Chapter 2. Development Environment and Basic Operations
• 2.1 Introduction

• 2.2 MATLAB Development Environment
  – Prior to R2012b
  – After R2012b

• 2.3 Help System

• 2.4 Basic Operations and Order of Precedence

• 2.5 Basic Functions and Utilities

• 2.6 Managing Work Session

• 2.7 For Advanced Users
  – Changing Preferences
  – Analyzing Code in Current Folder
  – Profiling

• 2.8 Problem-Solving Methodology
Starting MATLAB

MATLAB 7.0.0

MATLAB R2011a

MATLAB R2014b

MATLAB R2019a

All rights reserved. No part of this publication may be reproduced, distributed, or transmitted, unless for course participation, in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the Publisher and/or Author. Contact the American Institute of Aeronautics and Astronautics, Professional Development Programs, 12700 Sunrise Valley Drive, Suite 200, Reston, VA 20191-5807.
Desktop Prior to R2012b

- **Current Directory browser** – shows a list of the files in the current directory.
- **Start Button** – can be used to launch toolboxes, MATLAB tools, and so on.
- **Workspace browser** – shows variables defined (created) in workspace.
- **Command Window** – the window you type your commands in and get responses.
- **Command History window** – displays previously issued commands.
- **Simulink**
- **GUIDE**
- **Profiler**
- **Help Browser**

\[
\begin{align*}
&\text{>> commandwindow} \\
&\text{>> commandhistory} \\
&\text{>> filebrowser} \\
&\text{>> workspace}
\end{align*}
\]
Managing MATLAB Desktop

Tabs
- Current folder
- Command window
- Command history
- File browser
- Workspace

Quick Access Toolbar
- >commandwindow
- >commandhistory
- >filebrowser
- >workspace

Search window

Toolstrip

MATLAB Drive

Search window

All rights reserved. No part of this publication may be reproduced, distributed, or transmitted, unless for course participation, in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the Publisher and/or Author. Contact the American Institute of Aeronautics and Astronautics, Professional Development Programs, 12700 Sunrise Valley Drive, Suite 200, Reston, VA 20191-5807.
Managing MATLAB Desktop

Guided by MATLAB's desktop layout, the image highlights key sections:

- **Profiler**
- **Simulink**
- **Help Browser**

The desktop layout is further detailed with options such as:

- **GUIDE**
- **Add-Ons**
- **Help options**

The desktop features sections like:

- **PLOTS**
- **APPS**
- **FILE**

With menu options for:

- **Preferences**
- **Search Documentation**

The entire view is a comprehensive guide to managing the MATLAB desktop environment.
Command History Window

Tap “↑” key

All rights reserved. No part of this publication may be reproduced, distributed, or transmitted, unless for course participation, in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the Publisher and/or Author. Contact the American Institute of Aeronautics and Astronautics, Professional Development Programs, 12700 Sunrise Valley Drive, Suite 200, Reston, VA 20191-5807.
Getting More Apps

[Image of MATLAB Add-On Explorer with various toolboxes and apps listed, such as MathWorks Toolboxes and Products with Apps, Community Toolboxes with Apps, and Community Apps.]
Typing Commands

Plotting options available for a selected variable

User input

Result of calculation

Suppressing the output
Path Setting
Creating Shortcuts

1. Open the `Shortcut Editor` in MATLAB.
2. Click on `Create Shortcut`.
3. Enter `Prepare` in the `Label` field.
4. In the `Callback` field, type `close all clear all clc`.
5. Click `Save` to create the shortcut.

Shortcut created: `Prepare`
Workspace Window

- **Right click**
  - Open Selection
  - Save As
  - Copy
  - Duplicate
  - Delete
  - Rename
  - Edit Value
  - Plot Catalog

- **Double click**
  - Table rows
  - Columns

- **Single click**
  - Select variable

- **Right click**
  - New
  - Save
  - Clear Workspace
  - Refresh
  - Choose Columns
  - Sort By
  - Paste
  - Select All
  - Print...
  - Page Setup

All rights reserved. No part of this publication may be reproduced, distributed, or transmitted, unless for course participation, in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the Publisher and/or Author. Contact the American Institute of Aeronautics and Astronautics, Professional Development Programs, 12700 Sunrise Valley Drive, Suite 200, Reston, VA 20191-5807.
Array Editor

Use Ctrl to select not adjacent cells

All rights reserved. No part of this publication may be reproduced, distributed, or transmitted, unless for course participation, in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the Publisher and/or Author. Contact the American Institute of Aeronautics and Astronautics, Professional Development Programs, 12700 Sunrise Valley Drive, Suite 200, Reston, VA 20191-5807.
Help Browser

MATLAB R2018a - academic use

HOME PLOTS APPS

New Script New Live Script New
New Find Files Import Data Save Workspace
Open Variable Clear Workspace
New Variable Run and Time Favorites
Clear Commands

Analyze Code
Preferences
Layout
Parallel
Add-Ons
Help

Community
Request Support Learn MATLAB

CODE SIMULINK ENVIRONMENT RESOURCES

Current Folder:
C: \Users \yakim \Desktop \MATLAB2Ed \Figure Generators

Current Folder:

Untitled5.m (Script)

Command Window:

> doc

Documentation

My Products
MATLAB
Simulink
Aerospace Blockset
Aerospace Toolbox
Communications System Toolbox
Computer Vision System Toolbox
Control System Toolbox
Data Acquisition Toolbox
DSP System Toolbox
Image Processing Toolbox
Instrument Control Toolbox
Mapping Toolbox
MATLAB Coder
MATLAB Compiler
MATLAB Compiler SDK
MATLAB Distributed Computing Server
Neural Network Toolbox
Optimization Toolbox
Parallel Computing Toolbox
Robotics System Toolbox
Signal Processing Toolbox

My Products
MATLAB

MATLAB
Parallel Computing
MATLAB Distributed Computing Server
Parallel Computing Toolbox
Math, Statistics, and Optimization
Neural Network Toolbox
Optimization Toolbox
Symbolic Math Toolbox
Control Systems
Aerospace Toolbox

MATLAB

Simulink
Control Systems
Aerospace Blockset
Robotics System Toolbox
Simulink Control Design
Signal Processing and Wireless Communications
Communications System Toolbox
Computer Vision System Toolbox
DSP System Toolbox

Hardware Support
For a complete list of hardware solutions, see Hardware Support.

Getting Started with MATLAB
Functions in MATLAB
Release Notes
Installation

Search Documentation

Search Help

Explore Examples
Explore Add-Ons

Aerospace and Electronics, Professional Development Programs, 12700 Sunrise Valley Drive, Suite 200, Reston, VA 20191-5807.
On-line Courses, Tutorials & Examples

>> demo

All rights reserved. No part of this publication may be reproduced, distributed, or transmitted, unless for course participation, in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the Publisher and/or Author. Contact the American Institute of Aeronautics and Astronautics, Professional Development Programs, 12700 Sunrise Valley Drive, Suite 200, Reston, VA 20191-5807.
MATLAB Demos
Examples

`bin2dec` - Convert text representation of binary number to decimal number

This MATLAB function interprets `binarystr`, text that represents a binary number, and returns the equivalent decimal number.

```matlab
bin2dec(binarystr)
```

**Description**

`bin2dec(binarystr)` interprets `binarystr`, text that represents a binary number, and returns the equivalent decimal number. `binarystr` must represent a nonnegative integer value smaller than or equal to the value returned by `flintmax`. `binarystr` can be a character array or a cell array of character vectors. `bin2dec` ignores any space (" ") characters in the input text.

**Examples**

Binary `010111` converts to decimal 23:

```matlab
bin2dec('010111')
ans = 23
```
Push a Tab key to have your command finished.
Getting Quick Help

Right click

All rights reserved. No part of this publication may be reproduced, distributed, or transmitted, unless for course participation, in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the Publisher and/or Author. Contact the American Institute of Aeronautics and Astronautics, Professional Development Programs, 12700 Sunrise Valley Drive, Suite 200, Reston, VA 20191-5807.
Exploring MATLAB Functions

This subdirectory (MATLAB\R2018b\toolbox\matlab) contains M-file sources to many of the functions supplied with MATLAB.

```
>> help ops
>> help elmat
>> help elfun
>> help specfun
>> help randfun
>> help strfun
```
## Arithmetic Operations

### Matrix form

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operation</th>
<th>MATLAB Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>^</td>
<td>exponentiation, $a^b$</td>
<td>$a^b$</td>
</tr>
<tr>
<td>*</td>
<td>multiplication, $ab$</td>
<td>$a*b$</td>
</tr>
<tr>
<td>/</td>
<td>right division, $a/b$</td>
<td>$a/b$</td>
</tr>
<tr>
<td>\</td>
<td>left division, $a\backslash b$</td>
<td>$a\backslash b$</td>
</tr>
<tr>
<td>+</td>
<td>addition, $a+b$</td>
<td>$a+b$</td>
</tr>
<tr>
<td>–</td>
<td>subtraction, $a−b$</td>
<td>$a-b$</td>
</tr>
</tbody>
</table>

### Element wise form

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operation</th>
<th>MATLAB Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>^</td>
<td>exponentiation, $a^b$</td>
<td>$a.^b$</td>
</tr>
<tr>
<td>*</td>
<td>multiplication, $ab$</td>
<td>$a.*b$</td>
</tr>
<tr>
<td>/</td>
<td>right division, $a/b$</td>
<td>$a./b$</td>
</tr>
<tr>
<td>\</td>
<td>left division, $a\backslash b$</td>
<td>$a./b$</td>
</tr>
</tbody>
</table>
## Embedded Functions

### Trigonometric functions

- **sin** - sine
- **asin** - inverse sine
- **cos** - cosine
- **acos** - inverse cosine
- **tan** - tangent
- **atan** - inverse tangent
- **atan2** - 4-quadrant inverse tangent
- **sec** - secant \((1/cos(x))\)
- **asec** - inverse secant
- **csc** - cosecant \((1/sin(x))\)
- **acsc** - inverse cosecant
- **cot** - cotangent \((1/tan(x))\)
- **acot** - inverse cotangent

<table>
<thead>
<tr>
<th>MATLAB Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sin</td>
<td>- sine</td>
</tr>
<tr>
<td>asin</td>
<td>- inverse sine</td>
</tr>
<tr>
<td>cos</td>
<td>- cosine</td>
</tr>
<tr>
<td>acos</td>
<td>- inverse cosine</td>
</tr>
<tr>
<td>tan</td>
<td>- tangent</td>
</tr>
<tr>
<td>atan</td>
<td>- inverse tangent</td>
</tr>
<tr>
<td>atan2</td>
<td>- 4-quadrant inverse tangent</td>
</tr>
<tr>
<td>sec</td>
<td>- secant ((1/cos(x)))</td>
</tr>
<tr>
<td>asec</td>
<td>- inverse secant</td>
</tr>
<tr>
<td>csc</td>
<td>- cosecant ((1/sin(x)))</td>
</tr>
<tr>
<td>acsc</td>
<td>- inverse cosecant</td>
</tr>
<tr>
<td>cot</td>
<td>- cotangent ((1/tan(x)))</td>
</tr>
<tr>
<td>acot</td>
<td>- inverse cotangent</td>
</tr>
</tbody>
</table>

### Rounding and Remainder

- **fix** - round towards zero
- **floor** - round towards minus infinity
- **ceil** - round towards plus infinity
- **round** - round towards nearest integer \(\text{round}(X,N)\) rounds to \(N\) digits
- **mod** - modulus (signed remainder after division)
- **rem** - remainder after division
- **sign** - signum (returns 1 if the element is greater than 0, 0 - equals 0, and -1 - less than 0)

---

All rights reserved. No part of this publication may be reproduced, distributed, or transmitted, unless for course participation, in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the Publisher and/or Author. Contact the American Institute of Aeronautics and Astronautics, Professional Development Programs, 12700 Sunrise Valley Drive, Suite 200, Reston, VA 20191-5807.
Examples of Using Functions

```matlab
>> tand(90)
anst =
   Inf
>> sin([0,0.5*pi,pi])
anst =
   0  1.0000  0.0000
>> floor([4.2 -4.2])
anst =
   4    -5
>> fix([4.3 -4.3])
anst =
   4    -4
>> mod(12,5)
anst =
   2
>> exp(1)
anst =
   2.7183
>> abs(3+4i)
anst =
   5
>> angle(2+2j)*180/pi
ananst =
   45
>> bin2dec('101')
anst =
   5
>> num2str(pi,3)
anst =
   3.14
>> lower('MATLAB')
anst =
   matlab
>> km2deg(111.2)
anst =
   1.0000
>> clock
ananst =
   1.0e+03 *
   2.0140 0.0110 0.0080 0.0230 0.0380 0.0536
>> datenum(2014,11,15,10,15,40)
anst =
   7.3592e+05
>> date
ananst =
   26-Apr-2015
```
Variable Names

The valid variable name may contain 1–63 characters (defined by the `namelengthmax` function of MATLAB). It must start with a letter, but may use letters, digits, and underscore after the first letter (remember, MATLAB is case-sensitive). The only expectation is that the variable name cannot be one of 20 keywords (type in `iskeyword` to see what these keywords are).

<table>
<thead>
<tr>
<th>Examples of valid names:</th>
<th>Invalid names:</th>
</tr>
</thead>
<tbody>
<tr>
<td>x6</td>
<td>6x</td>
</tr>
<tr>
<td>lastValue</td>
<td>end</td>
</tr>
<tr>
<td>n_factorial</td>
<td>n!</td>
</tr>
</tbody>
</table>

Try to give your variables the meaningful names, so that anyone who looks at your code understands what it does.

You may check existence of variable, function, folder, or class by typing in

```matlab
>> exist name
```
Embedded Functions

Exponential functions

- exp - exponential
- log - natural logarithm
- log10 - common (base-10) logarithm
- log2 - base-2 logarithm and dissect floating point number
- pow2 - base-2 power and scale floating point number
- realpow - power that will error out on complex result
- reallog - natural logarithm of real number
- realsqrt - square root of number greater than or equal to zero
- sqrt - square root
- hypot - square root of sum of squares
- nextpow2 - next higher power of 2

Complex functions

- abs - absolute value
- angle - phase angle
- complex - construct complex data from real and imaginary parts
- conj - complex conjugate
- imag - complex imaginary part
- real - complex real part
- unwrap - unwrap phase angle
- isreal - true for real array
- cplxpair - sort numbers into complex conjugate pairs
### Embedded Functions

#### Conversion functions
- **bin2dec** - converts binary string to decimal integer
- **dec2bin** - converts decimal integer to a binary string
- **dec2hex** - converts decimal integer to IEEE hexadecimal string
- **hex2dec** - converts hexadecimal string to decimal integer
- **num2hex** - converts singles and doubles to IEEE hexadecimal strings
- **hex2num** - converts hexadecimal number string to double-precision number
- **base2dec** - converts base-B string to decimal integer
- **dec2base** - converts decimal integer to base-B string

#### Different useful functions
- **factorial** - factorial function
- **rand** - generates uniformly distributed random numbers
- **randn** - generates normally distributed random numbers
- **randi** - generates uniformly distributed pseudorandom integers
- **lower** - convert string to lowercase
- **upper** - convert string to uppercase
- **int2str** - convert integer to string
- **num2str** - convert number to string
- **str2num** - convert string to number
Mapping Toolbox Functions

**Angle conversions**
- `angl2str` - formats angle strings
- `angledim` - converts angles units or encodings
- `deg2dm`, `deg2dms`, `deg2rad` - converts degrees to deg:min, deg:min:sec and to radians
- `dms2deg`, `dms2dm`, `dms2rad`, `dms2mat` - converts deg:min:sec to deg, deg:min, rad, [deg min sec] matrix
- `mat2dms` - converts [deg min sec] matrix to deg:min:sec
- `rad2deg`, `rad2dm`, `rad2dms` - converts radians to degrees, deg:min and deg:min:sec
- `str2angle` - converts strings to angles in degrees

**Distance conversions**
- `deg2km`, `deg2nm`, `deg2sm` - converts degrees to kilometers, nautical miles, statute miles
- `dist2str` - formats distance strings
- `distdim` - converts distance units
- `km2deg`, `km2nm`, `km2rad`, `km2sm` - converts kilometers to degrees, nautical miles, radians, and statute miles
- `nm2deg`, `nm2km`, `nm2rad`, `nm2sm` - converts nautical miles to degrees, kilometers, radians and statute miles
- `rad2km`, `rad2nm`, `rad2sm` - converts radians to kilometers, nautical miles and statute miles
- `sm2deg`, `sm2km`, `sm2nm`, `sm2rad` - converts statute miles to degrees, kilometers, nautical miles and radians

**Time conversions**
- `hms2hm`, `hms2hr`, `hms2sec`, `hms2mat` - converts hrs:min:sec to hrs:min, hours, sec, [hrs min sec] matrix
- `hr2hm`, `hr2hms`, `hr2sec` - converts hours to hrs:min, hrs:min:sec and seconds
- `mat2hms` - converts [hrs min sec] matrix to hrs:min:sec
- `ec2hm`, `ec2hms`, `ec2hr` - converts time from seconds to hrs:min, hrs:min:sec and hours
- `time2str` - formats time strings
- `timedim` - converts time units or encodings

**Coordinate conversions**
- `cart2pol`, `cart2sph` - transforms Cartesian coordinates to polar/cylindrical or spherical
- `pol2cart`, `sph2cart` - transforms polar/cylindrical or spherical coordinates to Cartesian
Useful Constants

**pi**  - the numerical equivalent to \(\pi = 3.14159265...\), the ratio of a circle’s circumference to its diameter

**i** and **j**  - imaginary unit, \(\sqrt{-1}\) (for complex numbers you may use an asterisk as in \(-2+3*\text{i}\) or have it implied as in \(-2+3\text{i}\))

**eps**  - floating-point relative precision of your computer, \(2^{-52}\)

**realmin**  - smallest floating-point number, \(2^{-1022}\)

**realmax**  - largest floating-point number, \((2-\varepsilon)2^{1023}\)

**inf**  - infinity, \(\infty\), results from operations like division by zero and overflow (i.e., exceed realmax), which lead to results too large to represent as conventional floating-point values

**NaN**  - undefined numerical result, Not-a-Number, results from expressions like \(0/0\), \(\text{inf}/\text{inf}\) or \(\text{Inf}/\text{Inf}\), as well as from arithmetic operations involving a **NaN** (also, if \(n\) is complex with a zero real part, then \(n/0\) returns a value with a **NaN** real part)

**NaT**  - is the representation for Not-a-Time, a value that can be stored in a **datetime** array to indicate an unknown or missing datetime value
Open Source Paradigm

>> open(which('fzero.m'))
Managing Work Session

command syntax

format
- default format with 4 decimal digits, same as short

format short
- short, fixed-decimal format with 4 digits after the decimal point (ADP)

format long
- long, fixed-decimal format with 15 and 7 digits ADP for double and single values, resp.

format shortE
- short scientific floating-point format with 4 digits ADP

format longE
- long scientific floating-point format with 15 and 7 digits ADP for double and single values, respectively

format shortEng
- the same as shortE, but with three digits dedicated to store exponent

format longEng
- the same as longE, but with three digits dedicated to store exponent at the expense of having only 14 and 6 digits ADP for double and single values, resp.

format shortG
- short or shortE, whichever is more compact

format longG
- long or longE, whichever is more compact

format hex
- hexadecimal format

format +
- the symbols +, -, and blank are printed for positive, negative, and zero elements, while imaginary parts are ignored

format bank
- fixed format for dollars and cents (2 decimal digits)

format rat
- approximation by ratio of small integers

format compact
- suppresses excess line feeds to show more output in a simple screen

format loose
- adds extra line feeds to make output more readable
## Managing Work Session

### Command Description

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>home</code></td>
<td>moves the cursor to the upper left corner of the Command window</td>
</tr>
<tr>
<td><code>clc</code></td>
<td>clears the Command window and homes the cursor</td>
</tr>
<tr>
<td><code>exist('name')</code></td>
<td>determines if a file or variable exists having the name 'name'</td>
</tr>
<tr>
<td><code>who</code></td>
<td>lists the variables currently in memory</td>
</tr>
<tr>
<td><code>whos</code></td>
<td>lists the current variables and sizes, and indicates if they have imaginary parts</td>
</tr>
<tr>
<td><code>clear</code></td>
<td>removes all variables from memory (workspace)</td>
</tr>
<tr>
<td><code>clear var1 var2</code></td>
<td>removes the variables <code>var1</code> and <code>var2</code> from memory</td>
</tr>
<tr>
<td><code>clearvars -except a b</code></td>
<td>clears all variables except for those specified following the -except flag</td>
</tr>
<tr>
<td><code>close all</code></td>
<td>closes all open figure windows</td>
</tr>
<tr>
<td><code>close('name')</code></td>
<td>closes the named figure window</td>
</tr>
<tr>
<td><code>quit</code></td>
<td>stops MATLAB</td>
</tr>
</tbody>
</table>

### Command Syntax

- `(colon)` generates an array having regularly spaced elements
- `, (comma)` separates elements of an array
- `; (semicolon)` suppresses screen printing also denotes a new row in an array
- `... (ellipsis)` continues a line

Also, see `pwd` (identifies current folder), `dir` or `ls` (lists folder contents), `what` (lists MATLAB files in folder), `cd` (changes current folder), `which` (locates functions and files), `fileparts` (gets parts of file name), and `memory` (displays memory information).
Preferences

>> preferences

All rights reserved. No part of this publication may be reproduced, distributed, or transmitted, unless for course participation, in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the Publisher and/or Author. Contact the American Institute of Aeronautics and Astronautics, Professional Development Programs, 12700 Sunrise Valley Drive, Suite 200, Reston, VA 20191-5807.
MATLAB saves statements that run in the Command Window to the history file, `History.xml`, located in the directory \AppData\Roaming\MathWorks\MATLAB\R2018b (type `prefdir` in the Command Window to see what the directory containing preferences, history, and layout files is). The `History.m` file is loaded when MATLAB starts. It stores a maximum of 25,000 commands (the default value), deleting the oldest entries as needed to maintain that size.
Toolbars / Colors Preferences

Preferences

MATLAB Tools Preferences

Toolbox:
- Quick Access

Layout:
- Default
- Custom
- Horizontal
- Vertical

Preferences

MATLAB Colors Preferences

Desktop tool colors
- Black
- Background
- Text
- Comments
- Strings
- Unrecognized strings
- System commands
- Syntax errors
- Validation sections
- MATLAB Command Window colors
- Error text
- Warning text
- Hyperlinks

Syntax highlighting sample

```
% create a file for output
fopen('testFile.txt', 'w');
for i=1:10
    fprintf(fid, ['%6.2f ', 'Re(%)']);
end
```

Restore Default Colors

OK  Cancel  Apply  Help
Variable Editor / Figure Copy Preferences
Current Folder Window
Annotating a file with the comments including text TODO, FIXME, or a string of your choice makes it easier to find areas of your code that you intend to improve, complete, or update later.

This report displays potential errors and problems, as well as opportunities for improvement in your code through messages.

Checks the help component of your MATLAB scripts.
Current Folder Reports

Allows creating the Contents.m file providing a summary of the programs in a particular folder. (The help, doc, and ver functions refer to Contents.m files to display information about folders.)

Shows dependencies among MATLAB code files in a current folder.

Determine how much of a file ran when you profiled it (when you run the Profiler on a file, some code might not run, e.g. a block containing an if statement).

>> help codetools
Profiling is a way to measure where a program spends time. Once you identify which functions are consuming the most time, you can determine why you are calling them. Then, you could look for ways to minimize their use and thus improve performance by:

- Avoiding unnecessary computation, which can arise from oversight
- Changing your algorithm to avoid costly functions
- Avoiding recomputation by storing results for future use

When profiling spends most of its time on calls to a few built-in functions, you have probably optimized the code as much as you can.
Problem Solving Methodologies

1) Think about the main goal you want to achieve.
2) Define required inputs and desired outputs.
3) Collect all necessary information.
4) Start with a simplified version of the problem and clearly state your assumptions.
5) Draw a sketch of the problem and think of names for your variables.
6) Check the dimensions and units of your variables.
7) Think about how you are going to proceed toward a solution and write down the general steps, maybe even labeling them (which will further be converted to the comments line in your code).
8) Make sure your program is robust enough to handle a variety of the expected and unexpected (wrong or missing) inputs as well as interface properly with any applications you use.
9) Start writing code, checking its pieces if appropriate by displaying intermediate results as needed.
10) Devote some time to develop a concise, yet informative output structure, which later on will include producing informative graphics.
11) Check the outputs of your program on some specific cases with known (expected) results.
12) Perform a “reality check” of the outputs your program produces in a general case.
13) If your algorithm involves some iterative computations (such as, optimization), try to limit the number of cycles first to be sure your program works properly.
14) Do not start batch computing unless you are absolutely sure your program runs properly for a single dataset (having lots of pretty output does not necessarily mean it is not garbage).
15) After debugging your program, make sure to suppress displacing all unnecessary intermediate results, so that your program produces only those outputs you actually need.

All rights reserved. No part of this publication may be reproduced, distributed, or transmitted, unless for course participation, in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the Publisher and/or Author. Contact the American Institute of Aeronautics and Astronautics, Professional Development Programs, 12700 Sunrise Valley Drive, Suite 200, Reston, VA 20191-5807.
The End of Chapter 2

Questions?