The Briefest of Introductions to MATLAB

A Computational Environment for the Behavioral Sciences

Lawrence Hubert

University of Illinois

These slides are available at:

cda.psych.uiuc.edu/matlab_class_material/matlab_presentation.pdf

MATLAB (Matrix Laboratory)

This is the computational environment that is becoming the universal standard for the Behavioral Sciences. It is already the standard for all the Engineering and Computer Science disciplines.

Developed and maintained by the MathWorks, physically located in Natick, Massachusetts (just outside of Boston).

One of the least predatory companies I've ever encountered, with just greatpricing for students and academic settings generally. This is in contrast to SPSS, SAS, and the like, or for that matter, Adobe.

All the help, manuals, and so on you will ever need (and for free), are at `http://www.mathworks.com`

Also, see

`www.mathtools.net`

This is a site maintained by the MathWorks for computational things generally.

In comparison to closed software systems like SPSS, SAS, and SYSTAT (where you don't get to see or modify the actual code, or chose different options if they don't deem it necessary), the very (old) cliched proverb to follow is appropriate (edited for the original sexist language):

"Give a person a fish; you will have fed him/her for today. Teach a person to fish, and you have fed him/her for a lifetime."

With some facility with Matlab, you are no longer dependent on what (obsolete) fish someone else tries to feed you, e.g., “in K-means clustering, local optima are not an issue, so we will give you only one solution” WRONG!

You can “roll your own” analyses —

Extra credit: what generation does this phrase come from?

More indirectly, the MathWorks gives generously to public radio in Boston (WBUR) —

They may be the “Ben and Jerry’s” of software environments —

There is an enormous amount of detail that we could go over, but I will mention or introduce only what I have used and found valuable.

The various demos we do now are only intended for the mechanics; you don't need to understand all the nuances of the things introduced.

What might be a good place to start is the free Getting Started Manual (`getstart.pdf`, available where these notes are placed).

But if you stick with me for the next few hours, you should have enough to at least begin learning "on the job", using all the "help" material that is freelyavailable.

Basic MATLAB

- The Mother of all Calculators —

  where everything can be done with matrices (in fact, the basic unit of analysis is a matrix).

  This is where we start, using a simple example to show MATLAB as a powerful computational scratchpad. For our carry-along illustration, the same data set from Psychology 406 is used that introduced multiple regression.

  There are many built-in functions, including lots of matrix functions, to get our work done (MATLAB started as an interface to LINPACK and EISPACK, both linear algebra packages, and continues in this tradition today using LAPACK and BLAS).

- Programming language —

  it is very straightforward to use, but extremely powerful and much easier than, for example, C or Fortran. (There is no explicit array dimensioning needed, or to indicate and keep track of whether you are dealing with real or integer numbers.)

  There is *nothing* you can do in Fortran or C that you can't do in MATLAB (and in a much easier fashion).

  It is an interpretive language (so it translates and executes the statements "on the fly"); this is in contrast to compiled C or Fortran (or Basic or Pascal, among others), which produce stand-alone executables.

Given recent enhancements, interpreted MATLAB *is as fast as* compiled C.

Generally, we can do better speedwise with compiled Fortran (about a tenfold increase). Also, we can now do so with automatic conversions from MATLAB code to Fortran (the old `.dll`'s). I will mention how in a moment.

There is one overriding advantage of writing your own functions in MATLAB — they are easily modifiable `ascii` files. So, if someone wishes to run these, I just send them over email, and he/she can just "fire-up" MATLAB (a non-hobled student version with several toolboxes [e.g., Statistics and Optimization] costs about a $100) —

or use one of the “free” alternatives that will run M-files — OCTAVE, for example:

`http://www.octave.org`

I’ve been a Fortran programmer for over four decades, and that is long enough to keep track of matrix dimension sizes or to worry about what is an integer and what is not. I can’t tell you emphatically enough how much easier it is now to write my code in MATLAB.

If I need the speed, I can translate to callable Fortran subroutines. But usually, you can just stay with the MATLAB code (written in the form of what are called M-files) because that is generally fast enough.

In Fortran, I was a “spaghetti coder”, with lots of “go to”s (with statement numbers). In fact, I was once known as the “go to” kind-of guy — (this is a joke)

In MATLAB, there are no “go to”s and no statement numbers. When you use the (chroma-coded) Editor, structured programming is done automatically for you, with nice indentations and things lined up appropriately for easy reading.

I would gladly be a poster child for the MathWorks; they could put my picture on a milk carton with the caption: “Have you seen this Fortran programmer?”

- Provides great graphics —

  2- and 3-D plotting, visualization, animation, image processing.

  In 2-D plotting of prime importance in presenting data and statistical results, graphs can be edited easily with tools directly from the MATLAB figure window and exported to common file types — `.eps`, `.jpg`, and `.pdf`, for example.

  Files of these types can be embedded into Word documents (if you must), or better, using LaTeX.

  The MATLAB figure format file type (called `.fig`) can be used so the figure can be reedited in MATLAB

- Graphical User Interface Development Environment (GUIDE) —

  You can develop your own data collection GUIs or demonstrations. A prime example of this would be incorporating the (free, and freely available, open-source) (visual) Psychophysics Toolbox (primarily developed by David Brainard, now at U. Penn., Psychology):

  `http://psychtoolbox.org`

  For more, take the course in MATLAB programming from our own Alejandro Lleras.

- Application Program Interface (API) —

  for calling C, Fortran, Java routines within the MATLAB environment.

  We mentioned this earlier in the context of speeding up MATLAB code using Fortran produced `.dlls` (actually, they now go by the extension `.mexw32` on windows machines using the current version of MATLAB [this is release 2007a]).

  The conversion program (in the way of M-files) is called `matlab2fmex.m`, and written by Benjamin Barrowes:

  `sourceforge.net/projects/matlab2fmex`

Toolboxes

A Toolbox is a collection of M-files that do analysis within some specific area. Typically, the source code for the M-files is available for modification, even when the original toolbox may have been purchased (for real money even, as Yogi Berra says).

The commercial Toolboxes through the MathWorks are all on our site license. The first five Toolboxes I list below are probably the most germane for us (for example, I have paid for a separate license for these for years so they can easily be put on my laptop without worry about access to the license server).

Statistics; Optimization; Image Processing; Curve Fitting; Bioinformatics

Others of interest —
Neural Networks; Splines; Wavelets, Symbolic Math; Genetic Algorithm and Direct Search; Mapping (Geographical)

Also note —
Compiler (this converts M-files directly to C or C++ code, and generates stand-alone applications that can be distributed)

Free (open source) Toolboxes —

`www.mathworks.com/matlabcentral` — over 1000 M-files contributed by users.

`www.mathtools.net/MATLAB/tooboxes.html`

Auditory Perception; ILAB (eye movement analyses); Brainstorm (MEG/EEG); Chemometrics; N-way Toolbox; Smoothing; Econometrics; Spatial Statistics; NURBS; ICA (EEG/ERP – independent component analysis); LYNGBY (fMRI analyses); NIH CORTEX: NMRLAB; Psychophysics; Speech Processing; EMEGS (for EEG and MEG); SPM2 (Statistical Parametric Mapping – neuroimaging)

My Toolboxes

I have put a lot of material (we will talk about) at my web site:

`cda.psych.uiuc.edu/matlab_class_material`

'cda' stands for "combinatorial data analysis"

One published item: Hubert, L., Arabie, P., & Meulman, J. (2006). *The structural representation of proximity matrices with MATLAB*. SIAM: Philadelphia.

An earlier draft:
`siam_final_submission_r1.pdf`

The M-files directory: `srpm_mfiles`

Three draft Toolboxes (I'm doing these with Doug Steinley and Frieder Köhn) —

Cluster Analysis:
`cluster_chapter_r1.pdf`
`clusteranalysis_mfiles`

Unidimensional Scaling:
`unidimensionalscaling_chapter_r1.pdf`
`unidimensionscaling_mfiles`

Multistructural Analysis:
`multistructural_chapter.pdf`
`multistructuralanalysis_mfiles`

Other Toolbox Items

I have put the relevant MATLAB `.pdfs` for basic MATLAB and the five Toolboxes I rely on, at my web site (even though you could get these from the MathWorks directly if you wanted, it might be easier to have everything in one spot for you to access):

`apiext.pdf`; `apiref.pdf`; `bioinfo_ref.pdf`; `bioinfo_ug.pdf`; `buildgui.pdf`; `curvefit.pdf`; `data_analysis.pdf`; `getstart.pdf`; `graphg.pdf`; `images_tb.pdf`; `math.pdf`; `matlab_env.pdf`; `matlab_prog.pdf`; `optim_tb.pdf`; `programming_tips.pdf`; `refbook.pdf`; `refbook2.pdf`;`refbook3.pdf`; `stats.pdf`; `visualize.pdf`

There are three books that are relevant to statistical issues and MATLAB; all three have downloadable M-files to go along with these sources.

Computational Statistics Handbook with MATLAB
Wendy L. Martinez/Angel R. Martinez
Chapman & Hall/CRC, 2002

Exploratory Data Analysis with MATLAB
Wendy L. Martinez/Angel R. Martinez
Chapman & Hall/CRC, 2005

Multi-way Analysis: Applications in the Chemical Sciences
Age Smilde/Rasmus Bro/Paul Geladi
Wiley, 2004

There are several other very good sources for learning about things MATLAB, for free –

The large collections of demos that come with MATLAB and the Toolboxes

The great number of Webinars that you can see for free, covering all aspects of MATLAB and the various toolboxes. We list below a few of the possibilities:

Introduction to MATLAB
Data Analysis with Curve Fitting and Statistics Toolboxes
Applied Optimization Using MATLAB

The easiest way to get help “on the fly” is to issue the command `help` in the command window; a collection of help directories come up.

The ones I use most often are:
`help matfun` — all the basic matrix functions
`help datafun` — all the basic data analysis functions
`help graph2d` — all the basic 2-dimensional plotting functions
`help stats` — all the functions in the MATLAB Statistics Toolbox
`help optim` — all the functions in the MATLAB Optimization Toolbox

MATLAB as a Calculator

Some of the capability of MATLAB as a calculator will be illustrated with the same data I use in Psychology 406 to introduce simple correlation and regression, least-squares, the general linear model, and all of the related concepts.

I have put these data into a file (`ascii`) with Notepad, with the name: `community_data.dat`

Note: all the personal files that you use or create in MATLAB should be on, say, your own flash drive; you can then make this your current directory (for the calculator use of MATLAB; creating your own M-files to use, and so on).

community	accidents(100s)	vehicles(1000s)	police
1	1	4	20
2	4	10	6
3	5	15	2
4	4	12	8
5	3	8	9
6	4	16	8
7	2	5	12
8	1	7	15
9	4	9	10
10	2	10	10

The `.dat` extension is important (so turn on “show known file types” in Windows — tools/Folder Options/view — and uncheck the box next to “Hide extensions for known file types”). The extension tells MATLAB it is an `ascii` file and can be loaded with the command: `load community_data.dat`

Once done, a 10×3 matrix called `community_data` is then available in your workspace to operate on and use.

The verbatim contents of `community_data.dat` are:

```
1   4  20
4  10   6
5  15   2
4  12   8
3   8   9
4  16   8
2   5  12
1   7  15
4   9  10
2  10  10
```

The Desktop

When the default Desktop is called up, we have the:

command window: this is where everything happens and all commands are entered

command history: if you would care to, you could reuse previous commands completed earlier that are saved here

workspace: this is where all the variables (matrices) exist that you can work on

current directory: set this to your current work area. I would suggest that this be a flash drive for all your ongoing stuff (with, obviously, continual backups to other places, right!)

You can access all the help possibilities you will ever need from the Desktop.

We will now load up MATLAB and show you the desktop. We will continue to flip back and forth between MATLAB and these slides, so have the hard copies in front of you.

Basic Data Analysis

The elementary (built-in) data analysis functions generally operate in a column-wise orientation on a matrix:

```
load community_data.dat

community_data

mean(community_data)
```

The functions `median`, `mode`, `max`, `min`, `std`, and `var`, work the same way.

To do it row-wise, use the transpose operation "`'`"; for example, `var(community_data')`

The following four executed statements illustrate a few ways matrices operate with MATLAB that are relevant to us.

the first shows how matrices may be entered directly by hand;

the second is the construction of a matrix by concatenation;

the third illustrates that scalar multiplication of a matrix is extended by MATLAB to scalar addition;

the fourth illustrates that matrices are indexed in a row-column order.

```
>> A = [1 2 3;2 1 3;3 1 2]

A =

     1     2     3
     2     1     3
     3     1     2

>> B = [A 3*A;A-(2*A) A+A]

B =

     1     2     3     3     6     9
     2     1     3     6     3     9
     3     1     2     9     3     6
    -1    -2    -3     2     4     6
    -2    -1    -3     4     2     6
    -3    -1    -2     6     2     4
```

```
>> C = A + 10

C =

    11    12    13
    12    11    13
    13    11    12

>> A(2,3)

ans =

     3
```

The two functions `cov` and `corrcoef` give the (3×3) covariance and correlation matrices.

```
corr_community_data = ...
corrcoef(community_data)

cov_community_data = ...
cov(community_data)

correlation_accident_vs_vehicle = ...
corr_community_data(1,2)

cov_accident_vs_police = ...
cov_community_data(1,3)
```

Let's see if we can get to the solution of the normal equations:

$$\mathbf{b} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}\mathbf{Y}$$

We wish to predict the dependent variable, number of accidents (placed in $\mathbf{Y}$, 10×1), from the number of vehicles and the number of police (second and third columns of the 10×3 design matrix $\mathbf{X}$, with the first column containing all ones for the additive constant). The 3×1 regression vector $\mathbf{b}$ contains, in order, the additive constant and the regression coefficients on the number of vehicles and number of police.

```
Y = community_data(:,1)

X = [ones(10,1),community_data(:,2),...
community_data(:,3)]

b = (inv(X'*X))*(X'*Y)

pred_values = X*b

correlations = ...
corrcoef([Y,pred_values])

R_squared = (correlations(1,2))^2
```

Other Matrix Operations

```
det(X'*X)

rank(community_data)

trace(X'*X)

inv(X'*X)
```

Some special matrices:

```
zeros(10,10) [ones(10,10)]

rand(10,10) [randn(10,10)]

eye(10)
```

For Multivariate Analysis

$[\mathbf{V}, \mathbf{D}] = \mathrm{eig}(\mathbf{A})$, where $\mathbf{A} = \mathbf{VDV}'$, for $\mathbf{A}$ square; $\mathbf{V}$ is orthogonal and contains eigenvectors (as columns); $\mathbf{D}$ is diagonal and contains the eigenvalues (ordered from *smallest to largest*).

$[\mathbf{U}, \mathbf{S}, \mathbf{V}] = \mathrm{svd}(\mathbf{B})$, where $\mathbf{B} = \mathbf{USV}'$; the columns of $\mathbf{U}$ and the rows of $\mathbf{V}'$ are orthonormal; $\mathbf{S}$ is diagonal and contains the non-negative singular values (ordered from *largest to smallest*).

Some Details

The semicolon (;) at the end of a statement, suppresses output.

MATLAB is case-sensitive.

The up-arrow key ($\uparrow$) retrieves the previous command (for editing purposes, possibly).

`clc` clears the screen.

An ellipsis (...) allows a long command to continue on the next line.

The colon operator (:) does a lot of things:
[1:10] produces 1 2 3 4 5 6 7 8 9 10 —
A(:,1) selects all rows and just the first column of A.
gives index ranges in “for” loops: `for i = 1:n`

NaN stands for “not a number”, and you get it, for example, from 0/0 —

There are many ways of getting a matrix into the workspace. We have mentioned two: the `ascii` format in a `.dat` file and using the `load` command; or enter directly as a matrix, e.g.,: `[2 2 2;3 4 2;1 6 7]` with semicolons separating rows, and blanks or commas separating row entries.

There is also an import wizard for `.xls` files (among others).

Generally, we will construct matrices through various operations or concatenations:

for rows: `[ --- ; --- ; --- ]`

for columns: `[ --- , --- , --- ]`

You can save the workspace, or part of it, as a `.mat` file, and reload it later. Or you can send it to someone else, and they could reload into their MATLAB environment.

`.mat` files are binary so they are not easily readable.

There is also a `diary` function that can save your output to a file (when `diary` is `on`). When `diary` is `off`, the output is not saved to a file. Use `diary filename`; if no file is specified, the file is just called `diary`; use `diary off` to stop saving the output.

Don't be afraid to use (a lot of redundant) parentheses to clarify the order of operations; it will never hurt.

Never, never, ever, rely on the order of operation precedence to clarify an ambiguous statement — make it explicit with parentheses. Trust me on this one; you will mess up big-time, many time(s), if you don't.

MATLAB for Plots

One very powerful feature of MATLAB is its (2 – and 3 – dimensional) plotting capabilities, along with the very nice environment to edit and label these plots extensively, and eventually, to save plots in a variety of file formats: `.eps. .pdf`, `.jpg`, among others.

I would suggest using `.eps` always — it is the (or at least, my) preferred way to put graphics into TeX and LaTeX documents; it is also editable and savable in all the usual drawing programs (Illustrator, Freehand, CorelDraw, and so on).

```
plot(community_data(:,2), ...
community_data(:,1),'ko')

for i = 1:10

  objectlabels{i,1} = int2str(i);

end

text(community_data(:,2), ...
community_data(:,1), ...
objectlabels,'fontsize',10, ...
'verticalalignment','bottom')
```

```
ylabel('number of accidents')

xlabel('number of vehicles')

title('Scatterplot of Accidents ...
versus Vehicles')

axis([0,20,0,6])

text(14,1,'{\it Note the italics}')

gtext('{\bf Note the boldface}')
```

Some Plotting Points

There are many tools and ways for editing your plots, both from the command line and by using the simple Adobe Illustrator-like interface in the Figure window. Also, on exporting the file, you can specify the size.

If you stay in one of the formats Illustrator can edit (e.g., `.eps`), you can use Illustrator to edit your graphics at will. The Department has a site license for Illustrator.

Once you have a plot the way you like it, you could even generate the M-file that produced it — which would then be ready to accept another set of data.

TeX and LaTeX notation can be used in the plots (for symbols, italics, formulas, and so on). Text can be placed at certain coordinates with `text(x,y,'string')`, or with cross-hairs with `gtext('string')`.

Script M-files

A script M-file is a collection of statements and commands placed in an file that can be executed when the file name is typed in the command window.

Scripts are easily reusable (and changeable). They operate on and produce variables in the workspace.

As an example, we use `ganmaa_script.m`, which produces boxplots for some two-independent sample data I was helping someone with last year.

Make sure you name your scripts meaningfully, so you will know what they are the next time you look into your current directory (not, e.g., `script_one.m`).

```
load pre_vitd_level.dat
load post_vitd_level.dat
pre_vitd_level(:,1)
post_vitd_level(:,1)

boxplot([pre_vitd_level(:,1), ...
post_vitd_level(:,1)],'notch', ...
'on','labels',{'Before','After'})

title('Vitamin D Levels in Children ...
Before and After Drinking Vitamin D ...
Fortified Whole Milk')

ylabel('25(OH)D, ng/ml')
```

Function M-files

A function M-file takes input argument(s) and produces output arguments. The form of two functions, `targlin.m` and `proxrand.m`, shows the general pattern.

`targlin.m` produces a symmetric proximity matrix of size $n \times n$ containing distances between equally and unit-spaced positions along a line;

`proxrand.m` produces a symmetric proximity matrix with entries that are a random permutation of those in a symmetric input matrix.

If you type, for example, `help targlin`, you will get the help header comments echoed. These are the lines "commented out" with "%".

targlin.m

```
function [targlinear] = targlin(n)

%  TARGLIN produces a symmetric proximity matrix of size
%  $n \times n$, containing distances
%  between equally and unit-spaced positions
%  along a line: targlinear(i,j) = abs(i-j).
%
%  syntax: [targlinear] = targlin(n)

targlinear = zeros(n,n);

for i = 1:n-1
   for j = (i+1):n

      targlinear(i,j) = abs(i-j);
      targlinear(j,i) = targlinear(i,j);

   end
end
```

proxrand.m

```
function [randprox] = proxrand(prox)

%  PROXRAND produces a symmetric proximity matrix RANDPROX
%  with a zero main diagonal having
%  entries that are a random permutation of those in the
%  symmetric input proximity
%  matrix PROX.
%
%  syntax: [randprox] = proxrand(prox)

n = size(prox,1);
change = randperm((n*(n-1))/2);
randprox = prox;

for i = 1:(n-2)
   for j = (i+1):n

      k = i + j;
```

```
      for ione = 1:(n-2)
         for jone = (ione+1):n

            kk = ione + jone;

            if(change(k) == kk)

               temp = randprox(i,j);
               randprox(i,j) = randprox(ione,jone);
               randprox(j,i) = randprox(i,j);
               randprox(ione,jone) = temp;
               randprox(jone,ione) = randprox(ione,jone);

            end
         end
      end

   end
end
```

To show how these two M-functions work, enter the following:

```
prox = targlin(10)

[randprox] = proxrand(prox)
```

Points About M-files

Functions work in their own workspace — the Las Vegas property of what goes on in the function space, stays in the function's own workspace. This is different from the use of scripts.

The names of the M-files and the functions should be exactly the same (and meaningful).

Notice the use of the editor, chroma-coding, structured programming, and so on. This makes writing these M-files much easy.

The Editor

- The MATLAB Editor is a very nice device for writing your functions and scripts, i.e., your M-files.
- It is chroma-coded so you can see the special words and commands highlighted in different colors.
- There is automatic nested indentation of the statements as you go along to make the programs readable and its structure obvious — the `end`s line up correctly with the `if`s, `for`s, `while`s, and so on.
- There are cute little sounds and colors (or annoying, demanding on your mood) that occur when you begin to commit certain errors.
- A debugger operates with the editor, but you don't really need it given the transparency of the written code — not so in Fortran where you can spend days debugging code.

Graphical User Interfaces (GUIs)

GUIDE (Graphical User Interface Development Environment) is a layout tool that produces two files: a figure file to hold the actual GUI produced, and a (template) code M-file in which you can program what are called the callbacks, i.e., what happens when a button is pressed or a slider is dragged.

It is generally easiest to modify an existing GUI to do what you want (assuming you can find one).

When you program experiments (and use, perhaps, the Psychophysics Toolbox), you may need to have a standalone MATLAB for accurate timings that doesn't need to go back to any server for license information.

We will demonstrate two GUIs — one demo from the MathWorks is on the traveling salesman problem: `travel.m`;

the second is part of the Statistics Toolbox: `randtool.m`.

We give the “help” files; look at the M-files; and run the GUIs.

Programming

Operational symbols to use (the colons here are not part of the symbol):

== : “is equal to”
~= : “is not equal to”
& : “and”
| : “or”
~ : “not”
>= : “greater than or equal”
<= : “less than or equal”
> : “greater than”
< : “less than”

Flow control:

```
if (statement)
    statements
elseif (statement)
    statements
else
    statements
end
```

```
if (statement)
    (statements)
end
```

for (statement)
    statements
end

while (statement)
    statements
end

Data Structures

Besides a (two-way) row $\times$ column matrix, we have *multidimensional arrays* (e.g., row $\times$ column $\times$ layer) that can be of arbitrary dimensionality.

Cell arrays contain as elements other arrays; these must be accessed with curly braces, "`{ }`".

Structures have entries accessed by textual field designators (with a ".") — much like a data-base entity.

Remember that, generally, text used in any context has single quotes: e.g., `S = 'Larry Hubert'`

```
>> roster(1).name = 'Larry Hubert';
>> roster(1).score = 95;
>> roster(1).grade = 'A';
>> roster(2).name = 'David Budescu';
>> roster(2).score = 100;
>> roster(2).grade = 'A+';
>> roster(3) = struct('name','Michael Regenwetter','score',89,'grade','A-');
>> roster
roster =

1x3 struct array with fields:
    name
    score
    grade

>> roster(3).name
ans =

Michael Regenwetter
```

MATLAB Speed Tricks

Vectorization: make sure you use the matrix operations instead of, say, many nested “for” loops.

Preallocation: allocate your matrices with, say, `zeros`, of the appropriate size. It is very slow to add to (and expand) the matrices entry by entry.

M-Lint code checker gives a report on your M-files and suggests obsolete code and speed-ups.

The Profiler (not the TV show) tells where in a specific M-file the process is spending the most time; if identified, maybe you can reprogram in some way (e.g., vectorize) to increase speed.

There is the program, `matlab2fmex.m`, that produces Fortran code and an interface gateway from a MATLAB M-file and compiles it. When the function is called in MATLAB, the `.dll` (really, now called `.mexw32`) is used first.

If speed is really important to you, a ten-fold increase is obtained using compiled and callable Fortran routines.

Images

A two-dimensional array can be displayed as an image where the array elements determine brightness or color of the image (in pixels).

If there are t distinct integers in the array, a `colormap` is a $t \times 3$ matrix, where the t integers are indexed into the colors.

`imread` and `imwrite` can read and write images in the standard formats (`.tiff`, `.jpg`, `.bmp`, and so on).

Try this:

```
>> load durer.mat
>> whos
  Name            Size                   Bytes  Class     Attributes

  X             648x509                2638656  double
  caption         2x28                     112  char
  map           128x3                     3072  double

>> colormap(map)
>> image(X)
>> axis image
```

Animation

We will illustrate one animation of coloring a proximity matrix and seeing how it changes going through row (and column) permutation.

The script is called: `matrixcolor_script.m`, and is given verbatim below. The function `matcolor.m` is given on the next slide.

```
[outperm,rawindex,allperms,index] = ...
 order(targlin(15),targlin(15),randperm(15),3);

matcolor(targlin(15),allperms,index)
```

```
function  matcolor(datamat,perms,numperms)

%MATCOLOR constructs a color movie of the effects of a series of
%  permutations on a proxmity matrix.
%  DATAMAT is an $n \times n$ symmetric proximity matrix;
%  PERMS is a cell array containing NUMPERMS permutations.

m=moviein(numperms);

for i=1:numperms
   pcolor(datamat(perms{i},perms{i}));
   axis ij off;
   colormap(bone(256));
   colorbar;
   m(:,i) = getframe;
end
   movie(m);
```