

The Abridged Matlab Crash Course

Adam Attarian

May 26, 2009

First thing is first. . .

- ▶ For help on any command, use the `help` command. This is the most important command there is. For more info, type `doc` command. This is how we learn how to use Matlab.
- ▶ `clear` clears all of the variables out of your workspace.
- ▶ `clc` clears the command window.
- ▶ Anything behind a `%` is a comment and is not executed. Comment liberally.

Entering Matrices

MATLAB works with essentially one kind of object: a rectangular matrix with possibly complex entries. 1-by-1 matrices are scalars, and matrices with one row or column are vectors.

Matrices can be input into MATLAB several ways:

- ▶ Entered explicitly
- ▶ Generated by functions or statements
- ▶ Imported from external data.

Example: `A=[1 2 3; 4 5 6; 7 8 9]` enters a 3×3 matrix and assigns it to the variable `A`, where the `';` starts a new row.

Matrix Operations

These are the available matrix operations in Matlab:

+	addition
-	subtraction
*	multiplication
^	power
'	(conjugate) transpose
\	left division
/	right division

The matrix division operations need a special comment. . .

Matrix Division

Matrix division is how you solve linear systems that aren't too big. If A is invertible and b is a solution vector, then

- ▶ $x=A \setminus b$ is the solution to $Ax = b$.
- ▶ $x=b/A$ is the solution to $xA = b$.

Do NOT use $x=\text{inv}(A)*b$!

The \setminus operator is pretty smart:

- ▶ If A is square, then \setminus tries a Cholesky factorization, then an LU factorization with Gaussian Elimination if necessary to solve the system.
- ▶ If A is not square, then \setminus uses a QR factorization to solve the over/under determined system in the least squares sense.

Array Operations

The matrix operations $+$ and $-$ already operate element-wise, but the others do not. So $A*B$ is matrix multiplication, not element-wise multiplication. All the other operations can be made element-wise by putting a dot before them. Examples:

- ▶ $[1,2,3,4] .* [1,2,3,4]$; (this wouldn't make sense without the dot!)
- ▶ $[1,2,3,4] .^2$
- ▶ $A .* B$ is $(AB)_{ij} = A_{ij} \cdot B_{ij}$.

Dimensions have to match. These ideas are particularly useful when making graphics or simply plotting functions.

Matrix Building Functions

Chances are you'll have to use at least one of these functions at least once. . .

<code>eye</code>	identity matrix
<code>zeros</code>	matrix of zeros
<code>ones</code>	matrix of ones
<code>diag</code>	create or extract a diagonal
<code>triu</code>	upper triangular portion of a matrix
<code>tril</code>	lower triangular portion of a matrix
<code>rand</code>	random matrix
<code>toeplitz</code>	toeplitz matrix
<code>meshgrid</code>	X and Y arrays for 3D plots

For example, `zeros(m,n)` creates an m by n zeros matrix and `zeros(n)` creates an n by n matrix. To match the size of another matrix, you can do `zeros(size(A))`.

for, while, if relations

Matlab flow control works nearly the same in every other programming language.

A for loop repeats a group of statements for a *fixed* number of times.

```
1 for m=1:100
2   num=1/(m+1);
3 end
```

A while loop executes until some condition is no longer satisfied:

```
1 v=1; num=1; i=1;
2 while num < 10000
3   num=2^i;
4   v=[v; num];
5   i=i+1;
6 end
```

for, while, if relations

if relations are fairly straightforward.

- ▶ *if condition*
 statements
end

You can also test more than one statement with `elseif`:

- ▶ *if condition*
 statements
elseif *other conditions*
 other statements
end

Relations

The relational operations in MATLAB are

<	less than
>	greater than
<=	less than or equal
>=	greater than or equal
==	equal
~=	not division
&	and
	or
~	not

Relations

When applied to scalars, a relation returns 1 or 0. When applied to matrices of the same size, a relation returns a matrix of 0's and 1's giving the value of the relation at each entry. So if you wanted to execute a statement when two matrices A and B are equal, you could use

- ▶ `if A==B; foo; end`

But if you wanted to do something when A and B were not equal, the relation would be

- ▶ `if any(any(A~=B)); foo; end`

The functions `any` and `all` can be creatively used to reduce matrix relations to vector or scalar relations.

Scalar Functions

Certain functions operate essentially on scalars, but operate element-wise when applied to a matrix. The most common functions are

sin	asin	exp	abs	round
cos	acos	log (natural log)	sqrt	floor
tan	atan	rem (remainder)	sign	ceil

Vector Functions

Other functions operate on vectors, though on matrices these functions act on each column separately. If you wanted row-by-row action, you would operate on the transpose of the matrix. Some of the more useful functions are

max	sum	median	any
min	prod	mean	all
sort	cumsum	std	linspace
length	cumprod		

So the maximum element in a matrix A is given by `max(max(A))` instead of `max(A)`, which would return the largest entry in each column of A .

Matrix Functions

Matlab stands for Matrix Laboratory, so it is no surprise that much of Matlab's power comes from its matrix functions. Some of the most commonly used are

eig	eigenvalues and eigenvectors
det	determinant
inv	inverse (avoid!)
rank	numerical rank estimate
rref	reduced row echelon form
poly	characteristic polynomial
sqrtn	matrix square root
expm	matrix exponential
lu	LU factorization
qr	QR factorization
svd	singular value decomp

Note that many Matlab commands have multiple outputs. So $y = \text{eig}(A)$ gives the eigenvalues, but $[v, d] = \text{eig}(A)$ gives you both the eigenvalues and eigenvectors!

Colon notation, submatrices, and vectorization

- ▶ Vectors and submatrices are often used to do some fairly complex data manipulation.
- ▶ Vectorization can dramatically speed up your code, so learn how to use it! Many for loops aren't really needed.
 - ▶ Compare `J=sum(sum((data-model).^2))` with `r=data-model; J=r'*r;`
- ▶ Using `0.2:0.2:1.2` creates the vector `[0.2, 0.4, 0.6, 0.8, 1.2]`
- ▶ and `5:-1:0` gives `[5 4 3 2 1 0]`.
- ▶ The colon notation can be used to access submats of a matrix.
 - ▶ `A(1:4,3)` is the first *four* entries of the *third* column of *A*.

Colon notation, submatrices, and vectorization

- ▶ A colon by itself denotes an entire row or column.
 - ▶ $A(:,3)$ is the third column of A , $A(1:4,:)$ is the first four rows.
- ▶ Arbitrary integer vectors can be used as subscripts
 - ▶ $A(:, [2\ 4])$ contains as columns, columns 2 and 4 of A .
- ▶ You can also do assignments with this notation
 - ▶ $A(:, [2\ 4\ 5])=B(:, 1:3)$ replaces columns 2,4,5 of A with the first three cols of B .
- ▶ Also useful here: `flipud(x)` and `fliplr(x)` where x is an n -vector.

M-files: script and functions

Most of your work will be in an m-file. There are two kinds of m-files: script files, and function files.

- ▶ A script file is just a sequence of normal Matlab commands. After you run the script file, you have access to all of the variables that were used – this is the *stack*.
 - ▶ Variables in the script file are global and will change the value of variables of the same name in the current session.
- ▶ Function files take an input and give an output. The stack is local, but can be made global (see `help global`).
 - ▶ Syntax for the first line:
`function output=funcname(inputs)`
- ▶ Useful objects when coding functions:
 - ▶ `nargin`, `vargin`, `nargout`
- ▶ Run the file by typing its name at the command line, or by pushing play in the editor.

Graphics and Plots

We'll just do a quick intro here. The main visualization commands in Matlab are `plot`, `plot3`, `mesh`, and `surf`. Read the documentation for each of these for all of the different plotting options. Also, type `demo matlab graphics` for a full tour of Matlab graphics.

- ▶ `plot` creates linear line plots. For instance, to plot $y = e^{-x^2}$ over the domain $[-1.5, 1.5]$, you could use `x=-1.5:.01:1.5; y=exp(-x.^2); plot(x,y);` Note the element wise exponentiation!
- ▶ Parametric plots are also easy to do. For example:
`t=0:.01:2*pi; x=cos(3*t); y=sin(2*t); plot(x,y)`
- ▶ `semilogy`, `semilogx`, `loglog` produces log plots.
- ▶ `mesh` and `surf` produce 3D surface plots. See the Matlab crash course for demos.

Graphics and Plots

Of course graphs can be given titles, axes labeled, and axes changed. Here are some commands to do just that.

title	graph title
xlabel	x-axis label
ylabel	y-axis label
<code>axis([x_{min}, x_{max}, y_{min}, y_{max}])</code>	set axis to these limits
<code>ylim([y_{min}, y_{max}]);</code>	set the y- limits
<code>xlim([x_{min}, x_{max}]);</code>	set the x- limits
axis square	square up the axes
axis equal	even up the axes
axis on/off	turn on/off axes

Use the axes command after you build the plot. The labels take strings as inputs. Example: `title('my awesome plot');`

Graphics and Plots

More than one plot in one figure! You sure can! Some code snippets to do just that... In Matlab there is typically more than one way to do things. Each has its own benefits and drawbacks. Let's name some.

```
1 x=0:.01:2*pi;
2 y1=sin(x); y2=sin(2*x); y3=sin(4*x);
3 plot(x,y1,x,y2,x,y3);
```

```
1 x=0:.01:2*pi;
2 Y=[sin(x)', sin(2*x)', sin(4*x)'];
3 plot(x,Y);
```

```
1 x=0:.01:2*pi;
2 y1=sin(x); y2=sin(2*x); y3=sin(4*x);
3 plot(x,y1)
4 hold on; plot(x,y2); plot(x,y3);
```

Solving Ordinary Differential Equations

Matlab has a range of functions for solving initial value problems of the form

$$\frac{d}{dt}y(t) = f(t, y(t)), \quad y(t_0) = y_0,$$

where t is a scalar, $y(t) \in \mathbb{R}^m$ is unknown, and $f(t, y(t)) \in \mathbb{R}^m$.

- ▶ The function f defines the ODE, and the IC $y(t_0) = y_0$ defines an initial value problem.
- ▶ The simplest way to solve is to write a function that evaluates the right hand side, and then call one of the available ODE solvers.
- ▶ Many options are available to tune the performance of the integrator.

ODE Example – First Order

To solve the scalar ODE

$$\dot{y}(t) = -y(t) - 5e^{-t} \sin 5t, \quad y(0) = 1$$

for $0 \leq t \leq 3$ with `ode45`, we create an odefile `my.f` containing

```
1 function yprime=myf(t,y)
2     yprime=-y-5*exp(-t)*sin(5*t)
```

and then type

```
1 tspan=[0 3]; yzero=1;
2 [t,y]=ode45(@myf,tspan,yzero);
3 plot(t,y,'*--'); xlabel t; ylabel y(t);
```

ODE Example – Higher Order

Higher order ODEs can be solved only if they are first rewritten as a system of first order equations. For example, consider the pendulum equation,

$$\frac{d^2}{dt^2}\theta(t) + \sin \theta(t) = 0.$$

Defining $y_1(t) = \theta(t)$ and $y_2(t) = \frac{d\theta(t)}{dt}$ gives us a new system

$$\dot{y}_1(t) = y_2(t)$$

$$\dot{y}_2(t) = -\sin y_1(t)$$

This is something that we can code up for use by `ode45` in a function `pend`, as follows.

ODE Example – Higher Order

```
1 function ydot = pend(t,y)
2     yprime=[y(2);
3             -sin(y(1))];
```

The following commands compute solutions over $0 \leq t \leq 10$ for a given initial condition. Since we are solving a system, in the output $[t,y]$ the i th row of the matrix y approximates $(y_1(t), y_2(t))$ at time $t = t(i)$.

```
1 tspan=[0 10];
2 yzero=[1;1];
3 [t,y]=ode45(@pend,tspan,yzero)
```

Solving Ordinary Differential Equations

The general form of a call to `ode45` is

```
[t,y]=ode45(@fun,tspan,yzero,options,p1,p2,...)
```

where the arguments `p1,p2,...` are parameters that will be passed to the ODE function.

- ▶ There are many other ODE integrators: `ode23`, `ode15s`, `ode113`. `ode45` is usually a good place to start though.
- ▶ The ODE options offer a lot. Don't pretend they're not there.

Optimization & Least Squares

Most generally, optimization involves iteratively finding the minimum of a vector valued function. These problems are everywhere in applied mathematics, and they take the form of

$$x^* = \min_{x \in \Omega} f(x)$$

Least squares problems arise in data fitting, parameter estimation, and look like

$$\min_{x \in \Omega} \|Ax - b\|_2^2$$

if linear least squares, and

$$\min_{x \in \Omega} \|F(x)\|_2^2$$

if nonlinear.

Optimization & Least Squares

This is how to get started.

- ▶ Define a cost function from \mathbb{R}^m to \mathbb{R} .
- ▶ Pick an algorithm or method to minimize the cost.

For example, if one wanted to find the parameters q to a model defined by a set of DE's that best fit collected data; one method that *may work* is as follows

1. Given a guess for the parameter, q_0 , solve the model at that parameter.
2. Given the model output, compare to the data. That is, compute the cost. Say, with a function that looks like

$$J(q) = \sum_{i=1}^N (\text{model}(q)_i - \text{data}_i)^2$$

3. Repeat with a better guess based on the result. (Let the algorithm do the work).

Optimization & Least Squares

The Optimization Toolbox in Matlab can help out with these problems.

- ▶ fminsearch
- ▶ fmincon
- ▶ fminunc
- ▶ lsqnonlin
- ▶ lsqnonneg

Also, google for Tim Kelley. Lots of optimization software available from his site.

There. Is. So. Much. More.

The only way to really learn Matlab is to figure things out as you need them. Or be like me and subscribe to the Matlab newsgroup, read the Matlab blogs, enter the Matlab programming contests. . . But you should definitely look up and read about:

- ▶ anonymous functions and subfunctions!
- ▶ structure and cell arrays!
- ▶ multi-dimensional arrays!
- ▶ numerical integration aka quadrature!
- ▶ sparse matrix operations!
- ▶ how to work with strings and parse data files!
- ▶ handle graphics (scary)!