# How do I compute the gradient vector of pixels in an image?

I'm trying to find the curvature of the features in an image and I was advised to calculate the gradient vector of pixels. So if the matrix below are the values from a grayscale image, how would I go about calculating the gradient vector for the pixel with the value '99'?

```
21 20 22 24 18 11 23
21 20 22 24 18 11 23
21 20 22 24 18 11 23
21 20 22 99 18 11 23
21 20 22 24 18 11 23
21 20 22 24 18 11 23
21 20 22 24 18 11 23
```

Apologies for asking such an open ended question, I've never done much maths and am not sure how to start tackling this.

(multivariable-calculus) (image-processing) (gradient-flows)

asked Aug 12 '15 at 13:54
Jack Simpson
**140** 2 9

you might try dsp.stackexchange for these type of questions – geometrikal Aug 12 '15 at 13:58

My first reaction would be that pixel luminosity is not a differentiable function at that point, i.e. there is no gradient. 99 is a rather extreme value in comparison to its neighbors. If anything, the gradient could vanish, because this is a local extremum. – Jyrki Lahtonen ♦ Aug 12 '15 at 21:04

Matlab has a function called "gradient" that will compute the discrete gradient for you. Just one line of code. – littleO Jun 18 '16 at 0:56

You can read up on convolution on wikipedia. en.wikipedia.org/wiki/Convolution#Discrete_convolution <-- this is what many methods mentioned use as long as the image is sampled on any evenly spaced and relatively nice grid. – mathreadler Jul 18 '16 at 21:45

## 3 Answers

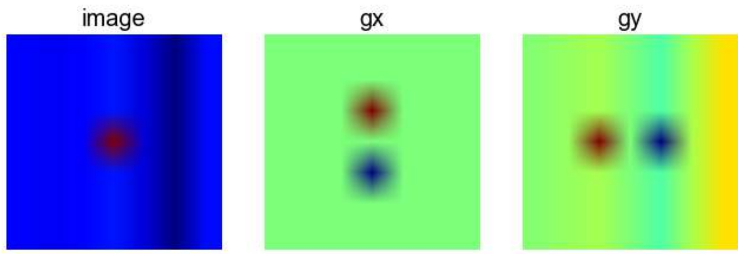
In Python you can use the numpy.gradient function to do this. This said function uses central differences for the computation, like so:

$$\nabla_x I(i,j) = \frac{I(i+1,j) - I(i-1,j)}{2}, \;\; \nabla_y I(i,j) = \frac{I(i,j+1) - I(i,j-1)}{2}.$$

Here is a code snippet for your specific image:

```
import numpy as np
import matplotlib.pyplot as plt

# load image
img = np.array([[21.0, 20.0, 22.0, 24.0, 18.0, 11.0, 23.0],
                [21.0, 20.0, 22.0, 24.0, 18.0, 11.0, 23.0],
                [21.0, 20.0, 22.0, 24.0, 18.0, 11.0, 23.0],
                [21.0, 20.0, 22.0, 99.0, 18.0, 11.0, 23.0],
                [21.0, 20.0, 22.0, 24.0, 18.0, 11.0, 23.0],
                [21.0, 20.0, 22.0, 24.0, 18.0, 11.0, 23.0],
                [21.0, 20.0, 22.0, 24.0, 18.0, 11.0, 23.0]])
print "image =", img

# compute gradient of image
gx, gy = np.gradient(img)
print "gx =", gx
print "gy =", gy

# plotting
plt.close("all")
plt.figure()
plt.suptitle("Image, and it gradient along each axis")
ax = plt.subplot("131")
ax.axis("off")
ax.imshow(img)
ax.set_title("image")

ax = plt.subplot("132")
ax.axis("off")
ax.imshow(gx)
ax.set_title("gx")
```

Image, and it gradient along each axis




To answer your specific question, the gradient (via central differences!) of the image at pixel with value 99 is 0 along the $x$ axis and $-2$ along the $y$ axis.

answered Aug 12 '15 at 21:15
dohmatob
**3,433** 4 25

Just jumping in here. But what happens at the ends of the matrices then? You wrote `gradient x at i,j = I(i + 1) - i(i - 1) ...` But what happens when `i` does not have `i + 1` or `i - 1` ? (at the end of each row/column) – AlanSTACK May 15 '16 at 6:13

At the boundaries, you can clip the value of the gradients to zero, for example. There are other kinds of boundary conditions though (wrapping, etc.). – dohmatob May 15 '16 at 9:07

---

Suppose the image is continuous and differentiable in $x$ and $y$. Then $I(x, y)$ is the value of the pixel at each $(x, y)$, i.e. $I : \mathbb{R}^2 \mapsto \mathbb{R}$. Recall that the gradient at a point $(u, v)$ is:

$$\nabla I(u, v) = \begin{bmatrix} \frac{\partial I}{\partial x}(u, v) \\ \frac{\partial I}{\partial y}(u, v) \end{bmatrix}$$

Given a discrete grid, you should approximate the partial derivative in $x$ and $y$ directions using finite difference approximations at the point of interest.

Assume your function $I$ is sampled over points $\{1, \ldots, 7\} \times \{1, \ldots, 7\}$ in image-coordinates, i.e. $I(1, 1) = 21$, $I(1, 7) = 23$, etc... So you're looking for the gradient at $(4, 4)$. If you assume the resolution between points is 1, then the forward difference approximation in the $x$ direction gives:

$$\frac{\partial I}{\partial x}(4, 4) \approx I(5, 4) - I(4, 4) = 24 - 99$$

Do the same in $y$ to obtain the full gradient at the point.

answered Aug 12 '15 at 14:49
Chester
**979** 8 13

---

The theoretical thing you may want to read up on is convolution and especially **discrete convolution**.

```
lImage = [21,20,22,24,18,11,23;
          21,20,22,24,18,11,23;
          21,20,22,24,18,11,23;
          21,20,22,99,18,11,23;
          21,20,22,24,18,11,23;
          21,20,22,24,18,11,23;
          21,20,22,24,18,11,23];
```

Then the command "conv2" creates a 2 dimensional convolution, first we can do one in the x direction by feeding it a row vector with the [1,0,-1] filter:

```
dx = conv2(lImage,[1,0,-1],'valid')
```

which gives us the output:

```
dx =

    1    4   -4  -13    5
    1    4   -4  -13    5
    1    4   -4  -13    5
    1   79   -4  -88    5
    1    4   -4  -13    5
    1    4   -4  -13    5
    1    4   -4  -13    5
```

and the y direction we can easily do by just applying a transpose (denoted by ') on the x filter row vector:

```
dy = conv2(lImage,[1,0,-1]','valid')
```

gives us the output:

```
dy=

    0    0    0    0    0    0    0
    0    0    0   75    0    0    0
    0    0    0    0    0    0    0
    0    0    0  -75    0    0    0
    0    0    0    0    0    0    0
```

answered Jul 18 '16 at 23:03

mathreadler
**13.1k** 7 17 54