

Nested Functions in MATLAB®

BY LOREN SHURE

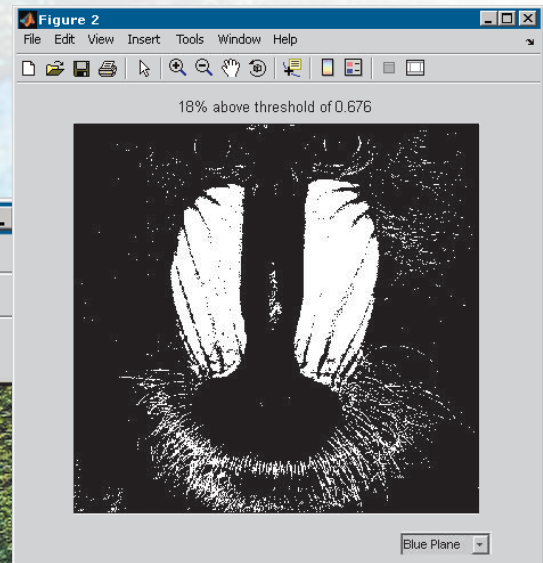
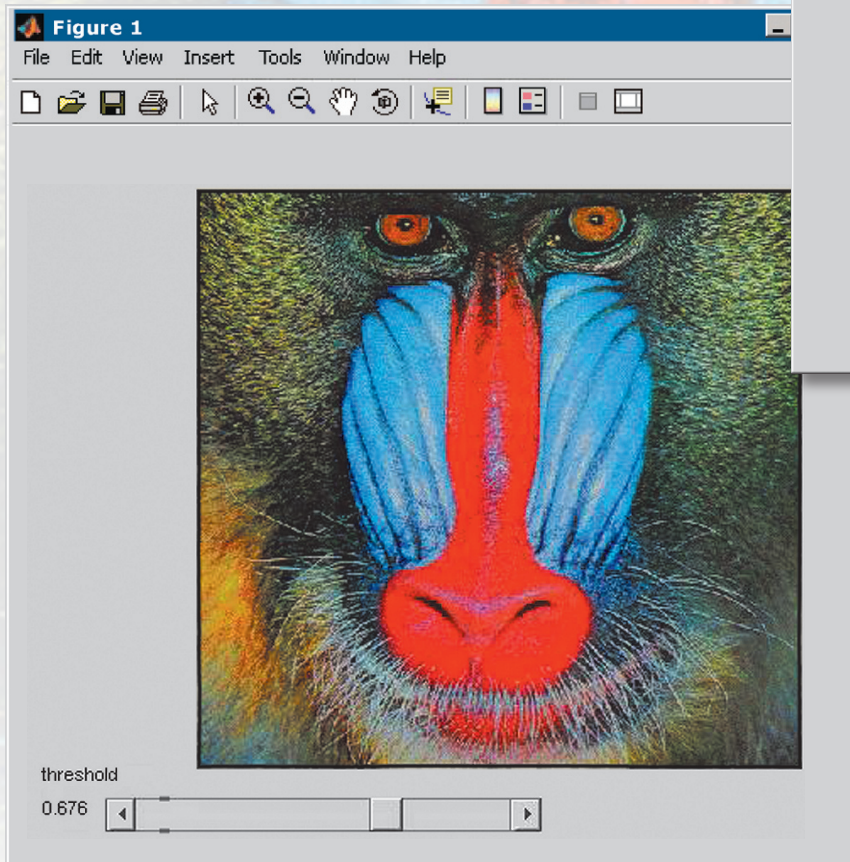


Figure 1. GUI with figures showing initial data and the selected threshold parameter (left), and the chosen plane to view and the calculated results (above).

At The MathWorks, we frequently receive questions on the MATLAB USENET newsgroup (comp.soft-sys.matlab) that ask how to share information among multiple elements, such as figures in a graphical user interface (GUI) application. In this article, we describe how to write such an application with elegant code using nested functions, a new feature in MATLAB 7.

Suppose we want to create a GUI application with which you can threshold an RGB image that produces a binary result. The GUI must show the input image and the result in separate windows to avoid conflicts between different color maps.

We find ourselves with some potentially serious code considerations in such a setup: how to use memory efficiently (we do not want to make unnecessary copies of image data) and how to scope the information (image data as well as GUI components) appropriately.

Before launching into the example details, let's explore nested functions.

Nested Functions

A nested function:

- Literally nests inside another function
- Can access variables available from the outer function's scope

Nested functions are not seen as functions on the `matlabpath` and therefore do not clutter the overall namespace. As a result, M-files created with nested functions are often more robust to environmental

changes. Because the nested function can access and change variables in the outer scope, there is less need to copy data, which can strain memory capacity if data sizes are large. You can use nested functions to simplify program structures and function interfaces because you can access variables from the enclosing workspace without having to pass them in or out as arguments. Consequently, nested functions let you simplify the code and improve performance.

As with other types of functions, you can make a nested function visible beyond its normal scope by using a function handle, which carries with it the context in which it is created—in particular, its workspace. A workspace associated with one handle to a given function is independent of the workspace associated with a handle to the same function formed at a different time and in a different context. This property enables the independence of different instantiations of the same GUI. You can download the example code for this article from the MATLAB Central File Exchange to try creating multiple instances of the example GUI, each with a different image, and see that they operate independently.

Two-Figure GUI Example

Our example GUI displays two figures (Figure 1). The first displays an RGB image and a slider to select a threshold parameter. The second includes a pop-up menu (pop-up menu) for selecting the color plane (red, green, or blue) and displays a black-and-white masked image identifying areas in that plane that exceed the threshold value. The title for the second window displays the percentage of this plane that meets the threshold criteria.

How do we make a memory-efficient GUI like this example, which must process a large dataset and perform some calculations on it, without creating unnecessary temporary, intermediate variables? How do we enable both figures to share information, such as a threshold value?

```

22 % Create threshold uicontrol for threshold value
23 thresh = uicontrol('style','slider', ...
24     'position',[60 10 256 20], ...
25     'value',0, ...
26     'parent', fig1, ...
27     'callback',@updateResultFig);
28
29 uicontrol('style','text', ...
30     'position', [20 30 80 20], ...
31     'HorizontalAlignment','left',...
32     'parent', fig1, ...
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78

```

Figure 2. Nested functions provide a convenient framework for creating a GUI application in which we can easily alter the UI layout. For example, on line 46, we could change the parent of RGBdrop from fig2 to fig1 and move the dropdown menu to Figure 1—with only that simple change.

Figure 2 shows the file `twofigs.m`, which produces the GUI windows in Figure 1.

At the beginning of `twoFigs`, the code first checks and possibly converts the data. Next, the code initializes the GUI, including placing the controls and setting their properties, such as position. The callback functions for two of the controls, the slider and the plane selector, are set to call the `updateResultFig` function when you interact with them. The initialization finishes with a call to the `updateResultFig` function.

Memory Efficiency

The code includes three functions: `twoFigs` (the main one), `updateResultFig`, and `checkRGB`. We can identify the functions in the file, and specifically their termination, because of their associated `end` statements (required for all functions in files containing nested functions).

The first executable statement in `twoFigs`, `checkRGB()`, calls that nested function in the same file. This function takes no inputs or outputs, but may alter the values of the RGB image that was input to `twoFigs`. Toward the bottom of the file, you can see `checkRGB`, indented for readability. This function checks to see if the input image is an integer type; if so, we convert the data to single.

The function `checkRGB` is *nested* inside the function `twoFigs`, by the structure of `end` statements in the file, and is indicated by the structure of `end` statements in the file (required) and the indentation. As a nested function, `checkRGB` can share the variables from `twoFigs`, the function inside which it is nested. If the array `RGB` were large and we were going to simply replace certain values in it, such as for a saturation calculation, for example

```
RGB(RGB > someValue) = someValue
```

we could do so while working on the original array without needing to make a copy, because it is shared with the main function. The file `twoFigs.m` contains a second nested function, `updateResultFig`.

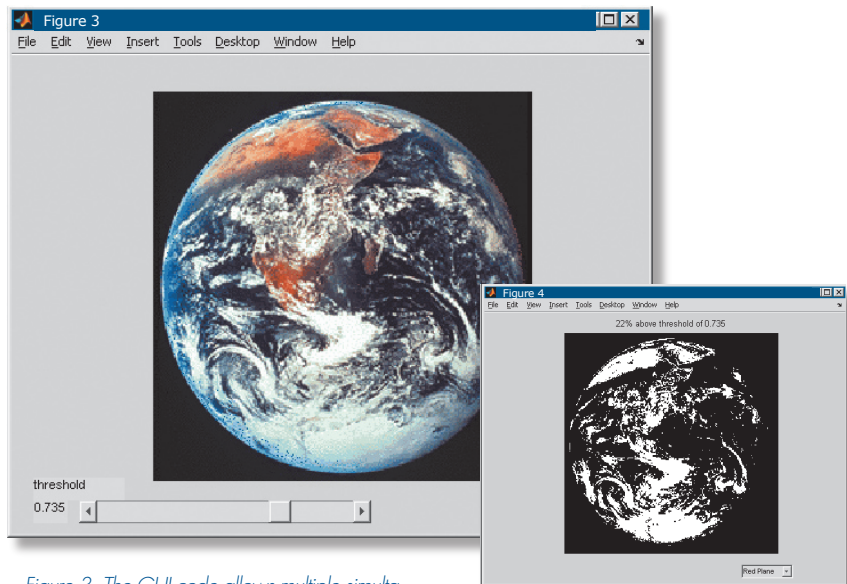


Figure 3. The GUI code allows multiple simultaneous images.

This function determines the plane of interest, queries the user setting for the threshold, and calculates the values that are above and below that threshold. `updateResultFig` figures out the percentage of the selected image plane that meets the threshold criterion.

The resulting calculations are displayed in a binary image annotated with the numeric results in the title. Because `updateResultFig` is a nested function, it has access to the workspace of the enclosing function, and therefore can find values for the threshold and the chosen plane, and can operate directly on the image data.

Scoping

When you interact with this GUI by moving the slider or selecting a new color plane, the action triggers the respective control's callback. These callbacks happen to be calls to the same nested function, `updateResultFig`, and share the same workspace. Because they point to common data, the callbacks are guaranteed to be robust.

To make all the action happen when you reposition the slider or change the color plane, we need to find out the threshold value and the plane. We can access these values by referencing the handles associated with their UI elements and reading the values. The handles we will use to access the values are available in

some of the variables from the main function. We can use these variables and the Handle Graphics® function `get` to find and use these parameters, without having to pass in the variables via the argument list.

Because these values are scalars and not large arrays, we do not worry about extracting both values, even though only one of them has changed; the overhead for refreshing the extra information is small. (If the stored values were large arrays or involved lengthy calculations, we might choose to optimize this step further.) After getting the values, we calculate the relevant output image and sum up the percentage of the image that meets the criteria. We finally display the new output values, including the output image and title, in the second window.

Nested functions provide a powerful alternative programming pattern that enables you to create more robust, elegant, and memory-efficient MATLAB applications. ◀◀

RESOURCES

- ▶ **MATLAB Central: Nested Functions in MATLAB**
www.mathworks.com/res/mlcnested
- ▶ **MATLAB Central: GUI Examples Using Nested Functions**
www.mathworks.com/res/mlcgui
- ▶ **MATLAB 7 Documentation: Nested Functions**
www.mathworks.com/res/nested
- ▶ **Book: MATLAB Programming for Engineers**
www.mathworks.com/res/book7380