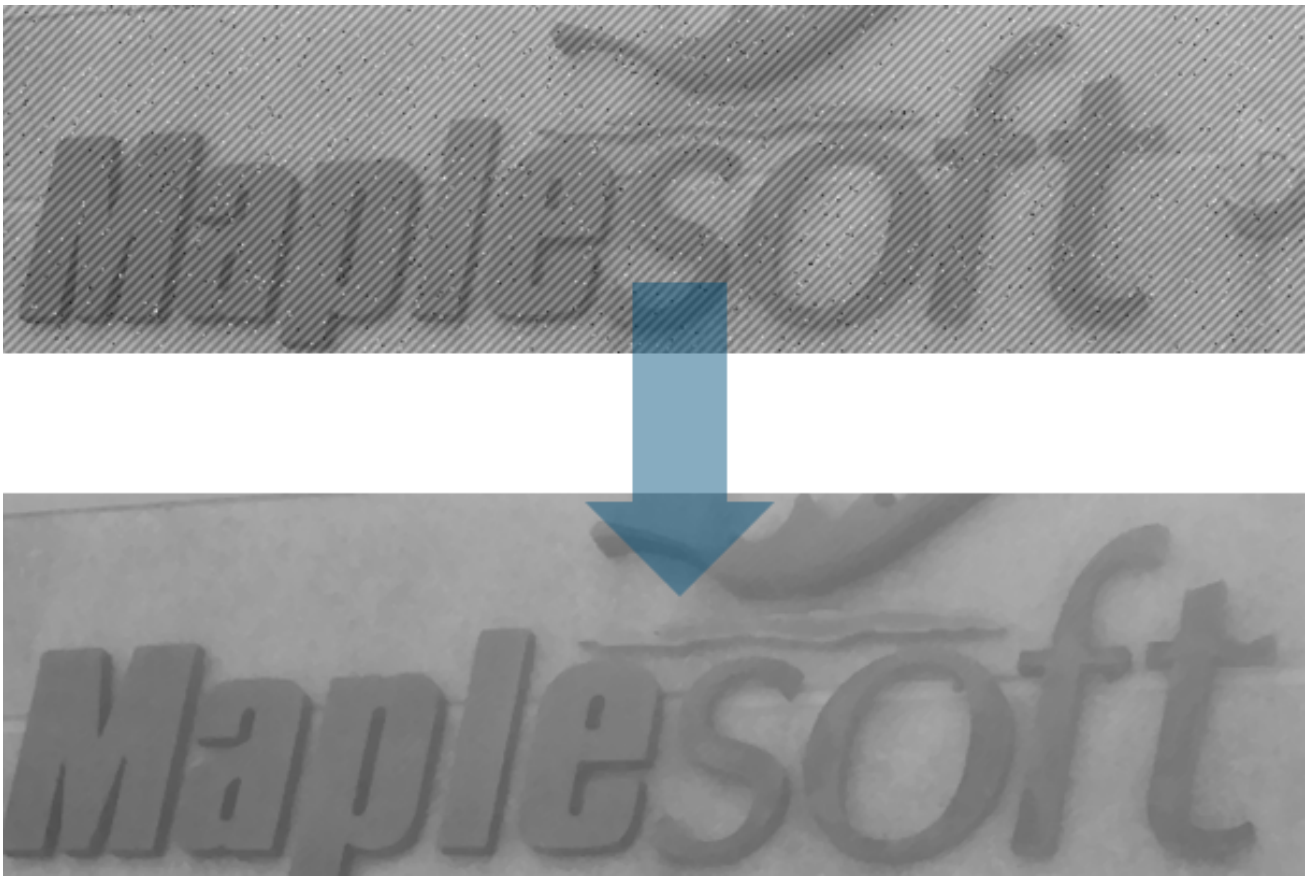
Removal of Periodic and Salt & Pepper Noise from an Image

Introduction

This application demonstrates how you can denoise an image polluted with periodic and salt & pepper noise:



Images are often polluted with unwanted variations in colouring or brightness, often produced by the sensor or circuitry of an image capture device. Since all signal processing devices are to some extent susceptible to electronic noises, many noise reduction techniques have been developed.

Two noise removal techniques are examined:

• Periodic noise is removed by applying a notch filter in the frequency domain
• Salt and pepper is removed with a median filter

Reference:

Rafael C. Gonzalez; Richard E. Woods (2007). Digital Image Processing. Pearson Prenctice Hall. ISBN 0-13-168728-X.
Linda G. Shapiro; George C. Stockman (2001). Computer Vision. Prentice-Hall. ISBN 0 -13-030796-3.
Charles Boncelet (2005). "Image Noise Models". In Alan C. Bovik. Handbook of Image and Video Processing. Academic Press. ISBN 0-12-119792-1.

```
> restart:
  with(ImageTools):
```

Import Image and Add Noise

In order to demonstrate the noise removal techniques, we will first artificially pollute a clean image with noise, to use as our example.

Import Image

Please ensure this image file is located in the same folder as this worksheet, or modify the path in the import command to point to the file's location.

```
> img:=Matrix(RGBtoGray(Import("image.jpeg"))):
> nRows, nCols := LinearAlgebra:-Dimension(img):
> Embed(Create(img));
```

Sale & Pepper Noise

We define a procedure that adds salt and pepper noises to the image. Salt and pepper noises result in the irregular appearance of dark pixels in the bright area and bright pixels in the dark area of the image. A major source of period noise is errors in the process of converting analog signals to digital signals. In this case we are artifically producing this error by adding white and black pixels randomly across the image.

The intensity is by default 50%, we will call the procedure with intensity=0.9.

```
> saltAndPepper := proc(img::Matrix, nCols, nRows,
  intensity::float:=0.5)
            local xrand, yrand, i:
            xrand := rand(1..nCols):
            yrand := rand(1..nRows):
            for i from 1 to ceil(intensity*0.004*nRows*nCols) do
                      img[yrand(),xrand()] := 0:
                      img[yrand(),xrand()] := 1:
            end do:
  end proc:
> saltAndPepper(img,nCols,nRows,0.9):
> Embed(Create(img));
```

Periodic Noise

Then we add periodic noises to the image. Period noises result in repetitive patterns being added to the original image, they are usually caused by electrical or electromechanical interference in the image capturing process (Gonzalez & Woords, 2007). Here we will artifically generate a periodic noise by adding a sinusoidal wave to the original frequency, which produces a diagnoal pattern across the image.

```
> noise := Matrix(nRows, nCols,(i,j) -> evalhf(sin((i+j))/10),
  datatype = complex[8]):
> imgNoisy := noise+~img:
> Embed(Create(imgNoisy));
```

Removing periodic noise with Fourier Transform

In order to remove the periodc noise, we first decompose the image into frequency waves by performing a 2D Fourier Transform on the image.

```
> FFT2D := proc(M)
     local nCols, nRows, fft_img, i:
     fft_img := Matrix(M, datatype = complex[8]):
     nRows, nCols := LinearAlgebra:-Dimensions(fft_img):
     for i from 1 to nCols do
        fft_img[.., i] := SignalProcessing:-FFT(fft_img[.., i]):
     end do:
     for i from 1 to nRows do
        fft_img[i, ..] := SignalProcessing:-FFT(fft_img[i, ..]):
     end do:
     return fft_img:
  end proc:
```

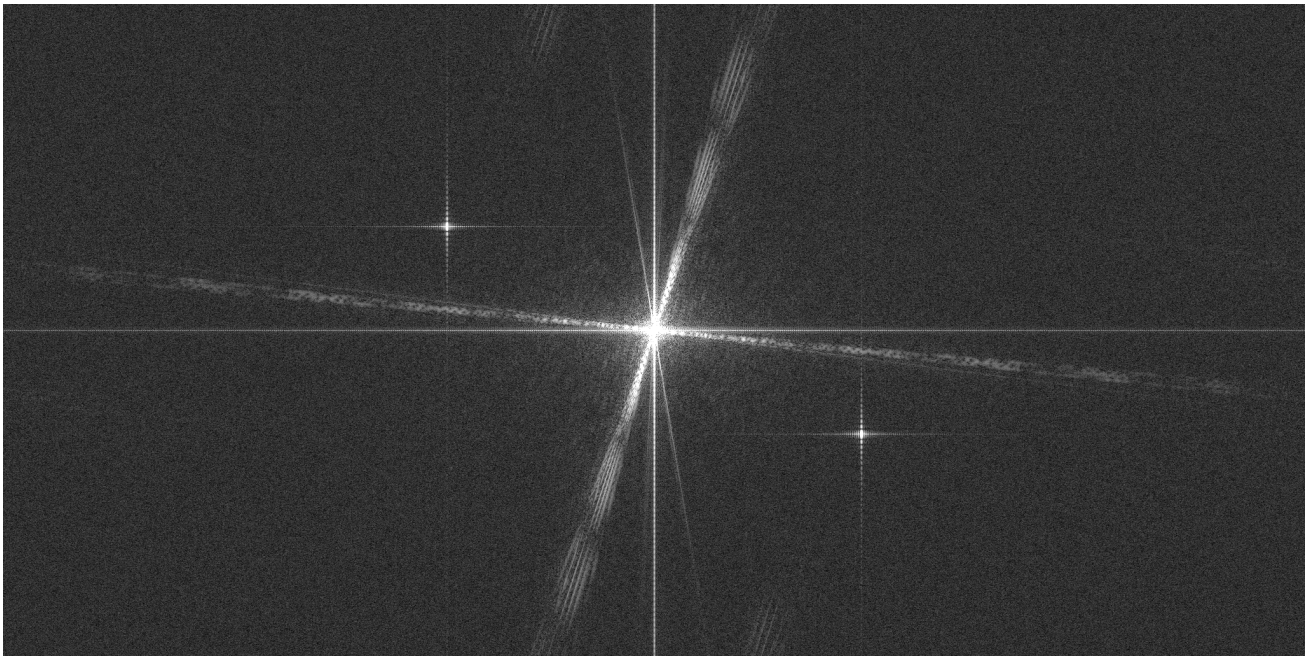
We resize the image matrix in order to apply the SignalProcesing:-FFT() command.

```
> rows_closest := 2^(ceil(MTM:-log2(nRows))):
  cols_closest := 2^(ceil(MTM:-log2(nCols))):
  spec_img := FFT2D(Matrix(1..rows_closest,1..cols_closest,
  imgNoisy)):
```

We define a procedure called fftshift to shift the zero frequency component to the centre of the image.

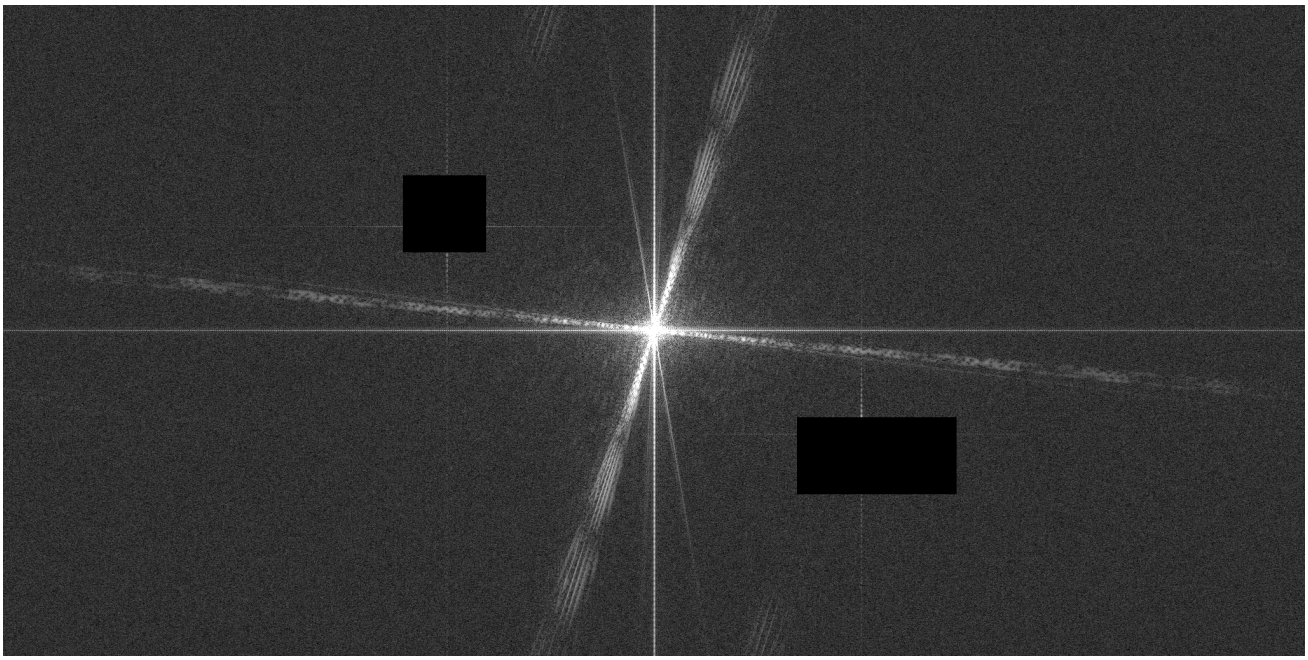
```
> fftshift := proc(M)
     local nRows, nCols, quad_1, quad_2, quad_3, quad_4, cRows,
```

```
  cCols;
     nRows, nCols := LinearAlgebra:-Dimensions(M):
     cRows, cCols := ceil(nRows/2), ceil(nCols/2):
     quad_1 := M[1..cRows,      1..cCols]:
     quad_2 := M[1..cRows,      cCols + 1..-1]:
     quad_3 := M[cRows + 1..-1, cCols + 1..-1]:
     quad_4 := M[cRows + 1..-1, 1..cCols]:
     return <<quad_3, quad_2 |quad_4, quad_1>>:
  end proc:
> spec_img := fftshift(spec_img):
> Embed(Create(abs~(spec_img)));
```



Looking at the frequency doman of the magnitude (abs~(img)) of the transformed image, we can see that other than the central area with bright frequency pixels (indicating strong intensity of pixels with corresponding frequencies in the original image), there are two separate, discrete spikes on each side of the image. Periodic noises tend to look like this in the frequency domain. Reduction of this type of noises can be done by filtering/removing those outlying frequencies. In this case, we will maually do so by setting the corresponding matrix entry to 0.

```
> spec_img[650..770,1250..1500]:=0:
> spec_img[270..390,630..760]:=0:
> Embed(Create(abs~(spec_img)));
```



Now we reverse the effect of fftshift by applying ifftshift and after that convert the image from frequency domain back to spatial domain using Inverse 2D FFT.

```
> ifftshift := proc(M)
     local nRows, nCols, quad_1, quad_2, quad_3, quad_4, cRows,
  cCols;
     nRows, nCols := LinearAlgebra:-Dimensions(M):
     cRows, cCols := floor(nRows/2), floor(nCols/2):
     quad_1 := M[1..cRows,      1..cCols]:
     quad_2 := M[1..cRows,      cCols + 1..-1]:
     quad_3 := M[cRows + 1..-1, cCols + 1..-1]:
     quad_4 := M[cRows + 1..-1, 1..cCols]:
     return <<quad_3, quad_2 |quad_4, quad_1>>:
  end proc:
> InverseFFT2D := proc(M)
     local nCols, nRows, i:
     nRows, nCols := LinearAlgebra:-Dimensions(M):
     for i from 1 to nCols do
        M[.., i] := SignalProcessing:-InverseFFT(M[.., i]):
     end do:
     for i from 1 to nRows do
        M[i, ..] := SignalProcessing:-InverseFFT(M[i, ..]):
     end do:
     return M:
  end proc:
> img := Re(InverseFFT2D(ifftshift(spec_img)))[1..nRows,1..nCols]
  :
> Embed(Create(img));
```

Removing salt and pepper noise with median filter

We will remove salt and pepper noise from the image with a median filter, in order to do so, we first allocate a bigger matrix of size (nRows+2) * (nCols+2) and then put the original matrix in the middle. In other words, we want to pad the original matrix with an extra row/column on each boundary.

```
> padded := Matrix(1..nRows+2, 1..nCols+2):
> for y from 2 to nRows+1 do
            for x from 2 to nCols+1 do
                        padded[y,x] := img[y-1,x-1]:
            end do:
  end do:
  for y from 1 to nRows+2 do
            padded[y,1] := 0:
            padded[y,nCols+2]:= 0:
  end do:
  for x from 1 to nCols+2 do
            padded[1,x] := 0:
            padded[nRows+2,x]:= 0:
  end do:
```

Then we apply a 3x3 kernel/window across every 3x3 sub-matrix, finding the median of the 9 elements after sorting and placing the median as the final entry back into the original matrix. This process is known as applying a median filter to an image.

```
> for y from 2 to nRows+1 do
```

```
            for x from 2 to nCols+1 do
                        inc := 1:
                        windows := Array(1..9,fill=0);
                        for i from -1 to 1 by 1 do
                                    for j from -1 to 1 by 1 do
                                                windows(inc) := padded
  [y+i,x+j]:
                                                inc := inc + 1:
                                    end do:
                        end do:
                        med := sort(convert(windows,list)):
                        padded[y,x]:=med[5]:
            end do:
  end do:
```

Removing the extra padding we added before, the resulting image will look like:

```
> Embed(Create(padded[2..-2,2..-2]));
```