Edge Detection

MATLAB:

```
Jb = rgb2gray(J);
imagesc(Jb);axis image; colormap(gray);
bw = edge(Jb,'canny');
```

Python:

```
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np
from skimage.feature import canny

Jb = np.array(Image.open('demo.png').convert('L'))
plt.figure(); plt.imshow(Jb, cmap='gray')
bw = canny(Jb)
```

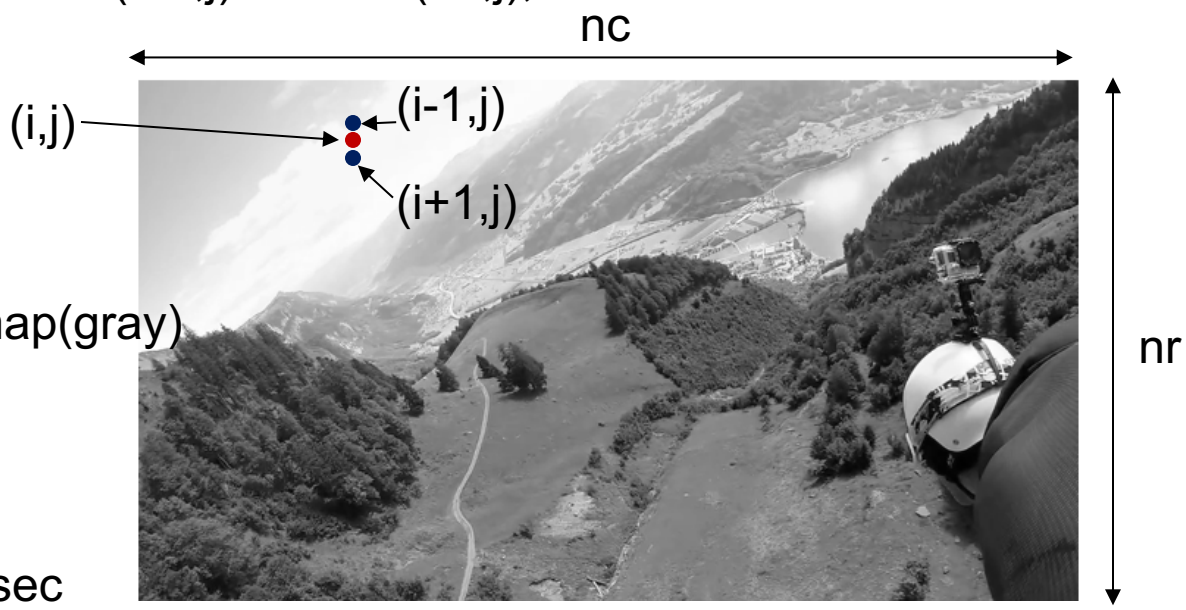
Numerical Image Filtering

MATLAB:

```
# Looping through all pixels
[nr,nc] = size(Jb);
J_out = zeros(nr,nc);
for i=1:nr,
    for j=1:nc;
        if (i<nr) && (i>1),
            J_out(i,j) = 2*Jb(i,j) - 0.8*Jb(i+1,j) - 0.8*Jb(i-1,j);
        else
            J_out(i,j) = Jb(i,j);
        end
    end
end
figure; imagesc(J_out);colormap(gray)
```

Filter

-0.8
2.0
-0.8



Computation time: 0.050154 sec

Python:

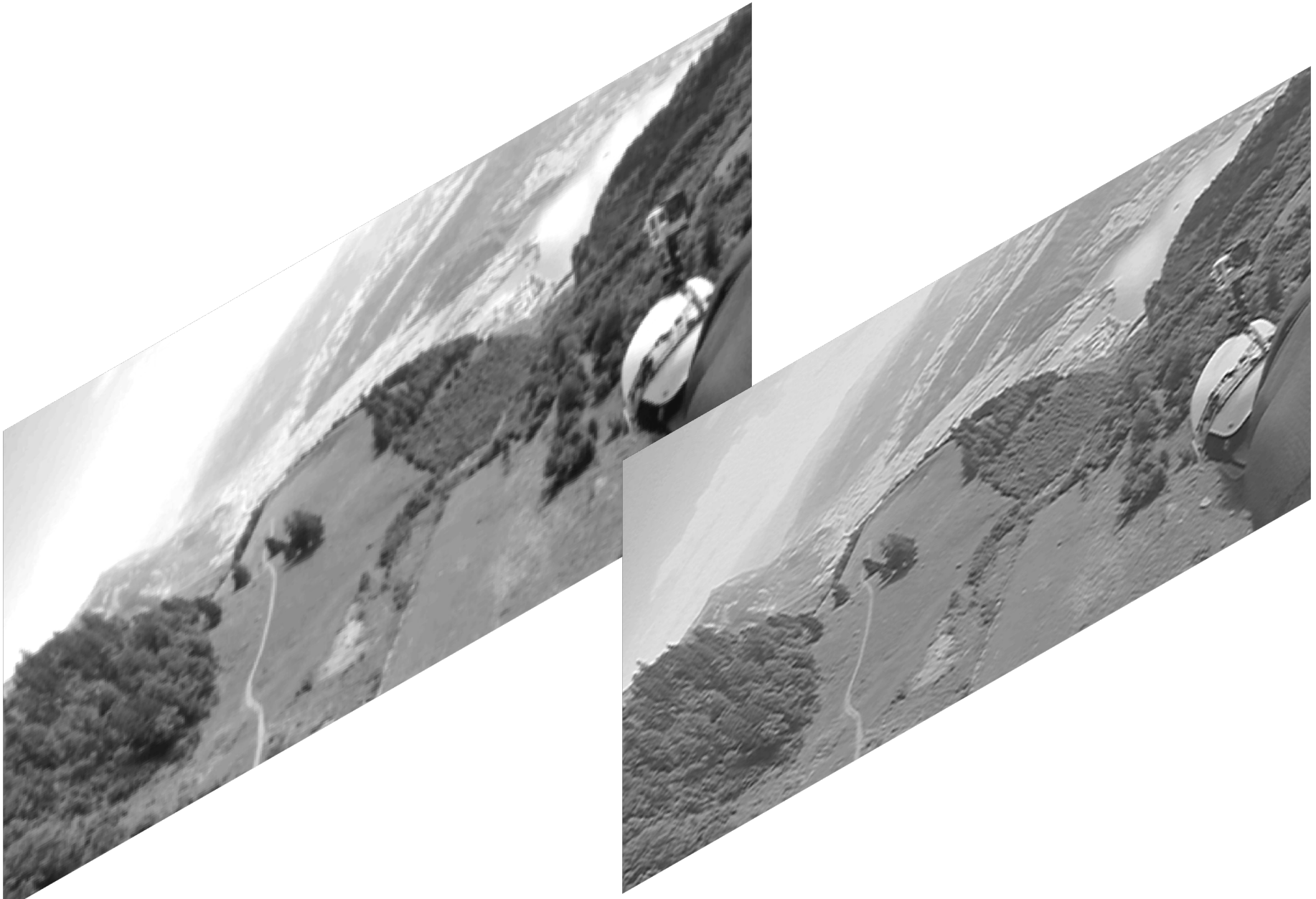
Array index start at 0

```
nr, nc = Jb.shape[0], Jb.shape[1]
J_out = np.zeros((nr,nc))
for i in range(nr):
        for j in range(nc):
                if (i < nr - 1) & (i > 0):
                        J_out[i,j] = 2 * Jb[i,j] - 0.8 * Jb[i+1,j] - 0.8 * Jb[i-1,j]
                else:
                        J_out[i,j] = Jb[i,j]
plt.imshow(J_out, cmap='gray')
plt.show()
```

Computation time: 1.45 sec

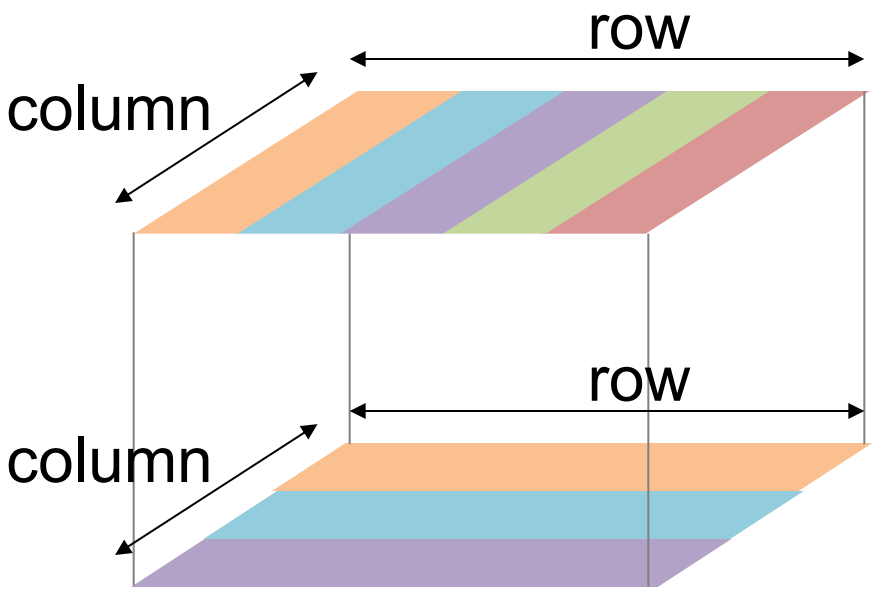
Python for loop is slower than Matlab

Numerical Image Filtering



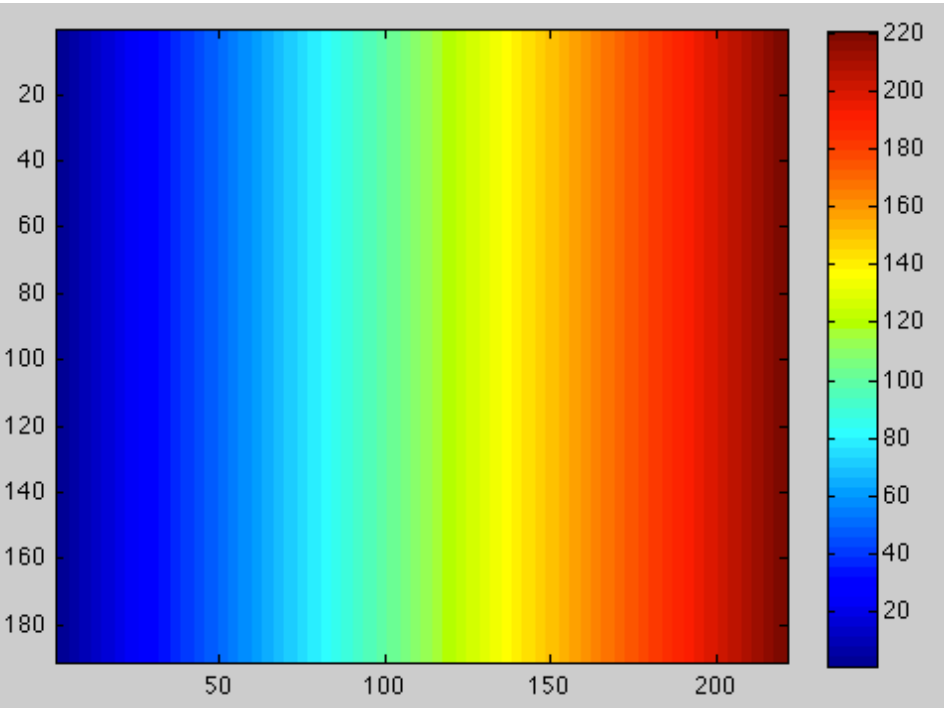
Convolution without Looping using meshgrid

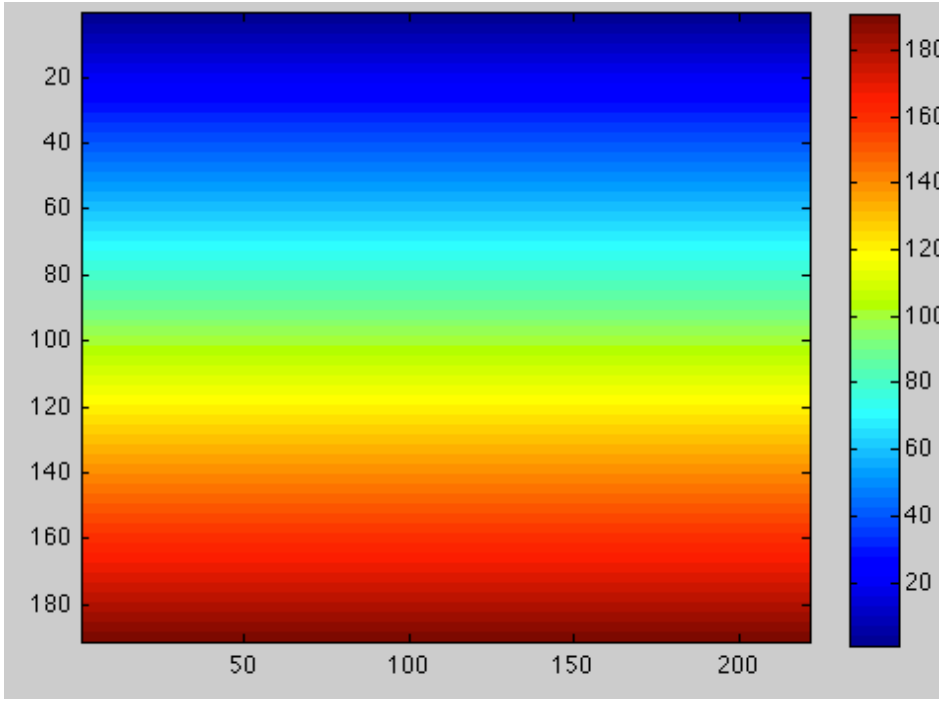
```
>> [x,y] = meshgrid(1:5,1:3)

x =

     1     2     3     4     5
     1     2     3     4     5
     1     2     3     4     5

y =

     1     1     1     1     1
     2     2     2     2     2
     3     3     3     3     3
```



MATLAB:

```
[x,y] = meshgrid(1:nc,1:nr);
figure(1); imagesc(x); axis image; colorbar;
colormap(jet);
figure(2); imagesc(y); axis image; colorbar;
colormap(jet);
```



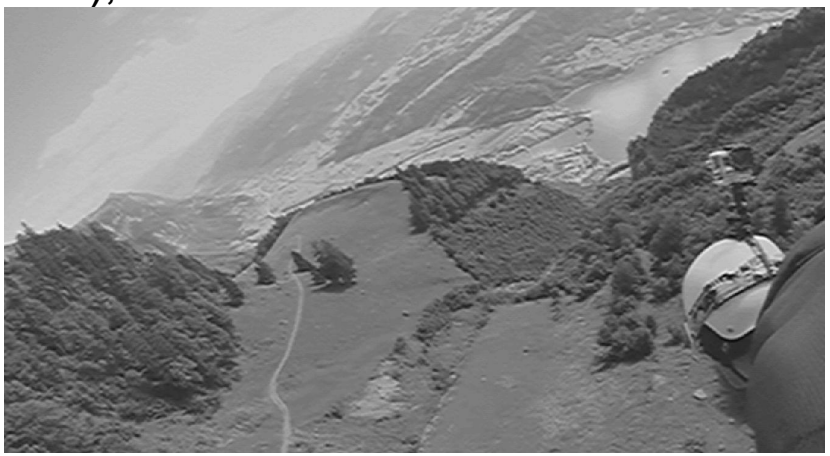


Convolution without Looping using meshgrid

MATLAB:

```
[x,y] = meshgrid(1:nc,1:nr);

y_up = y-1;
y_down = y+1;

y_up = min(nr,max(1,y_up));  % keep y_up index within legal range of [1,nr]
y_down = min(nr,max(1,y_down));

ind_up = sub2ind([nr,nc],y_up(:),x(:));  % create linear index
ind_down = sub2ind([nr,nc],y_down(:),x(:));

J_out = 2*Jb(:) - 0.8*Jb(ind_up) - 0.8*Jb(ind_down);
J_out = reshape(J_out, nr, nc);

figure; imagesc(J_out);colormap(gray)
```

Computation time: 0.024047 sec



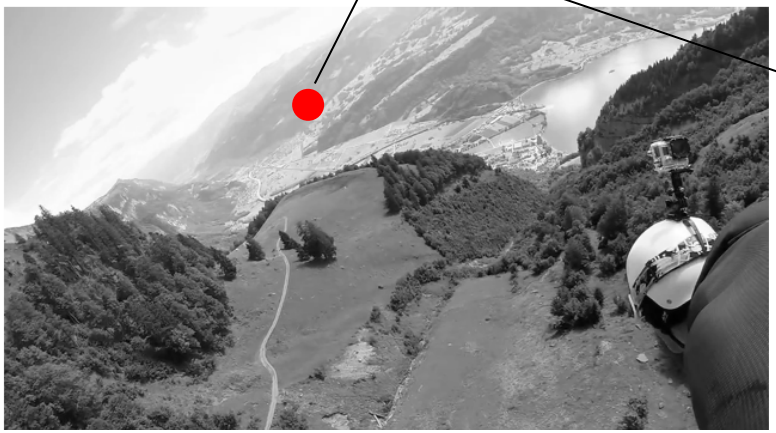
Convolution without Looping using meshgrid

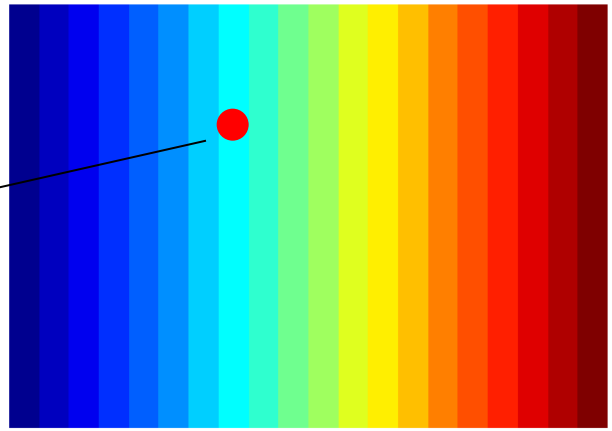
MATLAB:

[x,y] = meshgrid(1:nc,1:nr);

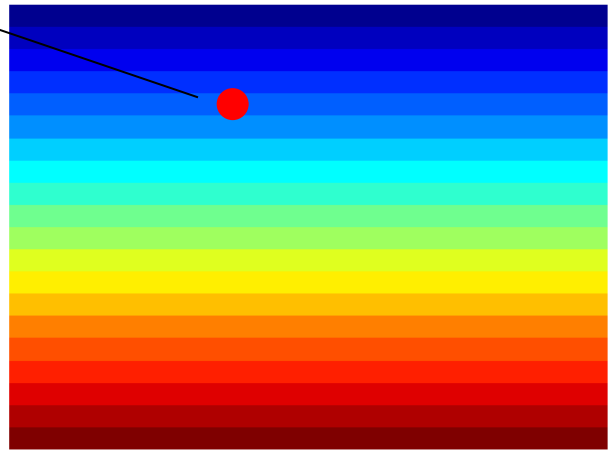
x and y are subscript indice.

Jb_{xy}





x



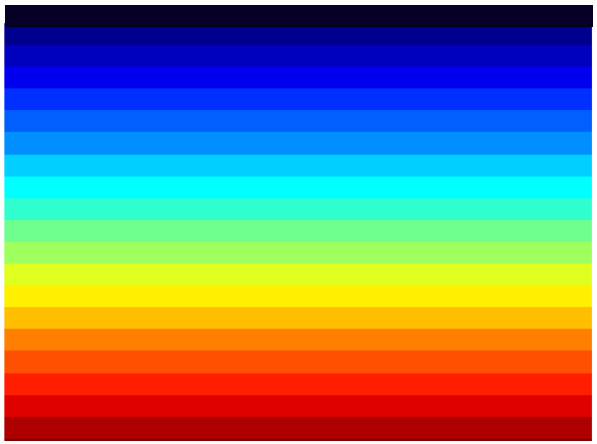
y

Convolution without Looping using meshgrid

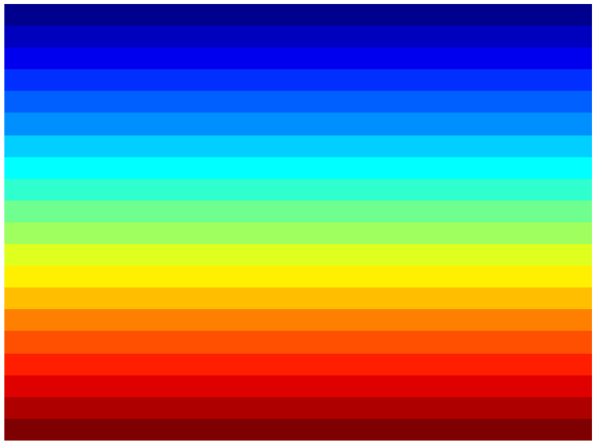
MATLAB:

```
[x,y] = meshgrid(1:nc,1:nr);

y_up = y-1;
y_down = y+1;
```

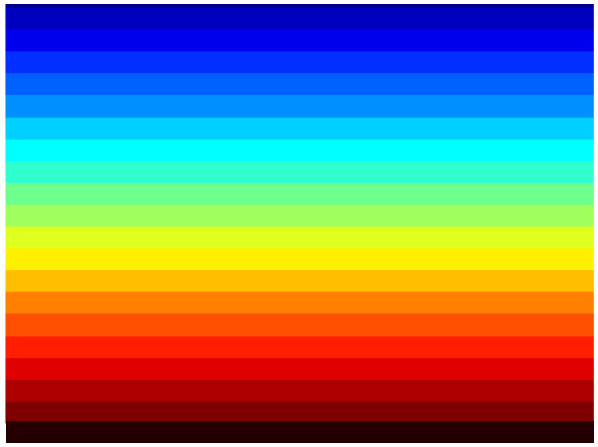
y_up



y



y_down



y_up =

0	0	0	0	0
1	1	1	1	1
2	2	2	2	2

y =

1	1	1	1	1
2	2	2	2	2
3	3	3	3	3

y_down =

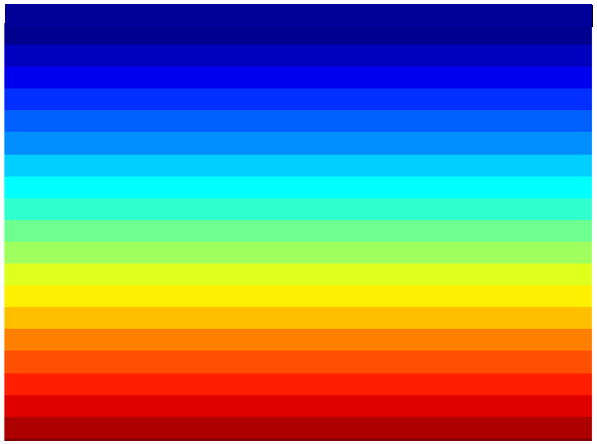
2	2	2	2	2
3	3	3	3	3
4	4	4	4	4

Convolution without Looping using meshgrid

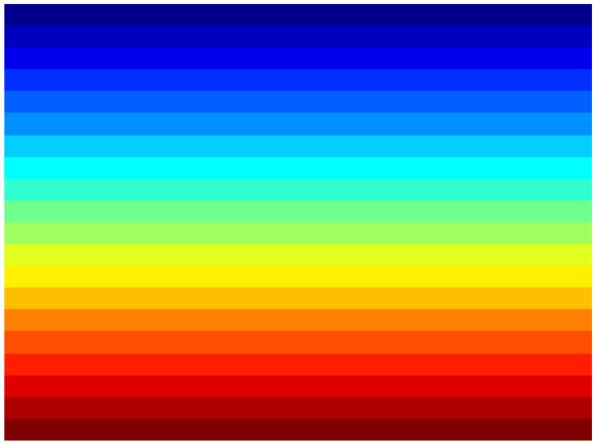
MATLAB:

```
y_up = y-1;
y_down = y+1;

y_up = min(nr,max(1,y_up));  % keep y_up index within legal range of [1,nr]
y_down = min(nr,max(1,y_down));
```

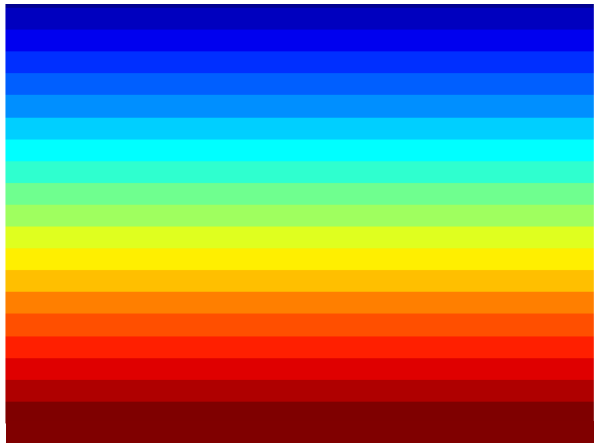
y_up



y



y_down



y_up =

1	1	1	1	1
1	1	1	1	1
2	2	2	2	2

y =

1	1	1	1	1
2	2	2	2	2
3	3	3	3	3

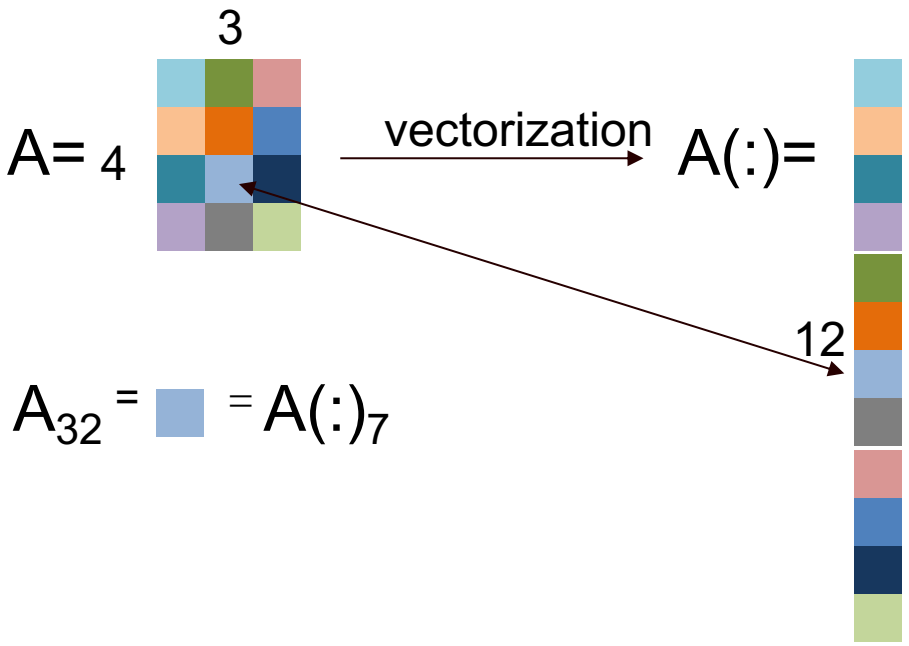
y_down =

2	2	2	2	2
3	3	3	3	3
3	3	3	3	3

Convolution without Looping using meshgrid

MATLAB:

```
y_up = min(nr,max(1,y_up));  % keep y_up index within legal range of [1,nr]
y_down = min(nr,max(1,y_down));

ind_up = sub2ind([nr,nc],y_up(:),x(:));  % create linear index
ind_down = sub2ind([nr,nc],y_down(:),x(:));

linear_index = sub2ind([n_row, n_col], row_subscript, col_subscript)
```

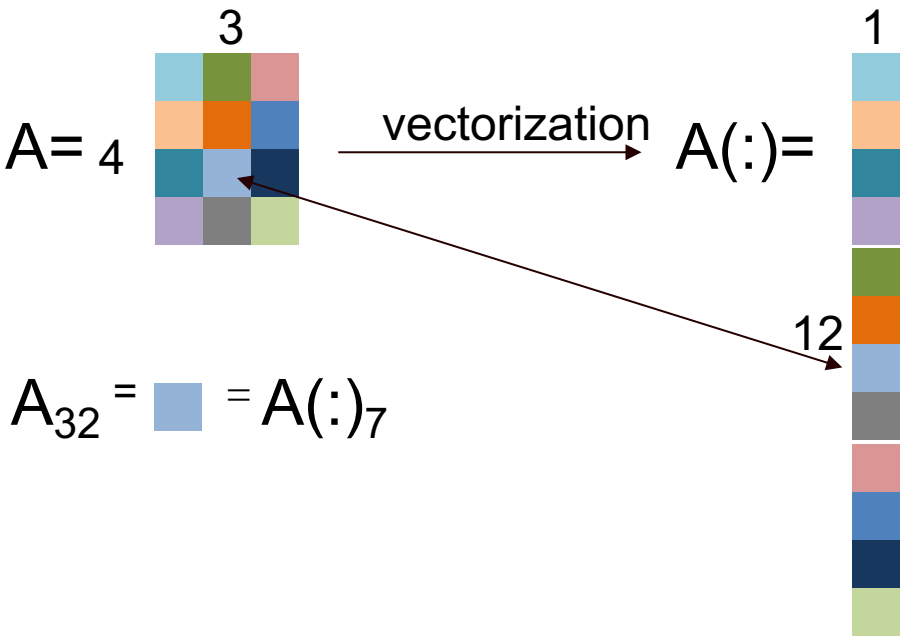


7 = sub2ind([4 3], 3,2)

Convolution without Looping using meshgrid

MATLAB:

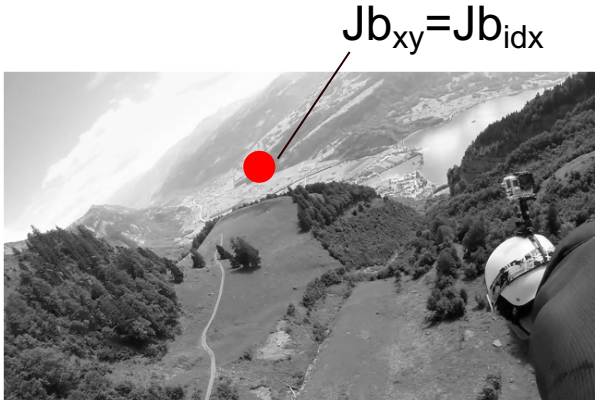
```
y_up = min(nr,max(1,y_up));  % keep y_up index within legal range of [1,nr]
y_down = min(nr,max(1,y_down));

ind_up = sub2ind([nr,nc],y_up(:),x(:));  % create linear index
ind_down = sub2ind([nr,nc],y_down(:),x(:));

linear_index = sub2ind([n_row, n_col], row_subscript, col_subscript)
```



7 = sub2ind([4 3], 3,2)

ind_up = sub2ind([nr,nc],y_up(:),x(:))

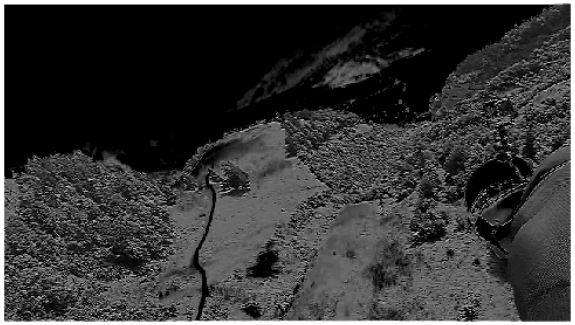
Operation on vectors



Convolution without Looping using meshgrid

MATLAB:

```
J_out = 2*Jb(:) - 0.8*Jb(ind_up) - 0.8*Jb(ind_down);
J_out = reshape(J_out, nr, nc);
figure; imagesc(J_out);colormap(gray)
```



J_out =

2 

-0.8 

-0.8 

Jb

Jb(ind_up)

Jb(ind_down)

Convolution without Looping using meshgrid

Python:

```
x,y = np.meshgrid(np.arange(nc), np.arange(nr))
y_up = y - 1
y_down = y + 1
y_up = np.clip(y_up, 0, nr - 1)
y_down = np.clip(y_down, 0, nr - 1)

coor_up = np.stack([y_up.flatten(), x.flatten()])
ind_up = np.ravel_multi_index(coor_up, (nr, nc))

coor_down = np.stack([y_down.flatten(), x.flatten()])
ind_down = np.ravel_multi_index(coor_down, (nr, nc))

Jb = Jb.flatten()
J_out = 2 * Jb - 0.8 * Jb[ind_up] - 0.8 * Jb[ind_down]
J_out = J_out.reshape(nr,nc)
plt.imshow(J_out, cmap='gray')
plt.show()
```

Converting multi dimension index array to 1d index array. Python is row major Matlab is column major

In python, we can't access multi dimension array using 1d index, so we need to flatten the array before accessing

Computation time: 0.01787 sec VS 1.45 sec (for loop)
Vectorization is necessary for python code !!!

Convolution without Looping using meshgrid

Python:

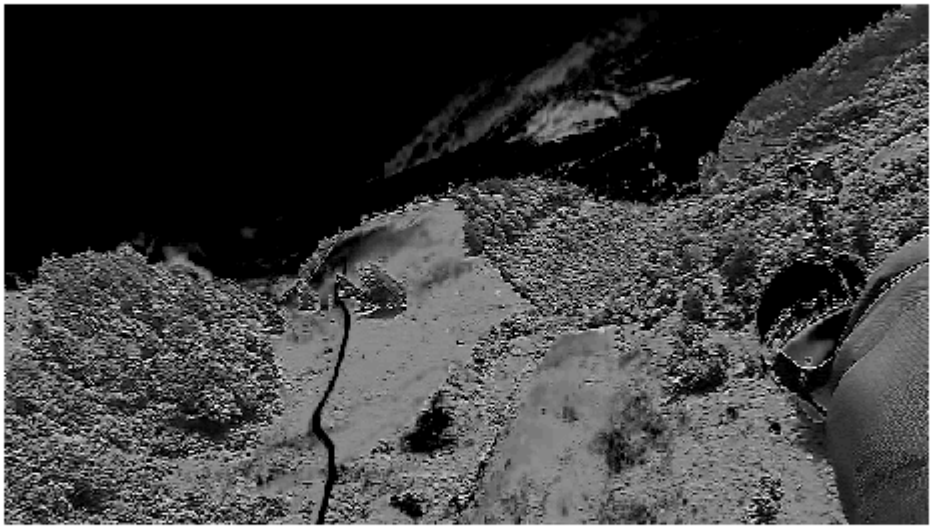
```
x,y = np.meshgrid(np.arange(nc),
np.arange(nr))

y_up = y - 1
y_down = y + 1
y_up = np.clip(y_up, 0, nr - 1)
y_down = np.clip(y_down, 0, nr – 1)

J_out = 2 * Jb - 0.8 * Jb[y_up, x] - 0.8 *
Jb[y_down,x ]

plt.imshow(J_out, cmap='gray')
plt.show()
```

We can use 2d index array to access the array directly and efficiently!

Computation time: 0.00715 sec
vs 0.01787 sec (1d index array)
vs 1.45 sec (for loop)

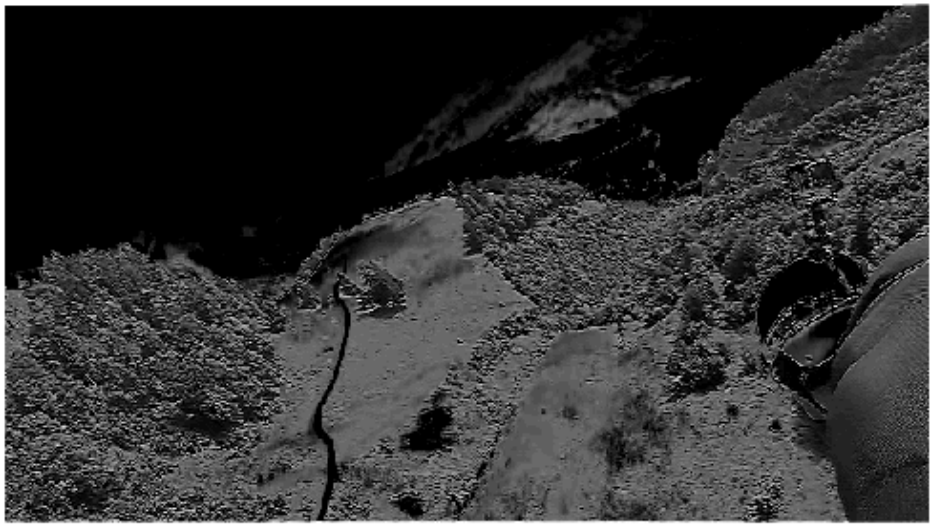
Numpy array in python is highly optimized for vectorization operation

With loop



Computation time: 0.050154 sec

Without loop



Computation time: 0.024047 sec

Canny Edge Detection

$$B(i,j) = \begin{cases} 1 & \text{if } I(i,j) \text{ is edge} \\ 0 & \text{if } I(i,j) \text{ is not edge} \end{cases}$$

Objective: to localize edges given an image.

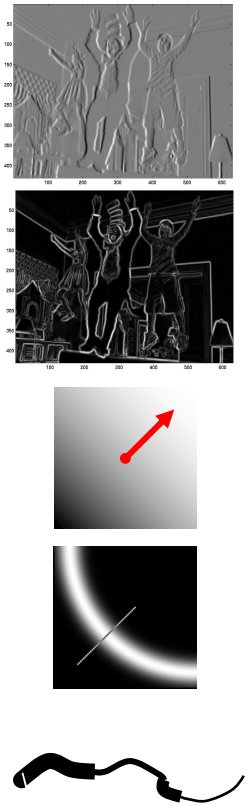
Original image, I

Binary image indicating edge pixels

Edge map image, B

Canny Edge Detection

1. Filter image by derivatives of Gaussian
2. Compute magnitude of gradient
3. Compute edge orientation
4. Detect local maximum
5. Edge linking



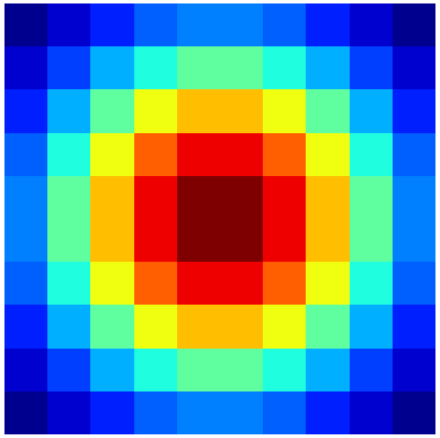
1) Compute Image Gradient

the first order derivative of Image I in x, and in y direction

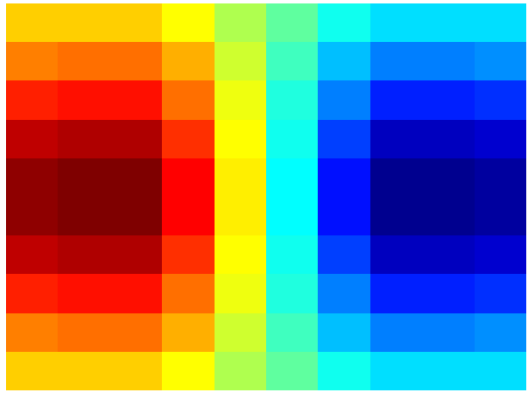
Edge Detection, Step 1, Filter out noise and compute derivative:

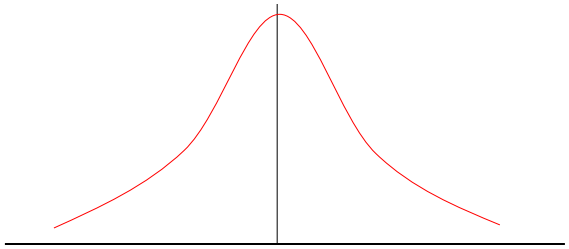
$$(\frac{\delta}{\delta x} \otimes G)$$

Gradient of Gaussian

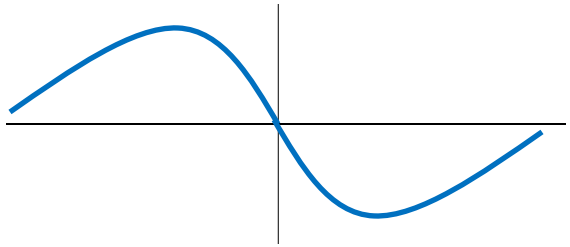


$$\frac{\delta}{\delta x}$$





$$\frac{\delta}{\delta x}$$

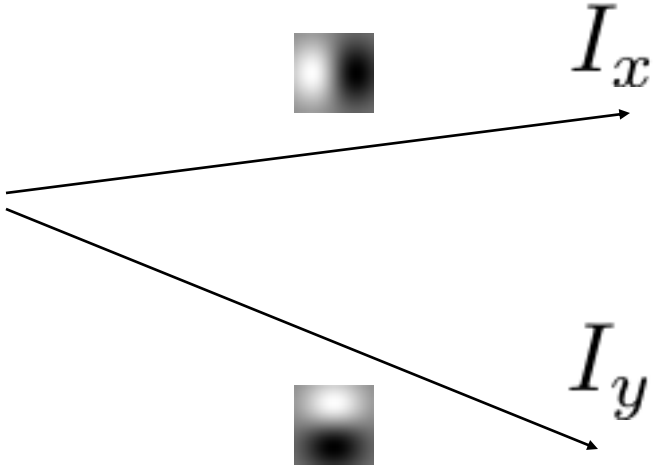


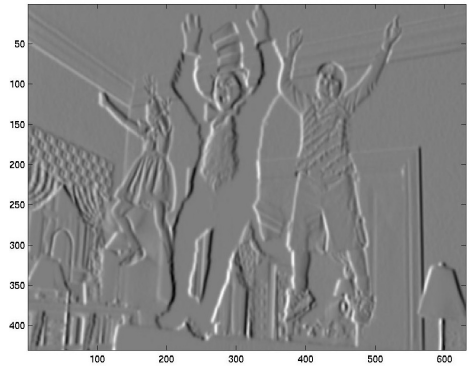
Edge Detection, Step 1, Filter out noise and compute derivative:

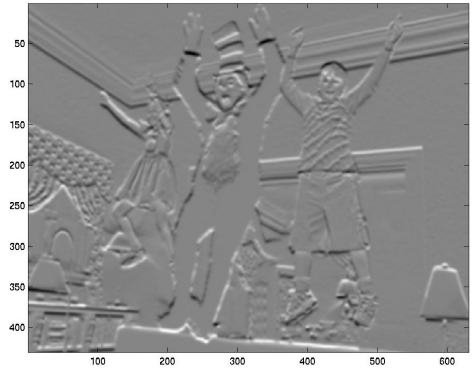
Image

$$\otimes\left(\frac{\delta}{\delta x} \otimes G\right)$$

Smoothed Derivative



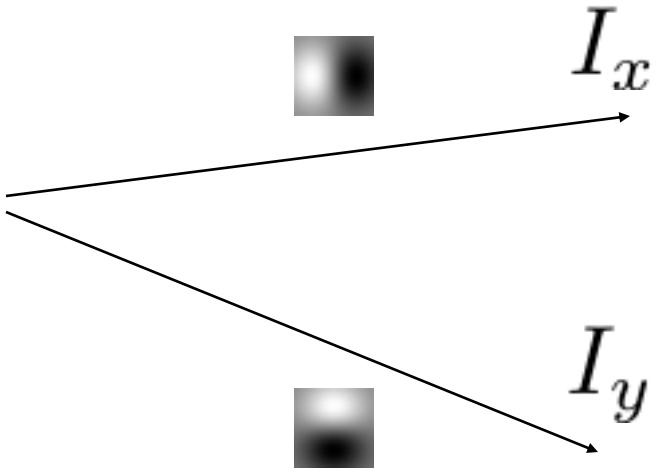


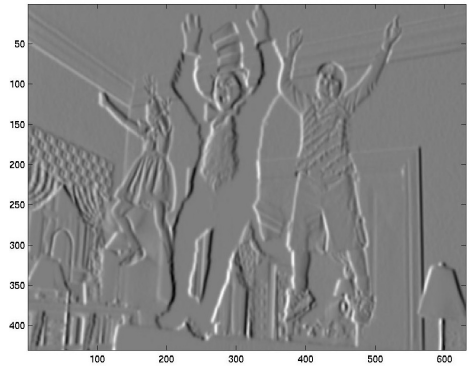


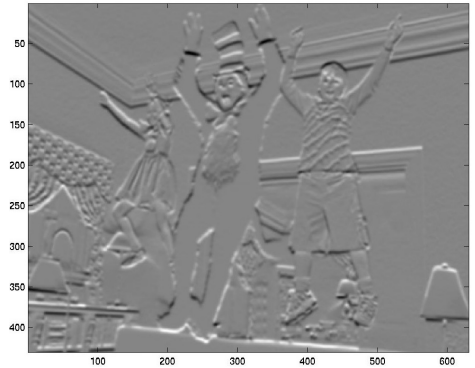
Edge Detection, Step 1, Filter out noise and compute derivative:

MATLAB:

```
>> [dx,dy] = gradient(G); % G is a 2D gaussain
>> Ix = conv2(I,dx,’same’); Iy = conv2(I,dy,’same’);
```







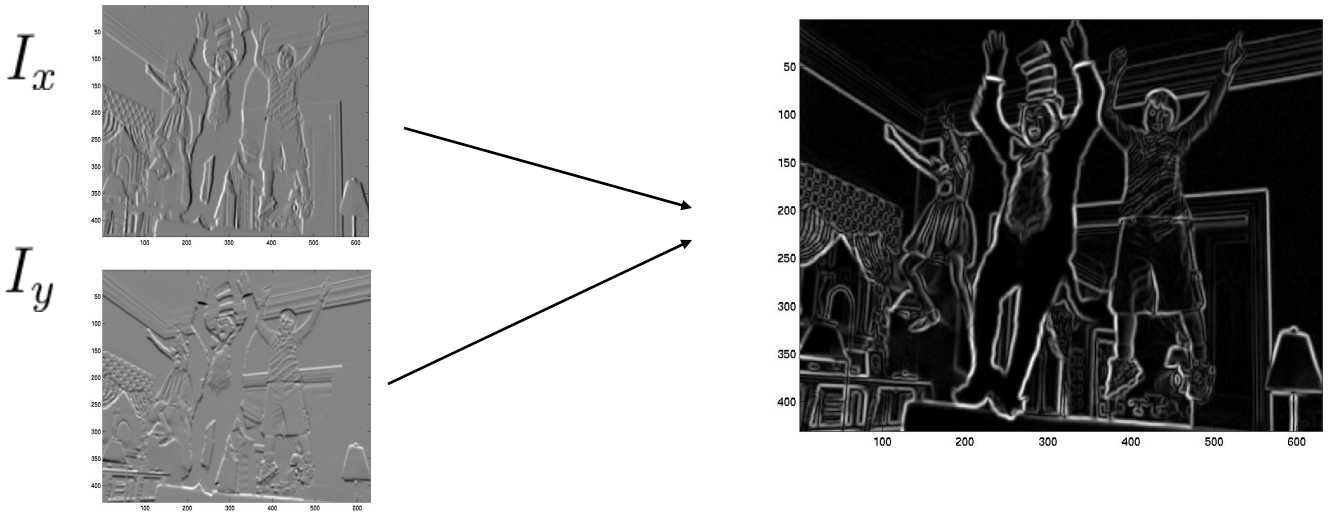
Python:

```
dx, dy = np.gradient(G, axis = (1,0))
Ix = signal.convolve2d(I,dx,'same')
Iy = signal.convolve2d(I,dy,'same')
```

Edge Detection: Step 2
Compute the magnitude of the gradient

Matlab:

```
>> Im = sqrt(Ix.*Ix + Iy.*Iy);
```



Python:

```
Im = np.sqrt(Ix*Ix + Iy*Iy);
```

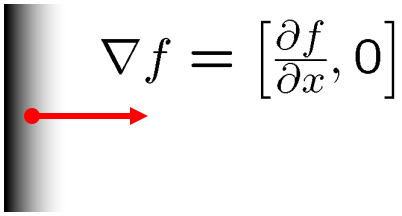
We know roughly where are the edges, but we need their precise location.

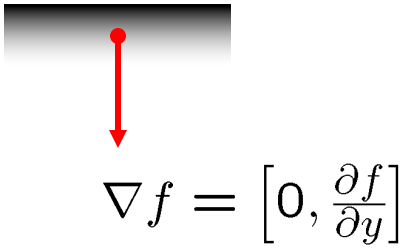
Finding the orientation of the edge

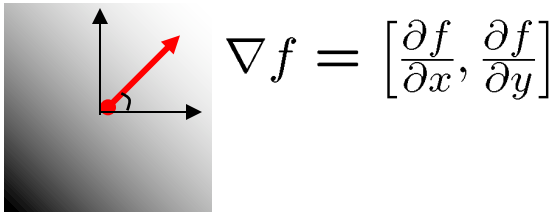
- The gradient of an image:

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

- The gradient points in the direction of most rapid change in intensity







- The image gradient direction is given by:

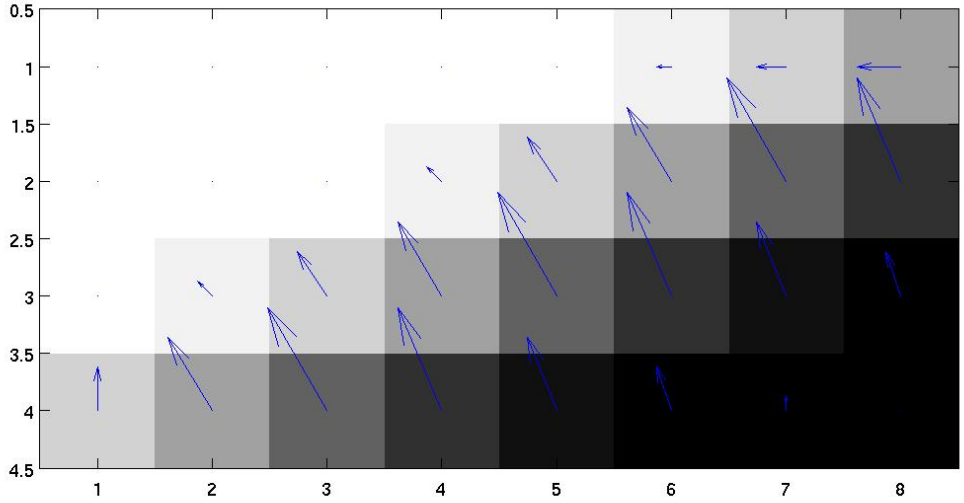
$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

  – how does this relate to the direction of the edge?

$$\theta_{edge} = tan^{-1}(-\frac{\delta f}{\delta x} / \frac{\delta f}{\delta y})$$

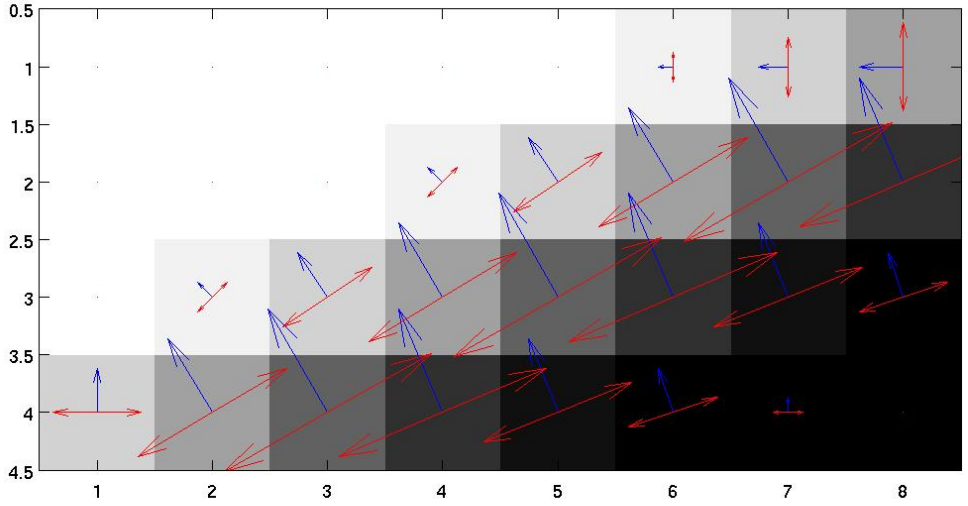
MATLAB:

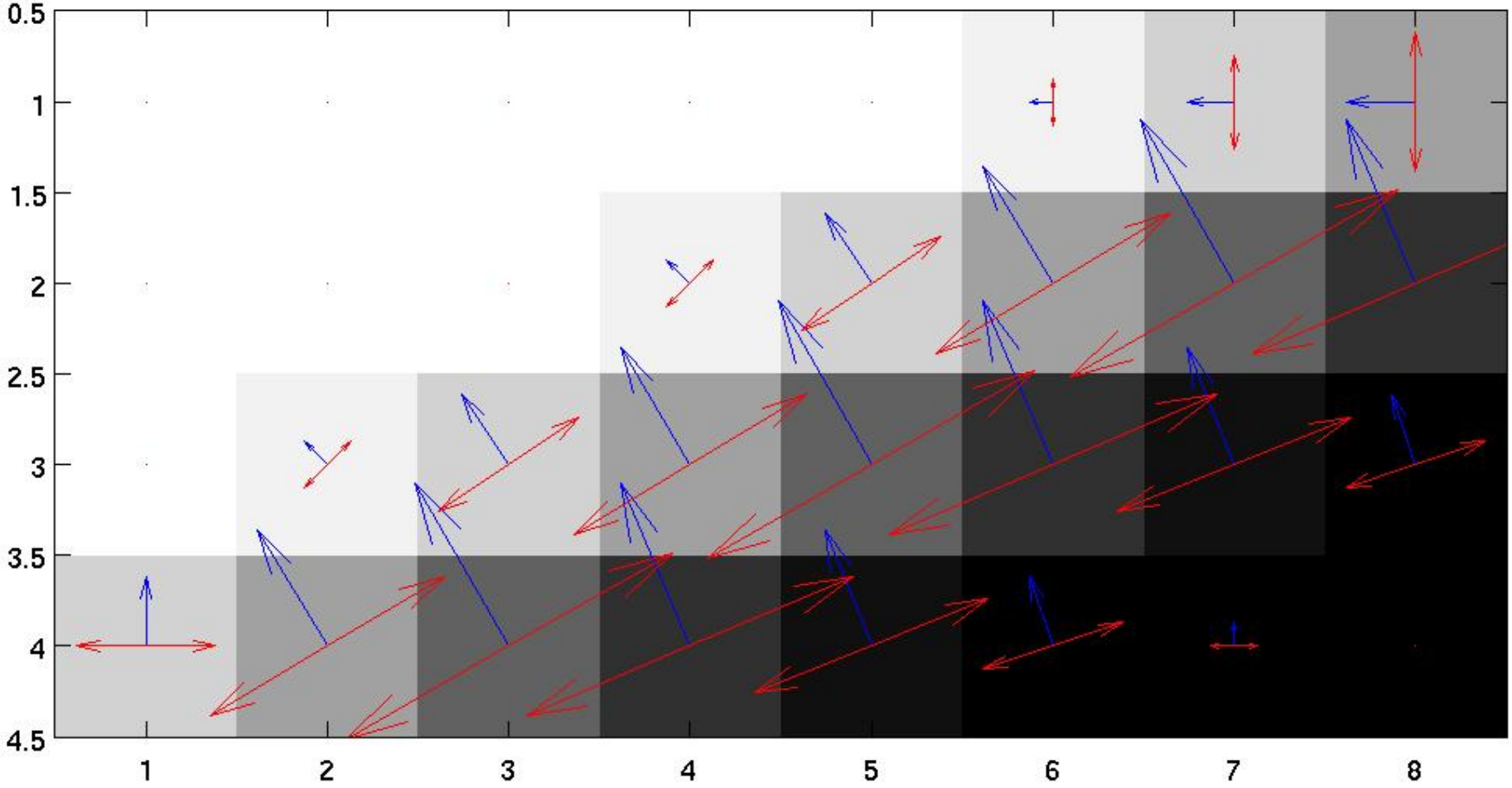
```
%% define image gradient operator
dy = [1;-1];
dx = [1,-1];

%% compute image gradient in x and y
Iy = conv2(I,dy,'same');
Ix = conv2(I,dx,'same');

%% display the image gradient flow
figure(3);clf;imagesc(J);colormap(gray);axis image;
hold on;
quiver(Jx,Jy);
quiver(-Jy,Jx,'r');
quiver(Jy,-Jx,'r');
```



Python:

```
dy = np.array([1,-1]).reshape(2,1)
dx = np.array([1,-1]).reshape(1,2)

%% compute image gradient in x and y
Iy = scipy.signal.convolve2d(I,dy,'same')
Ix = scipy.signal.convolve2d(I,dx,'same')

plt.quiver(-1 * Ix,-1 *  Iy)
```




0.5
1
1.5
2
2.5
3
3.5
4
4.5
1
2
3
4
5
6
7
8

[gx,gy] = gradient(J);
mag = sqrt(gx.*gx+gy.*gy); imagesc(mag);colorbar

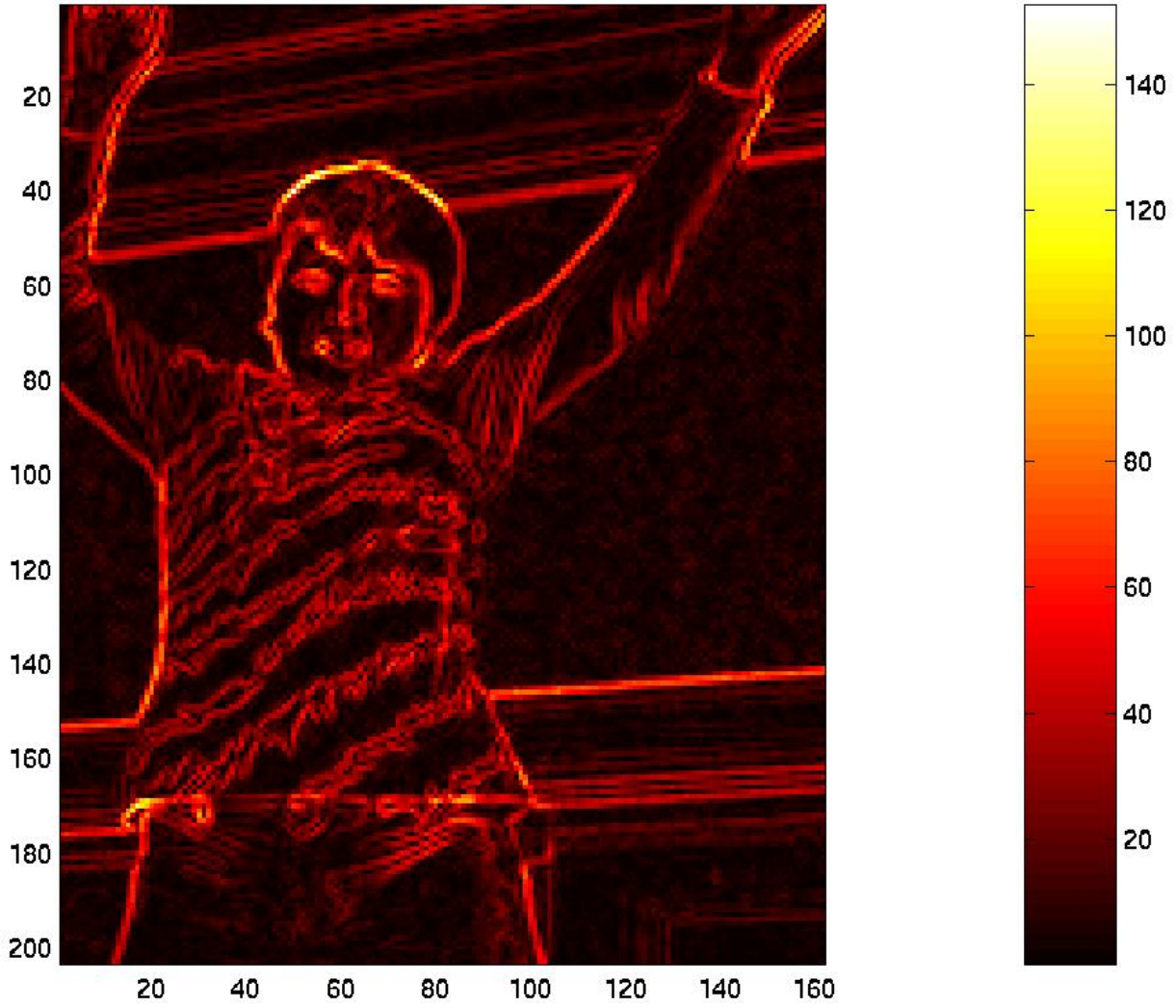
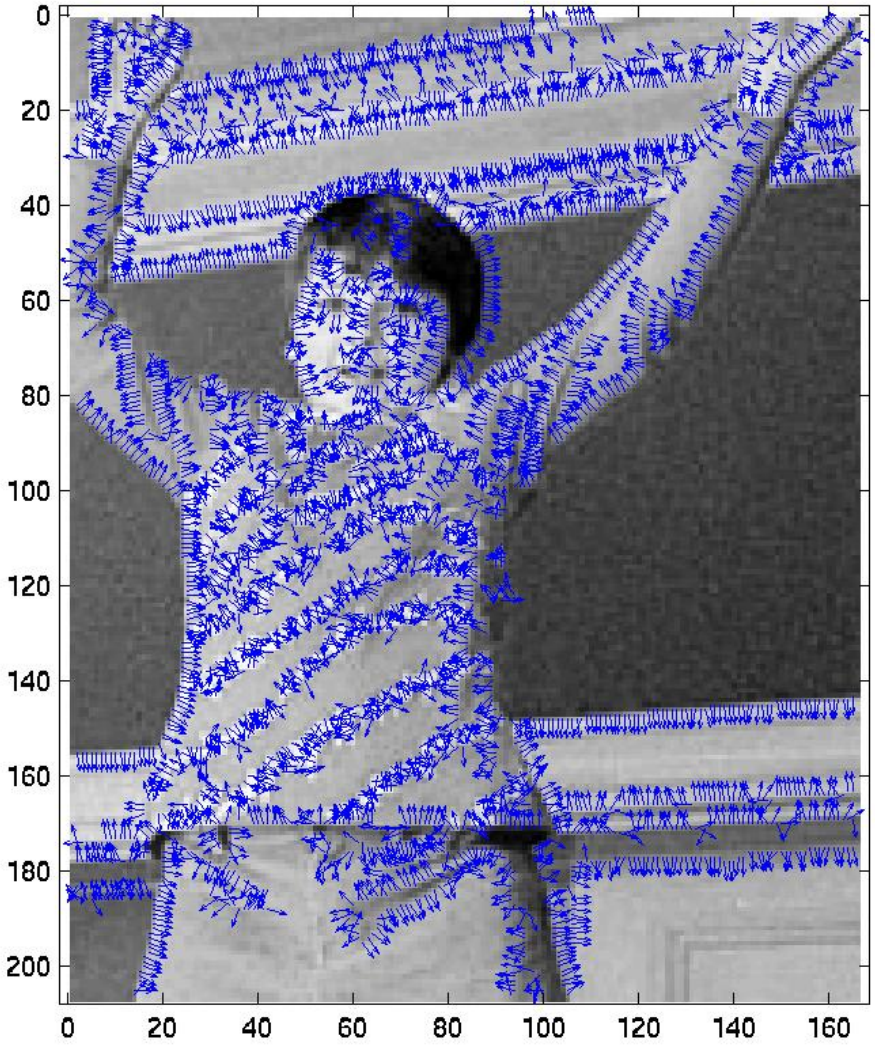
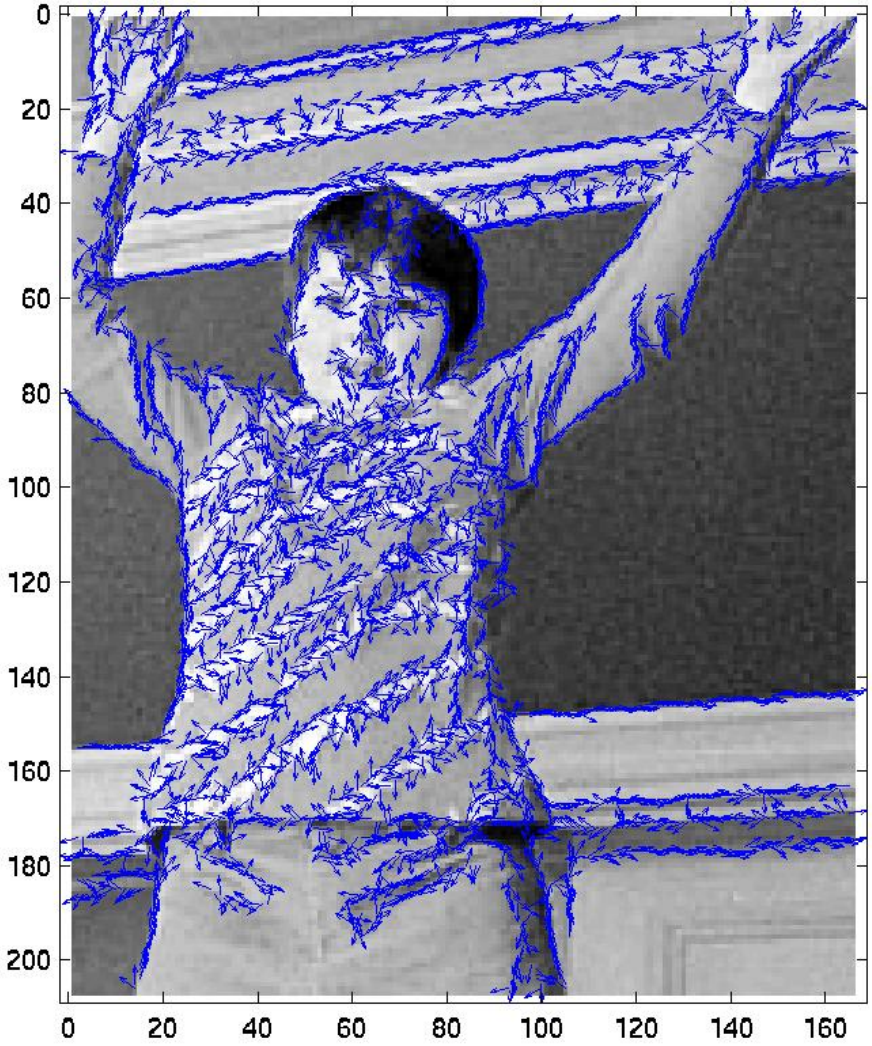


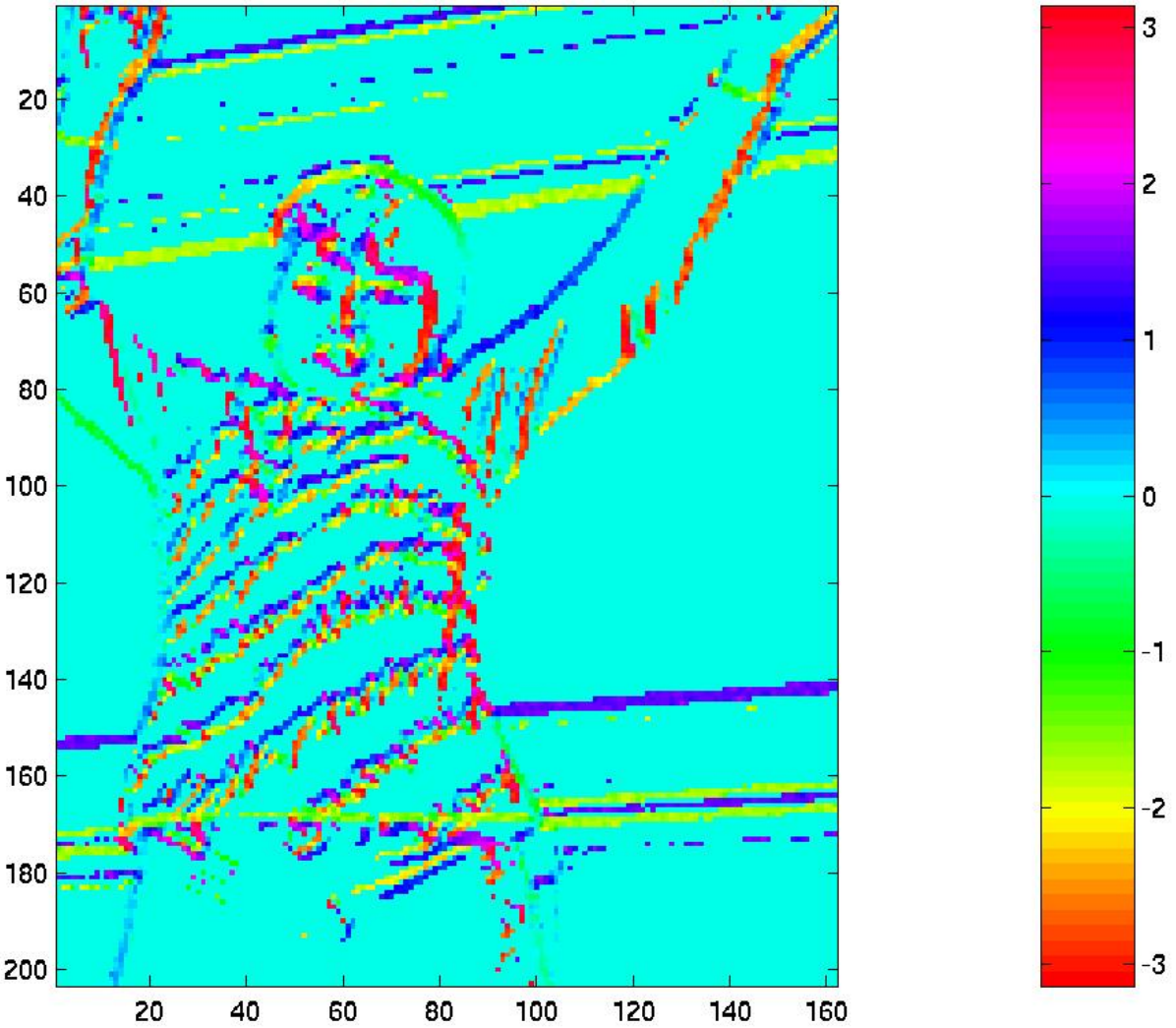
image gradient direction:



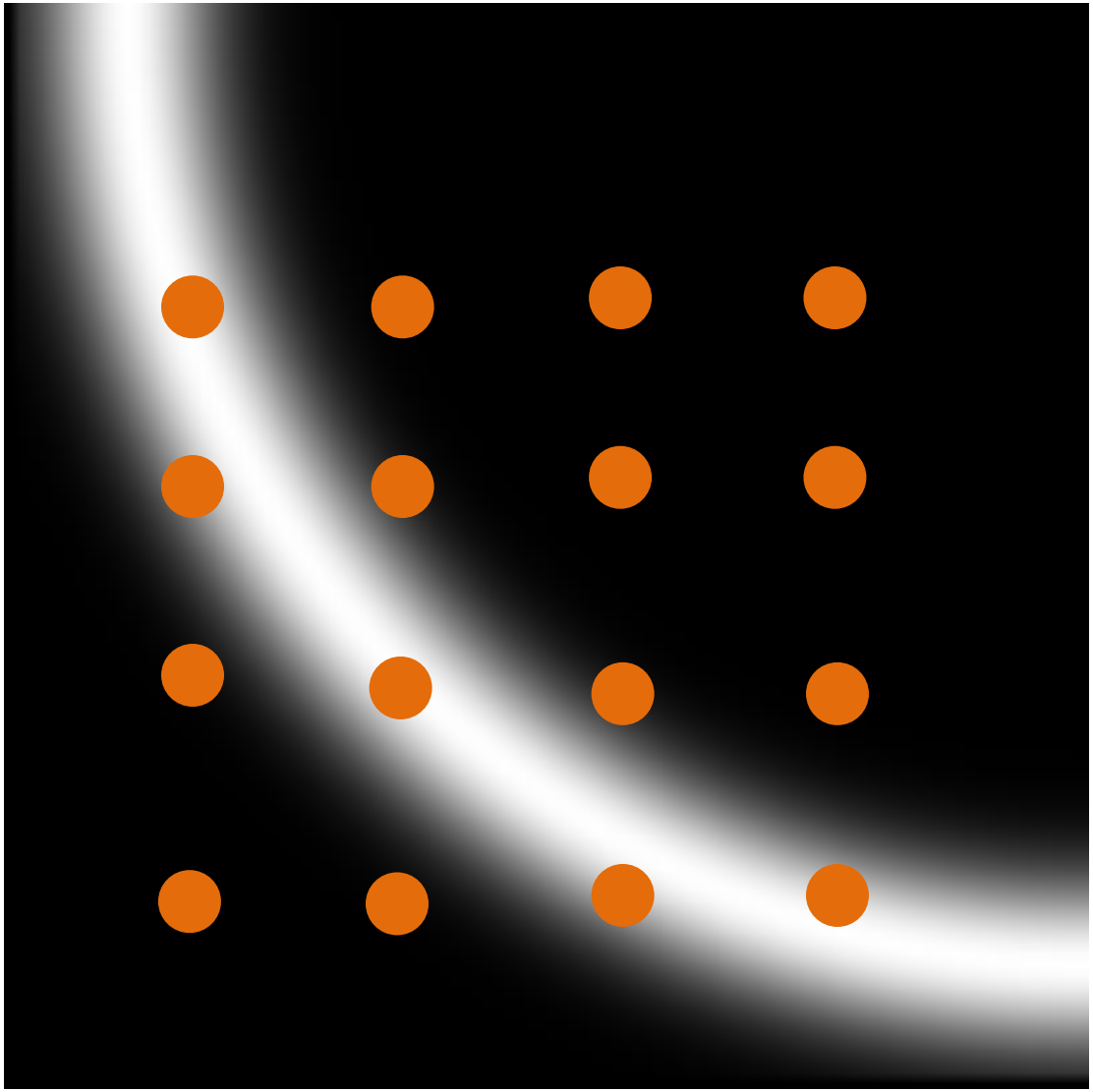
Edge orientation direction:



[gx,gy] = gradient(J);
th = atan2(gy,gx); % or you can use:[th,mag] = cart2pol(gx,gy);
imagesc(th.*(mag>20));colormap(hsv); colorbar

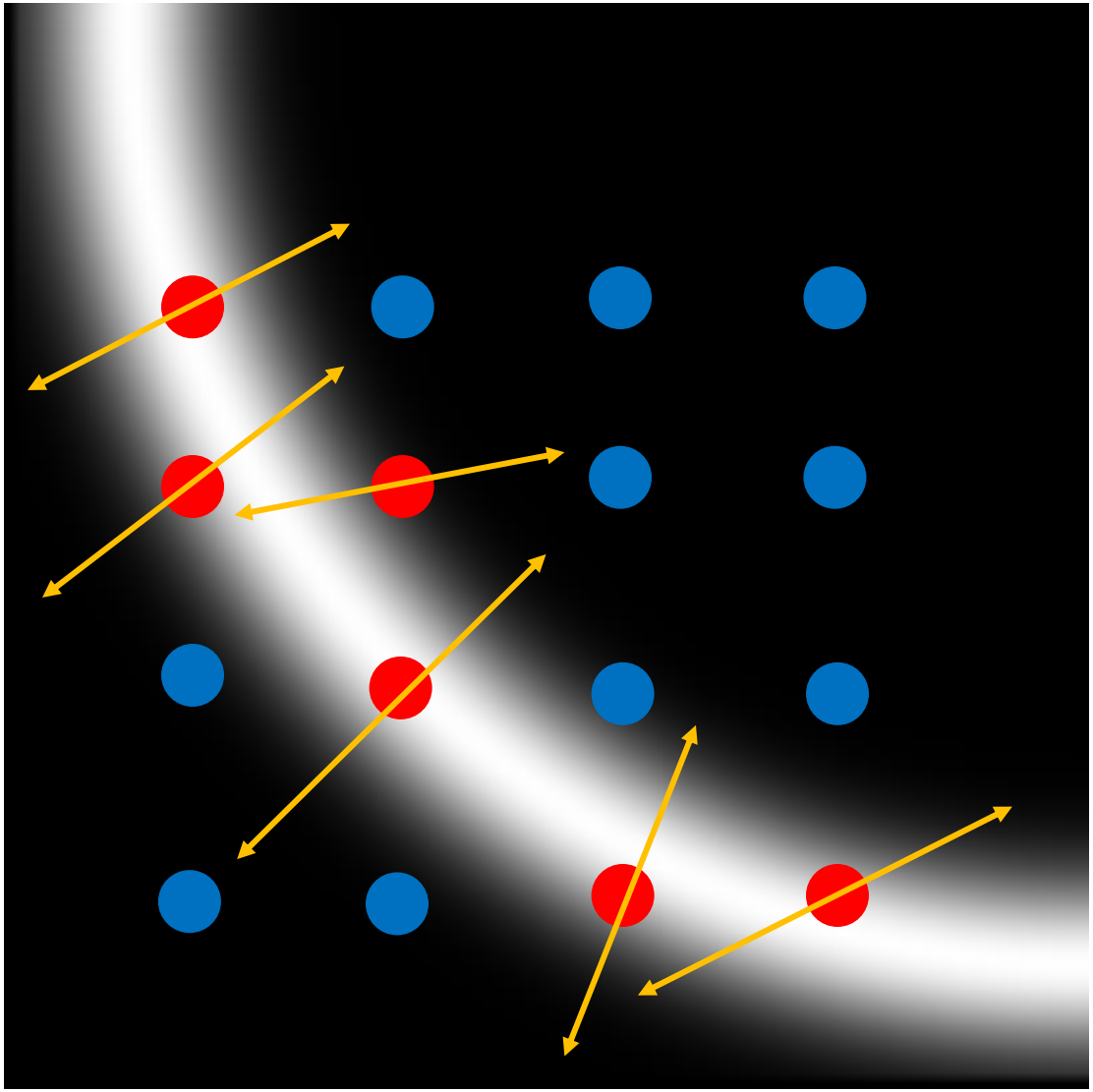


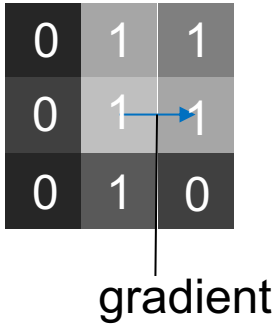
Discretized pixel locations



(Forsyth & Ponce)

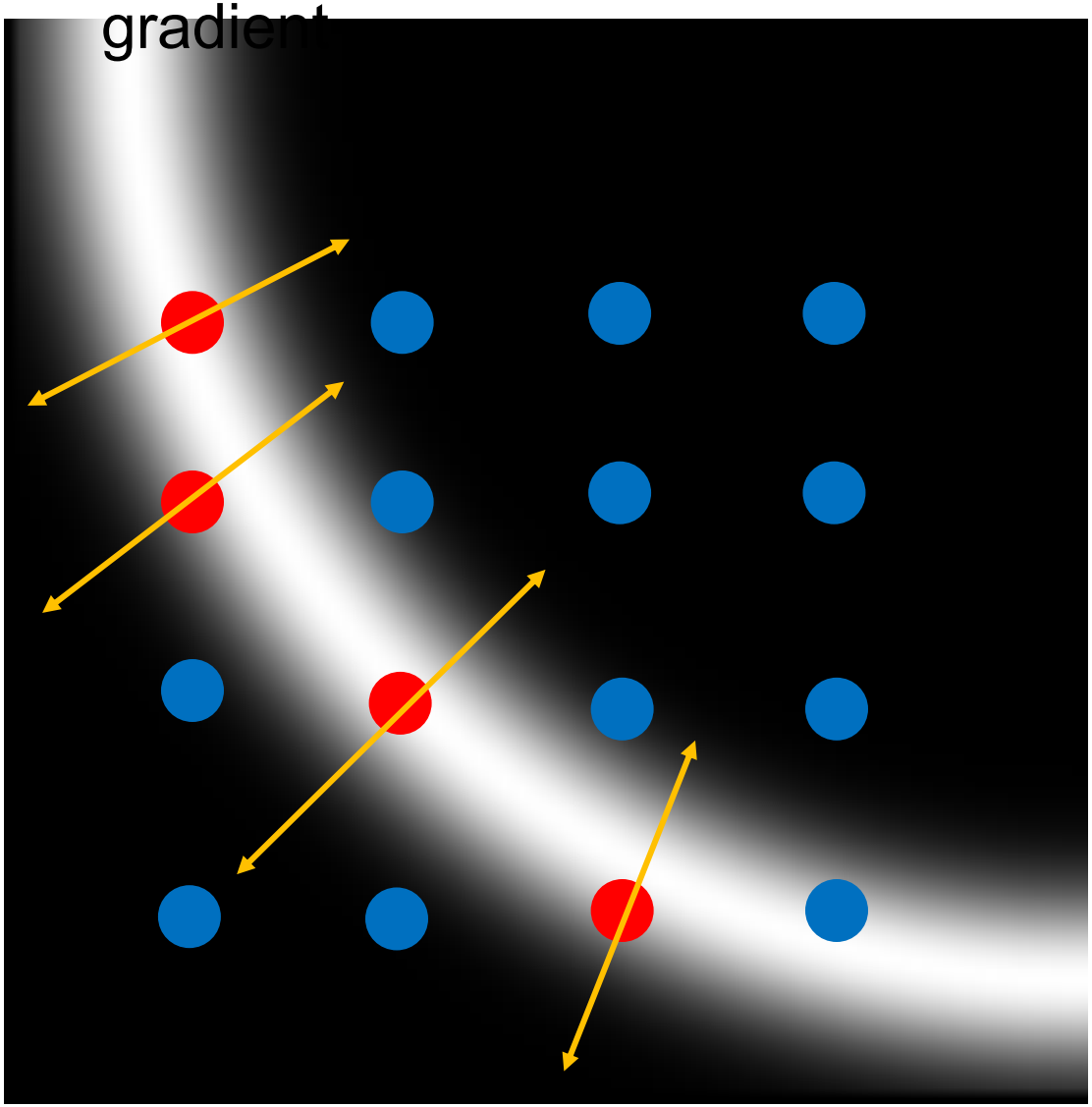
Thesholding

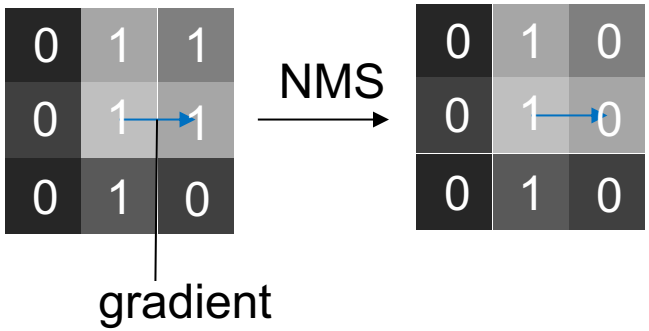




(Forsyth & Ponce)

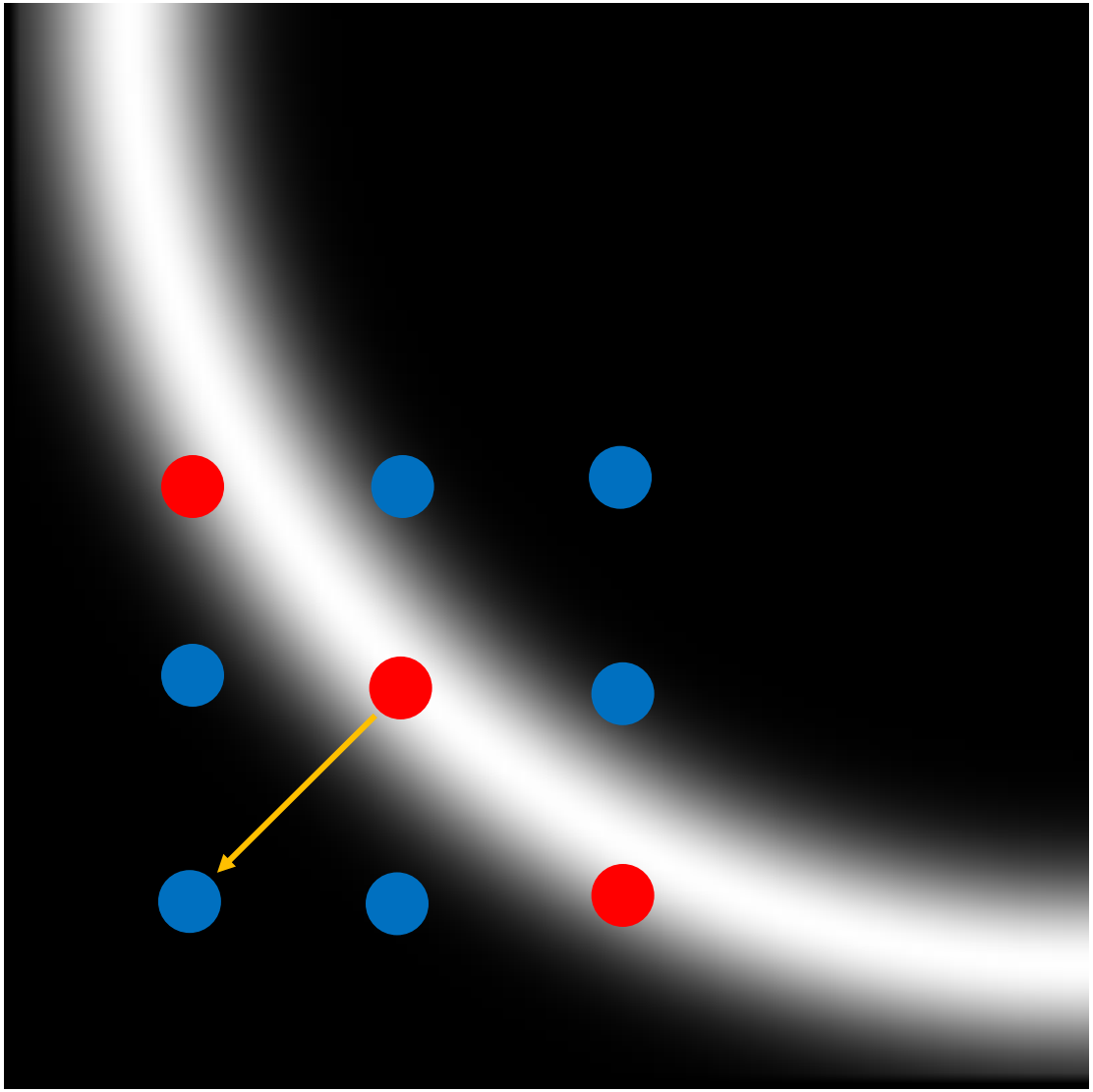
Non-maximum suppression along the line of the gradient

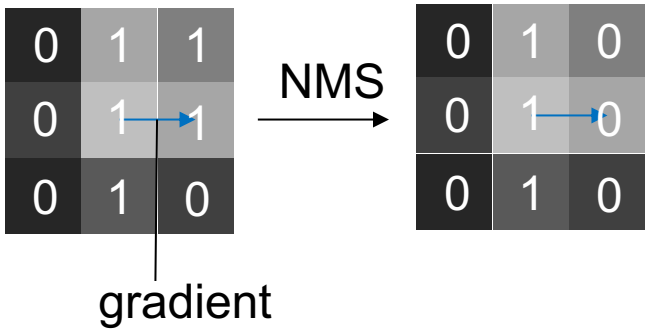




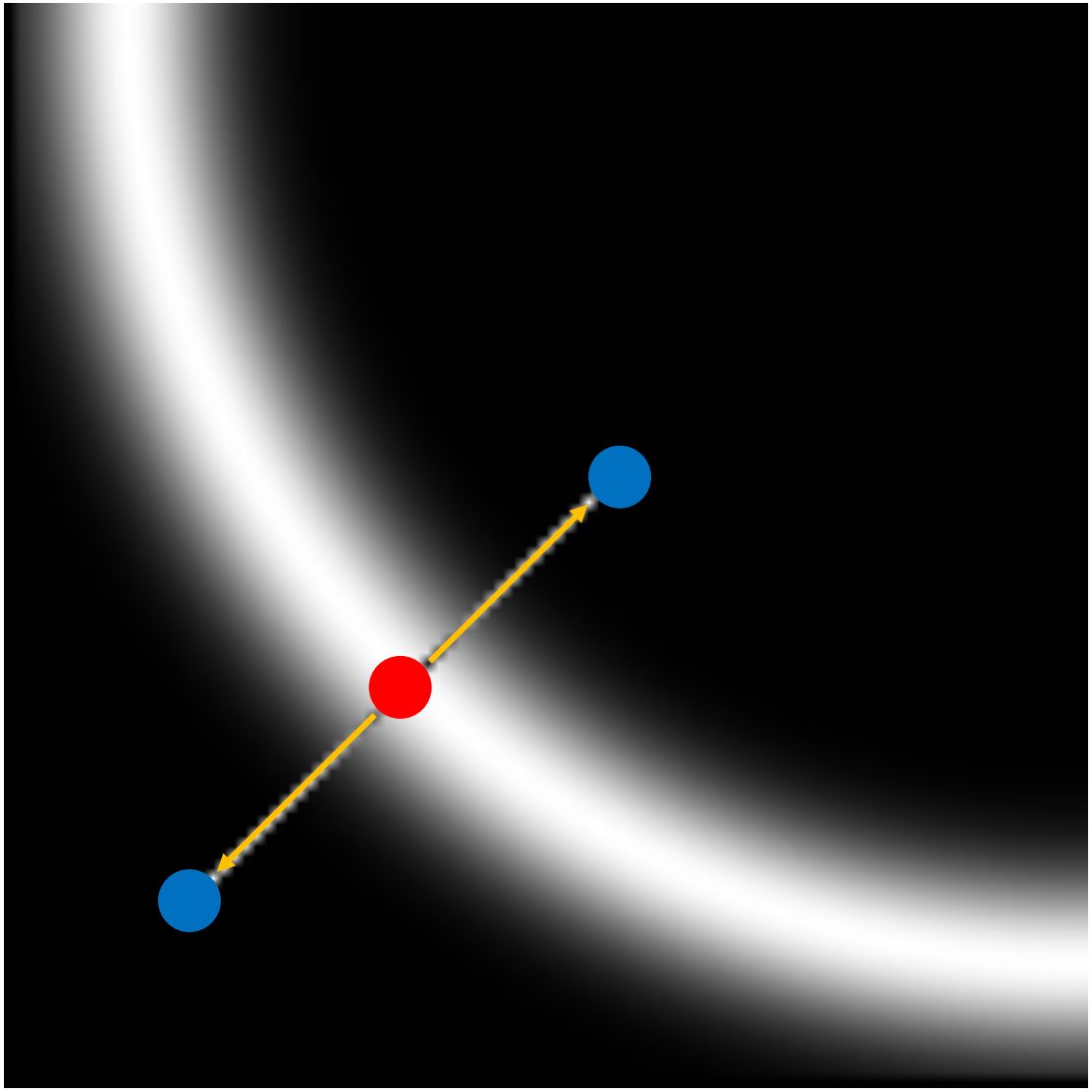
(Forsyth & Ponce)

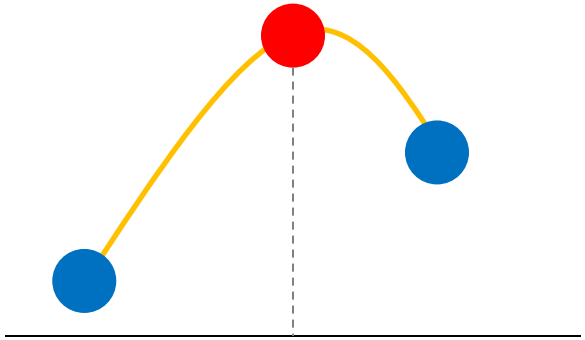
Gradient direction





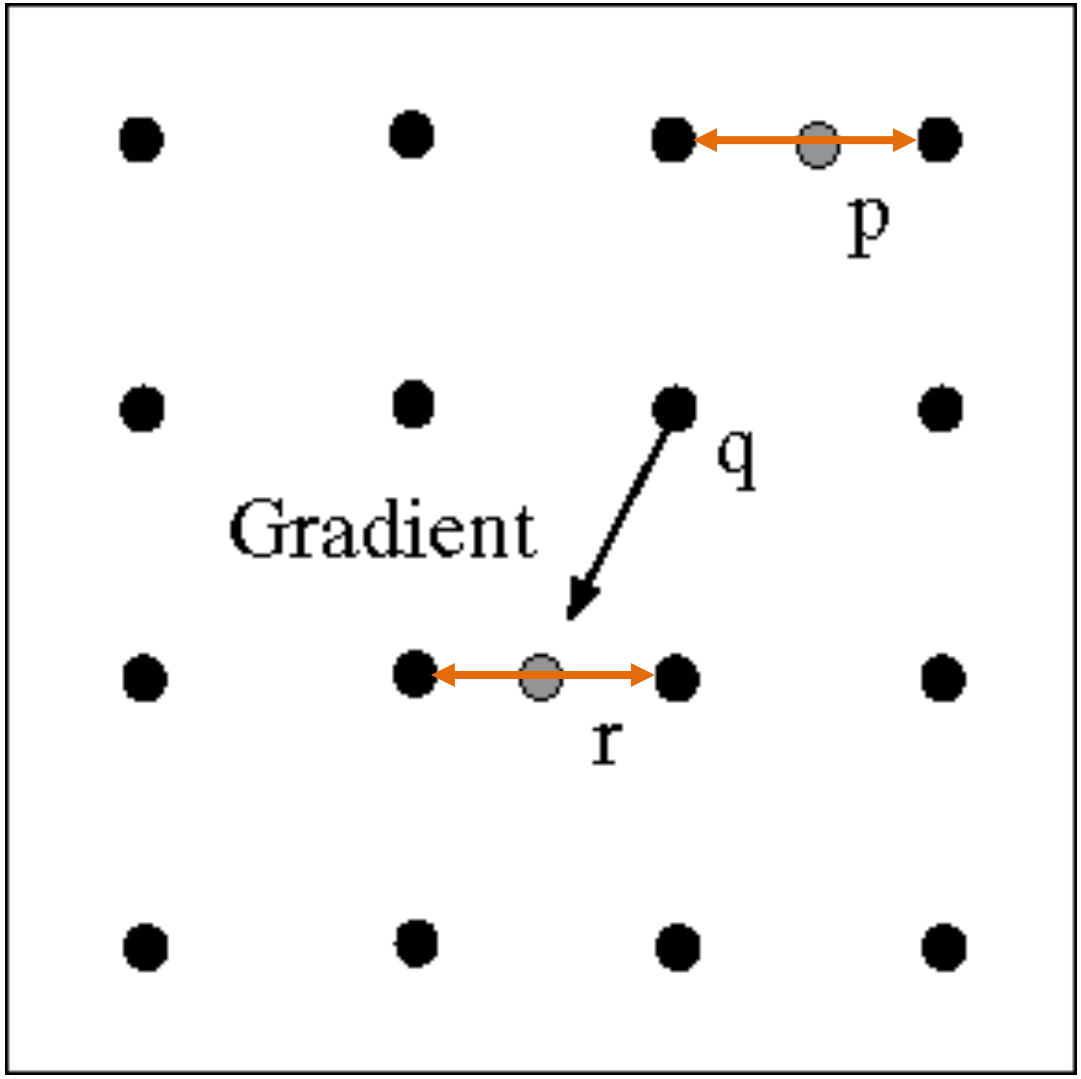
(Forsyth & Ponce)





Local maximum

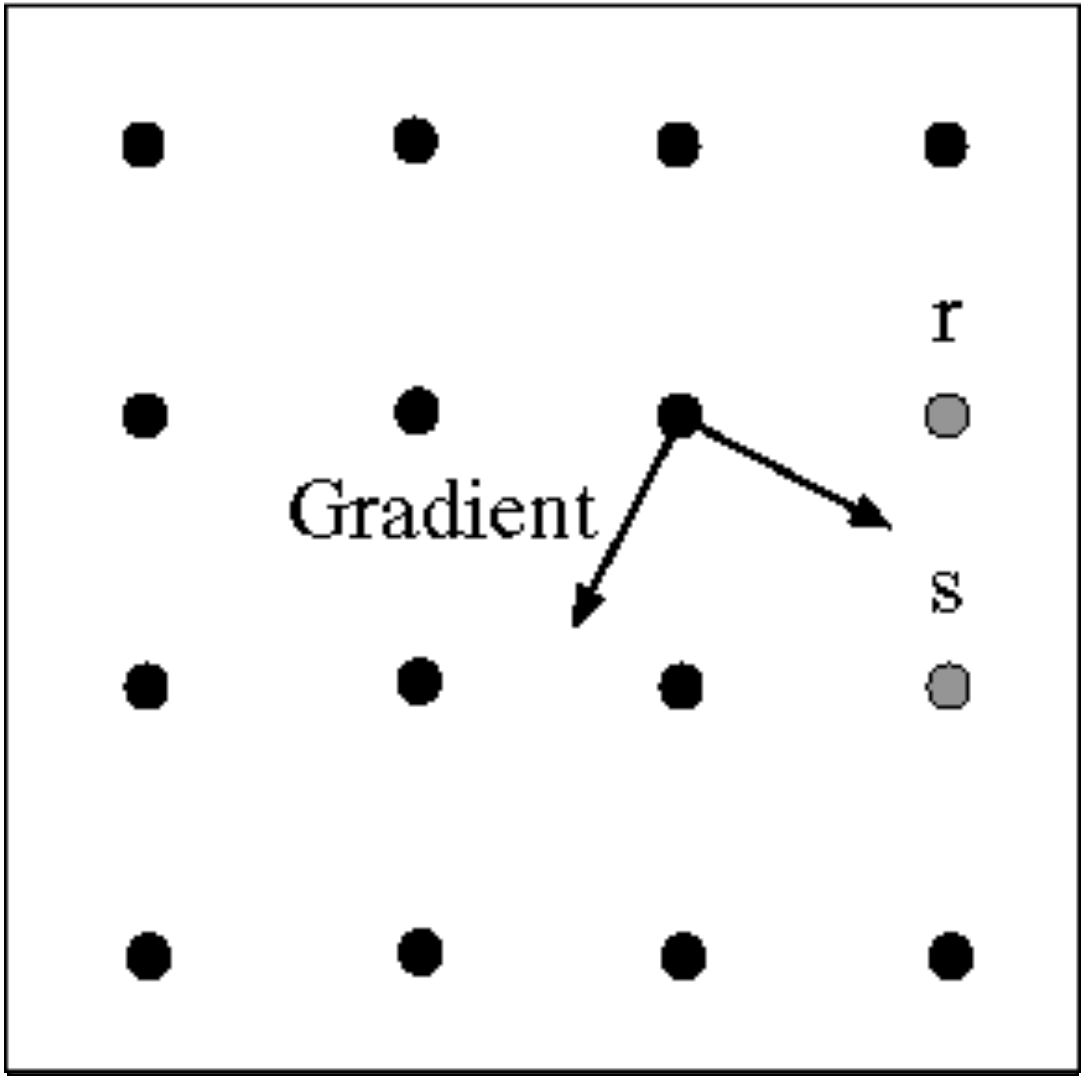
No intensity values at r and p:
Interpolate these intensities using neighbor pixels.

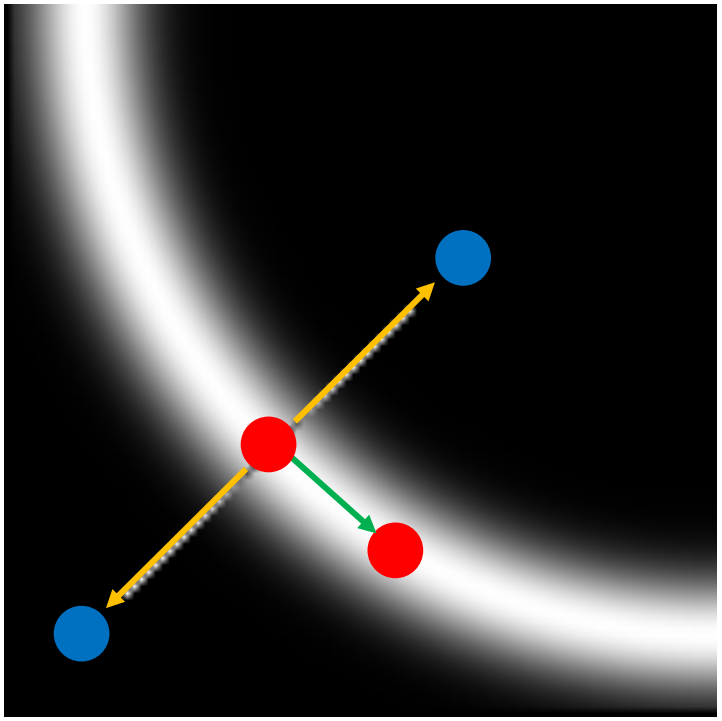


Where is next edge poin

Where is next edge point?

we construct the tangent to the edge curve (which is normal to the gradient at that point) and use this to predict the next points





Where is next edge point?

we construct the tangent to the edge curve (which is normal to the gradient at that point) and use this to predict the next points