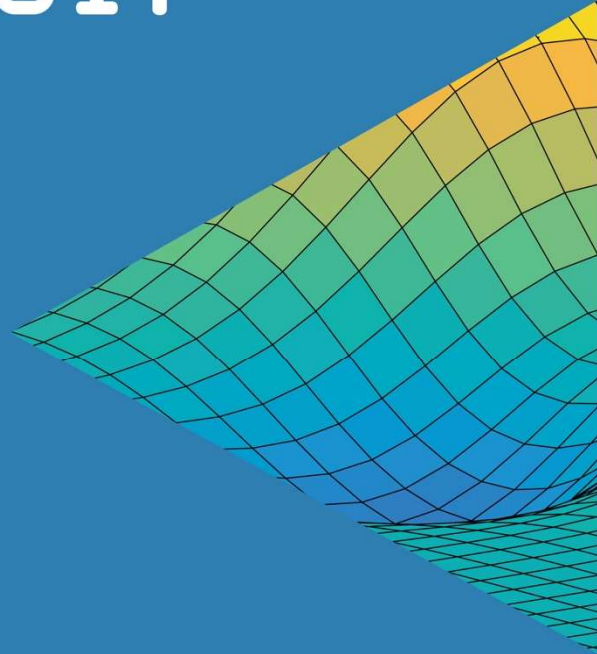


# MATLAB EXPO 2017

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?

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

- Calling `foo`  
`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*x + 3;
```

```
y = 2*x + 3;
```

# In-place Optimizations

When does MATLAB do calculations “in-place”?

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

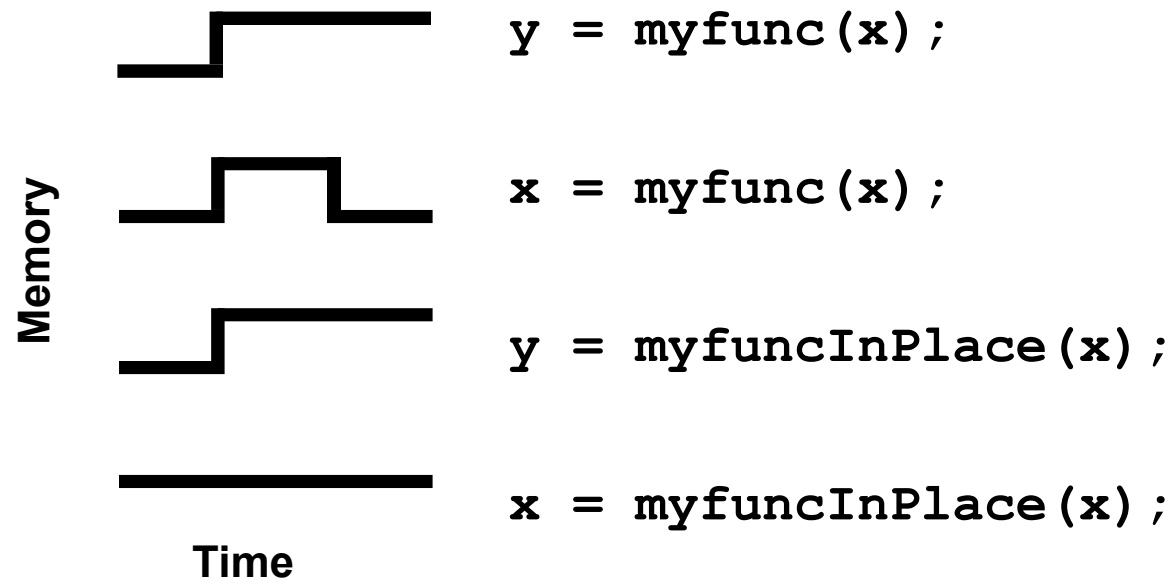
```
function x = myfuncInPlace(x)
x = sin(2*x.^2+3*x+4);
```

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

```
>> edit myfuncInPlace myfunc showInPlace
```

MATLAB EXPO 2017 %separate functions, separate files

# In-place Optimizations



## Memory Used for Different Array Types

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

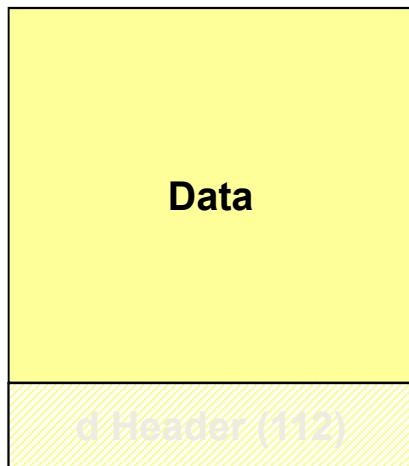
```
whos
```

```
>> edit overhead
```

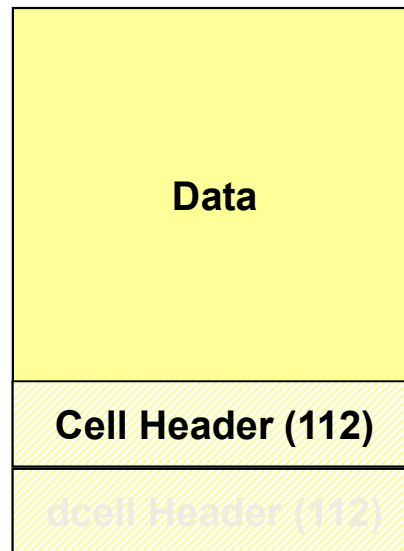
MATLAB EXPO 2017

# How does MATLAB store data?

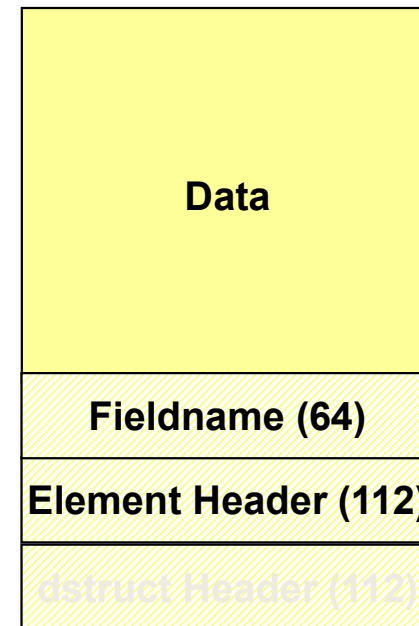
## *Container overhead*



`d = [1 2]`



`dcell = {d}`

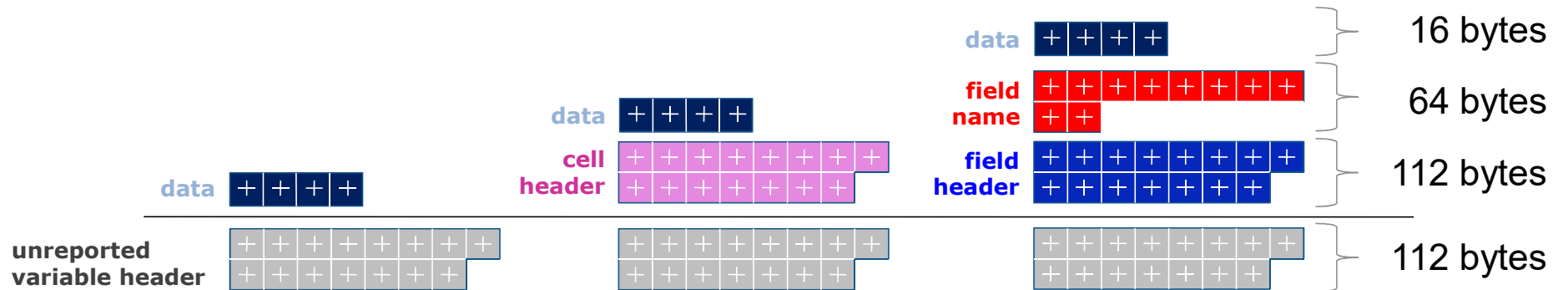


`dstruct.d = d`



# Container Overhead

Command	Data Type	Reported Memory	Unreported Memory	Total Memory	
>> 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 and Memory

How does MATLAB store structures?

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

```
sNew = s;
```

```
s.A(1,1) = 17;
```

```
>> edit structmem1
```

## MATLAB and Memory

How does MATLAB store structures?

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

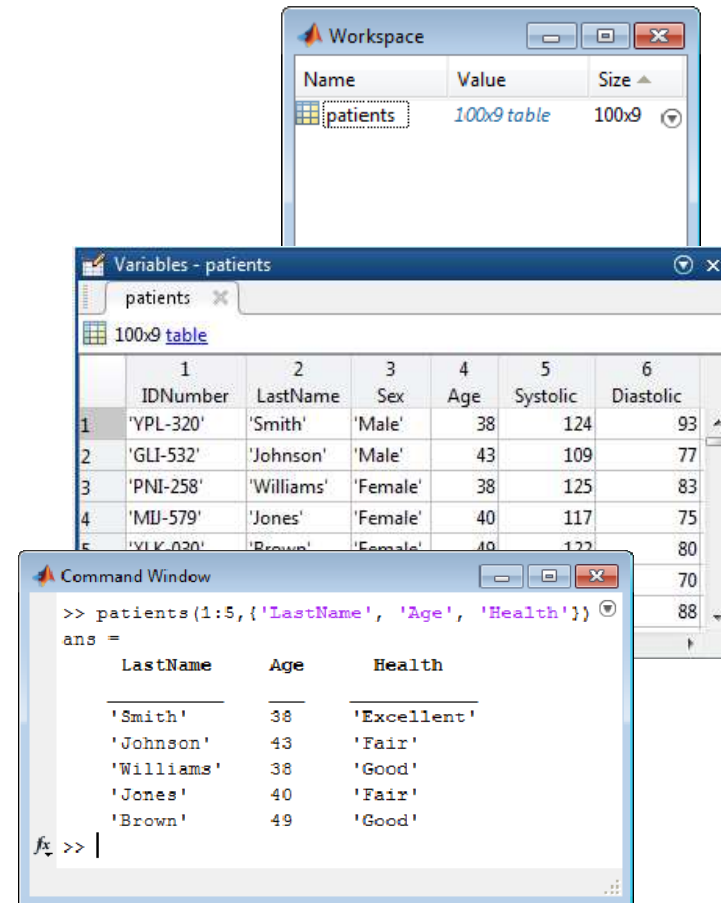
versus

```
% 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
```

# Tables

- New fundamental data type
- For mixed-type tabular data
  - Holds both data and metadata
- Supports flexible indexing
- Built-in functionality (merge, sort, etc.)



The screenshot shows three MATLAB windows:

- Workspace:** Shows a variable named 'patients' of type '100x9 table' with a size of 100x9.
- Variables - patients:** Displays a preview of the table with columns: IDNumber, LastName, Sex, Age, Systolic, Diastolic. The first five rows are visible.
- Command Window:** Shows the command `patients(1:5, {'LastName', 'Age', 'Health'})` and the resulting output:
 

LastName	Age	Health
'Smith'	38	'Excellent'
'Johnson'	43	'Fair'
'Williams'	38	'Good'
'Jones'	40	'Fair'
'Brown'	49	'Good'

# Working with Big Data Just Got Easier

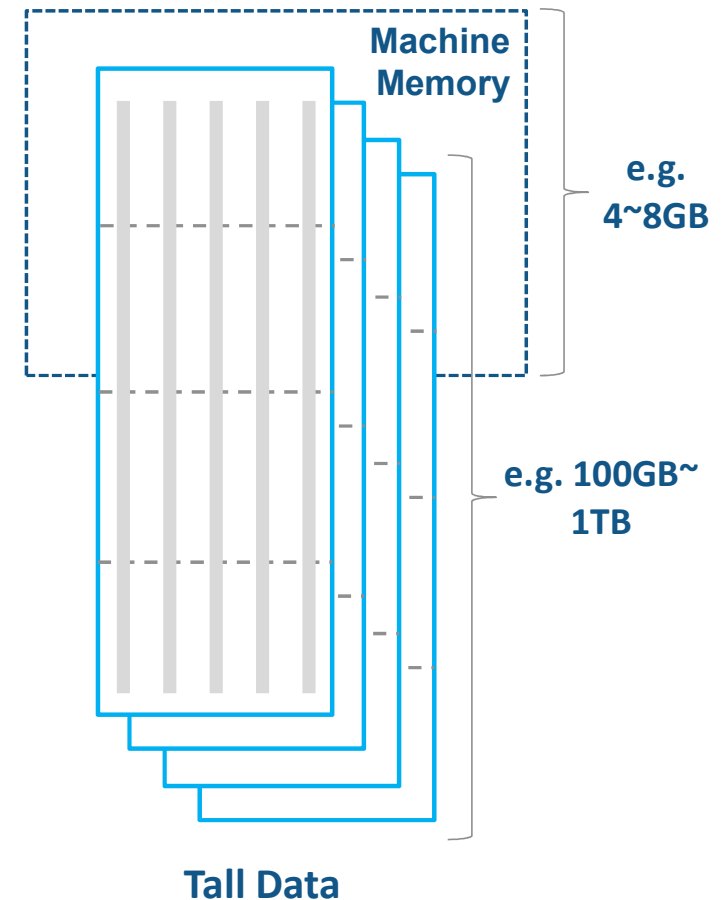
**R2016b** **R2017a**

## 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

Provides purpose-built functions for use with data that does not fit in memory

## Data Access

ASCII File  
 Database (SQL)  
 Spreadsheet  
 Custom Files

- table
- cell
- numeric
- cellstr & string
- Date & Time
- categorical

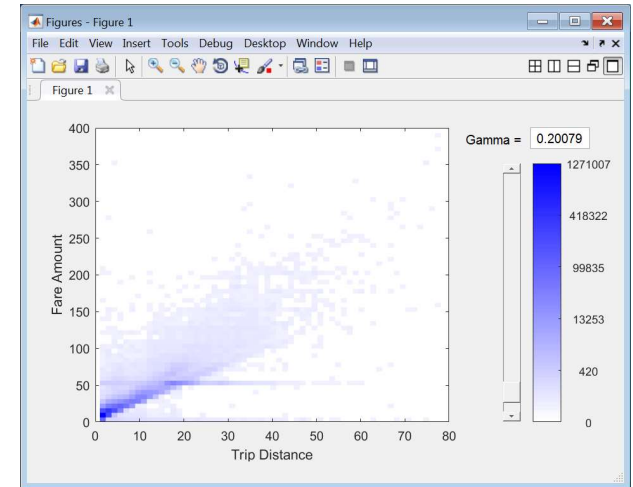
## Data Munging

*(100's of functions)*

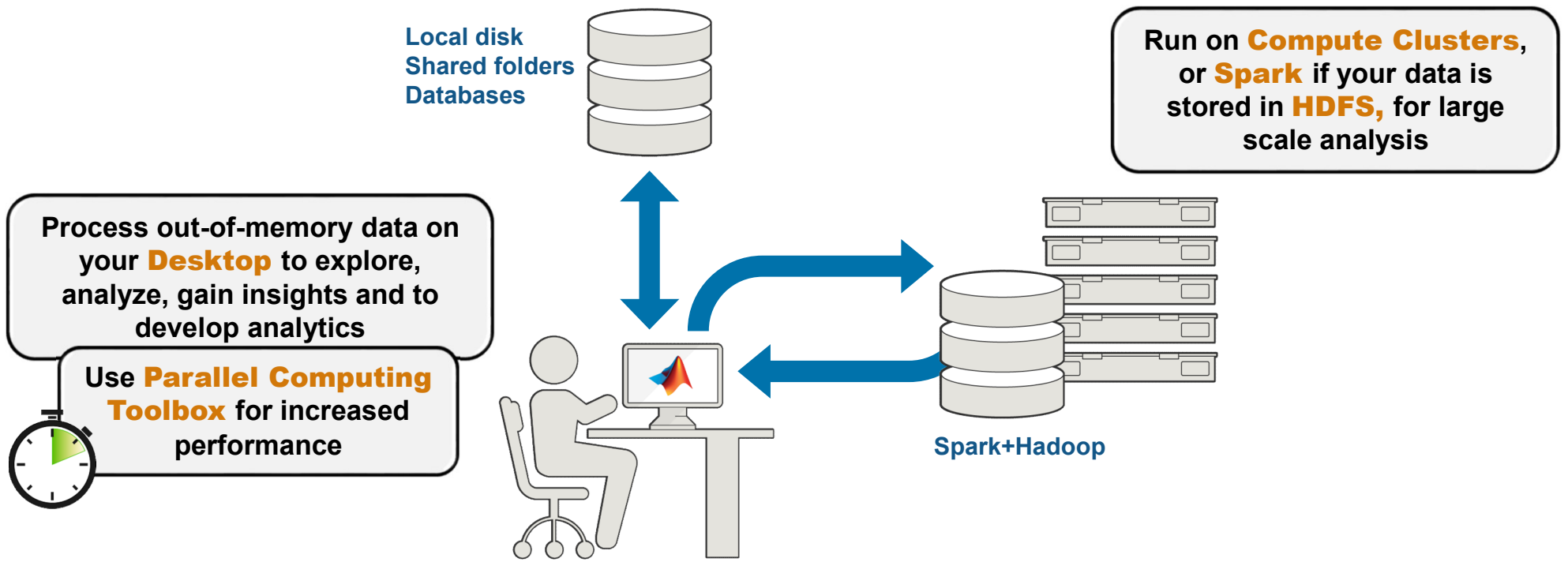
Math  
 Statistics  
 Missing Data  
 Visualization  
 Date/Time  
 String

## Machine Learning

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



# Execution Environments for Tall Arrays



MATLAB EXPO 2017

# 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

```
Command Window
>> patients.Health(1:5)
ans =
    Excellent
    Fair
    Good
    Fair
    Good
fx >> |
```

Variables - patients

PLOTS VARIABLE VIEW

patients

100x9 table

	7	8	9
	Height	Weight	Health
1	1.8000	80	Excellent
2	1.7500	74	Excellent
3	1.6300	59	Fair
4	1.7000	60	Good
5	1.6300	54	Poor

```
Command Window
>> patients2.LastName(patients2.Health < 'Good')
ans =
    'Thomas'
    'Kelly'
    'Wood'
    'Foster'
    'Griffin'
    'Bayer'
fx >> |
```

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"
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](http://mathworks.com) -> Recorded Webinars
- Memory Management information from doc page for `memory`

## Functions

Everything you wanted to know,  
but were afraid to ask

## A comp.soft-sys.matlab Post

### 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?*

## comp.soft-sys.matlab Post Answered

### Cleve's Answer

*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 today is:
  - How to include extra parameters to define the objective function

## Optimization Example (unconstrained)

Objective function:

$$ax_1^2 + bx_1x_2 + cx_2^2 + d|x_2 - x_1| + e \text{ randn}$$

$a, b, c$  – “Regular” parameters

$d, e$  – Additional parameters that might alter the type of problem by making the objective function either

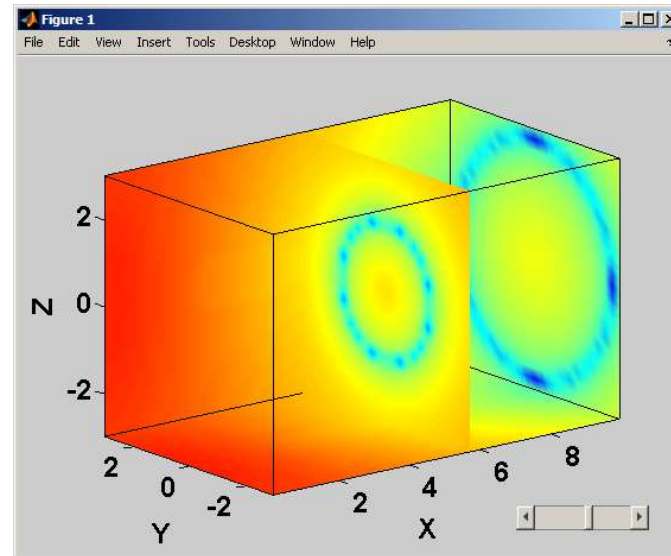
- non-smooth ( $d$ ) or
- stochastic ( $e$ )

```
>> edit optimLocal.m
```

MATLAB EXPO 2017

## 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:
  - `fh = @(x,y) a*sin(x)*cos(y)`
- Useful for “fun-funs” (function functions)
  - Optimization
  - Solving differential equations
  - Numerical integration
  - Plotting
  - Array functions (`cellfun`, `structfun`, ...)
- Convenient for simple callbacks
  - `getXLSData = @(worksheet) 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 = 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
  - <http://www.mathworks.com/company/newsletters/digest/2005/sept/dynfunctions.html>
- **Examples**
  - Anonymous functions
  - <http://www.mathworks.com/products/matlab/demos.html?file=/products/demos/shipping/matlab/anondemo.html>
  - Nested Functions
  - <http://www.mathworks.com/products/matlab/demos.html?file=/products/demos/shipping/matlab/nesteddemo.html>
- **The Art of MATLAB Blog**
  - Look at Category: Function Handles
  - <http://blogs.mathworks.com/loren/>

## 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