StatWeave: Flexible Software for Literate Programming in Statistics

Russ Lenth

Department of Statistics & Actuarial Science
The University of Iowa

Spring Research Conference
Atlanta, GA
May 19, 2008

Integrating code and documentation

Why?

- Literate programming
  - Code comments (primitive)
  - WEB (Knuth 1979) and relatives
- Documentation generation (e.g., **doc++**, **javadoc**)
- Reproducible statistical analysis
  - Research, consulting
    - Document what is done
    - Possibility of re-running if data change
  - Manuals, course handouts
    - Output shown guaranteed to be result of code shown

StatWeave design goals

Goals

- Like **Sweave**, **SASweave**, **odfWeave**, combined and unified
- Support lots of languages
- Support different file formats; currently. . .
  - LaTeX
  - OpenDocument Text (ODT): `www.openoffice.org` or MSOffice plug-in from `www.sun.com/software/star/odf_plugin/index.jsp`
- Portability: Usable on all platforms
  - Written in Java
- Extensibility
  - Add languages
  - Add file formats (and possibly syntax rules)

How **StatWeave** works

- Source file is like a regular document, but with *code chunks* added (delineated with special tags)
- Two basic operations:

  Weaving Process source file into a complete document with code listings, output listings, and graphs added

  Tangling Extract code from source file, to run or compile later

- Tangling is most useful for literate programming in languages like Fortran, and C.
- Weaving is most useful for reproducible statistical analysis

Running **StatWeave**

- Currently a command-line interface

```
statweave myfile.swv
statweave --tangle myfile-swv.odt
statweave --target dvi myfile-swv.tex
statweave --config new.cfg --keepall src.swv
```

- Available for download at
  www.stat.uiowa.edu/~rlenth/StatWeave/
- GUI easy to do, and portable—to come

Example (LaTeX format)

Source file (oxide.swv)

```
\documentclass[12pt]{article}
\usepackage[margin=1in]{geometry}
\begin{document}
\section*{Split-plot analysis}
In this example, we analyze some data on
a semiconductor manufacturing process.
\begin{Rcode}{fig, scale=.5}
library(nlme)
plot(Oxide)
\end{Rcode}

Fit a multi-level model with random intercepts for
lots and wafers within lots.
\begin{Rcode}
lme (Thickness ~ Source*Site, ~ 1 | Lot/Wafer, data=Oxide)
\end{Rcode}
\end{document}
```

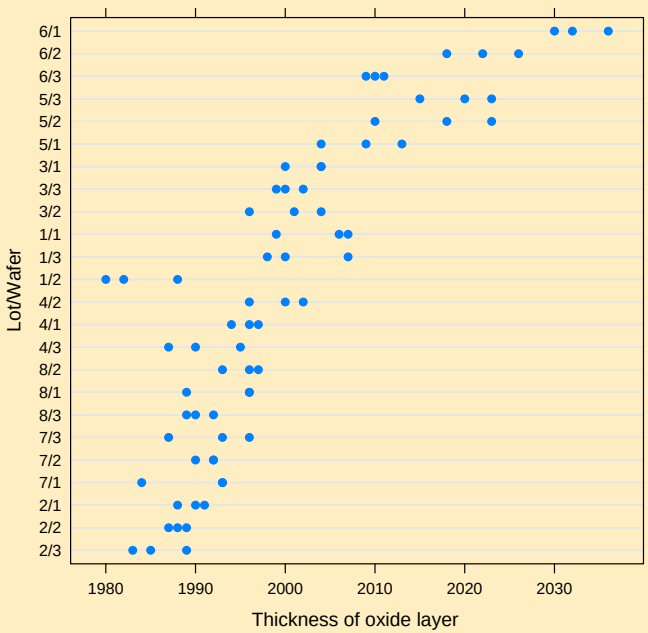
Result after weaving with **StatWeave**

oxide.pdf

Split-plot analysis

In this example, we analyze some data on a semiconductor manufacturing process.

```
R> library (nlme)
R> plot (Oxide)
```



Results (cont'd)

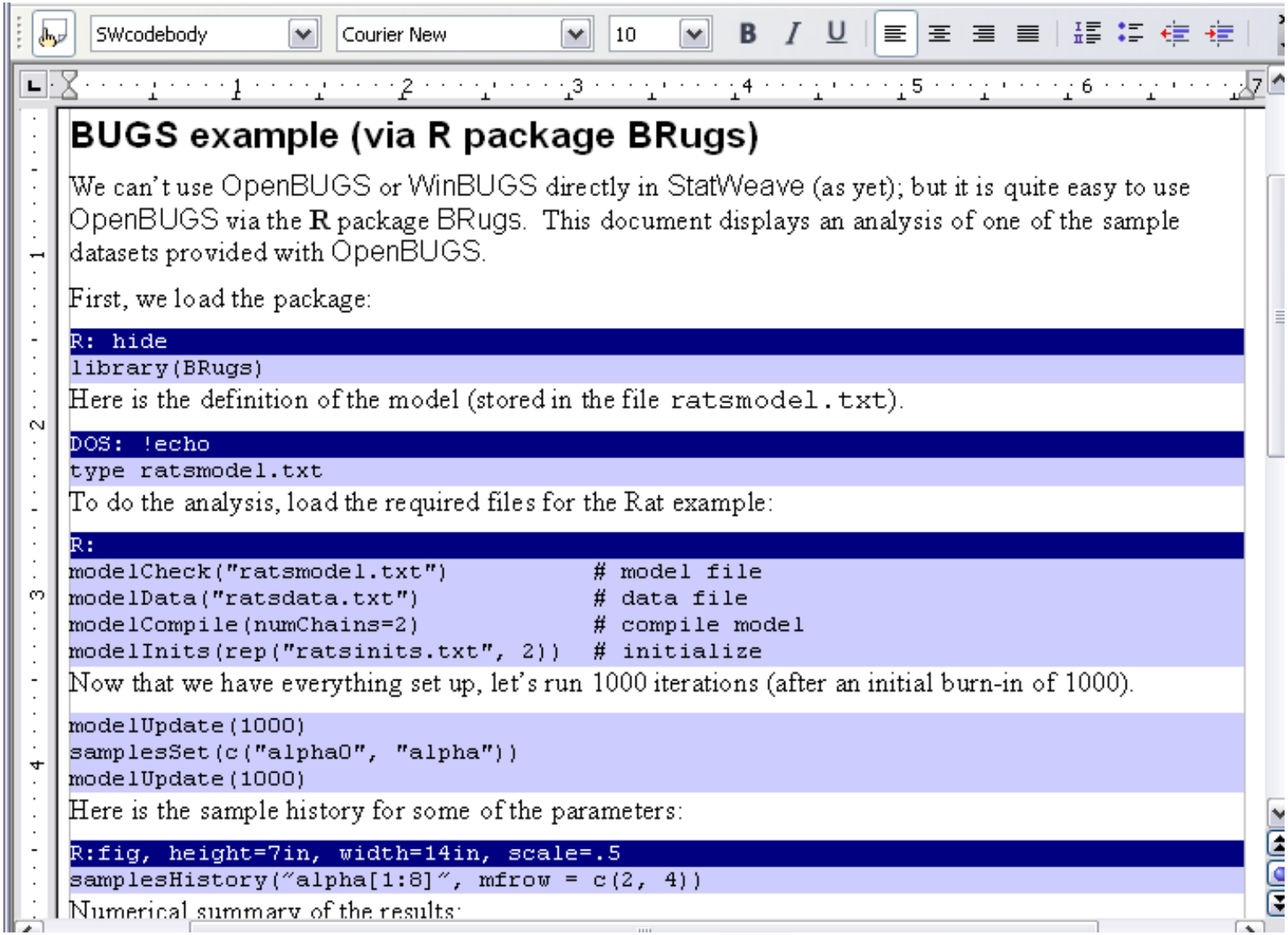
Fit a multi-level model with random intercepts for lots and wafers within lots.

```
R> lme (Thickness ~ Source*Site,
R>                ~ 1 | Lot/Wafer, data = Oxide)

Linear mixed-effects model fit by REML
  Data: Oxide
  Log-restricted-likelihood: -215.4166
  Fixed: Thickness ~ Source * Site
  (Intercept)       Source2         Site2         Site3 Source2:Site2
 1994.0833333    11.7500000     0.1666667     2.9166667    -0.8333333
Source2:Site3
   -4.1666667
Random effects:
 Formula: ~1 | Lot
        (Intercept)
StdDev:    10.94954
 Formula: ~1 | Wafer %in% Lot
        (Intercept) Residual
StdDev:    6.003643 3.469201
Number of Observations: 72
Number of Groups:
           Lot Wafer %in% Lot
             8              24
```

ODT example

Input file (Rats-swv.odt)



SWcodebody | Courier New | 10

BUGS example (via R package BRugs)

We can't use OpenBUGS or WinBUGS directly in StatWeave (as yet); but it is quite easy to use OpenBUGS via the **R** package BRugs. This document displays an analysis of one of the sample datasets provided with OpenBUGS.

First, we load the package:

```
R: hide
library(BRugs)
```

Here is the definition of the model (stored in the file `ratsmodel.txt`).

```
DOS: !echo
type ratsmodel.txt
```

To do the analysis, load the required files for the Rat example:

```
R:
modelCheck("ratsmodel.txt")           # model file
modelData("ratsdata.txt")             # data file
modelCompile(numChains=2)             # compile model
modelInits(rep("ratsinits.txt", 2))   # initialize
```

Now that we have everything set up, let's run 1000 iterations (after an initial burn-in of 1000).

```
modelUpdate(1000)
samplesSet(c("alpha0", "alpha"))
modelUpdate(1000)
```

Here is the sample history for some of the parameters:

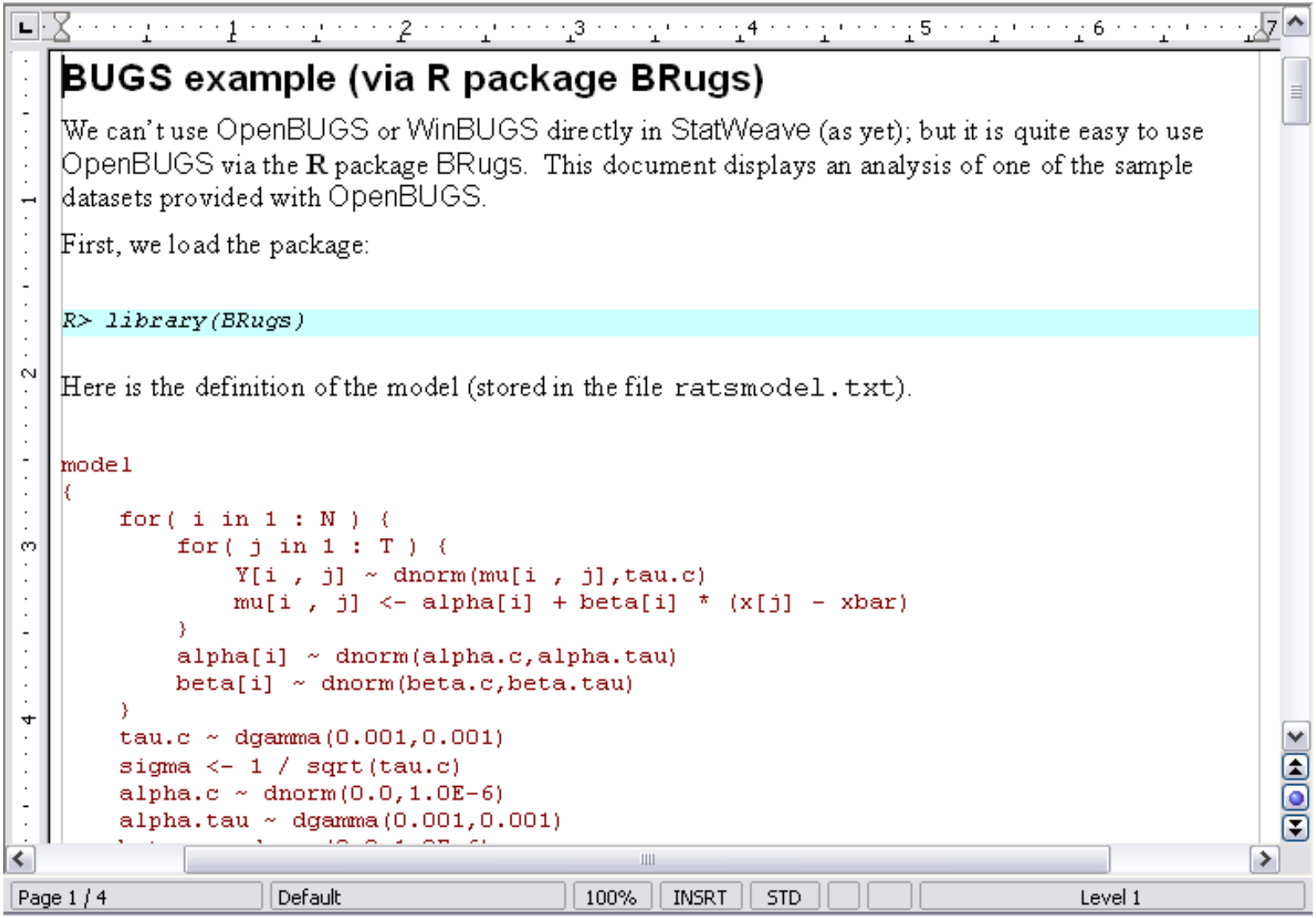
```
R:fig, height=7in, width=14in, scale=.5
samplesHistory("alpha[1:8]", mfrow = c(2, 4))
```

Numerical summary of the results:

ODT example—Results

The University of Iowa

Output file (`Rats.odt`)



BUGS example (via R package BRugs)

We can't use OpenBUGS or WinBUGS directly in StatWeave (as yet); but it is quite easy to use OpenBUGS via the **R** package BRugs. This document displays an analysis of one of the sample datasets provided with OpenBUGS.

First, we load the package:

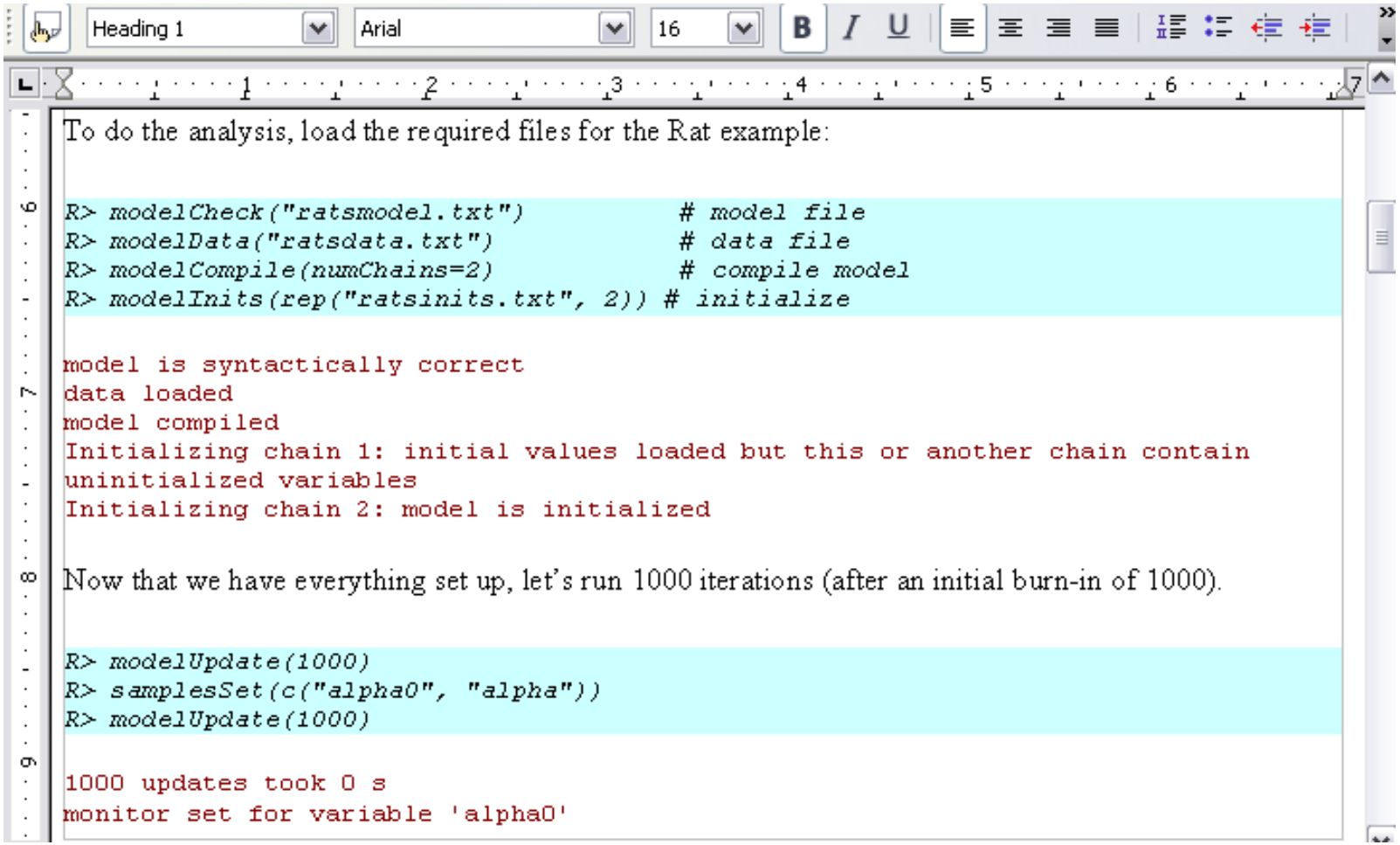
```
R> library(BRugs)
```

Here is the definition of the model (stored in the file `ratsmodel.txt`).

```
model
{
    for( i in 1 : N ) {
        for( j in 1 : T ) {
            Y[i , j] ~ dnorm(mu[i , j],tau.c)
            mu[i , j] <- alpha[i] + beta[i] * (x[j] - xbar)
        }
        alpha[i] ~ dnorm(alpha.c,alpha.tau)
        beta[i] ~ dnorm(beta.c,beta.tau)
    }
    tau.c ~ dgamma(0.001,0.001)
    sigma <- 1 / sqrt(tau.c)
    alpha.c ~ dnorm(0.0,1.0E-6)
    alpha.tau ~ dgamma(0.001,0.001)
```

Page 1 / 4 | Default | 100% | INSRT | STD | Level 1

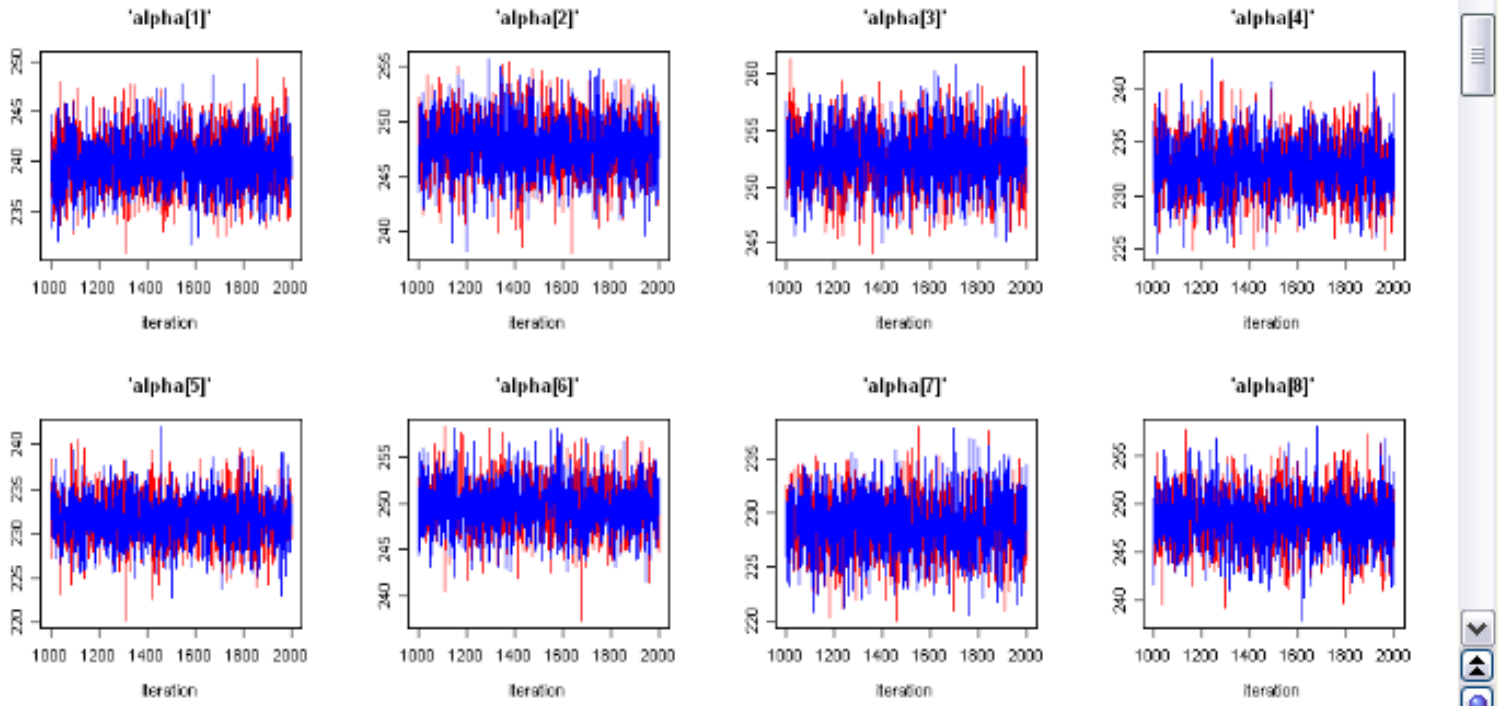
ODT results (cont'd)



ODT results (cont'd)

Here is the sample history for some of the parameters:

```
R> samplesHistory("alpha[1:8]", mfrow = c(2, 4))
```



Maple example (LaTeX format)

```
Here is a messy expression and its messier derivative
\begin{Maplecode}
a := sin(x) * x^(x^x);
diff(a, x);
\end{Maplecode}

We can make this look much nicer, though:
\begin{align}
  a                      &= \Mapleexpr{latex(a)} \\
  \partial a/\partial x &= \Mapleexpr{latex(diff(a,x))}
\end{align}
```

Maple example results

Here is a messy expression and its messier derivative

```
Maple> a := sin(x) * x^(x^x);
Maple> diff(a, x);
```

```
                                                   x
                                                 (x )
                                  a := sin(x) x
              x               x   /                              x \
            (x )            (x )  | x                          x   |
cos(x) x         + sin(x) x       |x  (ln(x) + 1) ln(x) + ----|
                                  \                            x   /
```

We can make this look much nicer, though:

$$a = \sin(x) x^{x^x} \quad (1)$$

$$\partial a/\partial x = \cos(x) x^{x^x} + \sin(x) x^{x^x} \left(x^x (\ln(x) + 1) \ln(x) + \frac{x^x}{x} \right) \quad (2)$$

Multi-language example (ODT)

Source-file excerpt

```
SAS:codestyle=blueStyle
R:codestyle=redStyle
Stata:codestyle=greenStyle
ompt=": ", scale=.75
```

Passing data around

There is a sample data set on shoe sales in the SASHELP library:

```
SAS:fig
proc gplot data=sashelp.shoes;
  plot returns * sales;
```

Let's export the data to a comma-delimited file.

```
proc export data = sashelp.shoes
            outfile = "shoes.csv"
            dbms = CSV  replace;
```

OK, now let's look at a few lines of this file.

```
DOS:
head shoes.csv
```

Read the file into R

```
R:
shoes = read.csv("shoes.csv")
str(shoes)
```

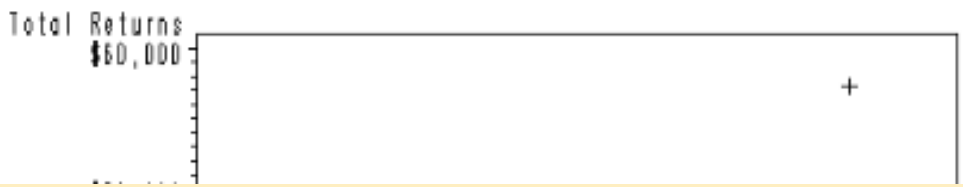
Note that this dataset has `R:nrow(shoes)` observations. The last 3 variables are messy because SAS formatted them as dollars with embedded commas. We need to fix this; let's write a function for this

Multi-language results

Passing data around

There is a sample data set on shoe sales in the SASHELP library:

```
SAS: proc gplot data=sashelp.shoes;
SAS:   plot returns * sales;
```



Read the file into R

```
R: shoes = read.csv("shoes.csv")
R: str(shoes)
```

```
'data.frame': 395 obs. of  7 variables:
 $ Region    : Factor w/ 10 levels "Africa","Asia",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ Product   : Factor w/ 8 levels "Boot","Men's Casual",..: 1 2 3 4 5 6 7 8 1 2 ..
 $ Returns   : Factor w/ 372 levels "$1,006","$1,009",..: 329 119 129 67 62 330 36
180 321 116 ...
```

Note that this dataset has 395 observations. The last 3 variables are messy because SAS formatted them as dollars with embedded commas. We need to fix this; let's write a function for this purpose

Multi-language example (end part)

```
write.csv(shoes, "newshoes.csv")
```

Read this dataset into Stata and plot it.

```
Stata: fig
insheet using "newshoes.csv", comma
label var logsales "ln of Sales"
label var logreturns "ln of Returns"
mean (logsales logreturns)
twoway (scatter logsales logreturns)
```

Finally, read this new dataset into SAS and plot it.

```
SAS: restart, fig
proc import
  datafile = "newshoes.csv"
  out = newshoes
```

```
R: write.csv(shoes, "newshoes.csv")
```

Read this dataset into Stata and plot it.

```
. insheet using "newshoes.csv", comma
. label var logsales "ln of Sales"
. label var logreturns "ln of Returns"
. mean (logsales logreturns)
. twoway (scatter logsales logreturns)

(11 vars, 395 obs)
Mean estimation                   Number of obs    =       395
--------------------------------------------------------------
             |       Mean   Std. Err.      [95% Conf. Interval]
```

Re-using code; Argument substitution

LaTeX source-file excerpt (from a class handout)

```
\begin{SAScode}{label=mixedcode, !eval, !echo}
proc mixed data = laundry;
  class TUB det blch stain pwash;
  model #1 = det|blch|stain|pwash / outp=diags;
  random TUB;
proc gplot data = diags;
  plot resid*pred;
run;
\end{SAScode}
%
\begin{SAScode}{hide, fig, savefig}
\coderef{*mixedcode}{Whiteness}
\end{SAScode}
% (Prepending the * makes it show the recalled code)
```

Re-using code (cont'd)

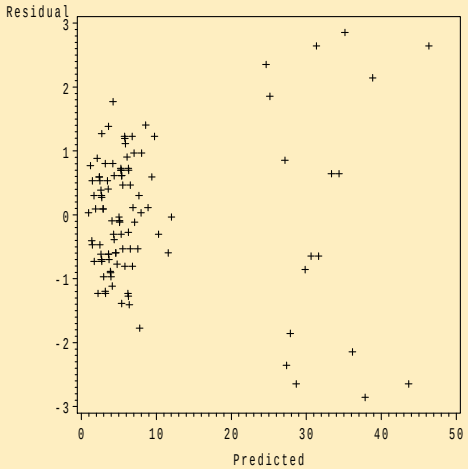
```
\begin{tabular}{ccc}
Untransformed & Square root & Log \\
\recallfig{lastchunk}
&
\begin{SAScode}{hide, fig}
\coderef{mixedcode}{sqWht}
\end{SAScode}
&
\begin{SAScode}{hide, fig}
\coderef{mixedcode}{logWht}
\end{SAScode}
\end{tabular}
```

Code-reuse results

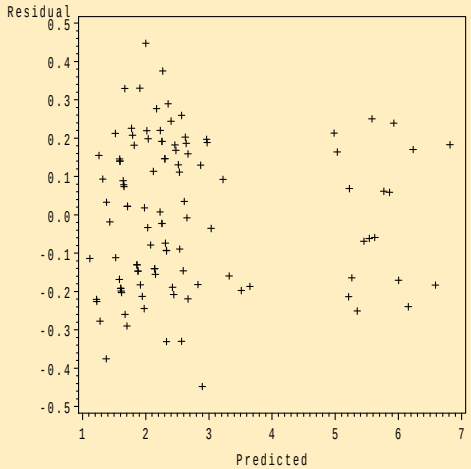
statements for fitting a model to the untransformed response and obtaining a residual plot; and the residuals-versus-fitted plots for the same model with the three transformations.

```
proc mixed data = laundry;
  class TUB det blch stain pwash;
  model Whiteness = det|blch|stain|pwash / outp=diags;
  random TUB;
proc gplot data = diags;
  plot resid*pred;
run;
```

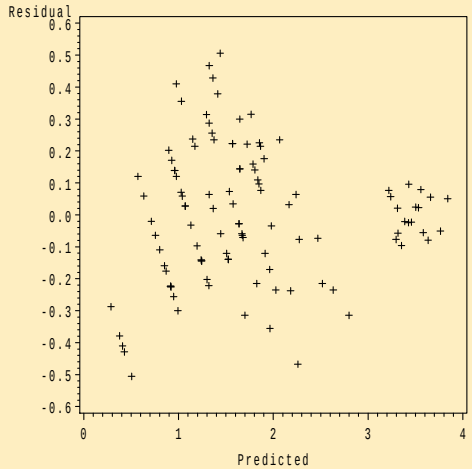
Untransformed



Square root



Log



The square root is the best choice. Here is the ANOVA table for the fixed effects for

StatWeave tags summary

Element	**LaTeX**	**ODT** <par style>
Code chunk	*lang* code environment {*opts*} environment args	<SWcodehead>*lang:opts* <SWcodebody>
Global opts	\weaveOpts{*opts*} *lang*weaveOpts{*opts*}	<SWoptions>*opts* <SWoptions>*lang:opts*
Expression	*lang*expr{*expr*}	<SWexpr>*lang:expr* (character style)
Code reuse	\coderef{*label*} \coderef{*label*}{*arg*}$\cdots$	<SWcoderef>*label* <SWcoderef>*label*{*arg*}$\cdots$
Results reuse	\recall*what*{*label*} where *what* = code, out, or fig	<SWrecall>*what:label*

Useful options

- Code: `echo`, `eval`, `prompt`, `ompt`, `codefmt`, `codestyle`
- Output: `hide`, `outfmt`, `outstyle`
- Graphics: `fig`, `width`, `height`, `scale`, `dispw`, `disph`
- Other: `label`, `showrefs`, `savecode`, `saveout`, `savefig`, `restart`, `newlang`

Java implementation

- `StatWeave` class: Main program
- `FileInterface` interface: Implementations include `LaTeXFile` (with subsidiary `SyntaxInterface` implementations like `LaTeXSyntax`) and `ODTFile`
- `EngineInterface` interface and `AbstractEngine` class: Extensions include `REngine`, `SASEngine`, `StataEngine`, `SplusEngine`, `MapleEngine`, `LaTeXEngine`, `UnixEngine`, `DOSEngine`
- Configuration file: Tells `StatWeave` what file formats and engines are available, etc.
- Miscellaneous classes: `Err`, `FigFile`, `Tag`

`FileInterface` required methods


```
public void setParent(StatWeave parent);
public void readSourceFile(String fileName);
public void writeResults(int mopUp);
public Tag nextTag();
public String getLine();
public String getPosition();
public String getLang();
public String getExpr();
public String getOptions();
public String getCodeLine();
public String getLabel();
public java.util.Vector<String> getArgs();
public void saveBookmark(String sig);
public void replaceBookmark(String sig, String text,
                            Tag context);
public int[] getFigFormats();
public void embedPlots(String sig, FigFile ff, int[] pages);
```

AbstractEngine methods

These usually do need to be overridden:

```
public void putExpr(String expr);
public void putSeparator(String text);
public FigFile setupFig(String chunkName);
public void closeFig();
```

Many usually don't need to be overridden:

```
public boolean openCodeFile(String baseName);
public void closeCodeFile();
public void deleteCodeFile();
public String[] getFileNames();
public void setParents(StatWeave parent,
                       FileInterface filei);
public void setBinary(String binloc);
public void putStartup(boolean weaving);
public void putComment(String text);
public void putCode(String line);
public Process runCode();
```

StatWeave courtesy methods


```
public static void message(String msg);
public static void warning(String msg);
public void error (String msg, int errCode);

public String getOption(String optName);
public String getOption(String optName, String dfault);
public boolean isTrue(String optName);
public double getDim(String optName, double dfault);
public String getConfig(String property);

public boolean isWeaving();
```

Configuration file .statweave or statweave.cfg


```
### File-format configuration ###...
FileInterfaces = LaTeXFile ODTFile

ODTFile.class = rvl.swv.ODTFile
ODTFile.sources = -swv.odt .swv.odt odt
ODTFile.target = odt

LaTeXFile.class = rvl.swv.LaTeXFile
LaTeXFile.sources = -swv.tex .swv.tex tex swv
LaTeXFile.target = pdf    # or dvi or tex
LaTeXFile.figfmt.pdf = PDF
LaTeXFile.figfmt.dvi = PS
LaTeXFile.bin.pdf = pdflatex --quiet
LaTeXFile.bin.dvi = latex --quiet
LaTeXFile.SyntaxInterfaces = LaTeXSyntax   #nowebSyntax
LaTeXSyntax.class = rvl.swv.LaTeXSyntax
```

Configuration file (cont'd)

```
### Language and engine configuration ###
Languages = S R Splus SAS IML Maple tex Unix
Engines = R Splus SAS tex Maple Unix

S.engine = R
IML.engine = SAS
# Don't need engine mappings when lang == engine

R.class = rvl.swv.REngine
R.binary = swvrun R %codefile% %outfile%
SAS.class = rvl.swv.SASEngine
SAS.binary = SAS %codefile%
# similar specs for other engines

# Global.options =
R.options = prompt=\"> \"
tex.options = results=tex
```

Summary

- Reproducible statistical analyses
  - Integrated documentation
  - Several languages, file formats
  - Portable, extensible
- To do next...
  - Finish draft documentation
  - Add support for Genstat, Matlab, Mathematica, ...(suggestions?)
  - Add `noweb` syntax for LaTeX
  - Add `docx` file format
  - Other ideas?