Introduction to MatLab

- MATLAB is very well suited to model communications systems:
  - Signals are naturally represented in MATLAB,
  - MATLAB has a very large library of functions for processing signals,
  - Visualization of signals is very well supported in MATLAB.
- MATLAB is used interactively.
  - Eliminates code, compile, run cycle.
  - Great for rapid prototyping and what-if analysis.

MATLAB's Built-in Help System

- MATLAB has an extensive built-in help system.
  - On-line documentation reader:
  - contains detailed documentation for entire MATLAB system,
  - is invoked by
    - typing `doc` at command line
    - clicking "Question Mark" in tool bar of main window,
    - via "Help" menu.
  - Command-line help provides access to documentation inside command window.
  - *Help*ful commands include:
    - **`help function`**`-name`, e.g., **`help fft`**.
    - **`lookfor`** `keyword`, e.g., **`lookfor`** `inverse`.
- We will learn how to tie into the built-in help system.

Interacting with MATLAB

- You interact with MATLAB by typing commands at the command prompt (») in the command window.
- MATLAB's response depends on whether a semicolon is appended after the command or not.
  - If a semicolon is **not** appended, then MATLAB displays the result of the command.
  - With a semicolon, the result is not displayed.
- **Examples:**
  - The command `xx = 1:3` produces

```
xx =
     1     2     3
```

  - The command `xx = 1:3;` produces no output. The variable `xx` still stores the result.
  - Do use a semicolon with `xx = 1:30000000;`

Signals and Vectors

- Our objective is to simulate communication systems in MATLAB.
  - This includes the signals that occur in such systems, and
  - processing applied to these signals.
- In MATLAB (and any other digital system) signals must be represented by samples.
  - Well-founded theory exists regarding sampling (Nyquist's sampling theorem).
  - Result: Signals are represented as a sequence of numbers.
- MATLAB is ideally suited to process sequences of numbers.
  - MATLAB's basic data types: vectors (and matrices).
  - Vectors are just sequence of numbers.

- **Task:** Generate samples of the sinusoidal signal

$$x(t) = 3 \cdot \cos(2\pi 440t - \frac{\pi}{4})$$

for t ranging from 0 to 10 ms. The sampling rate is 20 KHz.

```
fs=20*10^3;
t=0:1/fs:0.01;
theta=-pi/4;
x=3*cos(2*pi*440*t+theta);
```

Addition and Subtraction

- The standard + and − operators are used to add and subtract vectors.
- One of two conditions must hold for this operation to succeed.
  - Both vectors must have exactly the same size.
    - In this case, corresponding elements in the two vectors are added and the result is another vector of the same size.
    - **Example:** `[1 3 2] + 1:3` produces `2 5 5`.
    - A prominent error message indicates when this condition is violated.
  - One of the operands is a scalar, i.e., a 1×1 (degenerate) vector.
    - In this case, each element of the vector has the scalar added to it.
    - The result is a vector of the same size as the vector operand.
    - **Example:** `[1 3 2] + 2` produces `3 5 4`.

Element-wise Multiplication and Division

- The operators `.*` and `./` operators multiply or divide two vectors element by element.
- One of two conditions must hold for this operation to succeed.
  - Both vectors must have exactly the same size.
    - In this case, corresponding elements in the two vectors are multiplied and the result is another vector of the same size.
    - **Example:** `[1 3 2] .* 1:3` produces `1 6 6`.
    - An error message indicates when this condition is violated.
  - One of the operands is a scalar.
    - In this case, each element of the vector is multiplied by the scalar.
    - The result is a vector of the same size as the vector operand.
    - **Example:** `[1 3 2] .* 2` produces `2 6 4`.
    - If one operand is a scalar the '.' may be omitted, i.e., `[1 3 2] * 2` also produces `2 6 4`.

Inner Product

- The operator $*$ with two vector arguments computes the inner product (dot product) of the vectors.
  - Recall the inner product of two vectors is defined as

$$\vec{x}' \cdot \vec{y} = \sum_{n=1}^{N} x(n) \cdot y(n).$$

  - This implies that the result of the operation is a scalar!
- The inner product is a useful and important signal processing operation.
  - It is very different from element-wise multiplication.
  - The second dimension of the first operand must equal the first dimension of the second operand.
    - MATLAB error message: `Inner matrix dimensions must agree.`
  - **Example:** `[1 3 2] * (1:3)' = 13`.
    - The single quote (`'`) transposes a vector.

Powers

- To raise a vector to some power use the `.^` operator.
  - **Example:** `[1 3 2].^2` yields `1 9 4`.
  - The operator `^` exists but is generally not what you need.
    - **Example:** `[1 3 2]^2` is equivalent to `[1 3 2] * [1 3 2]` which produces an error.
- Similarly, to use a vector as the exponent for a scalar base use the `.^` operator.
  - **Example:** `2.^[1 3 2]` yields `2 8 4`.
- Finally, to raise a vector of bases to a vector of exponents use the `.^` operator.
  - **Example:** `[1 3 2].^(1:3)` yields `1 9 8`.
  - The two vectors must have the same dimensions.
- The `.^` operator is (nearly) always the right operator.

Complex Arithmetic

- MATLAB support complex numbers fully and naturally.
  - The imaginary unit $i = \sqrt{-1}$ is a built-in constant named `i` and `j`.
  - Creating complex vectors:
    - **Example:** `xx = randn(1,5) + j*randn(1,5)` creates a vector of complex Gaussian random numbers.
- A couple of "gotchas" in connection with complex arithmetic:
  - Never use `i` and `j` as variables!
    - **Example:** After invoking `j=2`, the above command will produce very unexpected results.
  - Transposition operator (`'`) transposes and forms conjugate complex.
    - That is very often the right thing to do.
    - Transpose only is performed with `.'` operator.

Functions Returning a Scalar Result

- Many other functions accept a vector as its input and return a scalar value as the result.
- Examples include
  - **`min`** and **`max`**,
  - **`mean`** and `var` or **`std`**,
  - **`sum`** computes the sum of the elements of a vector,
  - **`norm`** provides the square root of the sum of the squares of the elements of a vector.
    - The norm of a vector is related to power and energy.

Accessing Elements of a Vector

- Accessing a range of elements of a vector:
  - **Example:** Let `xx = ones(1,10);`, change the first five elements to -1.
  - **Solution:** `xx(1:5) = -1*ones(1,5);` Note, `xx(1:5) = -1` works as well.
  - **Example:** Change every other element of `xx` to 2.
  - **Solution:** `xx(2:2:end) = 2;`;
    - Note that **`end`** may be use to denote the index of a vector's last element.
    - This is handy if the length of the vector is not known.
  - **Example:** Change third and seventh element to 3.
  - **Solution:** `xx([3 7]) = 3;`;
- A set of elements of a vector is accessed by providing a vector of indices in parentheses.

A Basic Plot

- The sinusoidal signal, we generated earlier is easily plotted via the following sequence of commands:
- Try **`help plot`** for more information about the capabilities of the **`plot`** command.

```
%% Plot
plot(tt, xx, 'r')          % xy-plot, specify red line
xlabel( 'Time (s)' )       % labels for x and y axis
ylabel( 'Amplitude' )
title( 'x(t) = A cos(2\pi f t + \phi)')
grid                       % create a grid
axis([0 10e-3 -4 4])       % tweak the range for the axes
```

Multiple Plots in One Figure

- MATLAB can either put multiple graphs in the same plot or put multiple plots side by side.
- The latter is accomplished with the **`subplot`** command.

```
subplot(2,1,1)
plot(tt, xx )              % xy-plot
xlabel( 'Time (s)' )       % labels for x and y axis
ylabel( 'Amplitude' )
title( 'x(t) = A cos(2\pi f t + \phi)')

subplot(2,1,2)
plot(tt, yy )              % xy-plot
xlabel( 'Time (s)' )       % labels for x and y axis
ylabel( 'Amplitude' )
title( 'x(t) = A sin(2\pi f t + \phi)')
```

Resulting Plot