

CSCI 124 - Discrete Structures II - Spring 2006
George Washington University

MATLAB Exercises

Typing “matlab” at a command prompt gets you into the matlab environment.

You can see your variables at any time by typing “who”.

Typing “help function” provides matlab documentation on the function. Matlab manuals may also be found online.

Use “save” to save your work to a file matlab.mat which you can load the next time using “load”. “help save” and “help load” will tell you more.

You can see the images and sounds using matlab viewers only if you are located at a hobbes machine, not if, for example, you have logged in remotely. In that case, you would need to write the images and audio clips onto files and view them through your installed viewers directly, and not through matlab.

1 Images in Matlab

Grey scale images are read into two dimensional arrays in matlab.

Step 1: Read Image

```
Image1 = imread('cameraman.tif');
```

Do not forget the semi-colon, else matlab will write the entire large matrix on the screen.

You can also try images: coin.png, peppers.png, pout.tif, saturn.png, rice.png, circles.png

Step 2A: See size of image

```
size(Image1)
```

without a semi-colon will show you the size of the image. If there is a third dimension, it should be “3”, and indicates a color image.

Step 2 B: Crop image if size too large

```
Image1Crop = Image1(xlow:xhigh, ylow:yhigh)
```

Step 3: View Image

```
imview(Image1Crop)
```

Step 4: Create the blur function

For right now, use a grey-scale image, as a color image will take too long.

```
ImageBlur = 0.04*ones(5, 5)
```

Notice, this is a blur of size 5×5 pixels, and $0.04 = \frac{1}{25}$, as the blurred image is an average of pixel values in the 5×5 window. If you had a color image, you would use: `0.04*ones(5, 5, 3)`

Step 5: Blur image

```
Blurred = imfilter(Image1Crop, ImageBlur);
```

Step 6: View it

```
imview(Blurred);
```

Step 7: Now see if you can undo the blur, knowing what it is

```
Image1Est = deconvwnr(ImageBlur, Blurred);
```

```
imview(Image1Est);
```

Compare the two images, Image1Crop and Image2Est, visually, as well as numerically:

```
ImageDiff = Image1Crop-Image1Est;
```

```
max(max(ImageDiff))
```

will give you the maximum estimation error at a pixel.

You may write images using `imwrite(a,filename,fmt)` onto system files which you can then view using installed image viewers, and not through matlab.

2 Sound clips in Matlab

Matlab has sound clips available: chirp, handel, splat, gong, laughter, train. handel, for example, may be loaded using `load handel`, and the sound will be loaded in variable `y`, and the sampling frequency in Hertz, in variable `Fs`.

Step 1: Load sound

```
load handel;
```

```
size(y)
```

will give you a sense of the size of the signal, it will typically be very long.

Step 2: Play sound Play first three seconds.

```
p = audioplayer(y, Fs);
```

```
play(p, [1 (Fs * 3)]);
```

where 1 is the start sample, and `Fs*3` the stop sample.

Step 3: Generate Blur function

```
SoundBlur = 0.2*ones(5, 1);
```

(`ones(5)` will generate a square matrix of size 5×5 .)

Step 4: Blur sound Again, you may want to blur only a small part of the sound (it might take too long otherwise)

```
BlurredHandel = conv(y(1:Fs, 1), SoundBlur);
```

Step 5A: Play blurred sound

```
p = audioplayer(BlurredHandel, Fs);
```

```
play(p);
```

Step 5B: Use a numerical measure Check its standard deviation, and compare with that of original.

```
std(y(1:Fs, 1))
```

std(BlurredHandel)

Is there a difference? Can you explain the difference?

Step 6: Reconstruct

HandelEst = deconv(BlurredHandel, SoundBlur);

Step 7: Compare. Again, both numerically and by ear.

If you wish, you may load .wav files using wavread. Three files may be found on the course website. You may write sound using wavwrite, and hear it through your own Winamp software.

3 Matrices in Matlab

Matrices in matlab are treated as two-dimensional arrays. While there are several ways of reading a matrix into the workspace, we can use the simplest one: by entering in the numerical values. For example,

$$A = [1.49, -0.41, -0.08; -0.21, 1.29, -0.08; -0.21, -0.41, 1.62]$$

enters the matrix:

$$A = \begin{bmatrix} 1.49 & -0.41 & -0.08 \\ -0.21 & 1.29 & -0.08 \\ -0.21 & -0.41 & 1.62 \end{bmatrix}$$

Hereafter, “A” refers to this matrix. Semi-colons separate rows, and commas separate elements.

Step 1: Enter a matrix of your choice.

Notice that, if you type a semi-colon at the end of the statement, matlab does not type back variable values.

Now determine the eigenvalues of your chosen matrix:

Step 2: Find its eigenvalues: type “eig(A)”. Typing “help eig” will provide the matlab documentation on “eig”.

Step 3: obtain the eigenvectors, type “[V, D] = eig(A)”. This will give you a diagonal matrix, D, containing the eigenvalues of A, and a matrix V whose columns are the corresponding eigenvectors.

Step 4: Try to isolate a single eigenvector To obtain a single eigenvector, say the second one, type “V(:, 2)”. A(:, i) is matlab’s notation for the i^{th} column of matrix A.

Step 5: Check norms of eigenvectors Type “norm(V(:, 2))”. This will give you the norm of the second column of V. You may notice that the eigenvectors are all normalized, their norms are all one.

Step 6: Question Suppose the matrix you entered was a color correction matrix. How would you expect it to behave with noise? Think about this.

Step 7: Reconstruct Now use matrix A to obtain a reconstruction of a measured color:

$$x = \begin{bmatrix} 100 \\ 40 \\ 40 \end{bmatrix}$$

That is, type “`y=A*[100; 40; 40]`”. You should get

$$y = \begin{bmatrix} 129.4 \\ 27.4 \\ 27.4 \end{bmatrix}$$

Step 8: Perceptual comparison A color image is stored as a three dimensional array (i.e. as three matrices). Make the above color into an image of size 64×64 for viewing. Type:

```

RGBImage = ones(64, 64, 3);
RGBImage(:, :, 1) = RGBImage(:, :, 1)*129.4;
RGBImage(:, :, 2) = RGBImage(:, :, 2)*27.4;
RGBImage(:, :, 3) = RGBImage(:, :, 3)*27.4;

```

“`ones(m, n, o)`” generates an array of all ones of size (m, n, o) . So we use it to initialize our image. We then multiply the respective frames by the number, as we want an image of a constant color. Then, view the image:

```
imshow(RGBImage)
```

Step 8: Add noise Now add, to `x`, noise as a multiple of any unit vector you please, say, as a multiple of the vector

$$z = \begin{bmatrix} -0.5 \\ 0.5 \\ 0.7 \end{bmatrix}$$

For example, add 20 times `z` to `x`, to obtain `xprime`.

```
xprime = x + 20*z;
```

Then obtain the corresponding corrected color, call it `yprime`.

```
yprime = A*xprime;
```

```
numericaldifference = y-yprime;
```

Display it next to the color obtained without noise, `y`.

```

RGBImage = ones(64, 128, 3);
RGBImage(:, 1:64, 1) = 129.4*RGBImage(:, 1:64, 1);

RGBImage(:, 65:128, 1) = 109.28*RGBImage(:, 65:128, 1);

RGBImage(:, 1:64, 2) = 27.4*RGBImage(:, 1:64, 1);

RGBImage(:, 65:128, 2) = 41.28*RGBImage(:, 65:128, 1);

RGBImage(:, 1:64, 2) = 27.4*RGBImage(:, 1:64, 1);

RGBImage(:, 65:128, 2) = 48.08*RGBImage(:, 65:128, 1);

```

How bad is the difference?