Image_convolution

January 14, 2016

In [1]:
```
# This is a cell to hide code snippets from displaying
# This must be at first cell!

from IPython.display import HTML, Image

hide_me = ''
HTML('''<script>
code_show=true;
function code_toggle() {
  if (code_show) {
    $('div.input').each(function(id) {
      el = $(this).find('.cm-variable:first');
      if (id == 0 || el.text() == 'hide_me') {
        $(this).hide();
      }
    });
    $('div.output_prompt').css('opacity', 0);
  } else {
    $('div.input').each(function(id) {
      $(this).show();
    });
    $('div.output_prompt').css('opacity', 1);
  }
  code_show = !code_show
}
$( document ).ready(code_toggle);
</script>
<form action="javascript:code_toggle()"><input style="opacity:0" type="submit" value="Click here
```

Out[1]:
```
<IPython.core.display.HTML at 0x7f945c47aa50>
```

1 Image convolution: Direct space versus Fourier space

In image processing, the convolution with a small matrix is used for blurring, sharpening, embossing, edge-detection, and more. This is accomplished by means of convolution between a kernel and an image. http://en.wikipedia.org/wiki/Kernel_%28image_processing%29

In this exercise we will focus on the gaussian blur and compare the execution time of two appoaches: direct space vs Fourier space for different width of gaussians: 1, 2, 4, 8, and 16pixels.

The raw image used is the famous 512x512 pixel image from Lena, and available as part of scipy.

Nota: Since begining of 2016, the image of Lena is planed for removal from scipy. One should use *scipy.misc.ascent()* instead of *scipy.misc.lena()*

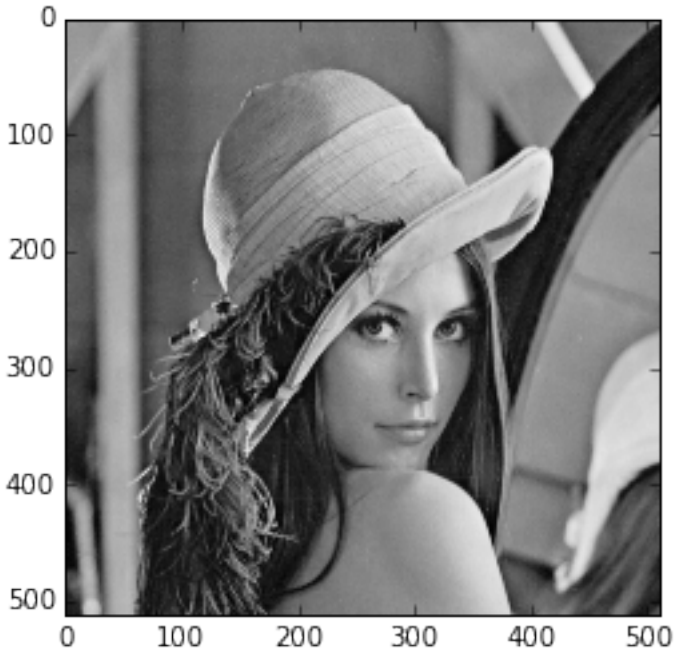
1.1 Loading the Scientific Python stack and the input image

```
In [2]: %pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
In [3]: from scipy.misc import lena
        img = lena()
        imshow(img, cmap="gray")
```

```
Out[3]: <matplotlib.image.AxesImage at 0x7f943f422910>
```



1.2 Properties of the gaussian for bluring an image

```
In [4]: hide_me
        from IPython.display import Image
        Image(filename="files/gaussian.png")
```

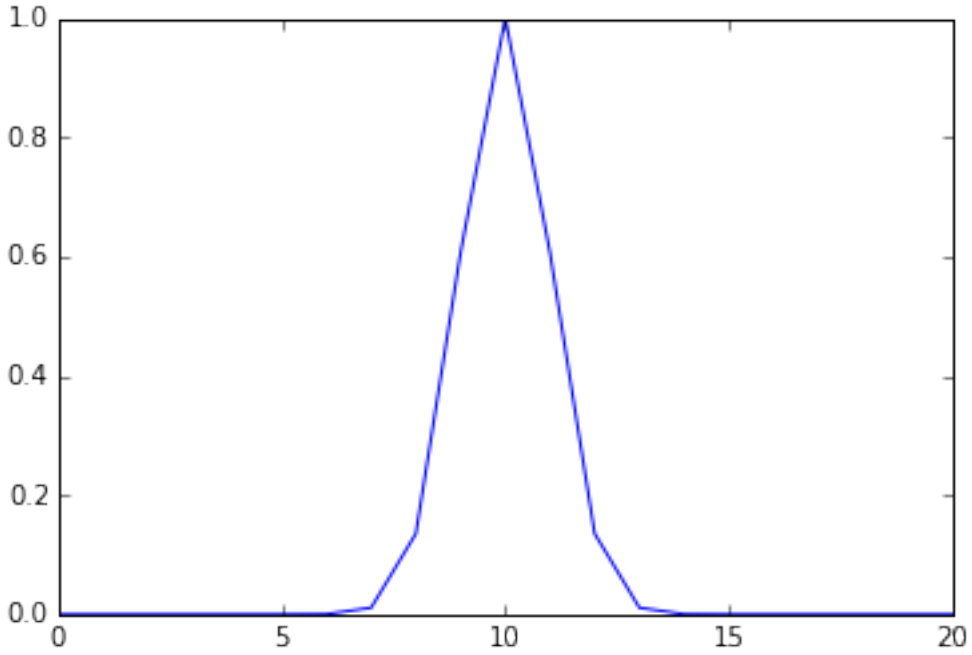
```
Out[4]:
```

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}.$$

The gaussian has some key advantages: * Very smooth bluring * Separable in x/y (hence much faster) * The Fourier transform is a gaussian of width proportional to 1/sigma

```
In [5]: from scipy.signal import gaussian
        plot(gaussian(21,1))
```

```
Out[5]: [<matplotlib.lines.Line2D at 0x7f94340c2a10>]
```



1.3 Direct space approach

Choice of the window size ?

One of the key parameter of a direct space convolution is the size of the window for the convolution. To calculate it, we take a gaussian with $\sigma = 1.0$ and evalute the number of integer position have non-neglectable values (>0.0001).

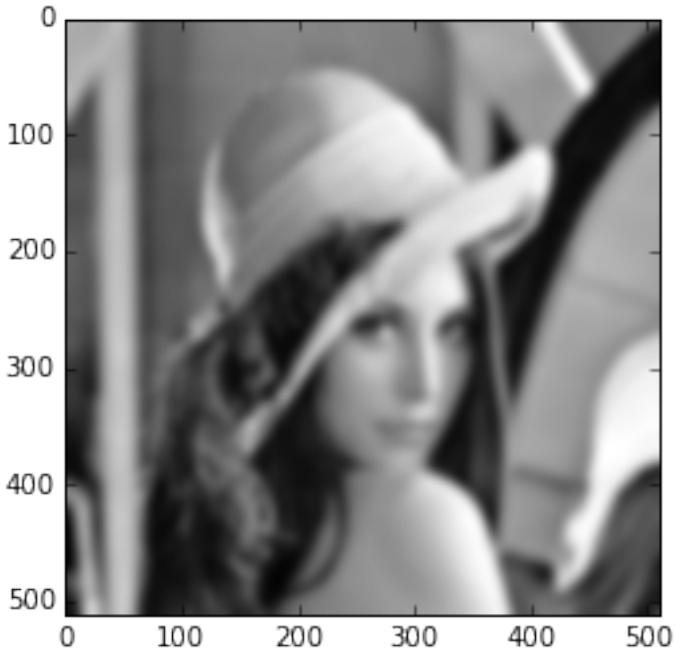
```
In [6]: (gaussian(21,1)>1e-4).sum()

Out[6]: 9
```

It is important to have an odd window size, we choose w = 8 sigma+1.

scipy.signal has a special function for direct space convolution of separable functions: *sepfir2d* https://docs.scipy.org/doc/scipy-0.7.x/reference/generated/scipy.signal.sepfir2d.html

Performing 2 times a 1D convolution is much faster than performing a single 2D convolution.

```
In [7]: from scipy.signal import sepfir2d
        def direct(img, sigma):
            gaus = gaussian(8 * sigma + 1, sigma)
            return sepfir2d(img, gaus, gaus)

In [8]: imshow(direct(img, 5), cmap="gray")

Out[8]: <matplotlib.image.AxesImage at 0x7f9434063d50>
```




```
In [9]: %timeit direct(img, 2)
```

```
100 loops, best of 3: 12.4 ms per loop
```

1.4 Fourier space convolution

The convolution in direct space is equivalent to a multiplication in Fourier space. http://en.wikipedia.org/wiki/Convolution_theorem

```
In [10]: hide_me
         from IPython.display import Image
         Image(filename="files/convolution.png")
```

Out[10]:

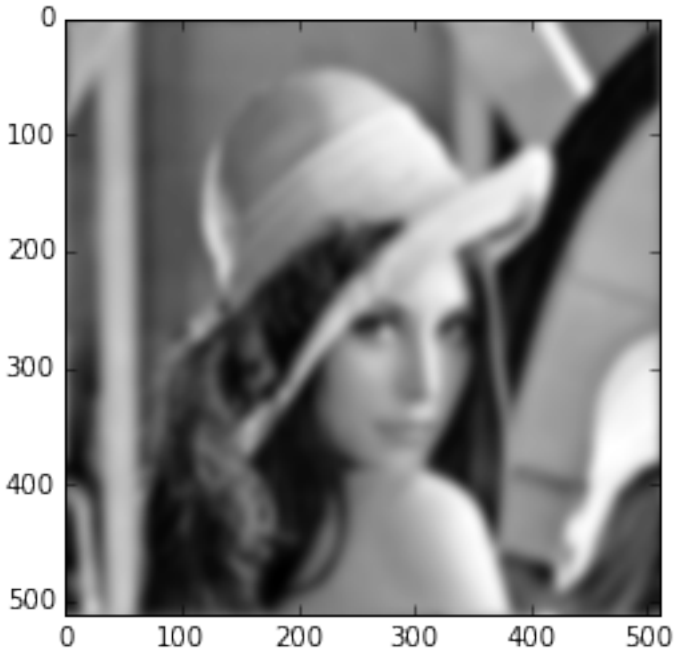
$$f * g = \mathcal{F}^{-1}\{\mathcal{F}\{f\} \cdot \mathcal{F}\{g\}\}$$

This is especially appealing for gaussian blurring as the FT of a gaussian is a gaussian. The total cost is reduced to one FT and its inverse.

```
In [11]: from scipy.fftpack import fft2, ifft2, fftshift
         def ft(img, sigma):
             size = len(img)
             gaus = gaussian(size, size / (2*pi*sigma))
             sgau = fftshift(gaus)
             return ifft2(fft2(img) * outer(sgau,sgau)).real
```

```
In [12]: imshow(ft(img, 5), cmap="gray")
```

Out[12]: <matplotlib.image.AxesImage at 0x7f942f8451d0>



In [13]: %timeit ft(img, 2)

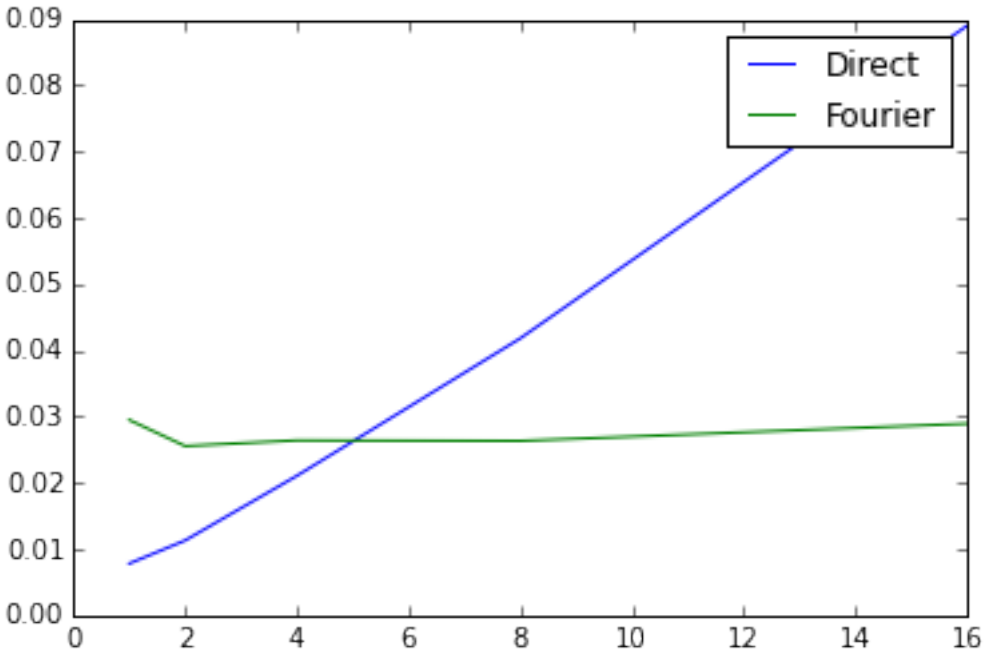
10 loops, best of 3: 26.8 ms per loop

1.5 Benchmark: Direct space vs Fourier space

In [14]:
```
import time
sigmas = []
d = []
f = []
for i in range(5):
    sigma = 1<<i
    t0 = time.time()
    direct(img, sigma)
    t1 = time.time()
    ft(img, sigma)
    t2 = time.time()

    print("sigma=%s \t Direct: %.3fs \t Fourier: %.3fs"%(sigma,t1-t0,t2-t1))
    sigmas.append(sigma)
    d.append(t1-t0)
    f.append(t2-t1)
plot(sigmas,d,label="Direct")
plot(sigmas,f,label="Fourier")
legend()
```

```
sigma=1          Direct: 0.008s          Fourier: 0.029s
sigma=2          Direct: 0.011s          Fourier: 0.026s
sigma=4          Direct: 0.021s          Fourier: 0.026s
sigma=8          Direct: 0.042s          Fourier: 0.026s
sigma=16          Direct: 0.089s          Fourier: 0.029s
```

```
Out[14]: <matplotlib.legend.Legend at 0x7f942f78b650>
```



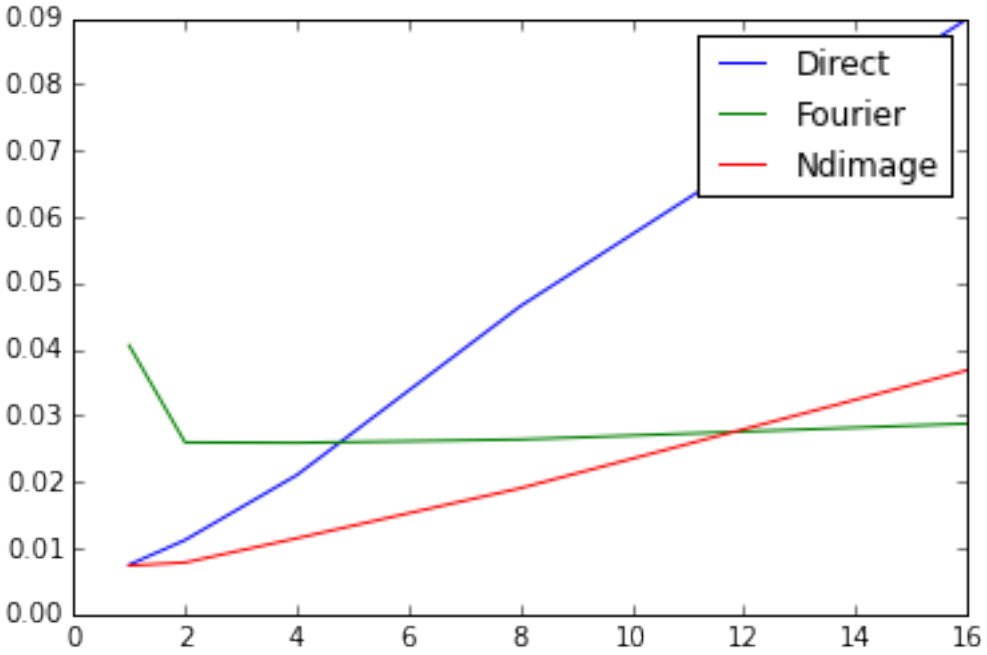
Nota: The gaussian blur filter exists also as "ready to use" function as part of scipy.ndimage.filters.gaussian_filter. We can compare how it behaves compared to our implementations:

```
In [15]: from scipy.ndimage import filters

In [16]: %timeit filters.gaussian_filter(img, 2)

100 loops, best of 3: 7.1 ms per loop

In [17]: %timeit filters.gaussian_filter(img, 4)

100 loops, best of 3: 10.5 ms per loop

In [18]: import time
         from scipy.ndimage import filters
         sigmas = []
         d = []
         f = []
         n = []
         for i in range(5):
             sigma = 1<<i
             t0 = time.time()
             direct(img, sigma)
```

```
            t1 = time.time()
            ft(img, sigma)
            t2 = time.time()
            filters.gaussian_filter(img, sigma)
            t3=time.time()
            print("sigma=%s \t Direct: %.3fs \t Fourier: %.3fs, \t ndimage: %.3fs"%(sigma,t1-t0,t2-t1,
            sigmas.append(sigma)
            d.append(t1-t0)
            f.append(t2-t1)
            n.append(t3-t2)
        plot(sigmas,d,label="Direct")
        plot(sigmas,f,label="Fourier")
        plot(sigmas,n,label="Ndimage")
        legend()
```

```
sigma=1          Direct: 0.007s          Fourier: 0.041s,          ndimage: 0.007s
sigma=2          Direct: 0.011s          Fourier: 0.026s,          ndimage: 0.008s
sigma=4          Direct: 0.021s          Fourier: 0.026s,          ndimage: 0.011s
sigma=8          Direct: 0.047s          Fourier: 0.026s,          ndimage: 0.019s
sigma=16          Direct: 0.090s          Fourier: 0.029s,          ndimage: 0.037s
```

```
Out[18]: <matplotlib.legend.Legend at 0x7f942f66f910>
```



2 Conclusion

The choice of the algorith is of prime importance for image manipulation: Fourier transform are extremely fast and optimized but they cannot compete with simpler algorithms when the window is small. Moreover, if your problem is common, there is probably an optimized implementation already available within *scipy*.