MATLAB has five flow control constructs:
- if statements
- switch statements
- for loops
- while loops
- break statements

**if statement**

```matlab
if A > B
    'greater'
elseif A < B
    'less'
elseif A == B
    'equal'
else
    error('Unexpected situation')
end
```

>, < and == work with **scalars**, but NOT matrices.

### Matrix Comparison - Beware!

Matrix comparison involves creating comparison matrices and logical matrices.

**Built-in Logic functions for matrices**

Several functions are helpful for reducing the results of matrix comparisons to scalar conditions for use with if, including:

- `isequal(A,B)`: returns '1' if A and B are identical, else return '0'
- `isempty(A)`: returns '1' if A is a null matrix, else return '0'
- `all(A)`: returns '1' if all elements A is non-zero
- `any(A)`: returns '1' if any element A is non-zero

```matlab
if isequal(A,B)
    'equal'
else
    'not equal'
end
```

### Control Flow - Switch & Case

Assume `method` exists as a string variable:

```matlab
switch lower(method)
    case {'linear','bilinear'}
        disp('Method is linear')
    case 'cubic'
        disp('Method is cubic')
    case 'nearest'
        disp('Method is nearest')
    otherwise
        disp('Unknown method.')
end
```

Use otherwise to catch all other cases.
Control Flow - For Loop

This makes it faster and uses less memory

\[
n = 4; \\
a = \text{zeros}(n,n) \% \text{Preallocate matrix} \\
\text{for} \ i = 1:n \\
\quad \text{for} \ j = 1:n \\
\quad \quad H(i,j) = 1/(i+j); \\
\quad \text{end} \\
\text{end}
\]

Life is too short to spend writing for-loops

Create a table of logarithms:

\[
x = 0; \\
\text{for} \ k = 1:1001 \\
\quad y(k) = \log_{10}(x); \\
\quad x = x + .01; \\
\text{end}
\]

A vectorized version of the same code is

\[
x = 0:.01:10; \\
y = \log_{10}(x);
\]

Control Flow - While Loop

Find root of the polynomial $x^3 - 2x - 5$ ..

\[
a = 0; \ fa = -\text{Inf}; \\
b = 3; \ fb = \text{Inf}; \\
\text{while} \ b-a > \text{eps*b} \\
\quad x = (a+b)/2; \\
\quad fx = x^3 - 2*x - 5; \\
\quad \text{if} \ \text{sign}(fx) == \text{sign}(fa) \\
\quad \quad a = x; \ fa = fx; \\
\quad \text{else} \\
\quad \quad b = x; \ fb = fx; \\
\quad \text{end} \\
\text{end}
\]

Control Flow - break

The \text{break} statement lets you exit early from a for or while loop.

In nested loops, break exits from the innermost loop only.

Why is this version of the bisection programme better?
Matrix versus Array Operations

Matrix Operators

Addition or unary plus. \( A + B \) adds \( A \) and \( B \). \( A \) and \( B \) must have the same size, unless one is a scalar. A scalar can be added to a matrix of any size.

Subtraction or unary minus. \( A - B \) subtracts \( B \) from \( A \). \( A \) and \( B \) must have the same size, unless one is a scalar. A scalar can be subtracted from a matrix of any size.

Matrix multiplication, \( C = A \times B \) is the linear algebraic product of the matrices \( A \) and \( B \). For nonscalar \( A \) and \( B \), the number of columns of \( A \) must equal the number of rows of \( B \). A scalar can multiply a matrix of any size.

\( / \) Slash or matrix right division, \( B/A \) is roughly the same as \( B \cdot inv(A) \). More precisely, \( B/A = (A \cdot B')' \). See \( \cdot \).

\( \backslash \) Backslash or matrix left division. If \( A \) is an \( n \times n \) matrix and \( B \) is a column vector with \( n \) components, or a matrix with several such columns, then \( X = A \cdot B \) is the solution to the equation \( A \cdot X = B \).

\( ^\wedge \) Matrix power, \( X^p \) is \( X \) to the power \( p \), if \( p \) is a scalar. If \( p \) is an integer, the power is computed by repeated multiplication.

\( ^\dagger \) Matrix transpose, \( A^\dagger \) is the linear algebraic transpose of \( A \). For complex matrices, this is the complex conjugate transpose.

Array Operators

- Element-by-element addition or unary plus.
- Element-by-element subtraction or unary minus.
- Array multiplication. \( A \cdot B \) is the element-by-element product of the arrays \( A \) and \( B \). \( A \) and \( B \) must have the same size, unless one of them is a scalar.
- Array right division. \( A / B \) is the matrix with elements \( A(i,j)/B(i,j) \). \( A \) and \( B \) must have the same size, unless one of them is a scalar.
- Array left division. \( A \backslash B \) is the matrix with elements \( B(i,j)/A(i,j) \). \( A \) and \( B \) must have the same size, unless one of them is a scalar.
- Array power. \( A \wedge B \) is the matrix with elements \( A(i,j)^{B(i,j)} \). \( A \) and \( B \) must have the same size, unless one of them is a scalar.
- Array transpose. \( A^\prime \) is the array transpose of \( A \). For complex matrices, this does not involve conjugation.

M-files: Scripts and Functions

There are two kinds of M-files:
- Scripts, which do not accept input arguments or return output arguments. They operate on data in the workspace.
- Functions, which can accept input arguments and return output arguments. Internal variables are local to the function.

% Investigate the rank of magic squares
r = zeros(1,32);
for n = 3:32
r(n) = rank(magic(n));
end
r
bar(r)
Functions

Return variable

Define function name and arguments

function r = myfunct(x)
% Calculate the function:
% r = x^3 - 2*x - 5
% x can be a vector
r = x.^3 - x.*2 - 5;

% on column 1 is a comment

This is how plot on p.2-27 was obtained

```matlab
X = 0:0.05:3;
y = myfunct(X);
plot(x,y)
```

Scopes of variables

- All variables used inside a function are local to that function
- Parameters are passed in and out of the function explicitly as defined by the first line of the function
- You can use the keyword `global` to make a variable visible everywhere
- As a good programming practice, only use global variables when it is absolutely required

MATLAB Programming Style Guide (1)

- This Style Guideline is originally prepared by Mike Cook
  - The first line of code in `script m-files` should be indicate the name of the file.
  - The first line of `function m-files` has a mandatory structure. The first line of a function is a declaration line. It has the word function in it to identifies the file as a function, rather than a generic m-file. For example, for a function named `abs_error.m`, the first line would be:
    ```matlab
    function [X,Y] = abs_error(A,B)
    ```
  - A block of comments should be placed at the top of the regular m-files, and just after the function definition in function m-files. This is the header comment block. The formats are different for m-files and functions.

Style Guide (2)

- Variables should have meaningful names. This will make your code easier to read, and will reduce the number of comments you will need. However here are some pitfalls about choosing variable names:
  - Meaningful variable names are good, but when the variable name gets to 15 characters or more, it tends to obscure rather than improve code.
  - The maximum length of a variable name is 19 characters and all variables must start with a character (not number).
  - Be careful of naming a variable that will conflict with matlab's built-in functions, or reserved names: if, while, end, pi, sin, cos, etc.
  - Avoid names that differ only in case, look similar, or differ only slightly from each other.
  - Make good use of white space, both horizontally and vertically, it will improve the readability of your program greatly.
Style Guide (3)

- Comments describing tricky parts of the code, assumptions, or design decisions should be placed above the part of the code you are attempting to document.
- Do not add comment statements to explain things that are obvious.
- Try to avoid big blocks of comments except in the detailed description of the m-file in the header block.
- Indenting. Lines of code and comments inside branching (if block) or repeating (for and while loop) logic structures will be indented 3 spaces. NOTE: don’t use tabs, use spaces. For example:

```matlab
for i=1:n
    disp('in loop')
    if data(i) < x
        disp('less than x')
    else
        disp('greater than or equal to x')
    end
    count = count + 1;
end
```

Style Guide (4)

- Be careful what numbers you “hardwire” into your program. You may want to assign a constant number to a variable. If you need to change the value of the constant before you re-run the program, you can change the number in one place, rather than searching throughout your program.

```matlab
% This program hardwires the constant 100 in three places in the code.
for i = 1:100
    data = r(i);
end
temp = data/100;
meanTemp = sum(temp)/100;
```

```matlab
% This program assigns the constant value to the variable, n.
% number of data points.
for i = 1:n
    data = r(i);
end
temp = data/n;
meanTemp = sum(temp)/n;
```

Style Guide (5)

- No more than one executable statement per line in your regular or function m-files.
- No line of code should exceed 80 characters. (There may be a few times when this is not possible, but they are rare).
- The comment lines of the function m-file are the printed to the screen when help is requested on that function.

```matlab
function bias = bias_error(X,Y)
% Purpose: Calculate the bias between input arrays X and Y
% Input: X, Y, must be the same length
% Output: bias = bias of X and Y
% filename: bias_error.m
% Mary Jordan, 3/10/96
% bias = sum(X-Y)/length(X);
```

Style Guide (6) - Another good example

```matlab
function [out1,out2] = humps(x)
% Y = HUMPS(X) is a function with strong maxima near x = .3
% and x = .9.
% [X,Y] = HUMPS(X) also returns X. With no input arguments,
% HUMPS uses X = 0:.05:1.
% Copyright (c) 1984-97 by The MathWorks, Inc.
% $Revision: 5.3 $ $Date: 1997/04/08 05:34:37 $
if nargin==0, x = 0:.05:1; end
y = 1 ./ ((x-.3).^2 + .01) + 1 ./ ((x-.9).^2 + .04) - 6;
if nargout==2,
    out1 = x; out2 = y;
else
    out1 = y;
end
```
Function of functions - fplot

FPLOT(FUN,LIMS) plots the function specified by the string FUN between the x-axis limits specified by LIMS = [XMIN XMAX].

Find Zero

FZERO(F,X) tries to find a zero of F. FZERO looks for an interval containing a sign change for F and containing X.

Find minimum

X = FMIN("F",x1,x2) attempts to return a value of x which is a local minimizer of F(x) in the interval x1 < x < x2.

Integration of Curve

Q = QUAD("F",A,B) approximates the integral of F(X) from A to B to within a relative error of 1e-3 using an adaptive recursive Simpson's rule.