Lesson 01: MATLAB at a Glance

MATLAB can be unbelievably *frustrating...*

I have lost count of the number of times this program has caused me immeasurable amounts of pain through no fault of my own. With poor language design, undocumented quirks, and, at times, seemingly non-deterministic behavior, MATLAB is a beast to tame. It will shamelessly stab you when you least expect it and show you absolutely zero remorse!

That said, it can be incredibly useful when the stars align, so here we are...

The MATLAB Environment

Before we jump head first into the fundamentals of MATLAB, it helps if we take a step back and go over how to interact with its environment.

So What's an M-File?

M-files, denoted with a `.m` extension, are the primary method for scripting MATLAB operations. MATLAB executes the contents of these files as if you had typed them explicitly into the command prompt. Confusingly, M-files can be both executable user scripts or define functions, which we can then reference in other M-files. To execute the contents of an M-file, type its name into the command prompt without the `.m` extension.

But Where to Find M-Files?

The MATLAB Search Path[1] specifies the paths MATLAB will search for M-files. It will first search the current working directory (accessible with `pwd`), your `userpath`, paths specified in the `MATLABPATH` environment variable, and finally, paths added by installed packages. To view the paths in the current search path, you can use the `path` command. To locate the path of an M-file, use the `which` command followed by the name of the M-file you're trying to find.

User-Defined Startup Script

If you'd like to execute some commands before MATLAB starts, you can add a `startup` M-file in the search path. This file is handy if you wish to modify the

[1] `https://www.mathworks.com/help/matlab/matlab_env/what-is-the-matlab-search-path.html`

environment at runtime (i.e. set graphic options). Feel free to take a look at my `startup.m` if you'd like to see an example.

Fundamentals

Now that we have the *slightest* idea of what's going on, it's out the frying pan and into the fire!

Everything's a Matrix!

Who would have thought that MATrix LABoratory represents its data as matrices? Matrices generally have three forms:

- Matrices of the form $\mathrm{M}_{m\times n}(\mathbb{F})$
- Vectors of the form $\mathrm{M}_{m\times 1}(\mathbb{F})$ or $\mathrm{M}_{1\times n}(\mathbb{F})$
- Scalars of the form $\mathrm{M}_{1\times 1}(\mathbb{F})$

Certain MATLAB commands may want a specific type of matrix (i.e. a row vector) or may operate on different portions of a matrix (i.e. its columns). We will see examples of this later. For now, let's go over how you might instantiate different types of matrices:

Aside— Well, actually, these are all represented as N-dimensional arrays with an infinite amount of singleton dimensions.[2]

```
c = 13;             % create a scalar
u = [5, 6, 7, 8];   % create a row vector
v = [1; 2; 3; 4];   % create a column vector

A = [               % create a matrix
     1, 3
     4, 2
    ];

B = [A, A];         % we can even compose matrices!
```

Array & Matrix Operations

Array operations execute on a per-element basis, while matrix operations obey the rules of linear algebra.

[2] https://blogs.mathworks.com/loren/2006/08/09/essence-of-indexing/

Operator	Purpose
`A + B`	Addition
`+A`	Unary addition
`A - B`	Subtraction
`-A`	Unary subtraction
`A .* B`	Element-wise multiplication
`A .^ B`	Element-wise power
`A ./ B`	Right array division
`A .\ B`	Left array division
`A.'`	Array transpose

Table 1: Array Operations

Operator	Purpose
`A * B`	Multiplication
`A \ B`	Left division
`A / B`	Right division
`A ^ B`	Power

Table 2: Matrix Operations

Memory Representation

MATLAB stores matrices in column-major order. Each entry can be real or complex-valued. In the case of a matrix consisting of complex values, MATLAB will internally allocate two matrices: one for the real component of an entry and another for the complex component of an entry. Although you don't see this duplicate matrix business on your end, it is important to keep this in mind as each element of these matrices is an IEEE 754 double[3], which can use a lot of memory as matrices grow in size!

[3] `https://en.wikipedia.org/wiki/IEEE_754`

Sparse Matrices

Sparse matrices[4] are matrices that consist mostly of zero entries. Although they *can* be represented as regular matrices, once they grow to insane sizes (i.e. $\mathrm{M}_{1{,}000{,}000\times 1{,}000{,}000}(\mathbb{F})$), the walls of the real world start crashing down. To get around this, MATLAB matrices either have a full or sparse ***storage class***, the latter made for efficiently storing sparse data.

MathWorks has worked hard for full and sparse matrices to play nicely, and for the most part, they do as almost any operation that works on a full matrix works for a sparse matrix. With the exception of a few operations, any operation performed with a sparse matrix will result in a sparse matrix.

To learn more about sparse matrices, visit `https://www.mathworks.com/help/matlab/sparse-matrices.html`, and if you're interested in their implementation, feel free to read `https://www.mathworks.com/help/pdf_doc/otherdocs/simax.pdf`.

Documentation

If you don't know what a function does, the `help` and `doc` commands are your best friends. There's also the official MATLAB documentation[5] and its PDF version[6] for those who like something you can sit and read. When all else fails, you can always just read the code `;)`

[4] `https://en.wikipedia.org/wiki/Sparse_matrix`
[5] `https://www.mathworks.com/help/matlab/`
[6] `https://www.mathworks.com/help/pdf_doc/matlab/`

