

# General Introduction

These notes are mainly designed to help people begin display programming with the Psychtoolbox. Although these notes also teach introductory programming, I'd also recommend the book:

*Basics of Matlab and Beyond, by Andrew Knight,  
published by Chapman and Hall/CRC*

In the Knight book Chapters 1-11, 28-31 are probably the most useful.

Beginning programmers should work their way from the beginning. Programmers who are comfortable with another programming language should skim over the first chapters and begin at Chapter 9.

Comments/suggests very welcome – [ifine@usc.edu](mailto:ifine@usc.edu)

Ione Fine (Assistant Professor)  
Doheny Retina Institute  
Keck School of Medicine, USC  
1450 San Pablo Street, DEI 3605  
Los Angeles, CA, 90033  
(858) 945-4793

## Table of Contents

Topic 1 – What is programming? .....	2
Topic 2– Starting Matlab .....	4
Topic 3 – Your first program.....	6
Program 1 – MixStrings.m .....	6
EXERCISE 1 .....	8
Topic 4 – Calculations .....	9
EXERCISE 2.....	10
Program 2 - Calculations.m .....	10
EXERCISE 3.....	11
Topic 5 –Graphics I .....	12
Program 3 – Squares.m.....	12
EXERCISE 4 .....	13
Topic 6 – Functions.....	14
EXERCISE 5.....	15
Topic 7 – Reading/Writing Data to files .....	16
EXERCISE 6.....	17
Topic 8 – Planning a program .....	18
Topic 9 – Starting Matlab (revisited).....	20
Topic 10 – Vectors, Matrices, Calculations, Operators.....	23

Program 4 (again) - Calculations.m .....	27
EXERCISE 7 .....	27
Program 6 – CosmoSexQuiz.m .....	28
EXERCISE 8 .....	29
Program 3 (again) – Squares.m.....	30
EXERCISE 9 .....	33
Topic 13 –More Reading/Writing Data to files .....	33
EXERCISE 6 (again).....	35
Topic 14 – The PsychToolbox.....	36
Topic 15 – Graphics, the good stuff.....	38
Program 7 – Tiling.m.....	38
Topic 16 Visual Angle.....	41
Program 8 – VisAng.m.....	41
Topic 17 Sinusoids and Gaussians .....	43
Program 9 – MakeGrating.m .....	43
Program 10 – GaussianWindow.m .....	45
Program 11 – Shuffle.m.....	46
Topic 18 Calibration .....	47
Program 12 – Calibrate.m.....	47
Program 13 – GammaFit.m .....	49
Program 14 – BestGamma.m .....	49
Program 15 – LtoP.m.....	50

## Topic 1 – What is programming?

*(what is hardware, what is software, what is a computer language)*

Programming is telling a computer what to do. There are only 2 tricky things

- 1) Computers are very stupid – you have to tell them exactly what to do
- 2) Computers don't speak English. Any programming language is a compromise between the computer's native language (0010001) and your native language (English). High level languages are closer to English, low level languages are closer to computer-ese. The closer a language is to English, the easier it tends to be to program, but the slower it is for the computer to interpret it. Low level languages tend to be hard to program but very fast to run. Matlab is a mid- to high level language.

Like any language, programming languages have grammar. Like real languages some things are easier to say in one language rather than another (Italian is the language of love etc. etc.). Some languages are better for computations, others for graphics. Unlike people who speak real languages (with the exception of the French) computers are very fussy about grammatical errors. Like learning a real language at first it will be hard to say the simplest thing, but it will get easier, fast.

**Hardware**

Hardware is the physical presence of the computer. The monitor, the hard drive, the CPU (central processing unit). Hardware is anything you can damage by poking it with a screwdriver.

**Software**

*software = programs*. Programs are instructions to your computer to behave in a particular way. So a software program like Microsoft Office gets all machines to behave in a particular way – the instructions are slightly different for different computers (different hardware), but the program makes all computers behave (almost) the same.

All software is written in a programming language. Some programs, (like matlab) are there to help you write new programs. Matlab will run on Macs, PC, and UNIX. But some of the commands only run on some computers.

**NOTE** – these days it is almost impossible to damage your computer by writing a program. The computer usually makes it very hard for you to do anything that will damage it. In this class you won't be using any commands that can do any permanent damage. So crash your computer hard. Have fun!

## Topic 2– Starting Matlab

(starting Matlab, ,strings, indexing strings, who, using the command window)

If you need to, install Matlab. A month trial version is available from [http://www.mathworks.com/web\\_downloads/](http://www.mathworks.com/web_downloads/)

Double click the Matlab icon. A window will come up. This is your “command window”. You can type the computer commands here. The computer will also send you messages back here, usually messages telling you about problems.

The Matlab command window has a prompt. If I want you to type something in the command window I’ll start it with the prompt (you don’t type the prompt, it’s already there).

Type at the prompt (>)

```
>str='this is my first program' ;
```

What you have done is told the computer to create a list of letters ‘this is my first program’ and name that list of letters “str”. Str is a **variable** - The single quotes tell the computer that it is a list of letters (not numbers, more on that later). A list of letters is called a *string*.

```
>who
```

Your computer will give you a list of all the variables you have. At the moment all you have is str.

```
>str
```

Your computer tells you what is contained within str – it’s a list of letters. Compare these two commands (look carefully, there is a difference.)

```
>str='hello there' ;
```

```
>str='hello there'
```

The semi-colon tells the computer whether or not you want it to display the output of each command.

```
>who
```

```
str is still there
```

```
>str
```

But the list of letters contained within str has changed

```
>str(3)
```

You’ve asked the computer to display the third letter in str. What is the third letter in ‘hello there’?

```
mixstr=str;
```

You've created a new variable called mixstr. You've told the computer to make mixstr the same as str

```
>mixstr
```

```
>str
```

See, they are the same.

```
>mixstr(3)=str(1);
```

Make the 3rd letter in mixstr the same as the 1st letter in str

```
>mixstr(1)=str(3);
```

Make the 1st letter in mixstr the same as the 3rd letter in str

You should now have 'lehlo there'

But we don't want to have to type every command in one at a time. We therefore create a program – a program is simply a sequence of commands. In Matlab most programs are written in files called m-files or functions.

## Topic 3 – Your first program

(creating a file, creating a header, setting the path, running a program, loops, if, clear, disp, lookfor, pause)

### **Program 1 – MixStrings.m**

Click on the command window. Go to File->New->M-file. You'll get a blank document. This will be your new program. Every new program is begun with a few lines of documentation. This is called a header. Good headers look like this:

```
% MixStrings.m
%
% Displays the string 'hello world'
% Then scrambles letters in that string and
% displays the scrambled string
%
% written by IF 5/6/2000
```

OK, type the header into your m-file.

Good headers contain, at the very least

- 1) Name of the function
- 2) Description of what it does
- 3) Who wrote it, and when

Make sure every new line begins with a %. The % tells the computer to ignore that line – these comments aren't for the computer, they're for you.

Save the file. For now save it on the desktop. Call it MixStrings.m (the same as the header).

Go to the command window and type

```
>help MixStrings
```

You may get an error message from your computer that says something like this:

```
MixStrings.m not found.
```

The computer has to know where on the computer to look for your file. To tell the computer where to look, you *set the path*. This tells the computer which folders within the computer to look for files within.

Click on the command window and on the menu bar go to:

File->Set Path. A pop-up window will appear  
Choose Add to Path and choose the Desktop.

Then go to File->Save Path. Then exit the pop-up window.

```
>help MixStrings
```

You'll see all the information you typed into the header. Every file has one of these headers (or should). This information is really useful, and will soon be your main source of info. For example:

```
>help disp
>help shuffle
>help pause
>help length
```

You should be able to get a good idea of what these commands do from their headers.

Another useful command is 'lookfor'

Lookfor finds all the m-files that contain a particular word in their header files. It's good for finding useful commands. For example, try

```
>lookfor string
>lookfor random
```

OK, back to work. Type into your m file, under the header:

```
clear;
str='hello there'
mixstr=str;
mixstr(3)=str(1);
mixstr(1)=str(3);
mixstr
```

Now go back to your command window and type:

```
>MixStrings
```

**You have run your first program!!**

Now let's have fun ... adapt your code so it is as follows

```
clear
str='hello there'
mixstr='xxxxxxxxxxxx'
for i=1:length(str)
```

```
    index=i;  
    mixstr(i)=str(index);  
    disp(mixstr);  
    pause  
end;
```

This is called a loop. Loops are very useful in programming (but in Matlab they are slow, you should only use them when you need them). You indent loops to make reading code easier.

### **EXERCISE 1**

- 1) See what `>i=10:-1:1` does. Now make MixStrings work backwards (from the end of the string to the beginning)
- 2) Make MixStrings replace letters in the string randomly

## Topic 4 – Calculations

(*vectors, matrices, ' , +, -, \*, /, .\* , ./, input, elseif, num2str, int2str, round, min, max*)

As well as having lists of characters (strings), you can also have list of numbers. A 1dimensional string of numbers is a vector. A two-dimensional list of numbers is a matrix.

```
>vect=[1 2 4 6 3]
>mat=[ 1 54 3; 2 1 5; 7 9 0]
>mat2=[ 1 54 3
>2 1 5
>7 9 0]
```

Take a look at vect, mat, mat2. As you can see, there is more than one way of entering the same matrix.

```
>vect=[1;2;4]
>vect2=vect'
```

Vectors can be tall instead of long, ' (a single quote) allows you to transpose rows and columns.

What does mat' look like?

You can add a scalar (e.g. 3) to a vector , and you can add & subtract vectors from each other:

```
>vect+3
>vect+vect
>vect-3
```

So far it's easy. However there are two sorts of multiplication and two sorts of division. The first is when you multiply or divide a vector or matrix by a single number – this is the sort of multiplication/division you are familiar with. This is done using the symbol .\*

```
>vect.*3
>mat.*0.5
>vect./2
>vect2=[2;2;4]
>vect.*vect2
```

```
>vect./vect2
>mat./mat'
```

You can also divide one vector by another 'pointwise'. Each element in the first vector is multiplied by the corresponding element in the second vector.

The same thing can be done with matrices.

```
>vect.*vect'
>mat./[ [ 1 54 3 4; 1 2 1 5;1 7 9 0]
```

The vectors or matrices have to be the same shape. If they aren't you'll get an error message

The second kind of multiplication and division is matrix multiplication and matrix division. We'll get to that later.

## EXERCISE 2

1) Make a new program called MixLetters (based on MixStrings) that replaces the numbers in a vector with zeros

### ***Program 2 - Calculations.m***

Create a folder on your desktop and call it Calculations. Add this folder to the path. Create a new m file called Calculations.m and save it in the Calculations folder.

*Note, when you are writing code and a line is longer than your page you can break it using three dots ...*

```
% Calculations.m
%
% Carries out a series of calculations on two numbers
%
% written by Susi Bloggs June 2000

clear;
num1=input('what is the first number ... ');
num2=input('what is the second number ... ');

disp([num2str(num1), '+', num2str(num2), ...
      '=' , num2str(num1+num2)]);
disp([num2str(num1), '-', num2str(num2), ...
      '=' , num2str(num1-num2)]);
disp([num2str(num1), '*', num2str(num2), ...
```

```

'=', num2str(num1.*num2)]);
disp([num2str(num1), '/', num2str(num2), ...
'=', num2str(num1./num2)]);
if ((round(num1)==num1) & ...
(round(num1)==num1))
    disp('num1 and num2 are integers');
elseif (round(num1)==num1)
    disp(['num1 is an integer, num2 rounded = ', ...
int2str(round(num2))]);
elseif (round(num2)==num2)
    disp(['num2 is an integer, num1 rounded = ', ...
int2str(round(num1))]);
end;
disp([num2str(num1), ' to the power of ', ...
num2str(num2), ' is ', num2str(num1.^num2)]);
disp(['the smallest number of ', num2str(num1), ...
' and ', ...
num2str(num2), ' is ', num2str(min(num1, num2))]);

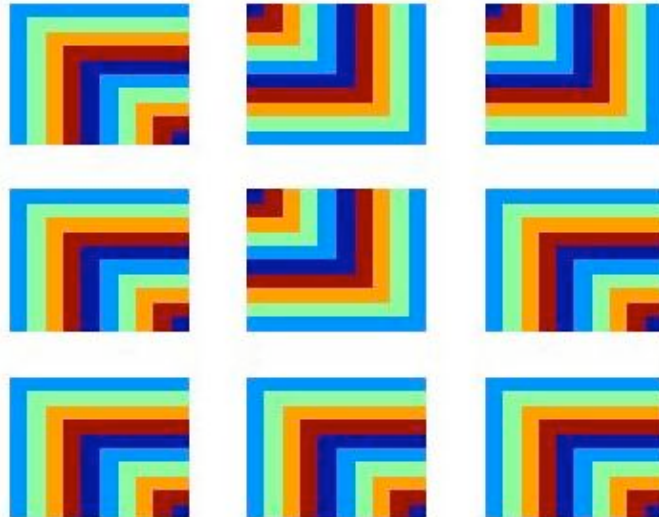
```

### EXERCISE 3

- 1) Add two lines displaying the square roots of num1 and num2 (use >help sqrt)
- 2) Add two lines displaying which of num1 and num2 is the largest number, and which is smallest.  
 "The bigger number is ..." (> help max)  
 "The smaller number is ..."

# Topic 5 –Graphics I

(zeros, linspace, for, indexing matrices, imagesc)



## Program 3 – Squares.m

Create a new m-file called Squares.m

```
% Squares.m
%
% Puts up a figure window with a series of squares
within it.
%
% written by Susi Bloggs June 2000

clear;
im=zeros(100, 100);
for square=[100 75 50 25];
    for rows=linspace(1, 100, 100);
        for cols=linspace(1, 100, 100);
            if((rows<=square) & (cols<=square))
                color=square./25;
                im(rows, cols)= color;
            end
        end
    end
end
```

```
        end
    end
end
imagesc(im);
```

Think about how the program does what it does. Use the command window to see what `im` looks like. Type `help linspace` to find out what `linspace` does

#### **EXERCISE 4**

- 1) Make there be 10 squares instead of 4
- 2) Make there only be 5 colors, not ten (try `>help mod`)
- 3) Change: `if ((rows<=square) & (cols<=square))`  
to `if ((rows<=square) | (cols<=square))`. What's going on?
- 4) Make the squares start from the top right instead of the top left  
*HINT: Play with this line of code: `im(rows, cols) = color;`*

## Topic 6 – Functions

*(functions, flipud, fliplr, ==, rand, subplot, axes, flags)*

A function is a self-contained block of code that performs a coherent task of some kind. It's like outsourcing in business - sending a task away to be performed somewhere else. Functions make code neater – every time you want to do that task you simply call that function.

When you send a task to a function, you give the function all the variables it needs to know, and it returns the variables that you want. The calculations within the function are hidden. We send materials to Taiwan, they make the sneakers, they send back the sneakers and we really don't know much about what happened in Taiwan.

We are going to turn Squares.m into a function. You might want to save a copy of Squares.m as something else first. We have to tell matlab that this is a function. You do that with the first line of code. Modify the first lines of Squares.m so it looks as follows

```
function im=Squares(flag)
% function im=Squares(flag)
%
% Puts up a figure window with a series of squares
within it.
% Takes as input a flag, saying whether the squares
start from the top left or top right
% Returns the image (of squares) as output
%
% written by Susi Bloggs June 2000
```

Then replace the line  
`imagesc(im)`

with:

```
if (flag==0)
    im=flipud(fliplr(im)); % There was an easier
way!
elseif(flag~=1)
    disp('sorry flag must be 0 or 1');
end;
```

```
imagesc(im);
```

That's it. You call Squares.m by typing Squares in the command window, but this time you have to give it input (the flag) which can be either 0 or 1.

```
>Squares(0);
```

also try Squares(1) and Squares(9). Now type

```
>clear
```

```
>myim=Squares(0);
```

```
>who
```

Notice that myim is the only variable shown. That's because functions hide their calculations. You can only see variables that you have specified as output. If you don't assign them a specific name (myim) they are called ans (for answer).

Actually you have been using functions already, many of the commands you have been using already are functions – e.g. round – you give it a number as input and it returns the closest integer as output.

## EXERCISE 5

1) Check out

```
>subplot(2, 2, 1)
```

```
>imagesc(Squares(0))
```

```
>axes(off)
```

```
>rand
```

Now write a program, Tiling.m, that creates a picture like this – 3 x 3 randomly oriented “squares” patterns.

2) Your program is probably very slow. See if you can speed it up a bit

*HINT : Squares.m is very slow. You don't want to call it any more often than you have to.*

3) Turn Tiling.m into a program where all but 1 (randomly chosen) square are pointing in the same direction

# Topic 7 – Reading/Writing Data to files

(*save, load, savesas, fopen, fclose, fprintf, fscanf, %d, %f, %d*)

So far all the experiments you have done haven't saved their output. Obviously you want to be able to do that. There are many ways you can save your data, depending on what the data is.

## **variables**

– in a mat file. A mat file is a saved list of variables. Easy to read/write by Matlab, not readable by any other programs

## **text**

– otherwise known as ascii. Inconvenient to read/write in Matlab. But readable by many programs, including simple text

## **figures**

– you can save your figures, by choosing save in the figure window

Add the following lines to Calculations.m and run it again.

```
%save data as ascii file  
fp=fopen('DataTxt.txt', 'w');  
fprintf(fp, '%f\t%f\n', num1, num2);  
fclose(fp)
```

'w' tells you to open the file for writing rather than reading

the \t adds a tab to your text file

\n adds a newline to your text file

You should be able to open this text file in another program (like excel or simpletext. We'll talk about how to read it into Matlab later (though one of the exercises is to read your data back in).

Saving variables as mat files is even easier. Add the following lines to Calculations.m and run it again.

```
%save variables as a mat file  
save DataMat
```

Reading in a mat file is easy. Just type:

```
>clear
```

```
>who
```

```
>load DataMat
```

```
>who
```

### **Doubles/floats/integers etc**

Numbers can be stored in various ways – in Matlab numbers are automatically stored as doubles unless you ask Matlab to store them differently.

Integers are round numbers

Floats/doubles allow decimals – doubles have better precision than floats. Unless you are trying to save memory you can leave things as doubles.

### **EXERCISE 6**

1) Try reading DataTxt.txt using simple text or some other program.

2) Replace `fprintf(fp, '%f\t%f\n', num1, num2);` with `fprintf(fp, '%.1f\t%.1f\n', num1, num2);`  
- you can choose how many decimal places you want to print.

3) Try printing text - `fprintf(fp, 'hello there');`

4) Try writing a little program for reading in the ascii file. The following commands will help.

```
fopen('filename', 'r');
```

```
num1=fscanf(fp, '%f');
```

*HINT: it's very important that you are specific about what it is (float, character, string that you want to print). Type*

```
>help fprintf
```

*for more information.*

# Topic 8 – Planning a program

You should be feeling almost ready to start work on your own program at this stage.

## *Example program:*

Observers are less sensitive to oblique lines than to vertical/horizontal lines. Suppose I am studying the oblique effect in the periphery. I want to run experiment where subjects fixate (observers have to carry out a fixation spot task) in the center of a display. A faint contrast Gabor will appear on either side, observers have to say on which side of the fixation spot the Gabor appeared. I'm looking at the effect of the orientation of the Gabor.

## **Information to begin the program**

- 1) date
- 2) subject initials
- 4) session number

## **Stimulus Parameters**

### Display

mean luminance (30 cd/m<sup>2</sup>)  
Stim appears for 1 second

### Gabors

Spatial frequency (3 c/deg)

Possible directions ([0 30 90 120 150], 0 = up)

Contrast (.1-10%)

Size (standard deviation) Gabor apertures (4 deg)  
Position dist fix spot (7 deg to center Gabor)

### FixSpot

changes from Circle/Square  
.5 deg size  
80 cd/m<sup>2</sup>  
Changes 1/10 trials

## **Conditions**

5 different orientations

appears left vs right

## **Task**

Press f or j depending on whether Gabor right or left

Press space bar instead if the fixation spot changed

Contrast adjusted by a 3up 1 down procedure (every time the observer gets the right side 3 times in a row the contrast is decreased, every time the observer gets the side wrong the contrast is increased)

### **Raw data collected during program**

for each trial ...

- 1) trial number
- 2) Gabor right or left
- 3) contrast Gabor
- 4) orientation Gabor
- 5) Observer think right or left?
- 6) Correct incorrect? (from 2 and 5)
- 7) did the fixation spot change on that trial?
- 8) did observers see a change in fixation spot?

What should the data file contain:

#### **Mock raw data file**

3/4/2000

gmb session 4

1	0	0.3	90	1	0	0	0
2	1	0.5	45	1	1	0	0
3	1	.5	270	X	X	1	1

and so on ...

#### **Data Analysis**

Need to fit Weibull functions and find 75% threshold for each orientation.

## Topic 9 – Starting Matlab (revisited)

*(starting Matlab, ,strings, indexing strings, who, using the command window)*

Some of this is repetition since people who knew other languages are starting on this chapter. If you started at the beginning, much of this will be familiar. Don't simply skip it though, there's some new stuff in there and many of the definitions are a little more technical.

Start Matlab and a command window will come up.

The Matlab command window has a prompt. You can enter commands directly into the command window. E.g.

```
>str='hello world'
```

```
>str(3)
```

If you add a semicolon the output of each line isn't displayed in the command window, otherwise it is displayed.

Matlab programs are written in something called an m file (e.g, MixString.m). Matlab compiles on the fly – there is no need to compile programs. It therefore acts as an interpreted language.

You can also write c code and compile them so that they can run in Matlab – these are called mex files. But almost things that you want to do you will be able to do in Matlab. Only very speed intensive things will require C.

Click on the command window and go to the menu bar and choose File->New->M-file. You'll get a blank document. This will be your new program.

In every new program you begin it with a few lines of documentation. This is called a header. Good headers contain the information 1) Name of the function 2) Description of what it does 3) Who wrote it, and when  
NOTE – comments have a % preceding them – makes matlab ignore those lines

```
% MixStrings.m
```

```
%
```

```
% Displays the string 'hello world'
```

```
% Then scrambles letters in that string and
```

```
% displays the scrambled string
```

```
%
```

```
% written by IF 5/6/2000
```

OK, type the header into your m-file.  
Make sure every new line begins with a %. The % tells the computer to ignore that line – these comments aren't for the computer, they're for you.  
Save the file. For now save it on the desktop. Call it MixStrings.m (the same as the header)

Go to the command window and type

```
>help MixStrings
```

You may get an error message from your computer

```
MixStrings.m not found.
```

The computer has to know where to look for your file. The computer can read files that are in the computer's current directory, or that are in the computer's search path.

To tell the computer where to look, you set the path. There are two ways to set a path.

#### **Setting the path via the menu bar**

Click on the command window and go to the menu bar and choose:

File->Set Path. A pop-up window will appear

Choose Add to Path and choose the Desktop.

Then go to File->Save Path. Then exit the pop-up window.

#### **Setting the path in your code**

```
>p=path;
```

```
>path(p, 'c:\Winnt\Profiles\Fine\Desktop')
```

Adds c:\Winnt\Profiles\Fine\Desktop to the current path. Other useful commands are

```
>pwd
```

tells you what your current directory is

```
>cd
```

allows you to change your current directory

```
>which MixStrings
```

tells you where a particular file is

**NOTE: if there are two files with the same name, and your path allows the computer to see both of them you won't get a warning, the computer will just use the first one it finds.**

So, you either have to be careful about naming, and never use the same name twice, or you have to be careful about your path, set it for each program so it can only see the files that are really relevant to that program. The command which will tell you which file the computer is using.

Matlab contains a large number of functions contained within toolboxes. There are two ways to find these functions.

### **lookfor**

Lookfor finds all the m files that contain a particular word in their header files. For example, try

```
>lookfor string
>lookfor random
```

### **help**

help gives you details about the function, and how to use it. Typing

```
>help shuffle
```

causes you to see the header file for that function. Bear that in mind when you are writing header files and try to include suitable keywords

```
>help disp
>help shuffle
> help pause
>help length
```

OK, back to work. Add after your header:

```
clear
str='hello there'
mixstr='xxxxxxxxxxxx'
for i=1:length(str)
    index=i;
    mixstr(i)=str(index);
    disp(mixstr);
    pause
end;
```

Now go back to your command window and type:

```
>help MixStrings
>MixStrings
```

You have run your first program!!

NOTE: Loops are slow in Matlab, and should be avoided where possible, more on that later.

# Topic 10 – Vectors, Matrices, Calculations, Operators

*(int, char, float, vectors, matrices, ', +, -, \*, /, .\* , ./, input, elseif, num2str, int2str, round, min, max, questdlg, inputdlg, strcmp, ==, &, |, ~, xor)*

Matlab does not require you to specify whether a number is a char, int, float etc. Numbers are normally doubles, though you can specify them to be unsigned 8bit integers if you need to save memory using the command uint8.

To assign variables

```
>int=8.2;
```

sets the variable int to equal 8.2

```
>str='8.2'
```

creates a string containing "8.2"

Matlab works using vectors and matrices. A 1dimensional string of numbers is a vector. A two-dimensional list of numbers is a matrix.

Square brackets [ ] are used for matrices.

( ) are used for phrasing.

You can output the content of a variable by typing it's name without a semi-colon or typing:

```
>disp(variable)
```

```
>who
```

tells you which variables are currently in memory

```
>clear
```

removes all variables from memory

## Entering and referencing vectors

There are lots of ways of entering vectors into Matlab

```
>a=[ 1 2 3 4 5 6 7 8 9 10]
```

```
>a=1:1:10
```

```
>a=10:-1:1
```

```
>a=1:10
```

```
>a=linspace(1, 10, 10)
```

```
>b=a'
```

The single quote is a transpose - flips a row vector to a column vector or vice versa

You can reference a particular element in a vector. It doesn't matter whether the vector is oriented horizontally (a row vector) or vertically (a column vector)

```
>a(3)
```

```
>b(3)
```

You can also concatenate vectors (or matrices)

```
>c=[1 2 3]
```

```
>d=cat(2, a, c)
```

The 2 in the cat, tell matlab to concatenate the vectors along the 2<sup>nd</sup> dimension – i.e. concatenate the columns.

### **Entering and referencing matrices**

Again, there are several ways to enter matrices

```
>m=[ 1 2 3; 4 5 6; 7 8 9]
```

```
>m=[1 2 3
```

```
>4 5 6;
```

```
>7 8 9];
```

Matrices are 2 dimensional, the first dimension refers to the vector's row. The second dimension refers to the vector's column

```
>m(1, 3)
```

```
>m(3, 1)
```

```
>m(:, 1)
```

refers to every element in the first column

```
>m(1, :)
```

refers to every element in the first row

```
>n(1, :)=1:3
```

```
>n(2, :)=4:6
```

```
>n(3, :)=7:9
```

```
>nt=n'
```

transposing a matrix flips the rows to be columns, and vice versa.

```
>clear
```

```
>a=[1;2;4]
```

```
>b=a'
```

Vectors can be tall instead of long, ' (single quote) allows you to transpose rows and columns. What does mat' look like?

You can add or subtract a scalar (e.g. 3) to a vector. You can add vectors to each other if they are the same shape.

```
>c=a+3
```

```
>c=a-3
```

```
>d=a+c
```

But not if they aren't the same shape

```
>e=a+b
```

So far it's easy. However there are two sorts of multiplication and two sorts of division. The first kind is when you multiply or divide a vector or matrix by a single number – this is the sort of multiplication/division you are familiar with. This is done using the symbol `.*`

```
>e=a .* 3
```

```
>d=a .* 0.5
```

You can also divide one vector by another 'pointwise'. Each element in the first vector is multiplied by the corresponding element in the second vector

```
>f=a ./ 2
```

```
>clear
```

clear deletes all variables

```
>a=[1; 2; 4]
```

```
>b=[2; 2; 4]
```

```
>c=a .* b
```

```
>d=a ./ b
```

The same thing can be done with matrices.

```
>o(1, :)=3:-1:1
```

```
>o(2, :)=4:-1:2
```

```
>o(3, :)=5:-1:3
```

```
>m./o
```

```
>a .* b'
```

```
>p= [ 1 5 3 4; 1 2 1 5; 1 7 9 0]
```

```
>m./p
```

The vectors or matrices have to be the same shape. If they aren't you'll get an error message

The second kind of multiplication and division is vector multiplication/division and matrix multiplication/division.

### Outer product

$A=B*C$  – the number of **rows** in **B** must be equal to the number of **columns** in **C**.  
Actually matrix multiplication isn't used very often in experimental programming, though it's useful for making filters. |

```
>p=[2 3 4 3 2];
```

```
>q=[2 3 4 3 2]';
```

```
>size(p)
```

tells you the size of p, rows then columns

```
>size(q)
```

```
>q*p
```

outer products

```
>p'*q'
```

### Inner product

The number of **columns** of **B** must equal the number of **rows** of **C**

```
>p*q
```

```
>q*p'
```

### 3d-4d matrices

Matrices in matlab (Matlab 5 onwards) can also be 3, 4 or more dimensions, though some commands won't work for matrices with more than 2 or three dimensions

```
>mat(1, :, :) =[2 4 3 ; 5 6 7];
```

```
>mat(2, :, :) =[1 3 2; 6 1 2];
```

```
>size(mat)
```

```
>mat
```

NOTE: all matrices in matlab are really kept as vectors. Sometimes it's useful to convert back and forth

```
>vect=mat(:)
```

string the matrix out as a vector

```
>ind=sub2ind(size(mat), 2, 3)
```

calculates the index in vect corresponding to the 2<sup>nd</sup> row and 3<sup>rd</sup> column of mat.

So:

```
>vect(ind)
```

```
>mat(2,3)
```

Check out ind2sub

## **Program 4 (again) - Calculations.m**

Create a folder on your desktop and call it Calculations. Add this folder to the path. Create a new m file called Calculations.m and save it in the Calculations folder.

Note, when you are writing code and a line is longer than your page you can break it using three dots ...

```
% CalculationVector.m
%
% Carries out a series of calculations on a vector
%
% written by Susi Bloggs June 2000

clear;
vect1=input('what is the first vector [1 2 3]... ');
vect2=input('what is the second vector [5 6 7]... ');
if (length(vect1)==length(vect2))
    disp(['vector 1 + vector 2 = ', num2str(vect1+vect2),
    '']);
    disp(['vector 1 - vector 2 = ', ...
    num2str(vect1- vect2), '']);
    disp(['vector 1 .* vector 2 = ', ...
    num2str(vect1.*vect2)]);
    disp(['vector 1 ./ vector 2 = ', ...
    num2str(vect1./vect2)]);
    disp(['vector 1 to the power of vector 2= [ ' , ...
    num2str(vect1.^vect2), ' ']);
    disp(['mean vector 1 = ', num2str(mean(vect1))]);
    disp(['sum( vector 2 = ', num2str(sum(vect2))]);
else
    disp('sorry, vectors are different lengths');
end;
```

## **EXERCISE 7**

1) Add two lines displaying the square roots of num1 and num2 (use >help sqrt)

2) Add two lines displaying which of num1 and num2 is the largest number, and which is smallest.

“The bigger number is ...”, (> help max)

“The smaller number is ...”

3) find the cos and the sin of vect1 and vect2, and display them in degrees (not radians)

### **Operators**

assign the value b to a

>a=b

check whether a is equal to b – if it is, returns 0, otherwise 1

>a==b

1 if both a and b are non zero, 0 otherwise

>a & b

1 if either a or b are nonzero, 0 otherwise

>a | b

exclusive or 1 if either a or b are non-zero, but 0 if both a and b are nonzero, 0 otherwise

>xor(a, b)

1 if a greater than b, 0 otherwise

>a>b

1 if a greater or equal to b, 0 otherwise

>a>=b

1 if a not equal to b, 0 otherwise

>a~=b

### ***Program 6 – CosmoSexQuiz.m***

```
%CosmoSexQuiz.m
%
% A program that uses a series of questions to determine
% whether your relationship will
% survive. Based on firm psychological principles.
%
% written by Dr. Ruth 6/4/2000

score=0;
gender=questdlg('Are you male or female?', ...
'Question', 'Male', 'Female', 'Male');

like=questdlg('Do you like your partner?', ...
'Question', 'Yes', 'No', 'No');
if strcmp(like, 'Yes')
    score=score+2;
else
    score=score-2;
end;
```

```
unfaithful=questdlg('Are you sleeping with anyone else?', ...
'Question', 'Yes', 'No', 'No');
if (strcmp(unfaithful, 'Yes') & strcmp(gender, 'Male'))
    score=score+3;
elseif (strcmp(unfaithful, 'Yes') & strcmp(gender,
'Female'))
    score=score-5;
else
    score=score+1;
end

str=(['Your score = ', num2str(score), ...
'. Look for someone new']);
questdlg(str, 'Answer', 'Quit', 'Quit');
```

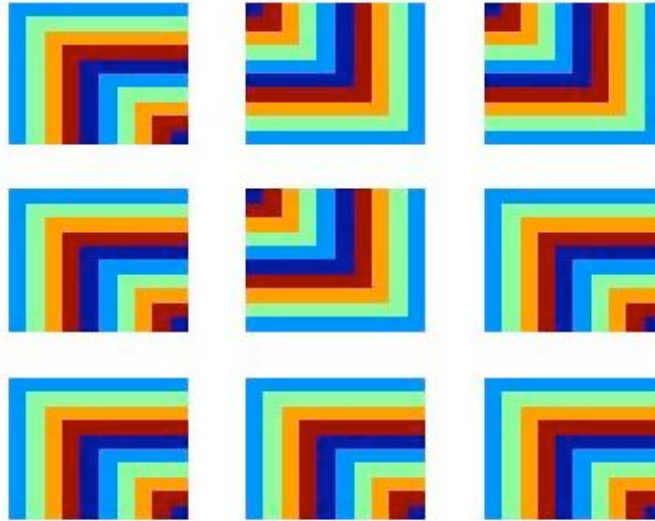
## **EXERCISE 8**

Continue CosmoSexQuiz, and use all the operators described above.

# Topic 12 –Graphics Again

(zeros, linspace, for, indexing matrices, image, Figure, subplot, image  
fliplr, flipud, colormaps)

There are many ways of displaying a 2d matrix.



## **Program 3 (again) – Squares.m**

Create a new m-file called Squares.m

```
% Squares.m
%
% Puts up a figure window with a series of squares within
it.
%
% written by Susi Bloggs June 2000

clear;
sz=100;
im=zeros(sz,sz);
for square=[100 75 50 25];
    for rows=linspace(1, sz, sz);
        for cols=linspace(1, sz, sz);
            if((rows<=square) & ...
                (cols<=square))
color=square.*(256/sz);
                im(rows, cols)= color;
            end
        end
    end
```

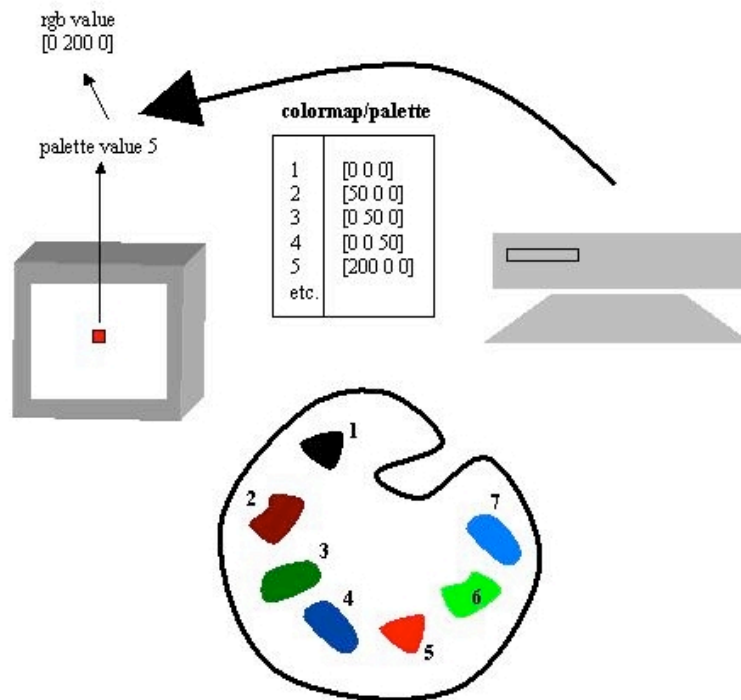
```

end
end;
colormap(hot);
imagesc(im);

```

Use the command window % to see what im looks like  
 Type help linspace to find out what linspace does  
 Use Edit->Pretty Indent in the menu bar to sort out indenting.

## ColorMaps



Two ways of changing color on the screen

- 1) Change the palette value of that pixel
- 2) Change the rgb triplet referenced by that palette value.

Every pixel in your monitor has three guns – red, green, blue.  
 Each gun takes an intensity value.

**Restriction 1.** In most monitors the intensity value of each gun varies between 0-255. Most monitors are 8 bit monitor. A bit is 2 alternatives. Monitors can take  $2^8$  intensity values. (Some monitors are 10 bit - each gun has 1024 possible intensity values).

**Restriction 2.** Most monitors can only display 256 colors at a time. These possible 256 colors are listed in the colormap/palette. Those 256 colors can be

any color possible, (given Restriction 1). Each palette/colormap color is defined by 3 numbers (for the red, green and blue guns) and 256 rows.

Each number in the matrix `im` refers to a number in the colormap/palette.

Matlab has some predefined colormaps

```
>colormap(hot)
>colormap(bone)
>colormap(gray(255))
and you can make your own
>mygray=[0:(1/255):1; ...
>0:(1/256):1; ...
0:(1/255):1]';
>invmygray=1-mygray;
>size(mygray)
>colormap(mygray)
>colormap(invmygray)
```

Add the following lines to your program

```
for i=1:5
    colormap(mygray);
    pause
    colormap(invmygray);
    pause
end;
```

Note that you made these changes to the image on the screen just by changing the colors in your palette – not by changing what you drew on the screen.

There are two versions of image

```
>image(im) % uses values in matrix. (Values
between 0-255)
>imagesc(im) % scales image to use the full
colormap (be wary)
```

## EXERCISE 9

1) Make there be 10 squares instead of 4

2) Make there only be 5 colors, not ten (try >help mod)

3) Change:

```
if ((rows<=square) & (cols<=square))  
to  
if ((rows<=square) | (cols<=square)).  
What's going on?
```

4) Make the squares start from the top right instead of the top left

*HINT: \_Squares.m is very slow. You don't want to call it any more often than you have to.*

## Topic 13 –More Reading/Writing Data to files

*(save, load, savesas, fopen, fclose, fprintf, fscanf, %d, %f, %d)*

So far all the experiments you have done haven't saved their output. Obviously you want to be able to do that. There are many ways you can save your data.

### variables

– in a mat file. A mat file is a saved list of variables. Easy to read/write by Matlab, not readable by any other programs

Saving/reading in a mat file is easy. Just type:

```
>var=rand(3);  
>who  
>save DataMat  
>clear  
>who  
>load DataMat  
>who
```

### text

– otherwise known as ascii. Inconvenient to read/write in Matlab. But readable by many programs, including simple text

#### **fprintf**

fprintf behaves like ANSI C with certain exceptions:

1. Only the real part of each parameter is processed.
2. Non-standard subtype specifiers are supported for conversion characters o, u, x, and X.
3. fprintf is "vectorized" for the case when A is nonscalar. The format string is recycled through the elements of A (columnwise) until all the elements are used up. It is then recycled in a similar manner through any additional matrix arguments.

For example:

```
x = 0:.1:1; y = [x; exp(x)];
fid = fopen('exp.txt','w');
fprintf(fid, '%6.2f %12.8f\n', y);
fclose(fid);
```

creates a text file containing a short table of the exponential function:

```
0.00 1.00000000
0.10 1.10517092
...
1.00 2.71828183
```

Have a look at this file using a text program (e.g. excel or simpletext)

### **fscanf**

fscanf is also "vectorized" in order to return a matrix argument. The format string is recycled through the file until an end-of-file is reached or the amount of data specified by SIZE is read in.

### **Doubles/floats/integers etc**

Numbers can be stored in various ways – in Matlab numbers are automatically stored as doubles unless you ask Matlab to store them differently.

Integers are round numbers

Floats/doubles allow decimals – doubles have better precision than floats. Unless you are trying to save memory you can leave things as doubles.

– Add the following lines to Calculations.m

```
%save data as ascii file
fp=fopen('DataTxt.txt', 'w');
fprintf(fp, '%f\t%f\n', num1, num2);
fclose(fp)
```

'w' tells you to open the file for writing % rather than reading

the \t adds a tab to your text file

\n adds a newline to your text file

Also add a line saving the data as a mat file  
%save variables as a mat file  
save DataMat

### figures

– you can save your figures, by choosing save in the figure window.

### EXERCISE 6 (again)

- 1) Try reading DataTxt.txt using simple text or some other program.
- 2) Replace `fprintf(fp, '%f\t%f\n', num1, num2);` with `fprintf(fp, '%.1f\t%.1f\n', num1, num2);`  
- you can choose how many decimal places you want to print.
- 3) Try printing text - `fprintf(fp, 'hello there');`
- 4) Try writing a little program for reading in the ascii file. The following commands will help.  
`fopen('filename', 'r');`  
`num1=fscanf(fp, '%f');`  
*HINT: it's very important that you are specific about what it is (float, character, string that you want to print). Type*  
`>help fprintf`  
*for more information.*

# Topic 14 – The PsychToolbox

<http://color.psych.ucsb.edu/psychtoolbox/>

PsychToolbox is a collection of matlab functions written to make presenting visual stimuli easier. Remember to cite the Toolbox.

"We wrote our experiments in MATLAB, using the Psychophysics Toolbox extensions (Brainard, 1997; Pelli, 1997)."

Brainard, D. H. (1997) The Psychophysics Toolbox, *Spatial Vision* , 10:443-446.

Pelli, D. G. (1997) The VideoToolbox software for visual psychophysics: Transforming numbers into movies, *Spatial Vision* 10:437-442.

If you use the PC version you should also thank Xuemei (Mei) Zhang.

## ***Website Explanation***

The Psychophysics Toolbox is a free set of MATLAB functions for doing visual psychophysics (Brainard, 1997; Pelli, 1997). The software runs in the MATLAB numerical programming environment. It's compatible with MATLAB 5 and Student MATLAB 5, or better. A polished version is available for Macintosh computers, and a rough alpha version is available for Windows. Complete sources are provided. Some of the functions are written in MATLAB, but the key routines are MATLAB extensions (MEX or DLL), which are called as high-level MATLAB functions but written in C. (Users of the Psychophysics Toolbox do not need to know C.) Many of these extensions provide access to Denis Pelli's (1997) VideoToolbox, a collection of C subroutines for doing psychophysics on Macintosh computers.

We now use this environment for all our experiments and modeling. The key test, for us, is how long it takes a new student to implement a new experiment. Canned programs fail because they usually can't do a really new experiment. In C, it generally takes six months (including learning C). In MATLAB, with the Psychophysics Toolbox, it takes a few weeks (including learning MATLAB).

An email poll of users revealed that the toolbox has been used to measure a variety of psychophysical thresholds (e.g. detection of gratings and letters in noise), to display stimuli for functional MRI and electrophysiological experiments, to measure asymmetric color matches, to evaluate image compression algorithms, and to study categorization, perceptual learning, visual search, and visual object recognition.

Version 1 (for Mac only, released 1995) was written by David Brainard, with some help from others. Version 2 (released 1996 for Mac, released 2000 for Windows) was written by David Brainard and Denis Pelli, with help from others, and is actively maintained (see Changes). (See credits for Mac and Windows versions.) Send questions, comments, and requests for update notification to [psychtoolbox@psych.ucsb.edu](mailto:psychtoolbox@psych.ucsb.edu).

You'll have to download Psychtoolbox from the web site. Make sure you download the right version (for Mac or PC).

*Put the PsychToolbox in the toolbox folder in Matlab. Remember to add the psychtoolbox folder to your path.*



```

% change what's on the screen by
% drawing directly on screen using PutImage

if strcmp(answer, 'Draw Directly');
    SCREEN(window, 'DrawText', ...
'Drawing directly onto screen',100,100,127);
    pause;

    for i=1:20
        screen(window, 'WaitBlanking');
        screen(window, 'PutImage', ...
            squeeze(square(1, :, :)), [0 0 1280 1024]);
        pause(.1);
        screen(window, 'WaitBlanking');
        screen(window, 'PutImage', ...
            squeeze(square(2, :, :)), [0 0 1280 1024]);
        pause(.1);
    end
    SysBeep
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% draw on an offscreen window, then
% copy quickly to the main window
if strcmp(answer, 'Copy from Offscreen');
    SCREEN(window, 'DrawText', 'Wait ... ',100,100,127);
    for i=1:20
        w(1,i)=SCREEN(window, 'OpenOffscreenWindow');
        w(2,i)=SCREEN(window, 'OpenOffscreenWindow');
        screen(w(1,i), 'PutImage', ...
            squeeze(square(1, :, :)), [0 0 1280 1024]);
        screen(w(2,i), 'PutImage', ...
            squeeze(square(2, :, :)), [0 0 1280 1024]);
    end
    screen(window, 'PutImage', ...
        squeeze(square(2, :, :)), [0 0 1280 1024]);
    SCREEN(window, 'DrawText', ...
        'Copying from Offscreen',100,100,127);
    pause;
    for i=1:20
        SCREEN(window, 'WaitBlanking');
        SCREEN('CopyWindow',w(1,i),window);
        pause(.1)
        SCREEN(window, 'WaitBlanking');
        SCREEN('CopyWindow',w(2,i),window);
    end
end;

```

```

        pause(.1)
    end
    SysBeep;
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%change what's on the screen by changing the clut/palette
if strcmp(answer, 'CLUT change');
    screen(window, 'PutImage', ...
        squeeze(square(2, :, :)), [0 0 1280 1024]);

    SCREEN(window, 'DrawText', 'Changing CLUT', 100, 100, 127);
    pause
    screen(window, 'PutImage', ...
        squeeze(square(2, :, :)), [0 0 1280 1024]);

    for i=1:20
        screen(window, 'SetClut', invmygray);
        screen(window, 'SetClut', mygray);
    end;
    sysbeep;
end;

screen('CloseAll');
ShowCursor;
clear mex;

```

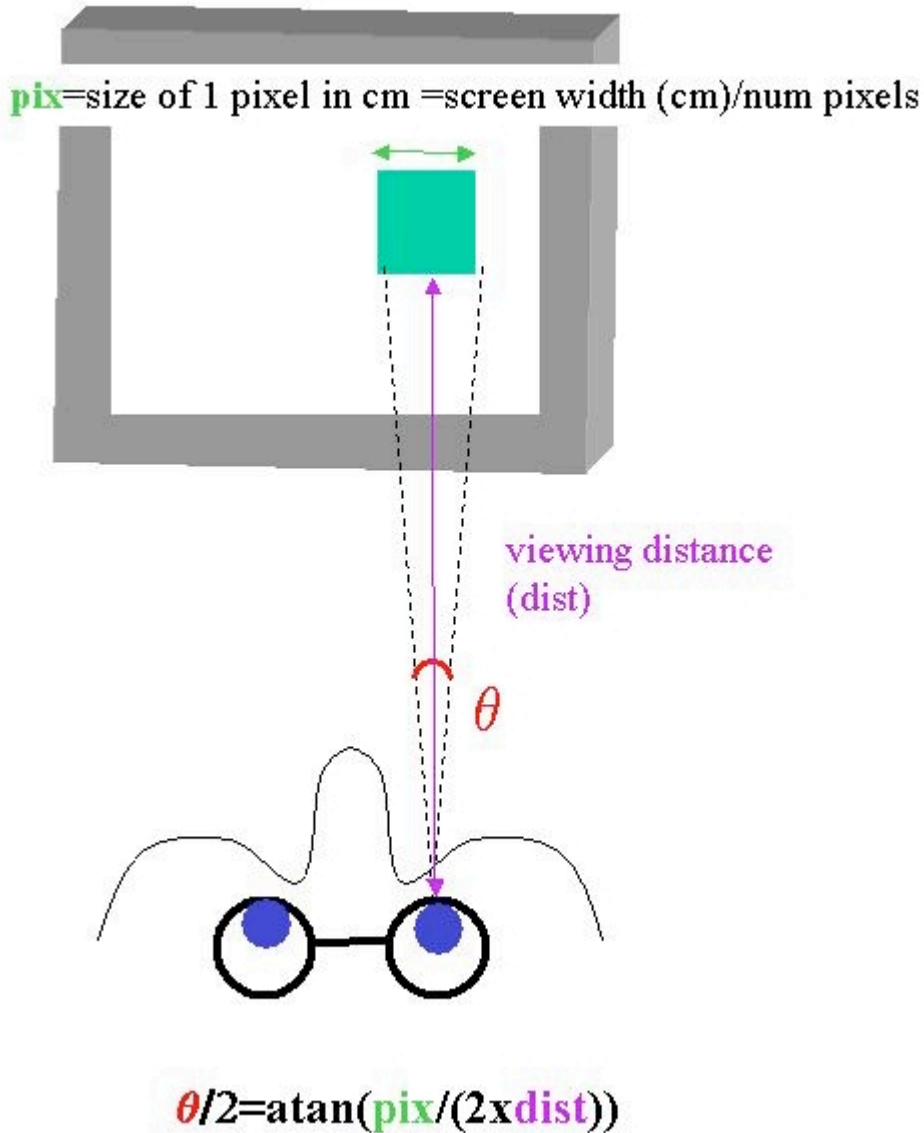
Run this program and try the different techniques. You should notice that some are faster than others

**NOTE:**

**Colormaps**

Matlab's "colormap" command - colormaps should vary between 0-1.  
 PsychToolbox - 'SetClut' wants values varying between 0-255, while 'PutImage' takes values between 1-256. This is an idiosyncrasy due to C indexing from 0 while Matlab indexes from 1.

## Topic 16 Visual Angle



Most stimuli in vision science are measured in degrees of visual angle – i.e. the angle subtended by that stimulus at the eye. To calculate the visual angle subtended by a stimulus involves two stages (make sure you use the same units, e.g. cm for both stages).

### **Program 8 – VisAng.m**

```
function [pixperdeg, degperpix]=VisAng(params)
% function [pixperdeg, degperpix]=VisAng(params)
```

```

%
% Takes as input a structure containing:
%   struct.res - the resolution of the monitor
%   struct.sz  - the size of the monitor in cm
% (these values can either be along a single dimension or
% for both the width and height)
%   struct.vdist - the viewing distance in cm.
%
% Calculates the visual angle subtended by a single pixel
%
% Returns the pixels per degree
% and it's reciprocal - the degrees per pixel (in degrees,
not radians)
%
% written by IF 7/2000

pix=params.sz./params.res; %calculates the size of a pixel
in cm
degperpix=(2*atan(pix./(2*params.vdist))).*(180/pi);
pixperdeg=1./degperpix;

```

### 1) Calculate the width of a single pixel in cm.

- (i) Measure the width and height of the screen (there is usually a black rim, don't measure that) in cm
- (ii) Find out the number of pixels for that monitor horizontally and vertically
  - width of single pix in cm=width screen/num horizontal pixels
  - height single pixel in cm=height screen/num vertical pixels

It's possible that your pixels aren't square. It may be that you don't care. If you do, then adjust your monitor to make the pixels squarer. If your pixels still aren't square then you may have to have different values for the number of pixels per degree in the horizontal and vertical directions.

### 2) Calculate the visual angle subtended by a single pixel

$\theta = \text{atan}(\text{pix}/\text{viewing distance})$

This gives you two values  $\theta = \text{degperpix}$ , and the reciprocal,  $1/\theta = \text{pixperdeg}$ .

# Topic 17 Sinusoids and Gaussians

The easiest way to make a grating is to make a ramp of values oriented in the orientation of your grating, and then take the sin of the ramp.

Start by looking at this code without it being a function, (comment out the first function line, then add lines defining sf, orient, sz and pixperdeg). *Note, you can always convert a function into an ordinary m-file. That's a good way of allowing yourself to see the working in the calculations (remember, functions hide the intermediate variables from the command window).*

Normally you would calculate pixperdeg (see the previous topic. A value of pixperdeg=60 is fairly realistic.

## **Program 9 – MakeGrating.m**

```
function im=MakeGrating(sf, orient, sz, pixperdeg);
% creates an oriented grating with values varying between -
1 and 1
% takes as input:
%     the spatial frequency of the grating in cpd
%     the orientation of the grating in degrees
%     the sz of the grating in degrees.
%     pixels per degree.

%convert to pixels and radians

sz=round(sz.*pixperdeg);
orient=orient*pi/180;

%create grating
step=pi*2./(sz-1);
[x, y]=meshgrid(-pi:step:pi, -pi:step:pi);
ramp=cos(orient)*x-sin(orient)*y;
im=sin(ramp*sf);
```

### **Displaying gabors**

Putting this grating up onto the screen, as follows (also windowing it with a Gaussian.

Note that code is all written in terms of degrees of visual angle.

```
%OrientatedGabor.m
```

```

%
% Displays an oriented gabor on the screen until a user
keypress.
% uses functions MakeGrating and GaussianWindow
%
% written by if 7/2000
clear

%size screen
params.sizescreen=[31 25]; %width height (cm)
params.viewdist=57; % viewer distance (cm)
params.screenpix=[1280 960]; %pixel resolution
params.degperpix=180/pi*mean(atan((params.sizescreen./param
s.screenpix)./params.viewdist));
params.pixperdeg=1./params.degperpix;
params.contrast=40;

sf=3; %spatial frequency in cpd
orient=15; %orientation in degrees
sz=5; %size in degrees

%create cluts
mygray=[0:1:255; 0:1:255; 0:1:255]';
%note - unlike image uses from 0-255
blank=repmat(127, 256, 3);
%which screen
screenNumber=0;

im=MakeGrating(sf, orient, sz, params.pixperdeg);
im=GaussianWindow(im, 1, params.pixperdeg);
im=(im*params.contrast)+127;
vals=CenterRect([0 0 size(im)], [0 0 size(im)]);

window=screen(screenNumber, 'OpenWindow', 127, [], 8);
screen(window, 'SetClut', blank);
HideCursor;

screen(window, 'PutImage', 127*ones(params.screenpix), ...
[0 0 params.screenpix]);
screen(window, 'PutImage', im, CenterRect([0 0 size(im)],
[0 0 params.screenpix]));
screen(window, 'WaitBlanking');
screen(window, 'SetClut', mygray);

pause;
screen(window, 'SetClut', blank);
screen('CloseAll');

```

```
ShowCursor;  
clear mex;
```

When displaying the matrix `im` in this way the value of each place in the matrix `im` is represented by a luminance intensity.

This function won't run just yet – it uses the function `GaussianWindow.m`

### ***Program 10 – GaussianWindow.m***

```
function im=GaussianWindow(im, sd, pixperdeg)  
%  
% windows an square image with a Gaussian filter.  
% takes as input the image, and the standard deviation of  
the  
% filter in the x and y direction (in degrees) and the  
pixels per degree  
% returns as output the filtered image.  
%  
% written by IF 7/2000  
  
filtSize =min(size(im));  
sd=sd*pixperdeg;  
  
[x,y] = meshgrid(round(-filtSize/2): ...  
round(filtSize/2)-1,round(-filtSize/2): ...  
round(filtSize/2)-1);  
filt = exp(-(x.^2+y.^2)/(2*sd.^2));  
filt=filt./max(max(filt(:)));  
im=im.*filt;
```

### **Meshgrid**

Once you have run oriented Gabor try

```
>surf(im)  
>colormap(mygray./max(mygray(:)));  
>shading interp  
>axis([0 200 0 200 90 150]);
```

This produces a figure where the value of each place in the matrix `im` is represented by height in the z-axis as well as by luminance intensity. Receptive fields are often plotted in this way.

You can also change the view from which you view this meshplot. Try:

```
>help view  
>view(0, 30)
```

```
>view(0, 90)
provides a view directly overhead
```

## **Miscellaneous useful functions ...**

### ***Program 11 – Shuffle.m***

```
function [Y,index] = Shuffle(X)
% [Y,index] = Shuffle(X)
%
% Randomly sorts X.
% If X is a vector, sorts all of X, so Y = X(index).
% If X is an m-by-n matrix, sorts each column of X, so
% for j=1:n, Y(:,j)=X(index(:,j),j).
%
% Also see SORT, Sample, and Randi.
%
% xx/xx/92 dh Brainard Wrote it.
% 10/25/93 dhb Return index.
% 5/25/96 dgp Made consistent with sort and "for
i=Shuffle(1:10)"
% 6/29/96 dgp Edited comments above.
[null,index] = sort(rand(size(X)));
Y = X(index);
```

# Topic 18 Calibration

There are three ways to calibrate:

- 1) Calibrate for luminance with the red/green/blue guns having the same gun value. Tends to give you a chromaticity that varies little with luminance, but may not be perfect gray. Typical values might be (x=.34 y=.35 z=.31)
- 2) Calibrate for grayscale. Adjusts the relative intensity of red/green/blue guns so as to get perfect gray.
- 3) Full chromatic calibration allowing you to display any color at any luminance. The code below covers 1, and can easily be extended to 2. Full chromatic calibration is tricky, there are tools to do it in the PsychToolbox.

## Measurement

To calibrate first requires measuring the luminance for each gun value:

### *Program 12 – Calibrate.m*

```
%Calibrate.m
%
% puts up a square of known gun
% value in the center of the screen.
% Can then measure the luminance of the square
% for each gun value

clear
type = questdlg('what sort of calibration?', ...
'question', 'gray' , 'color', 'color');
if strcmp(type, 'color')
    type = questdlg('what sort of calibration?', ...
'question','red', 'green', 'blue', 'blue');
end;
%size screen
params.sizescreen=[31 25]; %width height (cm)
params.viewdist=57; % viewer distance (cm)
params.screenpix=[1280 960]; %pixel resolution
params.degperpix=180/pi*mean(atan((params.sizescreen. ...
./params.screenpix)./params.viewdist));
params.pixperdeg=1./params.degperpix;
params.sizesquare=4;
%size of central calibration spqare in degrees.

sf=3; %spatial frequency in cpd
orient=15; %orientation in degrees
sz=5; %size in degrees

%create cluts
```

```

if strcmp(type, 'gray')
    myclut=[0:1:255; 0:1:255; 0:1:255]';
%note - unlike image uses from 0-255
elseif strcmp(type, 'red')
    myclut=[0:1:255; zeros(1, 256);zeros(1, 256)]';
elseif strcmp(type, 'green')
    myclut=[zeros(1, 256); 0:1:255; zeros(1, 256)]';
elseif strcmp(type, 'blue')
    myclut=[zeros(1, 256); zeros(1, 256); 0:1:255]';
end;

blank=repmat(127, 256, 3);
%which screen
screenNumber=0;

im=ones(round(params.sizesquare*params.pixperdeg), ...
    round(params.sizesquare*params.pixperdeg));
vals=CenterRect([0 0 size(im)], [0 0 size(im)]);

window=screen(screenNumber, 'OpenWindow', 0, [], 8);
screen(window, 'SetClut', blank);
HideCursor;

screen(window, 'WaitBlanking'); screen(window, ...
'SetClut', myclut);
for i=255:-30:0
    screen(window, 'PutImage', ...
zeros(params.screenpix), [0 0 params.screenpix]);
    str=strcat('gun value=', num2str(i));
    SCREEN(window, 'DrawText', str, 100, 100, 255);
    screen(window, 'PutImage', ...
im*i, CenterRect([0 0 size(im)], [0 0 params.screenpix]));
    pause;
end;

screen('CloseAll');
ShowCursor;
clear mex;

```

### Gamma Fitting

For most monitors the luminance as a function of gun value tends to follow a gamma function where  $L=a*\text{gun value}.^b$ . This function fits a gamma function for gray monitor calibration.

## **Program 13 – GammaFit.m**

```
% GammaFit.m
%
% Finds the best fitting gamma function and then creates a
colormap
% that produces a linear range of values between l_range
% max and min
% uses BestGamma.m
%
% IF 6/2001

clear;
%measured values
i=[1 31 61 91 121 151 181 211 241 252];
Y=[.3 .9 5.9 17.8 37.8 67.1 106 156 218 255];

l_range=[5 80]; %luminance range for which we calibrate in
cd

%get the best fitting gamma function
g=[.2 2];
g=fmins('BestGamma', g,[0],[], i, Y);
plot(i, Y, 'kx', i, g(1)*(i.^g(2)), 'b');

%the range of luminance values desired
dY_vals=linspace(l_range(1), l_range(2), 256);
%the gun_values at each lum step.
gun_v=round(spline( g(1)*(i.^g(2)), i, dY_vals));

temp.pwdir=pwd;
ans=inputdlg(pwd, 'Where do you want to save your
colormap', 1, cellstr(pwd));
cell2struct(ans, 'dir',1);
cd(ans.dir);

clrmap= repmat(gun_v, 3, 2)';
clrmap=(clrmap./255)';
save clrmap
```

## **Program 14 – BestGamma.m**

The function that finds the best fitting gamma function

```
function mse=BestGamma(g,i, Y)
%g=function BestGamma(i, Y)
%
% finds the best fitting gamma function
```

```

% takes as input the gunvalues and the measured luminance.
% returns the best value of gamma

Yfit=g(1)*(i.^g(2));
mse=sum((Yfit-Y).^2);

```

But we still need to know which palette value corresponds to which luminance

### ***Program 15 – LtoP.m***

```

function pal=LtoP(lum)

%put in the luminance value. Returns the
%palette value

load clrmap
nlum=(lum-l_range(1))/(l_range(2)-l_range(1));
pal=round((nlum*(256-1)));

```

#### **Warnings for calibration:**

- 1) if you are calibrating grayscale the chromaticity of the three guns should not change drastically with intensity. It's worth checking especially for very low and high luminances.
- 2) calibration should not be different in different regions of the monitor
- 3) the monitor may take time to warm up and reach stable chromatic values
- 4) the monitors calibration may change over time. Depending on how sensitive your experiment is to contrast you should check calibration 3xweek-1x/month

#### **Calibration Checklist**

- 1) gamma correction – as described above. Intensity is not linear with gun value
- 2) stationarity – different parts of the display calibrate differently (e.g. misconvergence)
- 3) gun drain (phosphor independence) –power of red gun alone may be different from when red, green and blue guns are firing
- 4) power drain (spatial independence) – power of guns may be affected by the extent of the image
- 5) quantization errors – guns only take a certain number of discrete levels, generally 8 bit (256 different levels)
- 6) flicker - a display of 80cd/m<sup>2</sup> needs a frame rate of 87 Hz to be flicker free. May be affected by phosphor persistence.
- 7) temporal stability –how fast does the monitor change over time? Monitors usually need to be calibrated once a month or so, but when you first use your monitor you should check every week or so to check that it is stable. If your experiment is sensitive, calibrate more often.