MATLAB Tips and Tricks
You Need to Know

Yair Altman

UndocumentedMatlab.com

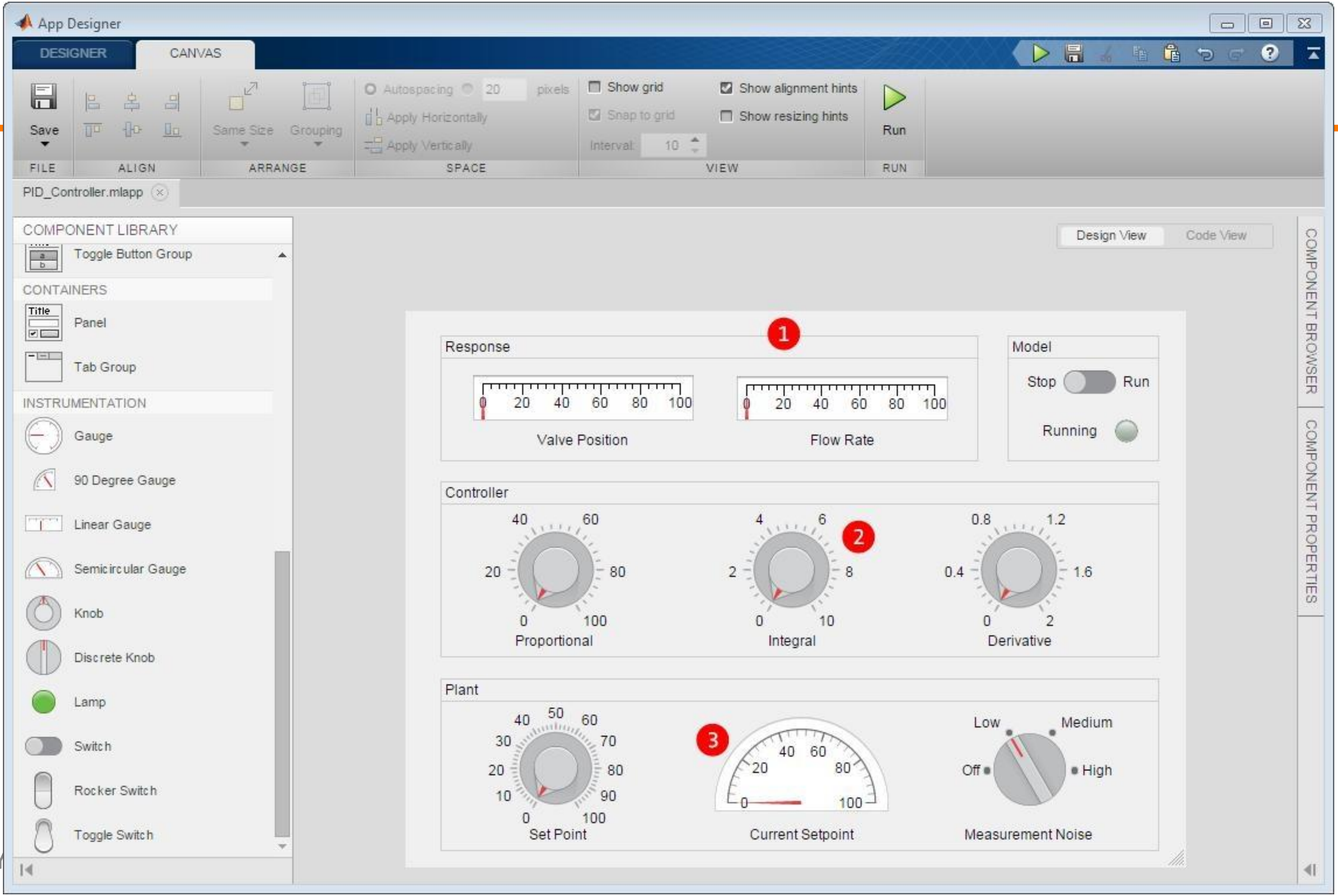
altmany@gmail.com

Key takeaways

- Tips and tricks to improve productivity with MATLAB
  - Usability and interactivity
  - Visualizing large data sets
  - Run-time performance
  - MATLAB Central
- Some tips/tricks are easy to apply and have great effect
- MATLAB can be used to create professional-quality user-facing applications

New App Designer in MATLAB

- GUIDE replacement in native MATLAB
- Web-based figures
- Great-looking widgets/dials/knobs
- Integrated layout designer, code editor
- Enables using CSS for professional-looking UI

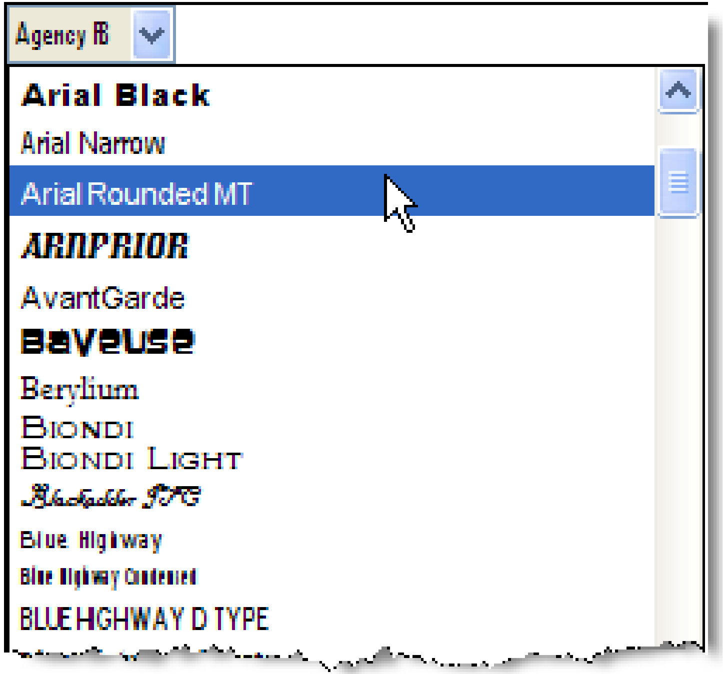

App Designer
DESIGNER
CANVAS
Save
FILE
ALIGN
Same Size
Grouping
ARRANGE
Autospacing
20
pixels
Apply Horizontally
Apply Vertically
SPACE
Show grid
Snap to grid
Interval:
10
Show alignment hints
Show resizing hints
VIEW
Run
RUN
PID_Controller.mlapp
COMPONENT LIBRARY
Toggle Button Group
CONTAINERS
Panel
Tab Group
INSTRUMENTATION
Gauge
90 Degree Gauge
Linear Gauge
Semicircular Gauge
Knob
Discrete Knob
Lamp
Switch
Rocker Switch
Toggle Switch
Design View
Code View
COMPONENT BROWSER
COMPONENT PROPERTIES
1
Response
Valve Position
Flow Rate
Model
Stop
Run
Running
Controller
2
Proportional
Integral
Derivative
Plant
3
Set Point
Current Setpoint
Low
Medium
Off
High
Measurement Noise

Old non-web GUI

- Programmed using GUIDE or straight m-code
- Based on Java Swing UI components
- Built-in support for HTML formatting
  (no Java knowledge/programming required!)
- Enables using Java for polished professional UI
- Might become unsupported in a future release

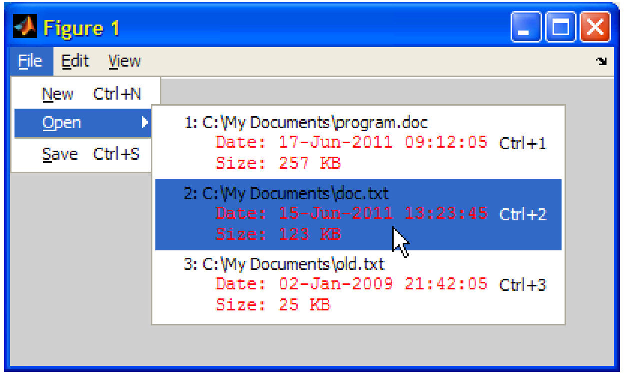
HTML formatting of uicontrol labels

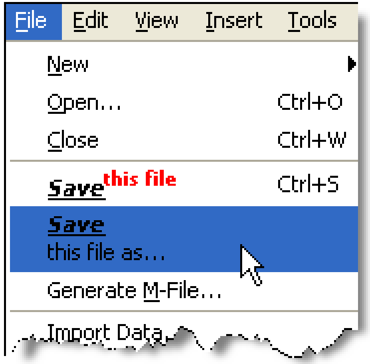
```
fontStr = @(font) ['<html><font face="' font '">' font];
htmlStr = cellfun(fontStr, listfonts, 'uniform',false);
uicontrol('style','popupmenu', 'string',htmlStr, 'pos',[20,350,60,20]);
```



HTML formatting of menu labels

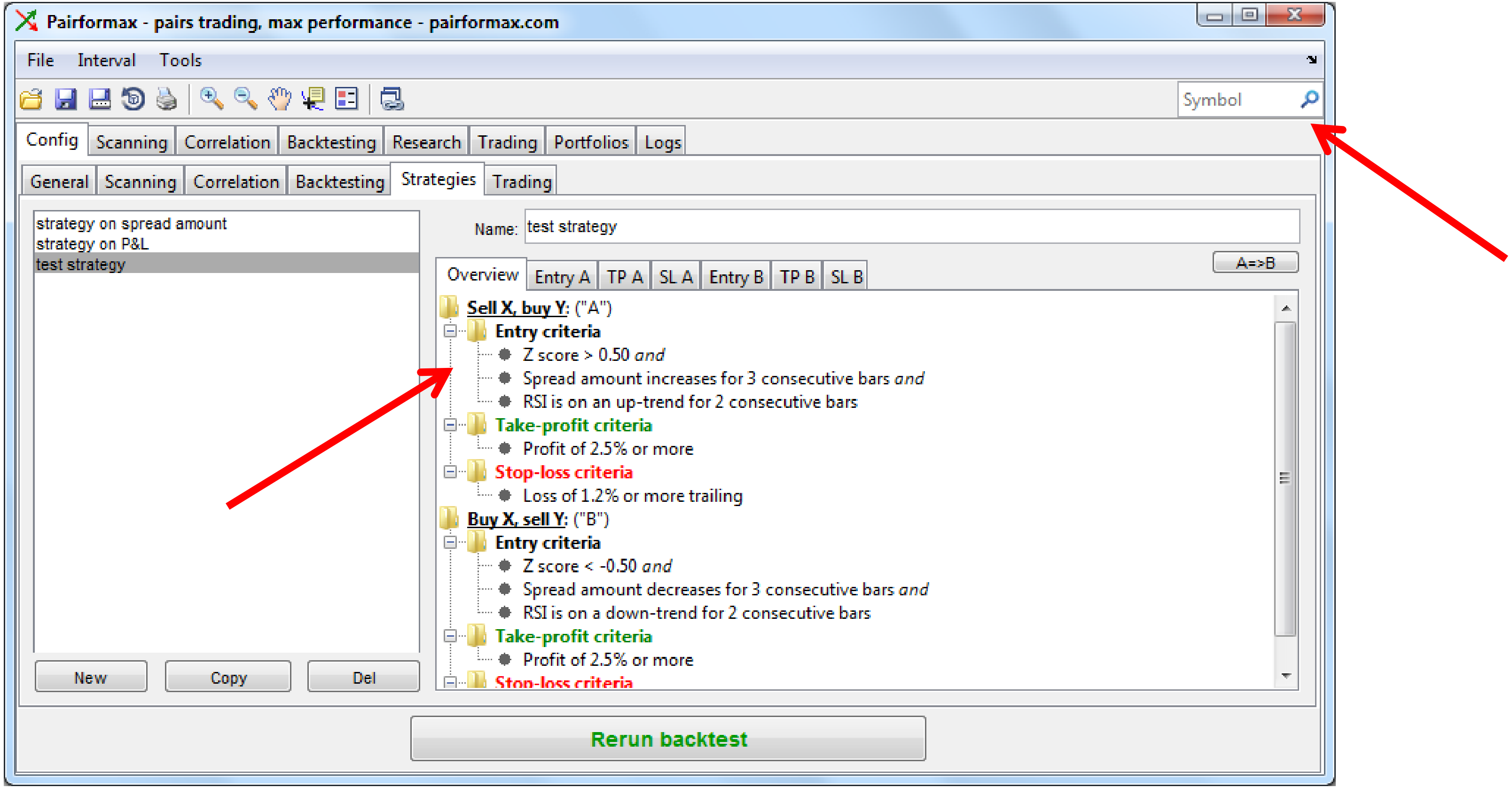
```
set(hMenuItem, 'Label',['<html>2: C:\My Documents\doc.txt<br>'...
    '<font size="-1" face="Courier New" color="red">   '...
    'Date: 15-Jun-2011 13:23:45<br>   Size: 123 KB']);
```






```
txt1 = '<html><b><u><i>Save</i></u>';
txt2 = '<font color="red"><sup>this file';
txt3 = '<br>this file as...';
set(findall(hFig,'tag','figMenuFileSave'),   'Label',[txt1,txt2]);
set(findall(hFig,'tag','figMenuFileSaveAs'), 'Label',[txt1,txt3]);
```

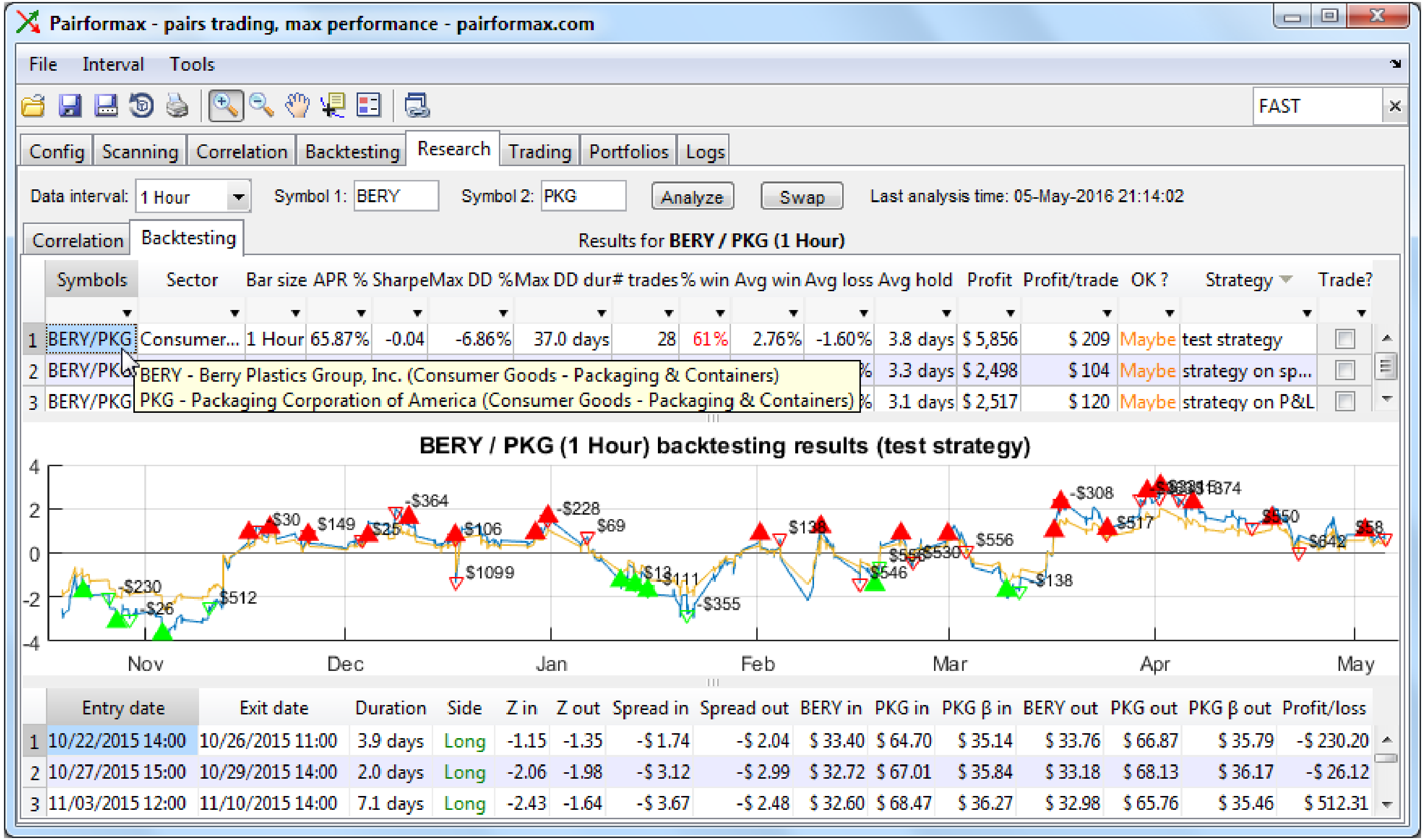
Polished UI using Java components



Visualizing complex data in a GUI

- General concepts
  - Focus user attention (summary, anomalies)
  - Hide less-important data by default
  - Combination of tables (quantitative) & charts (qualitative)
  - Enable data drill-down (e.g., heat-map)
- Enable easy sorting, filtering, searching
  - Pre-sort displayed data
- Summarize data in separate tabs, sub-tables
- Use *GUI Layout Toolbox (GUILT)*: large tabs, flex-panels
  - Excellent free toolbox on the MATLAB File Exchange

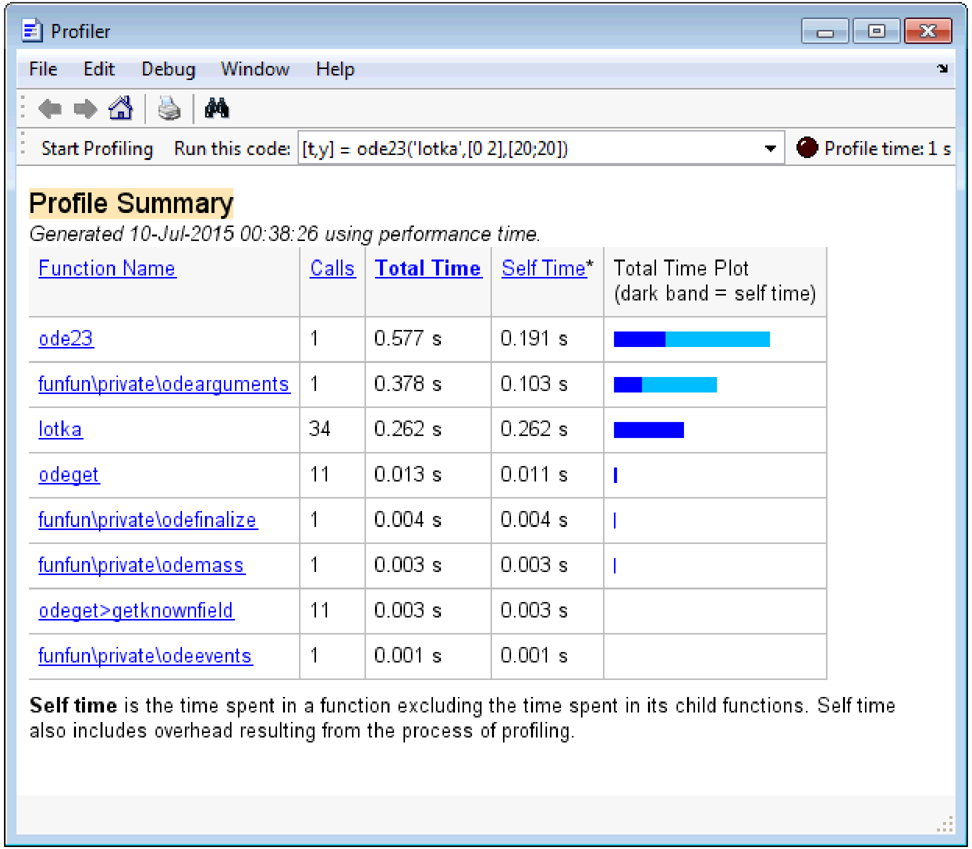
Real-world example



Run-time performance

1. MATLAB’s built-in Profiler tool
   - Very useful, but largely under-appreciated
2. General programming tips
   - Caching
   - Loop-invariant hoisting (LIH)
3. Parallelization
   - Minimize broadcast data
   - Vectorization vs. CPU/GPU parallelization
4. Graphics

Built-in Profiler tool in MATLAB



Profiler

File Edit Debug Window Help

Start Profiling Run this code: [t,y] = ode23('lotka',[0 2],[20;20]) Profile time: 1 s

Profile Summary

Generated 10-Jul-2015 00:38:26 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
ode23	1	0.577 s	0.191 s	
funfun\private\odearguments	1	0.378 s	0.103 s	
lotka	34	0.262 s	0.262 s	
odeget	11	0.013 s	0.011 s	
funfun\private\odefinalize	1	0.004 s	0.004 s	
funfun\private\odemass	1	0.003 s	0.003 s	
odeget>getknownfield	11	0.003 s	0.003 s	
funfun\private\odeevents	1	0.001 s	0.001 s	

Self time is the time spent in a function excluding the time spent in its child functions. Self time also includes overhead resulting from the process of profiling.

Caching example – *datestr*

```
% faster variant of datestr, for integer date values since 1/1/2000
function dateStrs = datestr2(dateVals, varargin)
    persistent dateStrsCache
    persistent dateValsCache
    if isempty(dateStrsCache)
        origin = datenum('1-Jan-2000');
        dateValsCache = origin:(now+100);
        dateStrsCache = datestr(dateValsCache, varargin{:});
    end
    [tf,loc] = ismember(dateVals, dateValsCache);
    if all(tf)
        dateStrs = dateStrsCache(loc,:);
    else
        dateStrs = datestr(dateVals, varargin{:});
    end
end  % datestr2
```

Caching example – *datestr*

```
% Prepare a 1000-vector of dates, starting 3 years ago until today
>> dateVals = fix(now)+(-1000:0);

% Run the standard datestr function → 50mS
>> tic; s1=datestr(dateVals); toc
⇒ Elapsed time is 0.049089 seconds.

>> tic; s1=datestr(dateVals); toc
⇒ Elapsed time is 0.048086 seconds.

% Now run datestr2 function → 0.3 mS (x150 faster)
>> tic; s2=datestr2(dateVals); toc
⇒ Elapsed time is 0.222031 seconds. % cache preparation: 222 mS

>> tic; s2=datestr2(dateVals); toc
⇒ Elapsed time is 0.000313 seconds. % subsequent calls: 0.3 mS
```

Loop-invariant hoisting (LIH)

```
for iter = 1 : 1000
    newData = constantFunction();
    result(iter) = max(max(newData)) + rand(1);
end

newData = constantFunction();
maxNewData = max(max(newData));
for iter = 1 : 1000
    result(iter) = maxNewData + rand(1);
end

result = maxNewData + rand(1,1000); %vectorized
```

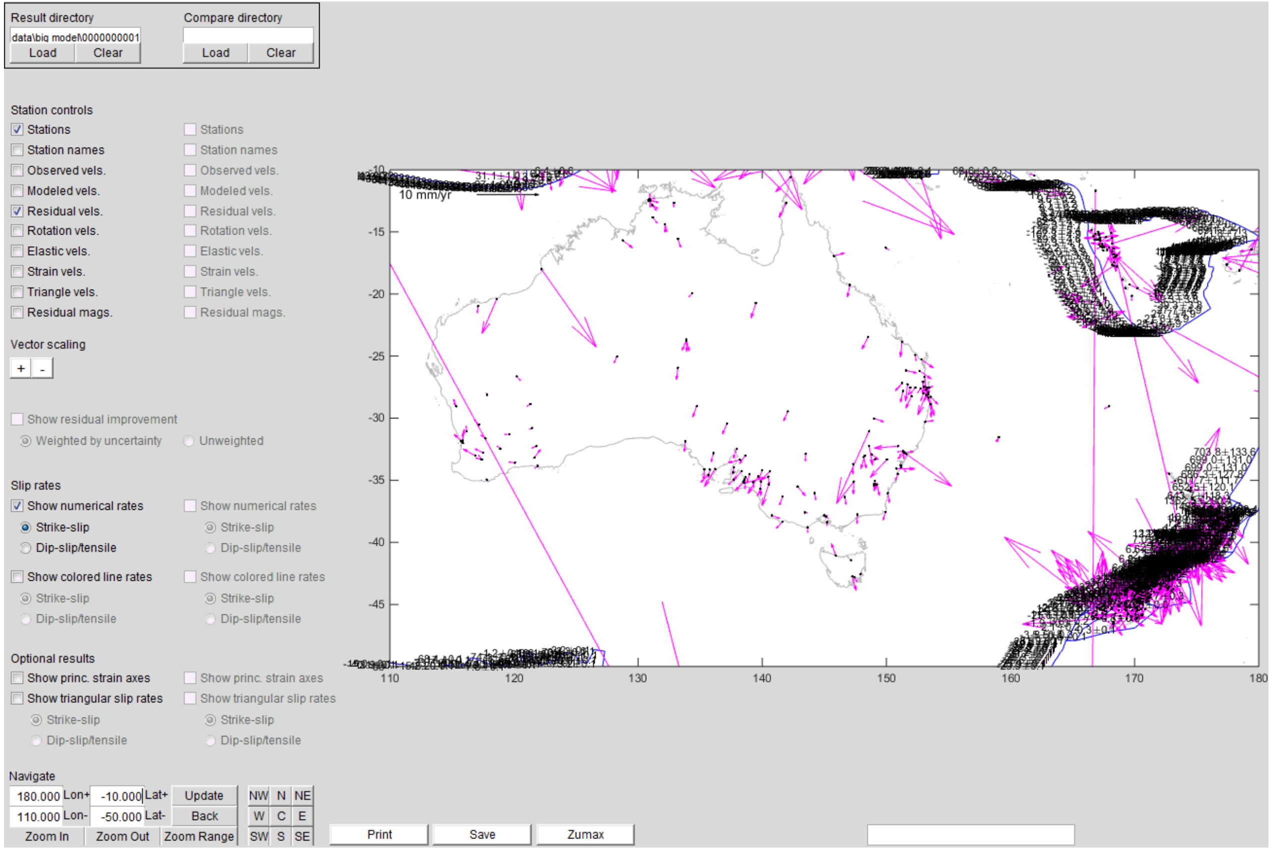
Parallel processing

- Use built-in implicit parallelization (vectorization)
- Use explicit parallelization with the Parallel and Distributed Computing toolboxes
- Control # workers: `parpool(feature('numcores'))`
- Minimize amount of broadcast data
- Have realistic speedup expectations (*Amdahl's law*)

Graphics performance

- Install latest graphics driver on your computer
- Reduce the number of graphic objects (plot markers, non-visible lines/patches)
- Multi-segment lines using NaN
- Update existing graphic objects, don't re-plot
- Minimize `drawnow` calls, use '`limitrate`'

Real-world example



Multi-segment lines using NaN

```
% Original (slow) code: 33 secs
line([lons1'; lons2'], [lats1'; lats2']);

% Faster code: limit the display to the axes limits
hAxes = handle(gca);   %trick: make hAxes.XLim work on old MATLABs
lonLimits = hAxes.XLim;
latLimits = hAxes.YLim;
isOk = (within(lons1,lonLimits) | within(lons2,lonLimits)) & ...
       (within(lats1,latLimits) | within(lats2,latLimits));
line([lons1(isOk)', lons2(isOk)'], [lats1(isOk)', lats2(isOk)']);

% Multi-segment line using NaNs (fastest): 0.6 secs
lons = [lons1'; lons2'; nan(1,numel(lons2))];
lats = [lats1'; lats2'; nan(1,numel(lats2))];
line(lons(:), lats(:));
```

Update existing graphic objects

```
data = [0];
while someCondition()
    newValue = getNewValue();
    data = [data, newValue];
    plot(hAxes, data);
    drawnow
end

% Faster, less flicker: update existing graphic object
hLine = plot(hAx, 0,0);  % initial line
while someCondition()
    hLine.XData(end+1) = hLine.XData(end) + 1;
    hLine.YData(end+1) = getNewValue();
    % alternatively: set(hLine, 'XData',xVals, 'YData',yVals)
    drawnow limitrate
end
```

MATLAB Central

- http://mathworks.com/matlabcentral
- Huge repository of community content
  - Blogs
  - File Exchange
  - Usage examples
  - Forum (“Answers”, newsgroup)
  - Coding challenges (“Cody”)
- Use the ratings for high-quality content

Conclusions

- MATLAB is **<u>not</u>** inherently ‘slow’
- MATLAB GUI is **<u>not</u>** inherently ‘simplistic’
- MATLAB can be used to create professional-quality user-facing applications, while enjoying MATLAB’s benefits (RAD, functionality, reliability)
- As with everything in life, there are tradeoffs:
  - For better performance, invest extra code development time
  - For better appearance, invest extra design time
  - For better usability, use Java controls