

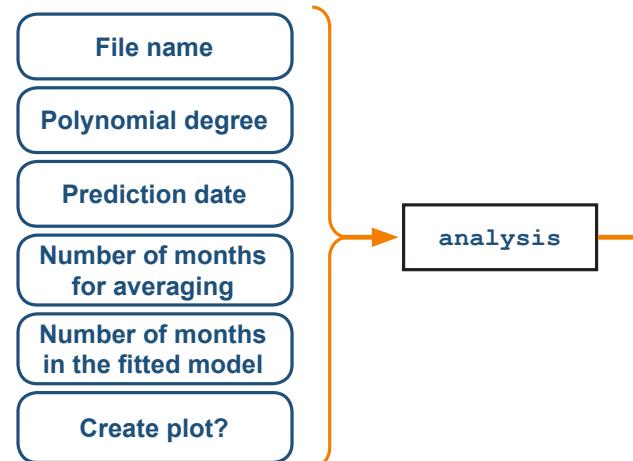
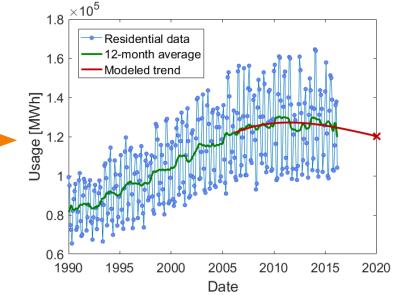
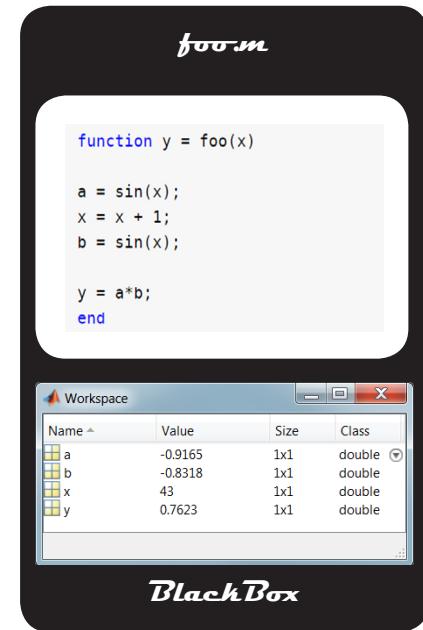
# Increasing Automation with Functions

MATLAB® Fundamentals for Aerospace Applications

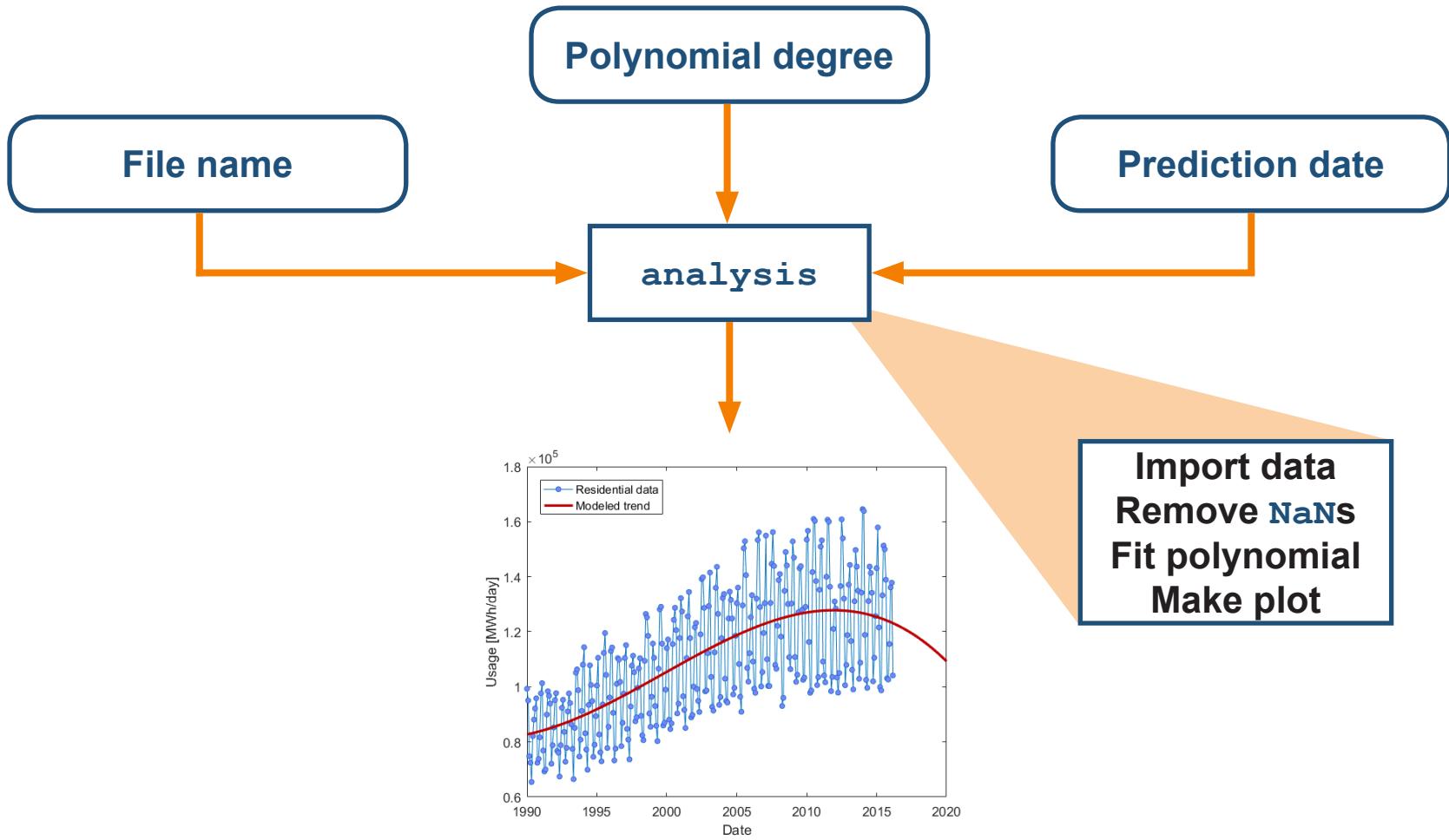


# Outline

- Creating and calling functions
- Workspaces
- Plain text code files
- Path and precedence
- Debugging
- Using structures

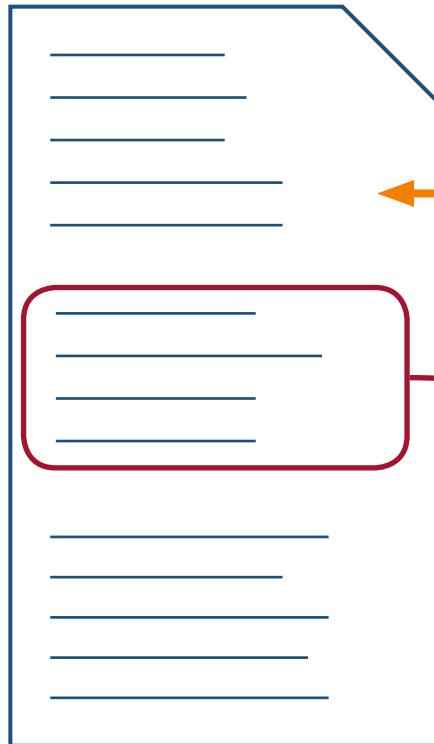


# Course Example: Electricity Modeling

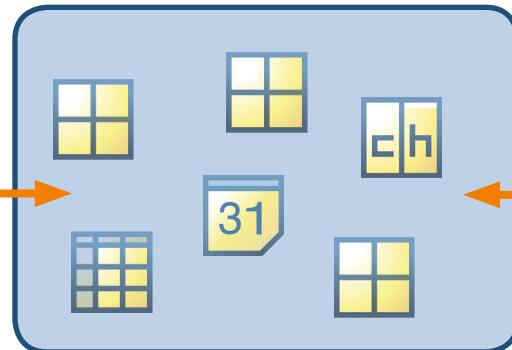


# Why Use Functions?

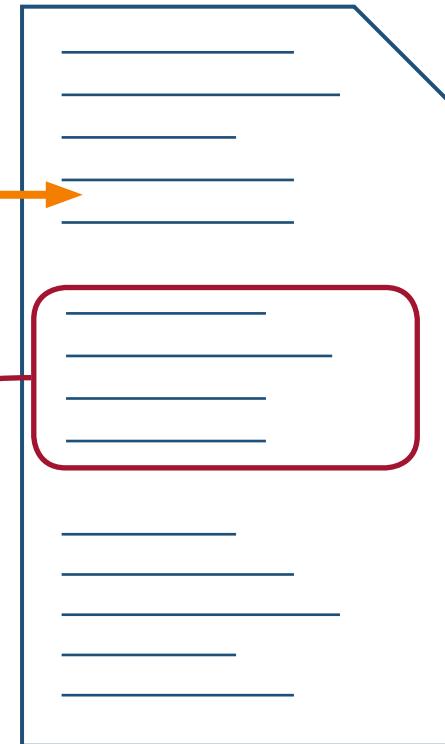
`script1 mlx`



**Workspace**



`script2 mlx`



**Managing code  
Managing variables**

# Creating a Function

```
datafile = 'elec_res.xlsx';
polydegree = 3;
predictdate = datetime(2020,1,1);
...
data = readtable(datafile);
...
c = polyfit(elapsedYears,usage,polydegree);
...
endDuration = years(predictdate - dates(1));
...
predictedusage = usageFit(end);
```

**Function declaration:**  
`function outputs = funcname(inputs)`

```
function predictedusage = analysis_func(datafile,polydegree,predictdate)
...
data = readtable(datafile);
...
c = polyfit(elapsedYears,usage,polydegree);
...
endDuration = years(predictdate - dates(1));
...
predictedusage = usageFit(end);
```

# Calling a Function

```
enddate = datetime(2018,9,1);
com2018 = polyPrediction('elec_com.xlsx',2,enddate)
```

1.2598e+05

```
function predictedusage = polyPrediction(datafile,polydegree,predictdate)
...
data = readtable(datafile);
...
c = polyfit(elapsedYears,usage,polydegree);
...
endDuration = years(predictdate - dates(1));
...
predictedusage = usageFit(end);
```

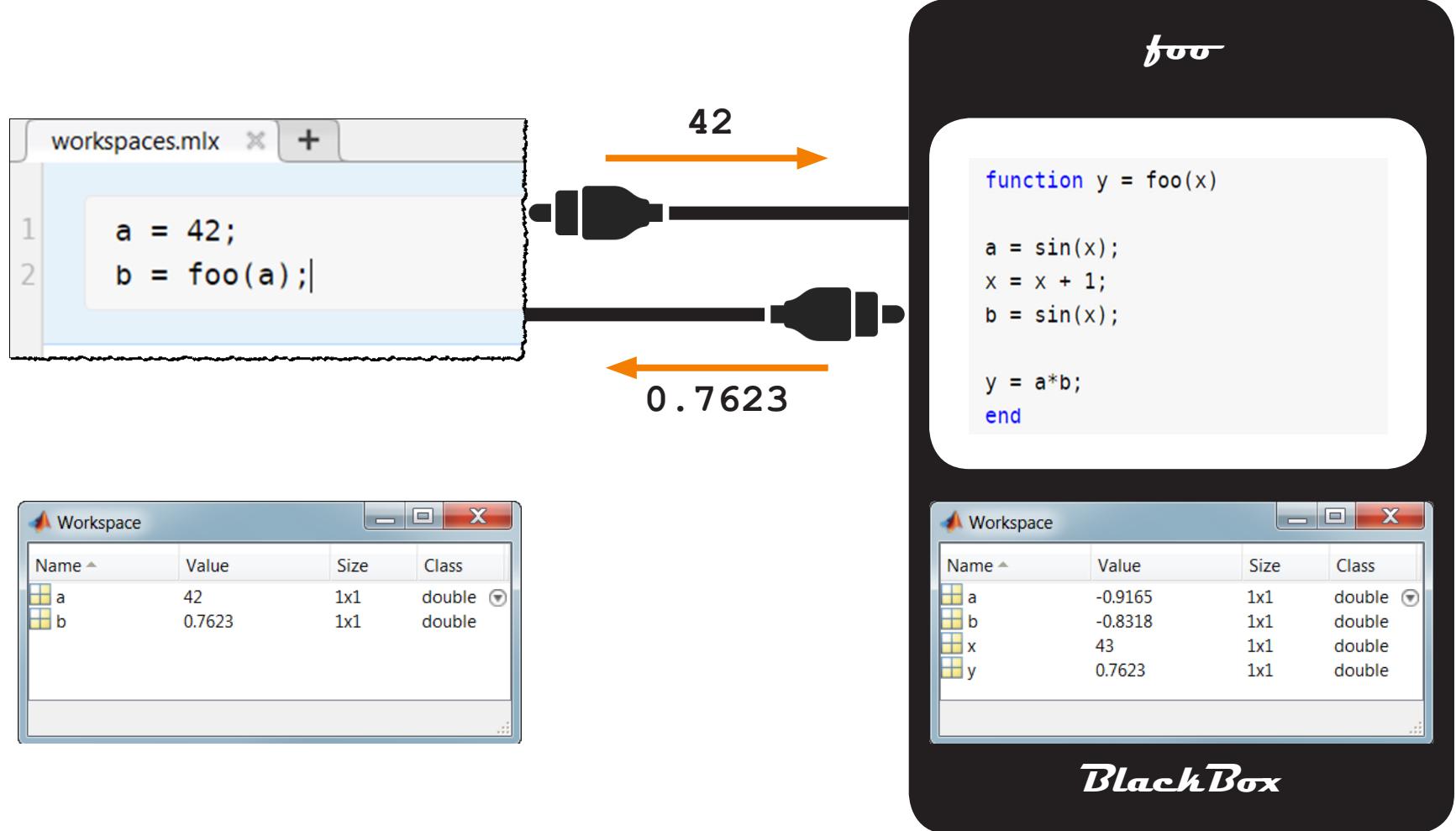
'elec\_com.xlsx'

2

01-Sep-2018

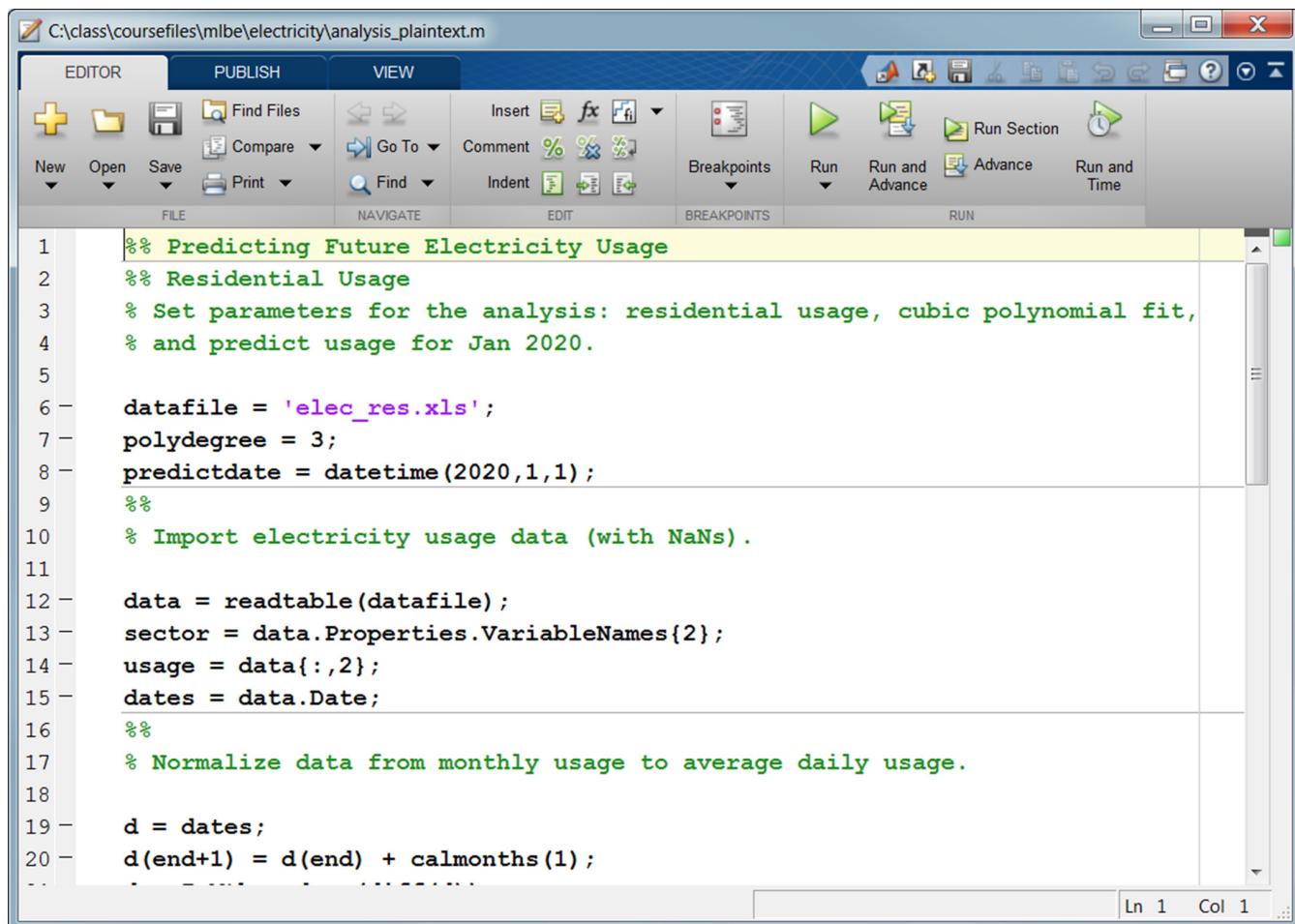
1.2598e+05

# Workspaces



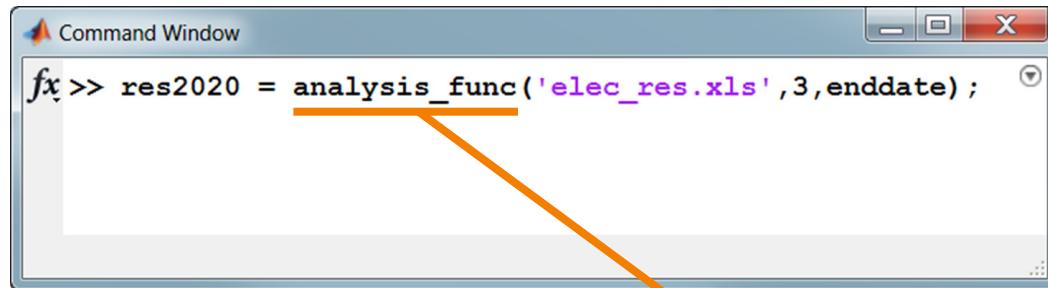
# The MATLAB® Editor

Plain Text  
Code File  
(.m extension)



```
%>> %% Predicting Future Electricity Usage
%>> %% Residential Usage
%>> % Set parameters for the analysis: residential usage, cubic polynomial fit,
%>> % and predict usage for Jan 2020.
%>>
%>> datafile = 'elec_res.xls';
%>> polydegree = 3;
%>> predictdate = datetime(2020,1,1);
%>> %%
%>> % Import electricity usage data (with NaNs).
%>>
%>> data = readtable(datafile);
%>> sector = data.Properties.VariableNames{2};
%>> usage = data{: ,2};
%>> dates = data.Date;
%>> %%
%>> % Normalize data from monthly usage to average daily usage.
%>>
%>> d = dates;
%>> d(end+1) = d(end) + calmonths(1);
%>> %
```

# Creating a Function File



A screenshot of the MATLAB Editor window. The file is named "analysis\_func.m". The code defines a function `predictedusage = analysis_func(datafile, polydegree, predictdate)`. The function performs data import, normalization, and calculation. A red circle highlights the function name "analysis\_func" in the title bar and the first line of the code. Another red circle highlights the documentation string at the top of the function.

```
function predictedusage = analysis_func(datafile,polydegree,predictdate)
% ANALYSIS_FUNC  Performs analysis on electricity usage data
%
% Import data (with NaNs)
data = readtable(datafile);
sector = data.Properties.VariableNames{2};
usage = data{:,sector};
dates = data.Date;

% Normalize data from monthly usage to average daily usage
d = dates;
d(end+1) = d(end) + calmonth(1);
```

# Calling Precedence

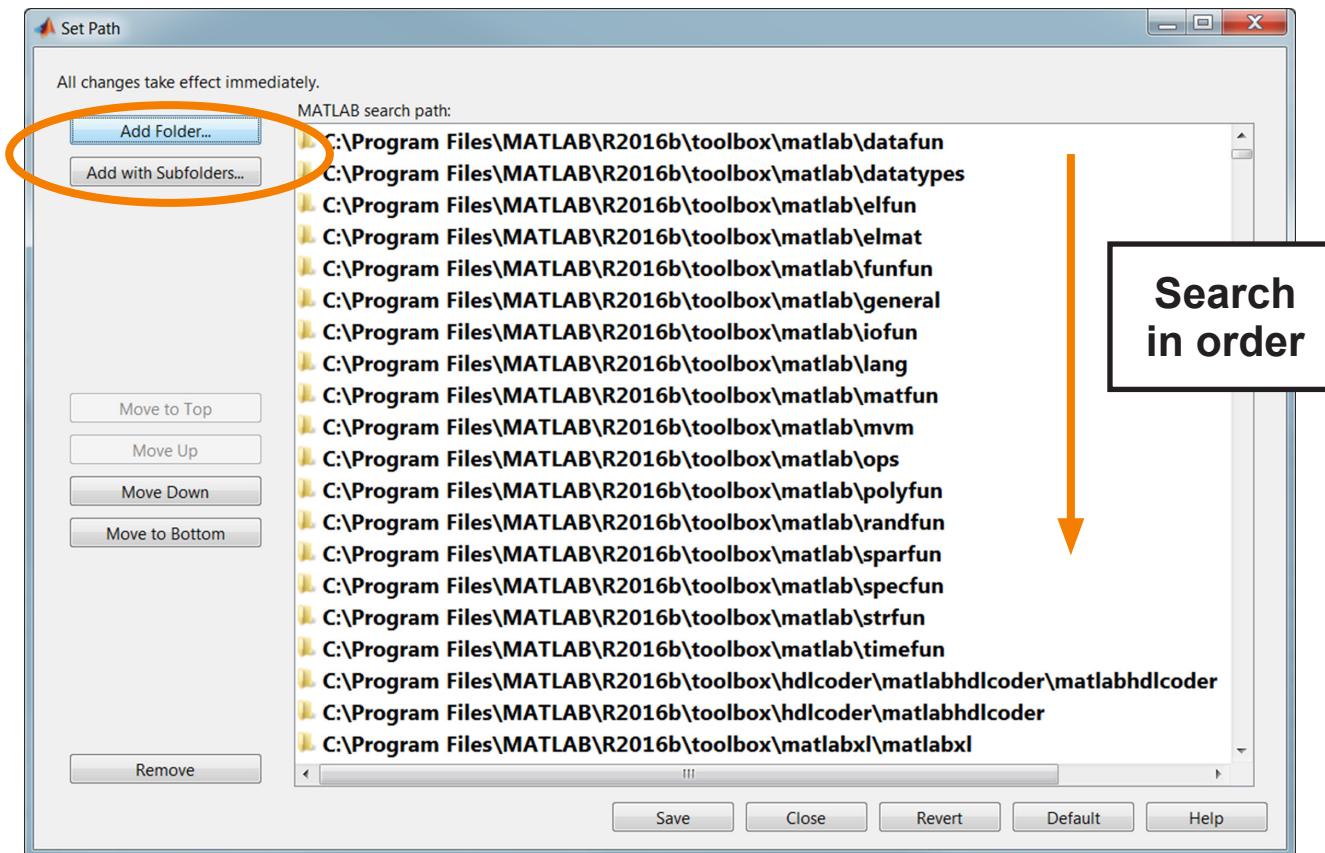
`>> analysis`

1. Variable
2. Imported package function
3. Nested function
4. Local function
5. Private function
6. Class method
7. Class constructor
8. File in current directory
9. File on the path



1. Built-in files
2. MATLAB executable (MEX) files
3. Simulink® models
4. MATLAB live scripts
5. Obfuscated MATLAB code files
6. MATLAB code files

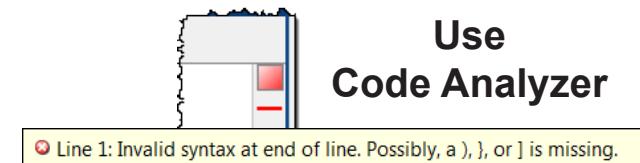
# The MATLAB® Path



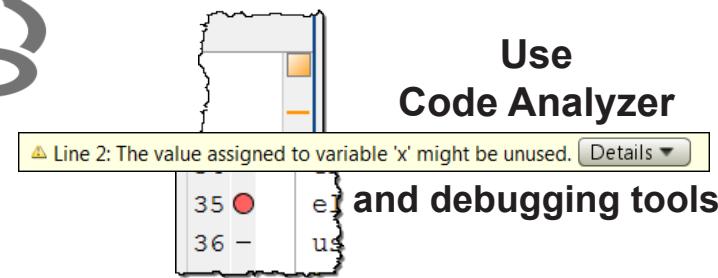
Search  
in order

# Debugging

## Syntax error



## Run-time error



# Using Breakpoints



```
>> res2020 = analysis_debug('elec_res.xlsx',3,enddate)
Error using plot
Vectors must be the same length.
```

```
Error in analysis_debug (line 44)
plot(datesFit,usageFit,'Color',[0.75 0 0],'LineWidth',2)
```

```
C:\class\coursefiles\mlbe\electricity\analysis_debug.m
EDITOR PUBLISH VIEW
FILE NAVIGATE EDIT BREAKPOINTS RUN
27 elapsedYears = years(dates - dates(1));
28 c = polyfit(elapsedYears,usage,polydegree);
29
30 endDuration = years(predictdate - dates(1));
31 elYrFit = linspace(0,endDuration,501);
32 usageFit = polyval(c,elYrFit);
33 predictedusage = usageFit(end);
34
35 % Display the predicted usage.
36 predictdate.Format = 'MMM yyyy';
37 disp(['Predicted ',sector,' usage in ',char(predictdate),...
38 ' is ',num2str(predictedusage),' MWh/day'])
39
40 % Plot the data and model together.
41 plot(dates,usage,'o-','MarkerSize',4,'MarkerFaceColor',[0.5 0.5 1])
42 hold on
43 datesFit = dates(1) + years(elapsedYears);
44 ● plot(datesFit,usageFit,'Color',[0.75 0 0],'LineWidth',2)
45 hold off
46 xlabel('Date')
47 ylabel('Usage [MWh/day]')
48 legend([sector ' data','Modeled trend','Location','NW']
49
50
51
```

# Examining Values

The screenshot shows the MATLAB IDE interface. On the left is the Editor window containing a script named `analysis_debug.m`. The code performs several operations, including fitting a trend model to usage data and plotting the results. A red circle highlights the variable `datesFit` in the code. An orange arrow points from this highlighted variable to the Workspace browser on the right, which lists all variables and their properties. Three specific variables are circled in orange: `datesFit`, `predictedusage`, and `usageFit`. The `datesFit` variable is a `315x1 datetime` array. The `predictedusage` and `usageFit` variables are `1x501 double` arrays.

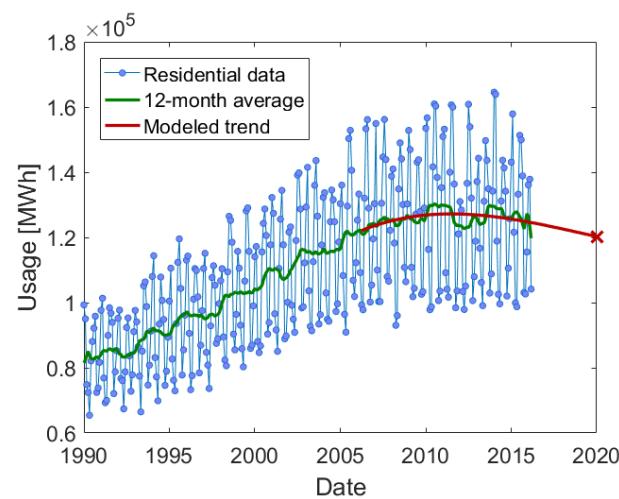
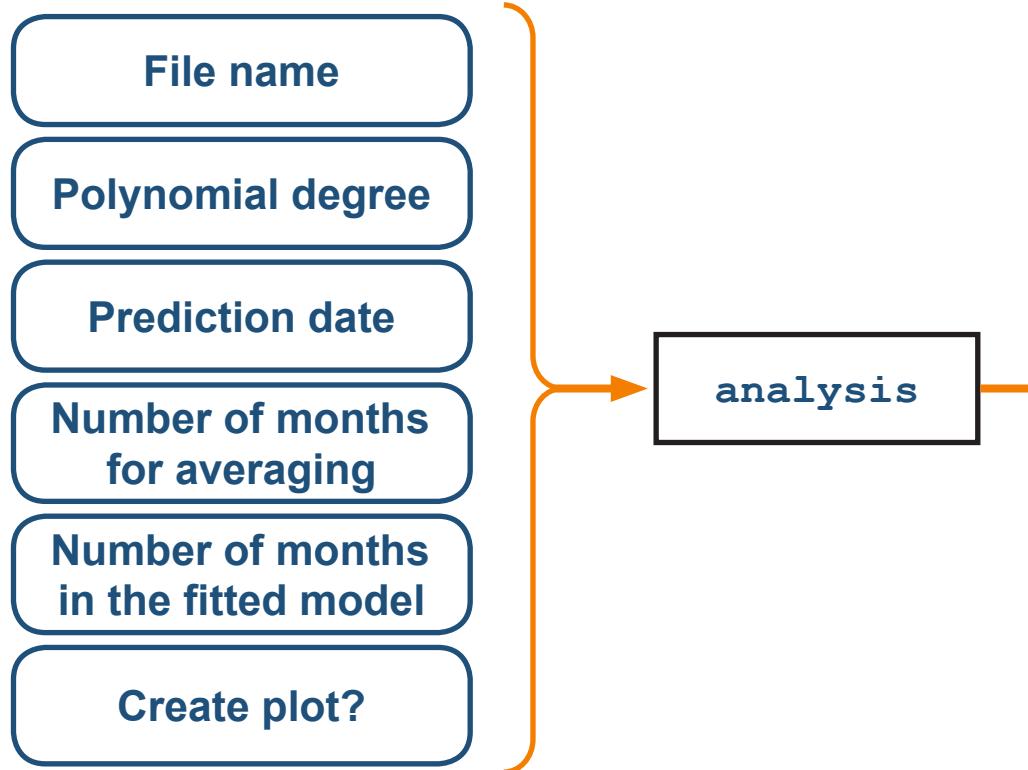
Name	Value	Size	Class
c	[ -6.3468, 184.4... ]	1x4	double
d	316x1 datetime	316x1	datetime
data	315x2 table	315x2	table
datafile	'elec_res.xls'	1x12	char
dates	315x1 datetime	315x1	datetime
datesFit	315x1 datetime	315x1	datetime
daysInMth	315x1 double	315x1	double
elapsedYears	315x1 double	315x1	double
elYrFit	1x501 double	1x501	double
endDuration	29.9992	1x1	double
polydegree	3	1x1	double
predictdate	1x1 datetime	1x1	datetime
predictedusage	1.0927e+05	1x1	double
sector	'Residential'	1x11	char
usage	315x1 double	315x1	double
usageFit	1x501 double	1x501	double

# Ending Debugging

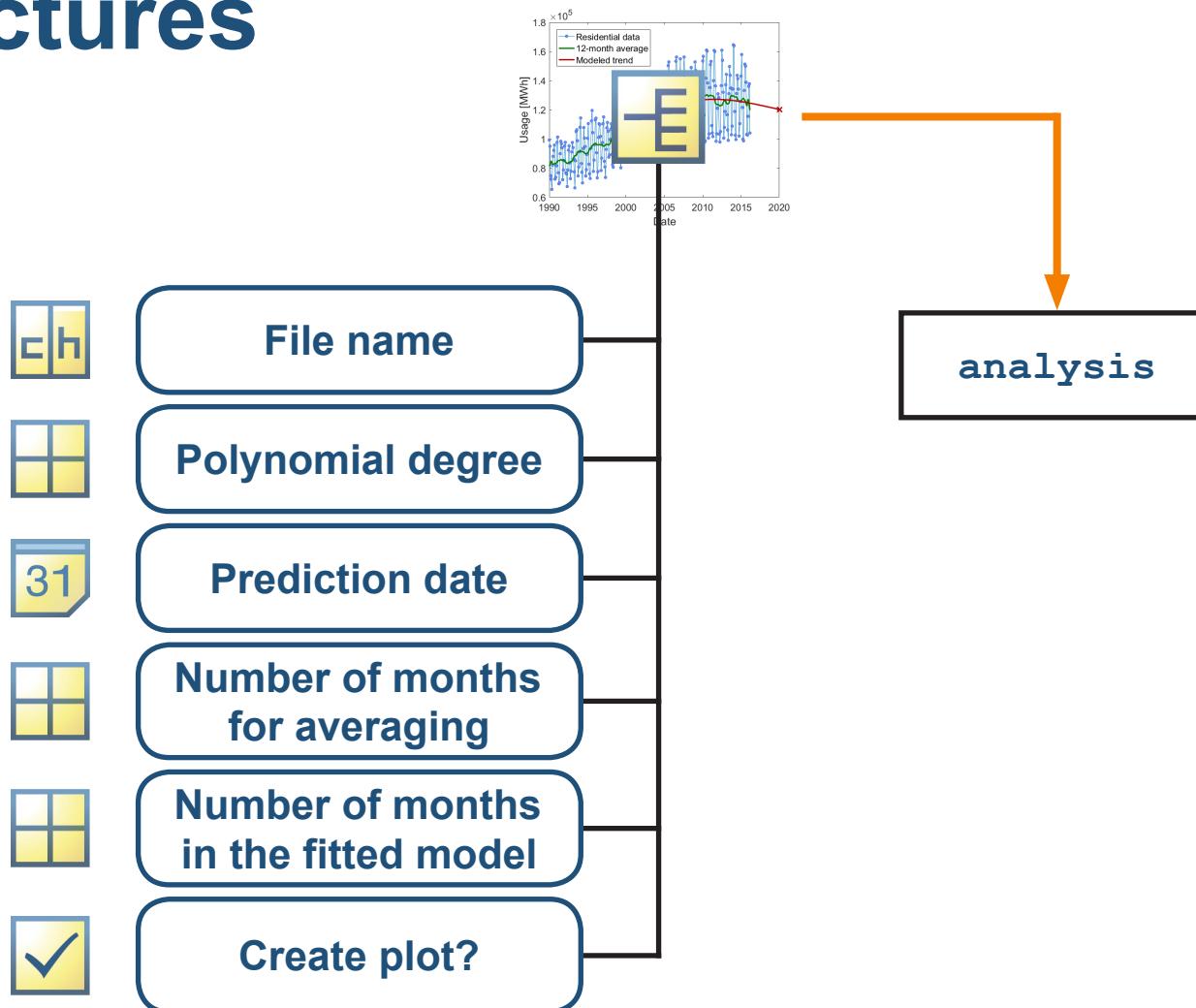
- Exit debug mode
- Correct errors
- Save changes
- Test and confirm
- Clear or disable breakpoints



# Course Example: Adding Model Parameters

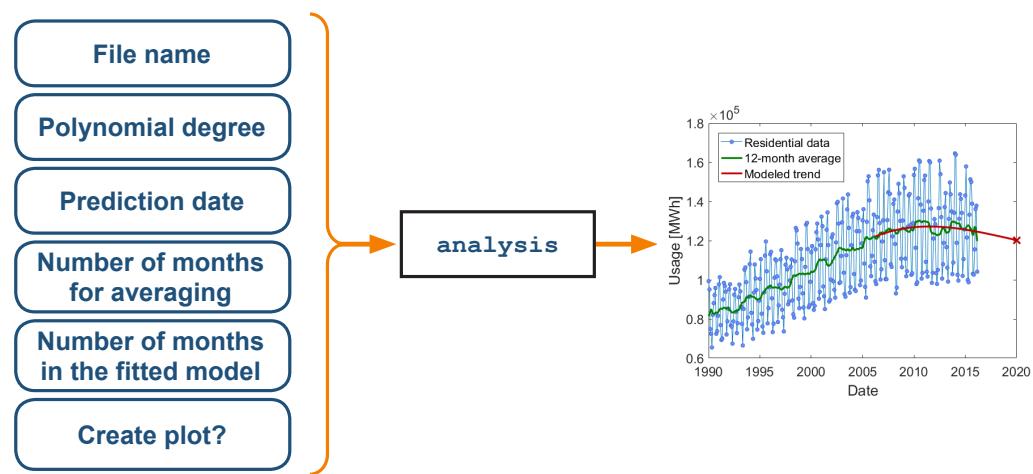
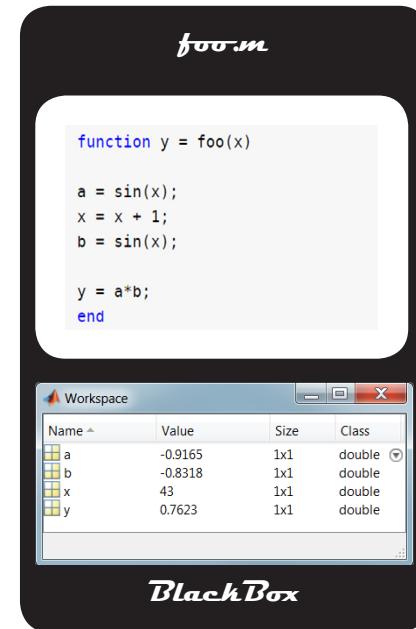


# Combining Heterogeneous Data with Structures



# Summary

- Creating and calling functions
- Workspaces
- Plain text code files
- Path and precedence
- Debugging
- Using structures



# Test Your Knowledge

1. Suppose the workspace contains a vector `x` and the function `foo` creates a scalar variable also called `x`. What will happen to the vector `x` if you issue the command `z = foo(7)`?
  - A. It will change to the scalar value defined in `foo`.
  - B. Nothing, because the two variables are of different dimensions (i.e., a vector cannot be changed to a scalar).
  - C. Nothing, because the vector `x` is not passed as an argument to `foo`.
  - D. Nothing, because `foo` maintains its own workspace, so there is no name conflict.

# Test Your Knowledge

2. Which of the following is a valid function declaration?
- A. `function [x,y] = foo(a,b)`
  - B. `function foo(a,b) = [x,y]`
  - C. `[x,y] = function foo(a,b)`
  - D. `[x,y] = foo(a,b)`

# Test Your Knowledge

3. (Select all that apply) Which of the following are valid calls to a function with the declaration line  
**function [x,y] = foo(a,b)** ?

- A. [x,y] = foo(a)
- B. x = foo(a,b)
- C. [x,y] = foo[a,b]
- D. (x,y) = foo(a)
- E. [x,y] = foo(a,b)