MATLAB

R. J. Renka

Department of Computer Science & Engineering
University of North Texas

01/19/2012
Matlab is a high-level language invented in the late 1970’s by Cleve Moler while he was chairman of the CS department at the University of New Mexico.

The name comes from Matrix Laboratory. It originally included only linear algebra, and was designed to be a teaching tool. It was essentially an interactive interface to LINPACK and EISPACK.

The fundamental data type was originally a double precision or complex matrix. It is now an array, dense or sparse, of two or more dimensions, containing any of the primitive types, as well as structures, java classes, function handles, and cells.

Matlab provides powerful array syntax. Control constructs have simple syntax; no header files are required; and variables need not be declared.
**Comparison with Alternatives**

**Matlab** serves as an alternative to C/C++, Fortran, Python, etc. for scientific computing. The salient properties that set it apart are as follows.

**Proprietary**  
Matlab is proprietary but well-maintained and widely used, especially by engineers. A compiler is available so that executables can be run on systems that do not have Matlab installed. Also, Octave is an open-source language that is mostly compatible with Matlab.

**Interpreted**  
Matlab can be used like a calculator with the addition of arbitrary precision, symbolic processing (Maple), both GUI and command-line help, and use of scripts. Debugging is easy, with informative error messages based on global information. There is a price in the inefficient use of storage, but with vectorization and just-in-time compiling, there may be no loss of execution efficiency.
Extensive IDE  The IDE includes the usual editor, profiler, debugger, source control system, and lint, as well as extensive help and documentation, graphics, GUI-building tools, interactive plot editing, data import and export, and external interface to Java classes, DLL’s, and functions written in C or Fortran. Furthermore, there are built-in functions and toolboxes for numerical methods, graphics, and engineering applications.

Uniform UI  The language and entire IDE are nearly identical on workstations running Windows, Unix/Linux, and Apple OS X. However, **MATLAB** only runs on workstations.

Flexible  Choices include procedural or object-oriented code, loops or vectorized code, and low-level or high-level access to graphics, numerics, and UI building.
**Cartalk Puzzler:** You’re 75 miles from your destination. You drive one mile at 75 mph, then the next mile at 74 mph, the next at 73 mph, etc., and finally the last mile at one mph. How long does it take?

**Solution** The sequence of time requirements is $1/75$ hours, $1/74$ hours, etc., giving a sum of

$$
\sum_{i=1}^{75} \frac{1}{i} \approx 4.9014
$$

computed in MATLAB as

$$
\gg \text{sum}(1./(1:75))
$$
**Identifiers:** Variable and function names are case-sensitive, begin with a letter, and include only letters, digits, and the underscore character. The maximum length is 63 characters.

**Predefined Variables:** ans, computer, eps, false, i, inf, intmax, intmin, j, Nan, pi, realmax, realmin, true, version

**Math Functions:** abs, exp, factorial, log, log10, sqrt, cos, sin, tan, cot, acos, asin, atan, acot, cosh, sinh, tanh, coth, ceil, fix, floor, rem, round, sign

**Commands:** addpath, clc, clear, dir, exist, format, help, hold, keyboard, ls, lookfor, pwd, which, who, whos

**I/O Commands:** input, fscanf, fopen, fread, fwrite, fgetl, fgets, disp, display, fprintf, error, fclose, xlsread, xlswrite
A vector is a matrix with one row or one column.

**Transpose Operator:** A’ denotes the transpose of A.

**Column Vector:** \([1;2;3]\) or \([1 \ 2 \ 3]'\)

**Row Vector:** \([1 \ 2 \ 3]\) or \([1,2,3]\) or \([1;2;3]'\) or \([\text{first:incr:last}]\), where the brackets are optional, and the increment defaults to 1; e.g.,

\([-3:1:2] \equiv -3:2 \equiv [-3 \ -2 \ -1 \ 0 \ 1 \ 2] \equiv \text{linspace}(-3,2,6)\).

Function `linspace` is defined by

\(\text{linspace(first,last,n)} \equiv [\text{first}:(\text{last-first})/(n-1):\text{last}]\)

with \(n\) as an optional argument (defaulting to 100).

A row vector or matrix \(v\) is converted to a column vector by

\(v = v(:)\).
Array indices start at 1, and 2-D arrays are stored in column-major order (like Fortran, and unlike C).

Example:

```matlab
>> A = [1:2:11; 0:5:25; linspace(10,60,6); 1 2 3 4 5 6]
A =
   1   3   5   7   9  11
   0   5  10  15  20  25
  10  20  30  40  50  60
   1   2   3   4   5   6
```

Special Matrices: zeros(m,n), ones(m,n), eye(n), rand(m,n), randn(m,n), magic(n)

Note that zeros(n) ≡ zeros(n,n), and n defaults to 1 so that, for example ones ≡ eye ≡ 1.
Subarrays

Array references use parentheses.

A(m:n, p:q) specifies rows m to n, columns p to q of A.

A(:, j) specifies column j of A.

Example:

```matlab
>> A = [10:-1:4; ones(1,7); 2:2:14; zeros(1,7)]
   A =
      10    9     8     7     6     5     4
      1     1     1     1     1     1     1
      2     4     6     8    10    12    14
      0     0     0     0     0     0     0
>> B = A([1,3], [1,3,5:7])
   B =
      10     8     6     5     4
      2     6    10    12    14
```
Array-handling Functions:

- `length(v)`: Number of elements in vector `v`
- `size(A)`: Row vector `[m n]` containing dimensions of `A`
- `reshape(A,m,n)`: `m` by `n` matrix, where `m*n` is the total number of elements in `A`; e.g., `reshape(A,m*n,1) ≡ A(:)`
- `diag(v)`: Creates a square matrix with diagonal `v`
- `inv(A)`: Inverse of square matrix `A`
- `det(A)`: Determinant of square matrix `A`

**Strings** are arrays of characters. Constant strings are delimited by single quotes. An array of strings can be created by the function `char('string 1', 'string 2', ...)` which blank-fills the shorter strings if necessary.
Matrix Addition and Subtraction: $C = A + B$, $C = A - B$. If arrays $A$ and $B$ are not the same size, one of them must be a scalar.

Matrix Multiplication: $C = A \times B$

Scalar Multiplication: $C = s \times A$ or $C = A \times s$

Left Division: $x = A \backslash b$ solves $Ax = b$ by Gaussian elimination

Right Division: $x' = b' / A$ solves $A^T x = b$ by Gaussian elimination

Algebraically $A \backslash B = A^{-1} B$ and $A / B = AB^{-1}$ but no matrix inverse is computed. $A$ need not be invertible or even square: a least-squares problem is solved if necessary.

Exponentiation: $C = A^s$
Additional Operations

Element-wise Operations:
- Multiplication: .*
- Right Division: ./
- Left Division: .\
- Exponentiation: .^  

Operands must have the same size, but a scalar is implicitly expanded to the size of the other operand.

Function Vectorization: The built-in math functions are applied element-wise, so that \texttt{f(A)} has size equal to \texttt{size(A)}.

Relational Operators: <, >, <=, >=, ==, ~=

Logical Operators: & , | , ~ , xor(a,b), && , ||

Logical Functions: all , any , find
Built-in Functions with Vector Arguments: mean, max, min, sum, median, std

If the argument is a matrix, the return value is a row vector obtained by applying the operation to each column.

For vectors $a$ and $b$ in $\mathbb{R}^n$ (either rows or columns),

- $\text{cross}(a, b)$ returns $a \times b$ as a row vector for $n = 3$, and
- $\text{dot}(a, b)$ returns $a^T b$.

A row vector containing a random permutation of $\{1, 2, \ldots, n\}$ is generated by

- $\text{randperm}(n)$
1) **Conditional Control**

```matlab
if logical-expression
    statements
elseif logical-expression
    statements
else
    statements
end
```

If logical-expression is a matrix, all elements must be nonzero for the value to be true. An empty array is treated as false.
switch expression
  case value1
    statements
  case value2
    statements
  ...
  otherwise
    statements
end

- The expression may be a scalar or string. If expression is a string, the test for a match is strcmp(value,expression) rather than value == expression.
- Unlike the C switch statement, break is not needed to prevent 'fall through'.
- A value may be a cell array {value1,value2}; either may match.
- The otherwise block is optional.
2) **Loop Control** — usually vectorizable

```matlab
for index = start: increment: end
    statements
end
```

- increment defaults to 1.
- Following execution, index has the value last assigned.

```matlab
for k = A
    statements
end
```

- if A is m by n, the number of iterations is n, and k takes on the m-vectors A(:,1), A(:,2), ..., A(:,n).
Loop Control: while

while expression
    statements
end

- if expression is a matrix, all elements must be 1 for execution to continue; the empty array represents false.

break: exits the innermost loop.

continue: passes control to the next iteration of the innermost loop.

3) Program Termination

return: terminates a function, script, or keyboard mode, returning control to the invoking function or keyboard.
function [output args] = function_name (input args)
%   H1 line
%
%   Help text
%
     Function body
end

- Keyword 'function' must be lowercase.
- Square brackets are needed only if there is more than one output argument; '=' may be omitted if there is no output.
- The name of the primary function should coincide with the name of the M-file.
- 'end' is required only if a nested function is included in the M-file.
- Each function has its own workspace — separate from the base workspace used by the command window and scripts.
**Built-in Function**: A function that is frequently used or costly to evaluate. The M-file exists but contains only help text.

**Primary Function**: First function in an M-file; callable from the command line and from other functions without creating a handle.

**Subfunction**: Function added to an M-file after the primary function; name can override another function because it is found first; help text is accessed by ’help primary>subfunction’.

**Nested Function**: Function defined inside the body of another function; has its own workspace but can also access the workspace of its parent.

**Overloaded Function**: Function whose M-file is contained in a subdirectory named ’@type’.

**Private Function**: Function whose M-file is contained in a subdirectory named ’private’ — searched first.
Semantically, arguments are passed by value, but MATLAB optimizes away unnecessary copy operations.

nargin and nargout are functions that evaluate to the number of input and output arguments in the call. The call may omit parameters from right to left. The function may compute all output values or omit some for increased efficiency if they are omitted from the call.

The last argument in a list may be varargin or varargout specifying cell arrays containing arbitrary numbers of arguments.
Function handles are used to pass functions to *function functions* and for Handle Graphics callbacks. The statement

\[
\text{fhandle} = @\text{functionname}
\]

maps the handle to the function currently associated with functionname, and retains that mapping even if the function goes out of scope (such as if the MATLAB path were changed to give precedence to a different function). An *anonymous function* is created by

\[
\text{fhandle} = @(\text{arglist})\text{expression}
\]

where the expression may include other variables, using their values at the time the handle is created; e.g.,

\[
sqr = @(x) x.^2;
\]
Following are efficient (but not obvious) solutions of some commonly encountered implementation problems.

A polynomial is represented by a row vector of coefficients ordered from the highest to lowest power; e.g., \([1 \ 0 \ -2 \ -5]\) represents \(x^3 - 2x - 5\).

**Problem 1:** Compute a sum of polynomials, \(r = p + q\).

**Solution:**

\[
\begin{align*}
  k &= \text{length}(q) - \text{length}(p); \\
  r &= \text{zeros}(1,k) \ p + \text{zeros}(1,-k) \ q;
\end{align*}
\]
Problem 2: Evaluate a polynomial, \( y = p(x) \), where \( x \) may be a vector.

Solution:

\[
y = 0;
\text{for } c = p \\
\quad y = y \times x + c;
\text{end}
\]

Problem 3: Given an \( n \) by 1 vector \( v \), vectorize the following loop.

\[
\text{for } j = n:-1:1 \\
\quad v(j+1) = v(j);
\text{end}
\]

Solution:

\[
j = 1:n; \\
v(j+1) = v(j);
\]
Problem 4: Given an ordered sequence of abscissae $x(1) < x(2) < \ldots < x(n)$, and a vector or matrix $u$ of evaluation points, find the vector or matrix $k$ of subinterval indices such that $x(k) \leq u < x(k+1)$.

Solution:

```matlab
n = length(x);
k = ones(size(u));
for j = 2:n-1
    k(x(j) <= u) = j;
end
```