Beeldbewerken
Assignment 4 : "Local structure"

Joris Stork, Lucas Swartsenburg

November 5, 2011

1 Analytical local structure

1.1 Find derivatives of $f(x, y) = A\sin(Vx) + B\cos(Wy)$

$$\begin{aligned} f_x &= AV\cos(Vx) \\ f_y &= -BW\sin(Wy) \\ f_{xx} &= -AV^2\sin(Vx) \\ f_{yy} &= -BW^2\cos(Wy) \\ f_{xy} &= 0 \end{aligned} \tag{1}$$

1.2 Discretise $f(x, y) = A\sin(Vx) + B\cos(Wy)$

See assignment description for a plot of this function.

1.2.1 Code

```
def f(X,Y, A = 1, B = 2, V = (6 * np.pi / 201), W = (4 * np.pi / 201)):
    """ Discretisation of f. """

    F = A * np.sin(V * X) + B * np.cos(W * Y)
```

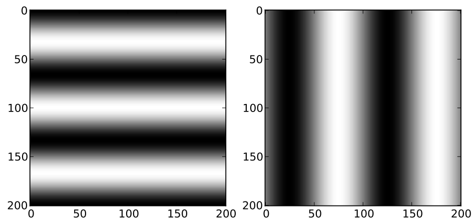

Figure 1: Fx and Fy

```
    return F
```

1.3 Generate images of Fx and Fy

See figure 1 for the corresponding plots.

1.3.1 Code

```
def fx(X,Y, A = 1, B = 2, V = (6 * np.pi / 201), W = (4 * np.pi / 201)):
    """  Discretisation of partial derivative of f wrt x. """

    F = A * V * np.cos(V * X)
    return F


def fy(X,Y, A = 1, B = 2, V = (6 * np.pi / 201), W = (4 * np.pi / 201)):
    """  Discretisation of partial derivative of f wrt y. """
```

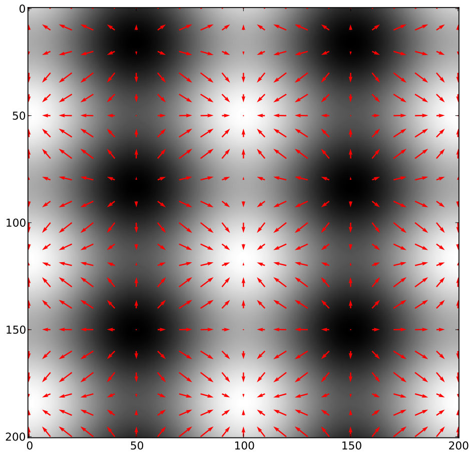

Figure 2: Gradient vectors for F

```
    F = -B * W * np.sin(W * Y)
    return F
```

1.4 Plot gradient vectors

2 Gaussian convolution

2.1 Implement gauss(s) function

See figure 3 for a plot of the two dimensional kernel obtained with `gauss()`

2.1.1 Code

```
def gauss(s):
    """ Gaussian kernel with scale s and dimensions s*6+1 by s*6+1  """
```

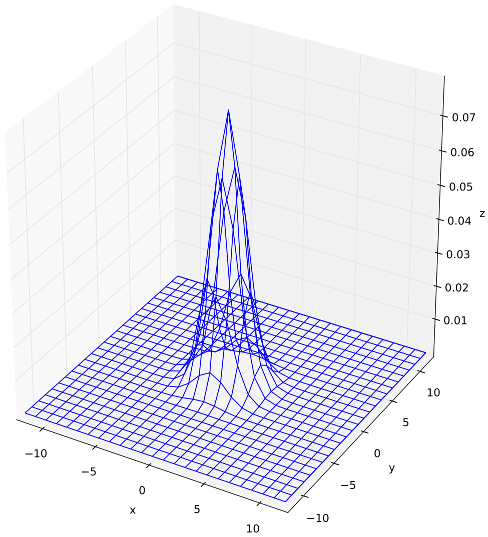

Figure 3: Discretised 2D Gaussian kernel

```
    size = s * 3
    x, y = np.meshgrid(np.arange(-size,size + 1), np.arange(-size,size + 1))
    kernel = np.exp(-(x**2  + y**2 / float(s)))
    kernel = kernel / kernel.sum()

    return x, y, kernel
```

2.2 Implement and time the Gaussian convolution

Figure 4 shows an image before an after applying this implementation of Gaussian blur.

2.2.1 Code

```
def convolve(f, kernel, m='nearest'):
    """ Wrapper for scipy's convolve(). Useful for the timeit function. """

    return scipy.ndimage.convolve(f, kernel, mode=m)
```

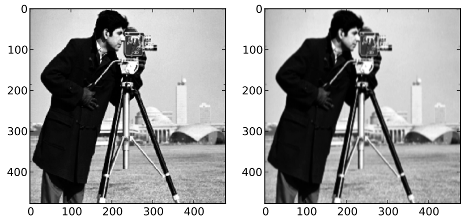

Figure 4: Image convolved with `gauss(4)` kernel

2.2.2 Function performance plot

The plot in figure 5 shows that this implementation, based on a 2D Gauss kernel from `gauss()` is in exponential time complexity.

Note that the labels of the y-axis in the following histogram are of the form `convolve(cameraman, gauss([nr])[2])` where `gauss([nr])[2]` represents the Gaussian kernel for scale = [nr].

3 Separable Gaussian convolution

3.1 Implement `gauss1()` function

3.1.1 Code

```
def gauss1(s):
    """ Returns 1D Gaussian kernel, for separable Gaussian convolution """
```

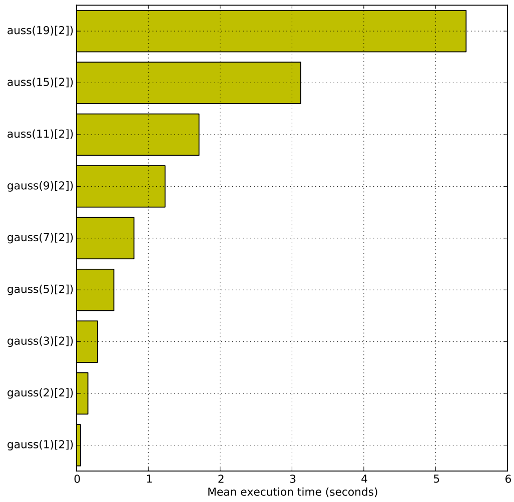

Figure 5: Timing of gauss() based convolution against s

```
    size = s * 3

    x = y = np.arange(-size, size + 1)

    ker_x = np.exp(-(x**2 / float(s)))
    return ker_x / ker_x.sum()
```

3.2 Obtain Gaussian convolution

Figure 6 shows an image before an after applying this implementation of Gaussian blur.

3.2.1 Code

```
def convolve1d(f, kernel1d, m='nearest'):
    """ Convolves along one axis, then along the other. """

    newimage_x = scipy.ndimage.convolve1d(f, kernel1d, axis = 0, mode = m)
```

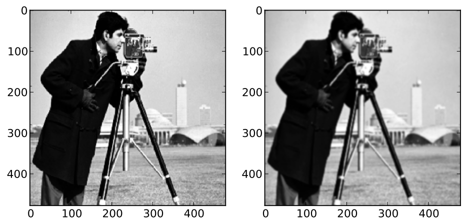

Figure 6: Image convolved with 1-D kernel from `gauss1(4)`

```
    newimage = scipy.ndimage.convolve1d(newimage_x, kernel1d, axis = 1, mode = m)

    return newimage
```

3.2.2 Function performance plot

The plot in figure 7 indicates that the seperable Gauss convolution implementation based on `gauss1()` is in linear time complexity.

Note that the labels of the y-axis in the following histogram are of the form `convolve1d(cameraman, gauss1([nr]))` where `gauss1([nr])` represents the 1D Gaussian kernel for scale = [nr].

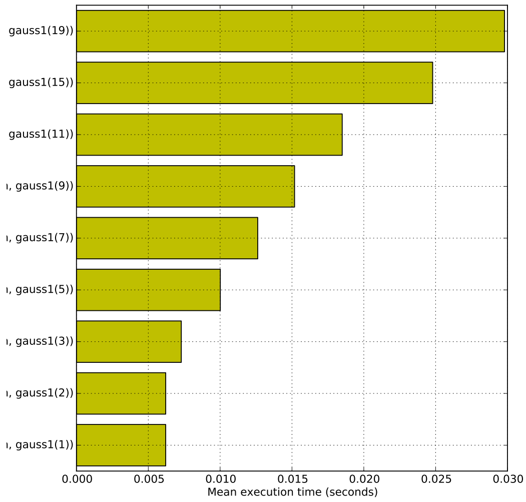

Figure 7: Timing of gauss() based convolution against s

4 Gaussian derivatives

4.1 Show that derivatives of 2d Gauss function are separable

The general 1d Gaussian functions has the form $f(x) = ae^{-\frac{(x-b)^2}{2c^2}}$ and the general 2d function has the form $f(x, y) = ae^{-\left(\frac{(x-x_O)^2}{2\sigma_x^2}+\frac{(y-y_O)^2}{2\sigma_y^2}\right)}$. We use the 1d Guassian function with a = 1, b = 0, and $2\sigma_x^2 = s$ and we use the 2d Gaussian funtion with a = 1, $x_0 = y_0 = 0$ and $2\sigma_x^2 = s$.

In the Separable Gaussian convolution section we have shown that we can express the 2d convolution as a multiplication of the two 1d convolutions. With this basis, we can show that the all derivatives are of the the same form, but with different constants since we are deriving an exponential function. This is import because the derivative of e^x is e^x. This means that the function continues to be a Gaussian function no matter how many times and no matter over witch dimension we derive.

To show this we derive our Gaussian function analytically in both dimensions up to the second order.

$$
\begin{aligned}
f(x,y) &= e^{(-\frac{(x)^2+(y)^2}{s})} \\
f(x,y)_x &= \frac{2x}{s} e^{(-\frac{(x)^2+(y)^2}{s})} \\
f(x,y)_y &= \frac{2y}{s} e^{(-\frac{(x)^2+(y)^2}{s})} \\
f(x,y)_{xx} &= \frac{4x^2-2s}{s^2} e^{(-\frac{(x)^2+(y)^2}{s})} \\
f(x,y)_{yy} &= \frac{4y^2-2s}{s^2} e^{(-\frac{(x)^2+(y)^2}{s})} \\
f(x,y)_{xy} &= \frac{4xy}{s^2} e^{(-\frac{(x)^2+(y)^2}{s})}
\end{aligned}
$$

4.2 Implement `gD(F, s, iorder, jorder)` function

4.3 Visualise 2-jet of cameraman image

5 Canny edge detector

5.1 Implement Canny edge detector

5.2 Test Canny function on cameraman image