ELEC2146

Electrical Engineering Modelling and Simulation

Simulation Programming and MATLAB

Dr Ray Eaton

S2, 2016

Lecture notes written by Dr Julien Epps

Overview

- Programming vs. simulation programming
- Programming principles
- How MATLAB works
- Debugging
- Good programming practise

Simulation Programming

- Most programming:
  - Uses a variety of data structures
  - Need to declare all variables (‘strongly typed’)
  - Often need to worry about memory allocation of variables
  - Visual component = GUI
  - Few pre-defined functions
- Simulation programming:
  - Often uses just numbers, arrays and matrices
  - Visual component = plots (GUI possible)
  - MATLAB
    - No need to worry about declaring variables or memory allocation
  - Nearly every function is predefined

Simulation Programming

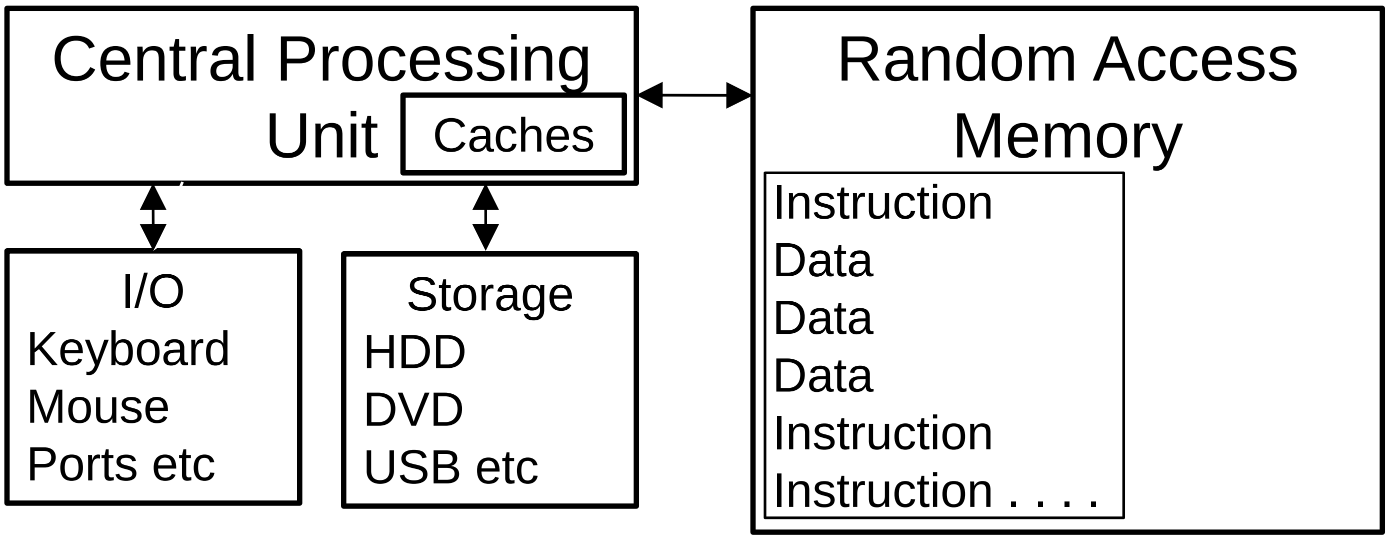
- Comparing programming languages

	MATLAB
– Readability	OK
– Writability	Good
– Reliability	
▪ Everything except memory management	Good
▪ Memory management	Bad
– Cost	
▪ Writing/debugging code	Good
▪ Execution time	Not great
– Portability	Bad
– Generality	Bad

Adapted from Reus, B. "Principles of Programming Languages" http://www.cogs.susx.ac.uk/users/bernhard/ppl2007/Slides/ppl2s.pdf

Programming Principles

- How computers are organised



- Programs are a set of instructions on data, executed by the CPU
- Data exists in storage (slow), RAM (fast) or in a CPU cache (fastest)

Programming Principles

- Top-down design
  - In procedural languages, code development is based on a step-wise refinement of the abstract function that the code is required to perform
  - Breaks a problem into sub-problems
    - And from there into sub-sub-problems
    - Deals with each sub-problem separately
  - Example:

```
% Problem: Get to UNSW
DoINeedToGo(date,timetable);         CheckTimetable(date,timetable);
                                     CheckSocialSchedule(date);

                                     CheckGearNeeded(date,timetable);
WhatDoINeed(date,timetable);         Raining(windowCam);
                                     HotCold(temperature);

                                     DrivingHassle(date,trafficCond);
GetThere(homeGPS,UNSWGPS);           NextBus(date,time);
                                     NextTrain(date,time);
```

Programming Principles

- Top-down design
  - Start solution to sub-problems by writing pseudo-code
    - Comments containing code-like statements or even just text descriptions of the function being performed
  - Wherever possible, find generic pieces of the problem and write them as functions
    - Should be as independent and self-contained as possible
    - Typically have one entry and one exit point
    - Usually short
    - Makes code easier to read; improves underlying logic
    - Makes code easier to test
    - Large projects: can split up work
    - MATLAB: can start new file for function or include it as part of a script file `function b = sqrt(a)`

Programming Principles

- Data structures: MATLAB
  - Most common type of variable: `double`
    - 64-bit double-precision floating-point
    - Can hold real, imaginary and complex numbers
  - Arrays
    - The fundamental data type of MATLAB
    - 1-D, e.g. `array = [1 3 2 4];`
    - 2-D (matrix), e.g. `A = [1 3; 2 4];`
    - 3-D and beyond, e.g. `B(2,1,4) = 16;`
  - Characters (not used much in this course)
  - Cell arrays (not used much in this course)
    - Generic containers – very handy
    - `ExampleCell{1} = [1 3 2 4]; ExampleCell{2} = 56;`

Programming Principles

- MATLAB detaches developer from memory management
  - Dynamic variable size

    `A = 1;`
    - `A` is a scalar: `double A;`

    `A(2) = 4;`
    - `A` is now a 1x2 array: `double A[2];`

    `A(3,2) = -1;`
    - `A` is now a 3x2 matrix: `double A[3][2];`

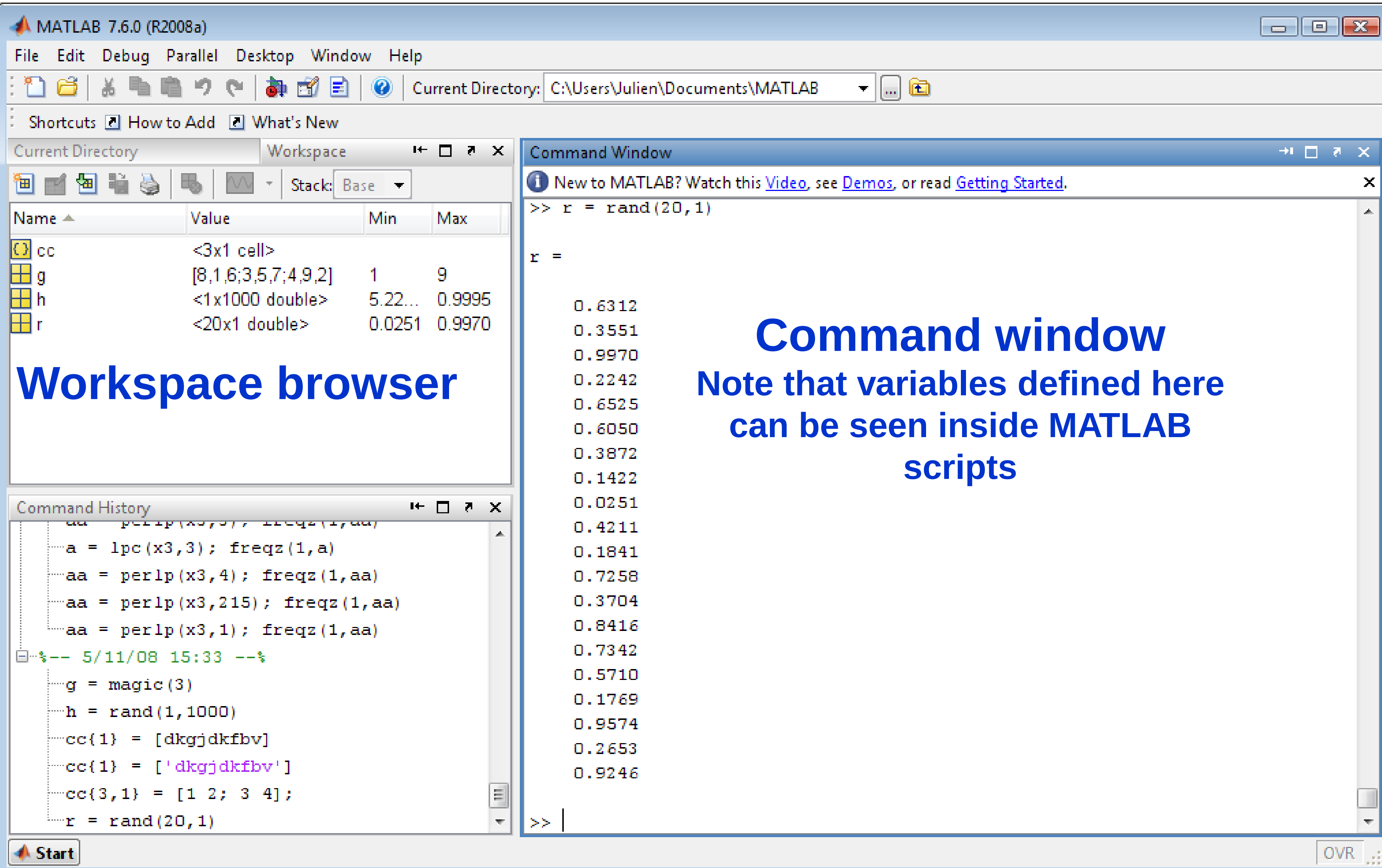
    `A(3,2,2) = 6;`
    - `A` is now 3x2x2: `double A[3][2][2];`
  - Growing an array within a loop

```
B=0; for k=1:100; B(k)=100-k; end
```

How MATLAB works

- High level language
    - Huge amount of abstraction from the machine-level instructions
    - Programming languages don't come much higher level than this
- Procedural language
    - Complex problems can be decomposed into a hierarchy of functions
- Interpreted language
  – Each line of code is decoded as the compiler reaches it
    - Slower than compiled languages. MATLAB now has a compiler
- Based on fast numerical methods
    - LINPACK, EISPACK
- Like an advanced calculator

How MATLAB works



How MATLAB works

- Where data is stored in memory

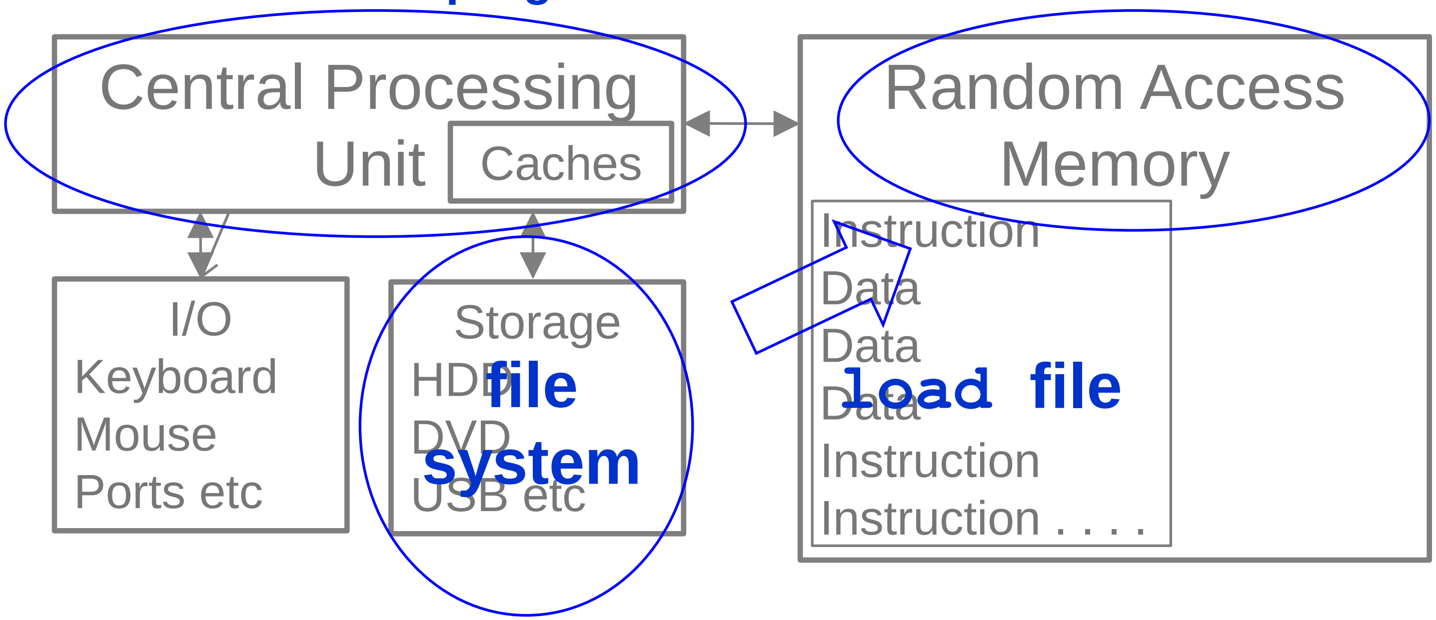
Used during command execution

Not accessible to programmer

MATLAB workspace

command window

workspace browser



How MATLAB works

- Scripts
  - A list of commands, stored in exampleScript.m
  - Execute by running `exampleScript` in the command window
  - Think of scripts as identical to the command window
- Functions
  - Should always be used in preference to scripts, where possible to produce generic, reusable code
  - Take inputs, produce outputs
  - Execute in command window by running e.g.

```
c = dist(a,b);
```

```
function d = dist(x,y)
d = sqrt(sum((x-y).^2));
```

How MATLAB works

- Built-in functions
  - Lots of them
  - Very useful
  - Functions are “overloaded”
    - Same function can different inputs or produce different outputs depending on how it is called
- Some MATLAB tips
  - If your code is slow, avoid `for` loops
    - Try to use vectors and matrices – MATLAB is fast for these
    - Other languages: `for` loops are fine
  - If your code is slow, work out where the problem is
    - Use the profiler: `help profile`
  - If your code is slow, reduce your memory usage

Debugging

- Writing code is the easy part
  - Figuring out *why* it doesn't work is the hard part

Some advice:

- Understand the big objective of debugging: Systematically isolating the point of error
  - Try to eliminate parts of the code that could not have produced the error
- Work out what should have happened where the bug occurred
  - Try the code using a simpler input

Adapted from White, A., "Teaching debugging – giving novices expert knowledge", *Artificial Intelligence in Education*, Boulay and Mizoguchi (Eds) , IOS Press, 1997

Debugging

- MATLAB will usually locate the problem line of code
- Try executing the code in the command window (if using a script or function)
- Try leaving the "`;`" off the end of the command
    - Prints output to command window

```
>> g(1,:) = r
??? Subscripted assignment dimension mismatch.
```

- Check the dimensions of `g` and `r`

Debugging

- Typical problem:
  – "I built it and it doesn't work"
- Understand exactly what "working" means
- Break "it" up into smaller subsections
- Test each subsection
  – With the simplest possible test you can construct
- Better still:
  – Do this as you create the code
  – Test as you go

Good programming practise

- Why ?
  - Good programming is about communication
  - Well-written code is more likely to have fewer errors
  - Well-written code is easier to debug
  - Well-written code is more extensible and re-usable
  - Well-written code is easier for someone else to understand
    - Very important in industry

Good programming practise

- Code formatting
  - A new level of indentation should be used for every nested statement

```
for k = 1:N,
    if array[k] > threshold,
        disp('above');
    else
        disp('below');
    end
end
```

  - MATLAB has “smart indent” – use it

Good programming practise

- Variables
  - Newly declared variables should have a comment explaining their use

```
VC = zeros(1,100); % Array of capacitor voltages
```

  - Use meaningful, descriptive variable names
    - E.g. VC or CapVolt or CapacitorVoltage for a capacitor voltage
  - Don’t change the value of loop variables within a loop
  - Avoid use of global variables
    - Easy if you use functions
  - Pre-allocate arrays to some fixed length
    - Not critical for MATLAB (no need to declare arrays), but faster and good practise
  - Avoid using *i*, *j* – MATLAB complex numbers

Good programming practise

- Function headers
  - All useful functions/subroutines/scripts/methods have headers
  - Should give:
    - The name of the function
    - Its purpose
    - Description of all inputs
    - Description of all outputs
    - Author, date
    - Version number (if you create more than one version)

```
% function C = lbg(data,csize)
%
% LBG algorithm for codebook design
% data  : matrix with training vectors in rows
% csize : desired codebook size (must be a power of 2)
% C     : codebook with csize rows
%
% Author: Julien Epps           Date: A few years ago
% Reference: Linde, Y., Buzo, A., and Gray, R. M. (1980). "An algorithm for vector quantiser
%            design", IEEE Trans. Commun., vol. COM-28, no. 1, pp. 84-95, January.
```

Good programming practise

- Use pseudocode

```
% linfilt - plots various quantities for a digital filter

% Define filter
NumCoeff = [1 -2 1] % numerator coefficients
DenCoeff = 1;       % denominator coefficients

% Plot frequency response
freqz(NumCoeff,DenCoeff);

% Plot pole-zero diagram
figure;
zplane(NumCoeff,DenCoeff);

% Plot impulse response
figure;
stem(NumCoeff);     % filter is FIR

% Plot step response
figure;
Step = [zeros(1,100) ones(1,100)];
y = filter(NumCoeff,DenCoeff,Step);
plot(y);
```

Good programming practise

- Keep your code modular
  - Use functions
- Bad:

```
for k = 1:N,
    if sqrt(sum((x(k,:)-y).^2)) < min,
        d(k) = sqrt(sum((x(k,:)-z).^2));
    end
end
```

- Good:

```
for k = 1:N,
    if dist(x(k,:),y) < min,
        d(k) = dist(x(k,:),z);
    end
end
```

New function `dist` created by the programmer

```
function [c,lags] = xcorr(x,varargin)
%XCORR Cross-correlation function estimates.
%   C = XCORR(A,B), where A and B are length M vectors (M>1), returns
%   the length 2*M-1 cross-correlation sequence C. If A and B are of
%   different length, the shortest one is zero-padded.  C will be a
%   row vector if A is a row vector, and a column vector if A is a
%   column vector.
%
%   XCORR produces an estimate of the correlation between two random
%   (jointly stationary) sequences:
%          C(m) = E[A(n+m)*conj(B(n))] = E[A(n)*conj(B(n-m))]
%   It is also the deterministic correlation between two deterministic
%   signals.
%
%   XCORR(A), when A is a vector, is the auto-correlation sequence.
%   XCORR(A), when A is an M-by-N matrix, is a large matrix with
%   2*M-1 rows whose N^2 columns contain the cross-correlation
%   sequences for all combinations of the columns of A.
%   The zeroth lag of the output correlation is in the middle of the
%   sequence, at element or row M.
%
%   XCORR(...,MAXLAG) computes the (auto/cross) correlation over the
%   range of lags:  -MAXLAG to MAXLAG, i.e., 2*MAXLAG+1 lags.
%   If missing, default is MAXLAG = M-1.
%
%   [C,LAGS] = XCORR(...)  returns a vector of lag indices (LAGS).
%
%   XCORR(...,SCALEOPT), normalizes the correlation according to SCALEOPT:
%     'biased'   - scales the raw cross-correlation by 1/M.
%     'unbiased' - scales the raw correlation by 1/(M-abs(lags)).
%     'coeff'    - normalizes the sequence so that the auto-correlations
%                  at zero lag are identically 1.0.
%     'none'     - no scaling (this is the default).
%
%   See also XCOV, CORRCOEF, CONV, CCONV, COV and XCORR2.
```

```
%   Author(s): R. Losada
%   Copyright 1988-2004 The MathWorks, Inc.
%   $Revision: 1.16.4.4 $  $Date: 2007/12/14 15:06:38 $
%
%   References:
%     S.J. Orfanidis, "Optimum Signal Processing. An Introduction"
%     2nd Ed. Macmillan, 1988.

error(nargchk(1,4,nargin,'struct'));

[x,nshift] = shiftdim(x);
[xIsMatrix,autoFlag,maxlag,scaleType,msg] = parseinput(x,varargin{:});
if ~isempty(msg), error(generatemsgid('SigErr'),msg); end

if xIsMatrix,
    [c,M,N] = matrixCorr(x);
else
    [c,M,N] = vectorXcorr(x,autoFlag,varargin{:});
end

% Force correlation to be real when inputs are real
c = forceRealCorr(c,x,autoFlag,varargin{:});


lags = -maxlag:maxlag;


% Keep only the lags we want and move negative lags before positive lags
if maxlag >= M,
    c = [zeros(maxlag-M+1,N^2);c(end-M+2:end,:);c(1:M,:);zeros(maxlag-M+1,N^2)];
else
    c = [c(end-maxlag+1:end,:);c(1:maxlag+1,:)];
end

% Scale as specified
```

```
% Scale as specified
[c,msg] = scaleXcorr(c,xIsMatrix,scaleType,autoFlag,M,maxlag,lags,x,varargin{:});
if ~isempty(msg), error(generatemsgid('SigErr'),msg); end

% If first vector is a row, return a row
c = shiftdim(c,-nshift);

%-----------------------------------------------------------------
function [c,M,N] = matrixCorr(x)
% Compute all possible auto- and cross-correlations for a matrix input

[M,N] = size(x);

X = fft(x,2^nextpow2(2*M-1));

Xc = conj(X);

[MX,NX] = size(X);
C = zeros(MX,NX*NX);
for n =1:N,
    C(:,(((n-1)*N)+1):(n*N)) = repmat(X(:,n),1,N).*Xc;
end

c = ifft(C);

%-----------------------------------------------------------------
function [c,M,N] = vectorXcorr(x,autoFlag,varargin)
% Compute auto- or cross-correlation for vector inputs

x = x(:);

[M,N] = size(x);

if autoFlag,
    % Autocorrelation
```

Good programming practise

- In this course:
  - 10% of lab and assignment marks allocated to good code formatting and programming practise
    - Plenty of examples exist, see http://www.datatool.com/downloads/matlab_style_guidelines.pdf
  - 5% of lab and assignment marks allocated to correct use of plotting
    - Correctly labelled axes
    - Axis ranges selected to show interesting part of plot
    - Good use of MATLAB plotting functions to visualise simulation results
      - e.g. Use of `hold` for comparing two curves
    - Only the key essential information shown (not plot after plot of similar results)

Object-Oriented Programming

```
classdef date
% write a description of the class here.

    properties
    % define the properties of the class here, (like fields of a struct)
        minute = 0;
        hour;
        day;
        month;
        year;
    end

    methods
    % methods, including the constructor are defined in this block

        function obj = date(minute,hour,day,month,year)
        % class constructor
            if(nargin > 0)
              obj.minute = minute;
              obj.hour   = hour;
              obj.day    = day;
              obj.month  = month;
              obj.year   = year;
            end
        end

        function obj = rollDay(obj,numdays)

            obj.day = obj.day + numdays;
        end

    end
end
```

see e.g. http://www.cs.ubc.ca/~mdunham/tutorial/objectOriented.html