MATLAB EXPO 2017
Advanced Programming with MATLAB
Agenda

- MATLAB and memory
  - What you as a programmer should know
    - Passing arrays
    - How structures use memory

- Functions of all types
  - Introduction/Review of MATLAB function types
  - Applications of new nested functions
    - Solving optimization problems
    - Building a graphical user interface for volume visualization
When Is Data Copied?

- Passing arrays to functions
  - When does MATLAB copy memory?

  ```matlab
  function y = foo(x,a,b)
  a(1) = a(1) + 12;
  y = a*x+b;
  ```

- Calling `foo`
  ```matlab
  y = foo(1:3,2,4)
  ```
  - i.e., `x = 1:3, a = 2, b = 4`

  >> edit foo.m
In-place Optimizations

- When does MATLAB do calculations “in-place”?

\[
x = 2 \times x + 3;
y = 2 \times x + 3;
\]
In-place Optimizations

When does MATLAB do calculations “in-place”? 

```matlab
function showInPlace
    xx = randn(n,1);
    xx = myfunc(xx); % vs. yy = myfunc(xx)
    xx = myfuncInPlace(xx); % vs. yy = myfuncInPlace(xx)
end

function x = myfuncInPlace(x)
    x = sin(2*x.^2+3*x+4);
end

function y = myfunc(x)
    y = sin(2*x.^2+3*x+4);
end
```

>> edit myfuncInPlace myfunc showInPlace
% separate functions, separate files
In-place Optimizations

\[ y = \text{myfunc}(x); \]
\[ x = \text{myfunc}(x); \]
\[ y = \text{myfuncInPlace}(x); \]
\[ x = \text{myfuncInPlace}(x); \]
Memory Used for Different Array Types

\[
d = [1 \ 2] \quad \text{% Double array}
dcell = \{d\} \quad \text{% Cell array}
dstruct.d = d \quad \text{% Structure}
\]

whos

>> edit overhead

MATLAB EXPO 2017
How does MATLAB store data?

Container overhead

\[ d = [1 \ 2] \]

\[ dcell = \{d\} \]

\[ dstruct.d = d \]
Container Overhead

<table>
<thead>
<tr>
<th>Command</th>
<th>Data Type</th>
<th>Reported Memory</th>
<th>Unreported Memory</th>
<th>Total Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;&gt; d = [1 2]</td>
<td>Double</td>
<td>16 bytes</td>
<td>112 bytes</td>
<td>128 bytes</td>
</tr>
<tr>
<td>&gt;&gt; dcell = {[1 2]}</td>
<td>Cell</td>
<td>128 bytes</td>
<td>112 bytes</td>
<td>240 bytes</td>
</tr>
<tr>
<td>&gt;&gt; dstruct.d = [1 2]</td>
<td>Struct</td>
<td>192 bytes</td>
<td>112 bytes</td>
<td>304 bytes</td>
</tr>
</tbody>
</table>

```matlab
>> d = [1 2]  % Double 16 bytes 112 bytes 128 bytes 16 = 16
>> dcell = {[1 2]}  % Cell 128 bytes 112 bytes 240 bytes 16+112 = 128
>> dstruct.d = [1 2]  % Struct 192 bytes 112 bytes 304 bytes 16+64+112 = 192
```

MATLAB EXPO 2017
MATLAB and Memory

How does MATLAB store structures?

```matlab
n = 10000;
s.A = rand(n,n);
s.B = rand(n,n);
sNew = s;

s.A(1,1) = 17;

>> edit structmem1
MATLAB EXPO 2017
```
MATLAB and Memory

How does MATLAB store structures?

```matlab
im1.red = redPlane;  % each plane is m x n
im1.green = greenPlane;
im1.blue = bluePlane;

versus

% each 1x3
im2(1,1).pixel = [red(1) green(1) blue(1)];
im2(2,1).pixel = [red(2) green(2) blue(2)];
...
im2(m,n).pixel = [red(m*n) green(m*n) ... blue(m*n)];

>> edit structmem2
```

MATLAB EXPO 2017
Tables

- New fundamental data type
- For mixed-type tabular data
  - Holds both data and metadata
- Supports flexible indexing
- Built-in functionality (merge, sort, etc.)
Working with **Big Data** Just Got Easier

Use tall arrays to manipulate and analyze data that is too big to fit in memory

- Tall arrays let you use familiar MATLAB functions and syntax to work with big datasets, even if they don’t fit in memory
- Support for hundreds of functions in MATLAB and Statistics and Machine Learning Toolbox
- Works with Spark + Hadoop Clusters

Learn more at this session: 
*Big Data and Machine Learning*

MATLAB EXPO 2017
Summary of Tall Array Capabilities

Data Access
- ASCII File
- Database (SQL)
- Spreadsheet
- Custom Files
  - table
  - cell
  - numeric
  - cellstr & string
  - Date & Time
  - categorical

Data Munging
(100’s of functions)
- Math
- Statistics
- Missing Data
- Visualization
- Date/Time
- String

Machine Learning
- Linear Model
- Logistic Regression
- Discriminant Analysis
- K-means
- PCA
- Random Data Sampling
- Summary Statistics

Provides purpose-built functions for use with data that does not fit in memory
Execution Environments for Tall Arrays

Process out-of-memory data on your Desktop to explore, analyze, gain insights and to develop analytics.

Use Parallel Computing Toolbox for increased performance.

Run on Compute Clusters, or Spark if your data is stored in HDFS, for large scale analysis.
Categorical Arrays

- New fundamental data type
- For discrete non-numeric data
  - Values drawn from a finite set of possible values ("categories")
- More memory efficient than a cell array of strings
- Can be compared using logical operators
  - Similar to numeric arrays

MATLAB EXPO 2017

>> edit showCategorical.m
Strings
The better way to work with text

- Manipulate, compare, and store text data efficiently
  
  ```
  >> "image" + (1:3) + ".png"
  ans =
  1×3 string array
    "image1.png"    "image2.png"    "image3.png"
  ```

- Simplified text manipulation functions
  - Example: Check if a string is contained within another string
    - Previously: `if ~isempty(strfind(textdata,"Dog"))`
    - Now: `if contains(textdata,"Dog")`

- Performance improvement
  - Up to 50x faster using `contains` with `string` than `strfind` with `cellstr`
  - Up to 2x memory savings using `string` over `cellstr`
Summary of MATLAB and Memory

- How MATLAB passes arrays to functions
  - By value, with “lazy” copy or copy on write
  - In-place optimization code pattern

- Memory use in array storage
  - Atomic types vs. cell arrays and structures
  - Array of structs vs. struct arrays
    - i.e., s(300,300).red vs. s.red(300,300)
  - Categorical arrays, tables, strings
MATLAB and Memory: Additional Resources

- Recorded Webinar:
  - Tackling Big Data in MATLAB
  - mathworks.com -> Recorded Webinars

- Memory Management information from doc page for memory
Functions

Everything you wanted to know, but were afraid to ask
Question:

With the new function types introduced in R14, I am curious as to which methods would be preferable from a speed standpoint. We have:

1. Functions defined in separate files
2. Subfunctions
3. Inline functions
4. Anonymous functions
5. Nested functions

Other than scope, persistence, or program structure, is there some difference that makes MATLAB work better with the type?
Good question, and a hard one to answer. As MATLAB evolves, the answer will change. The function call mechanism in the current version of MATLAB is pretty expensive, for any kind of function. One of the most important tasks facing our Accelerator/JIT team today is to make function calls faster. This is why we have introduced the @ notation for both function handles and anonymous functions. Future versions of MATLAB should have improved speed for anything involving the @ sign.

Inline functions were an experiment in the use of the overloading mechanism that turned out to be useful. But we’ve always regarded them as a bit of a hack. So our advice now is to phase out their use over time. Other than that, your choice today among the other kinds of functions should be based on style and convenience.
Function Quiz

Let’s see how much you know about functions in MATLAB

This quiz covers
- Anonymous functions
- Nested functions
- Function handles
- Regular (“simple”) functions
- Subfunctions

This quiz does not cover
- MEX functions, private functions, …
Function Quiz Review

- Regular (“simple”) functions
- Function handles
- Anonymous functions
- Local functions
- Nested functions

Q: Which of these function types can be combined in a single file?
Advice

- You can use the function `functions` to explore details of the function referred to by a particular function handle.

- NOTE: Changing the values of the struct output from `functions` does NOT alter the function handle details!
Nested Function Applications

- Solving optimization problems
- Building a graphical user interface for volume visualization
- Building 2-figure GUIs (optional)
Application 1: Solving Optimization Problems

- We get many posts on comp.soft-sys.matlab about optimization. The problems fall into several categories. The one we will address to today is:
  - How to include extra parameters to define the objective function
Optimization Example (unconstrained)

Objective function:

\[ ax_i^2 + bx_1x_2 + cx_2^2 + d|x_2 - x_1| + e \text{ randn} \]

\( a,b,c \) – “Regular” parameters
\( d,e \) – Additional parameters that might alter the type of problem by making the objective function either

- non-smooth (d) or
- stochastic (e)

>> edit optimLocal.m
Application 2: Building a Graphical User Interface for Volume Visualization

- Application: Building a custom tool for volume visualization
- This example illustrates:
  - Using function handles to export nested functions
  - Using nested functions for object callbacks
Why Use Nested and Anonymous Functions?

Benefits of nested and anonymous functions

- More robust
- Changes in path cause fewer problems with function handles
- Data sharing and number of copies of data
  - Memory savings, especially for large shared data
- Program structure
- Over time, higher performance

Additional benefits of nested (and sub) functions

- Scope and persistence
- Reduce variable and function namespace clutter
Summary and Advice
Anonymous Functions

- Create simple functions without creating m-files:
  - \( \text{fh} = @(x,y) a \cdot \sin(x) \cdot \cos(y) \)

- Useful for “fun-funs” (function functions)
  - Optimization
  - Solving differential equations
  - Numerical integration
  - Plotting
  - Array functions (cellfun, structfun, …)

- Convenient for simple callbacks
  - \( \text{getXLSData} = @(\text{worksheet}) \text{xlsread('records.xls', worksheet)}; \)
Summary and Advice
Nested Functions

- Embed one function within another, with shared, persistent workspace:
  - function main
  -   function nest
  -   end
  - end

- Useful for “fun-funs” (function functions)
- Very convenient for callbacks (shared workspace)
- Encapsulate functionality and data (export as function handles)
Function Usage Advice

- You can call functions without inputs using this notation:

\[ y = \text{foo}(); \]

- Put all nested functions at the bottom of the enclosing function. You may prefer to have them defined right where you will use them, but this can become unwieldy.

- Place a comment next to a call to a nested function stating what workspace variables are altered by the call (since the nested function can see the enclosing workspace).
More Function Usage Advice

- Nested functions are good when there is significant shared information (many pieces or large arrays).

- Nested functions are very good when there is much large data to share for reading and writing.

- Local functions (or private functions) are useful when users don't need these directly. Those who need them can see.

- Data sharing inside local functions is only reading large arrays, not making copies and changing them.
MATLAB Function Types: Additional Resources

- **MATLAB Digest - September 2005**
  - Dynamic Function Creation with Anonymous and Nested Functions

- **Examples**
  - Anonymous functions
  - Nested Functions

- **The Art of MATLAB Blog**
  - Look at Category: Function Handles
Summary

- MATLAB memory usage
  - When is data copied

- Nested and anonymous functions
  - Very good at data encapsulation
  - Efficient for sharing memory between functions
  - Another choice for name / variable scope
  - Less clutter on MATLAB path