Getting Started with Matlab (in Computer Science at UBC)

Ian Mitchell
Department of Computer Science
The University of British Columbia
Outline

• Why Matlab?
  – Why not C / C++ / Java / Fortran?
  – Why not Perl / Python?
  – Why not Mathematica / Maple?

• A Brief Taste of Matlab
  – where to find it
  – how to run it
  – interactive Matlab
  – m-files & debugging

• Sources of Additional Information
Choosing Programming Languages

- Compare / contrast compiled languages C++ and Java

**C++**
- Fast: close to hardware
- Flexible: interfaces to almost any other language
- Flexible: pointers, references, explicit memory allocation
- Flexible: everybody provides C / C++ libraries
- Popular: commonly used, available everywhere
- Prone to bugs: complex syntax, memory leaks

---

**Java**
- Easy to use: references only, garbage collection
- Popular: commonly used, widely available
- Portable: common byte code
- Developed with a clear vision: Standard libraries for security, threading, distributed systems
- Slower: interpreted or JIT for byte code
The Right Tool for the Job

• C / C++ / Fortran:
  – Statically typed and compiled languages
  – Well developed algorithm, known platform, execution time is key

• Java:
  – Simpler, partially compiled language
  – Unknown platform, less experienced programmer, development time is important, broad standard library

• Perl / Python:
  – Interpreted “dynamic” languages: no typing, no compilation(?)
  – Unknown platform, development time is key, concise but powerful code, huge standard library

• Many others (8000+)

http://en.wikipedia.org/wiki/Comparison_of_programming_languages
The Job: Scientific Computing

• Why not use Numerical Recipes / LAPACK / BLAS?
  – “simple” CLAPACK routine for solving $Ax = b$ (general $A$):
    ```c
    int dgesv_(integer *n, integer *nrhs, doublereal *a, integer *lda,
              integer *ipiv, doublereal *b, integer *ldb, integer *info)
    ```
  – what library to use to plot $\sin(2\pi x)$ for $x \in [0, 1]$?
  – too much programmer overhead: time consuming and too many opportunities for mistakes

• Why not use Mathematica / Maple?
  – Originally designed for symbolic mathematics
  – Some numerical capabilities, but not as efficient to code and/or execute
MATLAB®

• Why use Matlab?
  – robust, dependable, easy to use routines for all basic linear algebra
  – intuitive, untyped, imperative language with garbage collection
  – huge library (toolboxes) of mathematical functions and algorithms
  – fast implementation of vector/matrix operations
  – portable interpreted language widely used in applied mathematics, engineering & physical sciences
  – powerful combination of visualization and debugging

• Why not use Matlab?
  – proprietary system (Mathworks Inc.)
  – occasionally erratic syntax
  – toolbox quality varies widely
  – no support for references/pointers

• Alternatives?
  – Numerical computing: Octave, SciLab, Sage, SciPy, …
  – Plotting: Gnuplot, XGraph, PLPlot, PGPLOT, matplotlib, …
Outline

• Why Matlab?
  – Why not C / C++ / Java / Fortran?
  – Why not Perl / Python?
  – Why not Mathematica / Maple?

• A Brief Taste of Matlab
  – where to find it
  – how to run it
  – interactive Matlab
  – m-files & debugging

• Sources of Additional Information
What is “Matlab”?

• Technically, a single product
  – Contains basic scientific computing tools
  – Linear algebra, quadrature, interpolation, approximation, differential equations
  – GUI, visualization and debugging
  – Programming data and control structures

• Domain specific algorithms packaged in separate “Toolboxes”
  – For example: multivariable optimization, advanced splines, image processing, neural networks, … (40+ available)
  – Why? Separate product development (eg: $$$)

• Additional products (not used in CS courses)
  – Simulink: simulation & model-based design
  – Engines, coders, targets, links, etc.
Where to find Matlab (UBC CS Dept)

• Undergraduate Labs
  – Unix or Windows
  – Release 7.10 (2010a)
  – Toolboxes: image processing, optimization & statistics
  – Licensed only for course work (grad or ugrad)

• Graduate Labs
  – Several releases available, 7.10 (2010a) the default
  – Toolboxes: image processing, image acquisition, statistics, wavelets, neural networks, optimization, PDEs, signal processing, control, robust control, identification
  – Licensed for research work

• Purchase Student Version
  – Full basic Matlab, a few common toolboxes (sufficient for 302/303)
  – Available immediately at UBC Bookstore $150(?)
  – Also available online (US $99), but requires validation of student status for full activation
How to run Matlab

• Windows (or Mac): click Matlab icon to start the Matlab desktop
• Unix (Linux): type `matlab` to start the Matlab desktop
• Command line alternatives:
  – text interface: `matlab -nodesktop`
  – see all the options: `matlab -help`
  – text interface still allows graphical visualization
• Remote use
  – Other than in MS Windows, Matlab uses X Windows for graphics
  – If you are sitting at an X Windows capable machine, you can remotely log into the ugrad Unix machines and use Matlab
  – Linux & Mac already include X Windows support
  – All CS students can download XManager software and get X Windows & ssh capabilities in Windows
  – See [https://www.cs.ubc.ca/support/toc/Undergrads/remote login](https://www.cs.ubc.ca/support/toc/Undergrads/remote login)
  – For faster response times, use Matlab’s text interface and a text-based editor opened inside an ssh window (eg: `emacs -nw`)
Interacting with Matlab

• Examples
  – Getting help
  – Constants: \texttt{pi}, \texttt{i}, \texttt{eps}, \texttt{inf}, \texttt{nan}
  – Matrices & Arrays: input, output, colon, concatenation, \texttt{find}
  – Operators: transpose, arithmetic, element-wise, logical
    • help topics: punct, ops, relop, arith, slash
  – Functions: \texttt{zeros}, \texttt{ones}, \texttt{diag}, \texttt{eye}, \texttt{rand}, \texttt{reshape}, \texttt{size}, ...
  – Text I/O: semicolon, ellipses, \texttt{format}, \texttt{diary}
  – Visualization: \texttt{plot}, \texttt{legend}, \texttt{xlabel}, \texttt{ylabel}, \texttt{title}, \texttt{subplot}, \texttt{set}, \texttt{figure}, \texttt{gcf}, \texttt{gco}, \texttt{close}, \texttt{clf}, ...
  – Graphical I/O: \texttt{print}, \texttt{orient}, \texttt{imread}, ...
  – Workspace management: \texttt{who}, \texttt{whos}, \texttt{save}, \texttt{load}, \texttt{clear}, \texttt{addpath}, ...
  – Other data types: strings, ints, sparse, structures, cells
• Command line includes tab completion, up & down arrow to find previous similar commands, ctrl-c to break execution
Programming

• Standard programming control flow constructs
  – All compound statements finish with `end`
  – `if/elseif/else`, `for`, `while`, `switch/case/otherwise`, `try/catch`, `continue`, `break`, `return`
  – Be careful with boolean operators, matrices and control flow (use `any`, `all`)

• Sequences of commands can be stored as a script in an m-file
  – Type name of file to execute commands (which run in the top level “workspace” scope)
  – Use “%” to denote comment lines

• Functions are m-files that start with `function` command
  – Have input and output parameters, local scope
  – May contain subfunctions and/or nested functions
  – Matlab also supports anonymous functions and a function handle datatype (`help function_handle`)
Data Structures

• No need to predefine variables
  – Variable is created in the current workspace when it appears on the left side of an assignment
• Many data types available
  – By default, all variables are two dimensional double precision floating point arrays
  – Higher dimensional arrays allowed (but no one-dimensional array)
  – Other types: single precision, integer, boolean, strings (specially interpreted double arrays), structures (actually more like dictionaries), cell arrays, function handles, classes
  – No pointers, (almost) no references
  – Dynamically typed: Matlab tries to determine a consistent type, but type errors can occur
• Function arguments are pass by value
  – Changes to input variables are not externally visible unless the same variables are returned as outputs
  – Copy on write implementation ensures fast execution if inputs are not modified
Debugging

• With Matlab desktop and editor
  – Breakpoints can be set and removed by clicking to the right of the (executable) line in the file
  – Single stepping can be accomplished with buttons at the top of the editor window

• Text based
  – Commands `dbstop` (set breakpoint), `dbstep` (single line step), `dbcont` (continue), `dbstatus` (current program counter), `dbstack` (examine call stack), etc.
  – see `help debug` for a full list
  – Extremely useful: `dbstop if error` causes Matlab to stop in the workspace (eg local scope) of the function that caused the error
  – Also: `keyboard` command is equivalent to setting a breakpoint

• In either version, you can examine the current workspace
  – Examine variable values (text or plots), call other Matlab functions, move up and down through the stack
Efficient Matlab Coding

• Use Matlab’s built-in functions
  – eg: `total = 0; for k = 1 : 10; total = total + k; end`
  vs `total = sum(1:10);`

• Preallocate arrays
  – eg: `n = 100; ys = zeros(n, 1);
    for k = 1 : n; ys(i) = sin(2*pi*k/n); end`

• Use “vectorization”
  – rather than loops, try operations that work on entire array of data at once.
  – eg: element-wise operations (arithmetic or boolean),
    `sin(2*pi*(0:0.1:1)), find`
  – version 7.0 onward: built-in JIT often makes loops fast

• Use functions, not scripts
• Use profiler to find slow code
• If all else fails, use MEX interface to C/C++/Fortran
Outline

• Why Matlab?
  – Why not C / C++ / Java / Fortran?
  – Why not Perl / Python?
  – Why not Mathematica / Maple?

• A Brief Taste of Matlab
  – where to find it
  – how to run it
  – interactive Matlab
  – m-files & debugging

• Sources of Additional Information
Sources of Further Information

• Matlab has extensive built-in documentation
  – Hyperlinked documentation: `helpdesk` and `doc <command>`
  – Basic textual command & function info: `help <command>`
  – Unknown command search: `lookfor <string>`
  – Implementation details: `type <command>`

• Online documentation
  – Mathworks website: [http://www.mathworks.com](http://www.mathworks.com)
  – Community code: [http://www.matlabcentral.com](http://www.matlabcentral.com)
  – Matlab resources website (including these slides): [http://www.cs.ubc.ca/~mitchell/matlabResources.html](http://www.cs.ubc.ca/~mitchell/matlabResources.html)
Getting Started with Matlab (in Computer Science at UBC)

For more information contact

Ian Mitchell
Department of Computer Science
The University of British Columbia

mitchell@cs.ubc.ca
http://www.cs.ubc.ca/~mitchell