Scripting without Scripts:
A User-Friendly Integration of R, Python, Matlab and Groovy into KNIME

Felix Meyenhofer
Technology Development Studio

3. March 2011
4th KNIME Users Group Meeting and Workshop
Outline

- Motivation
- Architecture and design
- Scripting languages
- Generic R
- Prototyping
- Perspectives
Outline

Motivation
Architecture and design
Scripting languages
Generic R
Prototyping
Perspectives
Our situation
The actors

I’m a scientist.
I use Excel because I rely on a proper UI, but it does not scale, does not work well for all types of experiments, etc.
So I often need to go to see my data analyst.

I’m a data analyst.
I can script in R, Python, Matlab and a little Java.
I hate user interfaces!
Are data analysts lazy people?

A (typical) research project

- Experiment
- Lots of data
- Fancy data mining scripts
- A little Excel
- Great results
- Paper
Are data analysts lazy people?

A (typical) research project

• Experiment
• Lots of data
• Fancy data mining scripts
• A little Excel
• Great results
• Paper

Answer: No, but staff-ratio = 8:1
Can **KNIME** fix the problem?

+ It is able to handle big data sets
+ Has ready to use data interfaces (txt, xls, sql...)
+ It’s modular architecture allows customization

- Native KNIME visualisation are insufficient (for our purpose)
- It lacks specific methods and tool
- node development requires java-geeks and takes time, where the data analysis requirements evolve quickly along with the experiments

In contrast:

• Scripting solutions can be quickly developed but are not end-user ready
• Scripting allows rapid prototyping of methods and tools
What if data analysts could write their scripts as usual, AND users could somehow access these within a well designed framework using a graphical interface?
The basic components

Motivation
Architecture and design
Scripting languages
Generic R
Prototyping
Perspectives
Reduction to the necessary
Template management

- Local or remote template repositories
- Plain-text template definition files
- Hierarchically organized
- Previews for visualization templates
The (new) situation

- Facilitated knowledge propagation within and among research groups
- Facilitated knowledge (work) preservation with the template system
- Applicable form different levels of expertise

Users
- Use Knime and templates
- Be happy!

Advanced Users
- Customize templates in-place
- Use flow variables and loops

Data Analysts
- Write scripts to fill Knime-gaps
- Evolve scripts into new templates

Developers
- Provide Knime-nodes for most popular templates
The (new) situation

- Facilitated knowledge propagation within and among research groups
- Facilitated knowledge (work) preservation with the template system
- Applicable form different levels of expertise

Motivation
Architecture and design
Scripting languages
Generic R
Prototyping
Perspectives

Users
- Use Knime and templates
- Be happy!

Advanced Users
- Customize templates in-place
- Use flow variables and loops

Data Analysts
- Write scripts to fill Knime-gaps
- Evolve scripts into new templates

Developers
- Provide Knime-nodes for most popular templates
Scripting nodes vs. conventional nodes

- Less integrated than native nodes
- ‘Real’ nodes are likely to be (much) faster
- ‘Real’ nodes scale better/work with huge data-sets
- ‘Real’ nodes can be updated by updating Knime
- ‘Real’ node-development requires 10x more resources
From templates to nodes

+ Templates become updateable
+ Rapid R prototyping
+ Trivial deployment (1node/min) into end-user ready nodes
From templates to nodes

- Templates become updateable
- Rapid R prototyping
- Trivial deployment (1 node/min) into end-user ready nodes
Scripting language support

R

Rserve as backend
Scripting language support

Rserve as backend

mpicbg-matlab web-client (needs a floating license!)
Scripting language support

- **R**
  - Rserve as backend

- **MATLAB**
  - mpicbg-matlab web-client (needs a floating license!)

- **Python**
  - mpicbg-python backend
## Scripting language support

<table>
<thead>
<tr>
<th>Scripting Language</th>
<th>Backend/Client</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>Rserve</td>
<td>Rserve as backend</td>
</tr>
<tr>
<td>MATLAB</td>
<td>mpicbg-matlab web-client</td>
<td>(needs a floating license!)</td>
</tr>
<tr>
<td>Python</td>
<td>mpicbg-python backend</td>
<td></td>
</tr>
<tr>
<td>Groovy</td>
<td>Using Knime API but in a dynamically complied environment</td>
<td></td>
</tr>
</tbody>
</table>
Scripting language support

<table>
<thead>
<tr>
<th>Scripting Language</th>
<th>Backend</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>Rserve</td>
<td>Uses mpicbg-matlab web-client (needs a floating license!)</td>
</tr>
<tr>
<td>MATLAB</td>
<td>mpicbg-matlab backend</td>
<td>Using Knime API but in a dynamically compiled environment</td>
</tr>
<tr>
<td>Python</td>
<td>mpicbg-python backend</td>
<td></td>
</tr>
<tr>
<td>Groovy</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Server infrastructure makes it easier for maintenance. (Power users can use a local instance)
What else?

All that is necessary is (1) a table conversion mechanism, (2) the possibility to invoke a script with the converted table as argument, and (3) a way to convert the results back into a table.
# Feature overview

<table>
<thead>
<tr>
<th>Features</th>
<th>R</th>
<th>MATLAB</th>
<th>Python</th>
<th>Groovy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• STATISTICS</td>
<td>• Statistics</td>
<td>• BIOINFORMATICS</td>
<td>• Java prototyping</td>
</tr>
<tr>
<td></td>
<td>• Visualization</td>
<td>• IMAGE PROCESSING</td>
<td>• Statistics</td>
<td>• regexp</td>
</tr>
<tr>
<td></td>
<td>• Handy data-types</td>
<td>• Visualization</td>
<td>• (Image Processing)</td>
<td>• ...</td>
</tr>
<tr>
<td></td>
<td>Exponential growth in packages</td>
<td>Many people know it (#29) *</td>
<td>#4 programming language *</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Memory management</td>
<td>rather slow without grid computing toolbox</td>
<td>interface</td>
<td>syntax (just joking)</td>
</tr>
</tbody>
</table>

• Knime is focused on table transformations
• Some problems do not fit into this scheme
• Solution: Generic R nodes
The MATLAB integration trio

Open in Matlab

Matlab Plot

Matlab Snippet
The MATLAB integration trio

Open in Matlab

Matlab Plot

Matlab Snippet

Loading data from KNIME ...

The data is available as the following variables in the Workspace:

'kn' is a dataset containing the table.

'names' is a structure containing column header information.

If the data was updated by KNIME by re-executing the OpenInMatlab node while it's checkbox 'Run a new MATLAB instance' in its configuration dialog was unticked, re-load the KNIME data table with the command:

[kn names] = loadKNIMETable('knime-table-dump.tmp', 'dataset')
The MATLAB integration trio

Open in Matlab

Matlab Plot

Matlab Snippet

Friday, March 4, 2011
The MATLAB integration trio

Open in Matlab

Matlab Plot

Matlab Snippet

Motivation
Architecture and design
Scripting languages
Generic R
Prototyping
Perspectives
The MATLAB integration trio

Open in Matlab

Matlab Plot

Matlab Snippet

Dialog – 2:9 – Matlab Snippet

Input attributes

D Universe_0_0
D Universe_0_1
S Cluster Membership

[p table stats] = anovai(kIn.Universe_0_0, kIn.ClusterMembership, 'off');
[c,m,h,gname] = multcompare(stats,'display','off');

mout = dataset();
mout.groupEntity1 = gnames(c(:,1));
mout.groupEntity2 = gnames(c(:,2));
mout.lower_cl = c(:,3);
mout.differenceOfMeans = c(:,4);
mout.upper_cl = c(:,5);
mout.different = double(sum(sign(c(:,[3,5])),2) == 0);

help
The MATLAB integration trio

Open in Matlab

Matlab Plot

Matlab Snippet

<table>
<thead>
<tr>
<th>Row ID</th>
<th>groupEntity1</th>
<th>groupEntity2</th>
<th>lower_ci</th>
<th>differenceOfMeans</th>
<th>upper_ci</th>
<th>different</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Cluster_0</td>
<td>Cluster_1</td>
<td>0.361</td>
<td>0.379</td>
<td>0.397</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>Cluster_0</td>
<td>Noise</td>
<td>0.236</td>
<td>0.254</td>
<td>0.273</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Cluster_1</td>
<td>Noise</td>
<td>-0.143</td>
<td>-0.125</td>
<td>-0.106</td>
<td>1</td>
</tr>
</tbody>
</table>
The R integration trio

Open in R

R Plot

R Snippet
The R integration trio

Open in R

R Plot

R Snippet

The R Console

> ls()
[1] "kIn"
> kIn[1:5]

<table>
<thead>
<tr>
<th>Universe_0.0</th>
<th>Universe_0.1</th>
<th>Universe_1.0</th>
<th>Universe_1.1</th>
<th>Cluster Membership</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0.003889515</td>
<td>0.7878118</td>
<td>0.7160954</td>
<td>0.5815642</td>
<td>Cluster_0</td>
</tr>
<tr>
<td>2 0.239316100</td>
<td>0.8990245</td>
<td>0.3723350</td>
<td>0.8991247</td>
<td>Cluster_0</td>
</tr>
<tr>
<td>3 0.263042111</td>
<td>0.9272407</td>
<td>0.2520652</td>
<td>0.7557655</td>
<td>Cluster_0</td>
</tr>
<tr>
<td>4 0.136886655</td>
<td>0.7868167</td>
<td>0.3661857</td>
<td>0.3005205</td>
<td>Cluster_0</td>
</tr>
<tr>
<td>5 0.044944104</td>
<td>0.6044932</td>
<td>0.6935082</td>
<td>0.6192118</td>
<td>Cluster_0</td>
</tr>
</tbody>
</table>
The R integration trio

Open in R

R Plot

R Snippet

Commands
?ggplot
ggplot(kin)
ggplot(kin[,1], fill = kin[,5])
params = which(names(kin))
params = which(names(kin))
params = which(names(kin)) %in% "Universe_"
params = which(names(kin)) %in% "Cluster"
ggplot(plotData, aes_string(x = names(plotData))
ggplot(kin, aes_string(x = names(kin))[param])
ggplot(kin, aes_string(x = names(kin))[param])
ompgplot(kin, aes_string(x = names(kin)[param])

Delete entry  Clear History
Load History  Save History
The R integration trio
The R integration trio

Open in R

R Plot

R Snippet

```r
selectedColumns = match(c("universe_0_0"), names(kin));
groupingColumn = match(c("Cluster Membership"), names(kin));
method = c("whisker");

# Get the groups
groups = unique(kin[,groupingColumn])

for (g in length(groups)) {
  # Get the measurements from a specific group
  tableSubset <- subset(kin, kin[,groupingColumn]==groups[g])
  subsetKeeper <- rep(TRUE, each=nrow(tableSubset))
  for (c in length(selectedColumns)) {
    # Get the column vector from the group
    columnSubset <- tableSubset[, selectedColumns[c]]
    # Determine the bounds
    if (method == "whisker") {
      output <- boxplot.stats(columnSubset)
      boxinfo <- output[2]
      lowerBound <- boxinfo[2]
      upperBound <- boxinfo[5]
    } else if (method == "meanSD") {
      std <- sd(columnSubset)
      mne <- mean(columnSubset)
      lowerBound <- mne + 3*std
      upperBound <- mne + 3*std
    }
    # Check the data points if there inside the outlier boundaries
    subsetKeeper <- subsetKeeper & (lowerBound <= columnSubset) & (columnSubset <= upperBound)
    }
  if (g == 1) {
    subsetKeeper <- subsetKeeper
  } else {
    subsetKeeper <- c(subsetKeeper, subsetKeeper)
  }
}
out = subset(kin, subsetKeeper)
```
The R integration trio

Open in R

R Plot

R Snippet
The Python integration trio

Open in Python

Python Plot

Python Snippet
The Python integration trio

Open in Python

Python Plot

Python Snippet

The Python integration trio
Creating a Python plot template
Creating a Python plot template

Stacked Graphs – Geometry & Aesthetics
Lee Byron & Martin Wattenberg

Abstract — In February 2008, the New York Times published an unusual chart of box office revenues for 7500 movies over 21 years. The chart was based on a similar visualization, developed by the first author, that displayed trends in music listening. This paper describes the design decisions and algorithms behind these graphics, and discusses the reaction on the Web. We suggest that this type of complex layered graph is effective for displaying large data sets to a mass audience. We provide a mathematical analysis of how this layered graph relates to traditional stacked graphs and to techniques such as ThemeRiver, showing how each method is optimizing a different "energy function". Finally, we discuss techniques for coloring and ordering the layers of such graphs. Throughout the paper, we emphasize the interplay between considerations of aesthetics and legibility.

Index Terms — Streamgraph, ThemeRiver, listening history, last.fm, aesthetics, communication-minded visualization, time series.

1 INTRODUCTION

In February 2008, The New York Times stirred up a debate. The famous newspaper is no stranger to controversy, but this time the issue was not political bias or anonymous sources—it was an unusual graph of movie ticket sales. On information design blogs, opinions of the chart ranged from “fantastic” to “unsavory.” Meanwhile, on other online forums and blogs, hundreds of people posted insights and questions spurred by the visualization.

The story of the design process and algorithms behind this engaging (and polarizing) graphic makes an illuminating case study in the role of aesthetics in visualization design. Our goal in this paper is to tell this story, while documenting and analyzing the specific geographic and accompanying online interactive visualization of the box office revenue for 7500 movies over a 21-year period.

In this paper we first provide a case study of the New York Times and last.fm visualizations. We pay special attention to the response on the web and the role of aesthetics in the appeal of visualizations. Second, we perform a detailed analysis of the algorithms that define these graphs. A key theme is the role of aesthetics in visualization design, and the process and trade-offs necessary to create engaging information graphics.

2 RELATED WORK
Creating a Python plot template

Recipe 576633: Stacked graphs using matplotlib (Python)

Creates stacked graphs (sometimes known as stream graphs, apparently) as recommended by Byron and Wattenberg,
http://www.leebyron.com/else/streamgraph/download.php?
file=stackedgraphs_byron_wattenberg.pdf

```python
# Author: Anand Patil
# License: MIT License

import matplotlib.pyplot as plt
import numpy as np

def symmetric(sorted_streams, stream_bounds):
    """Symmetric baseline""
    lb, ub = np.min(stream_bounds[:,0,:], axis=0), np.max(stream_bounds[:,1,:], axis=0)
    return .5*(lb+ub)

def pos_only(sorted_streams, stream_bounds):
    """Lumps will only be positive""
    lb, ub = np.min(stream_bounds[:,0,:], axis=0), np.max(stream_bounds[:,1,:], axis=0)
    return lb

def zero(sorted_streams, stream_bounds):
    """Zero baseline""
    return np.zeros(stream_bounds.shape[2])

def min_weighted_wiggles(sorted_streams, stream_bounds):
    """Baseline recommended by Byron and Wattenberg""
    lb, ub = np.min(stream_bounds[:,0,:], axis=0), np.max(stream_bounds[:,1,:], axis=0)
```

Python, 109 lines
Creating a Python plot template

```python
# name: Steam plot
# author: Felix Meyenhofer (python script from Anand Patil)
# category: experimental

Creates a steam plot (stacked histogram)

Inputs:
Choose random variables (columns)

```
Creating a Python plot template

```python
# name: Steam plot
# author: Felix Meyenhofer (python script from Anand Patil)
# category: experimental

Creates a steam plot (stacked histogram)

Inputs:
Choose random variables (columns)

###

```html
<rgg>

<!-- 1. Title and short description -->

<!-- 2. Configuration -->

# 1. Parameter selection
# select the readouts that span the phenotypic space
keys = <panellistbox label="Features of interest" items="$\$NUM_ATTRIBUTES$" visible-row-count="6" span="full"/>

#bins = int(<textfield label="Number of bins" var="" data-type="number" default-value="30" size="10" span="full"/>)

colorSequence = <combobox items="random, linear" label="color chooser"/>

# Script
<![CDATA[

]]>

</rgg>
```
Creating a Python plot template

---
# name: Steam plot
# author: Felix Meyenhofer (python script from Anand Patil)
# category: experimental

Creates a steam plot (stacked histogram)

Inputs:
Choose random variables (columns)

```xml
<rgg>

<!-- 1. Title and short description -->
<h3 text="Steam graph" alignment="center" span="full"/>  
<separator label="Description" span="full"/>
<labelarea span="full">Generate a steam graph</labelarea>
<gaprow height="1"/>

<!-- 2. Configuration-->
<separator label="Options" span="full"/>
<gaprow height="2"/>

# 1. Parameter selection
<group>
# 1) select the readouts that span the phenotypic space
keys = <panellistbox label="Features of interest" items="$NUM_ATTRIBUTES$" visible-row-count="6" span="full"/>
Nbins = int(<textfield label="Number of bins" var="" data-type="number" default-value = "30" size="10" span="full"/>)
colorSequence = <combobox items="random,linear" label="color chooser"/>
</group>

# Script
<![CDATA[

]]></rgg>
```
# name: Steam plot
# author: Felix Meyenhofer (python script from Anand Patil)
# category: experimental

Creating a steam plot (stacked histogram)

Inputs:
Choose random variables (columns)

```python
# Script
<![CDATA[

```
Creating a Python plot template

```plaintext
# name: Steam plot
# author: Felix Meyenhofer (python script from Anand Patil)
# category: experimental

Creates a steam plot (stacked histogram)

Inputs:
Choose random variables (columns)

```
Creating a Python plot template

```plaintext
# name: Steam plot
# author: Felix Meyenhofer (python script from Anand Patil)
# category: experimental

Creates a steam plot (stacked histogram)

Inputs:
Choose random variables (columns)

```
<rgg>

<!-- 1. Title and short description -->

<h3 text="Steam graph" aligment="center" span="full"/>
<separator label="Description" span="full"/>
<labelarea span="full">Generate a steam graph</labelarea>
<gaprow height="1"/>

<!-- 2. Configuration-->

<separator label="Options" span="full"/>
<gaprow height="2"/>

# 1. Parameter selection
<group>
# 1) select the readouts that span the phenotypic space
keys = <panellistbox label="Features of interest" items="$$NUM_ATTRIBUTES$$" visible-row-count="6" span="full"/>
Nbins = int(<textfield label="Number of bins" var="" data-type="number" default-value = "30" size="10" span="full"/>)
colorSequence = <combobox items="random,linear" label="color chooser"/>
</group>

# Script
<![CDATA[

]]>

</rgg>
```
for stream in sorted_streams[2::]:
    side *= -1
    if side==1:
        stream_bounds.append(np.vstack((stream_bounds[-2][1], stream_bounds[-2][1]+stream)))
    else:
        stream_bounds.append(np.vstack((stream_bounds[-2][0]-stream, stream_bounds[-2][0])))
stream_bounds = np.array(stream_bounds)

# Compute baseline
baseline = baseline_fn(sorted_streams, stream_bounds)

# Choose colors
t_poly = np.hstack((t,t[::-1]))
if color_seq=='linear':
    colors = np.linspace(0,1,streams.shape[1])
elif color_seq=='random':
    colors = np.random.random(size=streams.shape[1])
else:
    raise ValueError, 'Color sequence %s unrecognized' %color_seq

# Plot
pl.axis('off')
for i in xrange(len(stream_bounds)):
    bound = stream_bounds[i]
    color = cmap(colors[i])
    pl.fill(t_poly, np.hstack((bound[0]-baseline,(bound[1]-baseline)][::-1])), facecolor=color,

# Demo
if __name__ == '__main__':
    v = []
    for key in keys:
        v.extend(kIn[key])
        mi = min(v)
        ma = max(v)
        dsets = []
        for key in keys:
            c,s,trash = pl.hist(kIn[key], Nbins, range=(mi, ma))
            dsets.append(c)
        pl.clf()
        stacked_graph(dsets, baseline_fn=symmetric, color_seq=colorSequence)
    pl.show()
Creating a Python plot template

```python
for stream in sorted_streams[2:]:
    side *= -1
    if side==1:
        stream_bounds.append(np.vstack((stream_bounds[-2][1], stream_bounds[-2][1]+stream)))
    else:
        stream_bounds.append(np.vstack((stream_bounds[-2][0]-stream, stream_bounds[-2][0])))

stream_bounds = np.array(stream_bounds)

# Compute baseline
baseline = baseline_fn(sorted_streams, stream_bounds)

# Choose colors
t_poly = np.hstack((t,t[::-1]))
if color_seq=='linear':
    colors = np.linspace(0,1,streams.shape[1])
elif color_seq=='random':
    colors = np.random.random(size=streams.shape[1])
else:
    raise ValueError, 'Color sequence %s unrecognized' % color_seq

# Plot
pl.axis('off')
for i in xrange(len(stream_bounds)):
    bound = stream_bounds[i]
    color = cmap(colors[i])
    pl.fill(t_poly, np.hstack((bound[0]-baseline,(bound[1]-baseline)[::-1])), facecolor=color,
            edgecolor=color, alpha=0.5)

# Demo
if name == 'main':
    v = []
    for key in keys:
        v.extend(kIn[key])
    mi = min(v)
    ma = max(v)
    dsets = []
    for key in keys:
        c,s,trash = pl.hist(kIn[key], Nbins, range=(mi, ma))
        dsets.append(c)
        pl.clf()
    stacked_graph(dsets, baseline_fn=symmetric, color_seq=colorSequence)
    pl.show()

}}><rgg>
```
Creating a Python plot template

Python Plot

- Template Categories
  - experimental
    - Steam plot

Description

Creates a steam plot (stacked histogram)
Inputs:
Choose random variables (columns)

Preview (Double-click to enlarge)

Author: Felix Meyenhofer (python script from Anand Patil)
Provides User Interface:

Use this template
Creating a Python plot template
Creating a Python plot template
Creating a Python plot template
Question to the audience:

Is this KNIME or R?
Question to the audience:

Is this KNIME or R?

95%
Our experience so far:

I love templates, because they provide me the exact tools I need. And my data analyst can create them almost instantaneously.

I love templates, because they can be used to quickly make custom solutions accessible for my scientists.
Additional features of our scripting integrations:

- Better editor (undo, redo)
- Better attribute name insertion
- Preserve column names
- UI templates + centralized user template repositories
- Worker engine can be run locally or on a centralized server
- Dynamic repainting of plots
- OpenIn* nodes for rapid prototyping
- Faster table conversion (~2x for R)
- Flow-variable support in scripts
- Generic R nodes to process arbitrary data structures

Additional scripting languages:

- Python
- Matlab
Acknowledgements

Holger Brandl (Software developer for Bioinformatics)

Tom Haux (Software developer SWENG)

Martin Stöter (Scientist TDS)

Michael Berthold and the entire KNIME team