Matlab for Engineers: Debugging - warnings and errors

Violeta Monasterio
Mauricio Villarroel

May 31st, 2012

Centre for Doctoral Training in Healthcare Innovation
Institute of Biomedical Engineering
Department of Engineering Science
University of Oxford

Supported by the RCUK Digital Economy Programme grant number EP/G036861/1
There are two types of errors

• Syntax errors: detected by matlab compiler

• Runtime errors: due to wrong logic used by the programmer:
  
  – Usually become apparent when one obtains erroneous or unexpected results
  
  – It is necessary to find the erroneous statements that caused the error: debugging
• Example:

```matlab
load ecgl; % loading ECG data
[hrv, R_t, R_amp, R_index, S_t, S_amp] = rpeakdetect(ecgl); % QRS detector
```

```
>> ex5_1
??? Error using ==> horzcat
CAT arguments dimensions are not consistent.

Error in ==> rpeakdetect at 120
left = find(diff([0 poss_reg'])==1); % remember to zero pad at start

Error in ==> ex5_1 at 2
[hrv, R_t, R_amp, R_index, S_t, S_amp] = rpeakdetect(ecgl); % QRS detector
```

```matlab
>>
```
Techniques to track down errors

• Use “Code Analyzer” (mlint)
• Removing / deleting semicolons
• Executing function as a script
  – The inputs can be fixed for which the results are known
• Keyboard statement
• Matlab debugger
Using the Keyboard statement

- Keyboard stops the execution
- Allows the programmer to examine the local workspace and execute statements from the command prompt (whos, size,...)
Using the debugger

- Breakpoints
- Workspace selection
- Execution control
## Debugging from the command line

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dbstop</code></td>
<td>set breakpoint</td>
</tr>
<tr>
<td><code>dbclear</code></td>
<td>clear breakpoint</td>
</tr>
<tr>
<td><code>dbclear all</code></td>
<td>clear all breakpoints</td>
</tr>
<tr>
<td><code>dbstop if</code></td>
<td>stop on warning, error or NaN/Inf</td>
</tr>
<tr>
<td><code>error</code></td>
<td>NaN/Inf generation</td>
</tr>
<tr>
<td><code>dbstep</code></td>
<td>single step execution</td>
</tr>
<tr>
<td><code>dbstep in</code></td>
<td>step into a function</td>
</tr>
<tr>
<td><code>dbstep nlines</code></td>
<td>execute one or more lines</td>
</tr>
<tr>
<td><code>dbcont</code></td>
<td>continue execution</td>
</tr>
<tr>
<td><code>dbquit</code></td>
<td>quit debugging</td>
</tr>
<tr>
<td><code>dbstack</code></td>
<td>list function call stack</td>
</tr>
<tr>
<td><code>dbstatus</code></td>
<td>list all breakpoints</td>
</tr>
<tr>
<td><code>dbtype</code></td>
<td>list M-file with line numbers</td>
</tr>
<tr>
<td><code>dbdown / dbup</code></td>
<td>change local workspace down / up</td>
</tr>
</tbody>
</table>
Preventing common errors

• Avoid dividing by zero: \( \frac{1}{x} \rightarrow \frac{1}{x + \text{eps}} \)

• Default \textbf{else for if-elseif,}

\textbf{Default otherwise for switch-case}

```java
if condition1,
    statement1;
elseif condition2
    statement2;
...
elseif conditionN,
    statementN;
else default_statement
end
```
Preventing common errors

• Check inputs: number, type, size
  – assume default values where possible
  – if a required input is missing: throw error and exit (assert)

function write2file(varargin)
min_inputs = 2;
assert(nargin >= min_inputs, 'You must call function...
  %s with at least %d inputs', mfilename, min_inputs)

infile = varargin{1};
assert(ischar(infile), 'First argument must be a filename.')

fid = fopen(infile, 'w');
assert(fid > 0, 'Cannot open file %s for writing', infile)

fwrite(fid, varargin{2});
Handling errors

• Try / catch block

```matlab
try
    [hrv, R_t] = rpeakdetect(ecg1); % QRS detector
catch err
    if(strcmp(err.identifier, ...
        'MATLAB:catenate:dimensionMismatch'))
        try % try again with transposed input
            [hrv, R_t] = rpeakdetect(ecg1);
        catch
            rethrow(err) % rethrow original error
        end
    end
end
```
Keeping things tidy (*onCleanup*)

- Leave your program environment in a clean state:
  - close any open files
  - restore the MATLAB path
  - set the working folder back to its default
  - make sure global variables are in the correct state

```matlab
function openFileSafely(fileName)
    fid = fopen(fileName, 'r');
    c = onCleanup(@()fclose(fid));
    s = fread(fid);
    
end
```
Other tools

- In the editor -> Tools -> Compare against
  - compares M-files, MAT-files and directories
Practice: QRS detector
(practice_5.m)

1. Low-pass filter
2. Derivation
3. Squaring
4. Integration
5. Thresholding