Computational Vision
U. Minn. Psy 5036
Daniel Kersten

2D Linear filtering and neural systems, FFTs

Initialization

In[1]:=
```
Off[SetDelayed::write]
Off[General::spell1]
```

Introduction

As we noted in an earlier notebook, spatial filtering by convolution can often be done much more quickly in the spatial frequency domain. Suppose we want to filter an image of a face. Let our image to be filtered be **face**. The idea is to generate a filter list, **filter**, and take its fourier transform, **filterft = Fourier[filter]**. Then we multiply the **filterft** by the fourier transform of face, call it **faceft.** Our filtered output is the inverse transform of the product:

InverseFourier[filterft faceft].

Although *Mathematica* can handle vectors of arbitrary dimension, it will use a special algorithm called the Fast Fourier Transform (FFT) if the image vector has dimensions which are powers of 2 (e.g. 8, 16, 32, 64, etc..).

Complex number notation allows us to handle phase in a very elegant way. So first a review.

Complex numbers

Complex numbers can be written in terms of real and imaginary parts:

```
z = 3 + 4 I;
Re[z]
Im[z]
```

3

4

A complex number can be thought of as a 2D vector whose x-value is the real part, and the y-value is the imaginary part. Alternatively, complex numbers can be written in polar notation where the complex number has a length r = **Abs[z]** and an angle θ = **Arg[z]** that it makes with the real axis.

z = r (cos θ + i sin θ)

```
Abs[z]
Arg[z]
```

5

$\text{ArcTan}\left[\frac{4}{3}\right]$

Using Euler's formula,
$e^{i\theta} = \cos\theta + i \sin\theta$,
we can put the length and angles back together.

```
z = Abs[z] Exp[I Arg[z]]
```

$5\, e^{i\, \text{ArcTan}\left[\frac{4}{3}\right]}$

```
N[%]
```

3. + 4. i

Fourier decomposition of image of a face

The input image -- face

In[13]:=
```
face = ImageData[ ];
```


Alternative input image -- substitute a cup for the face image

In[15]:=
```
face = ImageData[ ];
```


In[16]:=
```
size = Dimensions[face][[1]];
hsize = size/2
```

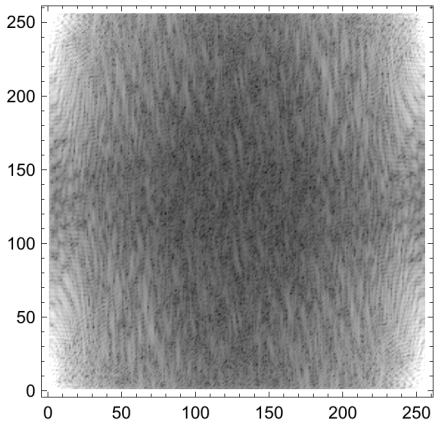
Out[17]= 128

Find the amplitude spectrum (spectrum) and the phase spectrum (phase) for the face picture. Chop[] sets small values to zero. Note that most of the energy is near zero.

In[6]:=
```
faceft = Fourier[face];
facespectrum = Chop[Abs[faceft]];
facephase = Chop[Arg[faceft]];
```

Fourier::fftl: Argument face is not a non-empty list or rectangular array of numeric quantities. >>

```
ListDensityPlot[facespectrum^.1, Mesh → False, ColorFunction → GrayLevel]
```



Put amplitude spectrum and phase back together

```
facerestored =
Chop[InverseFourier[facespectrum Exp[I facephase]]];

Image[facerestored]
```

You can play with changes to the amplitude or phase spectrum to see what happens. For example, what happens if you multiply the amplitude spectrum by an array whose values are proportional to the negative of the sum of the squares of the horizontal and vertical spatial frequencies? (Answer: this is the spatial frequency equivalent to applying a Laplacian operator in the space domain).

Some assorted functions and filters

Shift operator - a useful function for aligning filter representations

In[9]:=
```
shift[mat_,size_] :=
    Transpose[RotateRight[Transpose[RotateRight[mat,size]],size]]

squash[x_] := N[1/(1 + Exp[-x])];
```

Below we define several different kinds of filters that you can play with. But BEWARE, some of them are

very slow to calculate.

Diffraction limited round pupil filter - this one is very slow to fill a table.

```
Airy2D[x_,y_] :=
If[x==0 && y==0,1.0,
N[(2 BesselJ[1,Pi Sqrt[x^2+y^2]]/(Pi Sqrt[x^2+y^2]))^2]]
```

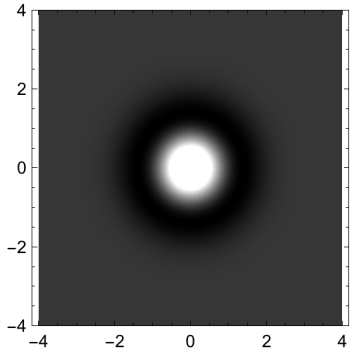
Low-pass Gaussian filter

```
gauss[x_,y_,sig_] :=
    N[Exp[(-x^2 - y^2)/(2 sig*sig)]];
filter = Table[gauss[i/hsize,j/hsize,2/hsize],
    {i,-hsize,hsize-1},{j,-hsize,hsize-1}];
```

$\nabla^2 G$

These filters are theoretically circularly symmetric. It is difficult to generate a circularly symmetric filter on a rectangular lattice. The techniques below are intended primarily to illustrate some symbolic computation features of *Mathematica*. One can define a DelSquaredG operator in terms of the second derivative of a Gaussian:

```
blur[x_, y_] := e^(-x^2-y^2);
dx[x_, y_] := ∂u blur[u, v] /.u → x /.v → y
dy[x_, y_] := ∂v blur[u, v] /.u → x /.v → y
delsqG[x_, y_] := ∂{v,2} blur[u, v] + ∂{u,2} blur[u, v] /.u → x /.v → y
DensityPlot[-delsqG[x, y], {x, -4, 4}, {y, -4, 4}, Mesh → False,
 PlotPoints → 64, PlotRange → {-2, 2}, ColorFunction → GrayLevel]
```



Band-pass oriented Gabor filters

Gabor filters are sine (or cosine) waves enveloped by a Gaussian. We define a cosine-phase filter, **cgabor[]**, which is even-symmetric about the origin:

```
cgabor[x_,y_, fx_, fy_,sig_] :=
N[Exp[(-x^2 - y^2)/(2 sig*sig)] Cos[
    2 Pi (fx x + fy y)]];
```

And a sine-phase filter, **sgabor[]** which is odd-symmetric:

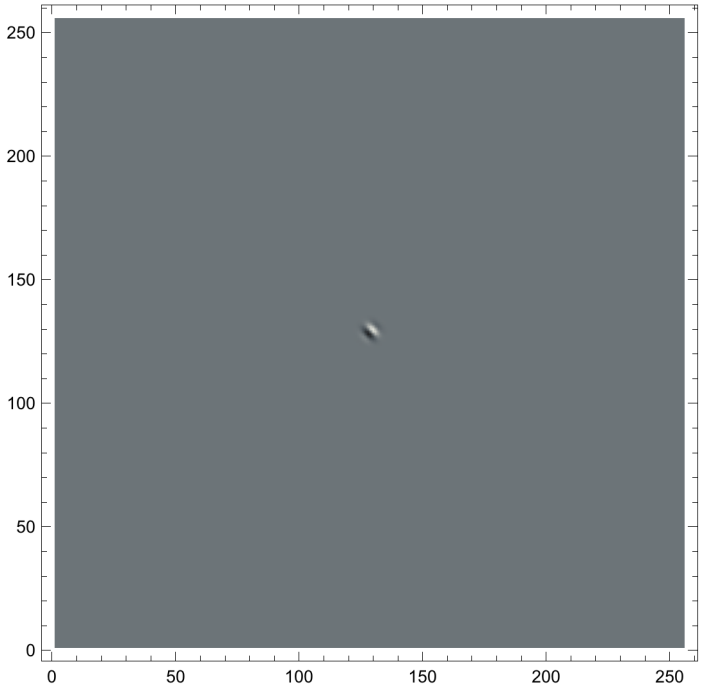
In[10]:=
```
sgabor[x_,y_, fx_, fy_,sig_] :=
    N[Exp[(-x^2 - y^2)/(2 sig*sig)] Sin[
        2 Pi (fx x + fy y)]];
filter = Table[sgabor[i/32,j/32,4,4,1/16],
    {i,-hsize,hsize-1},{j,-hsize,hsize-1}];
```

Part::partw: Part 1 of {} does not exist >>

Table::iterb: Iterator $\left\{i, -\frac{1}{2}\{\}[[1]], -1+\frac{\{\}[[1]]}{2}\right\}$ does not have appropriate bounds >>

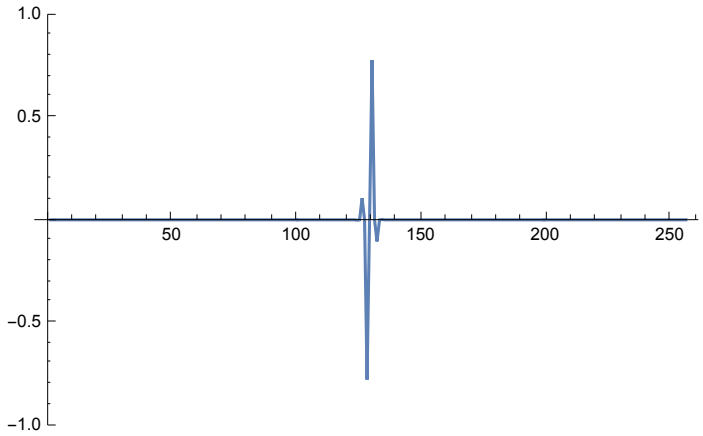
Table::iterb: Iterator $\left\{i, -\frac{1}{2}\{\}[[1]], -1+\frac{\{\}[[1]]}{2}\right\}$ does not have appropriate bounds >>

```
ListDensityPlot[filter, Mesh → False,
 PlotRange → {-1, 1}, ColorFunction → "GrayTones"]
```



The DensityPlot above shows a view of the "receptive field" of an oriented band-pass cell. Let's plot the filter along the diagonal:

```
ListPlot[Table[filter[[i, i]], {i, 1, size}], Joined → True, PlotRange → {-1, 1}]
```

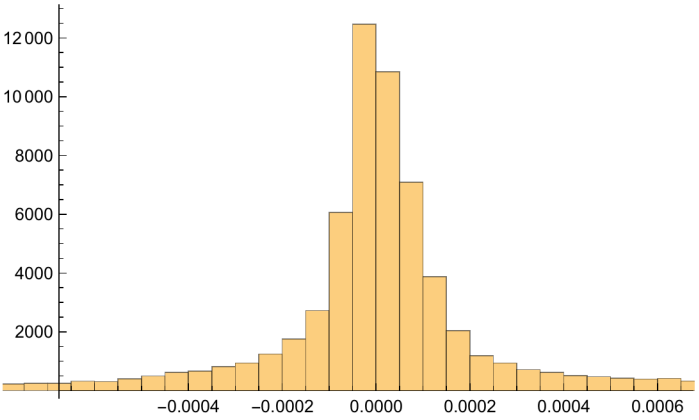


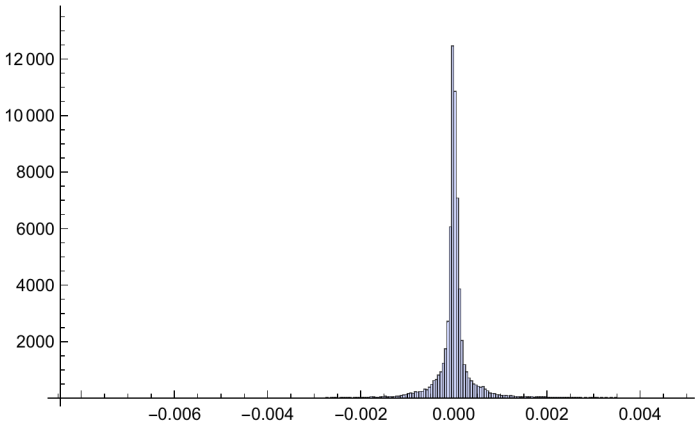
"Neural images": Apply an oriented filter to the face image

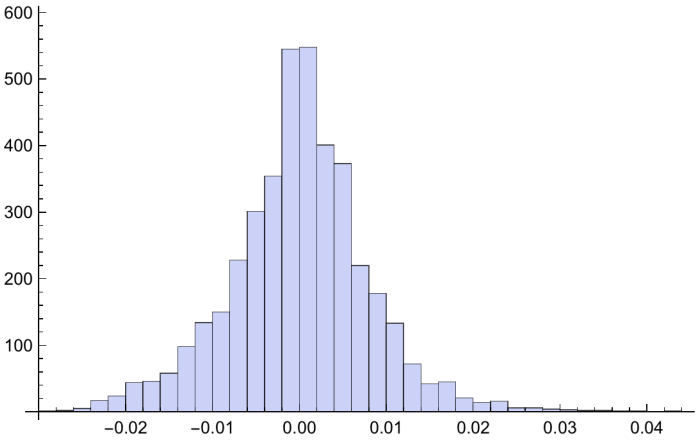
Below we show the results of applying a 45 degree oriented simple cell (a Hubel and Wiesel "edge detector") to the face image. This picture can be thought of as a "neural image" corresponding to the outputs of that sub-class of simple cells whose filtering properties can be modeled with this particular spatial frequency and orientation. The filter is generated using the sgabor filter above.

```
filterft = Fourier[shift[filter, hsize]];
ArrayPlot[fface = Chop[t = InverseFourier[filterft faceft]],
 Mesh → False, ColorFunction → "GrayTones", DataReversed → False]
```

```
Histogram[Flatten[fface]]
```








```
Kurtosis[Flatten[fface]]
```

```
29.1087
```

Exercises: Some general properties of the spectra of filters and images

Verify the following general properties with filter lists generated from cgabor and sgabor filters.

- Exercise 1: What is the imaginary part of the fourier transform of a real even symmetric (e.g. cosine-phase filter) image? What is the real part of an odd-symmetric image (e.g. sine-phase filter)?
- Exercise 2 - Similarity theorem: If the spatial scale of a filter or image is doubled (i.e. 2 x -> x), what happens to the scale of the filter's amplitude spectrum? What happens to the amplitude of the filter's amplitude spectrum?

  Verify this with the filter from cgabor. Below, F[] stands for fourier transform, I for an image, and L for the fourier transform of I.

$$F[I(ax, by)] = \frac{1}{ab} L\left(\frac{f_x}{a}, \frac{f_y}{b}\right)$$

- Exercise 3 - Parseval's theorem: What is the relationship between the length of an image, and the length of its transform?

- Exercise 4 - Shift Theorem

Verify that if you shift cgabor by 90 degrees to get sgabor, you can calculate the fourier transform of sgabor by multiplying the fourier transform of cgabor by an appropriate exponential. L() below is the fourier transform of cgabor.

$$F[l(x-a, y-b)] = L(f_x, f_y)\mathrm{e}^{-2\pi i(f_x a + f_y b)}$$

Importance of the phase spectrum in visual recognition

In the early days of pattern vision research, it was not often appreciated that the phase spectrum carries the significant information for recognition. This exercise demonstrates this by making an images with 1) random phase and face amplitude spectrum; 2) a random amplitude spectrum and face's phase spectrum.

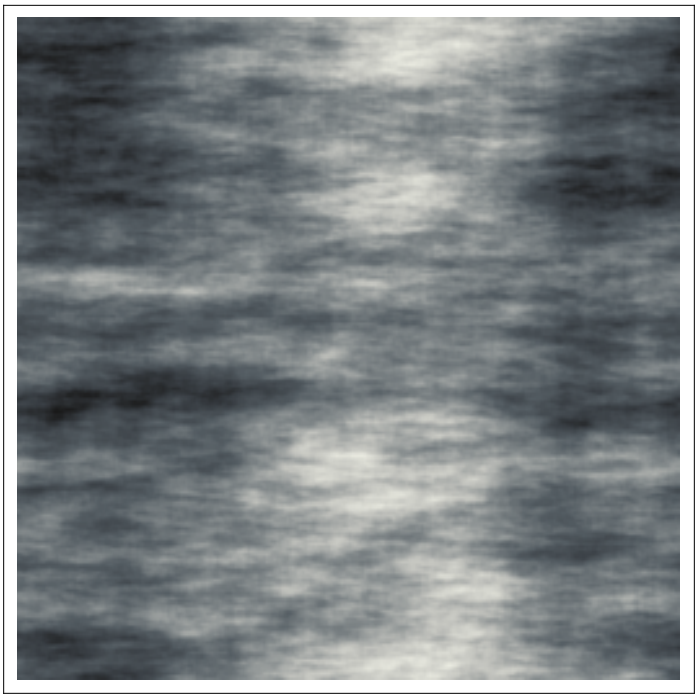
Make some uniformly distributed white noise. Then extract the amplitude spectrum and phase

```
ft = Table[N[π (2 RandomReal[] - 1)], {i, 1, size}, {j, 1, size}];
ft = Fourier[ft]; randomphase = Chop[Arg[ft]]; randomspectrum = Chop[Abs[ft]];
```

What does face "restored" face look like with his original amplitude spectrum but with random phases?

```
facewithrandomphase =
Chop[InverseFourier[facespectrum Exp[I randomphase]]];
```

```
ArrayPlot[facewithrandomphase, Mesh → False,
 ColorFunction → "GrayTones", DataReversed → True]
```



What does face look like if we substitute a noise spectrum (randomspectrum) for his amplitude spectrum?

```
facewithrandomspectrum =
Chop[InverseFourier[randomspectrum Exp[I facephase]]];
```

```
ArrayPlot[facewithrandomspectrum, Mesh → False,
 ColorFunction → "GrayTones", DataReversed → True]
```

- Exercise: What is the relative importance of the phase and amplitude spectrum for “Glass Patterns”. E.g. see Barlow, H. B., & Olshausen, B. A. (2004). Convergent evidence for the visual analysis of optic flow through anisotropic attenuation of high spatial frequencies. Journal of Vision, 4(6), 1–1. http://doi.org/10.1167/4.6.1