Assume that you will be working with images of size $M \times N$ whose intensities are in the range [0,1], so that salt pixels will have value 1 and pepper pixels value 0.

(b) Use your analysis from (a) and the uniform random number generator from Problem 5.2 as the basis for writing a function **g = saltPepper4e(f,ps,pp)** to add salt-and-pepper noise to image **f**. Parameters **ps** and **pp** are the probabilities mentioned in (a).

(c)* Read the image **testpattern512.tif** and repeat the four levels of noise outlined in Problem 5.2(a), using equal values for **ps** and **pp**. Show your images and values used.

(d)* Repeat (c) using only pepper noise.

(e) Repeat (c) using only salt noise.

5.4 This project deals with extending the linear spatial filtering concepts introduced in Chapter 3. The objective is to use project function **twodConv4e** to implement the mean filters discussed in Section 5.3. Although some of those filters (e.g. the geometric mean filter) perform nonlinear operations on the pixels of a neighborhood, it is possible to convert the nonlinear operations to a form that allows linear filtering (i.e., sum-of-products operations) using spatial convolution. The solution to (b) shows how do to this. In the following, **g** is a noisy input image, **f_hat** is the filtered (estimate) image, and $m \times n$ is the size of the neighborhood that defines the filter size.

(a)* Write a function **f_hat = aMean4e(g,m,n)** that implements the arithmetic mean filter defined in Eq. (5-23).

(b)* Write a function **f_hat = geoMean4e(g,m,n)** that implements the geometric mean filter defined in Eq. (5-24).

(c) Write a function **f_hat = harMean4e(g,m,n)** that implements the harmonic mean filter defined in Eq. (5-25).

(d) Write a function **f_hat = ctharMean4e(g,m,n,q)** that implements the contraharmonic mean filter in Eq. (5-26). (*Hint:* Because you will performing a ratio computation of two filtered images, you should disable the autoscaling in project function **twodConv4e**, and do the scaling at the end using project function **intScaling4e**.)

(e)* Read the image **circuitboard-gaussian.tif** and use function **aMean4e** to duplicate the result in Fig. 5.7(c).

(f) Read the image **circuitboard-gaussian.tif** and use function **geoMean4e** to duplicate the result in Fig. 5.7(d).

(g) Read the image **circuitboard-pepper.tif** and use function **ctharMean4e** to duplicate the result in Fig. 5.8(c).

(h) Read the image **circuitboard-salt.tif** and use function **ctharMean4e** to duplicate the result in Fig. 5.8(d).

5.5 The objective of this project is to implement nonlinear (order-statistic) spatial filters. You have to implement a function in the language you are using, capable of performing sliding neighborhood operations. The concept is the same as convolution, except that, instead of performing a sum-of-products (linear) operation on every neighborhood, the function you need now must be capable of performing neighborhood operations that in general are nonlinear, and are specifiable by you. This capability is required to implement some of the filters discussed in Section 5.3. If you are using MATLAB and have the MATLAB Image Processing Toolbox installed in your system, you can use function **nlfilter**. If you do not have the Image Processing Toolbox installed, you can use utility function **mynlfilter**, which is included in the utilities folder of your *DIP4E Student Support Package*. See the solution to (a) below for details on the characteristics of the required sliding neighborhood function.

(a)* Implement a general sliding neighborhood filtering function capable of accepting user-defined operations, and performing these operations on a neighborhood of size $m \times n$.

(b)* Use your function from (a) as the basis for writing a function **f_hat = minFilter4e(g,m,n)** that implements a min filter of size $m \times n$.

(c) Use your function from (a) as the basis for writing a function **f_hat = maxFilter4e(g,m,n)** that implements a max filter of size $m \times n$.

(d) Use your function from (a) as the basis of a function **f_hat = medianFilter4e(g,m,n)** that

implements a median filter of size $m \times n$. (*Hint:* If you are using MATLAB, start with function **median**, but be aware that this function only works with 1-D arrays, so your neighborhoods will have to be converted to 1-D.)

(e) Read the image **hubble.tif** and determine the minimum size of a square neighborhood needed to generate *in one pass* an image in which only part of the largest object remains. (*Hint:* Use either a max or a min filter—you have to determine which.)

(f) Read the image **circuitboard-saltandpep.tif** and use your function from (d) to duplicate the results in Figs. 5.10(b)-(d).

5.6 Experimenting with notch filters.

(a)* Write a function **H = notchReject4e(type,M,N, param,C)** for implementing an $M \times N$ notch reject filter transfer function of specified **type: 'ideal'**, **'gaussian'**, **'butterworth'**, or **'rectangular'**. Array **param** is a list of the parameters needed to implement the chosen filter **type**, and **C** contains specified locations for the notches. Only one notch location is required. The function should generate the symmetric notch automatically. Your function need only be capable of generating one notch reject transfer function each time it is called.

(b)* Read the image **astronaut-interference.tif** and use notch filtering in the frequency domain to remove the sinusoidal interference. Do the filtering using project function **dftFiltering4e** without padding. Display the original image, the spectrum, the filter transfer function you used, the processed image, and the interference pattern. The processed image should look like Fig. 5.16(d). (*Hint:* Compute the spectrum and magnify the area near the center of the spectrum so you can see the location of the energy bursts caused by the periodic interference. Then use a very narrow Gaussian notch reject filter transfer function with centers at those locations.)

(c)* Repeat (b) using zero padding and explain the principal reason for the difference between your results. Your explanation will reveal why we did not use image padding in