

1 Matlab Primer

The purpose of these notes is a step-by-step guide to solving simple optimization and root-finding problems in Matlab. To begin, the basic object in Matlab is an array; in two dimensions, these are matrices:

$$A = \begin{bmatrix} 1 & 6 \\ 78 & 9 \end{bmatrix}.$$

Note: everything is case-sensitive, so a and A are different. One enters the above matrix as

$$A = [1, 6; 78, 9];$$

you can use spaces instead of commas. Commas or spaces mean move to the next column in the same row, semicolons mean move to the next row. The semicolon at the end of the line means 'do not display the result of the calculation.' If you instead typed

$$A = [1, 6; 78, 9]$$

Matlab would display the A matrix:

$$A = \begin{bmatrix} 1 & 6 \\ 78 & 9 \end{bmatrix}.$$

Now imagine multiplying two matrices together:

$$A * B$$

where

$$B = \begin{bmatrix} 4 & 1 \\ 7 & 8 \end{bmatrix}.$$

You simply type this in Matlab and it produces

$$\text{ans} = \begin{bmatrix} 46.0 & 49.0 \\ 375.0 & 150.0 \end{bmatrix}.$$

If you want to assign the answer to some other array (like C) you would type

$$C = A * B$$

and this would produce

$$C = \begin{bmatrix} 46.0 & 49.0 \\ 375.0 & 150.0 \end{bmatrix}.$$

There are about a million things you can do to a matrix in Matlab. Some useful ones are:

1. invert:

$$C = \text{inv}(A)$$

produces

$$C = \begin{bmatrix} -0.0196 & 0.0131 \\ 0.1700 & -0.0022 \end{bmatrix}$$

2. trace:

$$C = \text{trace}(A)$$

produces

$$C = 10$$

3. compute eigenvalues:

$$C = \text{eig}(A)$$

produces

$$C = \begin{bmatrix} -17 \\ 27 \end{bmatrix}$$

this is a vector of the eigenvalues

4. transpose:

$$C = A'$$

produces

$$C = \begin{bmatrix} 1 & 78 \\ 6 & 9 \end{bmatrix}$$

5. norm:

$$C = \text{norm}(A)$$

produces

$$C = 78.5356$$

Matlab uses the least-squares norm, which basically takes the components one at a time, squares them, sums them, and takes the square root; if you type

$$C = \text{norm}(A, \text{inf})$$

you get the sup-norm, which is the maximum difference between any element and zero

6. solving a linear system $Ax = b$ can be done two ways:

$$x = \text{inv}(A) * b$$

$$x = A \backslash b$$

where \backslash is a shorthand for the operation invert and multiply.

Operations in Matlab have the following priority:

1. Exponentiation:

$$A^n$$

means raise A to the n th power and is done first;

2. Multiplication/Division;
3. Addition/Subtraction.

All calls to functions are put first, so that

$$C = \text{trace}(A)^2$$

would take the trace of A and square it, but

$$C = \text{trace}(A^2)$$

would square the matrix and then take the trace. Only conformable matrices can be operated on, and Matlab will tell you when it can't legally perform an operation. But be warned that many Matlab functions are built to accommodate generalized operations; for example, taking the inverse of a non-square matrix will tend to return something, just likely not what you expected.

Some useful built in functions are:

1. exponentiation `exp`:

$$C = \exp(5) = e^5 = 148.41$$

where

$$1 = \log(e)$$

is Euler's constant;

2. natural logarithm `log`:

$$\log(1) = 0$$

3. sines and cosines and all trigonometric functions as well.

All of these functions can be applied to arrays, in which case they go element by element. To do the same with basic operations like multiplication, use the 'dot' type:

$$A .* C$$

would create an array the same size as A and C with each element being the product of the corresponding elements in A and C . For example, if

$$A = \begin{bmatrix} 1 \\ 4 \end{bmatrix}$$

and

$$C = \begin{bmatrix} 2 \\ 9 \end{bmatrix}$$

then

$$A * C = \begin{bmatrix} 2 \\ 36 \end{bmatrix}$$

whereas

$$A * C = 38.$$

That should be enough to get you started. Let's work out a couple of problems. First, a minimization (remember that maximization is just minimizing the negative; for some reason, optimization packages all do minimization):

$$\min_{x_1, x_2} f(x_1, x_2) = \{x_1^2 + x_2^2\}.$$

This has no constraints and results in the solution

$$(x, y) = (0, 0).$$

How would we do this in Matlab? We would do two steps.

1. Define a m-file containing the function we wish to minimize. Click on the New button and choose m-file. The Editor will open. You define your function this way:

```
function z = fname(x);  
z = x(1)^2 + x(2)^2;
```

Save that file. The first line says fname is the name of the function which associates the vector x with the scalar z ; fname can be whatever you want, but if you use a name already being used by Matlab you lose the ability to access that file. For example, if you called your function sin, you couldn't access the built-in sine function.

2. Now we call the function. In the Command Window type

```
f = fname([1 1])
```

and you will be returned the value of fname at (1, 1):

$$f = 2$$

3. Finally, we will minimize the function by choosing $x(1)$ and $x(2)$. In the Command Window type

```
x = fminsearch('fname',[1 1],optimset('fminsearch'))
```

and you will get back quickly

$$x = [0.0; 0.0].$$

fminsearch is a built-in minimization routine; the call says find the function fname and minimize it, using [1 1] as a starting place and using the default settings for fminsearch (the last term is not important at this stage, as it

controls the optional parts of the algorithm being used to minimize the function). Good initial guesses are helpful because they speed up the process of finding the answer and sometimes a bad guess can lead the algorithm astray. When minimization is done, return the value and name it x .

If you change the command to

```
[x, f] = fminsearch('fname',[1 1],optimset('fminsearch'))
```

it will return not only x , the maximizer, but f , the function value at that point:

$$\begin{aligned}x &= [0.0; 0.0] \\f &= 0.0\end{aligned}$$

When you save files, you must be sure that they are on the path that Matlab uses to search for files. Use the Set Path command under File to add the folder that you save things in. The path is searched from the top down, so filenames that conflict are resolved according to a 'first come, first serve' approach, which is why you usually lose access to canned Matlab files if you use their name for your program. Better not to do that.

Now imagine we want to solve for the solution to a nonlinear equation, like a first-order condition for minimization:

$$x^2 - x^{\frac{2}{3}} - 4 = 0.$$

The solution to this equation is

$$x = 2.4076$$

which is not readily obtainable by hand. More generally, we would want to solve systems:

$$\begin{aligned}x_1^2 - x_2 \exp(x_1) - 9 &= 0 \\x_2^2 - x_1 &= 0\end{aligned}$$

which has solution

$$(x_1, x_2) = (1.6152, -1.2709).$$

To solve this system:

1. create an m-file that looks like

```
function y = fname(x);
y(1) = x(1)^2 - x(2)*exp(x(1)) - 9.0;
y(2) = x(2)^2 - x(1);
```
2. At the Command Window prompt type

```
x = fsolve('fname',[1 1],optimset('fsolve'))
```

It will return the solution as well as a message telling you either (1) Optimization terminated successfully (you have found a solution) or (2) Optimizer is converging to a solution that is not a root (meaning your initial guess was bad and you should try a different one). For single equation systems like the first example, the function `fzero` will be a bit faster and it allows you to specify the bounds, so that you can search only for roots between two particular points. To solve one equation:

1. create an m-file


```
function y = fname(x);
y = x^2 - x^(2/3) - 4
```
2. At the Command Window prompt type


```
x = fzero('fname',[1 6],optimset('fzero'))
```

Matlab will come back with the answer

$$x = 2.4076.$$

If instead of `[1 6]` you only specify one number (like 1) `fzero` will start searching near 1 but expand outward until it can't anymore. Given two numbers it will stay between them.

Finally, we want to know how to solve constrained minimization problems:

$$\min_{\{x_1, x_2\}} y = 2x_1 + 6x_2$$

subject to

$$\begin{aligned} 0.5 \log(x_1) + 0.5 \log(x_2) &\geq -2.0 \\ x_1 &\geq 0 \\ x_2 &\geq 0 \end{aligned}$$

Solving constrained minimization problems is a bit more complicated because Matlab deals with every type of constraint differently in order to exploit the fastest methods. The file you will use is called `fmincon`. It solves any problem that can be written in the form

$$\min_x F(x)$$

subject to

$$\begin{aligned} Ax &\leq B \\ A_{eq}x &= B_{eq} \\ C(x) &\leq 0 \\ C_{eq}(x) &= 0 \\ L &\leq x \leq U \end{aligned}$$

where A , B , A_{eq} , and B_{eq} are matrices, C and C_{eq} are nonlinear functions, and L and U are vectors of lower and upper bounds. You must specify each piece if you are using something that comes after it and pass it to `fmincon` in the right order. That is, if you intend to have A_{eq} in your problem you must leave spaces for A and B .

1. Define your m-file and save it

```
function y = fname(x)
y = 2*x(1) + 6*x(2);
```

2. In the Command Window set up the matrices and the upper and lower bounds by typing

```
A = []
B = []
Aeq = []
Beq = []
U = []
L = [0.0;0.0]
```

Because the problem does not have linear constraints but does have lower bounds, we get this setup. `[]` means empty matrix, which holds places for the constraints that are not present in the problem. Matlab can do any operation to an empty matrix and it returns another empty matrix.

3. Define an m-file for your nonlinear constraints In this case, we have only C , and be careful to write it in the right way (as a less-than sign).

```
function [C Ceq] = fconstraint(x)
C(1) = -2.0 - 0.5*log(x(1)) - 0.5*log(x(2));
Ceq(1) = [];
```

4. Now type

```
x = fmincon('fname',[1 1],A,B,Aeq,Beq,L,U,'fconstraint',optimset('fmincon'))
which produces the solution:
```

$$(x_1, x_2) = (0.2345, 0.0781).$$

An important feature of these built-in routines is that they can accommodate parameters passed to the functions. Let's say that you want to minimize the function

$$p_1x_1 + p_2x_2,$$

an expenditure function, for many different values of p_1 and p_2 . Then do the following:

1. Define the m-file as

```
function y = fname(x,p);  
y = p(1)*x(1) + p(2)*x(2) (which can be written more compactly as p *  
x)
```

2. make the command call

```
x = fminsearch('fname',[1 1],optimset('fminsearch'),p)
```

where p is a vector of prices you define in the Command Window.

In this way you can pass numbers to many different files but only need to define them in one place, which makes changing them a lot easier. To get additional explanation for, say `fzero`, type `help fzero` at the Command Prompt and the comments will appear, giving you instructions on what each argument of a function means.

We will also have interest in plotting things in Matlab. If you have two vectors x and y , say where $y = x^2$, and you want to plot them you simply type

```
plot(x,y)
```

and this produces a figure with the plot of x versus x^2 . For general functions, you can also use

```
fplot('fname',[1 10])
```

which would provide a plot of the function `fname` over the interval `[1 10]`. Finally, scatterplots can be created where both x and y are functions of some other variable:

```
scatter(x,y)
```

creates a plot of dots in (x, y) space, where each coordinate is (x_i, y_i) . A third argument controls the size of the dots and a fourth one gives you colors.

Example: First create a vector x , say evenly spaced from 0 to 10 with 100 points:

```
x = linspace(0,10,100);
```

`linspace` is simply a routine that produces a vector of a particular size that is equally spaced between two fixed points. Next, calculate y as $\exp(x)$:

```
y = exp(x);
```

this will do the exponentiation term by term. Then plot them as above.

Finally, sometimes we will want to do operations within a loop. A 'for' loop is executed a fixed number of times:

```
for i=1:100
```

does an operation 100 times, with the counter variable i increasing from 1 to 100 in steps of 1. You can change the increment size:

```
for i=1:5:100
```

goes in steps of 5. You can also go downward:

```
for i=100:-1:1
```

counts down in increments of -1 . A 'while' loop executes until a condition is met:

```
while x<=10
```

would repeat until x was larger than 10. Both types of loops are terminated by an 'end' statement. 'break' will jump you out of the current loop. It is generally better programming practice to use while instead of for and break.