

## CPS296.1- Homework 2

Due on September 24, 2007

---

Questions may continue on the back. Please write clearly. What I cannot read, I will not grade.

1. In this problem, image pixel positions are denoted by their row, column coordinates  $(r, c)$ , with the top-left pixel at row 1 and column 1. Directions in the image are denoted by  $x$  and  $y$ :  $x$  is horizontal and growing to the right (positive  $c$  direction), and  $y$  is vertical and growing upwards (negative  $r$  direction).

The Matlab function `gradient` computes an approximation of the image gradient by central differences. For instance, the  $x$  derivative at row  $r$  and column  $c$  is computed as follows:

$$(I(r, c + 1) - I(r, c - 1))/2 .$$

The function `xyGradient` provided on the homework web page approximates the derivative by convolution with the discrete samples of the derivatives of a Gaussian function, as explained in the class notes on image processing. The function `xyGradient` also returns the magnitude of the gradient if a third output argument is specified in the call.

Concerning image boundaries, `gradient` uses asymmetric differences when needed to extend the derivatives to every pixel. For instance, along the left image boundary the  $x$  derivative at position  $(r, 1)$  is approximated as follows:

$$I(r, 2) - I(r, 1) .$$

In contrast, `xyGradient` does not compute the derivative wherever the convolution kernel would exceed the image boundaries. This results in a rim of zeros around the image. To infer the size of the rim, you can call `xyGradient` with five output arguments. The last two are the smoothing and differentiation kernel, respectively. Their lengths are equal and odd, and depend on the value of the smoothing parameter `sigma`. This can be provided as a second input argument to `xyGradient`. If it is not, then `sigma` is set to 1.

To avoid quantization and saturation issues, all images in this problem are double images.

(a) Define a  $100 \times 100$  ramp image

$$I(r, c) = c .$$

Hand in the code you used to make it (no `for` loops!). Compute the exact gradient of this image analytically. What is it? You may want to look at the image with the Matlab function `imagesc`. However, do *not* hand in a printout of the image.

(b) Except for boundaries, `gradient` and `xyGradient` seem to compute the same result. What values do they return for the gradient when called on the ramp image you made earlier? Just give the pixel values somewhere in the middle of the output images. Use the default value `sigma = 1` for `xyGradient`.

(c) Now write a `for` loop that runs the two functions `gradient` and `xyGradient` on the image

$$I_n = I + \text{sigmaNoise} * \text{randn}(\text{size}(I))$$

Change the scalar variable `sigmaNoise` in a `for` loop from 0 to 50 in steps of 10. At each iteration, compute the root-mean squared error `rms` as follows:

```
dx = Ix(rs:re, cs:ce) - Ix0;
rms = std(dx(:));
```

In this code, `rs`, `re`, `cs`, `ce` are defined so as to avoid the rim of boundary pixels computed by `xyGradient`, and `Ix0` is a number representing the exact value of the gradient of the ramp image in the  $x$  direction (one of the values you computed earlier). The image `Ix` is the  $x$  gradient image returned by `gradient`. In the same loop, also compute the root-mean squared errors for the  $x$  gradient image returned by `xyGradient` with the second input argument `sigma` of this function set to 1, 2, 4, 8, respectively.

Hand in both your code and a single plot that shows the five resulting error curves as a function of `sigmaNoise`. Label the plot axes properly, and make sure in particular that the  $x$  axis shows the actual values of `sigmaNoise`. Put a legend on your graph (`help legend`). The code you hand in should also show how you made the plot.

(d) If you double the value of the parameter `sigma` to `xyGradient`, by how much would you expect the root-mean-squared error in the previous experiments to decrease, and why? Does this agree with your data? If so, you are done with this question. If not, can you guess possible reasons for the discrepancy?

2. A ramp is an easy signal for differentiation. Your earlier experiments show that even then it is important to account for noise in the design of a differentiator. This problem focuses on the fact that neither `gradient` nor `xyGradient` gives an exact derivative (in the sense discussed in the class notes), even in the absence of noise.

- (a) The Matlab function with header

```
function p = plaid(n, fM)
```

provided on the homework web page computes the  $n \times n$  image with pixel values

$$A(r, c) = S(r)S(c) \quad \text{where} \quad S(k) = \sin \left( 2\pi f_M \left( \frac{k-1}{n} \right)^2 \right).$$

This image contains a plaid of horizontal and vertical sinusoids with frequencies that increase from zero to a maximum value  $f_M$  periods per image size.

- (b) Derive an expression for the two partial derivatives of  $A(r, c)$ :

$$\frac{\partial A}{\partial x} = \frac{\partial A}{\partial c} \quad \text{and} \quad \frac{\partial A}{\partial y} = -\frac{\partial A}{\partial r}.$$

- (c) Modify the function `plaid` to have the new header

```
function [p, px, py] = plaid(n, fM)
```

and hand in images `px` and `py` with the two derivatives you just computed for  $n = 512$  and  $f_M = 4$ . Display images with `imagesc` for proper scaling of the pixel values.

- (d) Compute the gradient numerically with both `gradient` and `xyGradient` (default value of `sigma`). Only hand in plots of the gradient images  $I_x^{(m)}$  and  $I_y^{(m)}$  computed with `gradient`.

- (e) Plot two mesh diagrams (`help mesh`) with the differences  $\epsilon_m = I_x^{(m)} - I_x$  and  $\epsilon_h = I_x^{(h)} - I_x$  where  $I_x$  is the derivative computed analytically,  $I_x^{(m)}$  is the result from `gradient`, and  $I_x^{(h)}$  is the result from `xyGradient`. [Note: if the differences are huge, you probably made a mistake in the analytical differentiation. Debug until discrepancies are reasonable, say, up to 0.1 or so.]

- (f) The previous result shows that even in the absence of noise neither `gradient` nor `xyGradient` does a perfect job. We should expect this from `gradient` because this function unashamedly computes a finite difference, not a derivative. We can expect problems with `xyGradient` as well, because the Gaussian is not an interpolation function, so the hybrid convolution with a Gaussian does not reconstruct the original, continuous signal exactly. Analysis of the errors from `xyGradient` is too difficult here.<sup>1</sup> Instead, we look at errors from `gradient`, which are about equal in magnitude to those from `xyGradient`.

Knowing that `gradient` works by central differences, as explained earlier, write a mathematical expression for the error  $\epsilon_m(r, c)$  defined earlier. Then compute an image with values of  $\epsilon_m$  from your expression, and compare it with the differences you computed numerically in the previous answer. Show your code for  $\epsilon_m$ , and state the results of your comparison qualitatively.

- (g) In *relative* terms, where is the error  $\epsilon_m$  greatest? That is, where is  $e_m = |\epsilon_m/I_x|$  greatest? Plot  $e_m$  wherever the denominator is nonzero, show your code, and state the result in qualitative terms.

---

<sup>1</sup>These errors could be substantially reduced with a better interpolation function than a Gaussian.