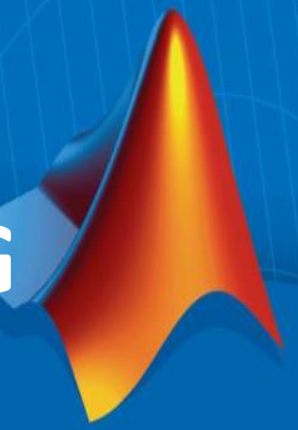
TeraSoft
www.terasoft.com.tw

MATLAB PROGRAMMING TECHNIQUES



Tim Yeh

support@terasoft.com.tw

MathWorks®
Accelerating the pace of engineering and science

Course Outline

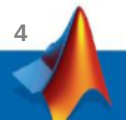
- Writing Functions
- Structuring Code
- Creating Robust Applications
- Troubleshooting Code and Improving Performance

Advanced MATLAB® Programming Techniques

Writing Functions

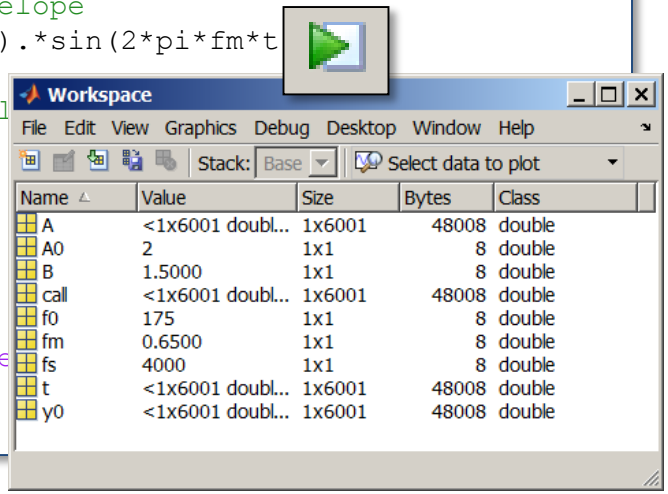
Section Outline

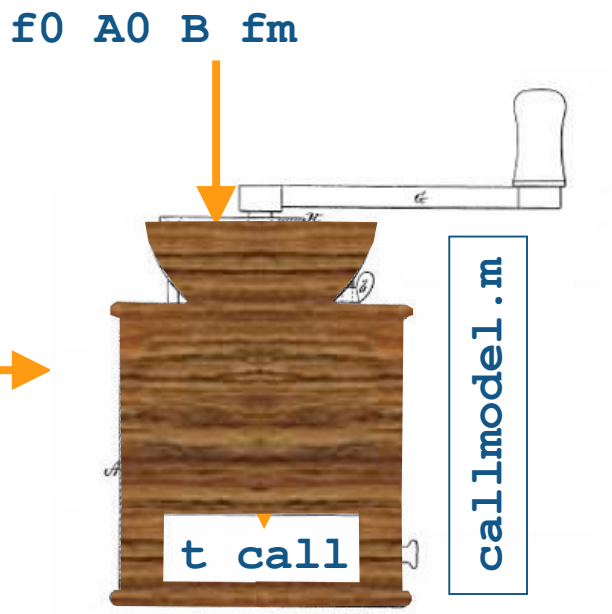
- Creating functions
- Calling functions
- Workspaces
- Path and precedence



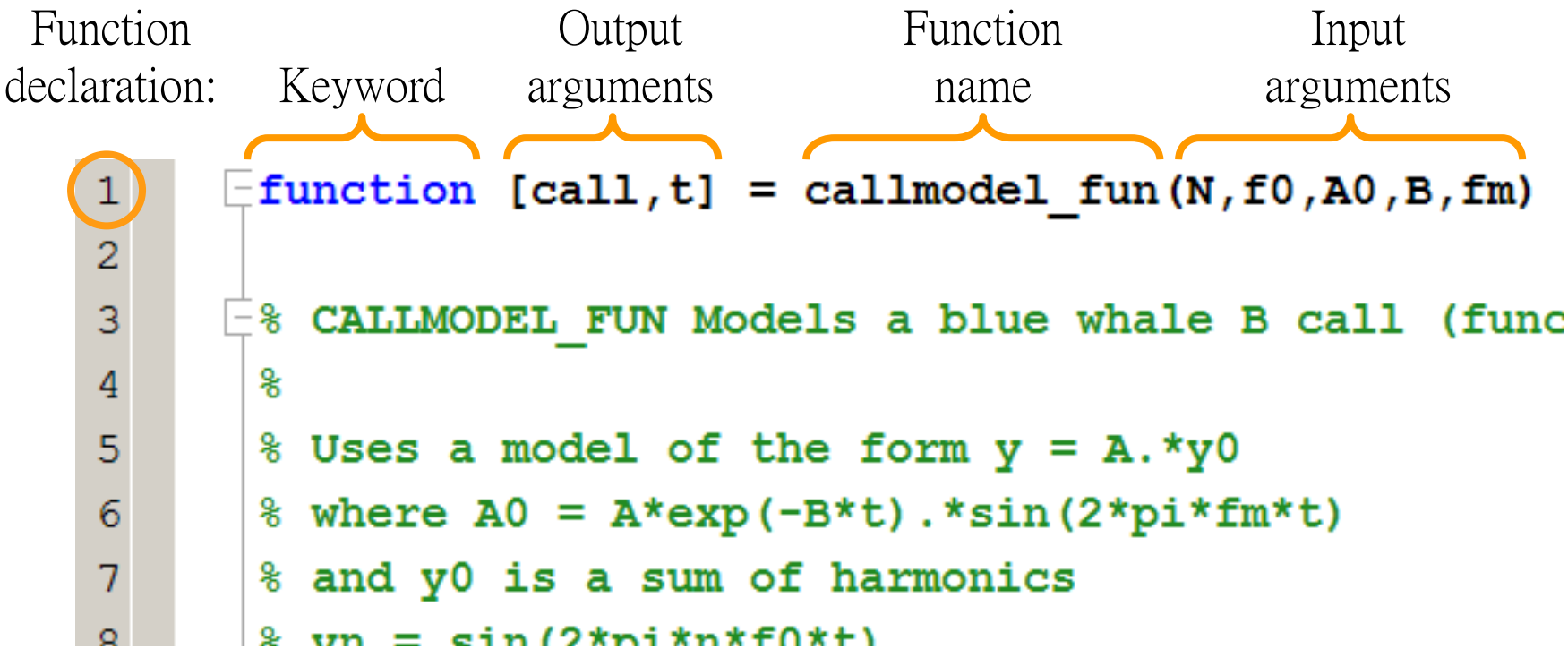
Increasing Automation

```
% Create the time base for the signal.
fs = 4000;
t = 0:(1/fs):1.5;

% Set the fundamental frequency of the call.
f0 = 175;

% Create the harmonics.
y0 = sin(2*pi*f0*t) + ...
    sin(2*pi*2*f0*t) + sin(2*pi*3*f0*t);

% Set the additional parameters in the model.
A0 = 2; % Initial amplitude.
B  = 1.5; % Amplitude decay rate.
fm = 0.65; % Frequency of the modulating envelope.
% Create the envelope
A  = A0*exp(-B*t).*sin(2*pi*fm*t

% Create the call
call = A.*y0;

% Plot the model
figure
plot(t,call)
xlabel('Time')
ylabel('Amplitude
title('{\bf Blue
soundsc(call,fs)
```



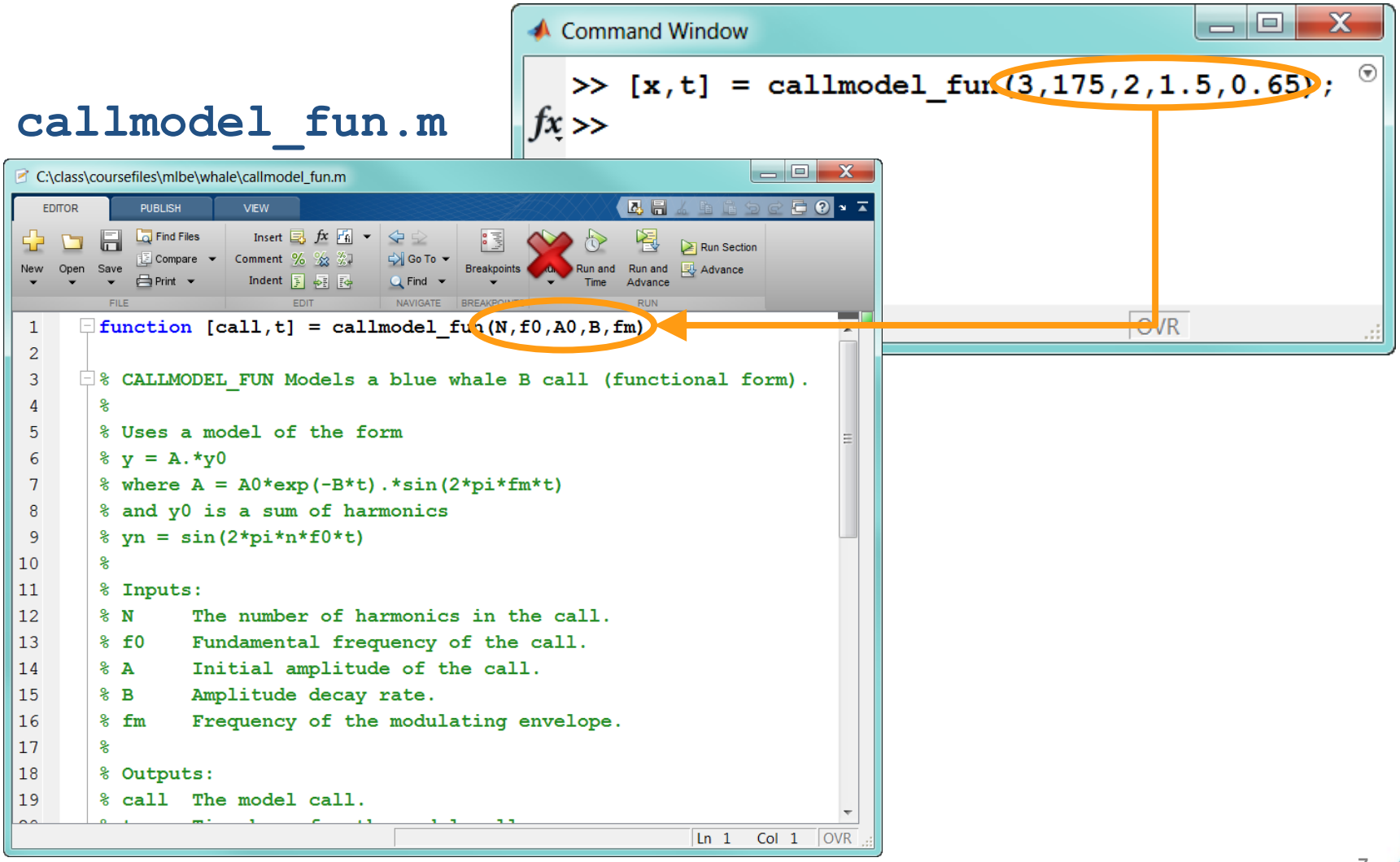


Creating a Function

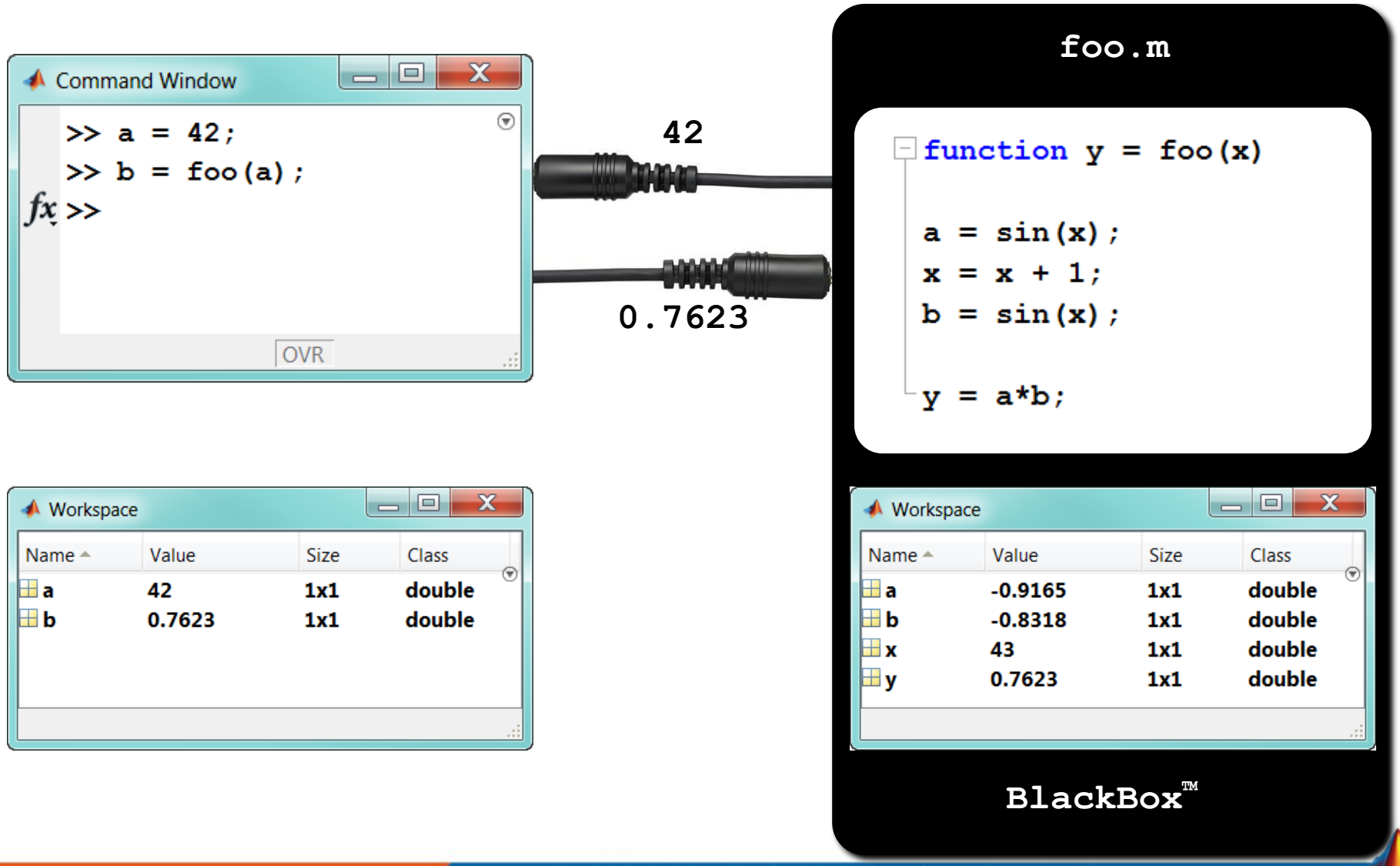


Calling a Function

callmodel_fun.m



Workspaces



Calling Precedence

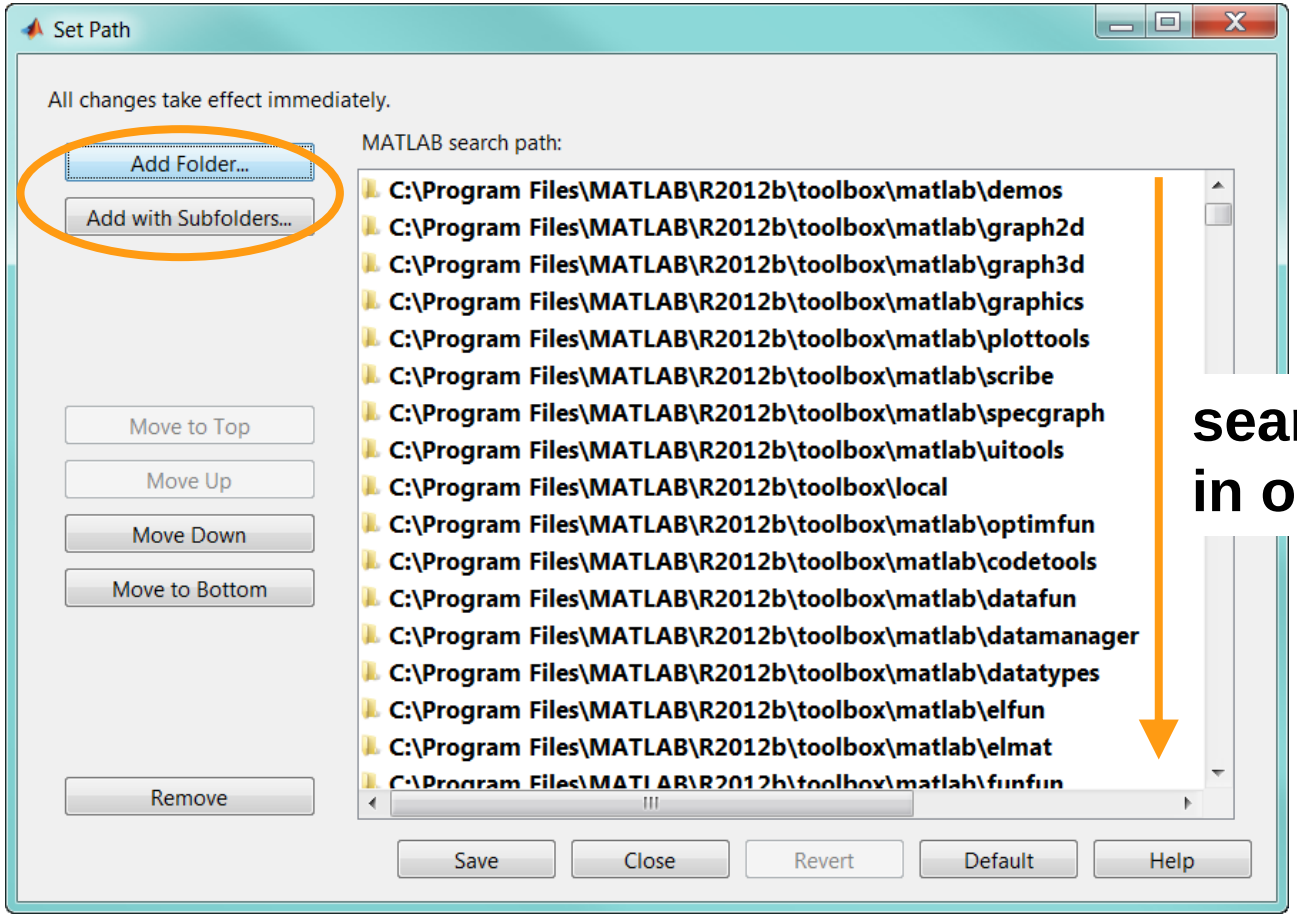
```
>> whale
```

1. Variable
2. Nested function
3. Subfunction
4. Private function
5. Class constructor
6. Overloaded method
7. File in the current directory
8. File on the path

1. whale.bi
2. whale.mexw32
3. whale.mdl
4. whale.p
5. whale.m

The MATLAB® Path

```
>> pathtool
```

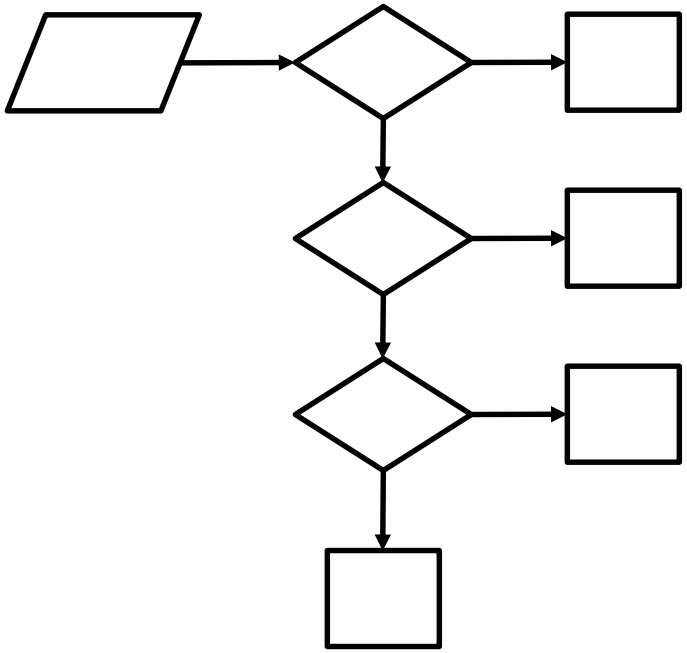


Advanced MATLAB® Programming Techniques

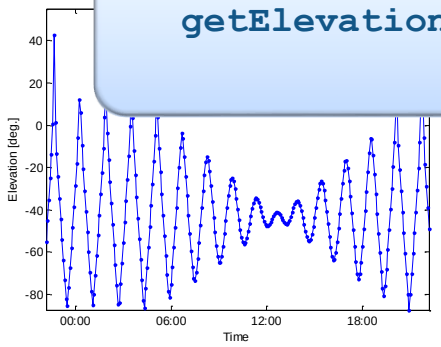
Structuring Code

Section Outline

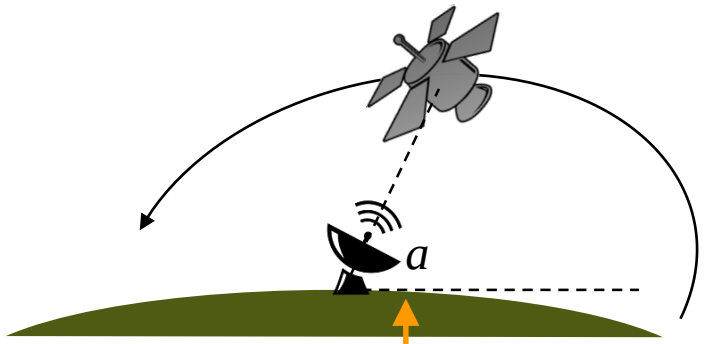
- Private functions
- Subfunctions
- Nested functions
- Function handles
- Anonymous functions
- Precedence rules
- Comparison of function types



Course Example: Satellite Tracking



`getElevations`



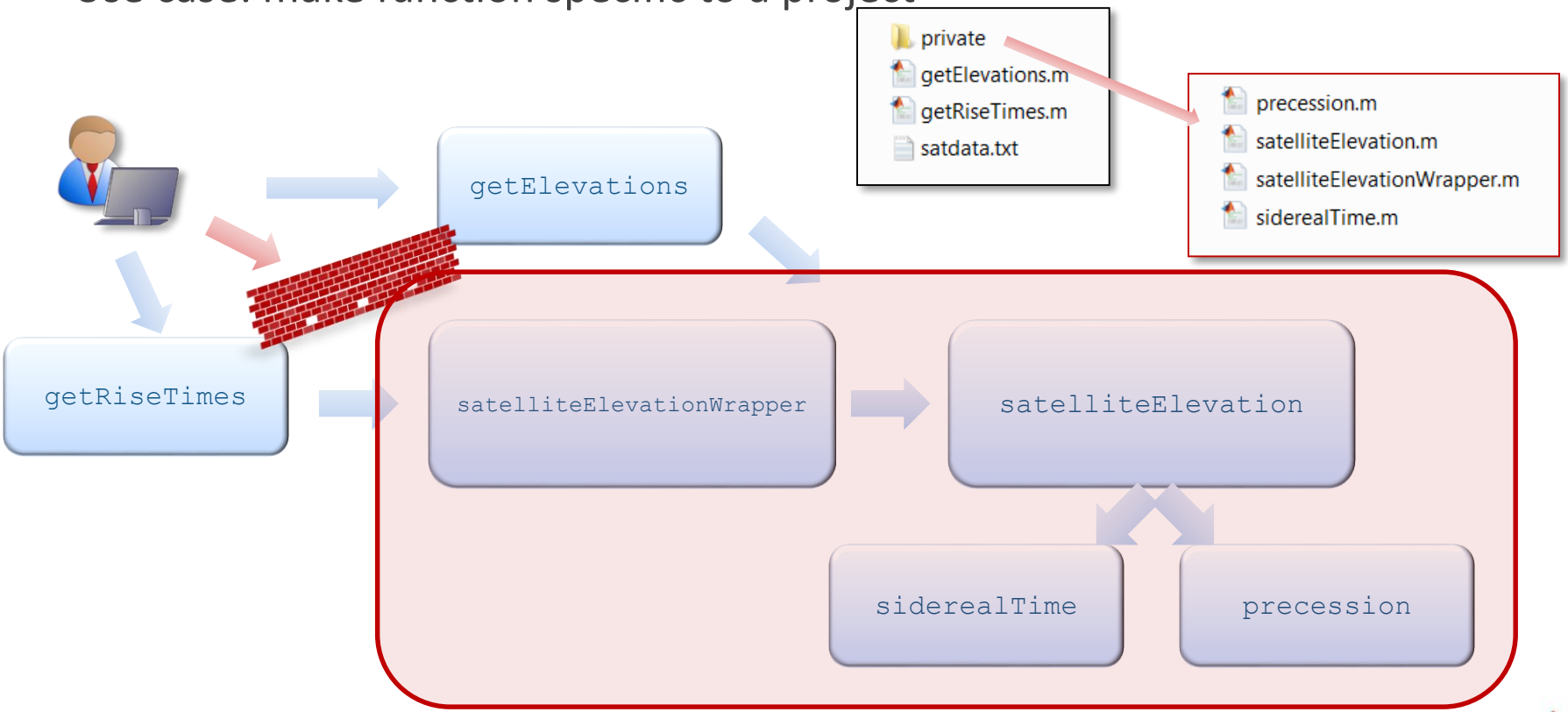
`satelliteElevation`

`siderealTime`

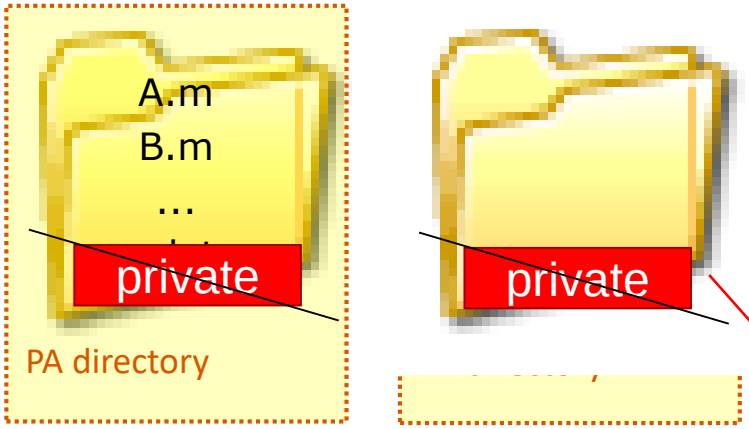
`precession`

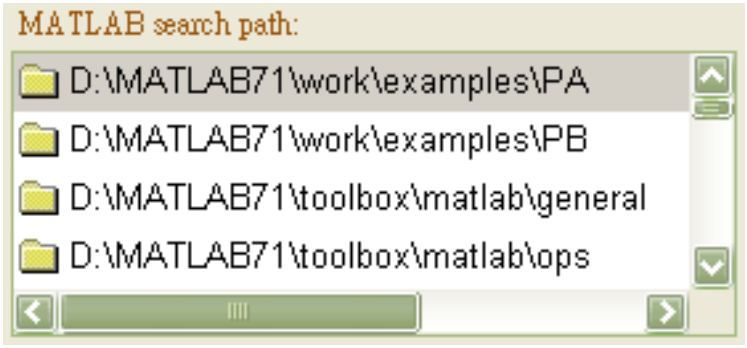
Private Functions

- Function files in a folder named private
- Accessible only from within this and the parent folder
- Use case: make function specific to a project



Private Functions





Accessed for parent directory only

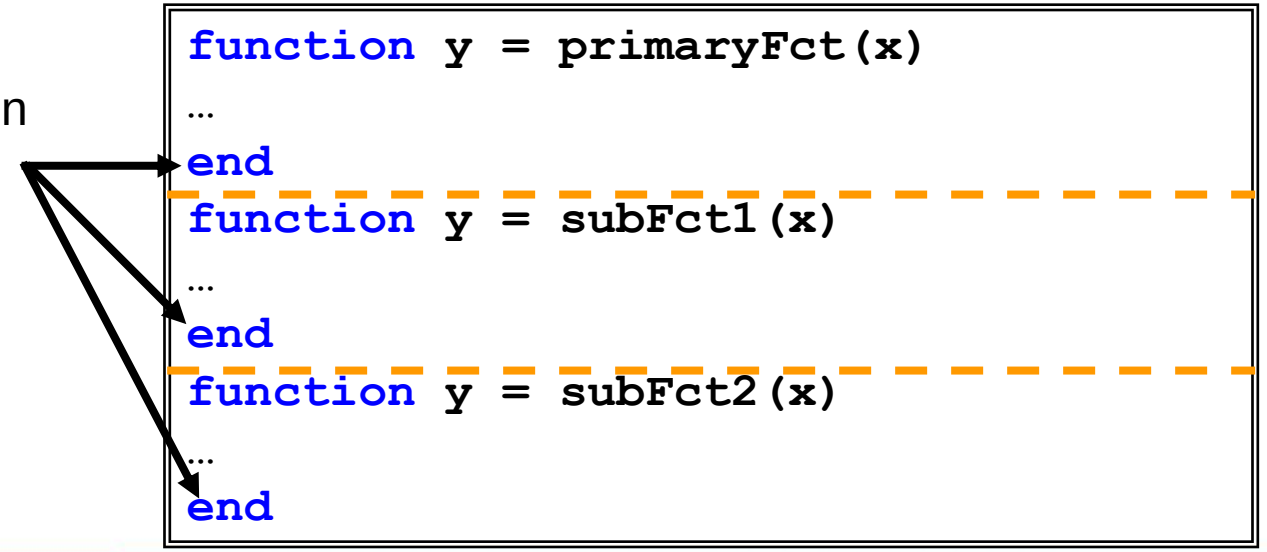
```
>> cd([matlabroot '/work/examples'])
>> edit AA.m
```

```
function AA(x)
C(x);
```

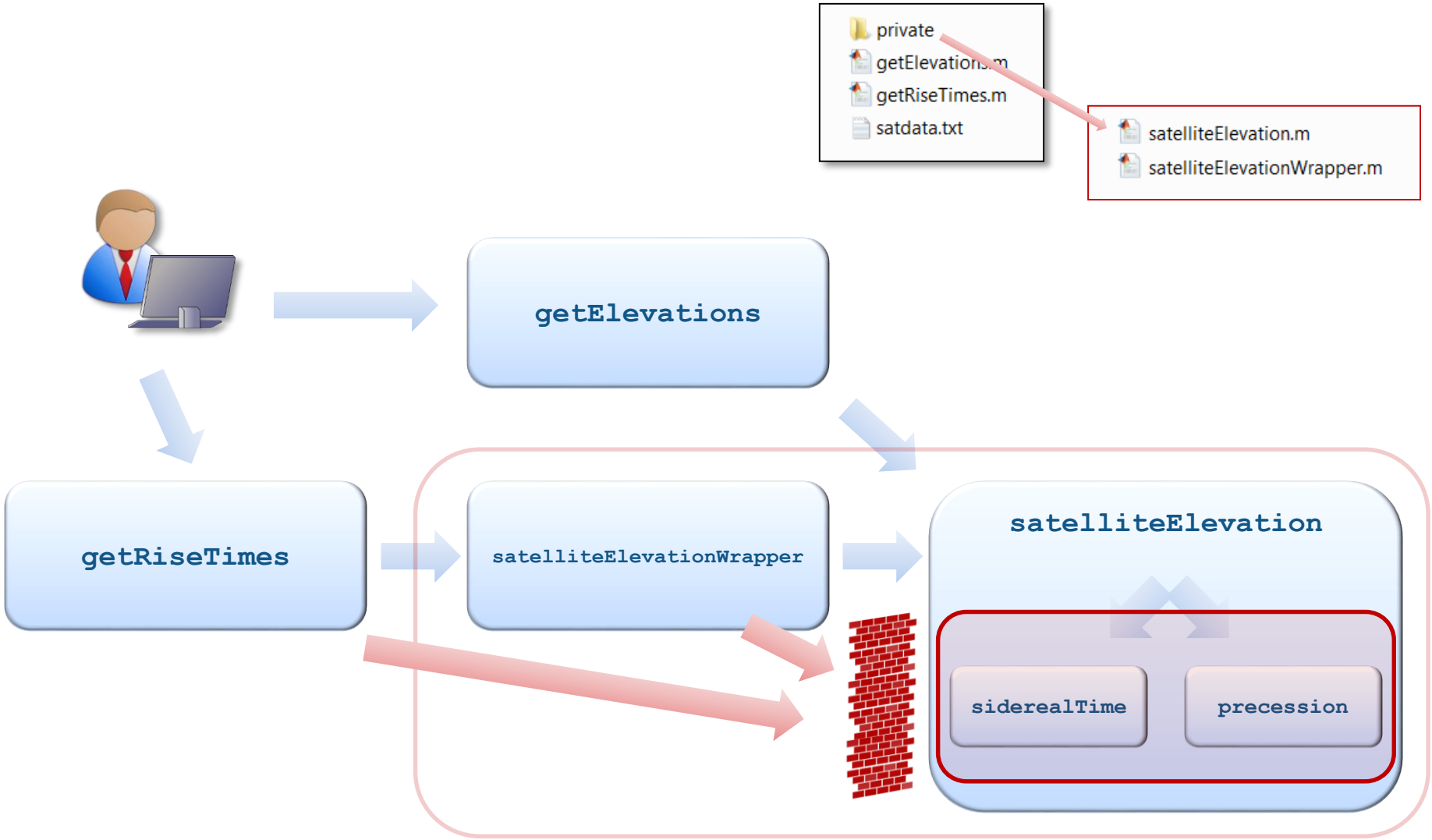
Subfunctions

- Several functions in one file
- Keyword `function` used as delimiter
- First function accessible from outside world
- Others accessible only from within the same file
- Use case: hide internal utility functions

Optional when using only subfunctions



Subfunctions

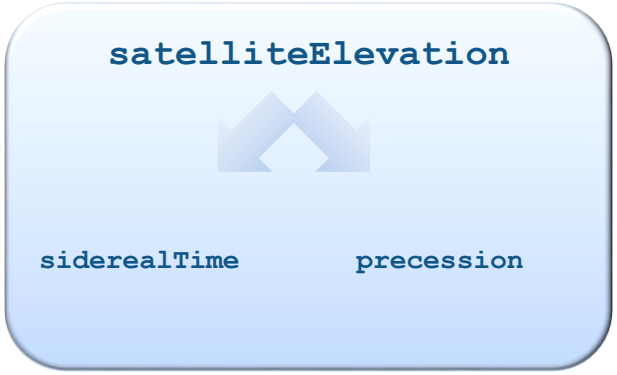


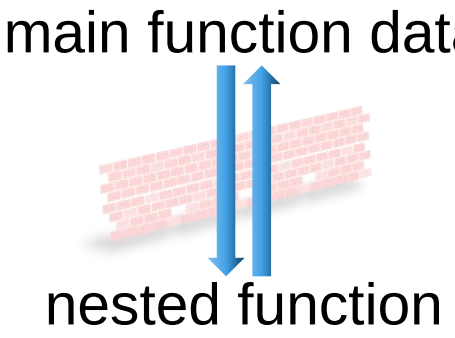
Nested Functions

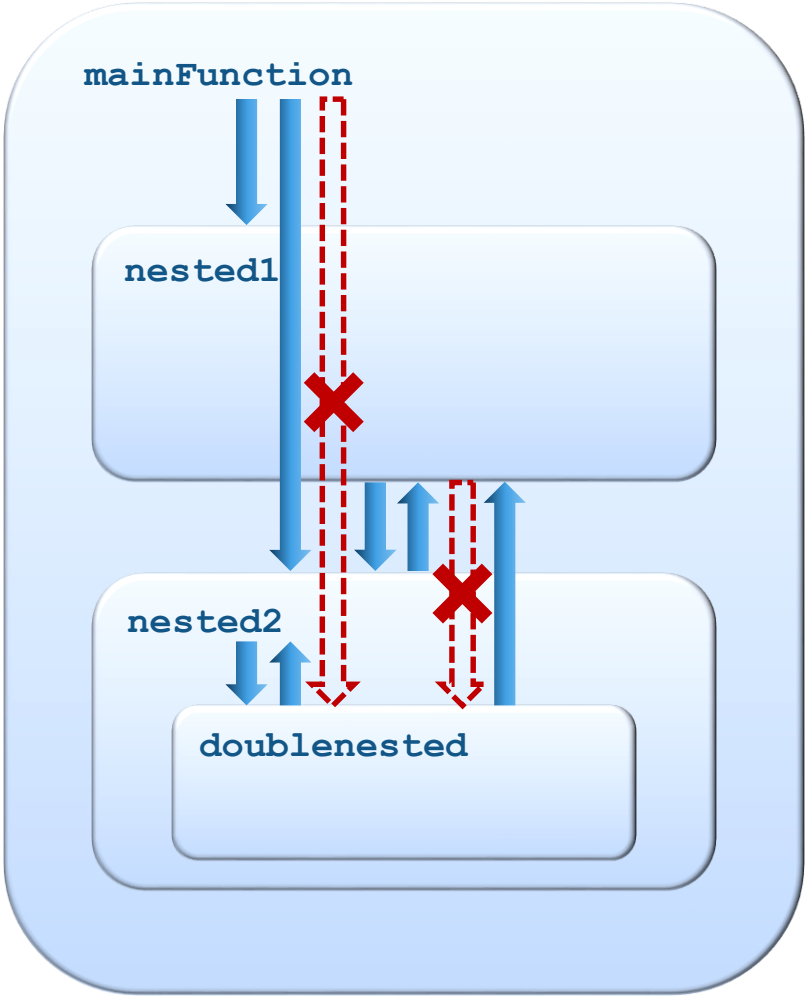
- Functions nested inside other functions
- Mark their extent using `function` and `end`
- Can be called
  - From level immediately above
  - From function at same level within same parent function
  - From a nested function at any lower level
- Access to superior function workspaces
- Have their own workspace

```
function y = outerFct(x)
  ...
    function y = innerFct(x)
      ...
    end
end
```

Nested Functions







Example: Nested Functions

```
function T = tax(income)
adjusted_income = max(income - 6000, 0);
T = compute_tax;

    function t = compute_tax
        t = 0.28*adjusted_income;
    end
end
```

```
>> edit tax
```


Scope of a Variable

Using Subfunction

```
function [A,B] = sub_scope(x,y)

A = subfun1(x);
B = subfun2(y);

function v = subfun1(u)
v = rand(u,1);

function v = subfun2(u)
v = randn(u,1);
```

separate workspaces

Using Nested Function

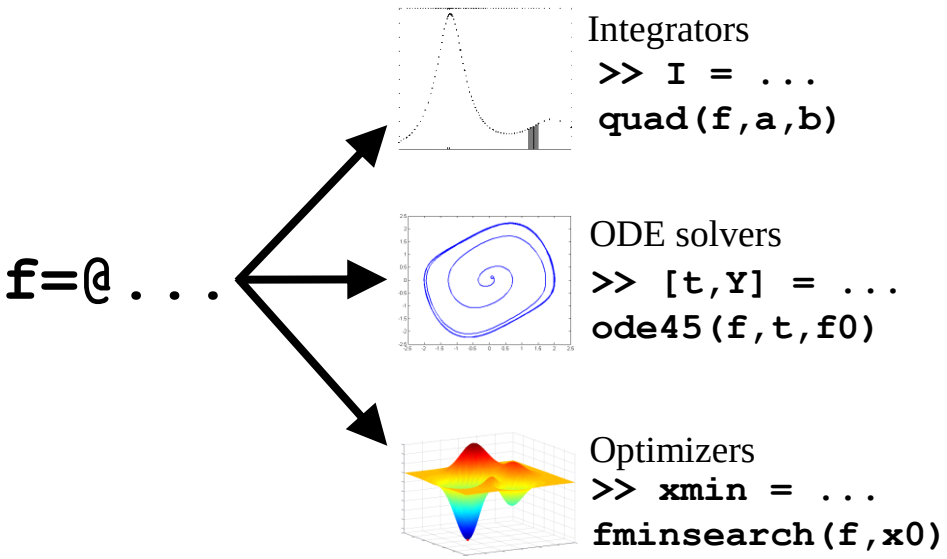
```
function T = tax(income)

adj_income = ...
max(income - 6000, 0);
T = compute_tax;

    function t = compute_tax
        t = 0.28*adj_income;
    end
end
```

shared workspaces

```
>> edit sub_scope
```


Function Handles

- Special MATLAB data type
- Create a variable for calling a function.
- Use case 1: Flexible/dynamic function calls
- Use case 2: Extending the visibility of a function
- Use case 3: Changing the function interface



Creating and Using Function Handles

- Syntax

```
fhandle = @functionname
```

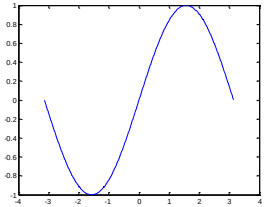
- Example

create `fhandle = @sin;`

use `fhandle(arg1, arg2, ...);`

```
function plot_fhandle(fhandle, data)
plot(data, fhandle(data));
```

```
>> plot_fhandle(@sin, -pi:0.01:pi)
```



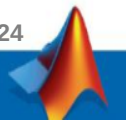
Example:
access to subfunction using function handle

main function

sub-function

```
function myFhandle = myFunction(myInput)
SomeOtherValue = 7;
myFhandle = @mySubFunction;

disp(['7 times ' num2str(myInput) ' is ' ...
    num2str(myFhandle(myInput, SomeOtherValue))]);

function myReturnValue = mySubFunction(x, y)
myReturnValue = x.*y;
```

```
>> myT
>> myTimes(8,2)
```

Anonymous Functions

- Wrapper to slightly change the function (interface)
- Write @ followed by list of arguments and function call
- No function name
- No file necessary

```
>> f = @myfun;
```

f

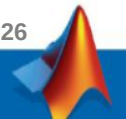
```
function y = myfun(a,b,c)
y = a*(b-sin(c));
```

```
>> f = @(a,b,c) a*(b-sin(c));
```

f

Comparison of Function Types

Aspect	Private	Sub	Nested	Anonymous
File	Yes	Yes	Yes	No
Workspace	Separate	Separate	Shared	Depends
Access	Files in `private` and parent folder	Within file	See page 2-6	Via function handle variable
Typical use	Project specific functionality	Hide utilities	Share application data	Change of function interface

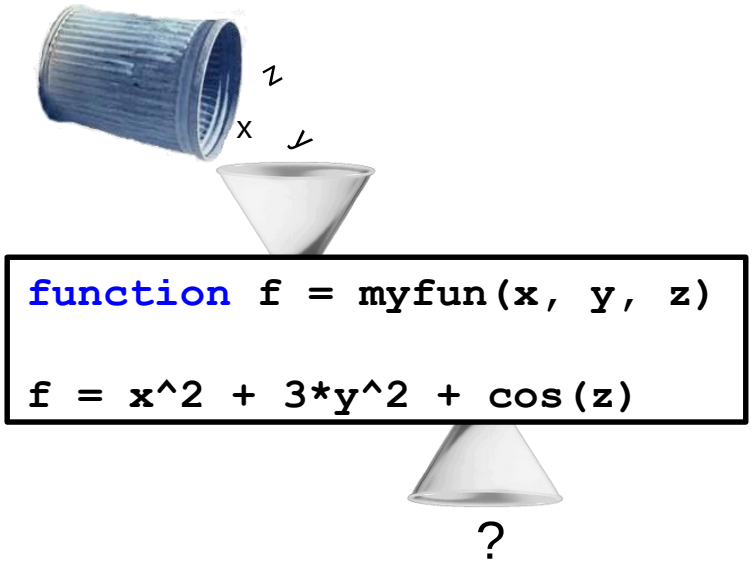


Advanced MATLAB®
Programming Techniques

Creating Robust Applications

Section Outline

- Creating flexible function interfaces
- Checking for warning and error conditions
- Working with the `try-catch` construct and `Mexception` objects



Setting Default Function Inputs

```
function [alt,t] = getElevations(tspan,dt,doplot)
```

```
>> [alt,t] = getElevations([t0,t0+1],5/1440,true);
```

✓

```
>> [alt,t] = getElevations([t0,t0+1],5/1440);
Error using getElevations (line 32)
Not enough input arguments.
```

✘

```
>> [alt,t] = getElevations([t0,t0+1],5/1440);
```

[doplot = true]

Warnings and Errors

```
>> elv = getElevations(1,5/1440);
```

problem created

problem explained?

getElevations.m

```
26 -    end
27
28      % Get parameters from file
29 -    fid = fopen('satdata.txt');
30 -    params =  textscan(fid,'%s%f');
31 -    fclose(fid);
32 -    params = params{2};
33
34      % Create
35 -    t = tspan(1):dt:tspan(2);
36
37      % Determi
38 -    elv = zeros(size(t));
39 -    for k = 1:numel(t)
40 -        elv(k) = satelliteElevation(t(k),params);
41 -    end
42
43      % Plot the satellite elevation
44 -    if doplot
45 -        plot(t,elv,'.-')
```

problem encountered

```
Attempted to access tspan(2); index out of bounds because numel(tspan)=1.

Error in getElevations (line 35)
t = tspan(1):dt:tspan(2);
```

Warning and Error Messages

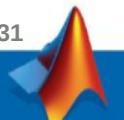
```
>> 1/0
Warning: Divide by zero.
(Type "warning off MATLAB:divideByZero" to suppress
this warning.)
ans =
   Inf
```

warning warndlg lastwarn

```
>> [1 2]*[3 4]
??? Error using ==> *
Inner matrix dimensions must agree.
```

error errordlg lasterr

>> edit warningEx



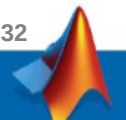
Example 6-4: Input Argument Checking

- Given the function `foo`,

```
function f = foo(x, y, z)
error(nargchk(2, 3, nargin))
% syntax
% msgstring = nargchk(minargs, maxargs, numargs)
```

```
>> foo(1)
??? Error using ==> foo
Not enough input arguments.
```

```
>> edit foo
```

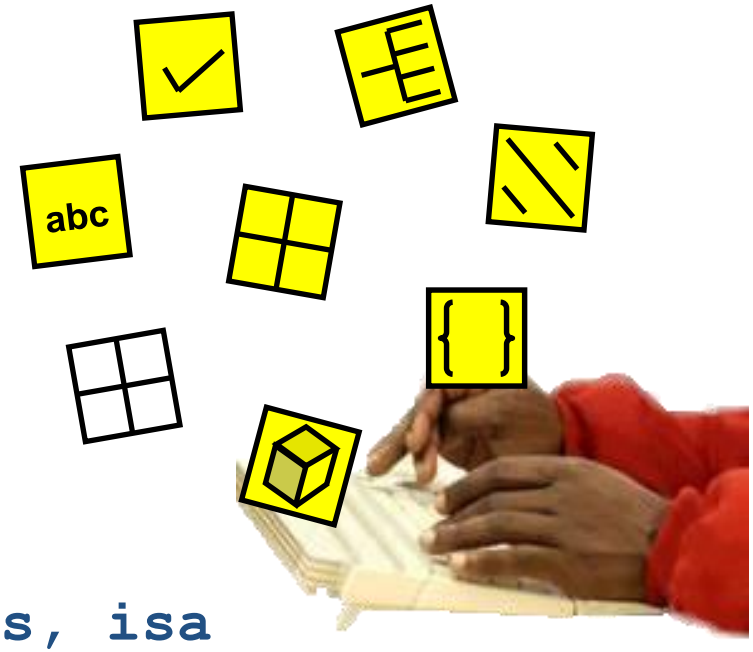



Checking for Error Conditions

```
elv = getElevations([t0,t0+1],5/1440);
```

Correct number of inputs?
Correct data type?
Correct dimension?
Within expected range?
…

Argument Checking



Did the user supply inputs of an appropriate type?

→ `class, isa`

Did the user supply an appropriate number of inputs and ask for an appropriate number of outputs?

→ `nargchk, nargoutchk`

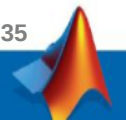
Example: Class Checking

- Class → Create object or return class of object

```
>> A=2;
>> nameStr = class(A)
nameStr =
double
```

- isa → Determine if input is object of given class

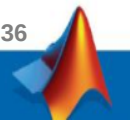
```
isa(rand(3,4),'double')
ans =
   1
```

Example: Input Argument Checking

```
function c = testarg1(a, b)
if (nargin == 1)
    c = a .^ 2;
elseif (nargin == 2)
    c = a + b;
end
```

```
>> testarg1(1)
ans =
     1
>> testarg1(1,2)
ans =
     3
```

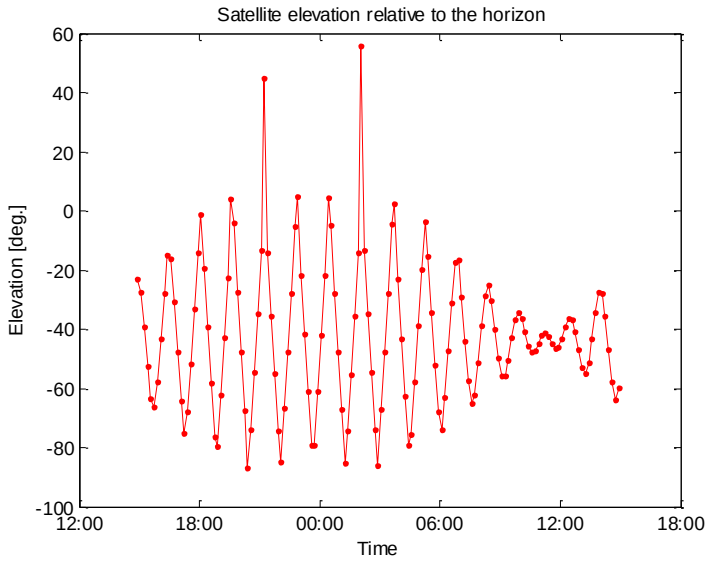



Flexible Interfaces

provide optional inputs (how many?)

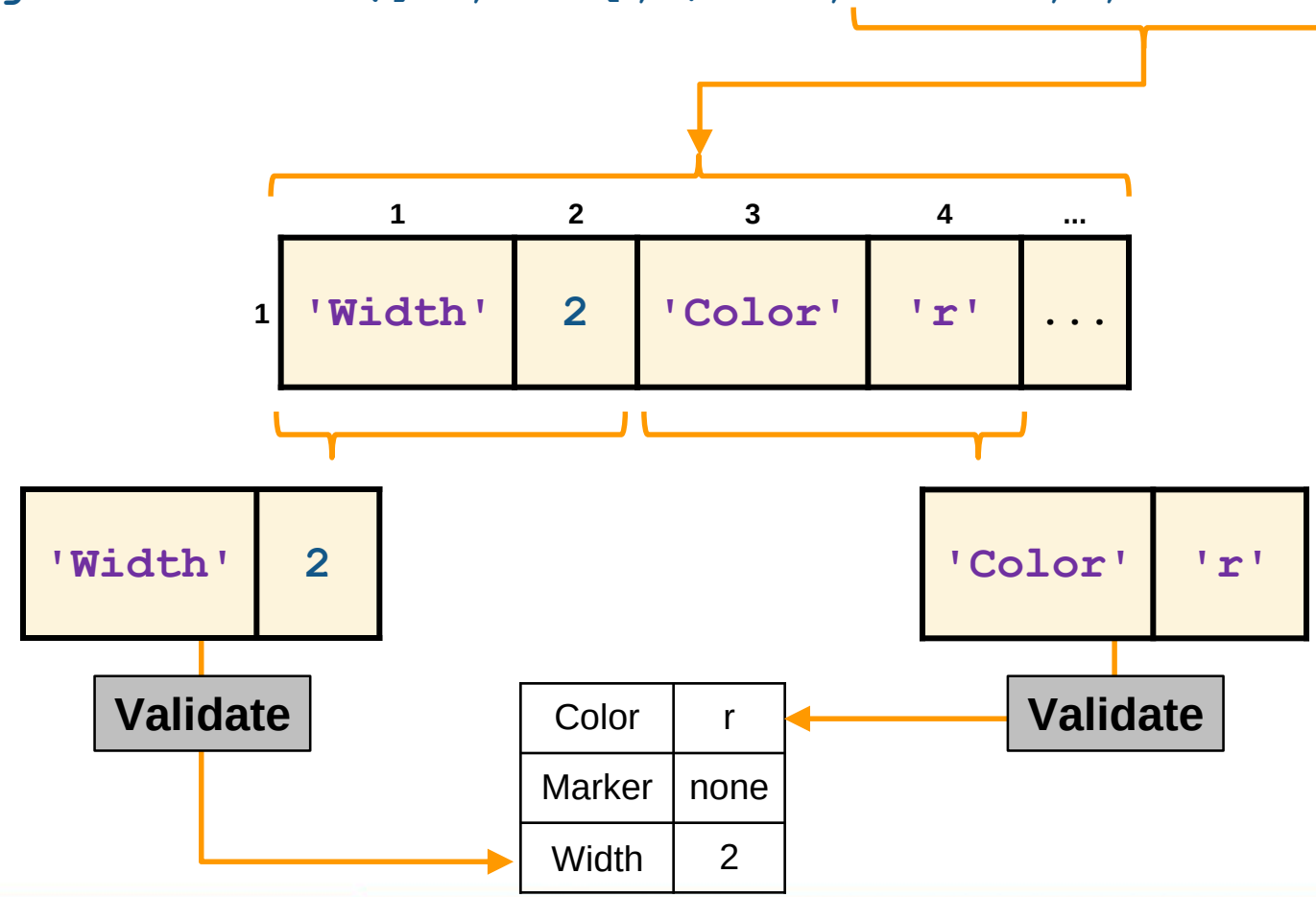
```
elv = getElevations([],10/1440,true,'Color','r');
```

use default value



Parsing Property-Value Lists

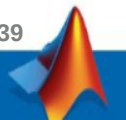
```
elv = getElevations([t0,t0+1],5/1440,'Width',2,'Color','r');
```



Variable Number of Inputs and Outputs

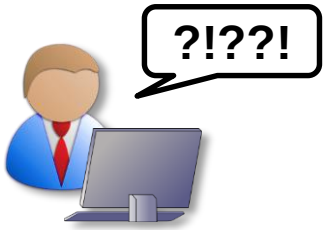
- The `varargin` and `varargout` functions let you pass any number of inputs or return any number of outputs to a function.
  - Fixed number of inputs and outputs:
    - `function [u,v] = myfun(x,y,z)`
  - Variable number of inputs and outputs:
    - `function [u,v] = myfun(varargin)`
    - `function varargout = myfun(x,y,z)`
    - `function [u,varargout] = myfun(x,y,varargin)`
    - `function varargout = myfun(varargin)`

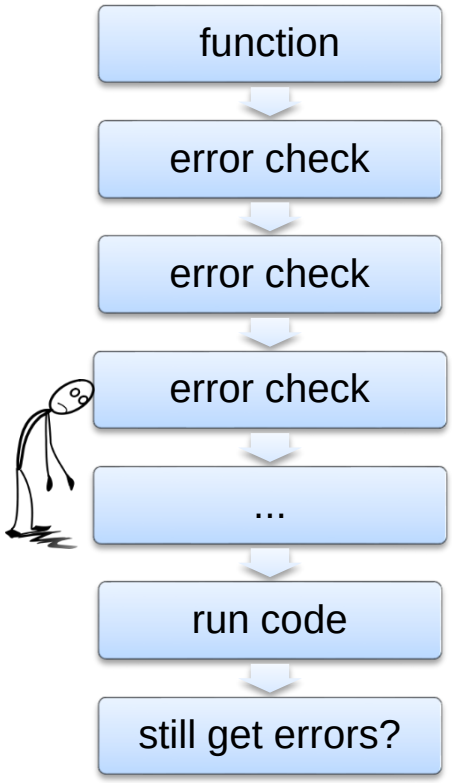
`varargin` and **`varargout`** are cell arrays

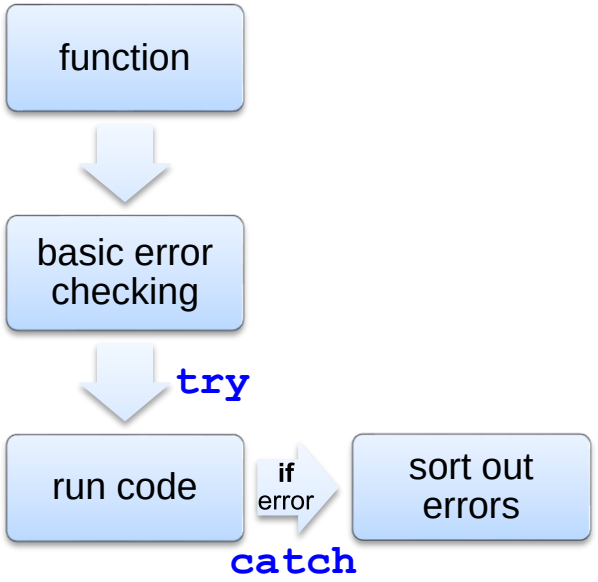


The try-catch Construct

```
>> elv = getElevations([t0,t0+1],[],true,'LineWidth','2');
Error using plot
Value must be numeric.
```

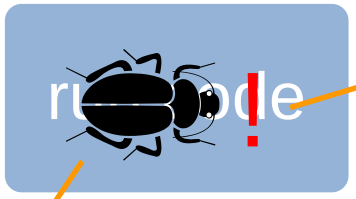






The MException Object

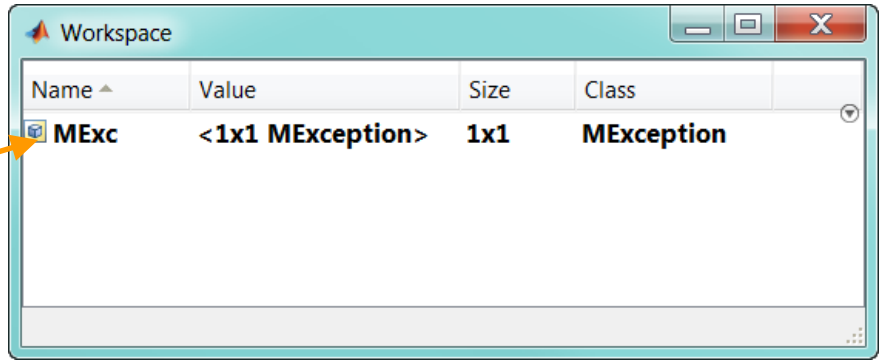
```
function
try
```

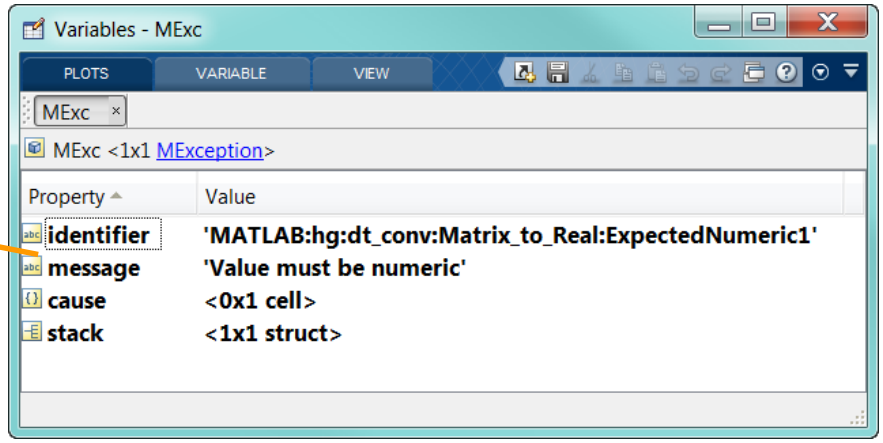



```
catch MExc
```

sort out errors

```
end
```





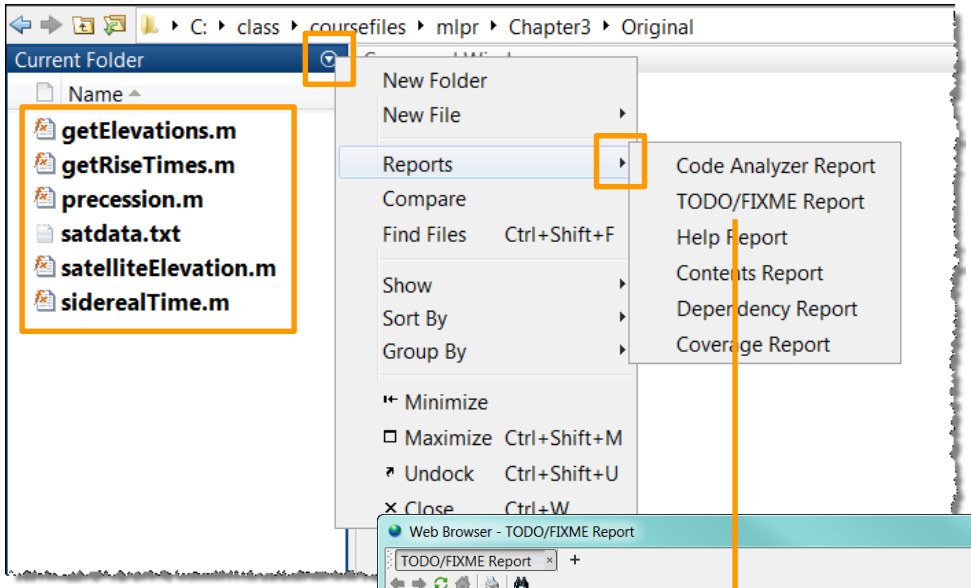
Advanced MATLAB®
Programming Techniques

Troubleshooting Code and Improving Performance

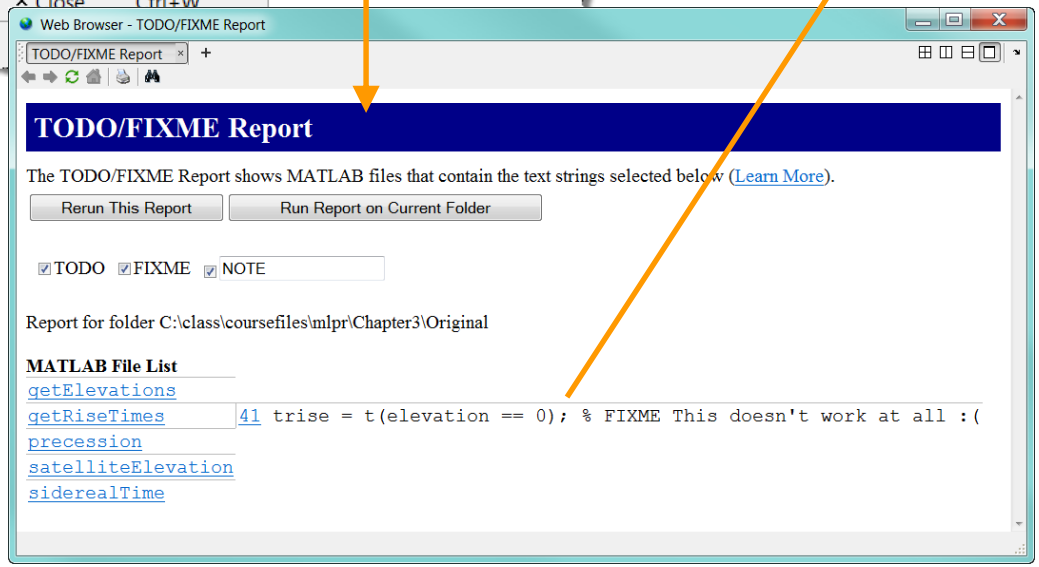
Section Outline

- Generating reports on multiple files
- Finding potential problems in code
- Debugging code
- Assessing code performance

Directory Tools




```
% FIXME  This doesn't work at all :(
trise = t(elevation == 0);
```

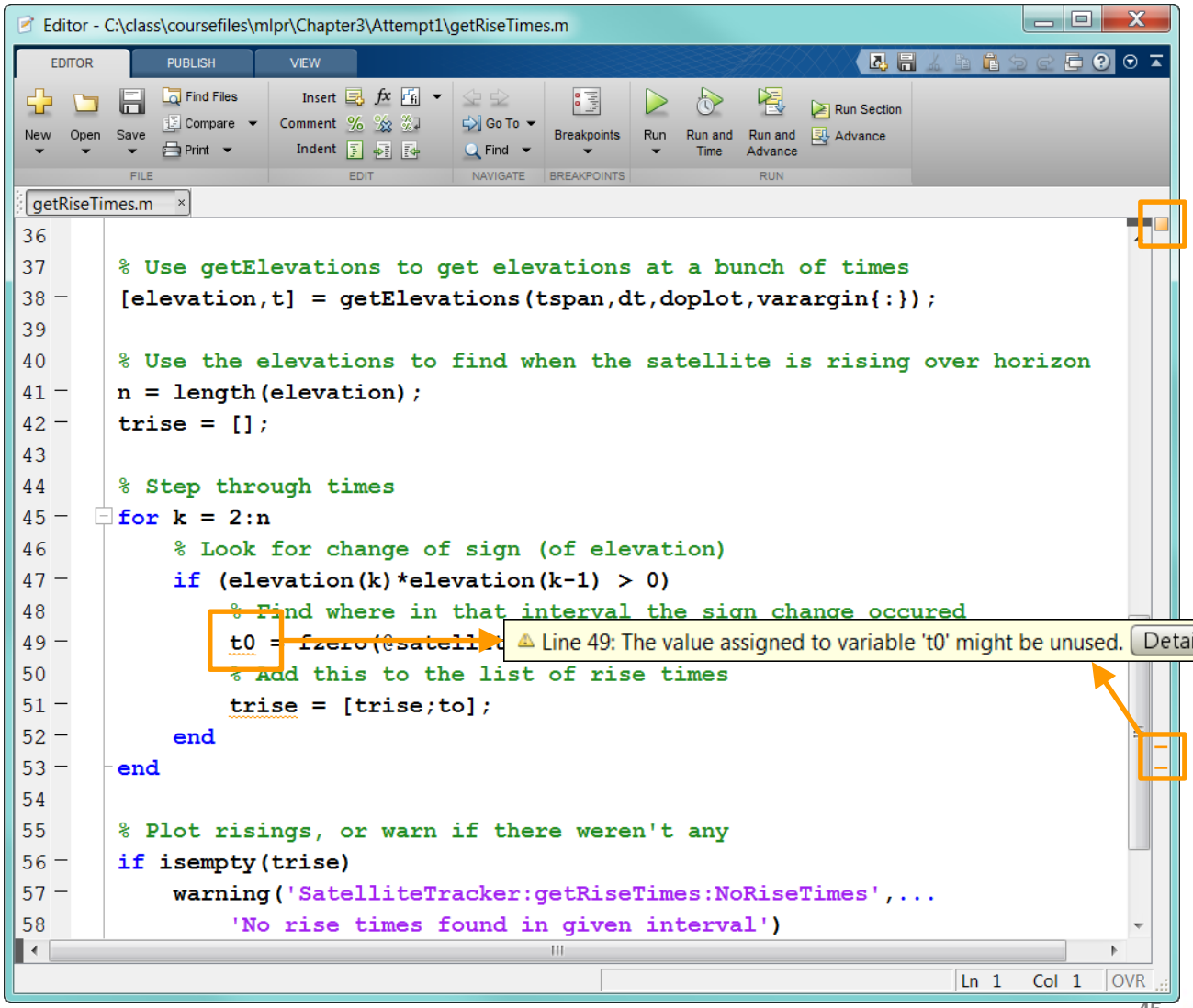


Analyzing Code in the Editor

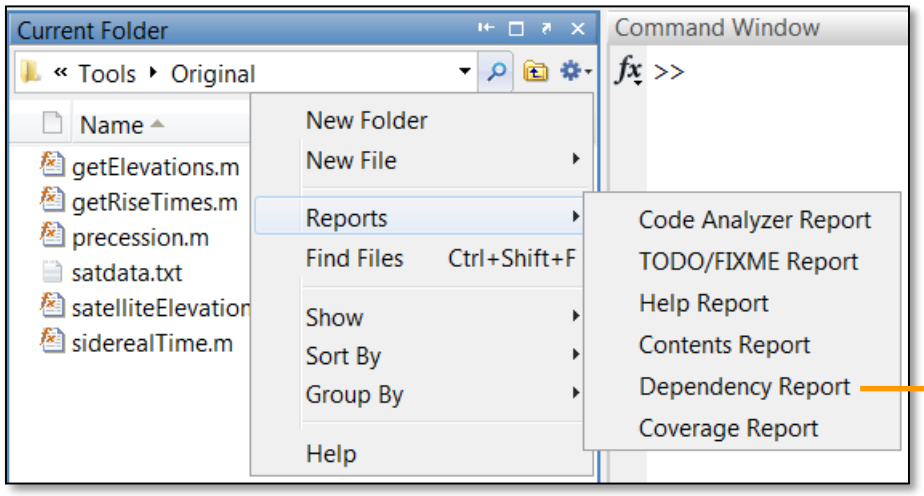
M-Lint messages display while you work in the Editor

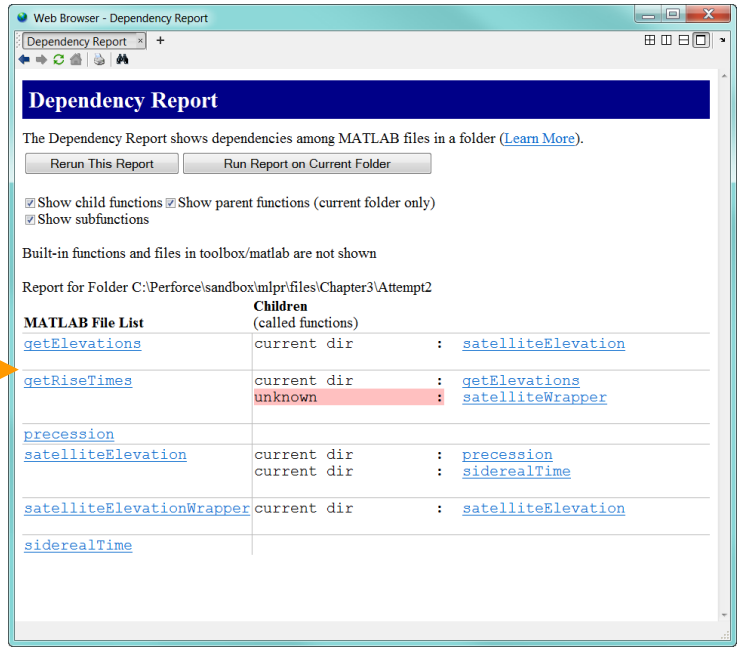
Red underline indicates code will have a syntax or run-time error (must fix)

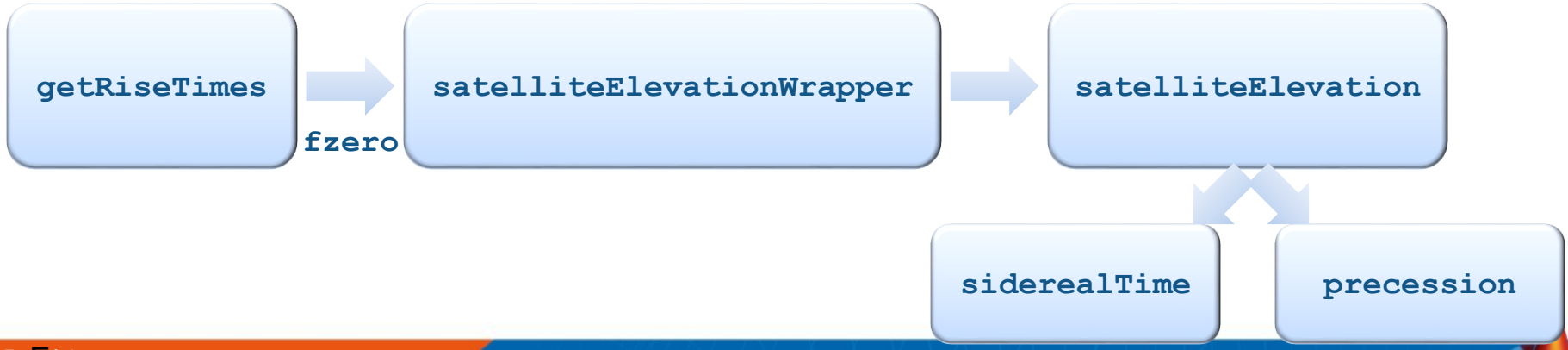
Orange underline indicates a variety of M-Lint warnings (investigate)



Resolving Dependencies



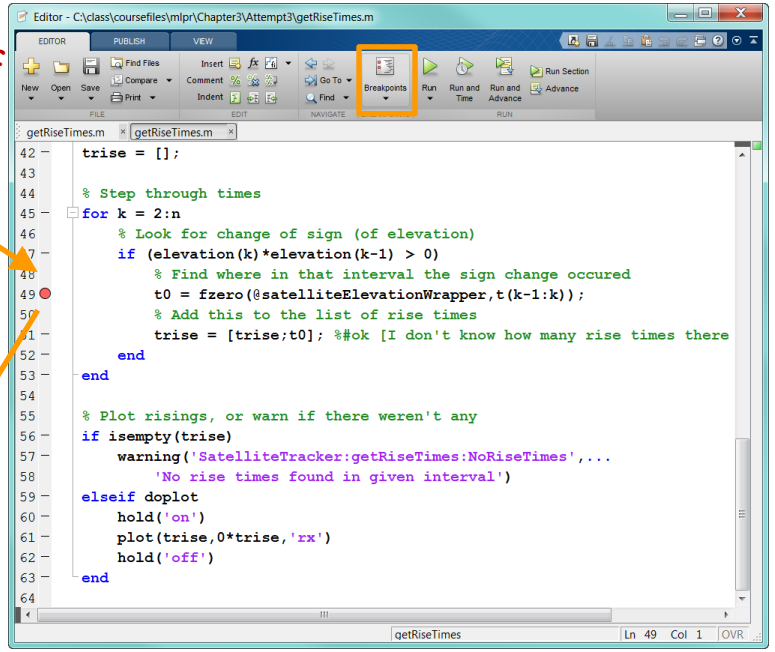




Entering Debug Mode

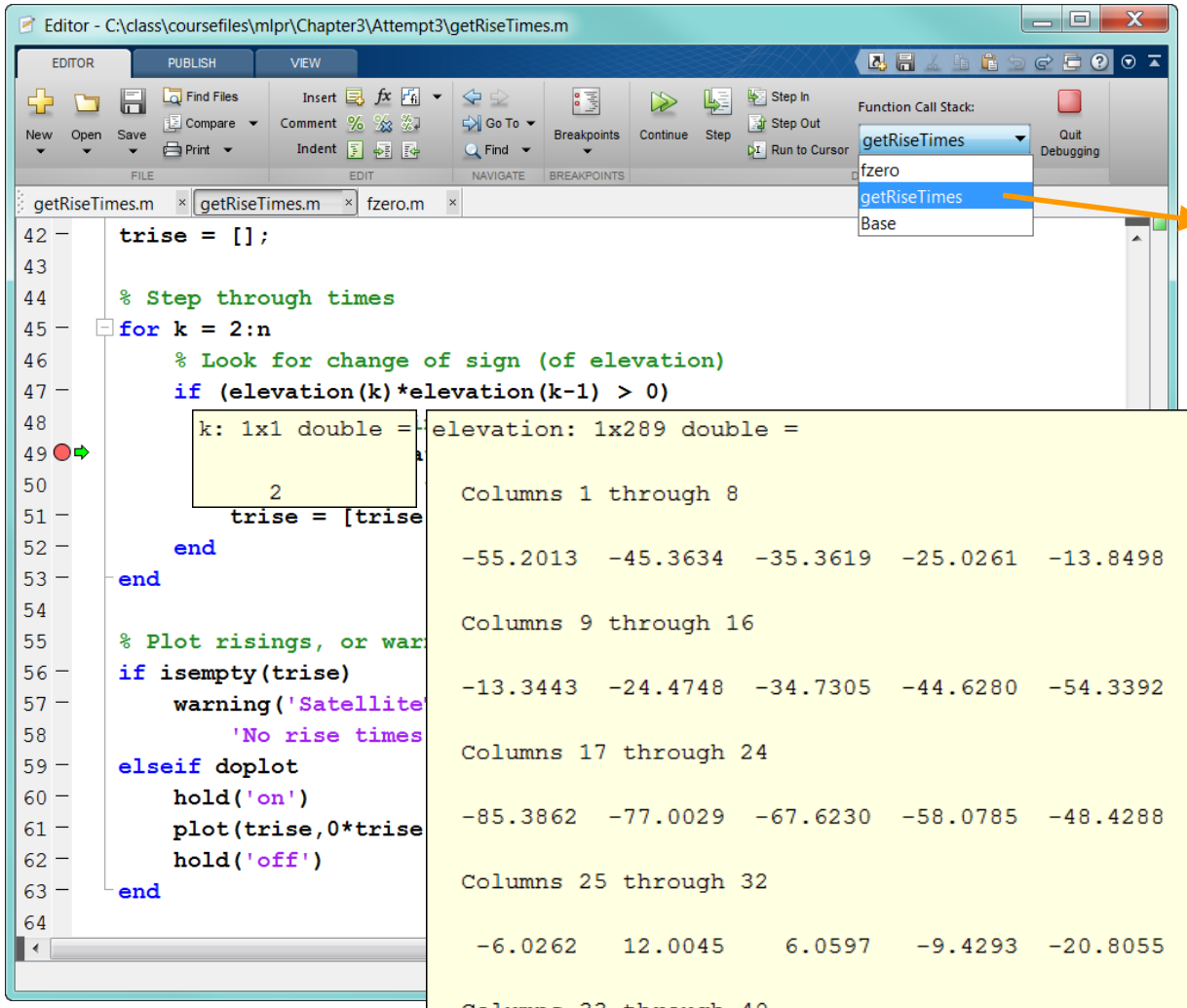
```
>> tr = getRiseTimes([t0,t0+1],5/1440)
Error using fzero (line 274)
The function values at the interval endpoints must differ in sign.

Error in getRiseTimes (line 49)
        t0 = fzero(@satelliteElevationWrapper
```

dbstop




```
>> tr = getRiseTimes([t0,t0+1],5/1440)
49          t0 = fzero(@satelliteElevationWrapper,t(k-1:k));
K>>
```

Examining Values

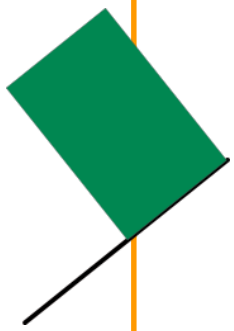



```
K>> dbstack
   In fzero at 274
> In getRiseTimes at 49
```

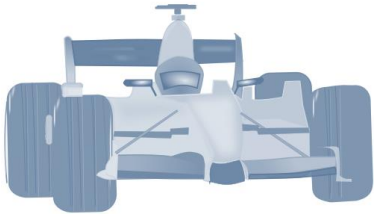
```
elevation: 1x289 double =

  Columns 1 through 8

  -55.2013  -45.3634  -35.3619  -25.0261  -13.8498    0.5581   42.5506    1.1157

  Columns 9 through 16

  -13.3443  -24.4748  -34.7305  -44.6280  -54.3392  -63.9286  -73.3900  -82.4574

  Columns 17 through 24

  -85.3862  -77.0029  -67.6230  -58.0785  -48.4288  -38.6440  -28.6216  -18.0786

  Columns 25 through 32

   -6.0262   12.0045    6.0597   -9.4293  -20.8055  -31.0932  -40.9609  -50.6185

  Columns 33 through 40

  -60.1405  -69.5231  -78.5776  -85.1389  -80.0699  -71.1252  -61.7409  -52.1936

  Columns 41 through 48
```

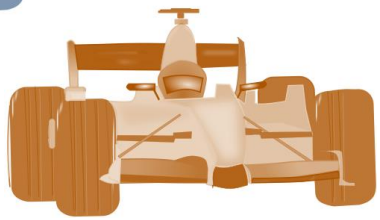
Diagnosing Performance

```
>> tic; tr = getRiseTimes([t0,t0+1],5/1440,false); toc
```

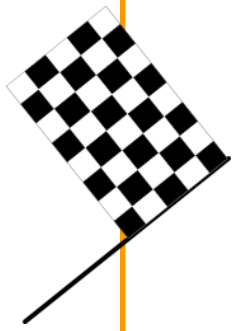


algorithm 1

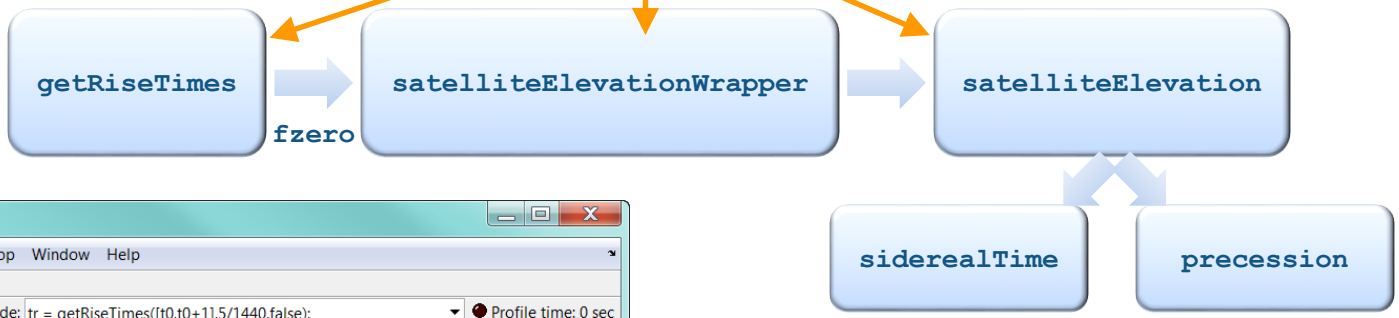


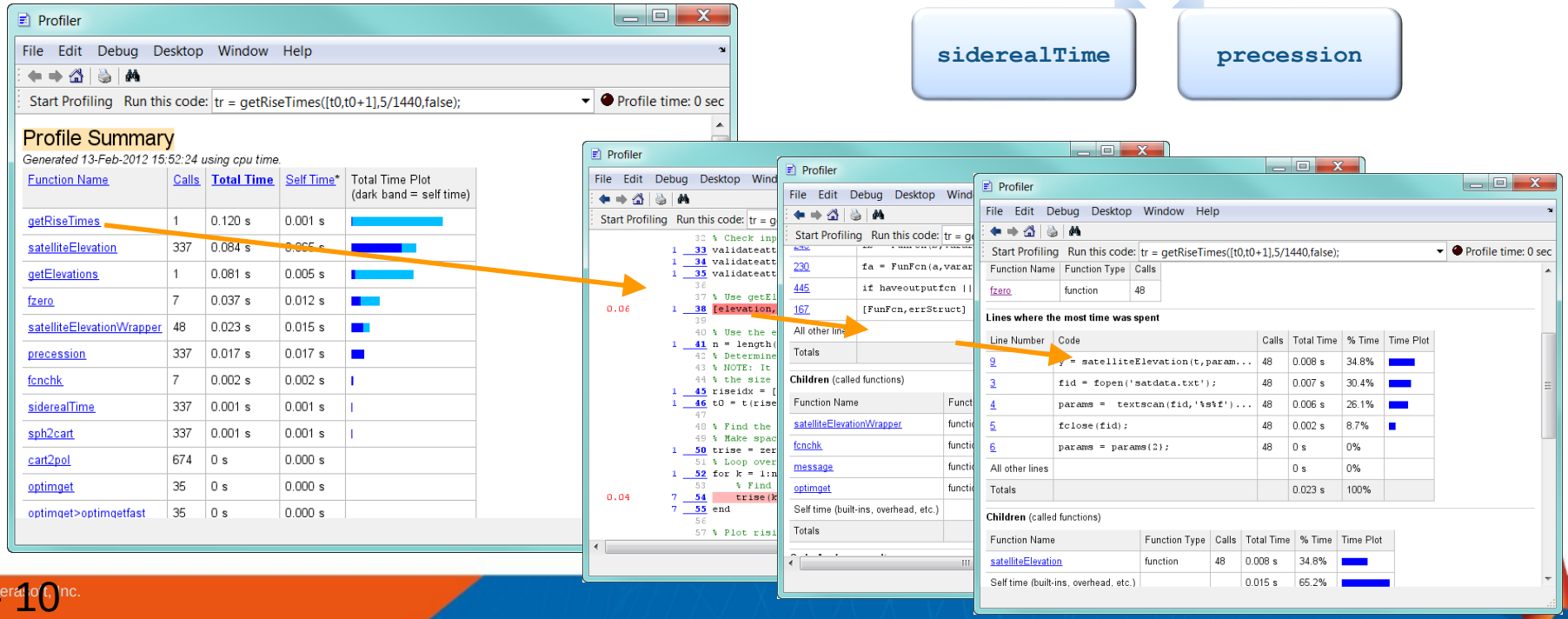


algorithm 2



MATLAB® Profiler

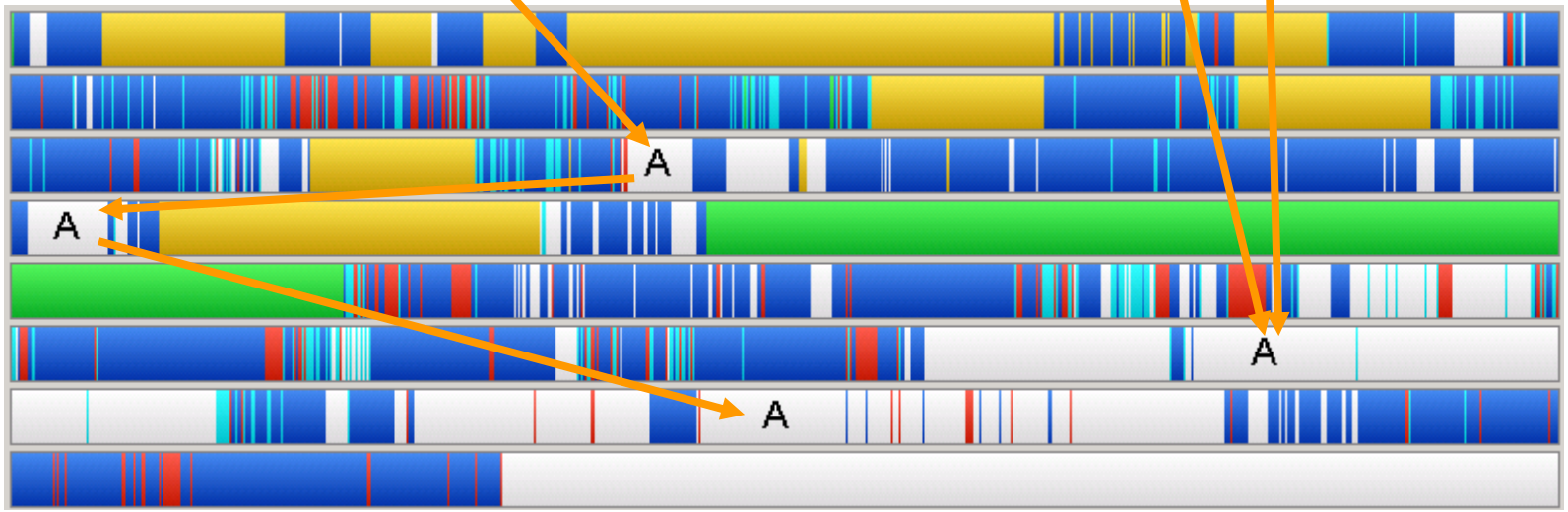




Preallocation of Memory

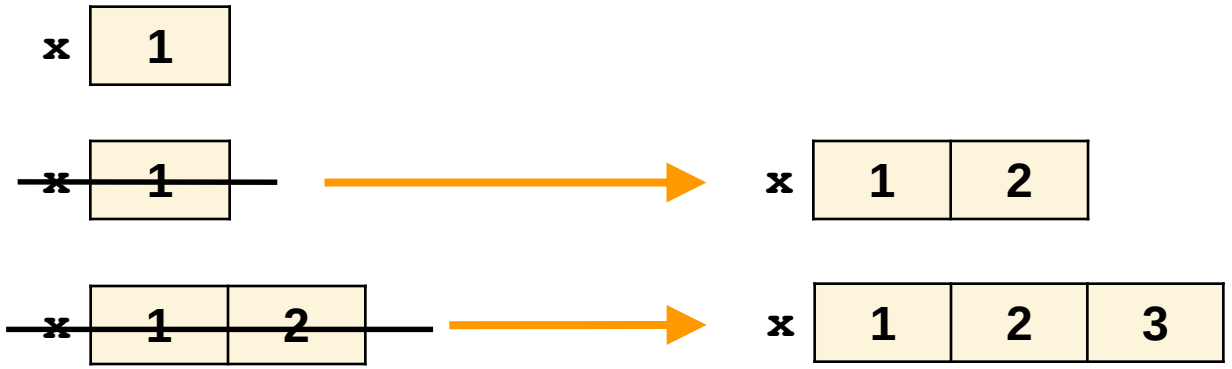
```
for J = 1:N
    A(J) = f(J);
end
```

```
A = zeros(1,N);
for J = 1:N
    A(J) = f(J);
end
```



Preallocation

```
x(1) = 1
x(2) = 2
x(3) = 3
```



```
x = zeros(1, 10)
```

x	0	0	0	0	0	0	0	0	0	0

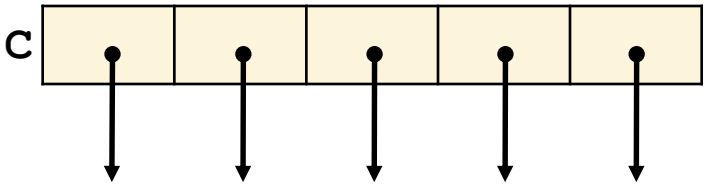
```
x(10) = 10
```

x	0	0	0	0	0	0	0	0	0	10

Preallocation (Continued)

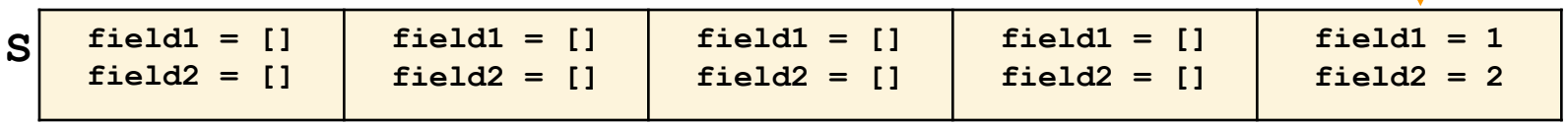
Cell Array

```
>> C = cell(1,5)
```



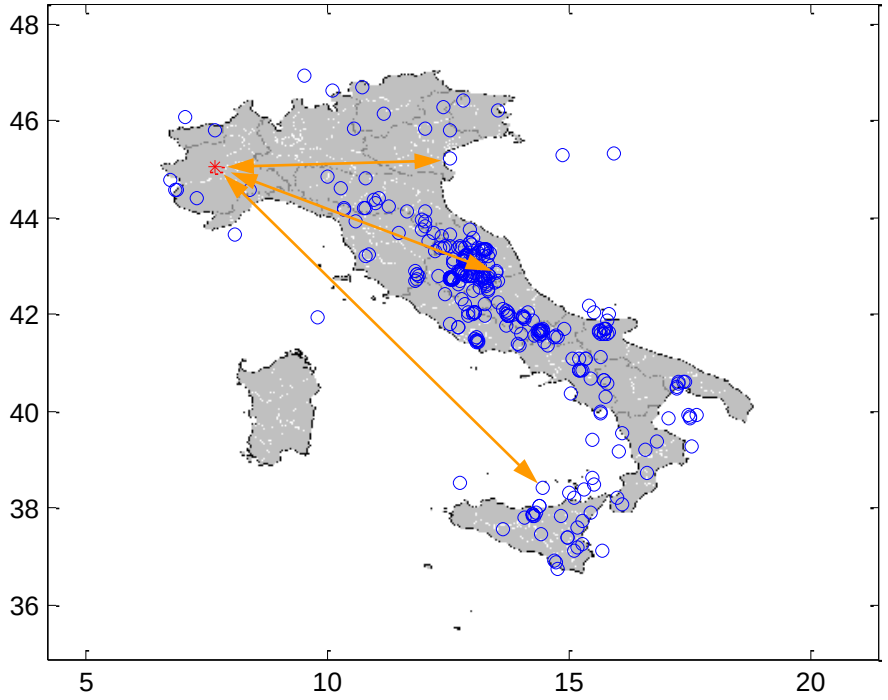
Structure Array

```
>> S(5).field1 = 1;
>> S(5).field2 = 2;
```



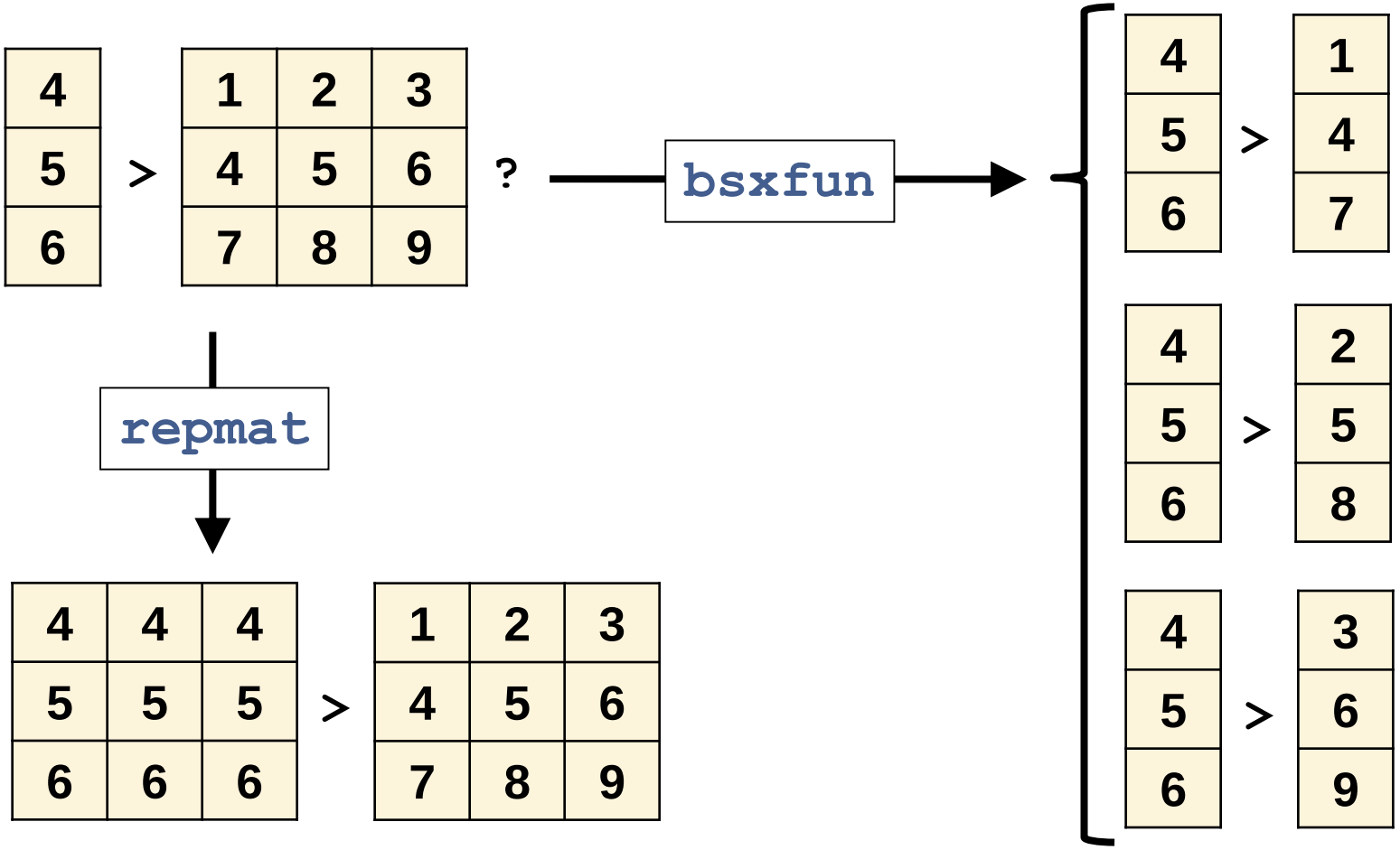
Vectorization

```
for k = 1:n
  d(k) = sqrt((x(k)-x0)^2 + (y(k)-y0)^2);
end
```




```
d = sqrt((x-x0).^2 + (y-y0).^2);
```

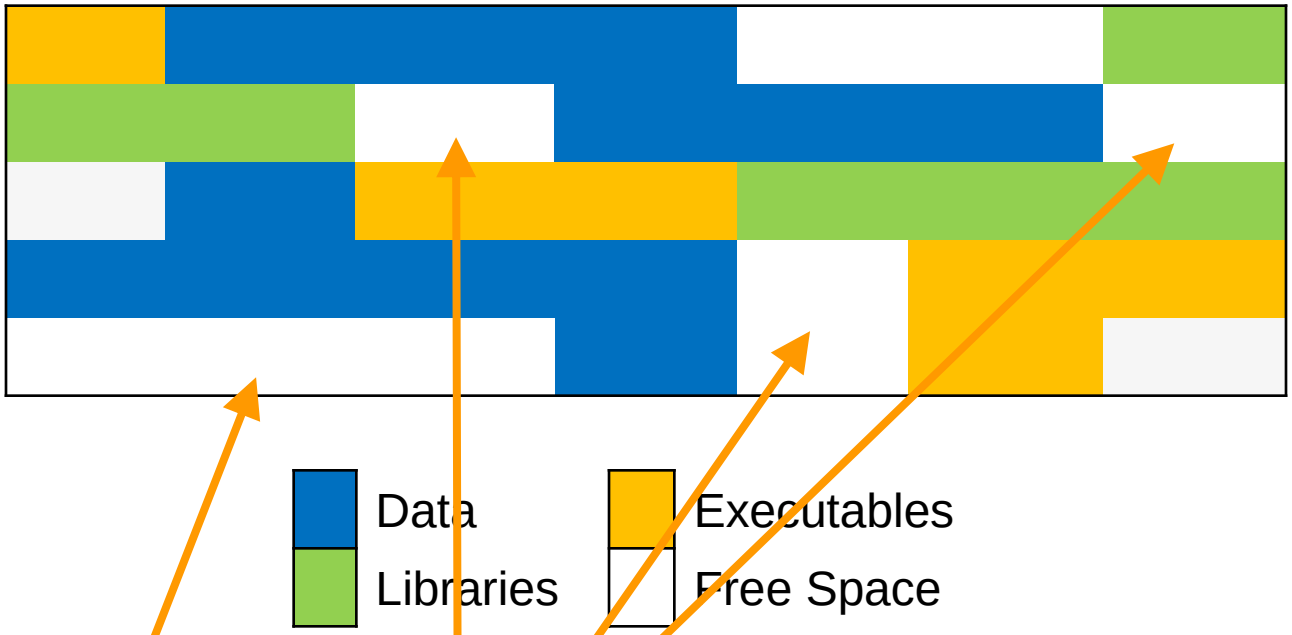
Vectorizing and Memory



Optimizing File I/O

	Read Functions	Write Functions
Text file	**`textscan`** is faster than **`textread`** to read large files	**`fprintf`** is vectorized, i.e., format is cycled through all the file
Binary file	**`memmapfile`** is faster than **`fread`**	**`fwrite`** **`multibandwrite`**

Memory Anatomy




```
>> memory
Maximum possible array:                804 MB (8.431e+008 bytes) *
Memory available for all arrays:      1273 MB (1.335e+009 bytes) **
Memory used by MATLAB:                 512 MB (5.367e+008 bytes)
Physical Memory (RAM):                3070 MB (3.219e+009 bytes)
```

Q & A

Thank you very much

Have a good time