Convolution Filters

Convolution operators usually refer to a class of 2D kernels that are convolved with an image to produce a desirable effect (simple linear filtering). In some cases it is more efficient to perform convolutions using the FFT (similar to the convolution example above), i.e., transform both the image and the filter waves, multiply the transforms in the frequency domain and then compute the inverse transformation using the IFFT. The FFT approach is more efficient for convolution with kernels that are greater than 13x13 pixels. However, there is a very large number of useful kernels which play an important role in image processing that are 3x3 or 5x5 in size. Because these kernels are so small, it is fairly efficient to implement the corresponding linear filter as direct convolution without using the FFT.

In the following example we implement a low-pass filter with equal spatial frequency response along both axes.

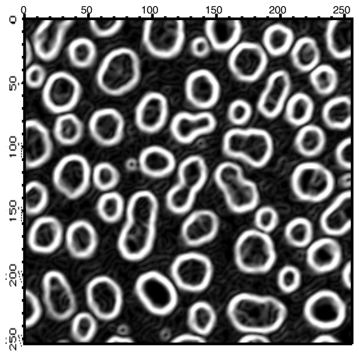
```
Make /O/N=(9,9) sKernel       // first create the convolution kernel
SetScale/I x -4.5,4.5,"", sKernel
SetScale/I y -4.5,4.5,"", sKernel

// Equivalent to rect(2*fx)*rect(2*fy) in the spatial frequency domain.
sKernel=sinc(x/2)*sinc(y/2)

// Remember: MatrixConvolve executes in place; save your image first!
Duplicate/O root:images:MRI mri
Redimension/S mri                         // to avoid integer truncation
MatrixConvolve sKernel mri
NewImage mri
ModifyImage mri ctab= {*,*,Rainbow,0}    // just to see it better
```

The next example illustrates how to perform edge detection using a built-in convolution filter in the **ImageFilter** operation (see page V-372):

```
Duplicate/O root:images:blobs blobs
ImageFilter findEdges blobs
NewImage blobs
```



Other notable examples of image filters are Gauss, Median and Sharpen. You can also apply the same operation to 3D waves. The filters Gauss3D, avg3D, point3D, min3D, max3D and median3Dare the extensions of their 2D counterparts to 3x3x3 voxel neighborhoods. Note that the last three filters are not true *convolution* filters.

Edge Detectors

In many applications it is necessary to detect edges or boundaries of objects that appear in images. The edge detection consists of creating a binary image from a grayscale image where the pixels in the binary image are turned off or on depending on whether they belong to region boundaries or not. In other words, the detected edges are described by an image, not a vector (1D wave). If you need to obtain a wave describing boundaries of regions, you might want to use the **ImageAnalyzeParticles** operation (see page V-363).