Intensity Transformations and Spatial Filtering

March 11, 2005

Szabolcs Sergyán

sergyan.szabolcs@nik.bmf.hu

BMF NIK

Content

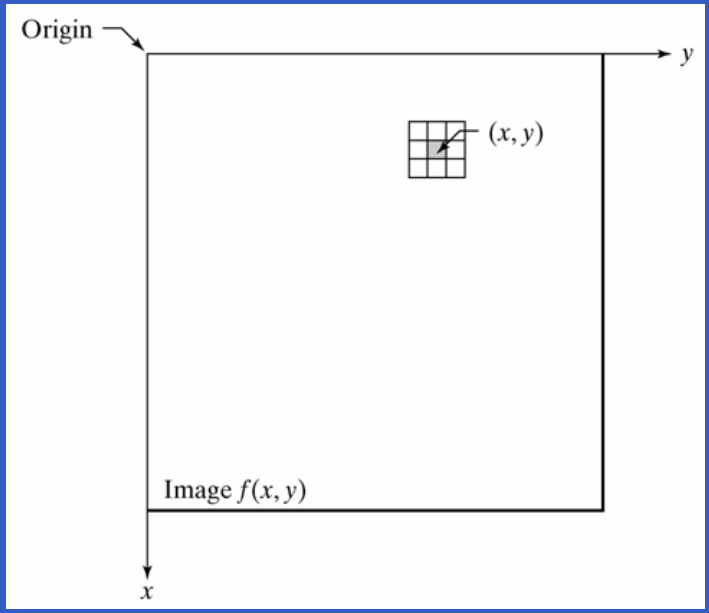
- Background
- Intensity Transformation Functions
- Histogram Processing and Function Plotting
- Spatial Filtering
- Image Processing Toolbox Standard Spatial Filters

Background

The *spatial domain* processes are denoted by the expression

$$g(x, y) = T\left[f(x, y)\right]$$

where $f(x, y)$ is the input image, $g(x, y)$ is the output (processed) image, and T is an operator on f, defined over a specified neighborhood about point (x, y).

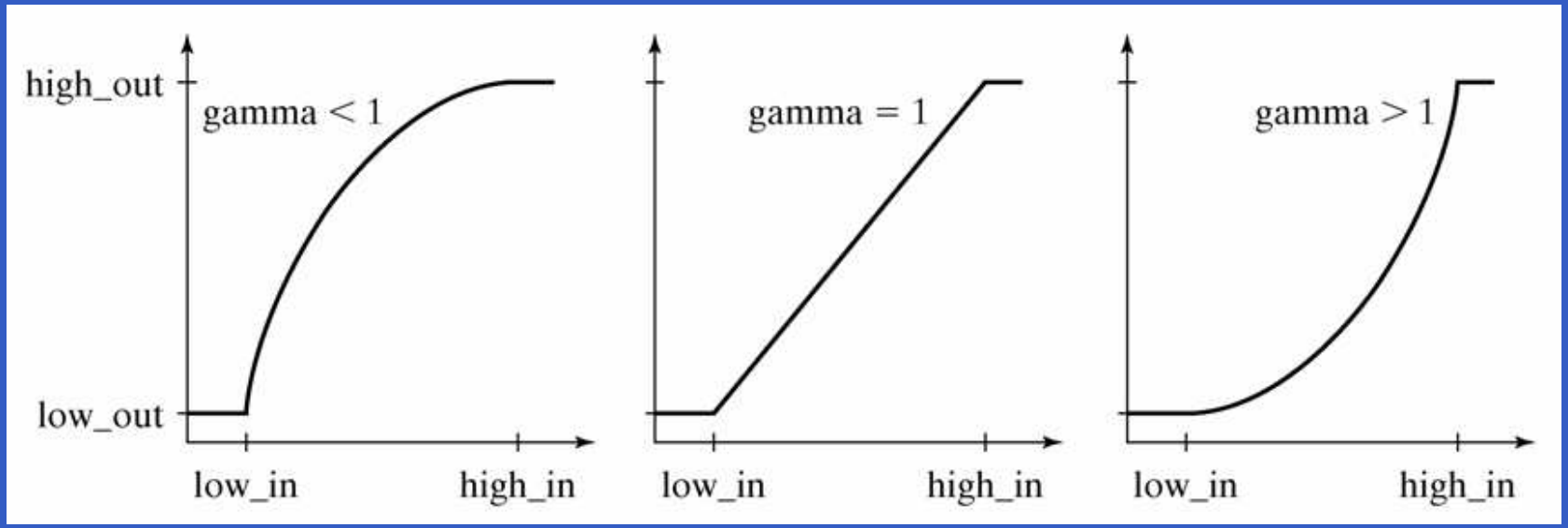


Intensity Transformation Functions

The simplest form of the transformation T is when the neighborhood is of size 1×1 (a single pixel). In this case, the value of g at (x, y) depends only on the intensity of f at that point, and T becomes an *intensity* or *gray-level* transformation function.

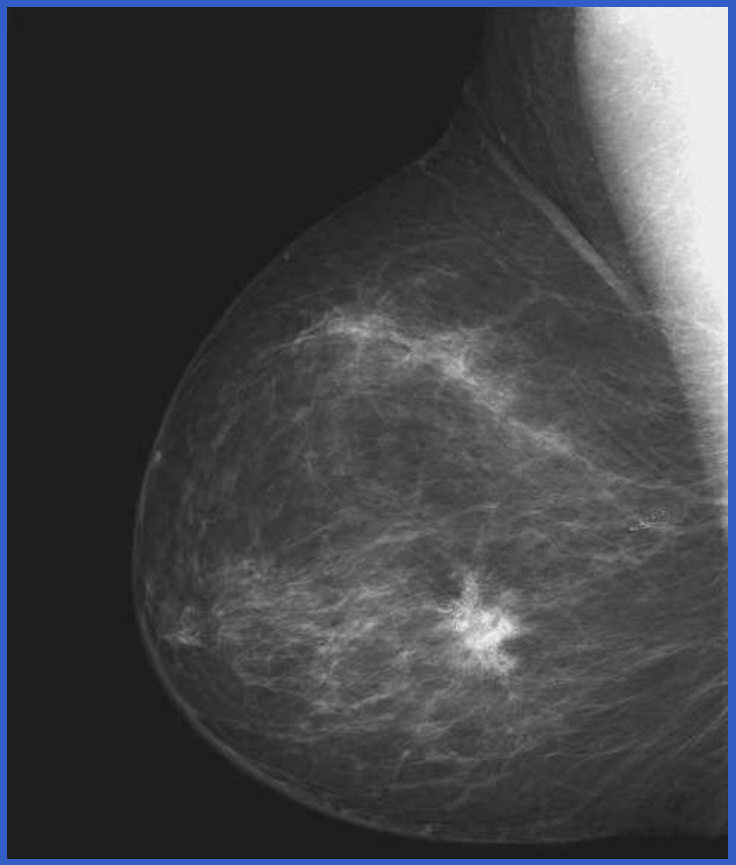
Function imadjust

```
g=imadjust(f,[low_in high_in],...
           [low_out high_out],gamma)
```



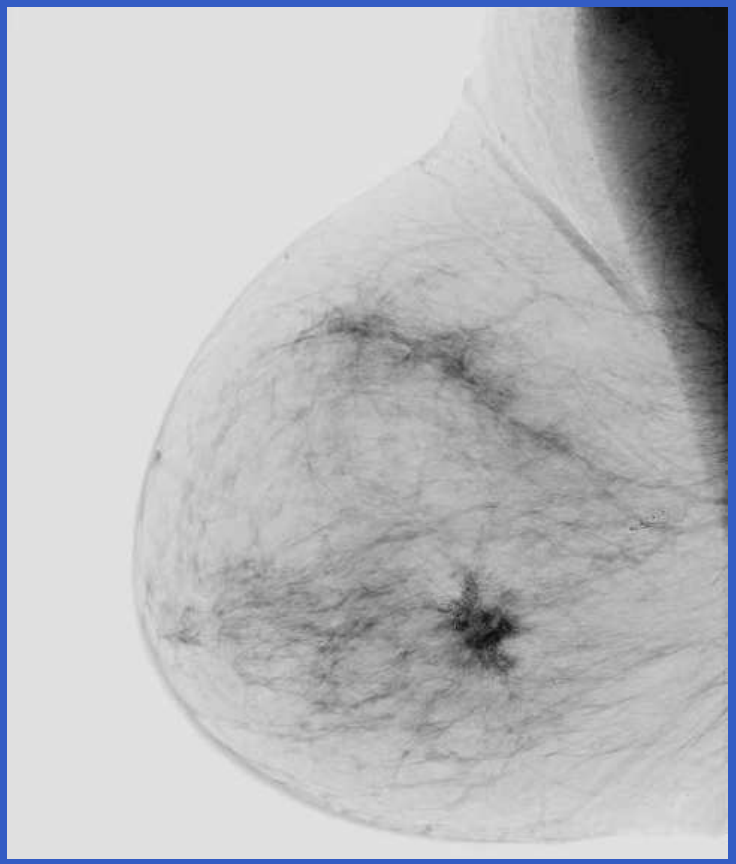
Function imadjust

```
>> f=imread('breast.tif');
```



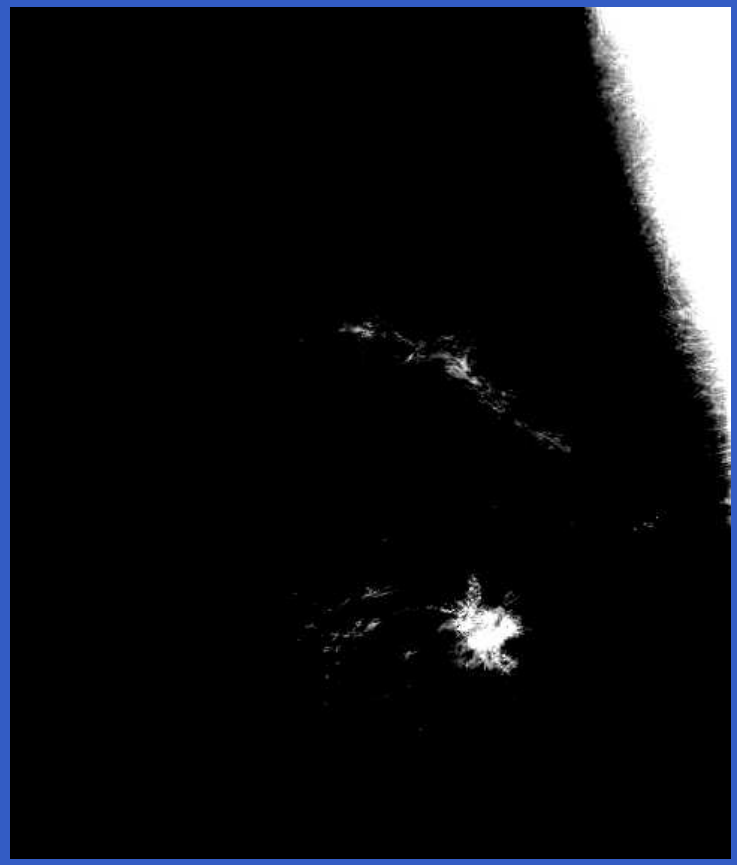
Function imadjust

```
>> g1=imadjust(f,[0 1],[1 0]);
```



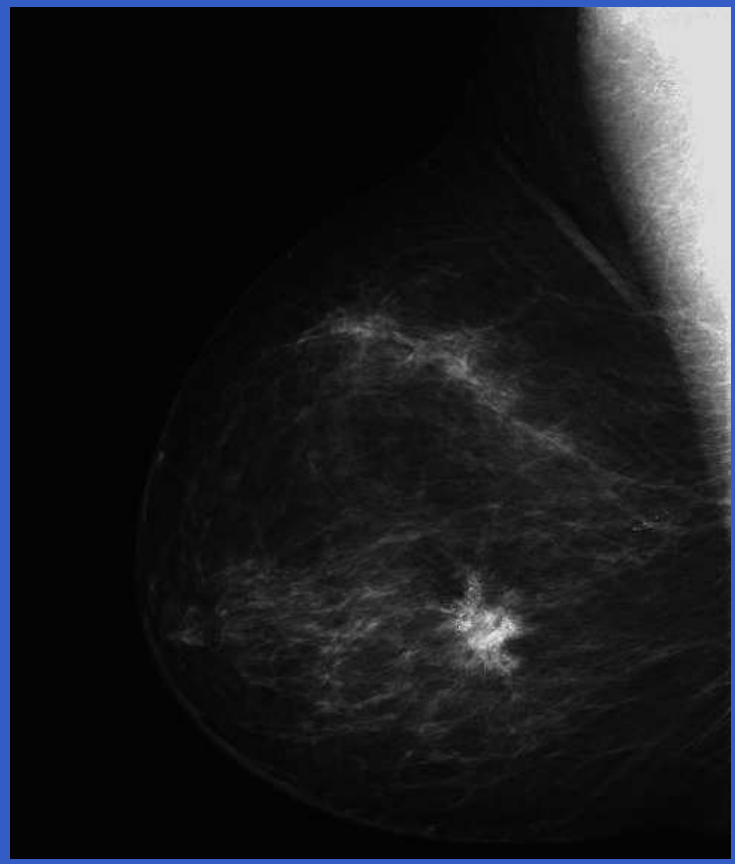
Function imadjust

```
>> g2=imadjust(f,[0.5 0.75],[0 1]);
```



Function imadjust

```
>> g3=imadjust(f,[],[],2);
```



Histogram Processing and Function Plotting

- Generating and Plotting Image Histograms
- Histogram Equalization
- Histogram Matching (Specification)

Generating and Plotting Image Histograms

The histogram of a digital image with L total possible intensity levels in the range $[0, G]$ is defined as the discrete function

$$h(r_k) = n_k$$

where r_k is the kth intensity level in the interval $[0, G]$ and n_k is the number of pixels in the image whose intensity level is r_k. The value of G is 255 for images of class `uint8`, 65535 for images of class `uint16`, and 1.0 for images of class `double`. Keep in mind that indices in MATLAB cannot be 0, so r_1 corresponds to intensity level 0, r_2 corresponds to intensity level 1, and so on, with r_L corresponding to level G. Note also that $G = L - 1$ for images of class `uint8` and `uint16`.

Generating and Plotting Image Histograms

Often, it is useful to work with *normalized* histograms, obtained simply by dividing all elements of $h(r_k)$ by the total number of pixels in the image, which we denote by n:

$$p(r_k) = \frac{h(r_k)}{n} = \frac{n_k}{n}$$

for $k = 1, 2, \ldots, L$.

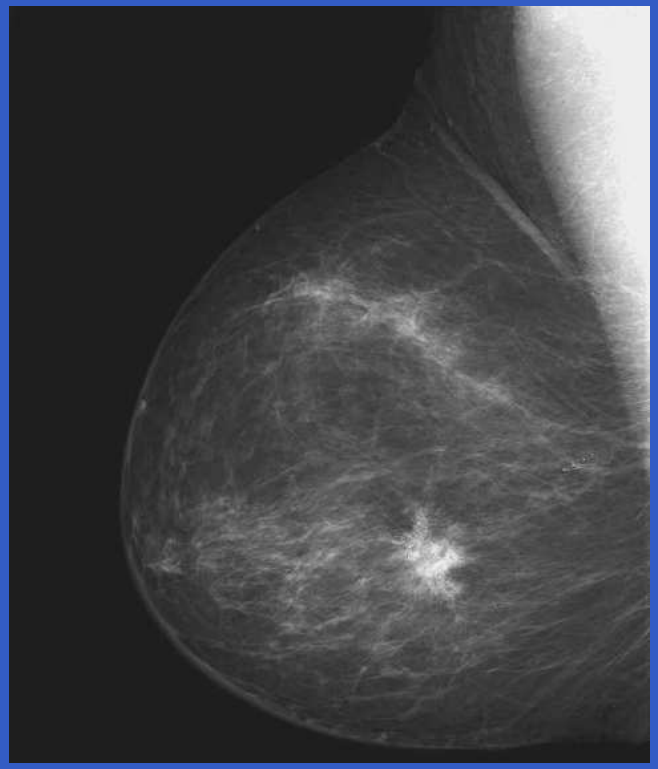
Generating and Plotting Image Histograms

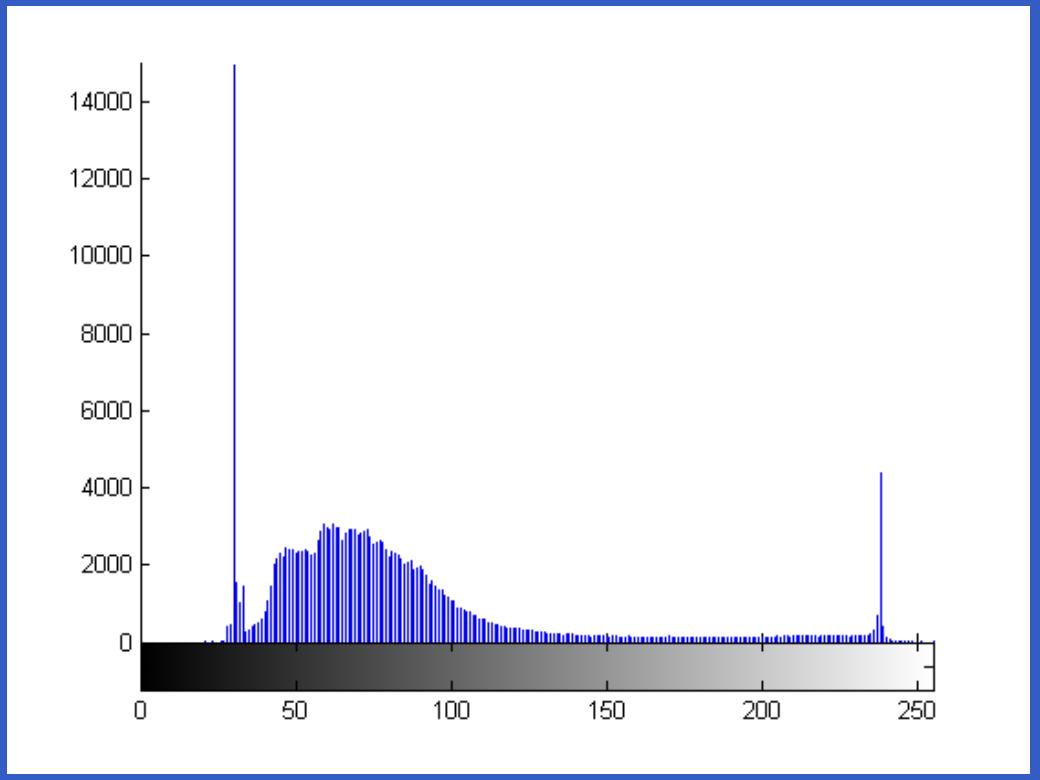
`h=imhist(f,b)`

where `f` is the input image, h is its histogram, $h(r_k)$, and b is the number of bins used in forming the histogram (if b is not included in the argument, `b=256` is used by default). A bin is simply a subdivision of the intensity scale. For example, if we are working with `uint8` images and we let b=2, then the intensity scale is subdivided into two ranges: 0 to 127 and 128 to 255. The resulting histogram will have two values: `h(1)` equal to the number of pixels in the image with values in the interval $[0, 127]$, and `h(2)` equal to the number of pixels with values in the interval $[128, 255]$.

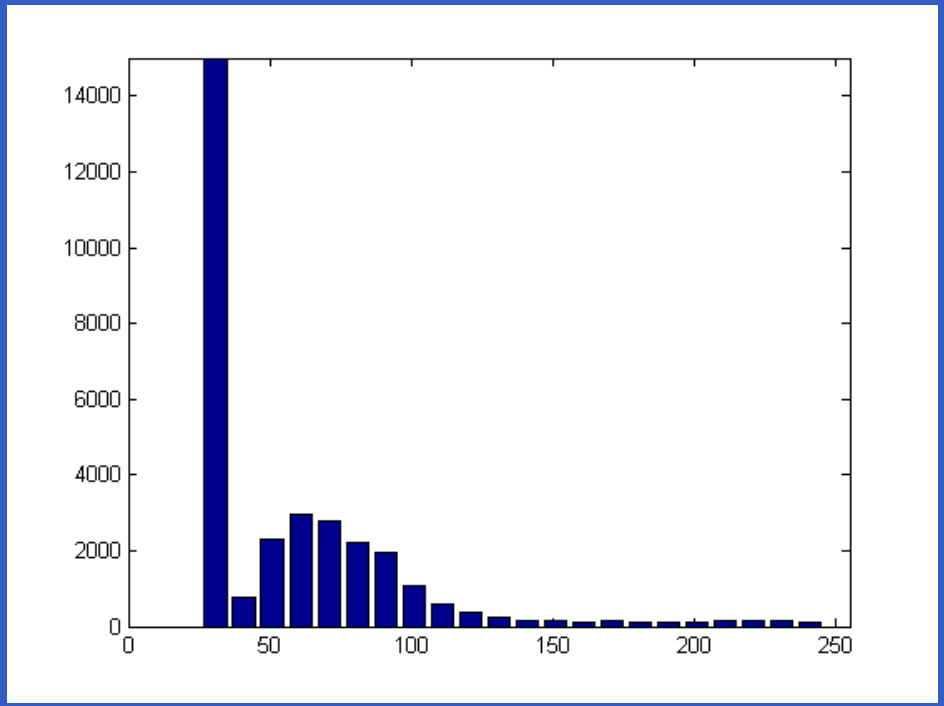
Generating and Plotting Image Histograms

```
>> f=imread('breast.tif');
>> imshow(f), imhist(f)
```





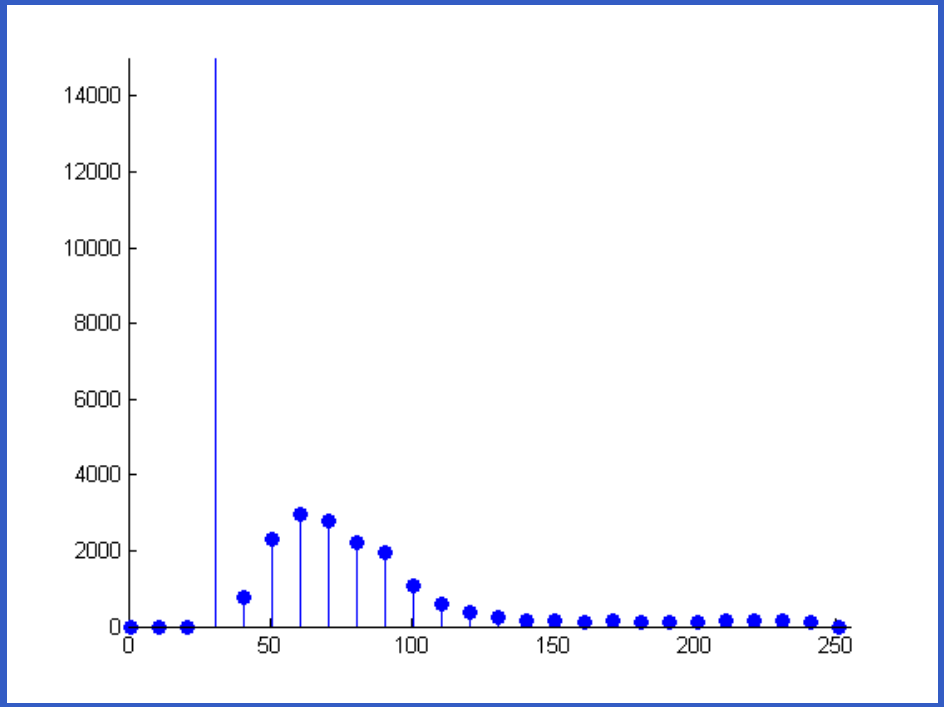
Generating and Plotting Image Histograms



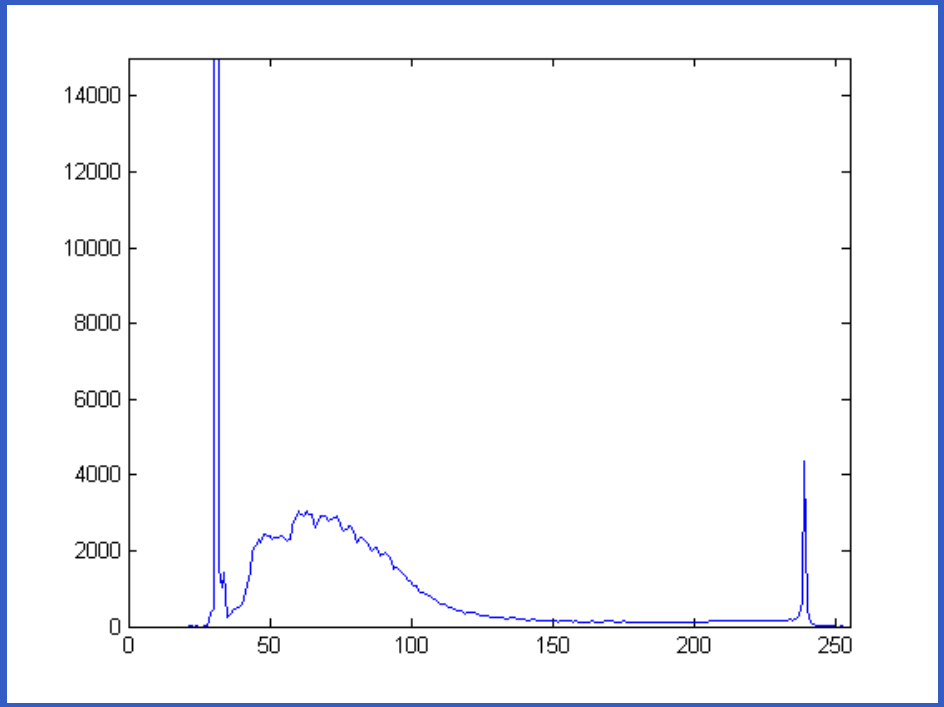

```
>> h=imhist(f);
>> h1=h(1:10:256);
>> horz=1:10:256;
>> bar(horz,h1)
>> axis([0 255 0 15000])
>> set(gca,'xtick',0:50:255)
>> set(gca,'ytick',0:2000:15000)
```

Generating and Plotting Image Histograms

```
>> h=imhist(f);
>> h1=h(1:10:256);
>> horz=1:10:256;
>> stem(horz,h1,'fill')
>> axis([0 255 0 15000])
>> set(gca,'xtick',0:50:255)
>> set(gca,'ytick',0:2000:15000)
```



Generating and Plotting Image Histograms




```
>> h=imhist(f);
>> plot(h)
>> axis([0 255 0 15000])
>> set(gca,'xtick',0:50:255)
>> set(gca,'ytick',0:2000:15000)
```

Some Useful Plotting Function

- `plot(horz,v,'color_linestyle_marker')`
- `bar(horz,v,width)`
- `stem(horz,v,'color_linestyle_marker','fill')`
- `axis([horzmin horzmax vertmin vertmax])`
- `xlabel('text string','fontsize',size)`
- `ylabel('text string','fontsize',size)`
- `text(xloc,yloc,'text string','fontsize',size)`
- `title('titlestring')`

Some Useful Plotting Function

Symbol	Color	Symbol	Line Style	Symbol	Marker
k	Black	-	Solid	+	Plus sign
w	White	--	Dashed	o	Circle
r	Red	:	Dotted	*	Asterisk
g	Green	-.	Dash-dot	.	Point
b	Blue	none	No line	x	Cross
c	Cyan			s	Square
y	Yellow			d	Diamond
m	Magenta			none	No marker

Histogram Equalization

$$s_k = \sum_{j=0}^{k} \frac{n_j}{n} \quad k = 0, 1, 2, \ldots, L-1$$

where n is the total number of pixels in the image, n_k is the number of pixels that have gray level r_k, and L is the total number of possible gray levels in the image. A processed image is obtained by mapping each pixel with level r_k in the input image into a corresponding pixel with level s_k in the output image.

Histogram Equalization

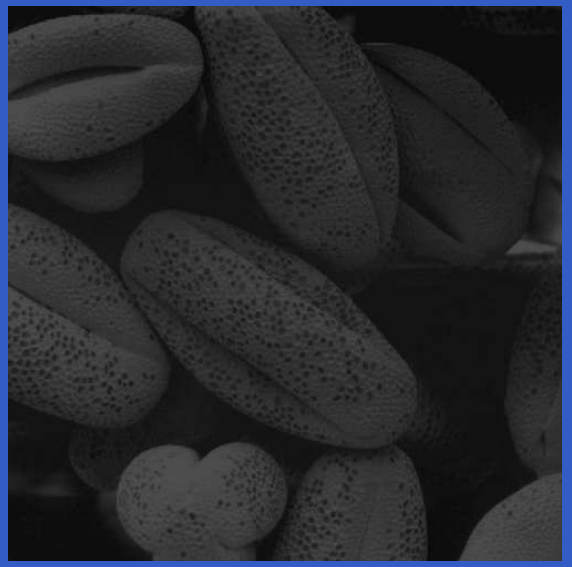
```
g=histeq(f,nlev)
```

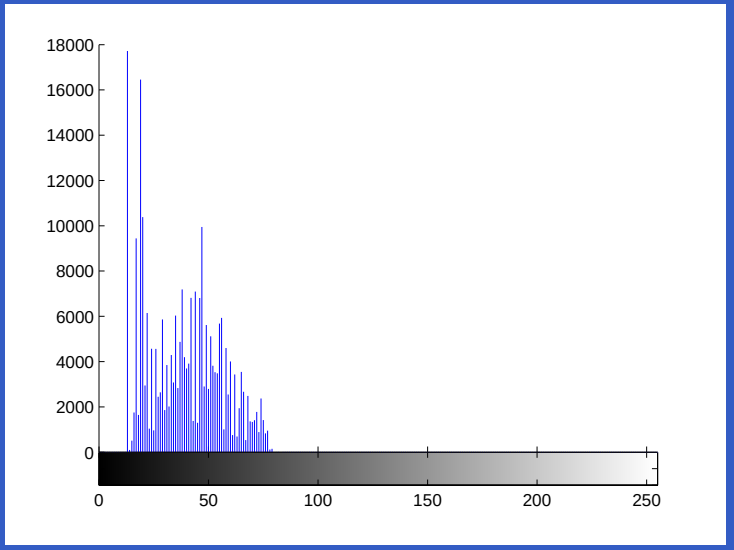
where `f` is the input image and `nlev` is the number of intensity levels specified for the output image. If `nlev` is equal to L (the total number of possible levels in the input image), then `histeq` implements the transformation function (described on the previous slide), directly. If `nlev` is less than L, then `histeq` attempts to distribute the levels so that they will approximate a flat histogram. Unlike `imhist`, the default value in `histeq` is `nlev=64`.

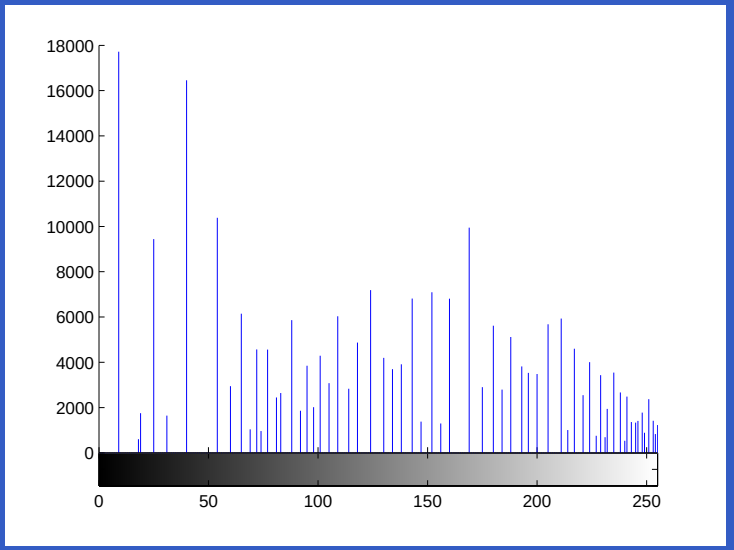
Histogram Equalization

```
>> f=imread('pollen.tif');
>> imshow(f)
>> figure, imhist(f)
>> ylim('auto')
>> g=histeq(f,256);
>> figure, imshow(g)
>> figure, imhist(g)
>> ylim('auto')
```

Histogram Equalization



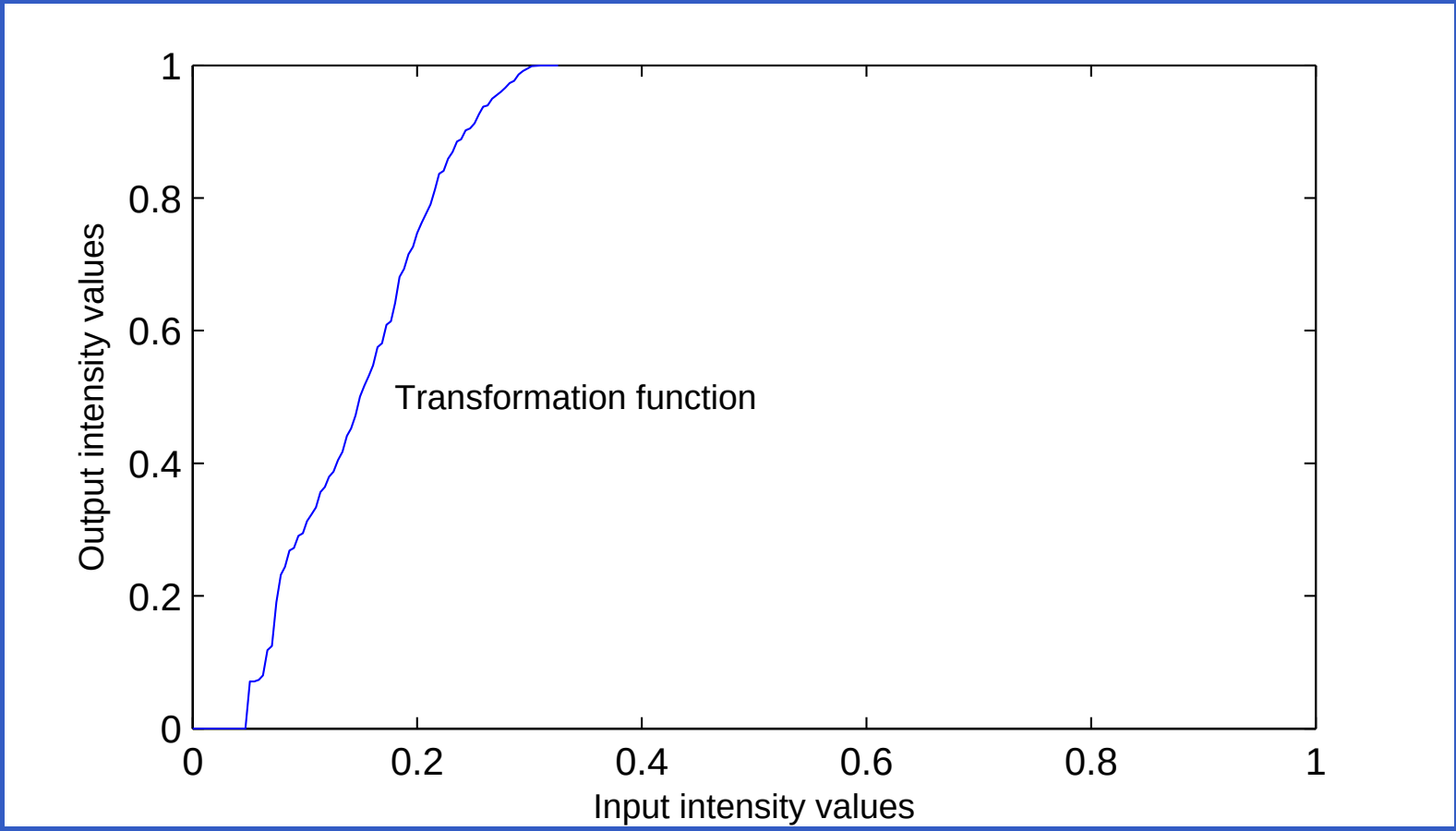




Histogram Equalization

```
>> hnorm=imhist(f)./numel(f);
>> %Cummulative distribution function:
>> cdf=cumsum(hnorm);
>> x=linspace(0,1,256);
>> plot(x,cdf)
>> axis([0 1 0 1])
>> set(gca,'xtick',0:.2:1)
>> set(gca,'ytick',0:.2:1)
>> xlabel('Input intensity values','fontsize',9)
>> ylabel('Output intensity values','fontsize',9)
>> %Specify text in the body of the graph:
>> text(0.18,0.5,'Transformation function',...
>>      'fontsize',9)
```

Histogram Equalization



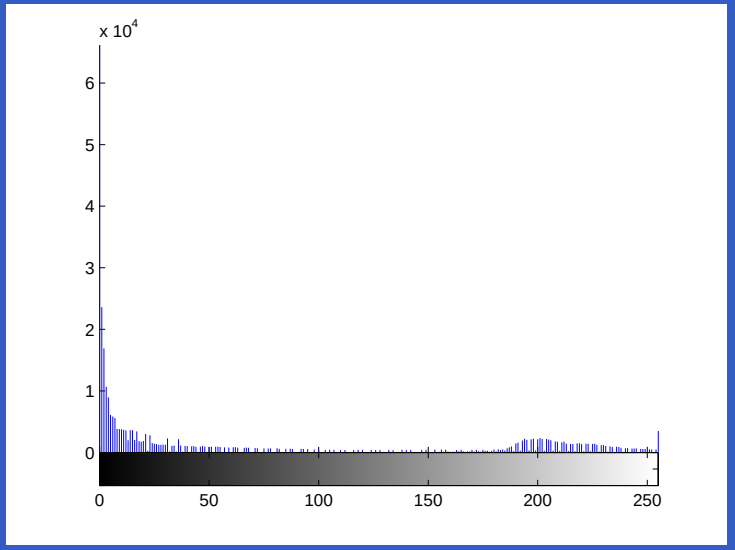
Histogram Matching

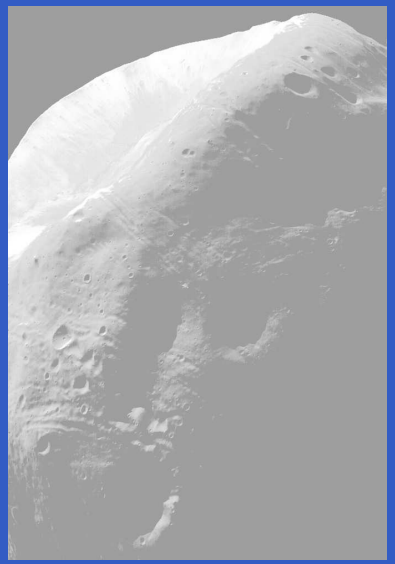
It is useful in some applications to be able to specify the shape of the histogram that we wish the processed image to have. The method used to generate a processed image that has a specified histogram is called *histogram matching*.

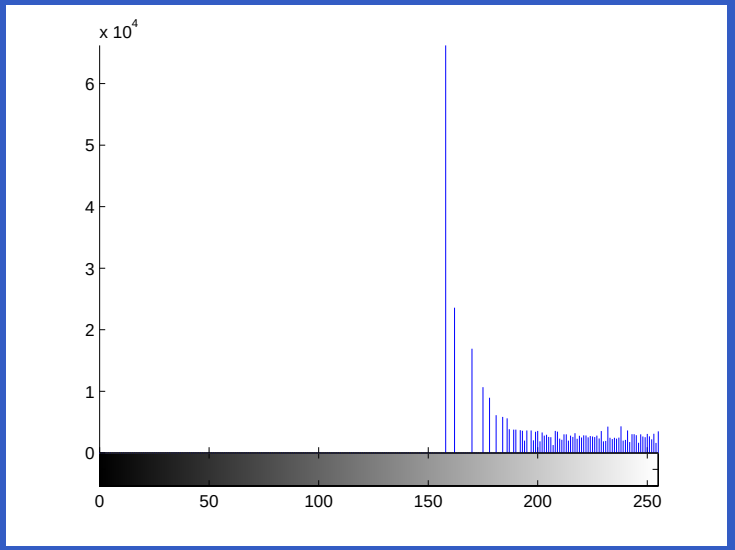
```
g=histeq(f,hspec)
```

where `f` is the input image, `hspec` is the specified histogram (a row vector of specified values), and `g` is the input image, whose histogram approximates the specified histogram, `hspec`.

Histogram Matching







Histogram Matching

```
function p=twomodegauss(m1,sig1,m2,sig2,A1,A2,k)
%TWOMODEGAUSS Generates a bimodal Gaussian function.
%   P=TWOMODEGAUSS(M1,SIG1,M2,SIG2,A1,A2,K) generates a bimodal,
%   Gaussian-like function in the interval [0,1]. P is a
%   256-element vector normalized so that SUM(P) equals 1. The
%   mean and standard deviation of the modes are (M1,SIG1) and
%   (M2,SIG2), respectively. A1 and A2 are the amplitude values
%   of the two modes. Since the output is normalized, only the
%   relative values of A1 and A2 are important. K is an offset
%   values that raises the "floor" of the function. A good set
%   of values to try is M1=0.15, SIG1=0.05, M2=0.75, SIG2=0.05,
%   A1=1, A2=0.07, and K=0.002.
```

Histogram Matching

```
c1=A1*(1/((2*pi)^0.5)*sig1);
k1=2*(sig1^2);
c2=A2*(1/((2*pi)^0.5)*sig2);
k2=2*(sig2^2);
z=linspace(0,1,256);

p=k+c1*exp(-((z-m1).^2)./k1)+...
    c2*exp(-((z-m2).^2)./k2);
p=p./sum(p(:));
```

Histogram Matching

```
function p=manualhist
%MANUALHIST Generates a bimodal histogram interactively.
%   P=MANUALHIST generates a bimodal histogram using
%   TWOMODEGAUSS(m1,sig1,m2,sig2,A1,A2,k). m1 and m2 are the
%   means of the two modes and must be in the range [0,1]. sig1
%   and sig2 are the standard deviations of the two modes. A1
%   and A2 are amplitude values, and k is an offset value that
%   raises the "floor" of histogram. The number of elements in
%   the histogram vector P is 256 and sum(P) is normalized to 1.
%   MANUALHIST repeatedly prompts for the parameters and plots
%   the resulting histogram until the user types an 'x' to quit,
%   and then it returns the last histogram computed.
%
%   A good set of starting values is: (0.15, 0.05, 0.75, 0.05, 1,
%   0.07, 0.002).
```

Histogram Matching

```
%Initialize.
repeats=true;
quitnow='x';

%Compute a default histogram in case the user quits before
%estimating at least one histogram.
p=twomodegauss(0.15,0.05,0.75,0.05,1,0.07,0.002);

%Cycle until x is input.
while repeats
    s=input('Enter m1, sig1, m2, sig2, A1, A2, k OR x to quit:','s');
    if s==quitnow
        break
    end
```

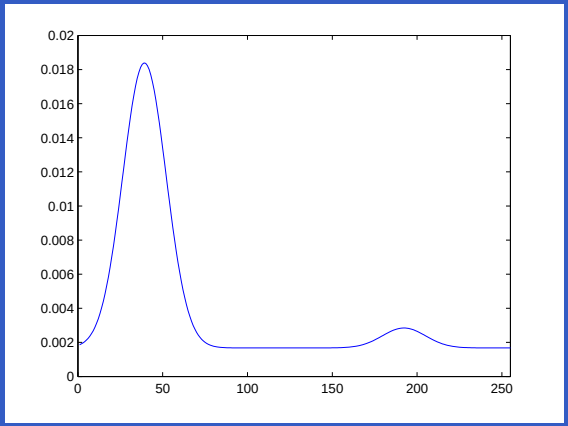
Histogram Matching

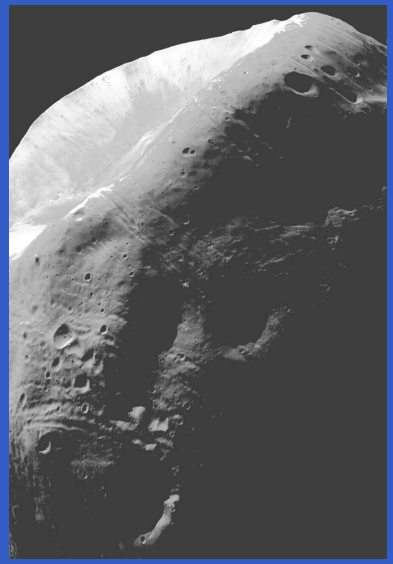
```
    %Convert the input string to a vector of numerical values and
    %verify the number of inputs.
    v=str2num(s);
    if numel(v)~=7
        disp('Incorrect number of inputs.')
        continue
    end

    p=twomodegauss(v(1),v(2),v(3),v(4),v(5),v(6),v(7));
    %Start a new figure and scale the axes. Specifying only xlim
    %leaves ylim on auto.
    figure, plot(p)
    xlim([0 255])
end
```

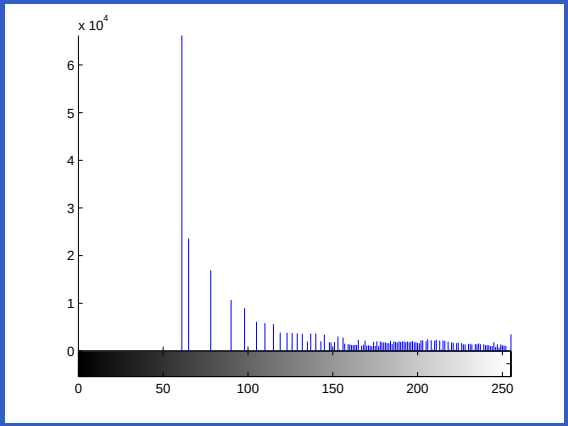
Histogram Matching

```
>> f=imread('moon_phobos.tif');
>> p=manualhist;
Enter m1, sig1, m2, sig2, A1, A2, k OR x to quit:x
>> g=histeq(f,p);
>> imshow(g)
>> figure, imhist(g)
```

Histogram Matching







Spatial Filtering

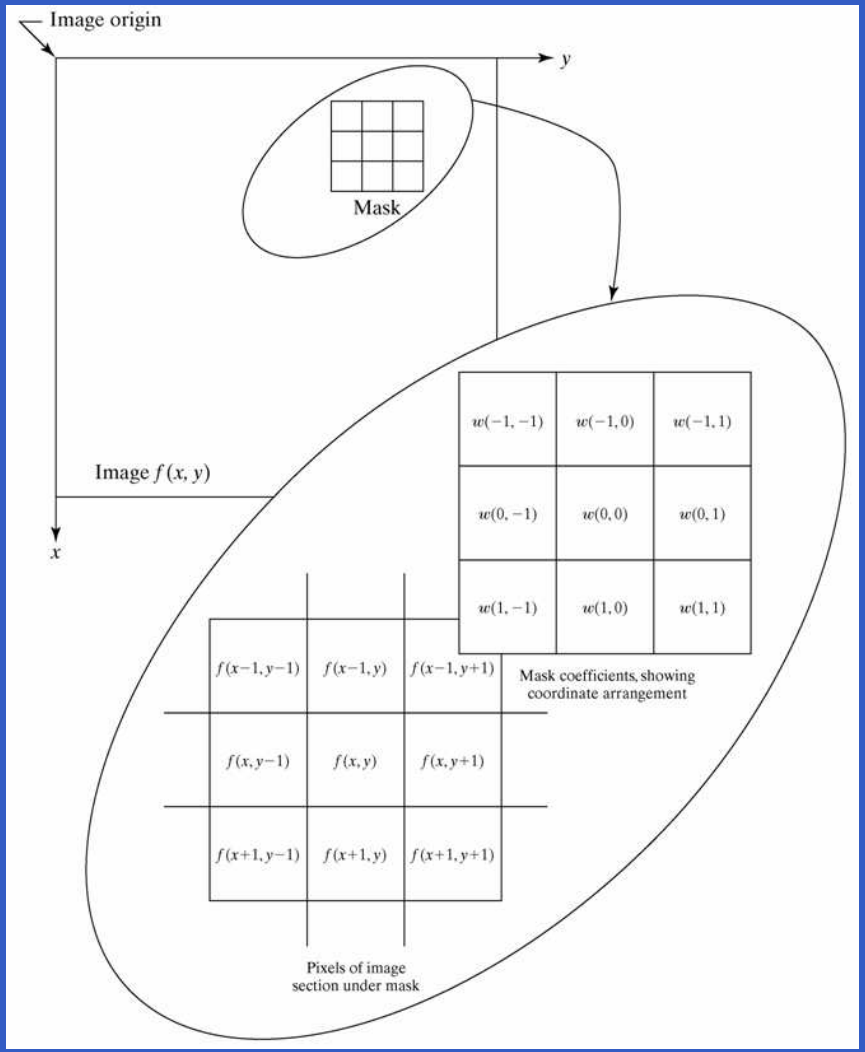
Neighborhood processing consists of

- defining a center point, (x, y);
- performing an operation that involves only the pixels in a predefined neighborhood about that center point;
- letting the result of that operation be the "response" of the process at *that* point; and
- repeating the process for every point in the image.

If the computations performed on the pixels of the neighborhoods are linear, the operation is called *linear spatial filtering*; otherwise it is called *nonlinear spatial filtering*.

Linear Spatial Filtering

The mechanics of linear spatial filtering:

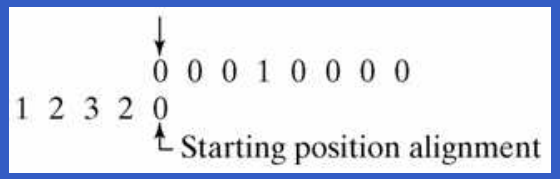


Linear Spatial Filtering

The process consists simply of moving the center of the filter mask w from point to point in an image f. At each point (x, y), the response of the filter at that point is the sum of products of the filter coefficients and the corresponding neighborhood pixels in the area spanned by the filter mask. For a mask of size $m \times n$, we assume typically that $m = 2a + 1$ and $n = 2b + 1$, where a and b are nonnegative integers.

There are two closely related concepts that must be understood clearly when performing linear spatial filtering. *Correlation* is the process of passing the mask w by the image array f in the manner described earlier. Mechanically, *convolution* is the same process, except that w is rotated by $180°$ prior to passing it by f.

Linear Spatial Filtering



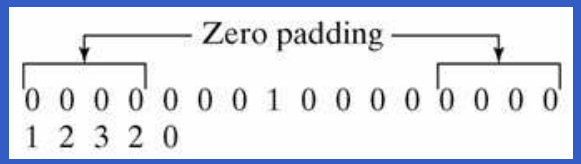
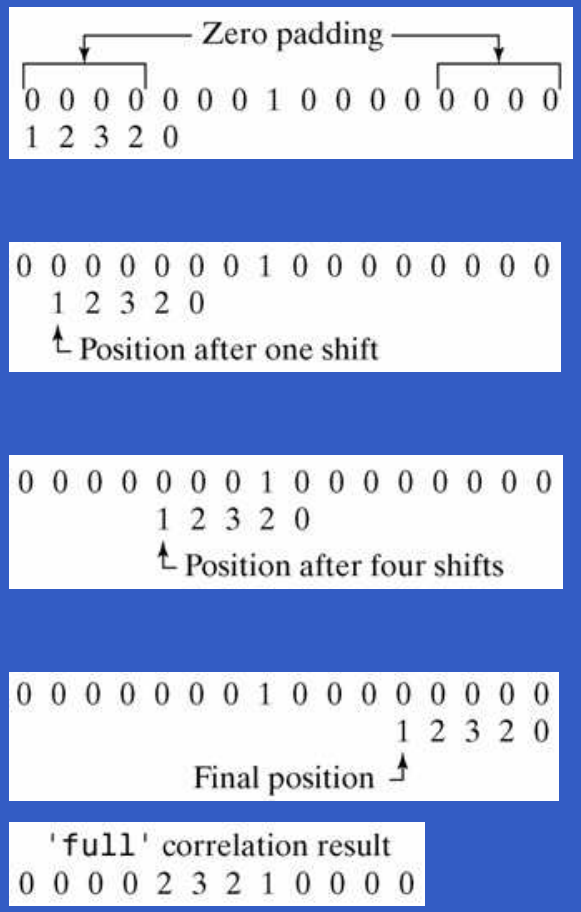


Figure shows a one-dimensional function, f, and a mask w.

To perform the correlation of the two functions, we move w so that its rightmost point coincides with the origin of f.

There are points between the two functions that do not overlap. The most common way to handle this problem is to pad f with as many 0s as are necessary to guarantee that there will always be corresponding points for the full excursion of w past f.

Linear Spatial Filtering



The first value of correlation is the sum of products of the two functions in the position shown in the figure.

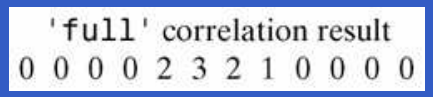
Next, we move w one location to the right and repeat the process.

After four shifts, we encounter the first nonzero value of the correlation, which is $2 \cdot 1 = 2$.

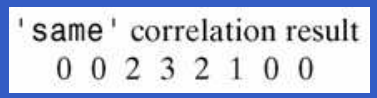
The ending geometry is shown in this figure.

If we proceed in this manner until w moves completely past f we would get this result.

Linear Spatial Filtering



The label `'full'` is a flag used by the IPT[a] to indicate correlation using a padded image and computed in the manner just described.

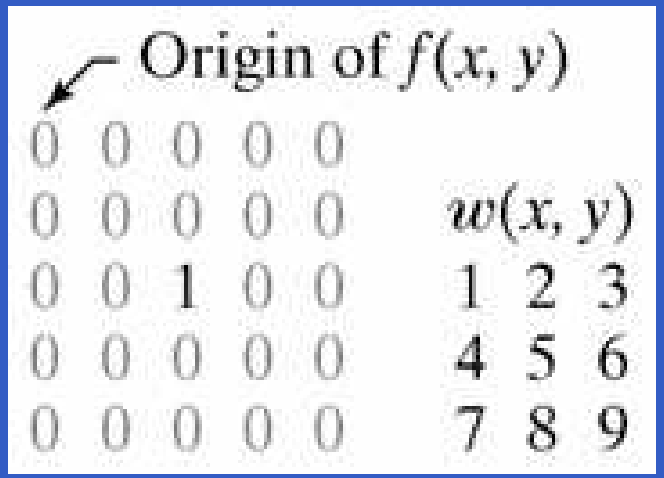


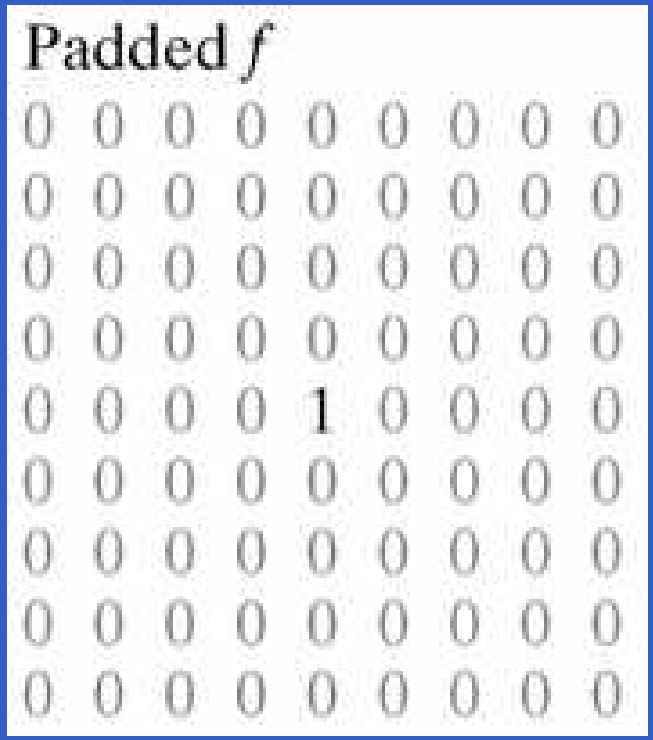
The IPT provides another option, denoted by `'same'` that produces a correlation that is the same size as f. This computation also uses zero padding, but the starting position is with the center point of the mask aligned with the origin of f. The last computation is with the center point of the mask aligned with the last point in f.

[a] Image Processing Toolbox of MATLAB

Linear Spatial Filtering

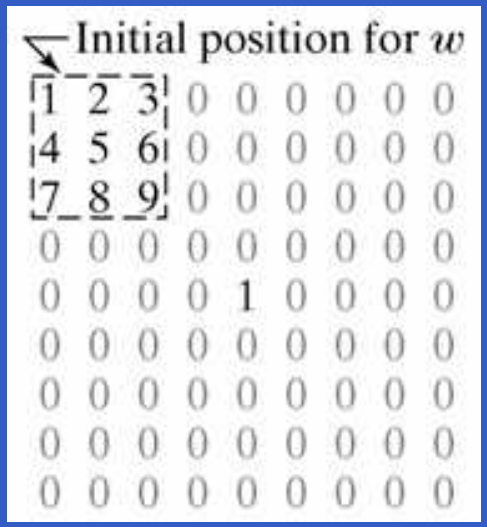
The preceding concepts extend easily to images, as illustrated in the following figures.

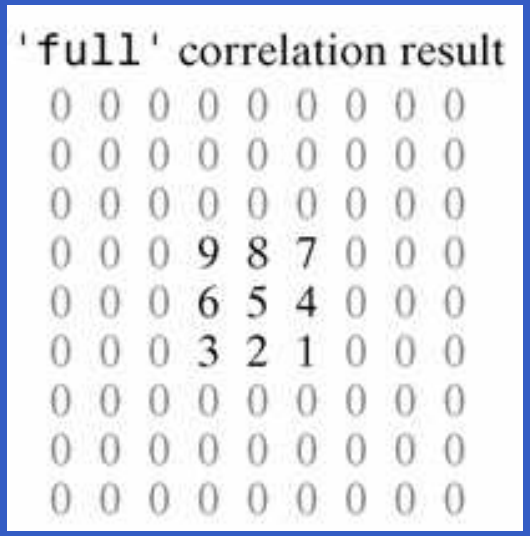


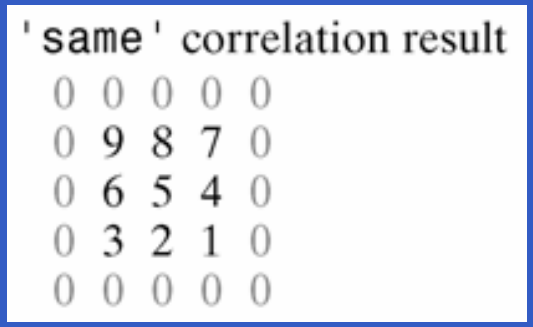


Linear Spatial Filtering

Correlation

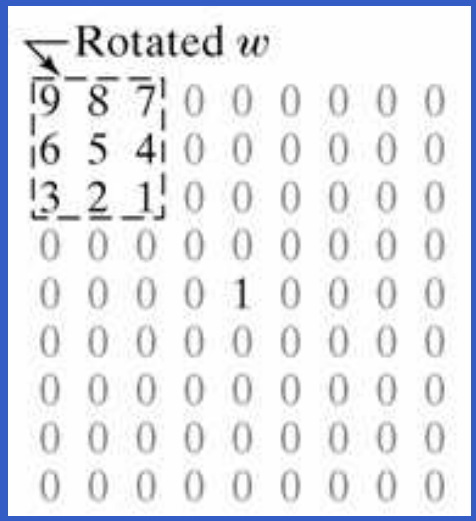


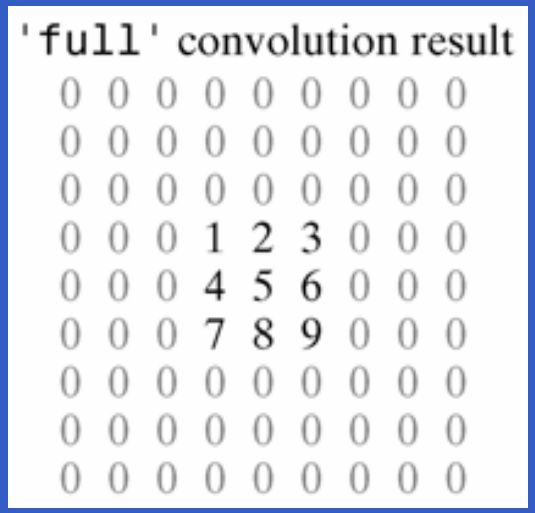


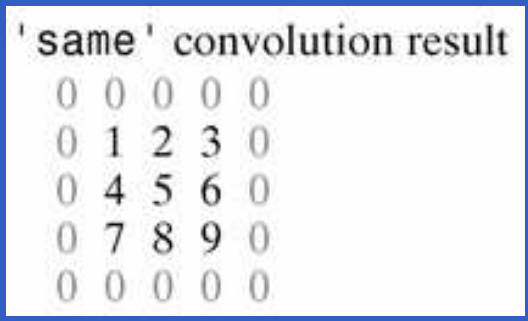


Linear Spatial Filtering

Convolution







Linear Spatial Filtering

```
g=imfilter(f,w,filtering_mode,...
           boundary_options,size_options)
```

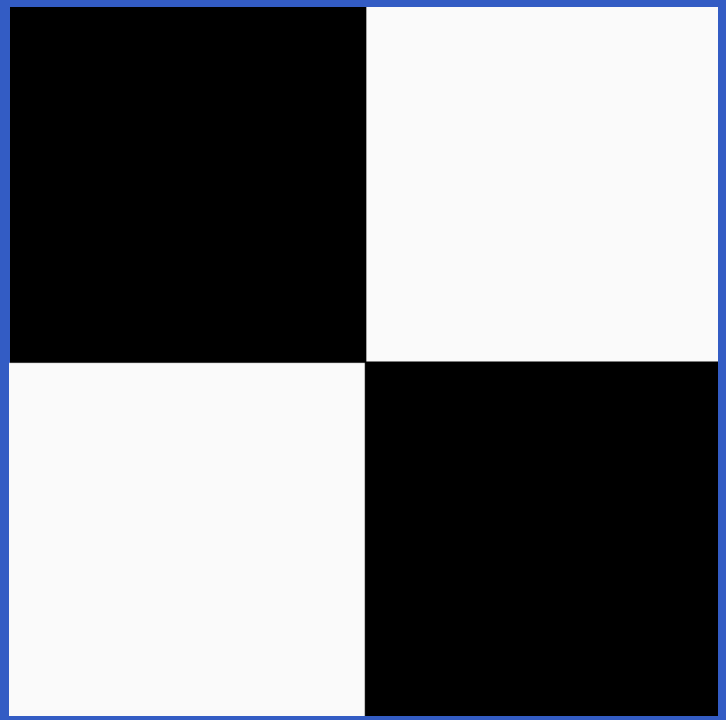
where `f` is the input image, `w` is the filter mask, `g` is the filtered result, and the other parameters are summarized in the following table.

Linear Spatial Filtering

Options	Description
Filtering Mode	
`'corr'`	Filtering is done using correlation. This is the default.
`'conv'`	Filtering is done using convolution.
Boundary Options	
`P`	The boundaries of the input image are extended by padding with a value, `P`. This is the default, with value 0.
`'replicate'`	The size of the image is extended by replicating the values in its outer border.
`'symmetric'`	The size of the image is extended by mirror-reflecting it across its border.
`'circular'`	The size of the image is extended by treating the image as one period a 2-D periodic function.
Size Options	
`'full'`	The output is of the same size as the extended (padded) image.
`'same'`	The output is of the same size as the input. This is the default.

Linear Spatial Filtering

```
>> f=imread('original_test_pattern.tif');
>> f=double(f);
>> w=ones(31);
```

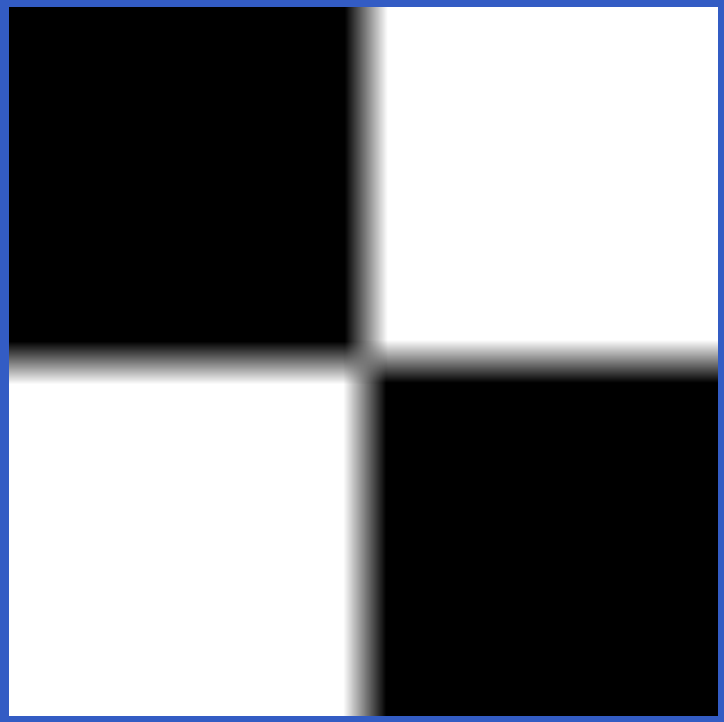


Linear Spatial Filtering

```
>> gd=imfilter(f,w);
>> imshow(gd,[])
```

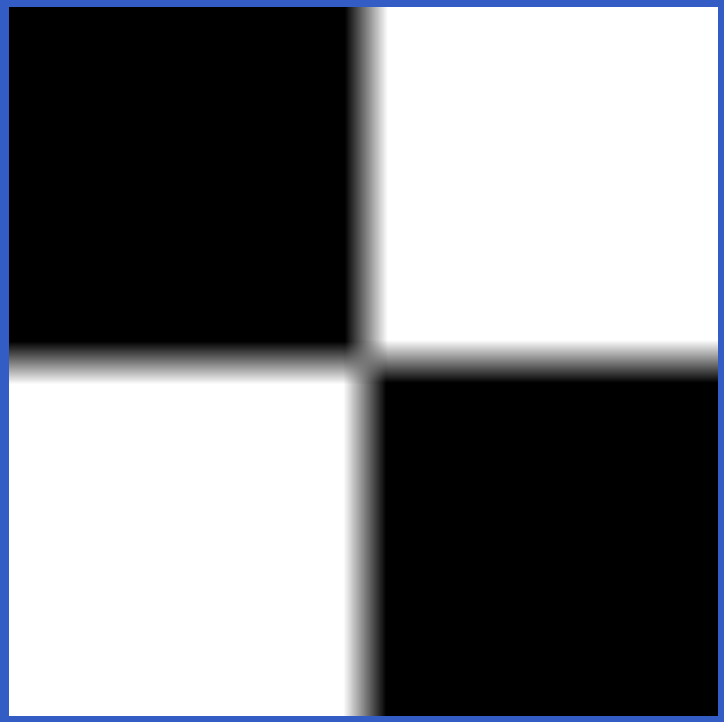
Linear Spatial Filtering

```
gr=imfilter(f,w,'replicate');
imshow(gr,[])
```



Linear Spatial Filtering

```
>> gs=imfilter(f,w,'symmetric');
>> imshow(gs,[])
```

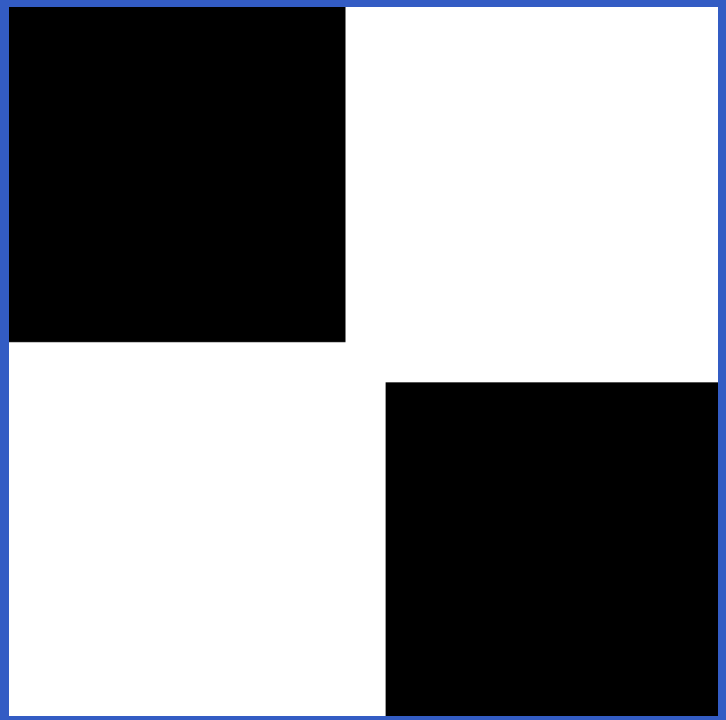


Linear Spatial Filtering

```
>> gc=imfilter(f,w,'circular');
>> imshow(gc,[])
```

Linear Spatial Filtering

```
>> f8=im2uint8(f);
>> g8r=imfilter(f8,w,'replicate');
>> imshow(g8r,[])
```



Nonlinear Spatial Filtering

Nonlinear spatial filtering is based on neighborhood operations also, and the mechanics of defining $m \times n$ neighborhoods by sliding the center point through an image are the same as discussed in linear spatial filtering. Nonlinear spatial filtering is based on nonlinear operations involving the pixels of a neighborhood. For example, letting the response at each center point be equal to the maximum pixel value in its neighborhood is a nonlinear filtering operation. Another basic difference is that the concept of a mask is not as prevalent in nonlinear processing. The ides of filtering carries over, but the "filter" should be visualized as a nonlinear function that operates on the pixels of a neighborhood, and whose response constitutes the response of the operation at the center pixel of the neighborhood.

Nonlinear Spatial Filtering

The IPT provides two functions for performing general nonlinear filtering: `nlfilter` and `colfilt`. The former performs operations directly in 2-D, while `colfilt` organizes the data in the form of columns. Altough `colfilt` requires more memory, it generally executes significantly faster than `nlfilter`. In most image processing applications speed is an overriding factor, so `colfilt` is preferred over `nlfilter` for implementing generalized nonlinear spatial filtering.

Nonlinear Spatial Filtering

Given an input image, `f`, of size $M \times N$, and a neighborhood of size $m \times n$, function `colfilt` generates a matrix, call it `A`, of maximum size $mn \times MN$, in which each column corresponds to the pixels encompassed by the neighborhood centered at a location in the image. For example, the first column corresponds to the pixels encompassed by the neighborhood when its center is located at the top, leftmost point in `f`. All required padding is handled transparently by `colfilt`.

Nonlinear Spatial Filtering

```
g=colfilt(f,[m n],'sliding',@fun,parameters)
```

where `m` and `n` are the dimensions of the filter region, `'sliding'` indicates that the process is one of sliding the $m \times n$ region from pixel to pixel in the input image `f`, `@fun` references a function, which we denote arbitrarily as `fun`, and `parameters` indicates parameters (separated by commas) that may be required by function `fun`. The symbol @ is called a *function handle*, a MATLAB data type that contains information used in referencing a function.

Nonlinear Spatial Filtering

```
fp=padarray(f,[r c],method,direction)
```

where `f` is the input image, `fp` is the padded image, `[r c]` gives the number of rows and columns, by which to pad `f`, and `method` and `direction` are as explained in the next table.

Nonlinear Spatial Filtering

Options	Description
Method	
`'symmetric'`	The size of the image is extended by mirror-reflecting it across its border.
`'replicate'`	The size of the image is extended by replicating the values in its outer border.
`'circular'`	The size of the image is extended by treating the image as one period of a 2-D periodic function.
Direction	
`'pre'`	Pad before the first element of each dimension.
`'post'`	Pad after the last element of each dimension.
`'both'`	Pad before the first element and after the last element of each dimension. This is the default.

Nonlinear Spatial Filtering

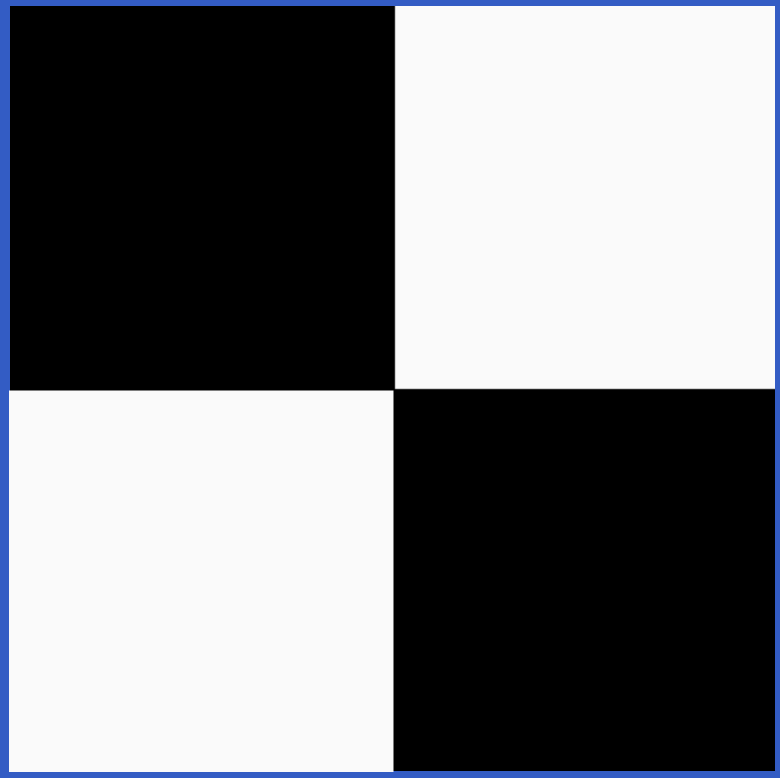
```
>> f=[1 2;3 4];
>> fp=padarray(f,[3 2],'replicate','post')

fp =

     1     2     2     2
     3     4     4     4
     3     4     4     4
     3     4     4     4
     3     4     4     4
```

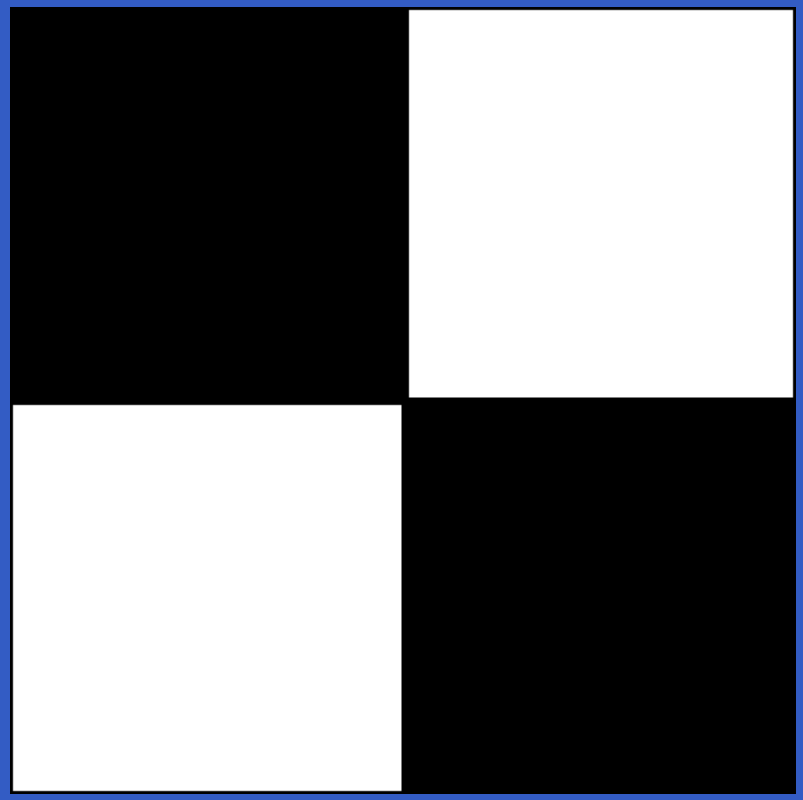
Nonlinear Spatial Filtering

```
function v=gmean(A)

%The length of the columns of A is always mn.
mn=size(A,1);
v=prod(A,1).^(1/mn);
```

```
>> f=padarray(f,[5 5],'replicate');
>> g=colfilt(f,[5 5],'sliding',@gmean);
```

Nonlinear Spatial Filtering





IPT Standard Spatial Filters

- Linear Spatial Filters
- Nonlinear Spatial Filters

Linear Spatial Filters

```
w=fspecial('type',parameters)
```

where `'type'` specifies the filter type, and `parameters` further define the specified filter. The spatial filters supported by `fspecial` are summarized in the following table, including applicable parameters for each filter.

Linear Spatial Filters

Type	Syntax and Parameters
`'average'`	`fspecial('average',[r c])`. A rectangular averaging filter of size `r`$\times$`c`. The default is 3×3. A single number instead of `[r c]` specifies a square filter.
`'disk'`	`fspecial('disk',r)`. A circular averaging filter (within a square of size 2r+1) with radius `r`. The default radius is 5.
`'gaussian'`	`fspecial('gaussian',[r c],sig)`. A Gaussian lowpass filter of size `r`$\times$`c` and standard deviation `sig` (positive). The defaults are 3×3 and 0.5. A single number instead of `[r c]` specifies a square filter.
`'laplacian'`	`fspecial('laplacian',alpha)`. A 3×3 Laplacian filter whose shape is specified by `alpha`, a number in the range $[0, 1]$. The default value for `alpha` is 0.5.
`'log'`	`fspecial('log',[r c],sig)`. Laplacian of a Gaussian (LoG) filter of size `r`$times$`c` and standard deviation `sig` (positive). The defaults are 5×5 and 0.5. A single number instead of `[r c]` specifies a square filter.

Linear Spatial Filters

Type	Syntax and Parameters
`'motion'`	`fspecial('motion',len,theta)`. Outputs a filter that, when convolved with an image, approximates linear motion (of a camera with respect to the image) of `len` pixels. The direction of motion is `theta`, mesaured in degrees, counterclockwise from the horizontal. The defaults are 9 and 0, which represents a motion of 9 pixels in the horizontal direction.
`'prewitt'`	`fspecial('prewitt')`. Outputs a 3×3 Prewitt mask, `wv`, that approximates a vertical gradient. A mask for the horizontal gradient is obtained by transposing the result: `wh=wv'`.
`'sobel'`	`fspecial('sobel')`. Outputs a 3×3 Sobel mask, `sv`, that approximates a vertical gradient. A mask for the horizontal gradient is obtained by transposing the result: `sh=sv'`.
`'unsharp'`	`fspecial('unsharp',alpha)`. Outputs a 3×3 unsharp filter. Parameter `alpha` controls the shape; it must be greater than or equal to 0 and less than or equal to 1.0; the default is 0.2.

Linear Spatial Filters

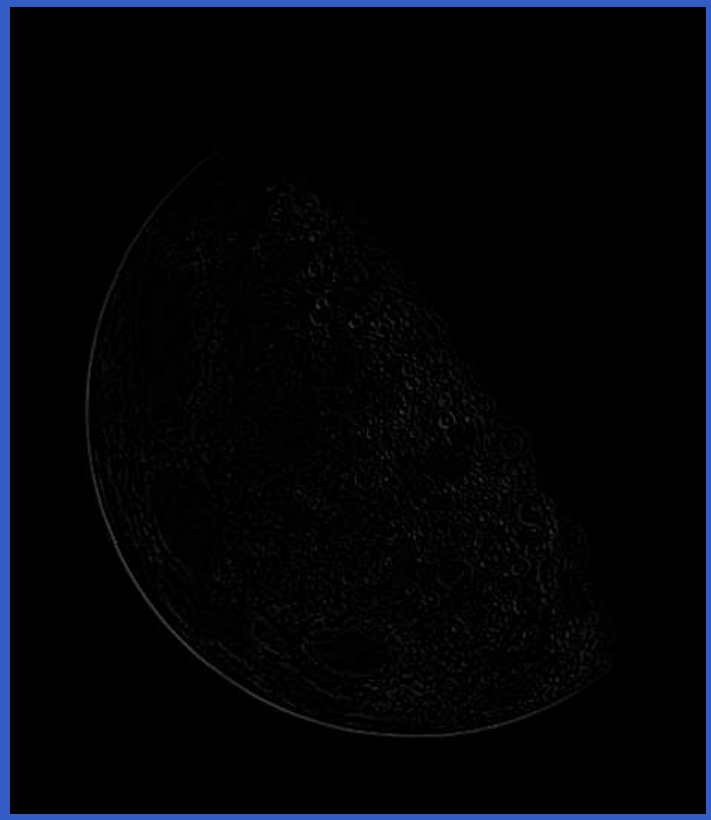
```
>> w=fspecial('laplacian',0)

w =

     0     1     0
     1    -4     1
     0     1     0
```

Linear Spatial Filters

```
>> f=imread('moon.tif');
```

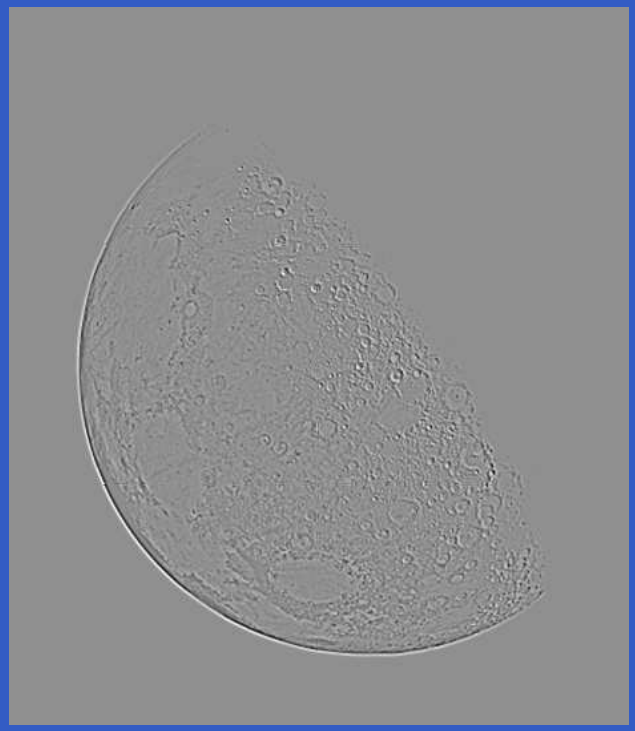
Linear Spatial Filters

```
>> g1=imfilter(f,w,'replicate');
>> imshow(g1,[])
```



Linear Spatial Filters

```
>> f2=im2double(f);
>> g2=imfilter(f2,w,'replicate');
>> imshow(g2,[])
```



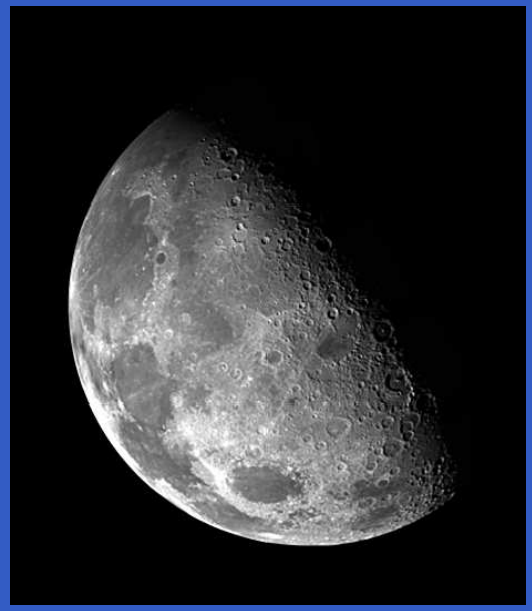
Linear Spatial Filters

```
>> g=f2-g2;
>> imshow(g)
```

Linear Spatial Filters

```
>> f=imread('moon.tif');
>> w4=fspecial('laplacian',0);
>> w8=[1 1 1;1 -8 1;1 1 1];
>> f=im2double(f);
>> g4=f-imfilter(f,w4,'replicate');
>> g8=f-imfilter(f,w8,'replicate');
>> imshow(f)
>> figure, imshow(g4)
>> figure, imshow(g8)
```

Linear Spatial Filters


Nonlinear Spatial Filters

`g=ordfilt2(f,order,domain`

This function creates the output image `g` by replacing each element of `f` by the `order`-th element in the sorted set of neighbors specified by the nonzero elements in `domain`. Here, `domain` is an $m \times n$ matrix of 1s and 0s that specify the pixel locations in the neighborhood that are to be used in the computation. In this sense, `domain` acts like a mask. The pixels in the neighborhood that corresponds to 0 in the `domain` matrix are not used in the computation.

Nonlinear Spatial Filters

Min filter of size $m \times n$:

```
g=ordfilt2(f,1,ones(m,n))
```

Max filter of size $m \times n$:

```
g=ordfilt2(f,m*n,ones(m,n))
```

Median filter of size $m \times n$:

```
g=ordfilt2(f,median(1:m*n),ones(m,n))
```

Nonlinear Spatial Filters

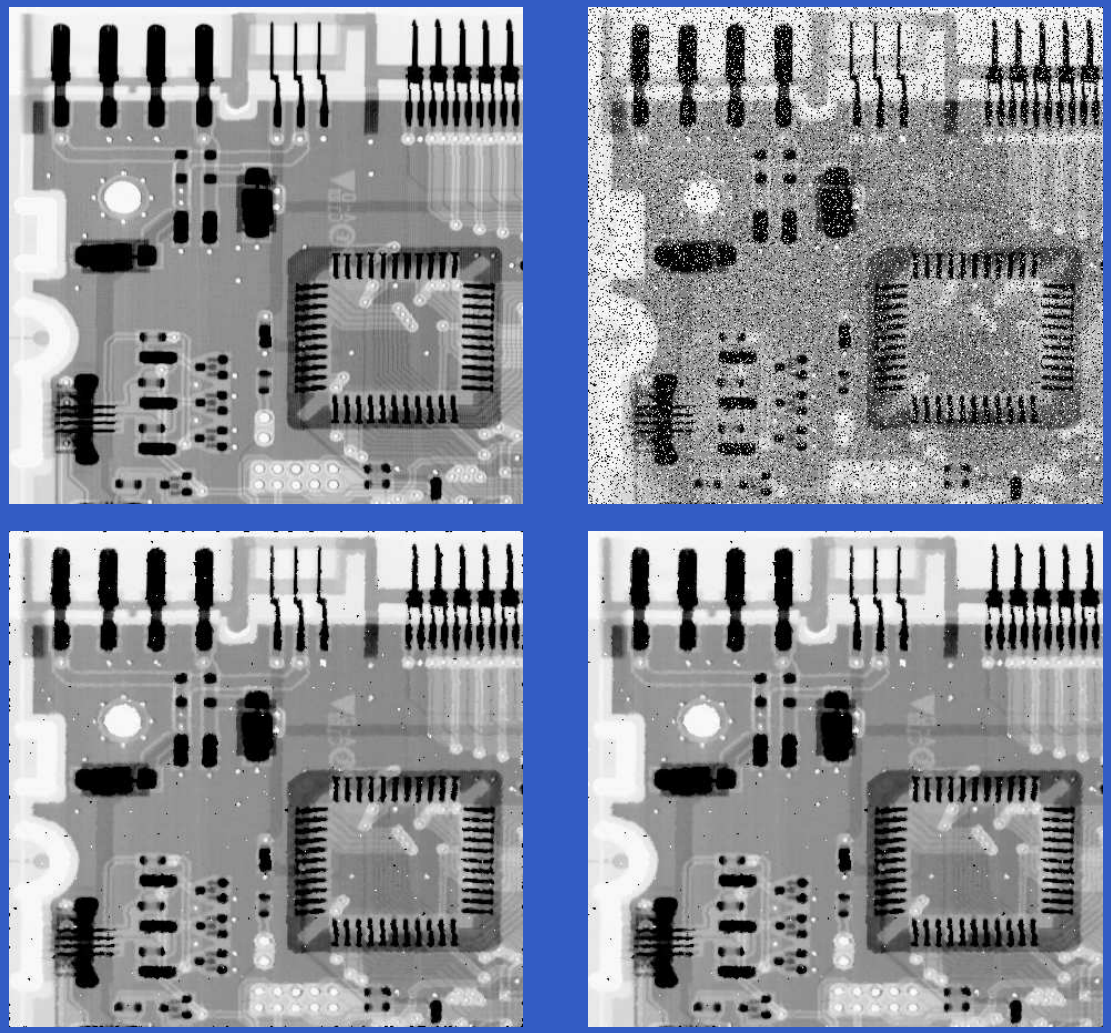
`g=medfilt2(f,[m n],padopt`

where the tuple `[m n]` defines a neighborhood of size $m \times n$ over which the median is computed, and `padopt` specifies one of three possible border padding options: `'zeros` (the default), `'symmetric'` in which `f` is extended symmetrically by mirror-reflecting it across its border, and `'indexed'`, in which `f` is padded with `1`s if it is of class `double` and with `0`s otherwise. The default form of this function is `g=medfilt2(f)` which uses a 3×3 neighborhood to compute the median, and pads the border of the input with `0`s.

Nonlinear Spatial Filters

```
>> f=imread('ckt-board.tif');
>> fn=imnoise(f,'salt & pepper',0.2);
>> gm=medfilt2(fn);
>> gms=medfilt2(fn,'symmetric');
>> subplot(2,2,1), imshow(f)
>> subplot(2,2,2), imshow(fn)
>> subplot(2,2,3), imshow(gm)
>> subplot(2,2,4), imshow(gms)
```

Nonlinear Spatial Filters



References

- R. C. Gonzalez, R. E. Woods, S. L. Eddins: Digital Image Processing Using MATLAB. Pearson Prentice Hall, 2004
- R. C. Gonzalez, R. E. Woods: Digital Image Processing. Prentice Hall, 2002
- `http://www.imageprocessingplace.com`