MATLAB Tutorial

Part 1: Basics

Applied Mathematics , Fall 2012

Marcus Pendergrass
Hampden-Sydney College

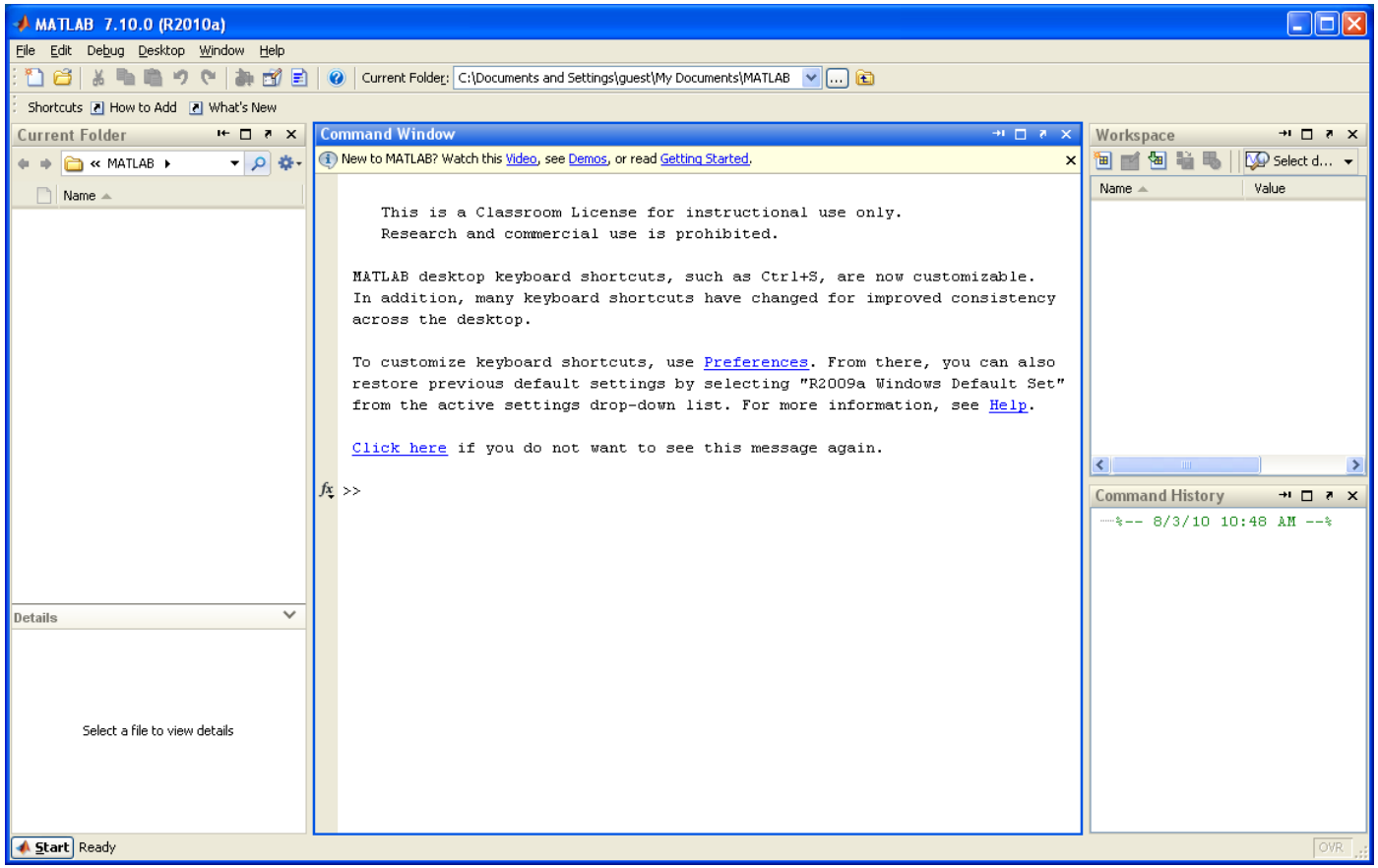
Outline

Starting MATLAB

- To start MATLAB, double-click the "MATLAB R2012a" icon:

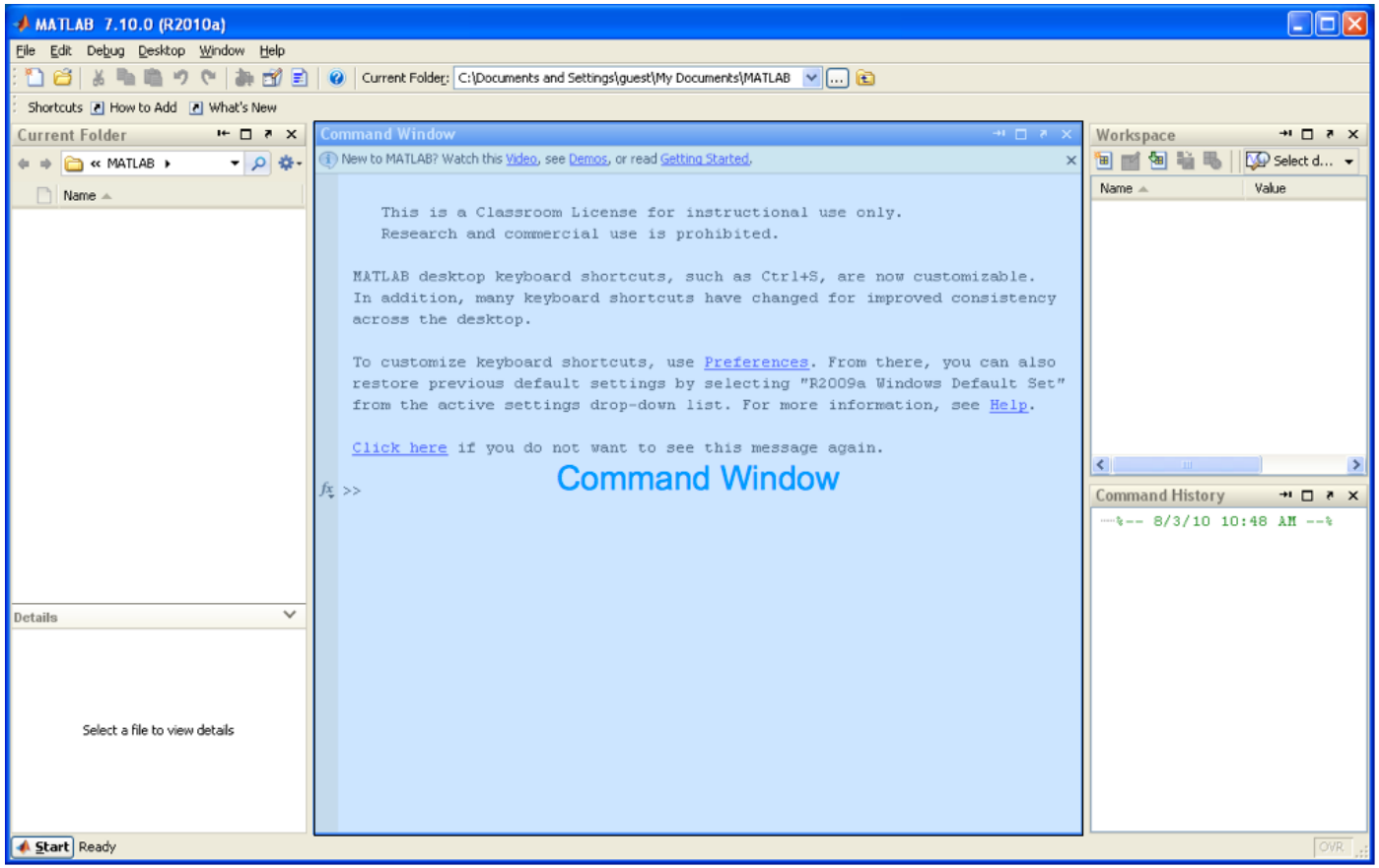
- This brings up the MATLAB desktop...

The MATLAB Desktop



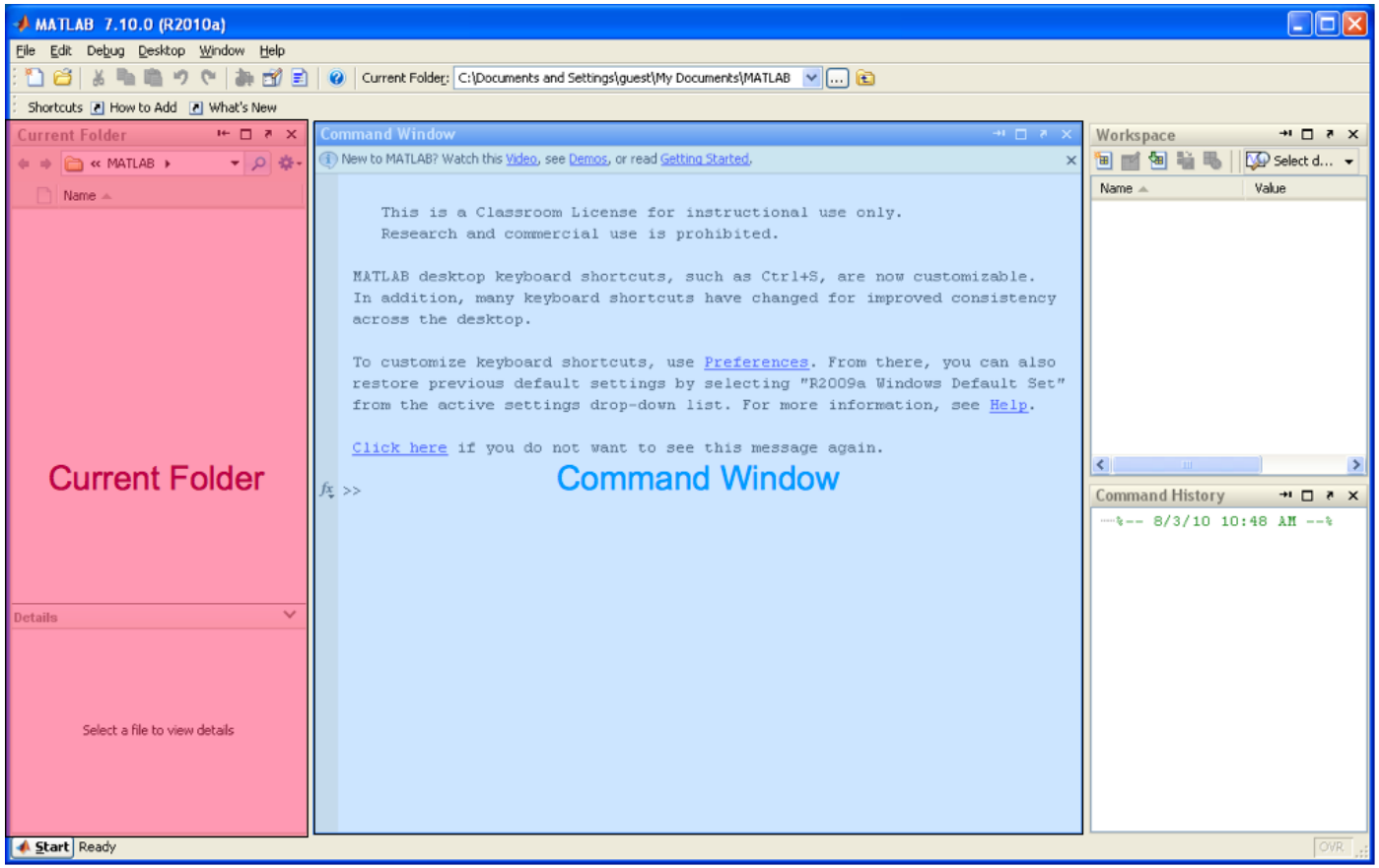
- The MATLAB desktop has several useful components...

The MATLAB Desktop – Command Window



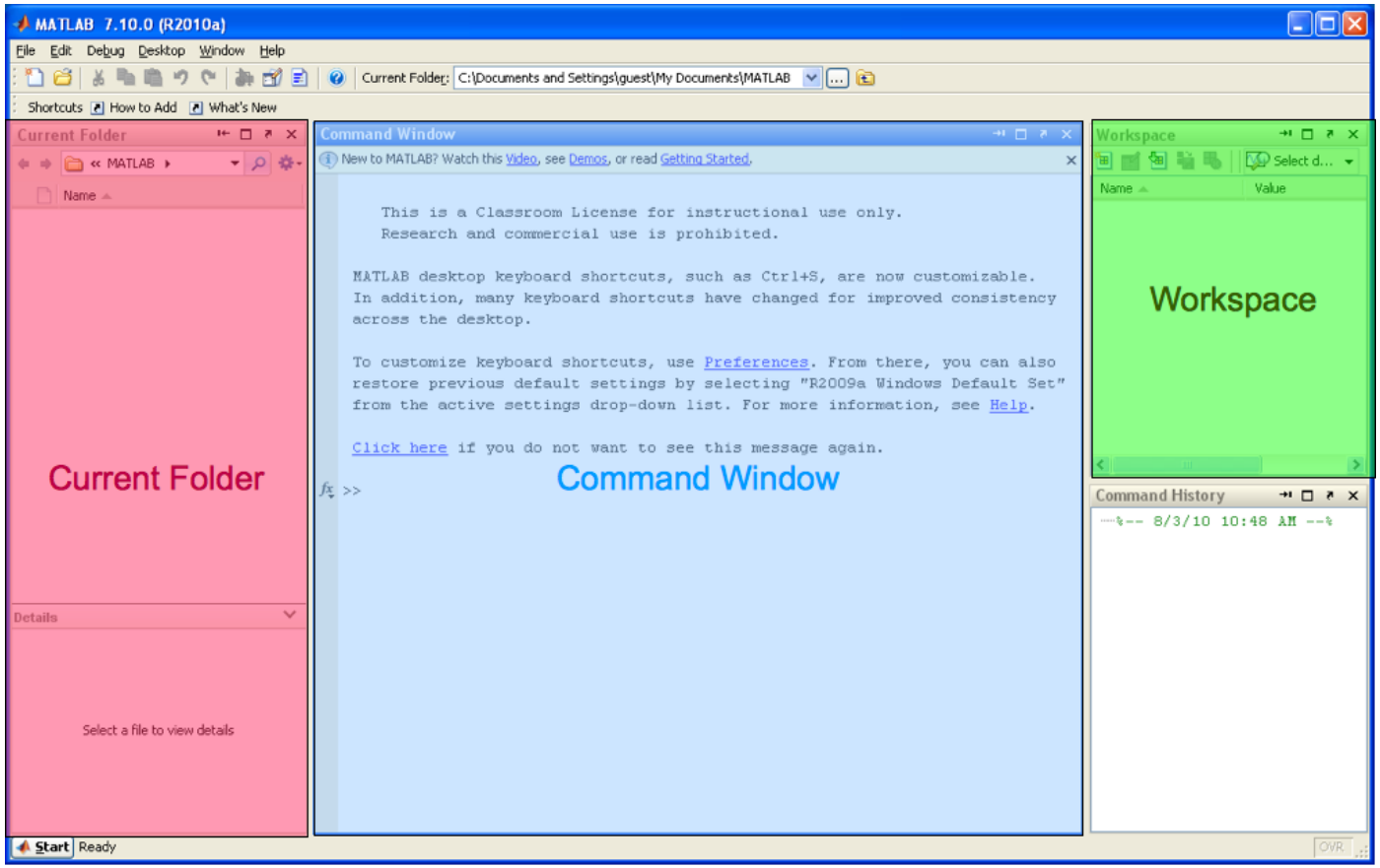
- You work with MATLAB interactively in the Command Window

The MATLAB Desktop – Current Folder



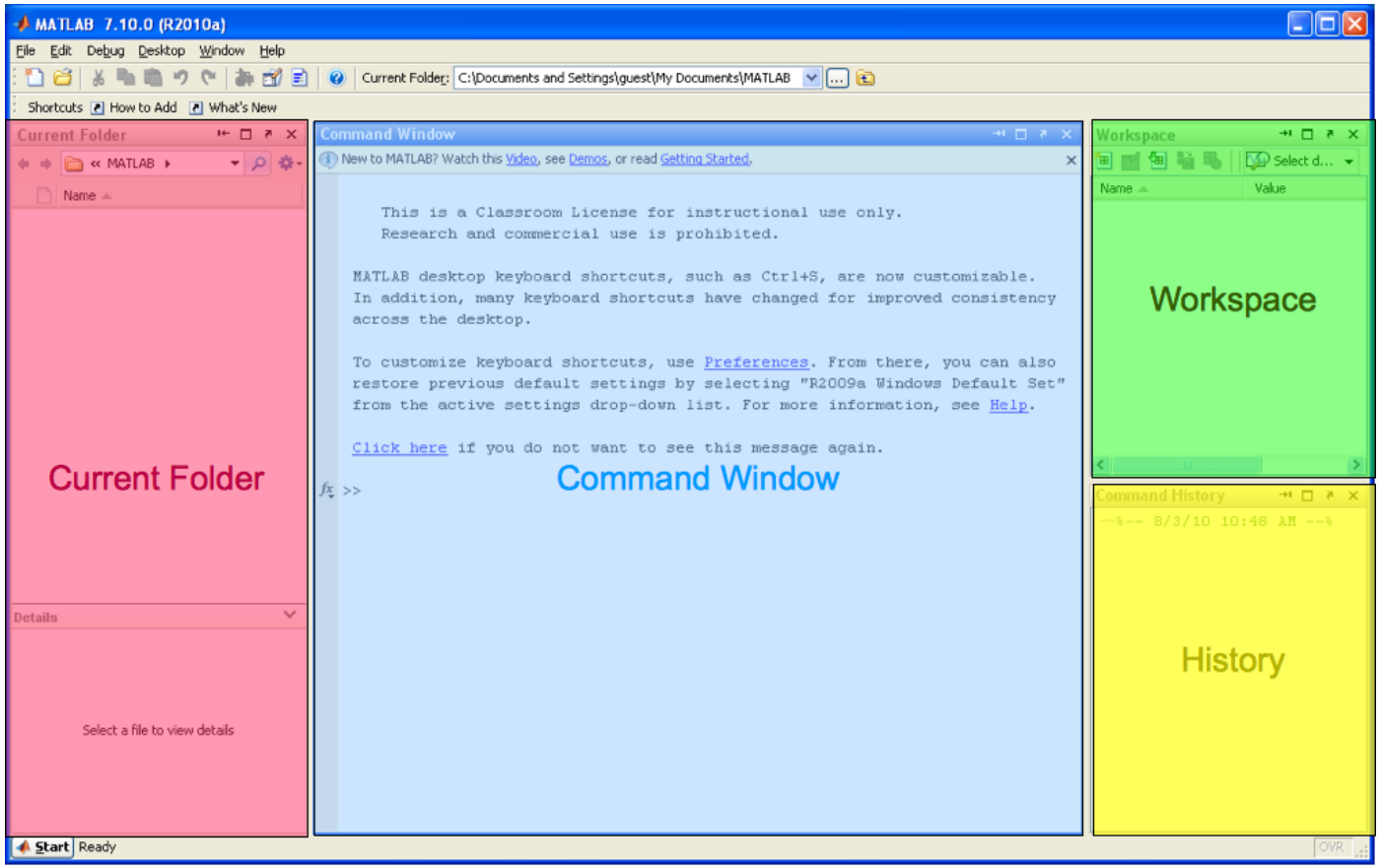
- The Current Folder window shows the current directory

The MATLAB Desktop – Workspace



- The Workspace window shows the variables you've defined

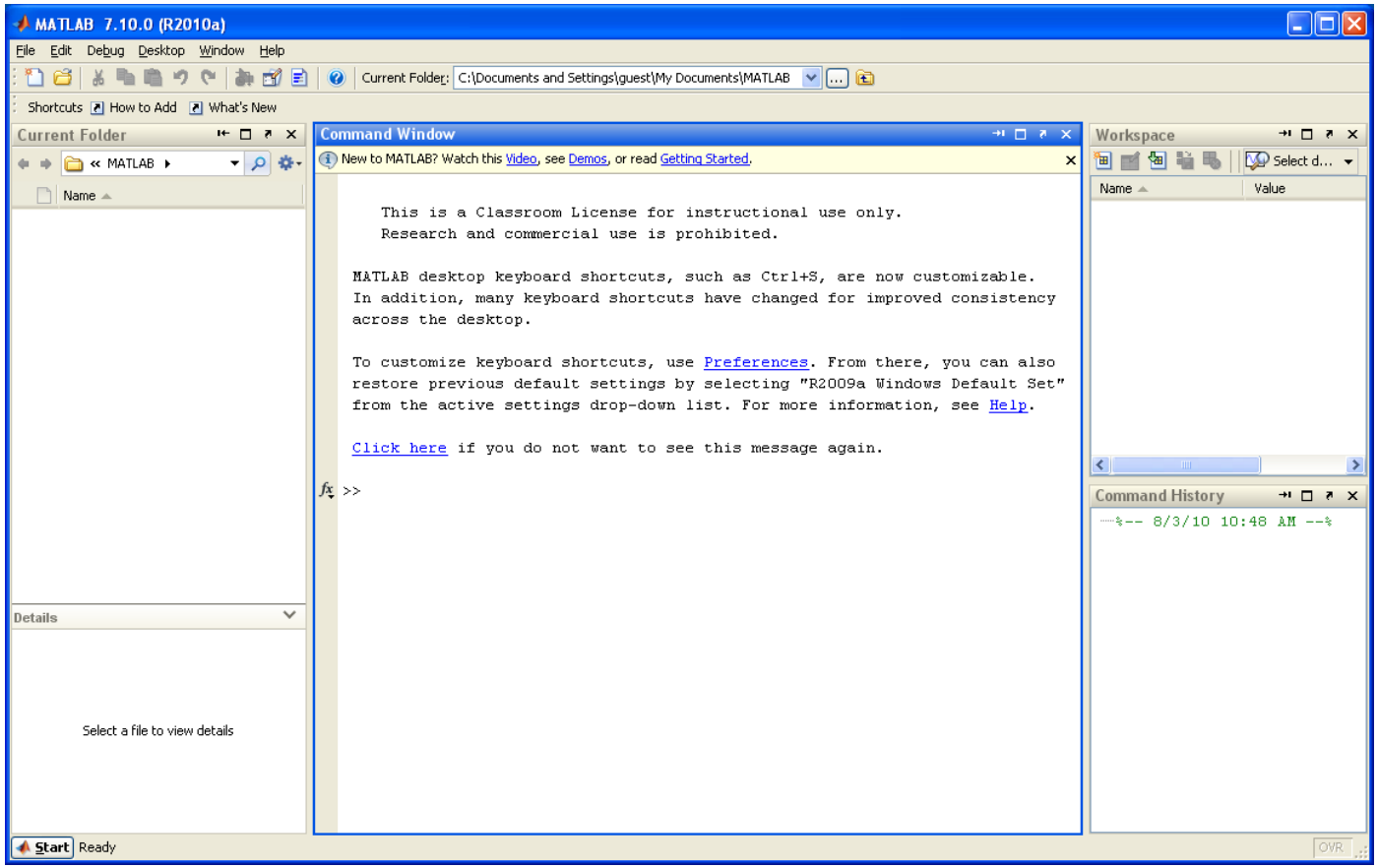
The MATLAB Desktop – History



- The History window shows the commands you've entered so far

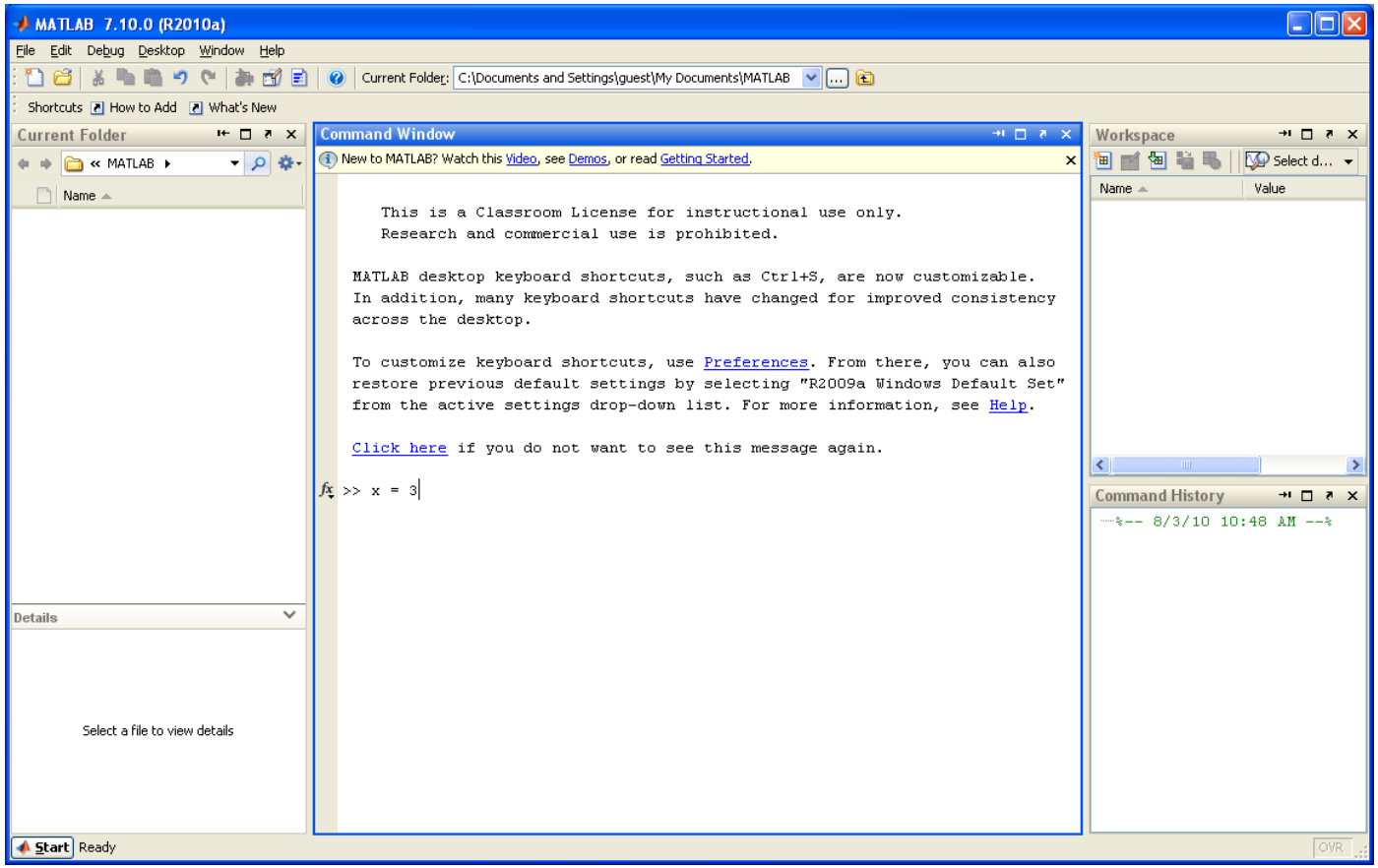
Outline

Define A Variable



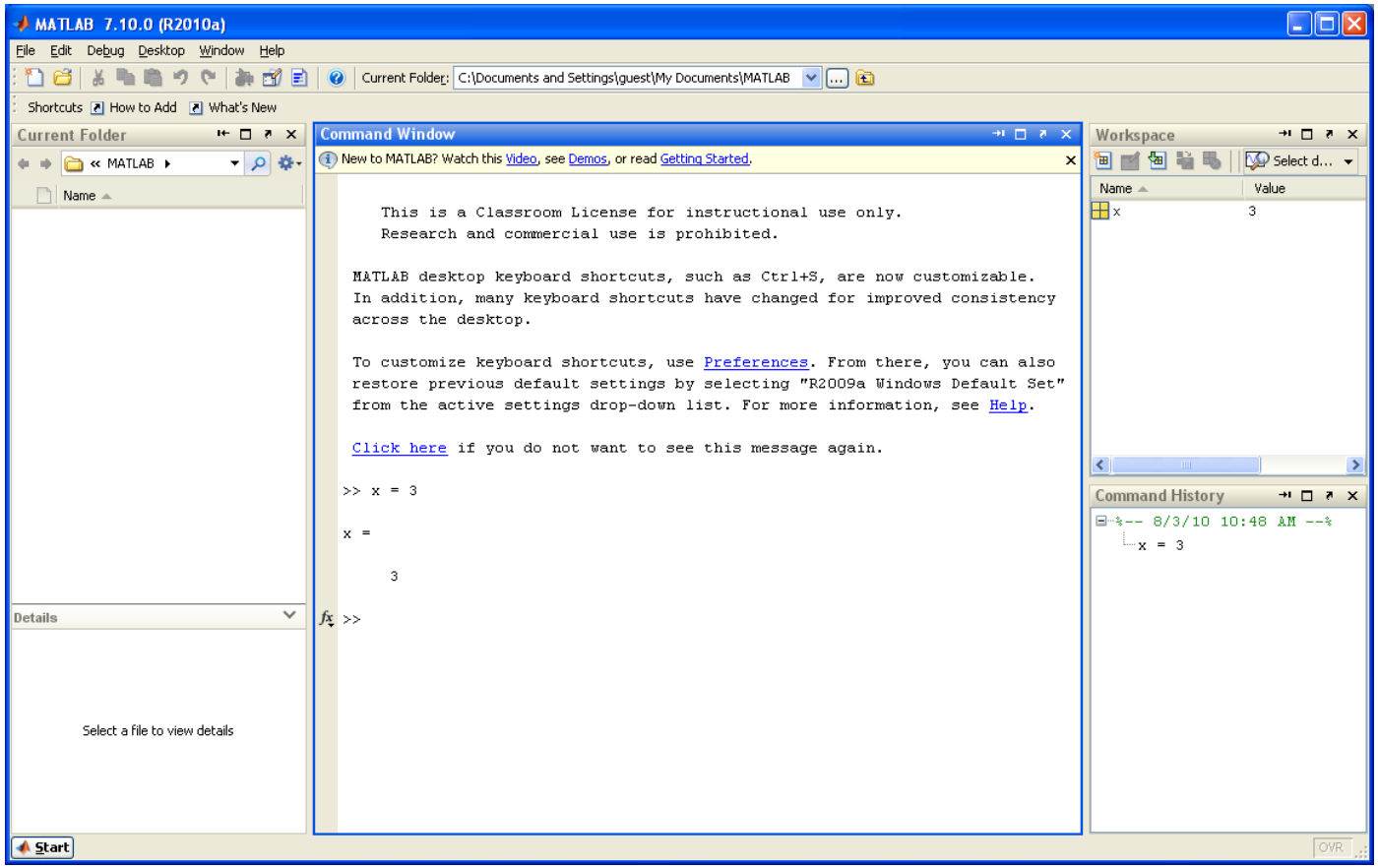
- Let's set a variable `x` equal to 3...

Define A Variable



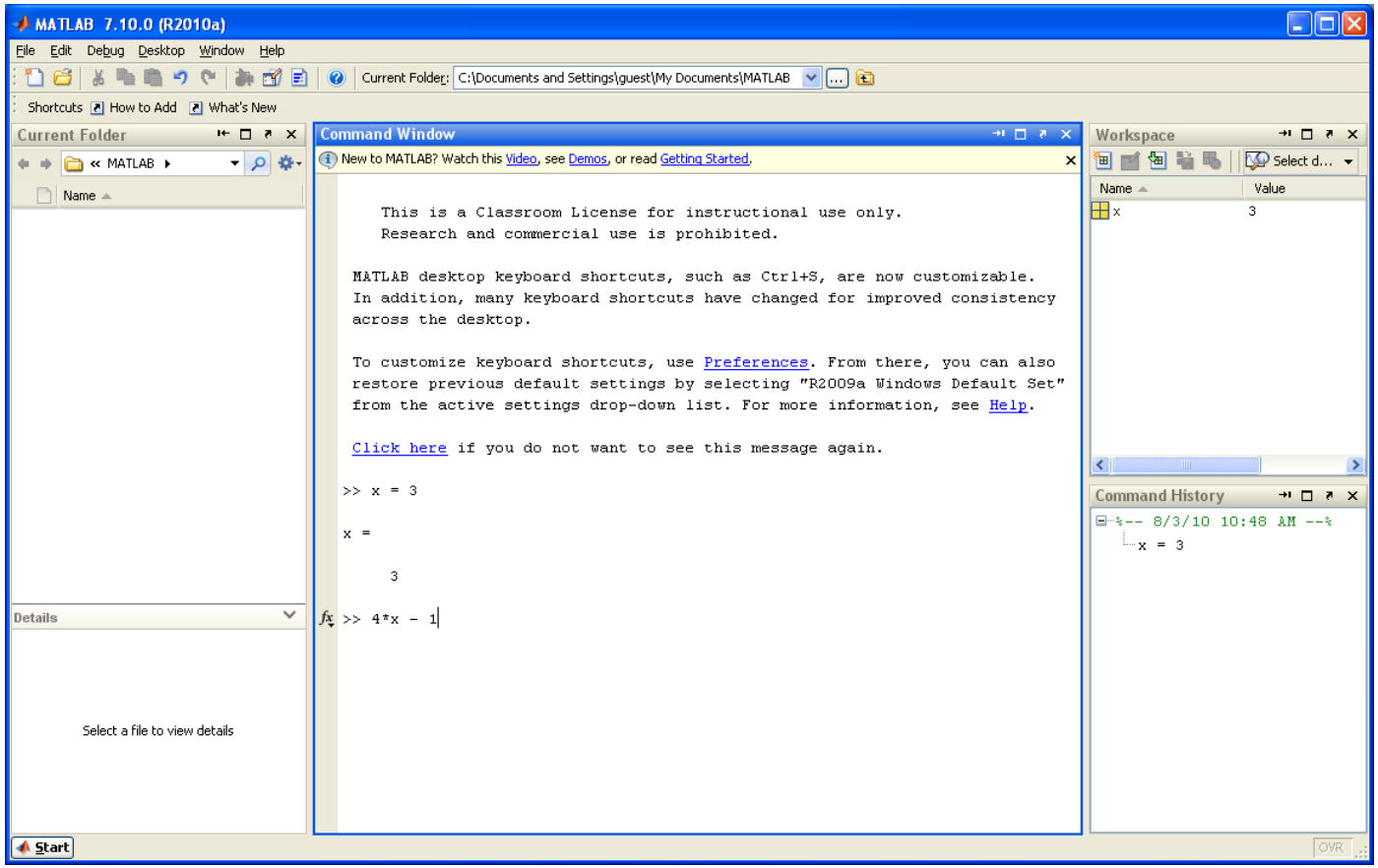
- Pressing return tells MATLAB to execute your command.

Define A Variable



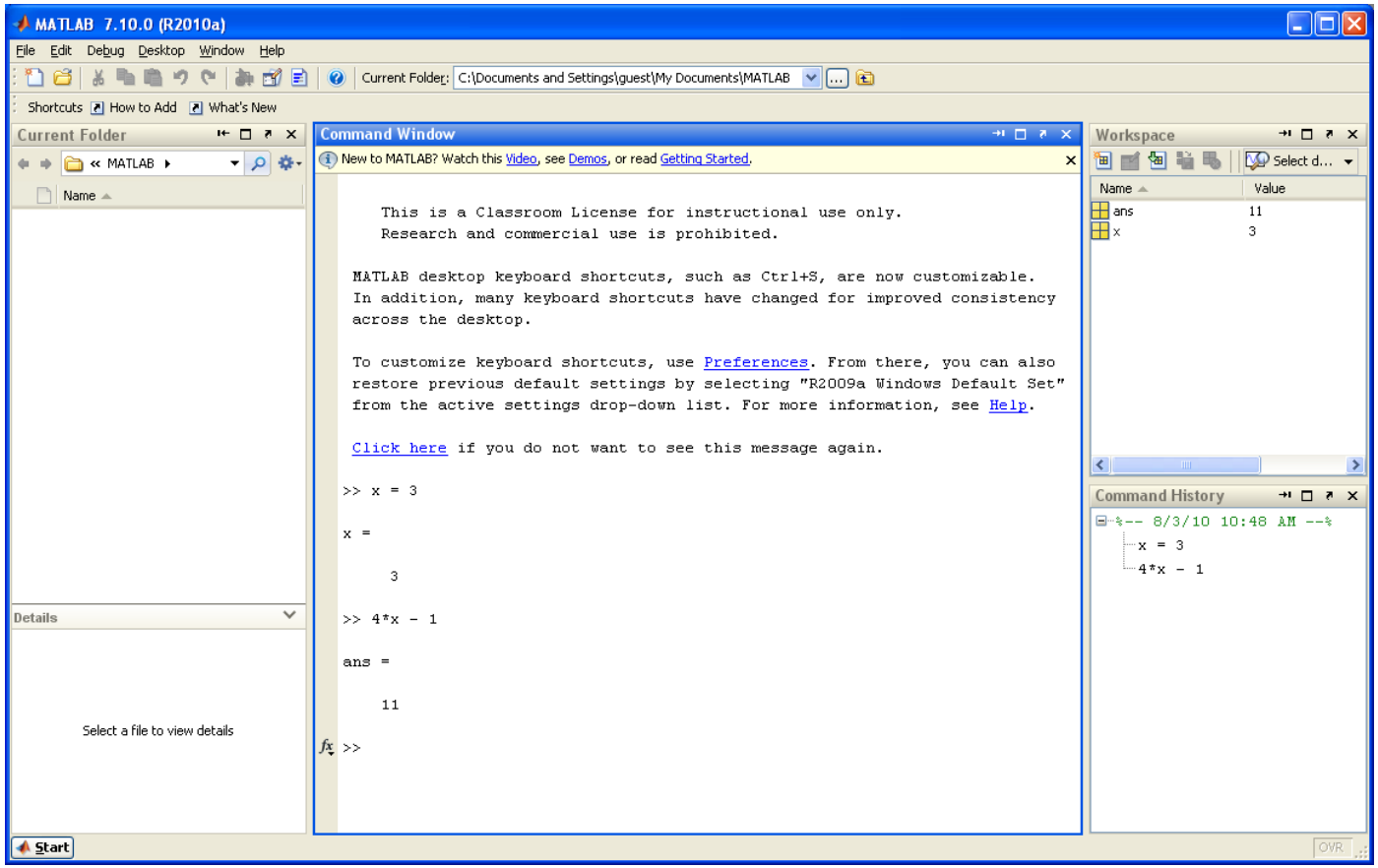
- MATLAB echoes the result back to you in the command window.

Manipulate A Variable



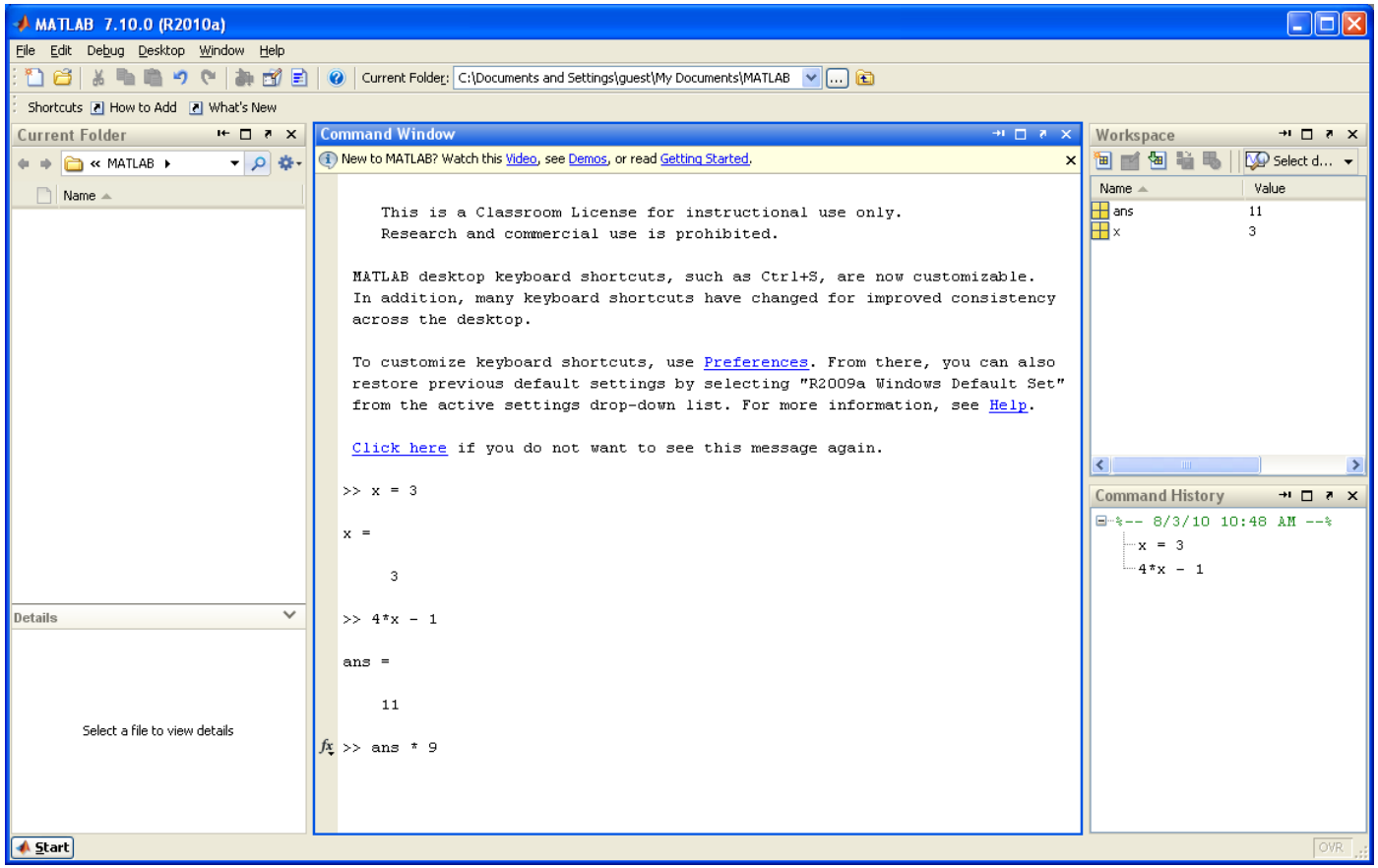
- Manipulate the variable using normal arithmetic operations...

Manipulate A Variable



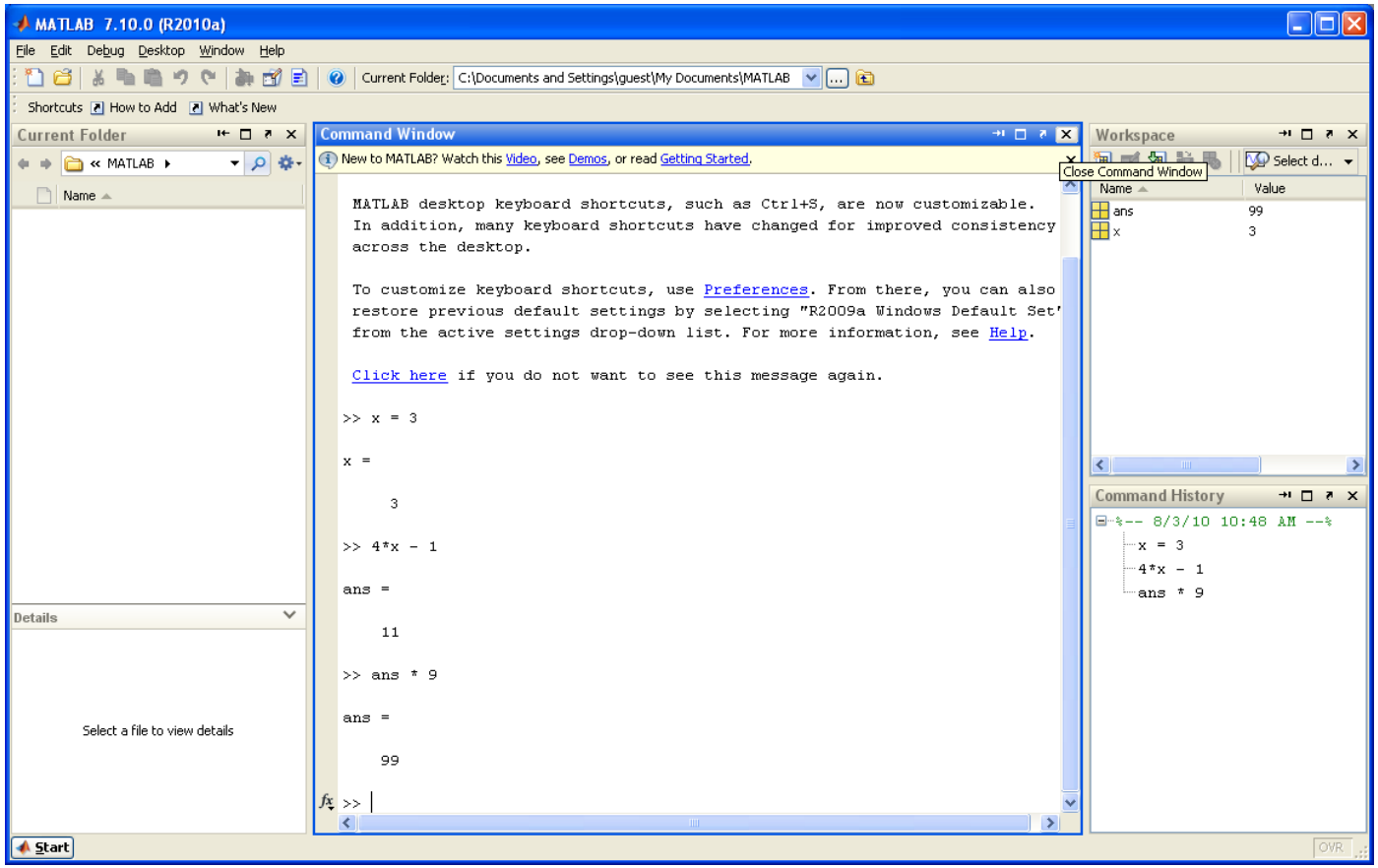
- MATLAB stores the result in the variable `ans`.

Manipulate A Variable



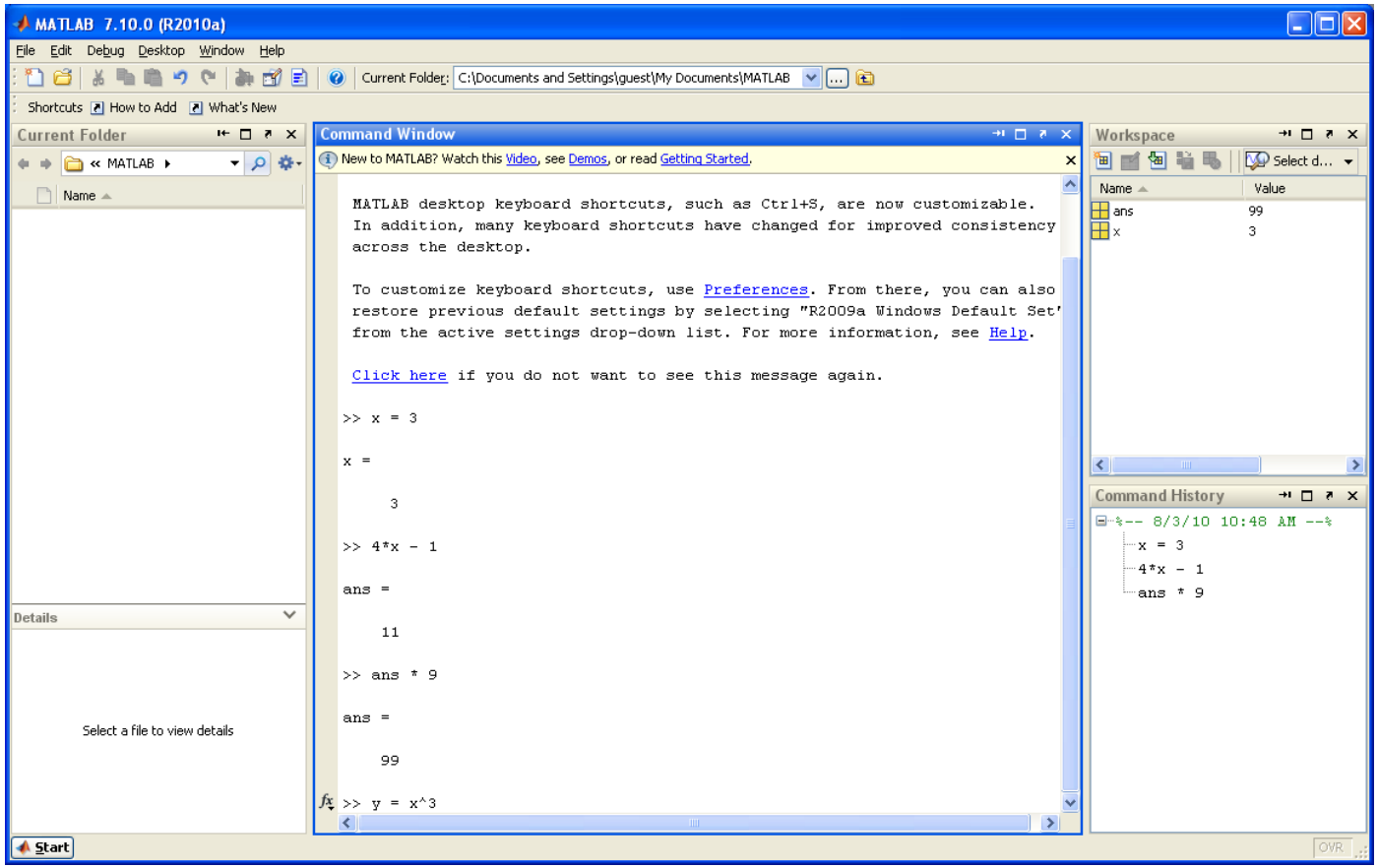
- `ans` can be manipulated like any other variable...

Manipulate A Variable



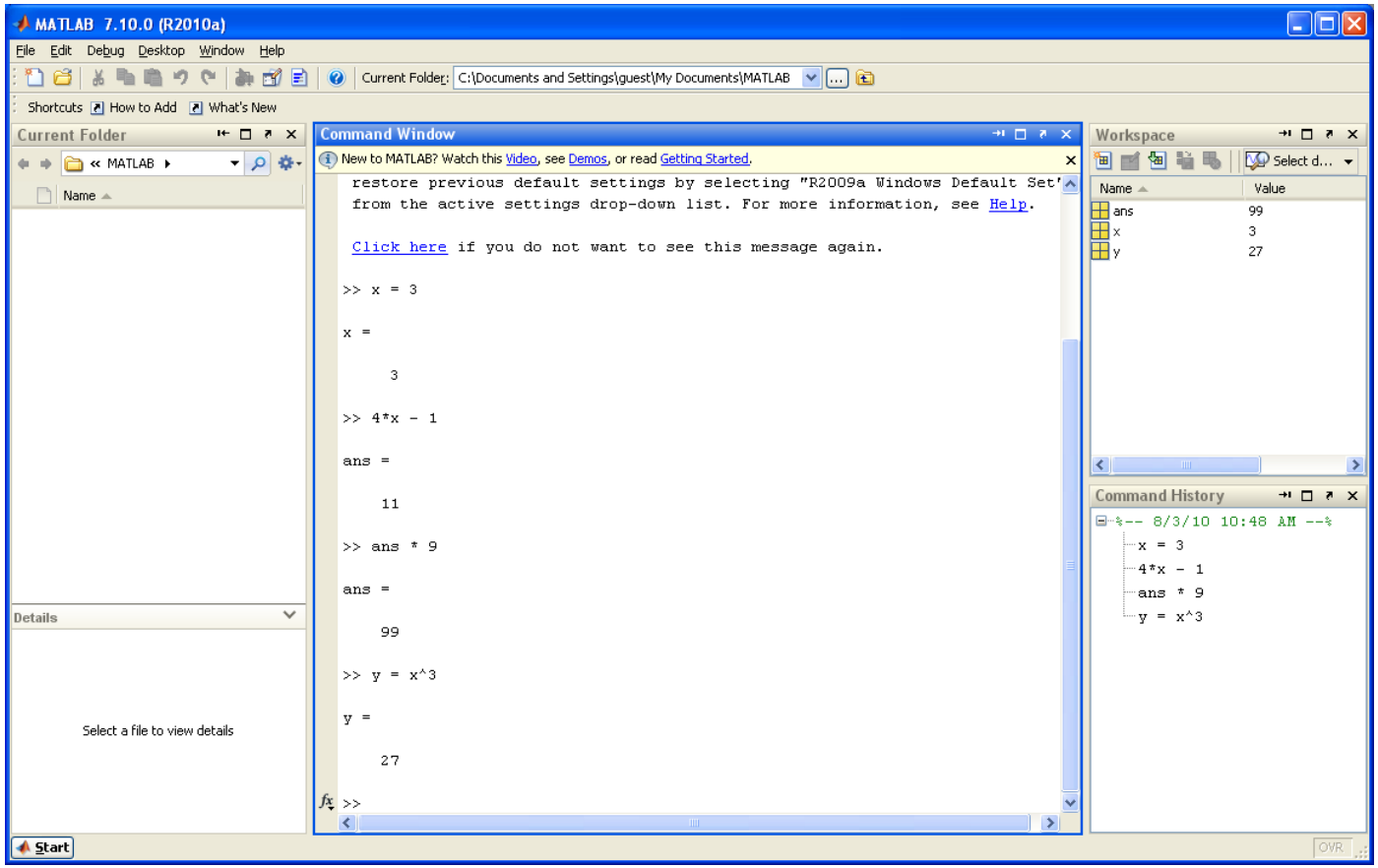
- `ans` can be manipulated like any other variable...

Manipulate A Variable



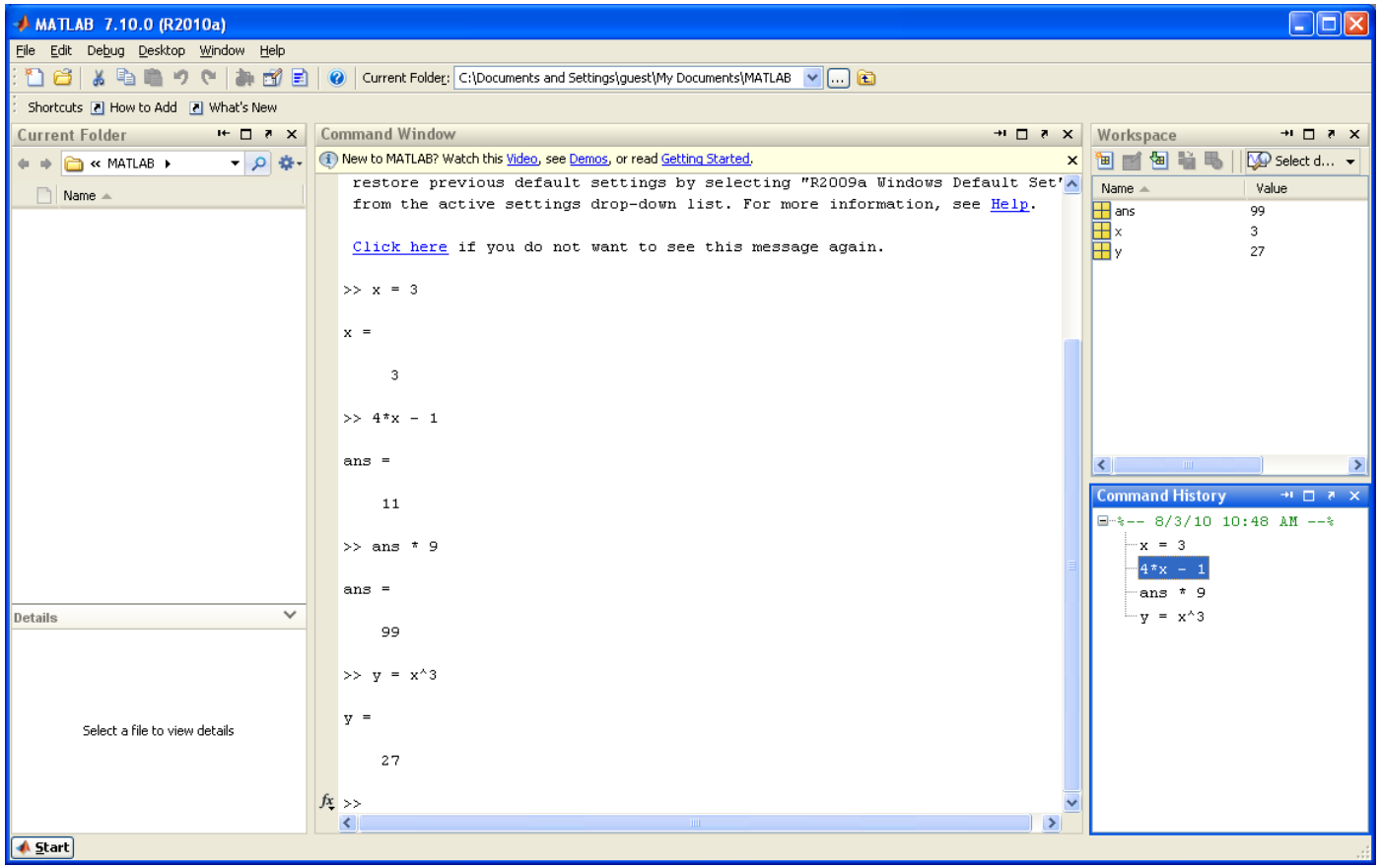
- ...but it's better to use your own variables to store results.

Manipulate A Variable



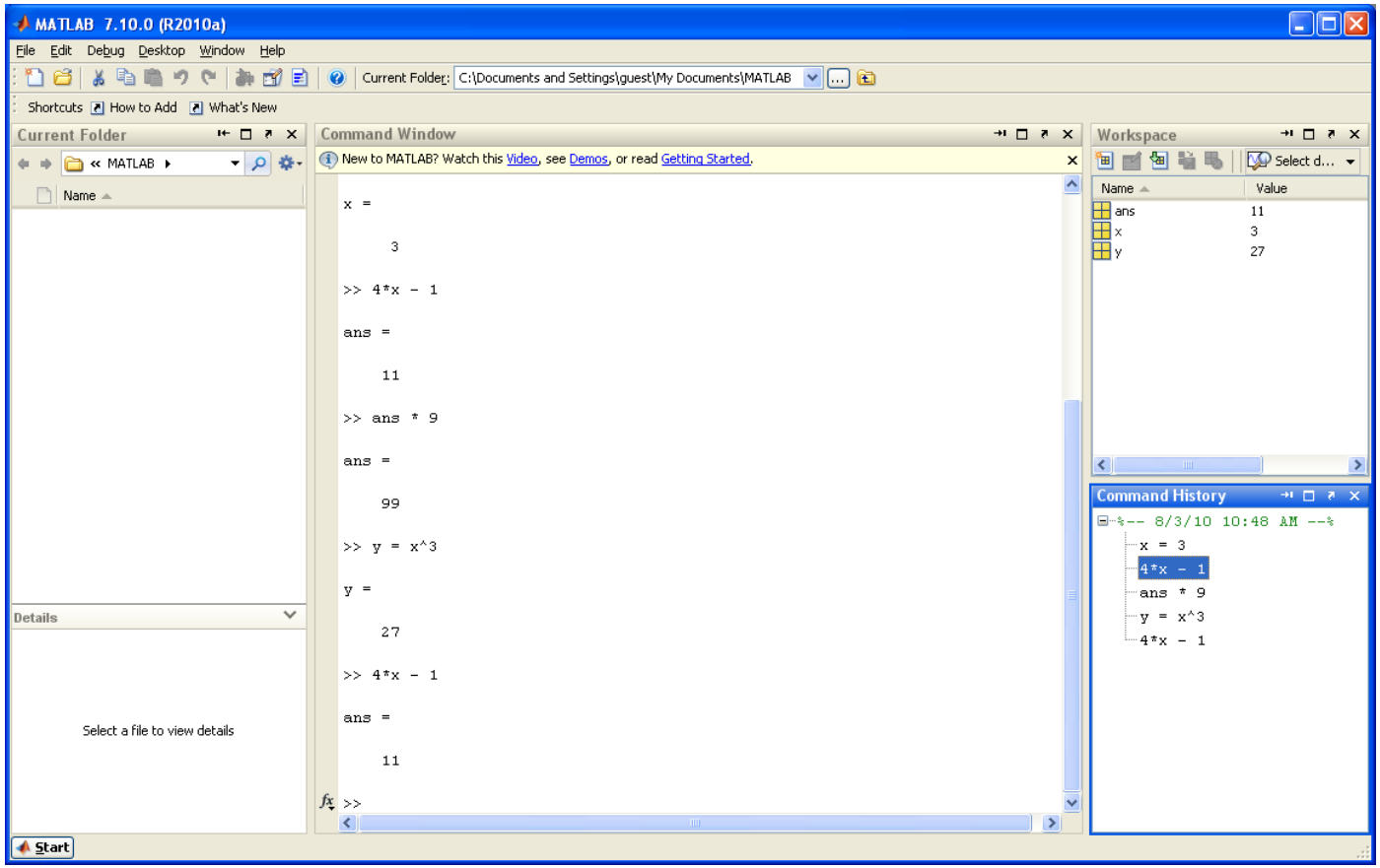
- ...but it's better to use your own variables to store results.

Using The History Window



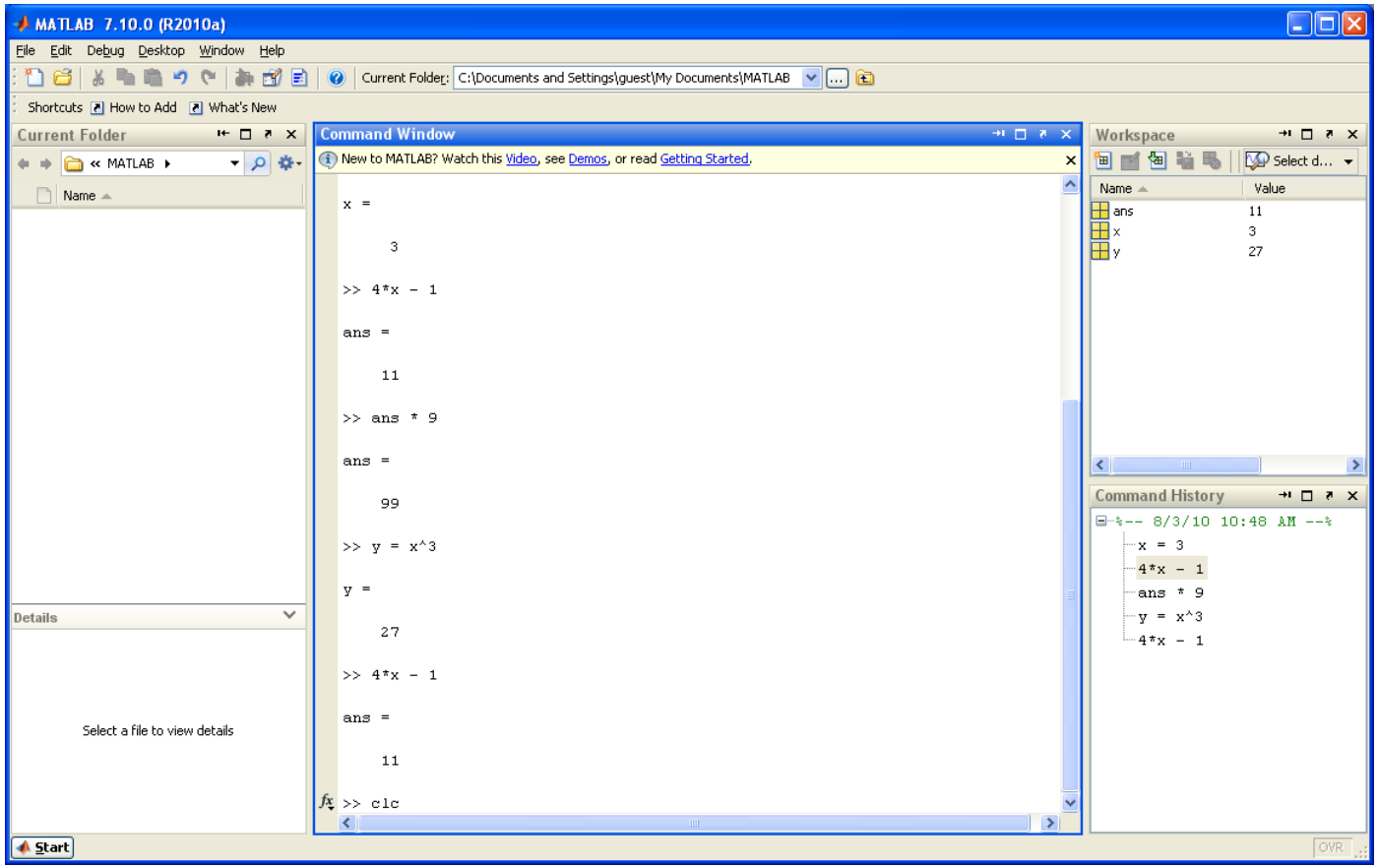
- Double-click on a command in the history window to re-execute it.

Using The History Window



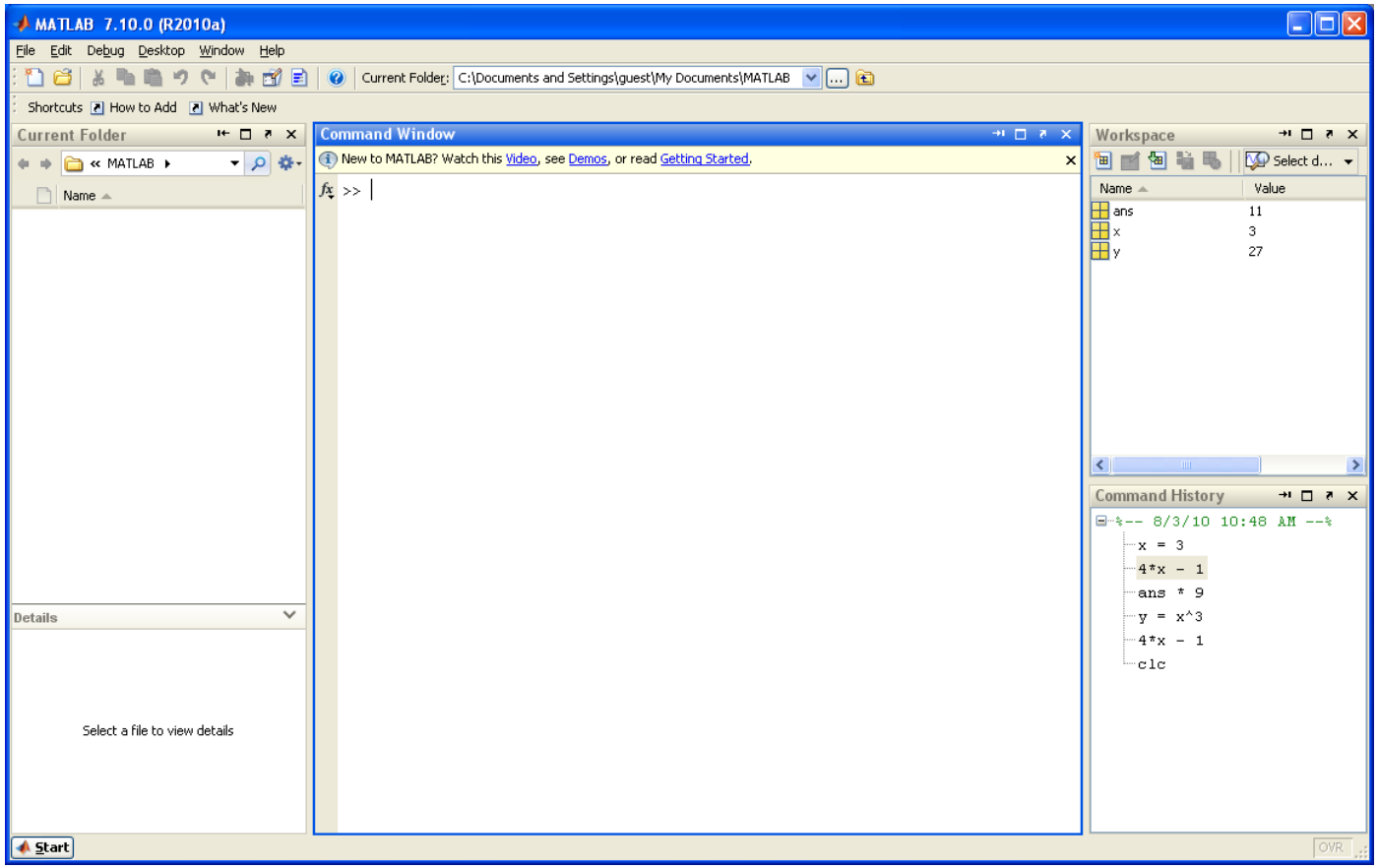
- Double-click on a command in the history window to re-execute it.

Clearing The Command Window



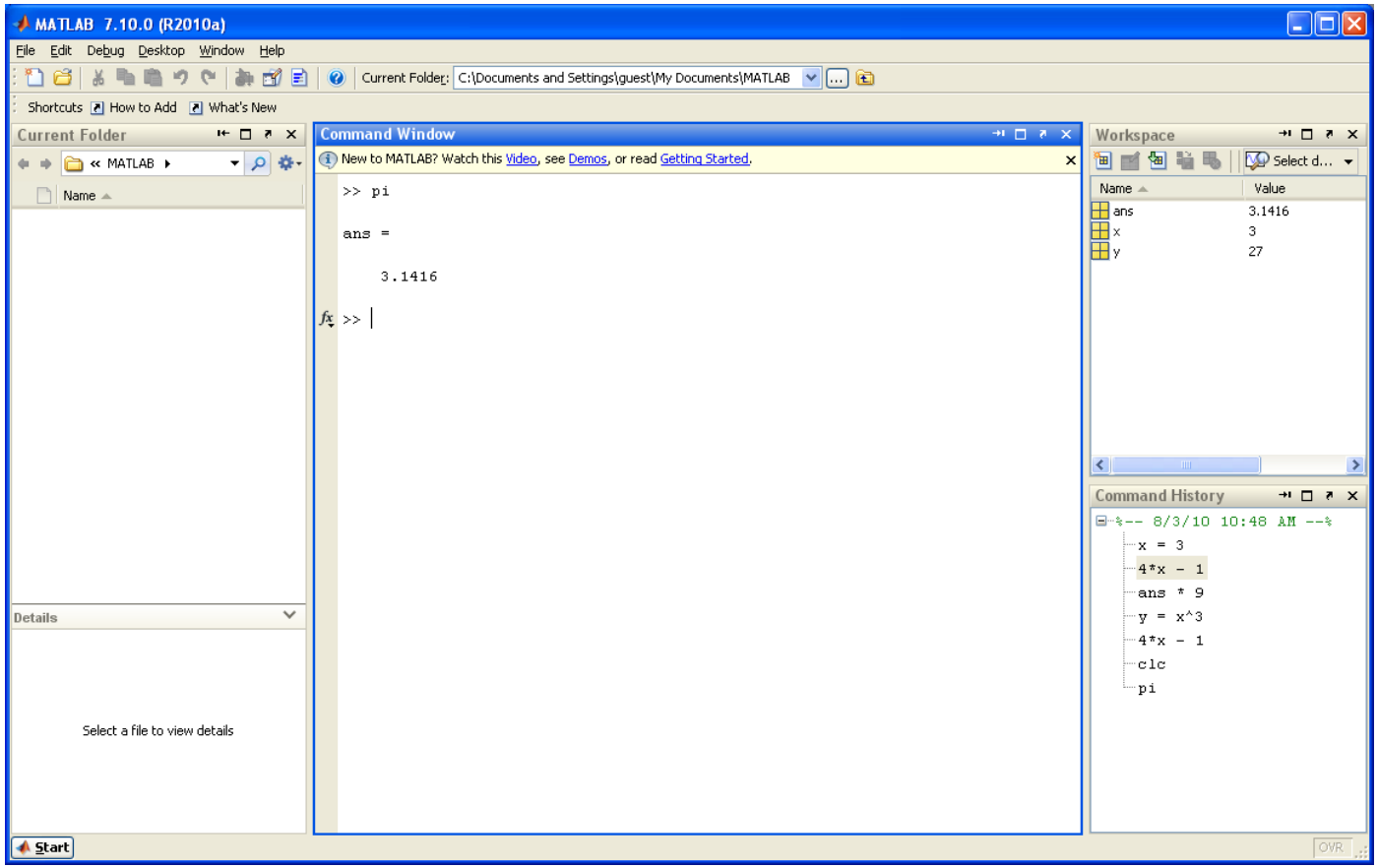
- The command `clc` clears the command window.

Clearing The Command Window



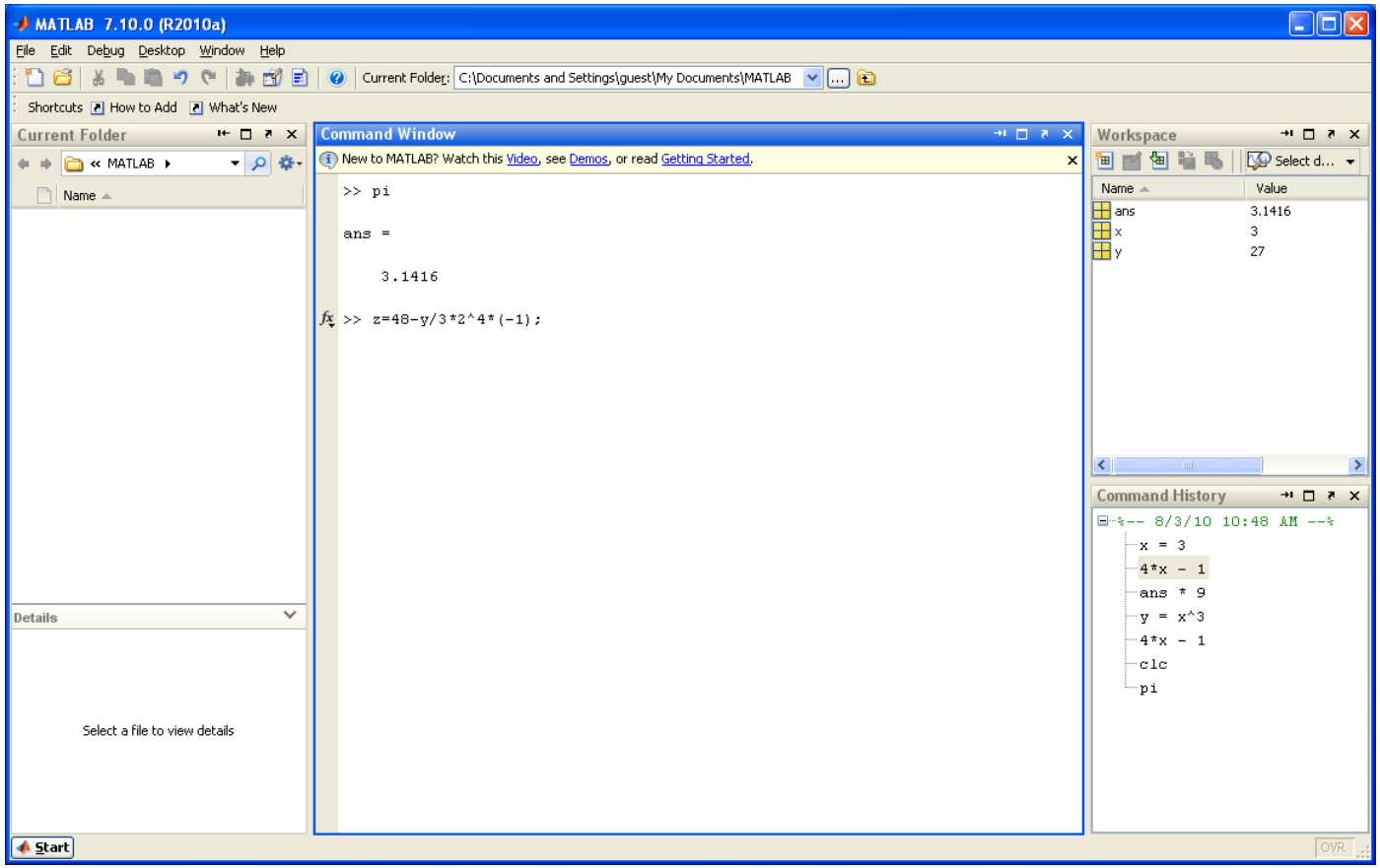
- `clc` doesn't clear the variables themselves, only the display.

Predefined Constants



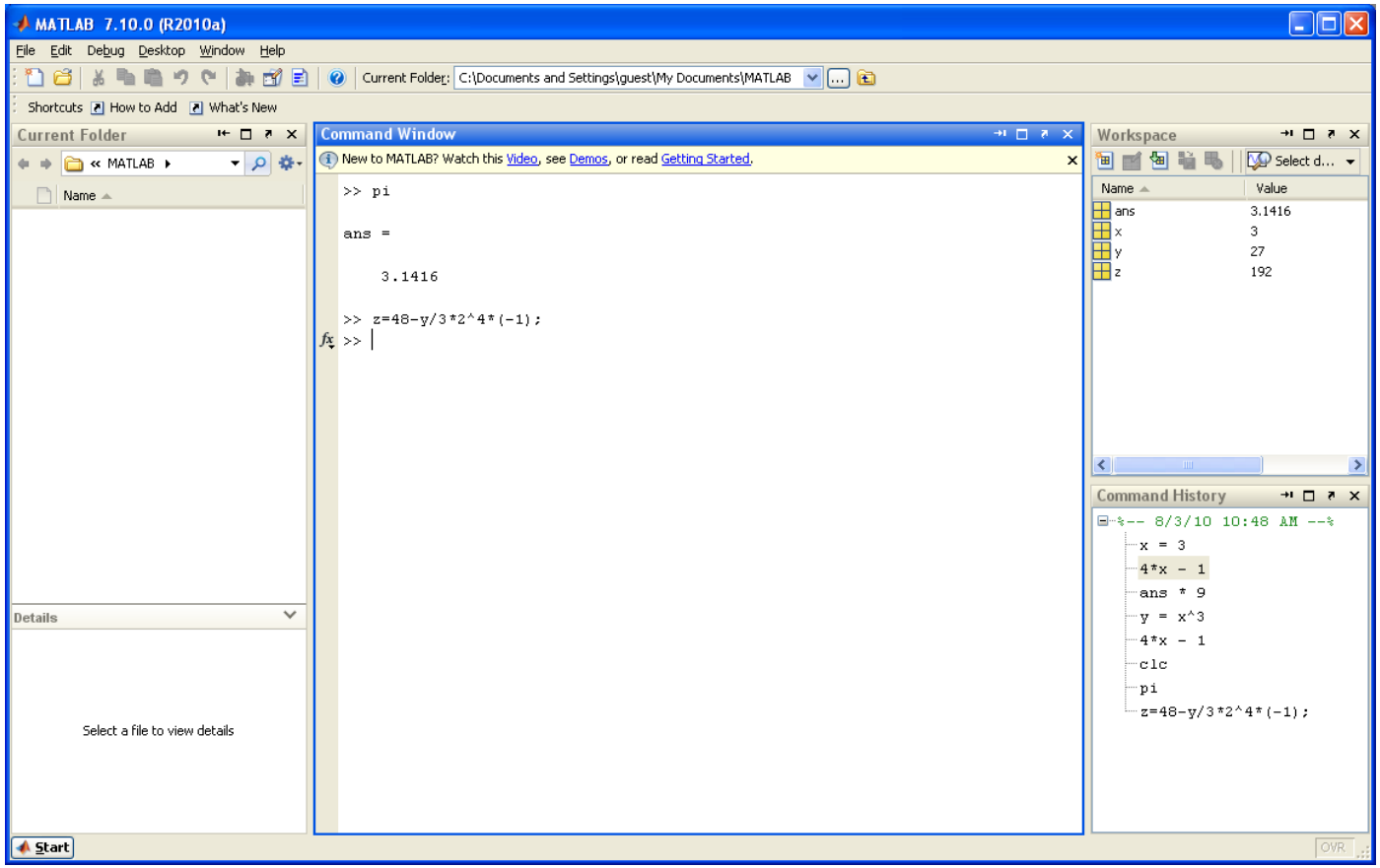
- MATLAB knows the value of π. It is stored in the pre-defined variable `pi`.

Suppressing Output



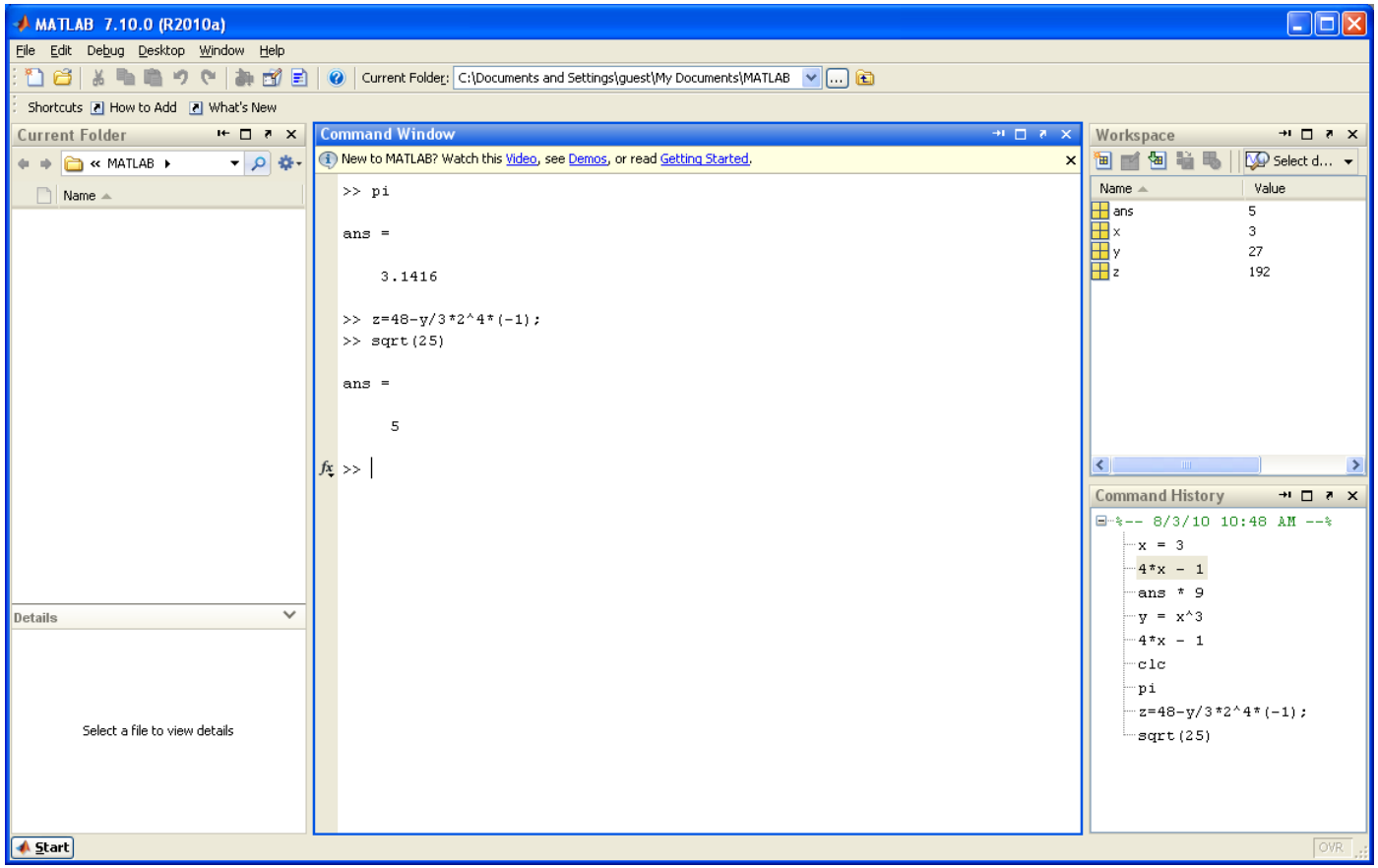
- You can suppress MATLAB's command window output by ending a command with a semicolon (;).

Suppressing Output



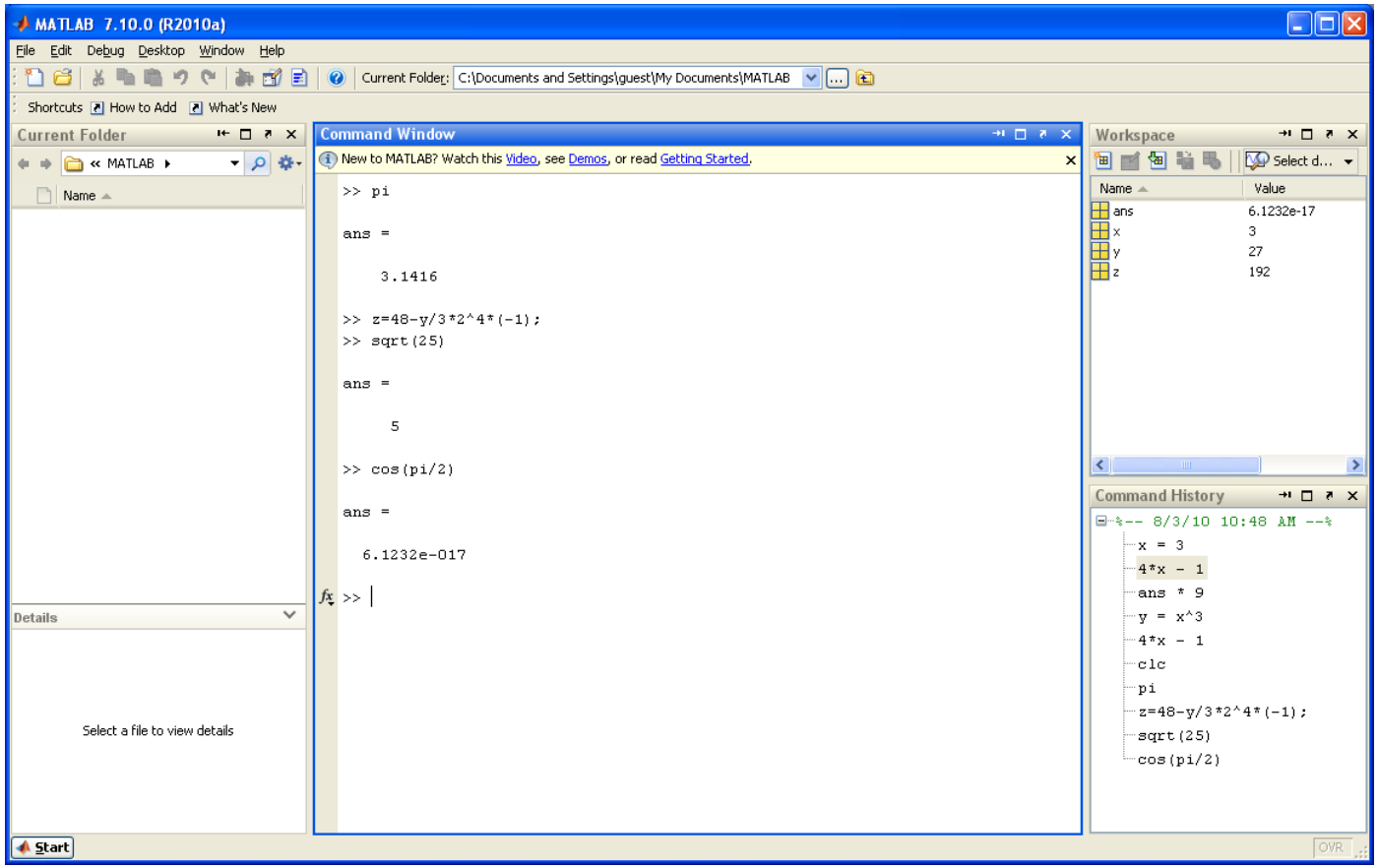
- You can suppress MATLAB's command window output by ending a command with a semicolon (;).

Built-In Functions



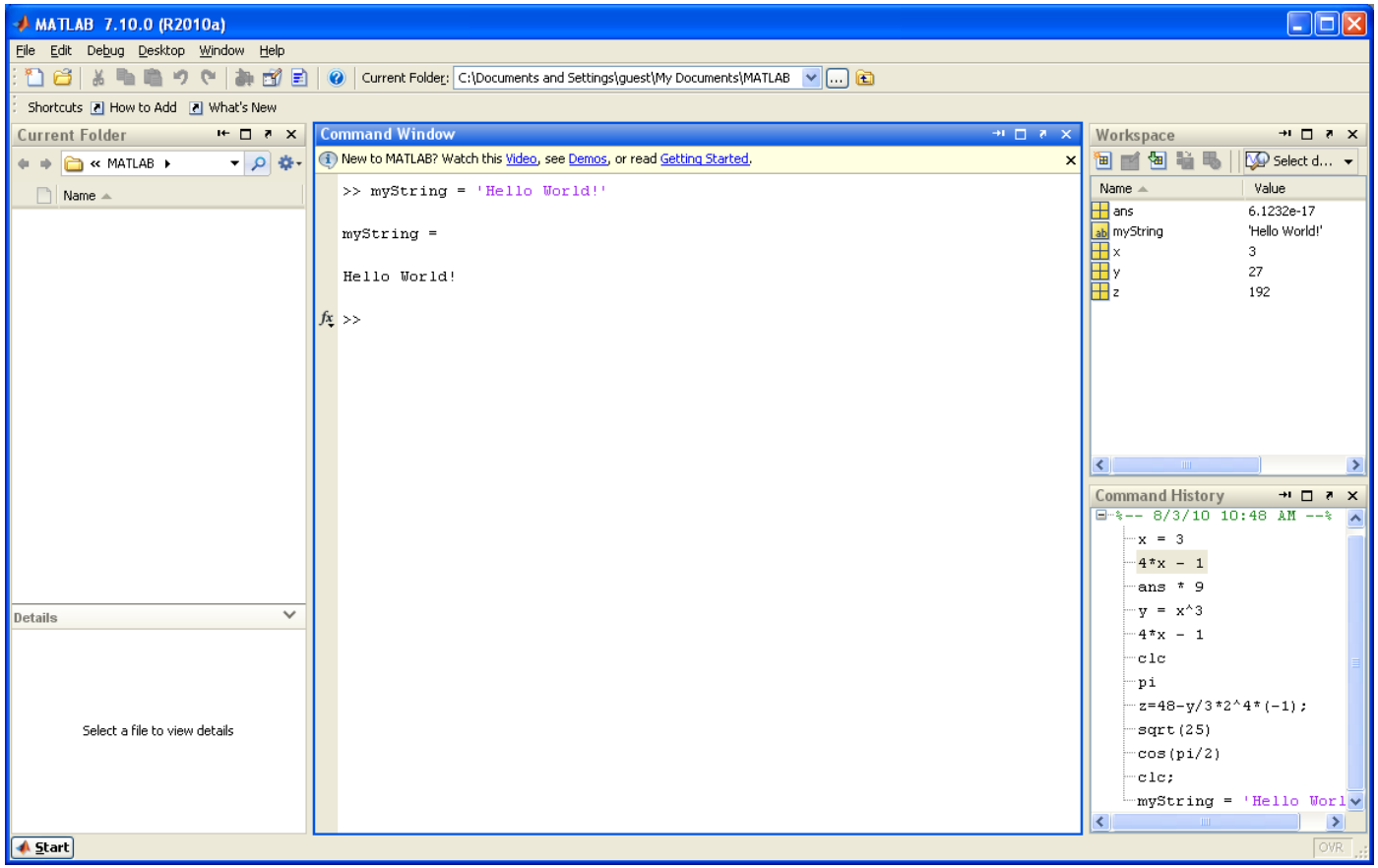
- Most common mathematical functions are built into MATLAB.

Built-In Functions



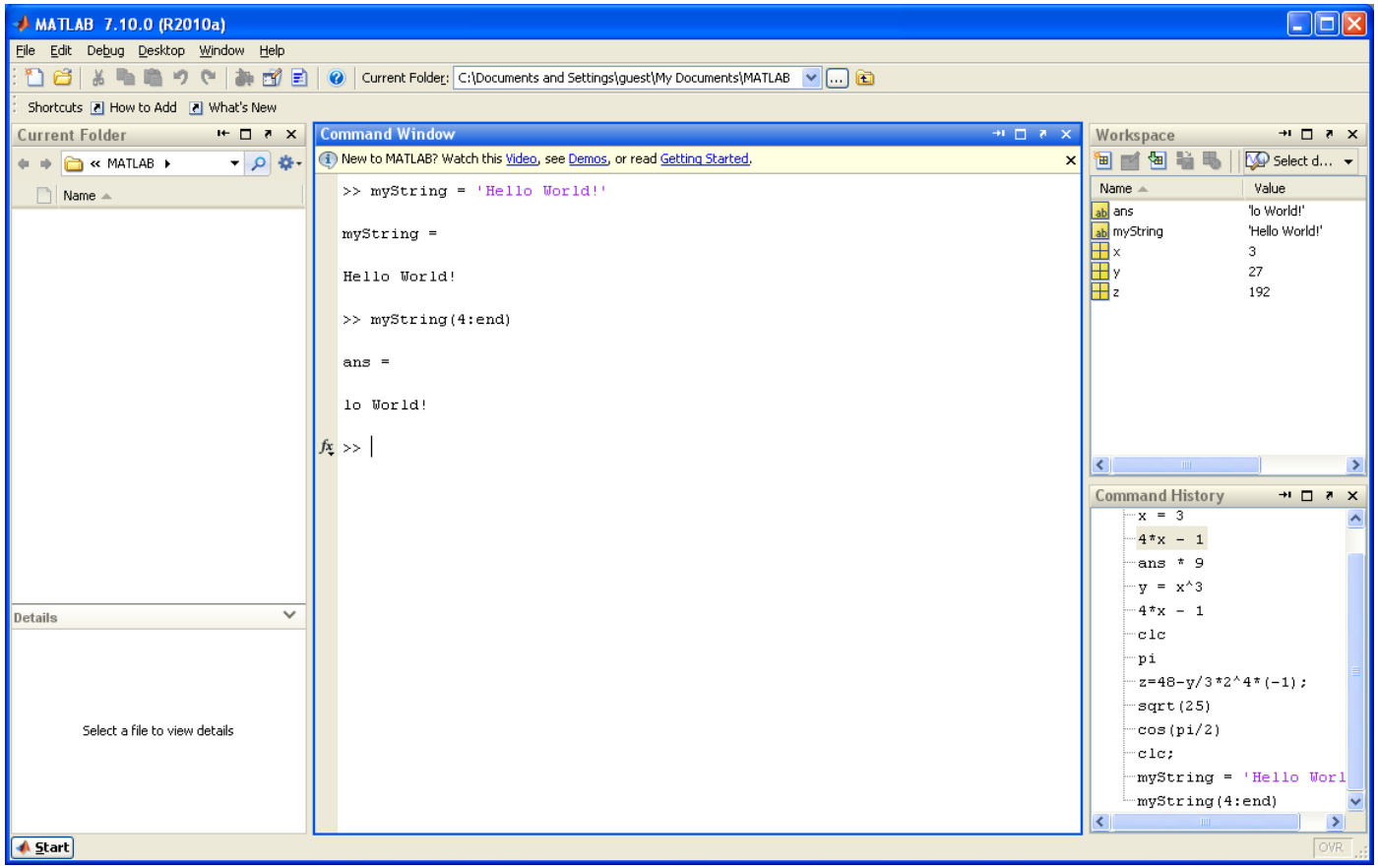
- Most common mathematical functions are built into MATLAB.

String Variables For Text



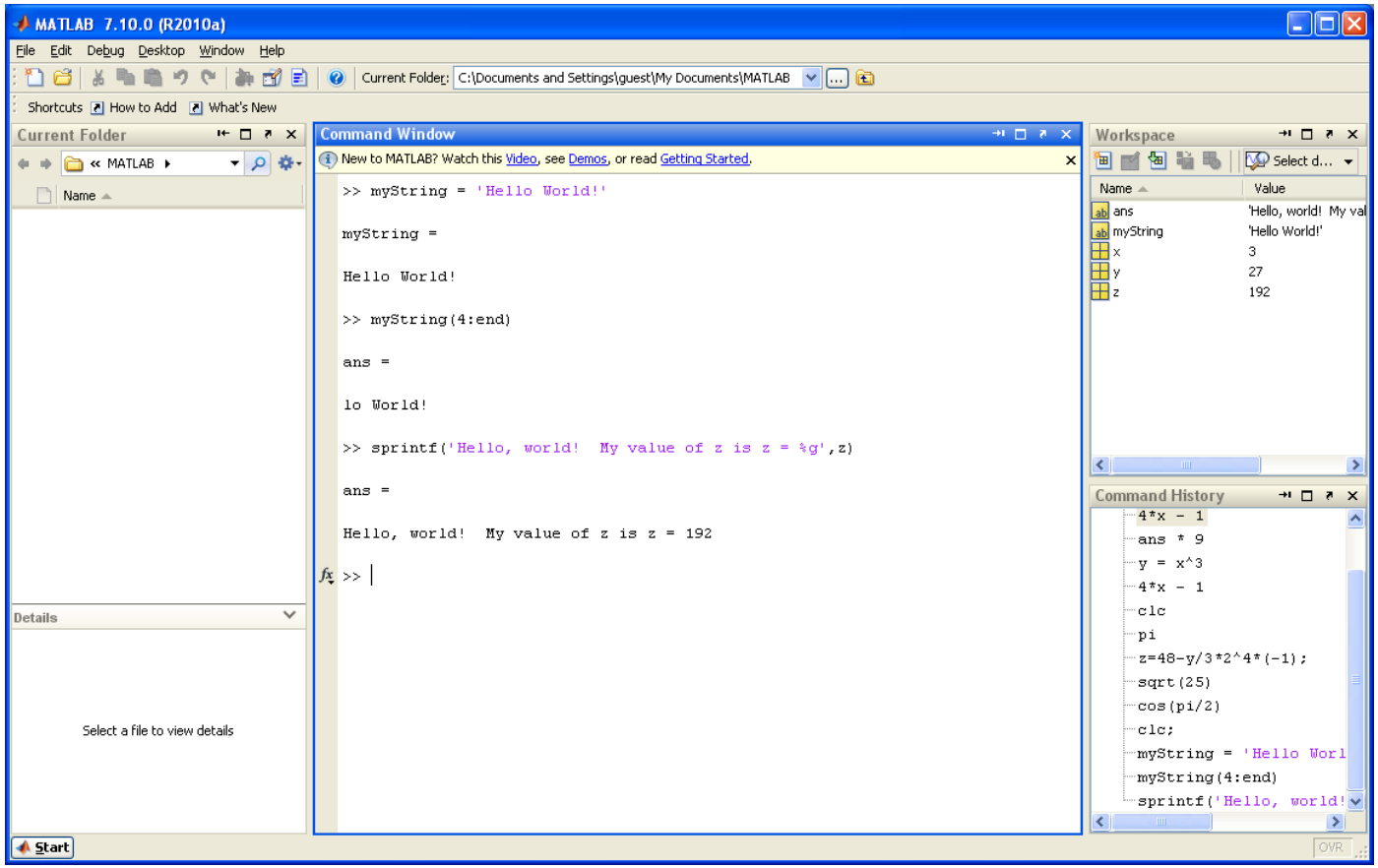
- Text can be stored in *string variables*.

String Variables For Text



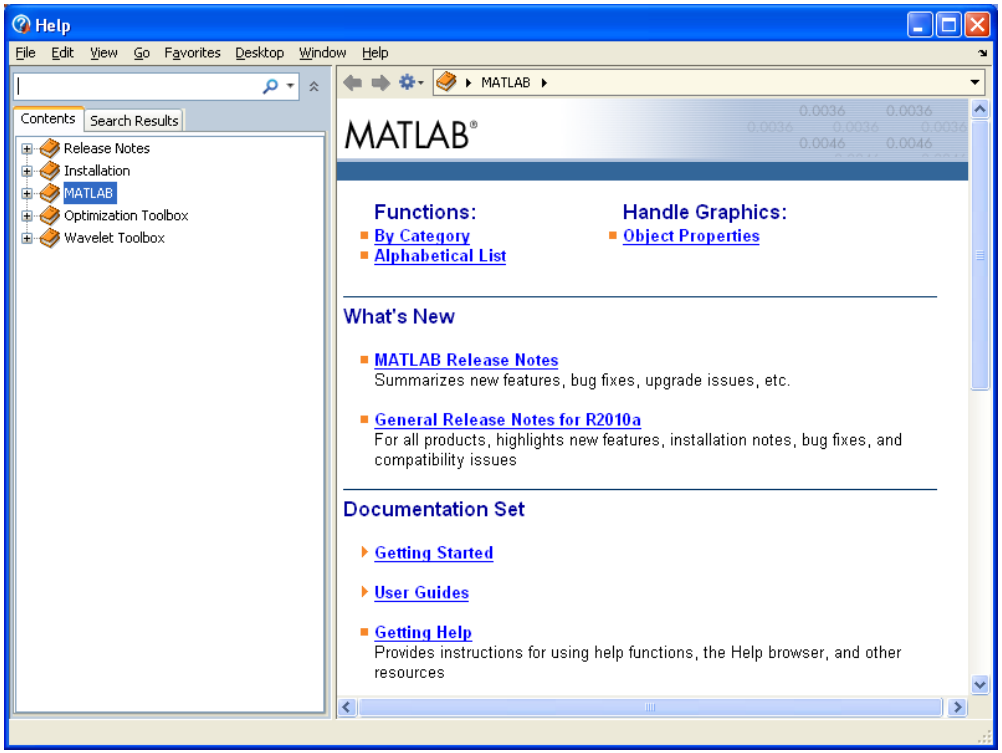
- You can do lots of manipulations with string variables as well.

String Variables For Text



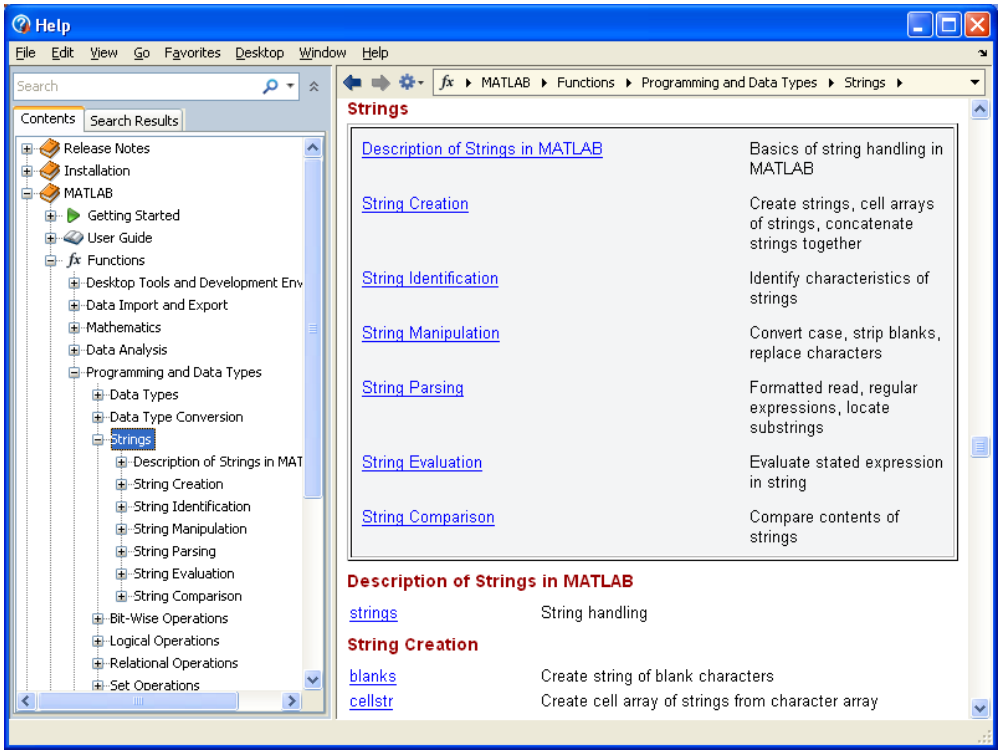
- You can do lots of manipulations with string variables as well.

The Help Browser



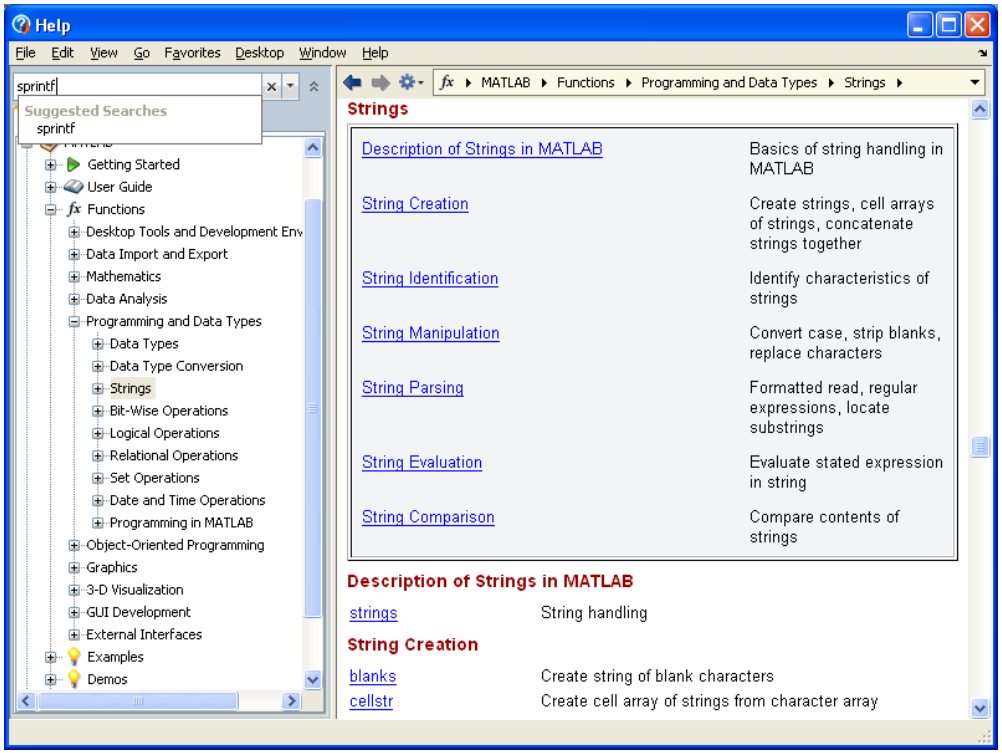
- The help browser is your friend. Access it from the Help menu (Help → Product Help).

The Help Browser



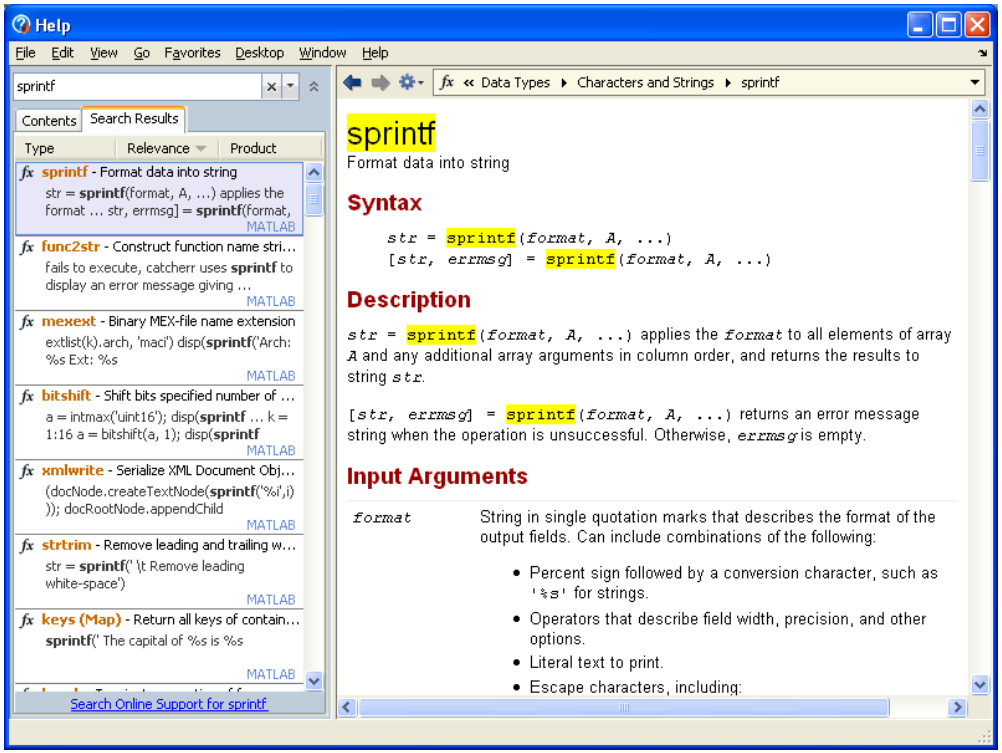
- You can either browse from the contents pane...

The Help Browser



- ...or search by typing in the search box.

The Help Browser



- ...or search by typing in the search box.

Practice

Practice 1.1

Let $x = 3$ and $y = 5$. Perform the following calculations in MATLAB's command window:

1. $z = 3x - 4y + 2$
2. $w = \frac{y}{3x+z}$
3. $\alpha = \sqrt{100 + z}$

Practice 1.2

Look up the `sprintf` command using MATLAB's help browser. Use it to display the string `'My value of alpha is xxxx.'`, where `xxxx` is replaced by the value of α you calculated in the last problem.

Outline

Arithmetic Operations

Basic Arithmetic Operators

+	addition
-	subtraction
*	multiplication
/	division
^	exponentiation

Order Of Operations

Order Of Operations

1. Calculations done from left to right
2. Parenthesized items evaluated first
   - Nested parentheses evaluated from innermost to outermost
3. function calls evaluated next (e.g. `sqrt`, `cos`, etc...)
4. exponents and radicals evaluated next
5. unary negations evaluated next
6. multiplications and divisions evaluated next
7. addition and subtraction evaluated last

Example

Example: Order Of Operations

$$
\begin{aligned}
\texttt{4+24/2\^{}3+6*(5+cos(pi))} &= \texttt{4+24/2\^{}3+6*(5+-1)} \\
&= \texttt{4+24/2\^{}3+6*(5-1)} \\
&= \texttt{4+24/2\^{}3+6*4} \\
&= \texttt{4+24/8+6*4} \\
&= \texttt{4+3+6*4} \\
&= \texttt{4+3+24} \\
&= \texttt{7+24} \\
&= 31
\end{aligned}
$$

Practice

Practice 1.3

Evaluate the following expressions by hand using the correct order of operations:

1. `8-6-3/2^2+1-5`
2. `4^1/2^3`
3. `2/3+2^sin(0)`

Practice 1.4

Give the MATLAB commands to evaluate the following expressions.

1. $\frac{x}{1+x^2}$
2. $\frac{x+1}{1+x^2}$
3. $2^{(2^x)}$
4. $4\cos(2\pi x)$

Outline

Variables And Variable Names

- To create a variable, just assign a value to a name:

```
>> a = 3;

>> favMovie1982 = 'Blade Runner';
```

- Rules for variable names:
  - first character must be a letter
  - other characters must be letters, numbers, or underscores (_).
  - variable names are *case sensitive*: `myVar` is different from `myvar`.

- Built-in variables (don't assign to these names):
  - `pi` stands for $\pi = 3.1415926....$
  - `ans` stores the last *unassigned* value.
  - `Inf` stands for positive infinity.
  - `NaN` stands for "not a number" (e.g. $0/0$).

Vectors, Matrices, and Arrays

- A array is a variable that can hold multiple values at the same time.
- The most common type of array is a matrix, which is a two-dimensional rectangular array.
- A vector is a one-dimensional rectangular array.
- There are also multidimensional arrays that have more than two dimensions.
- Sometimes we refer to plain everyday numbers as scalars.

Creating Vectors

- Create a vector by typing entries within square brackets:

```
>> myVec = [-1 3 pi 2.7183]

myVec =
    -1.0000      3.0000      3.1416      2.7183
```

Creating Vectors

- There's a special syntax to create a vector of evenly-spaced entries:

```
>> myArray = [0 : 2 : 10]

myArray =

     0     2     4     6     8    10
```

- The general syntax for this is

$$[startValue : skipValue : endValue]$$

Referencing Vector Entries

- Reference individual entries in a vector by their position:

```
>> myArray(1)

ans =

     0

>> myArray(3)

ans =

     4
```

Creating Matrices

- Similarly you can create a matrix by typing in the entries individually. Use a semicolon (;) for the row separator.

```
>> myMatrix = [1 2 3; 4 5 6]

myMatrix =

     1     2     3
     4     5     6
```

- This is a 2×3 matrix: 2 rows, 3 columns.

Referencing Matrix Entries

- Again, you reference individual entries in a matrix by their position, first the row number, then the column number:

```
>> myMatrix(1,3)

ans =

     3

>> myMatrix(2,1)

ans =

     4
```

Matrix Creation Commands

- You can create a matrix of all zeros using the `zeros` command. For example, to create a 2×4 zero matrix, use

```
>> M = zeros(2,4)

M =

     0     0     0     0
     0     0     0     0
```

- The first argument is the number of rows, the second is the number of columns.

- There is a similar command called `ones`.

Creating Random Matrices

- The `rand` command creates a matrix whose entries are random numbers uniformly distributed on $[0, 1]$:

```
>> rM = rand(3,2)

rM =

    0.8147    0.9134
    0.9058    0.6324
    0.1270    0.0975
```

- There is a similar command called `randn`. It creates matrices whose entries have the standard normal distribution.
- There's another command called `randi`. It creates matrices whose entries are random integers.
- For more info consult the Help Browser.

Row and Column Vectors

- Notice that a vector is just a matrix with only one row; i.e. a $1 \times n$ matrix. Sometimes we call these row vectors.
- A $n \times 1$ matrix (i.e. just one column) is called a column vector.

```
>> colVec = randn(3,1)

colVec =
    -0.4336
     0.3426
     3.5784
```

- note: MATLAB distinguishes between row and column vectors!
- note: The transpose operator (.') converts row vectors to column vectors and *vice versa*.

Operations On Arrays

+, -, *, /, and ^ are defined for array variables, but only in certain cases:

- `A+B` and `A-B` are defined so along as `A` and `B` are the same size (same number of rows and columns). In this case the sum or difference is computed *entrywise*:

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 6 & 8 \\ 10 & 12 \end{pmatrix}$$

- The definitions of `A*B`, `A/B`, and `A^n` come from the mathematical field of Linear Algebra. We do not need them at present.

Operations On Arrays

`+`, `-`, `*`, `/`, and `^` are defined for array variables, but only in certain cases:

- `A+B` and `A-B` are defined so along as `A` and `B` are the same size (same number of rows and columns). In this case the sum or difference is computed *entrywise*:

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 6 & 8 \\ 10 & 12 \end{pmatrix}$$

- The definitions of `A*B`, `A/B`, and `A^n` come from the mathematical field of Linear Algebra. We do not need them at present.

Operations On Arrays

`+`, `-`, `*`, `/`, and `^` are defined for array variables, but only in certain cases:

- `A+B` and `A-B` are defined so along as `A` and `B` are the same size (same number of rows and columns). In this case the sum or difference is computed *entrywise*:

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 6 & 8 \\ 10 & 12 \end{pmatrix}$$

- The definitions of `A*B`, `A/B`, and `A^n` come from the mathematical field of Linear Algebra. We do not need them at present.

Vectorized Operations On Arrays

MATLAB does define an entrywise product (`.*`), entrywise quotient (`./`), and entrywise exponentiation (`.^`).

- `A.*B` and `A./B` compute the entrywise product and quotient of `A` and `B`. `A` and `B` must be of the same size.

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} .* \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 5 & 12 \\ 21 & 32 \end{pmatrix}$$

- `A.^n` raises each entry of `A` to the n^{th} power:

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} .^3 = \begin{pmatrix} 1 & 8 \\ 27 & 64 \end{pmatrix}$$

Vectorized Operations On Arrays

MATLAB does define an entrywise product (`.*`), entrywise quotient (`./`), and entrywise exponentiation (`.^`).

- `A.*B` and `A./B` compute the entrywise product and quotient of `A` and `B`. `A` and `B` must be of the same size.

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} .* \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 5 & 12 \\ 21 & 32 \end{pmatrix}$$

- `A.^n` raises each entry of `A` to the n^{th} power:

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} .^3 = \begin{pmatrix} 1 & 8 \\ 27 & 64 \end{pmatrix}$$

Vectorized Operations On Arrays

MATLAB does define an entrywise product (`.*`), entrywise quotient (`./`), and entrywise exponentiation (`.^`).

- `A.*B` and `A./B` compute the entrywise product and quotient of `A` and `B`. `A` and `B` must be of the same size.

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} .* \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 5 & 12 \\ 21 & 32 \end{pmatrix}$$

- `A.^n` raises each entry of `A` to the $\mathtt{n}^{\text{th}}$ power:

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} .^3 = \begin{pmatrix} 1 & 8 \\ 27 & 64 \end{pmatrix}$$

Vectorized Operations On Arrays

Note: With the exception of exponentiation, these vectorized operations require the variables to be the same size. There is one exception to this: when one of the variables is a scalar (i.e. ordinary number)

- `3 + A` adds three to every entry of `A`. Similarly `A - 3` subtracts three from every entry of `A`.
- `4 * A` multiplies every entry of `A` by four. Similarly `A / 4` divides every entry of `A` by four.
- `2 .^ A` computes powers of two using the entries of `A`.

```
>> 2 .^ [1 2 3 4]

ans =

    2.0000    4.0000    8.0000   16.0000
```

Vectorized Operations On Arrays

Note: With the exception of exponentiation, these vectorized operations require the variables to be the same size. There is one exception to this: when one of the variables is a scalar (i.e. ordinary number)

- `3 + A` adds three to every entry of `A`. Similarly `A - 3` subtracts three from every entry of `A`.
- `4 * A` multiplies every entry of `A` by four. Similarly `A / 4` divides every entry of `A` by four.
- `2 .^ A` computes powers of two using the entries of `A`.

```
>> 2 .^ [1 2 3 4]

ans =

    2.0000    4.0000    8.0000   16.0000
```

Vectorized Operations On Arrays

Note: With the exception of exponentiation, these vectorized operations require the variables to be the same size. There is one exception to this: when one of the variables is a scalar (i.e. ordinary number)

- `3 + A` adds three to every entry of `A`. Similarly `A - 3` subtracts three from every entry of `A`.
- `4 * A` multiplies every entry of `A` by four. Similarly `A / 4` divides every entry of `A` by four.
- `2 .^ A` computes powers of two using the entries of `A`.

```
>> 2 .^ [1 2 3 4]

ans =

    2.0000    4.0000    8.0000   16.0000
```

Vectorized Operations On Arrays

Note: With the exception of exponentiation, these vectorized operations require the variables to be the same size. There is one exception to this: when one of the variables is a scalar (i.e. ordinary number)

- `3 + A` adds three to every entry of `A`. Similarly `A - 3` subtracts three from every entry of `A`.
- `4 * A` multiplies every entry of `A` by four. Similarly `A / 4` divides every entry of `A` by four.
- `2 .^ A` computes powers of two using the entries of `A`.

```
>> 2 .^ [1 2 3 4]

ans =

    2.0000    4.0000    8.0000    16.0000
```

Built-In Functions Are Vectorized

MATLAB's built-in functions are vectorized. If you give them an array as input, the function will be applied to each entry in the array.

```
>> v = [1 4 9 16];

>> sqrt(v)

ans =

    1.0000    2.0000    3.0000    4.0000
```

Example

Vectorized Operations

Compute $\frac{1}{1+x^2}$, where `x = [-3 : 3]`.

Solution

```
>> x = [-3 : 3]

x =

    -3.0000   -2.0000   -1.0000    0.0000    1.0000    2.0000    3.0000

>> y = 1 ./ (1 + x.^2)

y =

    0.1000    0.2000    0.5000    1.0000    0.5000    0.2000    0.1000
```

Example

Vectorized Operations

Compute $\frac{1}{1+x^2}$, where `x = [-3 : 3]`.

Solution

```
>> x = [-3 : 3]

x =

    -3.0000   -2.0000   -1.0000    0.0000    1.0000    2.0000    3.0000

>> y = 1 ./ (1 + x.^2)

y =

    0.1000    0.2000    0.5000    1.0000    0.5000    0.2000    0.1000
```

Practice

Practice 1.5

What do the following MATLAB commands do?

1. `[1 : 99]`
2. `var1 = [0 : 5 : 200];`
3. `var2 = [10 : -1 : 1]`

Practice 1.6

Use the Help Browser to look up the `randi` command. Use `randi` to create a 3×5 array of random integers, with all entries between 1 and 100. What is the position (row number, column number) of the maximum entry in your array?

Practice

Practice 1.7

Write a MATLAB command to

1. make a row vector containing the odd numbers from 1 to 37.
2. make a 3×4 matrix containing all 1's.
3. make a column vector containing five random integers, each between 1 and 6.

Practice 1.8

In three lines of MATLAB code, create two random 3×3 arrays, and compute the entrywise product of them. Use `sprintf` to display the (3,2) entry to the command window.

Practice 1.9

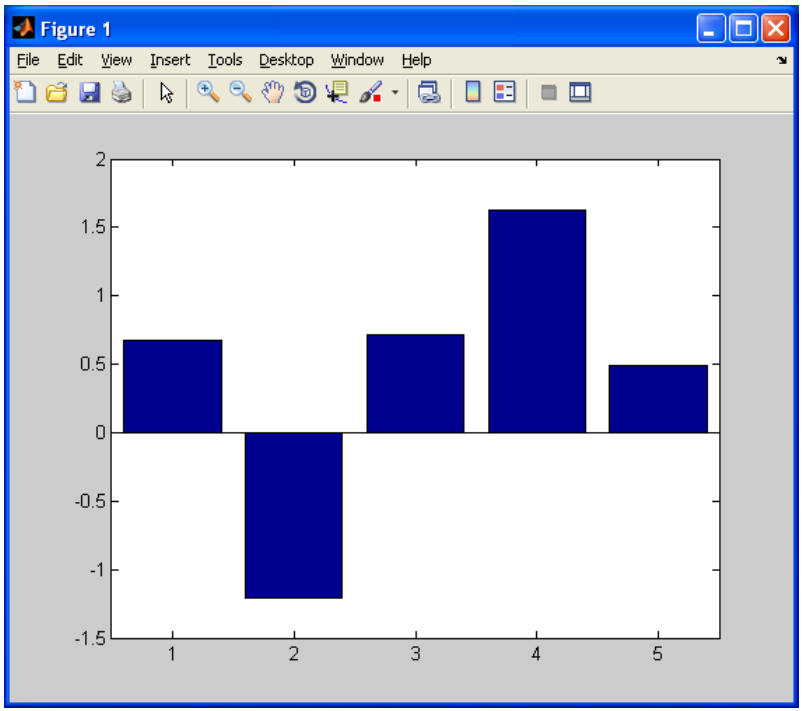
In a single line of MATLAB code compute the cube roots of the even integers between 2 and 20 inclusive.

Outline

Bar Plots Of Vectors

- The function `bar` makes a bar plot of a vector.

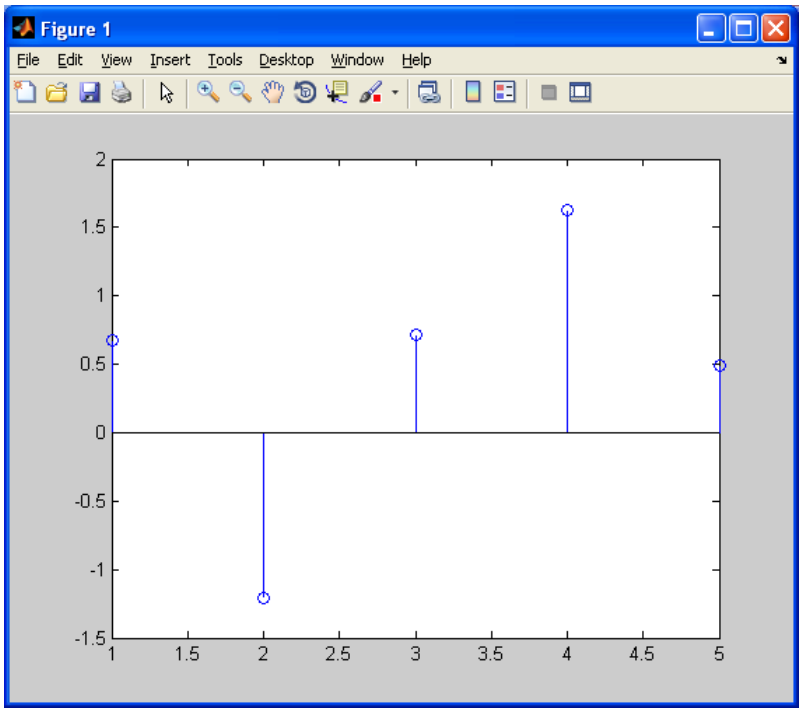
```
>> v = randn(1,5);
>> bar(v);
```



Stem Plots Of Vectors

- The function `stem` makes a "stem plot" of a vector.

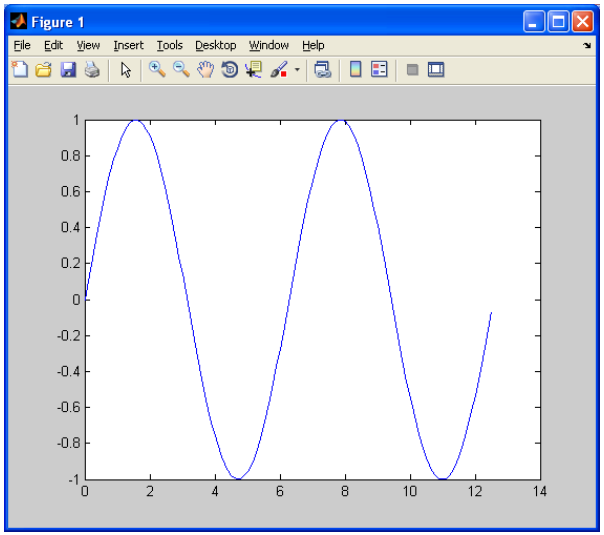
```
>> v = randn(1,5);
>> stem(v);
```



Function Plotting

- To plot a function, put the x-values in one array, the y-values in another array, and use the `plot` command.

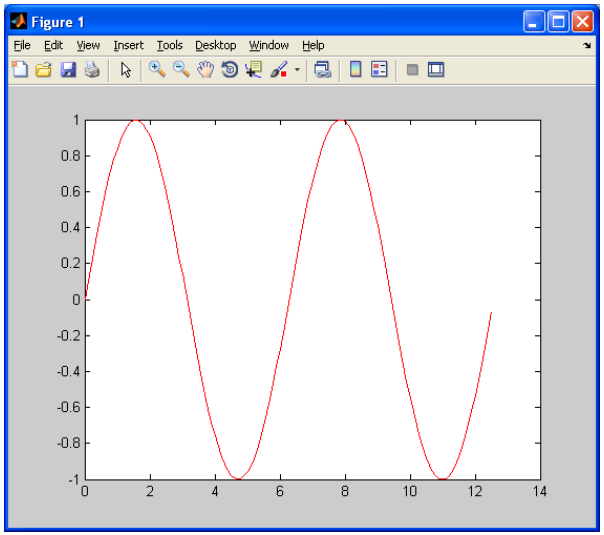
```
>> xVals = [0 : 0.1 : 4*pi];
>> yVals = sin(xVals);
>> plot(xVals,yVals);
```



Function Plotting

- Use a "format string" to change the color of the plot. Let's redo the previous plot in red.

```
>> xVals = [0 : 0.1 : 4*pi];
>> yVals = sin(xVals);
>> plot(xVals,yVals, 'r');
```



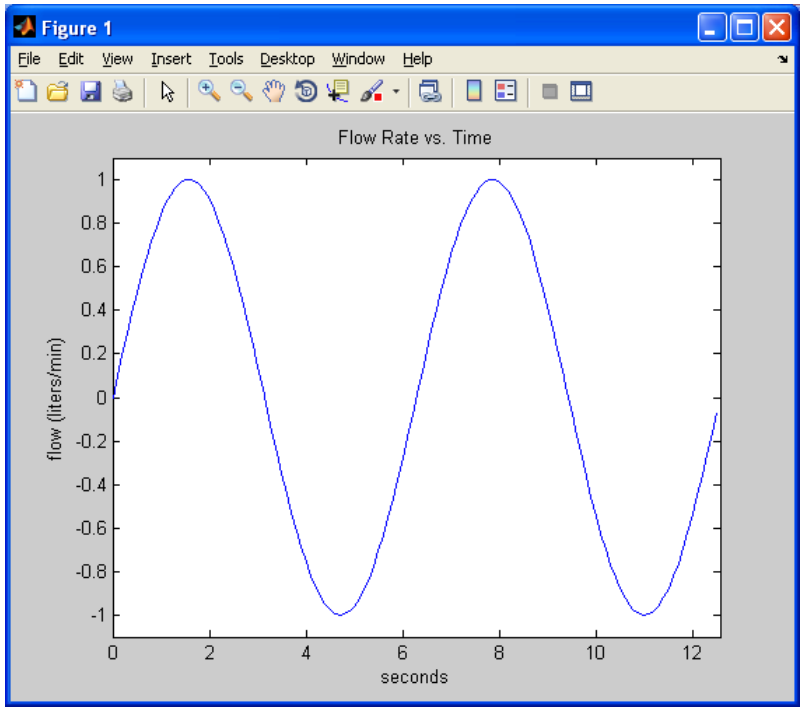
Function Plotting

- There is a host of commands for adding titles, axis labels, legends, and other decorations to a graph. Here's a small sample of the possibilities:

```
>> xVals = [0 : 0.1 : 4*pi];
>> yVals = sin(xVals);
>> plot(xVals,yVals);
>> xlim([0,4*pi]);
>> ylim([-1.1,1.1]);
>> xlabel('seconds');
>> ylabel('flow (liters/min)');
>> title('Flow Rate vs.  Time');
```

- This gives us...

Function Plotting



- You can also use the controls on the figure's menu and toolbar to edit the graph.
- Play around with these commands, and explore the possibilities...

Example

Vectorizing In Plots

Plot $y = \sin(x)/x$ for $x \in [-20, 20]$.

Solution

```
>> x = [-20 : 0.1 : 20];
>> y = sin(x) ./ x;  % note vectorized arithmetic

>> plot(x,y)
```

Example

Vectorizing In Plots

Plot $y = \sin(x)/x$ for $x \in [-20, 20]$.

Solution

```
>> x = [-20 : 0.1 : 20];
>> y = sin(x) ./ x;  % note vectorized arithmetic
>> plot(x,y)
```

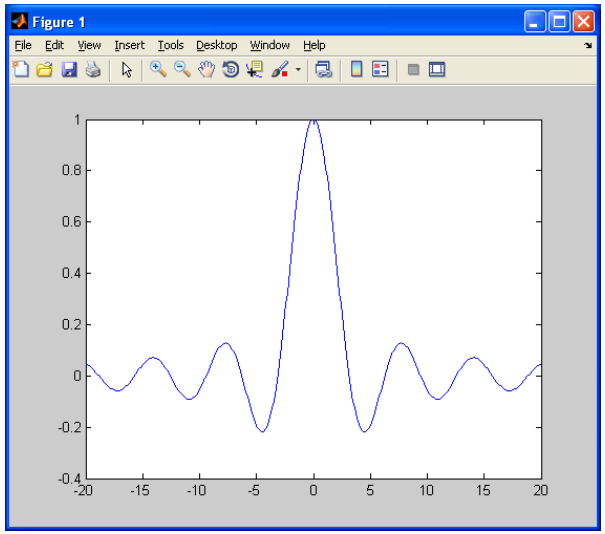
Example

Vectorizing In Plots

Plot $y = \sin(x)/x$ for $x \in [-20, 20]$.

Solution

```
>> x = [-20 : 0.1 : 20];
>> y = sin(x) ./ x;  % note vectorized arithmetic
>> plot(x,y)
```



Practice

Practice 1.10

Use the Help Browser to look up the `randi` command. Use `randi` to create a 1×20 column vector of random integers, with all entries between 1 and 10. Make a stem plot of this vector.

Practice 1.11

Plot $y = e^x$ for x between -1 and 1. Label your axes, and provide a title for your plot.

Practice 1.12

Plot $y = \frac{1-\cos(x)}{x^2}$ for x between -30 and 30. Label your axes, and provide a title for your plot.

Outline

The save And load Commands

Use the save command to save workspace variables to the current directory:

- Save all workspace variables to the file myFile.mat in the current directory:

```
>> save myFile;
```

  note: MATLAB automatically adds the .mat extension.

- Save only the variables x, y, and z to myFile.mat:

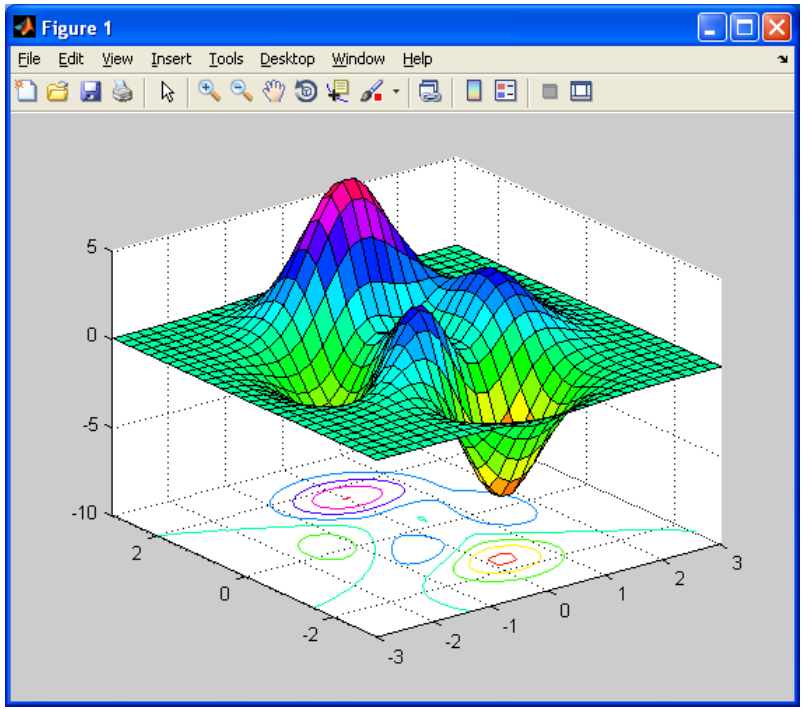
```
>> save myFile x y z;
```

The load command loads these variables to the workspace:

```
>> load myFile;
```

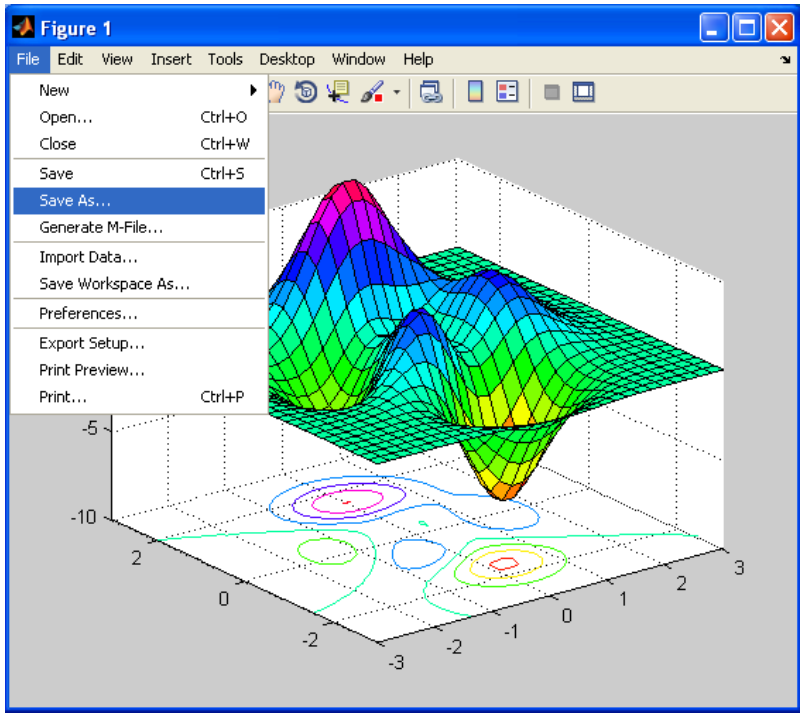
Saving MATLAB Figures

Saving figures as `.fig` files enables you to reload them later without having to regenerate all the data and decorations.



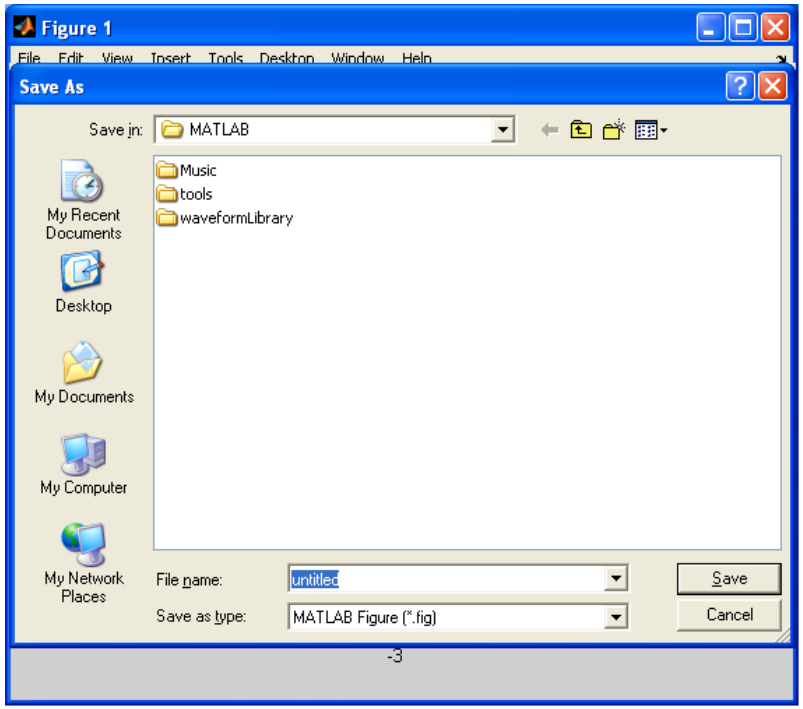
Saving MATLAB Figures

Saving figures as `.fig` files enables you to reload them later without having to regenerate all the data and decorations.



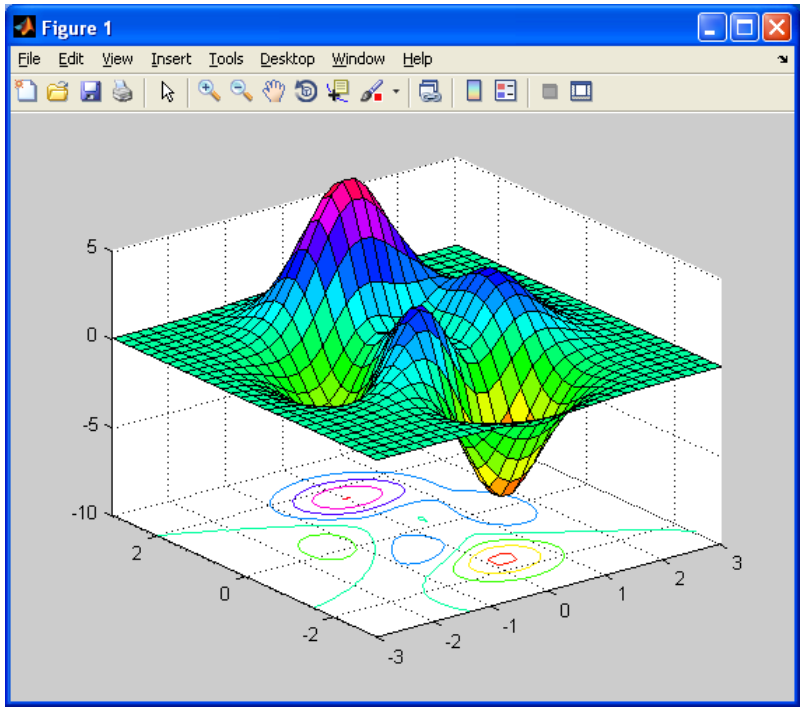
Saving MATLAB Figures

Saving figures as `.fig` files enables you to reload them later without having to regenerate all the data and decorations.



Saving MATLAB Figures

Saving figures as `.fig` files enables you to reload them later without having to regenerate all the data and decorations.



You can also save figures in standard graphics formats, like `.png`, `.jpg`, etc. (`.png` is recommended.)

Practice

Practice 1.13

Create variables $x = 3$, $y =$ `[10 : -1 : 1]`, and $z =$ `'robertFrost'`. Save these to a file named `foo.mat` in the current folder. Look up the help on the `clear` command, and use it to clear all your workspace variables. Now if you double-click the file `foo.mat` from MATLAB's current folder window, what happens?

Practice 1.14

Make an accurate plot of $y = \tan(x)/x$ for $x \in [-2\pi, 2\pi]$. Save your plot to a figure file named `tan.fig`. Kill the existing plot window, then double click on `tan.fig` in the current folder window. What happens?