,x1,x2,(x2-x1)/(size-1)}];
```

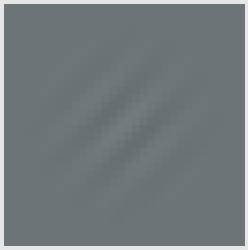
And it looks like this at full contrast:

```
ArrayPlot[signal, Mesh → False, Frame → False, ColorFunction → "GrayTones"]
```



...and like this at 6.25% contrast:

```
ArrayPlot[signal, Mesh → False, Frame → False, ColorFunction → "GrayTones",
 PlotRange → {-1, 1}]
```



(Note: On an 8-bit display, graylevels go in integer steps from 0 to 255. We use the option PlotRange->{-1,1} to map our signal values {-1,1} to pixel intensity values L, {0,255}. Contrast of a grating is defined as $\delta L/L_0$, where L_0 is the mean intensity, and δL is the amplitude of the grating. Zero maps to graylevel L_0= 128 (rounding up). A signal amplitude of 0.0625 maps to 0.0626*128 = 8. So the contrast, $\delta L/L_0$= 8/128 ~ 0.0626*128/128 = 0.0625. If you were actually doing psychophysics, it would be important to check to make sure that the computer monitor can actually supply the graylevel differences you want. E.g.if the amplitude gets below 1/128, an 8-bit display card won't be able to deliver--and human contrast thresholds can be lower than this. Another note, is that the monitor's intensity isn't necessarily linearly related to the requested graylevel. This problem can be fixed by doing "gamma correction", which involves setting the lookup table in your computer's graphics card to compensate for the particular display you may be using.)

■ Problem 1: The noise

Define a function **noise** that will generate a **size x size** array of Gaussian noise with standard deviation of **sigma= 0.25**, and a **mean** of **0.** (Use the built-in function, NormalDistribution[]). Make sure that you define it in such a way that each time you call **noise**, it gives you a new random image.

Add the **signa**l to the **nois**e to generate an observation vector called **observatio**n, and use **ArrayPlot[]** (with the options used above) to display a sample observation.

■ Problem 2-a: Ideal's theoretical d'

Use the formula:

dprime := Sqrt[**s.s**]/sigma

to calculate the ideal's d' using *Mathematica*.

■ **Problem 2-b: Write a function, dprime2[], to calculate d' in a two-alternative forced-choice experiment given the z-score of the proportion correct.**

■ **Answer 2-b.** I'll give you the answer to this one. Here is the answer using the inverse of a standard mathematical function called **Erf[].**

```
z[p_] := Sqrt[2] InverseErf[1 - 2 p];
```

d' for a 2AFC task is given by the formula:

dprime := -Sqrt[2] Z(Pc)

where Z(*) is the z-score for P_c , the proportion correct. (See the supplementary class notes on SDT.).

```
dprime2[x_] := N[-Sqrt[2] z[x]]
```

Sidenote: Psychophysicists sometimes design their experiments to look for the signal strength (e.g. contrast) that gives 76% correct (rather than, say 75 which is half way between 50 and 100%). The reason is that 76% is very close to 76.025%, for which dprime2 is exactly 1. Check it out.

```
dprime2[.76025]
```

```
1.
```

■ **Problem 3: Ideal's simulated d' in a two-alternative forced-choice task (2AFC)**

Generate **ntrials** = 100 observation pairs of **signal+noise**, and **noise**. For each pair compute an ideal decision variable (i.e. the output of a cross-correlator, which is the dot product of the observation with the signal). Pick which observation has the higher value of the decision variable. Decide whether this decision is the right answer or not. Total up the proportion of right answers and compute d' using **dprime2[]**. Repeat this four times (total of 400 trials) to get four estimates of d'. How does the average of these four compare to your result from the closed form prediction above?

■ **Problem 4: What would performance be like (measured in terms of d') be if the template was wrong? Run your simulation above using the template below:**

```
fx= 1.0; fy = 1.4;
x1=-2; x2=2;
wrongtemplate = Table[amplitude*GratingPatch[x,y,fx,fy],
{x,x1,x2,(x2-x1)/(size-1)},{y,x1,x2,(x2-x1)/(size-1)}];
```

```
ArrayPlot[wrongtemplate, Mesh → False, Frame → False,
 ColorFunction → "GrayTones"]
```