

Introduction to the Stat-JR software package

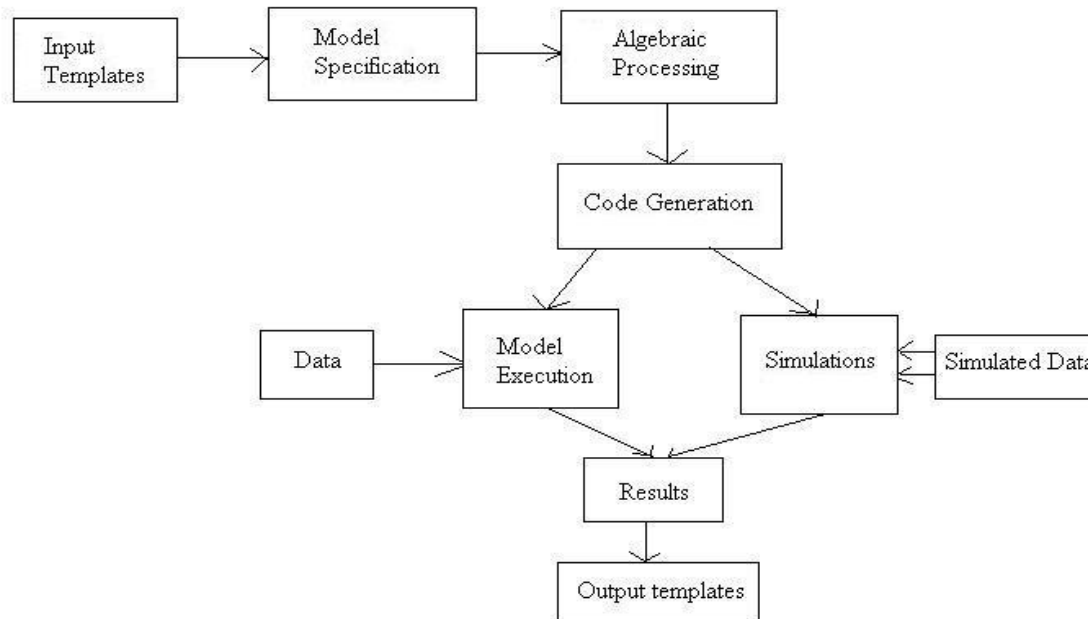
Professor William Browne

Video 1 What is StatJR

- A statistical software package written in Python and first released in 2013.
- Named after our former colleague Jon Rasbash and pronounced “Stature”.
- Stat-JR is meant to appeal to novice users, expert users and other algorithm developers
- It has its own MCMC estimation engine built into the software but also allows interoperability with other software packages (this talk).
- Has several interfaces including an electronic book interface including “statistical analysis assistant” features (talk 2).
- Can also be used to create “bespoke” training materials in combination with the SPSS software package (talk 3).

StatJR component based approach

Below is an early diagram of how we envisioned the system. Here you will see boxes representing components some of which are built into the STAT-JR system. The system is written in Python with a VB.net algebra processing system. A team of coders have worked together on the system.



Templates

Backbone of Stat-JR.

Consist of a set of code sections for advanced users to write. A bit like R packages.

For a model template it consists of at least:

- an *inputs* method which specifies inputs and types
- A *model* method that creates (BUGS like) model code for the algebra system
- An (optional) *latex* method can be used for outputting LaTeX code for the model.

Other optional functions required for more complex templates

Regression I Example

```
from EStat.Templating import *

class Regression1(Template):
    'A model template for fitting 1 level Normal multiple regression model
    in eStat only.'
    tags = [ 'Model', '1-Level', 'eStat', 'Normal' ]
    engines = ['eStat']
    inputs = ""
    y = DataVector('Response: ')
    x = DataMatrix('Explanatory variables: ', allow_cat=True, help=
        'predictor variables')
    beta = ParamVector(parents=[x], as_scalar=True)
    tau = ParamScalar()
    sigma = ParamScalar(modelled = False)
    sigma2 = ParamScalar(modelled = False)
    deviance = ParamScalar(modelled = False)
    ""
```

```
model = ""
model{
    for (i in 1:length($y)) {
        ${y}[i] ~ dnorm(mu[i], tau)
        mu[i] <- ${mmult(x, 'beta', 'i')}
    }

    # Priors
    % for i in range(0, x.ncols()):
    beta${i} ~ dflat()
    % endfor
    tau ~ dgamma(0.001000, 0.001000)
    sigma2 <- 1 / tau
    sigma <- 1 / sqrt(tau)
}
""

    latex = r""
\begin{aligned}
\mbox{\${y}}_i &\sim \mbox{N}(\mu_i, \sigma^2) \\
\mu_i &= \\
&\quad \mathbf{mmult}(\mathbf{x}, \mathbf{\beta}, 'i') \\
\%for i in range(0, len(x)):
\beta_{\${i}} &\propto 1 \\
\%endfor
\tau &\sim \Gamma(0.001, 0.001) \\
\sigma^2 &= 1 / \tau
\end{aligned}
""
```

An example of STAT-JR – setting up a model

The screenshot shows the Stat-JR web interface in a browser window. The browser address bar shows 'localhost:51610/run/'. The interface has a dark header with 'Stat-JR: TREE' and navigation options like 'Start again', 'Dataset', 'tutorial', 'Template', and 'Regression1'. A green 'Ready (1s)' indicator is visible. The main content area lists various configuration options, each with a 'remove' link:

- Response:** normexam remove
- Explanatory variables:** cons,standlrt remove
- Number of chains:** 3 remove
- Random Seed:** 1 remove
- Length of burnin:** 500 remove
- Number of iterations:** 2000 remove
- Thinning:** 1 remove
- Use default algorithm settings:** Yes remove
- Generate prediction dataset:** No remove
- Use default starting values:** Yes remove
- Name of output results:** out

Below the settings is an orange 'Next' button. At the bottom, there are two informational boxes:

Current input string: `{'burnin': '500', 'defaultsv': 'Yes', 'thinning': '1', 'nchains': '3', 'defaultalg': 'Yes', 'iterations': '2000', 'y': 'normexam', 'x': 'cons,standlrt', 'seed': '1', 'makepred': 'No'}`

Command: `RunStatJR(template='Regression1', dataset='tutorial', invars = {'y': 'normexam', 'x': 'cons,standlrt'}, estoptions = {'burnin': '500', 'defaultsv': 'Yes', 'thinning': '1', 'nchains': '3', 'defaultalg': 'Yes', 'iterations': '2000', 'seed': '1', 'makepred': 'No'})`

An example of STAT-JR – setting up a model



Response: normexam [remove](#)

Explanatory variables: cons,standlrt [remove](#)

Number of chains: 3 [remove](#)

Random Seed: 1 [remove](#)

Length of burnin: 500 [remove](#)

Number of iterations: 2000 [remove](#)

Thinning: 1 [remove](#)

Use default algorithm settings: Yes [remove](#)

Generate prediction dataset: Yes [remove](#)

Use default starting values: Yes [remove](#)

Name of output results:



Equations for model

The screenshot shows the Stat-JR 1.0.2:TREE web interface. The browser address bar shows `localhost:8080/run/`. The interface includes a navigation bar with `Stat-JR:TREE`, `Start again`, `Dataset`, `tutorial`, `Template`, `Regression1`, `Ready (3s)`, `Settings`, and `Debug`. Below the navigation bar, there is a section for the current input string: `{'burnin': '500', 'defaultsv': 'Yes', 'outdata': 'out', 'thinning': '1', 'nchains': '3', 'defaultalg': 'Yes', 'iterations': '2000', 'y': 'normexam', 'x': 'cons,standlrt', 'seed': '1', 'makepred': 'No'}`. A `Set` button is present. Below this, the command is shown: `RunStatJR(template=Regression1, dataset=tutorial, invars = {'y': 'normexam', 'x': 'cons,standlrt'}, estoptions = {'burnin': '500', 'defaultsv': 'Yes', 'thinning': '1', 'nchains': '3', 'defaultalg': 'Yes', 'iterations': '2000', 'outdata': 'out', 'seed': '1', 'makepred': 'No'})`. An `Edit` button is next to a dropdown menu showing `equation.tex` and a `Popout` button. The main content area displays the following equations for the model:

$$\begin{aligned} \text{normexam}_i &\sim N(\mu_i, \sigma^2) \\ \mu_i &= \beta_0 \text{cons}_i + \beta_1 \text{standlrt}_i \\ \beta_0 &\propto 1 \\ \beta_1 &\propto 1 \\ \tau &\sim \Gamma(0.001, 0.001) \\ \sigma^2 &= 1/\tau \end{aligned}$$

- All objects created available from one pull down and can be popped out to separate tabs in browser.

Equations for model

$$\text{normexam}_i \sim N(\mu_i, \sigma^2)$$

$$\mu_i = \beta_0 \text{cons}_i + \beta_1 \text{standlrt}_i$$

$$\beta_0 \propto 1$$

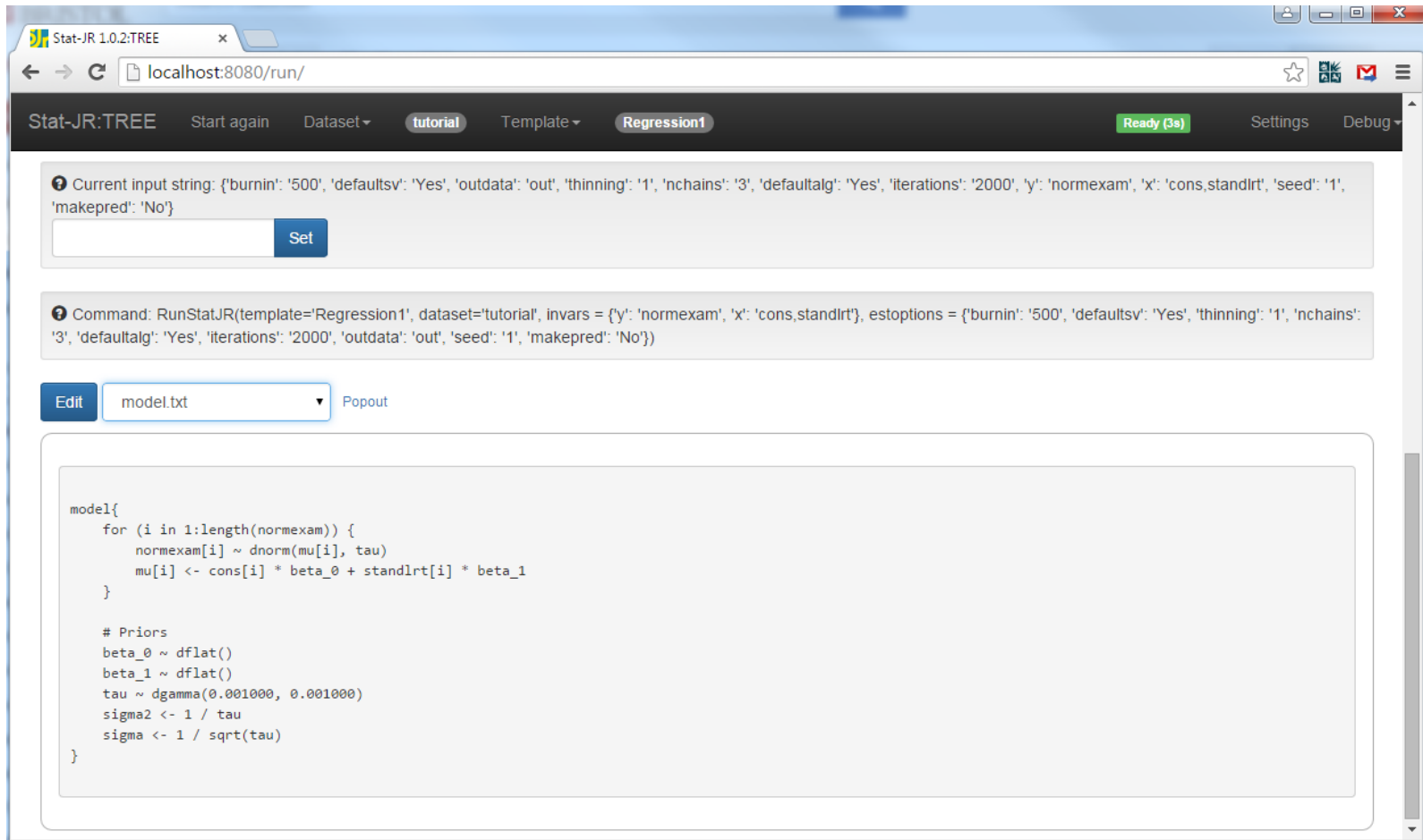
$$\beta_1 \propto 1$$

$$\tau \sim \Gamma(0.001, 0.001)$$

$$\sigma^2 = 1/\tau$$

- Note: Equations use MATHJAX and so underlying LaTeX can be copied and paste. The model code is based around the WinBUGS language with some variation.

Model code



The screenshot shows the Stat-JR 1.0.2:TREE web interface. The browser address bar shows 'localhost:8080/run/'. The interface includes a navigation bar with 'Stat-JR:TREE', 'Start again', 'Dataset', 'tutorial', 'Template', and 'Regression1'. A 'Ready (3s)' indicator is visible. Below the navigation bar, there is a section for the current input string: '{'burnin': '500', 'defaultsv': 'Yes', 'outdata': 'out', 'thinning': '1', 'nchains': '3', 'defaultalg': 'Yes', 'iterations': '2000', 'y': 'normexam', 'x': 'cons,standlrt', 'seed': '1', 'makepred': 'No'}'. A 'Set' button is next to the input field. Below this, the command being executed is shown: 'Command: RunStatJR(template='Regression1', dataset='tutorial', invars = {'y': 'normexam', 'x': 'cons,standlrt'}, estoptions = {'burnin': '500', 'defaultsv': 'Yes', 'thinning': '1', 'nchains': '3', 'defaultalg': 'Yes', 'iterations': '2000', 'outdata': 'out', 'seed': '1', 'makepred': 'No'})'. An 'Edit' button and a dropdown menu showing 'model.txt' are also present. The main area contains the following R code:

```
model{
  for (i in 1:length(normexam)) {
    normexam[i] ~ dnorm(mu[i], tau)
    mu[i] <- cons[i] * beta_0 + standlrt[i] * beta_1
  }

  # Priors
  beta_0 ~ dflat()
  beta_1 ~ dflat()
  tau ~ dgamma(0.001000, 0.001000)
  sigma2 <- 1 / tau
  sigma <- 1 / sqrt(tau)
}
```

- All objects created available from one pull down and can be popped out to separate tabs in browser.

Model code in detail

```
model{
  for (i in 1:length(normexam)) {
    normexam[i] ~ dnorm(mu[i], tau)
    mu[i] <- cons[i] * beta0 + standlrt[i] * beta1
  }
# Priors
  beta0 ~ dflat()
  beta1 ~ dflat()
  tau ~ dgamma(0.001000, 0.001000)
  sigma2 <- 1 / tau
  sigma <- 1/sqrt(tau)
}
```

For this template the code is, aside from the length function, standard WinBUGS model code.

Algebra system steps

Stat-JR:TREE

Log posterior $\tau \left(\sum_{i=1}^{\text{length}(\text{normexam})} \text{cons}_i (\text{normexam}_i - \text{beta}_{-1} \text{standlrt}_i) \right) \text{beta}_{-0} - \frac{\tau \left(\sum_{i=1}^{\text{length}(\text{normexam})} \text{cons}_i^2 \right) \text{beta}_{-0}^2}{2}$

Distribution `dnorm`

Match $A = \tau \sum_{i=1}^{\text{length}(\text{normexam})} \text{cons}_i (\text{normexam}_i - \text{beta}_{-1} \text{standlrt}_i)$

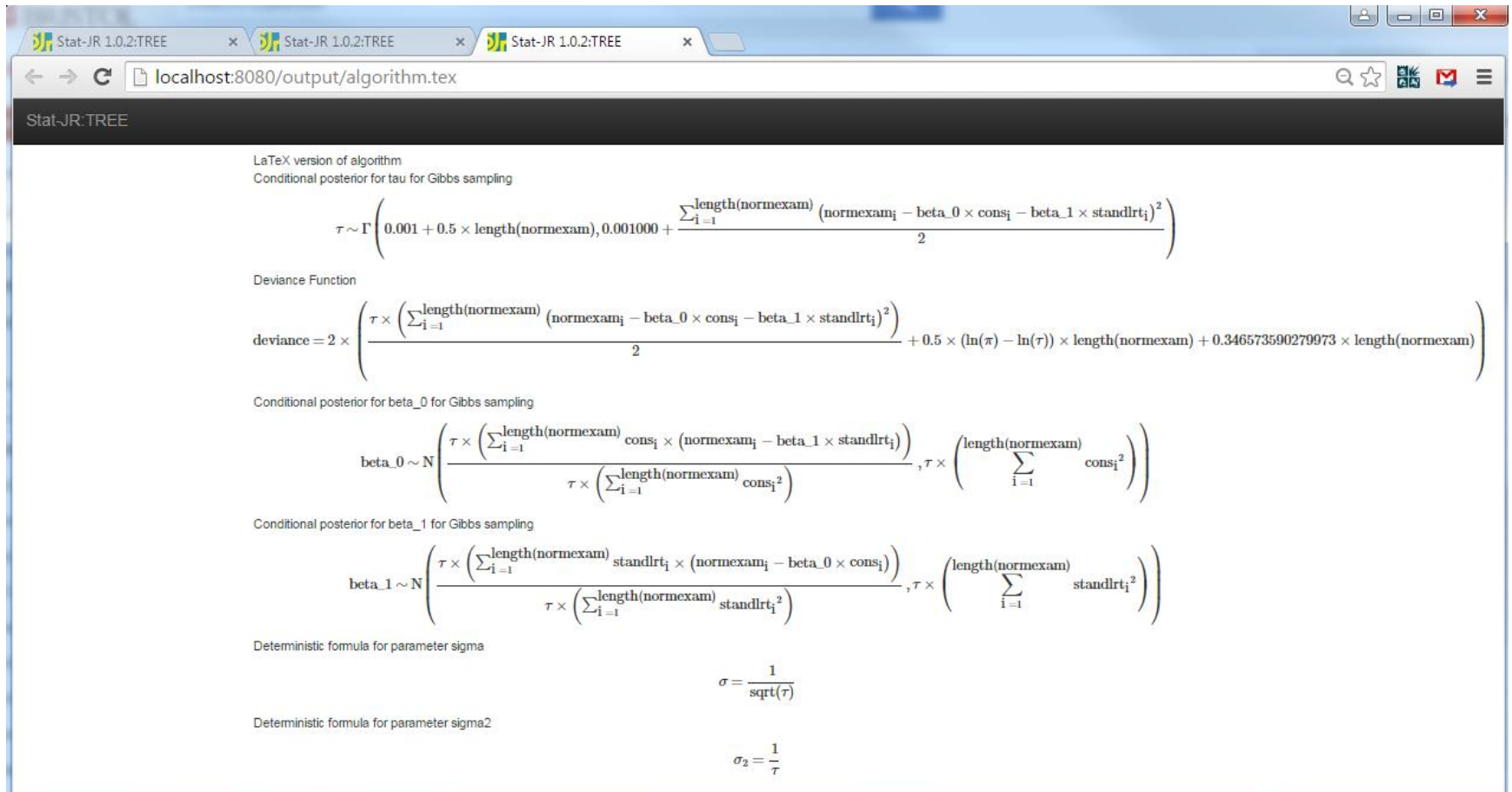
Match $B = - \left(\frac{\tau \sum_{i=1}^{\text{length}(\text{normexam})} \text{cons}_i^2}{2} \right)$

Sampling parameter $\mu = \frac{\tau \sum_{i=1}^{\text{length}(\text{normexam})} \text{cons}_i (\text{normexam}_i - \text{beta}_{-1} \text{standlrt}_i)}{\tau \sum_{i=1}^{\text{length}(\text{normexam})} \text{cons}_i^2}$

Sampling parameter $\tau = \tau \sum_{i=1}^{\text{length}(\text{normexam})} \text{cons}_i^2$

Sampling distribution $\text{beta}_{-0} \sim \text{dnorm} \left(\frac{\tau \sum_{i=1}^{\text{length}(\text{normexam})} \text{cons}_i (\text{normexam}_i - \text{beta}_{-1} \text{standlrt}_i)}{\tau \sum_{i=1}^{\text{length}(\text{normexam})} \text{cons}_i^2}, \tau \sum_{i=1}^{\text{length}(\text{normexam})} \text{cons}_i^2 \right)$

Algebra system steps



Stat-JR 1.0.2:TREE

localhost:8080/output/algorithm.tex

Stat-JR: TREE

LaTeX version of algorithm

Conditional posterior for tau for Gibbs sampling

$$\tau \sim \Gamma \left(0.001 + 0.5 \times \text{length}(\text{normexam}), 0.001000 + \frac{\sum_{i=1}^{\text{length}(\text{normexam})} (\text{normexam}_i - \text{beta}_0 \times \text{cons}_i - \text{beta}_1 \times \text{standlrt}_i)^2}{2} \right)$$

Deviance Function

$$\text{deviance} = 2 \times \left(\frac{\tau \times \left(\sum_{i=1}^{\text{length}(\text{normexam})} (\text{normexam}_i - \text{beta}_0 \times \text{cons}_i - \text{beta}_1 \times \text{standlrt}_i)^2 \right)}{2} + 0.5 \times (\ln(\pi) - \ln(\tau)) \times \text{length}(\text{normexam}) + 0.346573590279973 \times \text{length}(\text{normexam}) \right)$$

Conditional posterior for beta_0 for Gibbs sampling

$$\text{beta}_0 \sim N \left(\frac{\tau \times \left(\sum_{i=1}^{\text{length}(\text{normexam})} \text{cons}_i \times (\text{normexam}_i - \text{beta}_1 \times \text{standlrt}_i) \right)}{\tau \times \left(\sum_{i=1}^{\text{length}(\text{normexam})} \text{cons}_i^2 \right)}, \tau \times \left(\sum_{i=1}^{\text{length}(\text{normexam})} \text{cons}_i^2 \right) \right)$$

Conditional posterior for beta_1 for Gibbs sampling

$$\text{beta}_1 \sim N \left(\frac{\tau \times \left(\sum_{i=1}^{\text{length}(\text{normexam})} \text{standlrt}_i \times (\text{normexam}_i - \text{beta}_0 \times \text{cons}_i) \right)}{\tau \times \left(\sum_{i=1}^{\text{length}(\text{normexam})} \text{standlrt}_i^2 \right)}, \tau \times \left(\sum_{i=1}^{\text{length}(\text{normexam})} \text{standlrt}_i^2 \right) \right)$$

Deterministic formula for parameter sigma

$$\sigma = \frac{1}{\text{sqrt}(\tau)}$$

Deterministic formula for parameter sigma2

$$\sigma_2 = \frac{1}{\tau}$$

Algebra system steps

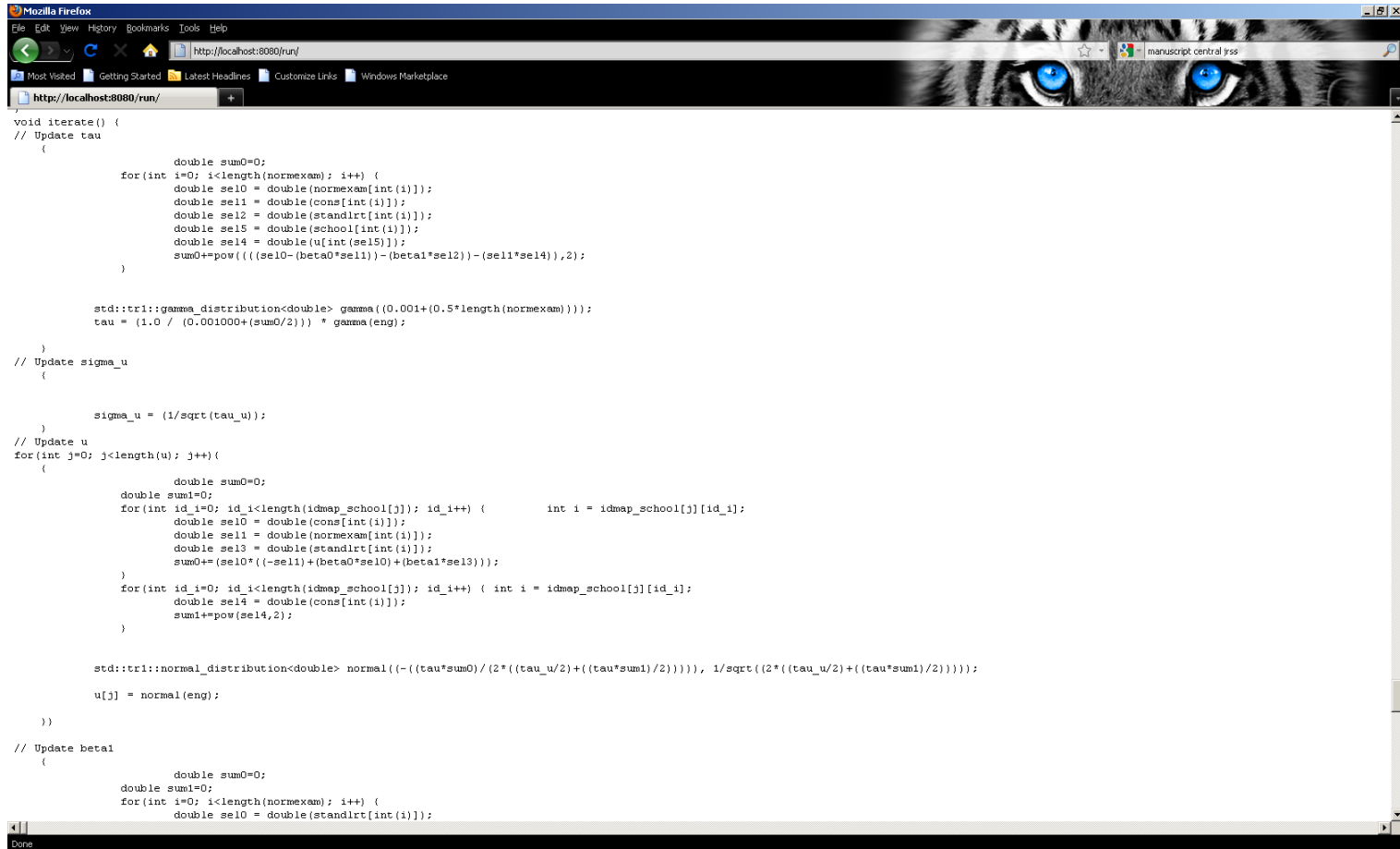
Use Gibbs sampling from conditional posterior for beta1:

$$\beta_1 \sim N \left(\frac{\tau \times \left(\sum_{i=1}^{\text{length}(\text{normexam})} \text{standlrt}_i \times (\text{normexam}_i - \beta_0 \times \text{cons}_i) \right)}{\tau \times \left(\sum_{i=1}^{\text{length}(\text{normexam})} \text{standlrt}_i^2 \right)}, \tau \times \left(\sum_{i=1}^{\text{length}(\text{normexam})} \text{standlrt}_i^2 \right) \right)$$

$$\beta_1 \sim N(0.000249799765395 \times (2382.12631198 + \beta_0 \times (-7.34783096611)), 4003.20632175 \times \tau)$$

- Here the first line is what is returned by the algebra system – which works solely on the model code.
- The second line is what can be calculated when values are added for constants and data etc.
- System then constructs C code and fits model

Output of generated C++ code



```
void iterate() {
// Update tau
{
    double sum0=0;
    for(int i=0; i<length(normexam); i++) {
        double se10 = double(normexam[int(i)]);
        double se11 = double(cons[int(i)]);
        double se12 = double(standlrt[int(i)]);
        double se15 = double(school[int(i)]);
        double se14 = double(u[int(se15)]);
        sum0+=pow(((se10-(beta0*se11))-(beta1*se12))-(se11*se14),2);
    }

    std::tr1::gamma_distribution<double> gamma(0.001+(0.5*length(normexam)));
    tau = (1.0 / (0.001000+(sum0/2))) * gamma(eng);
}

// Update sigma_u
{
    sigma_u = (1/sqrt(tau_u));
}

// Update u
for(int j=0; j<length(u); j++){
{
    double sum0=0;
    double sum1=0;
    for(int id_i=0; id_i<length(idmap_school[j]); id_i++) {          int i = idmap_school[j][id_i];
        double se10 = double(cons[int(i)]);
        double se11 = double(normexam[int(i)]);
        double se13 = double(standlrt[int(i)]);
        sum0+=(se10*(-se11)+(beta0*se10)+(beta1*se13));
    }
    for(int id_i=0; id_i<length(idmap_school[j]); id_i++) {          int i = idmap_school[j][id_i];
        double se14 = double(cons[int(i)]);
        sum1+=pow(se14,2);
    }

    std::tr1::normal_distribution<double> normal(-((tau*sum0)/(2*((tau_u/2)+((tau*sum1)/2)))), 1/sqrt((2*((tau_u/2)+((tau*sum1)/2))));
    u[j] = normal(eng);
}
}

// Update beta1
{
    double sum0=0;
    double sum1=0;
    for(int i=0; i<length(normexam); i++) {
        double se10 = double(standlrt[int(i)]);
```

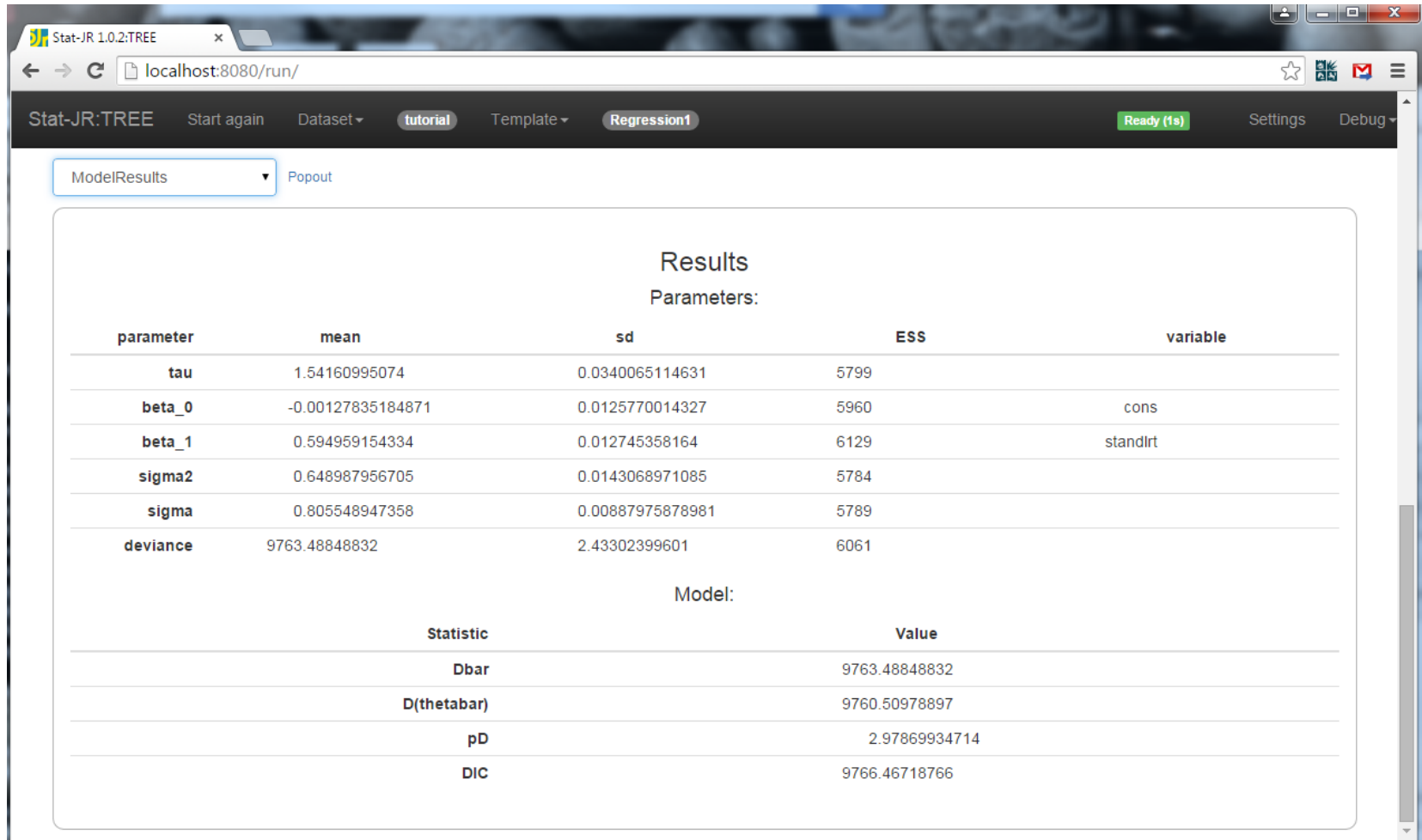
- The package can output C++ code that can then be taken away by software developers and modified.

Output of generated C++ code

```
// Update beta1
{
beta1 = dnorm((0.000249799765395*(2382.12631198+(beta0*(-
7.34783096611))))),(4003.20632175*tau));
}
// Update beta0
{
beta0 = dnorm(((((-0.462375992909)+((-
7.34783096611)*beta1))*0.000246366100025),(tau*4059.0));
}
```

- Note now that the code includes the actual data in place of constants and so looks less like the familiar algebraic expressions

Output from the E-STAT engine



The screenshot shows the Stat-JR 1.0.2:TREE web interface. The browser address bar shows 'localhost:8080/run/'. The interface includes a navigation bar with 'Stat-JR:TREE', 'Start again', 'Dataset', 'tutorial', 'Template', and 'Regression1'. A 'Ready (1s)' status indicator is visible. Below the navigation bar, there is a dropdown menu for 'ModelResults' and a 'Popout' button. The main content area displays the following results:

Results
Parameters:

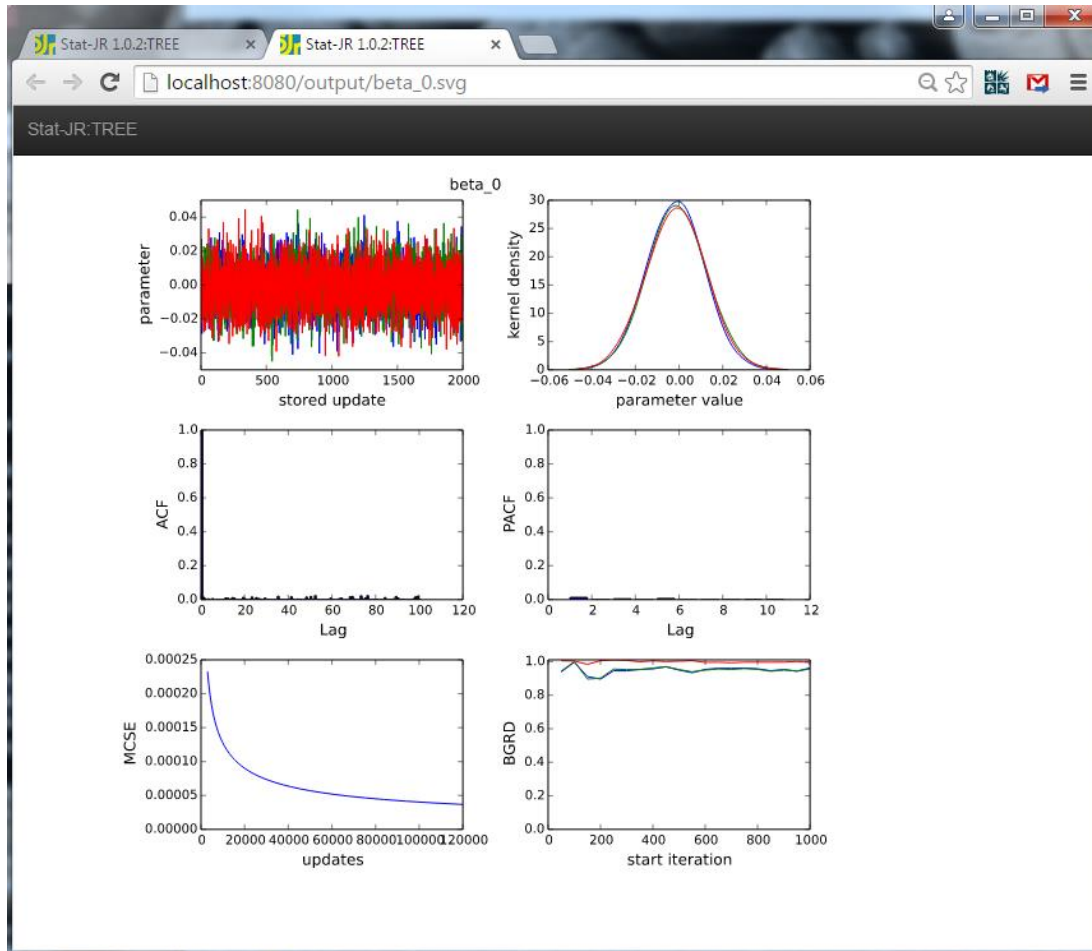
parameter	mean	sd	ESS	variable
tau	1.54160995074	0.0340065114631	5799	
beta_0	-0.00127835184871	0.0125770014327	5960	cons
beta_1	0.594959154334	0.012745358164	6129	standlirt
sigma2	0.648987956705	0.0143068971085	5784	
sigma	0.805548947358	0.00887975878981	5789	
deviance	9763.48848832	2.43302399601	6061	

Model:

Statistic	Value
Dbar	9763.48848832
D(thetabar)	9760.50978897
pD	2.97869934714
DIC	9766.46718766

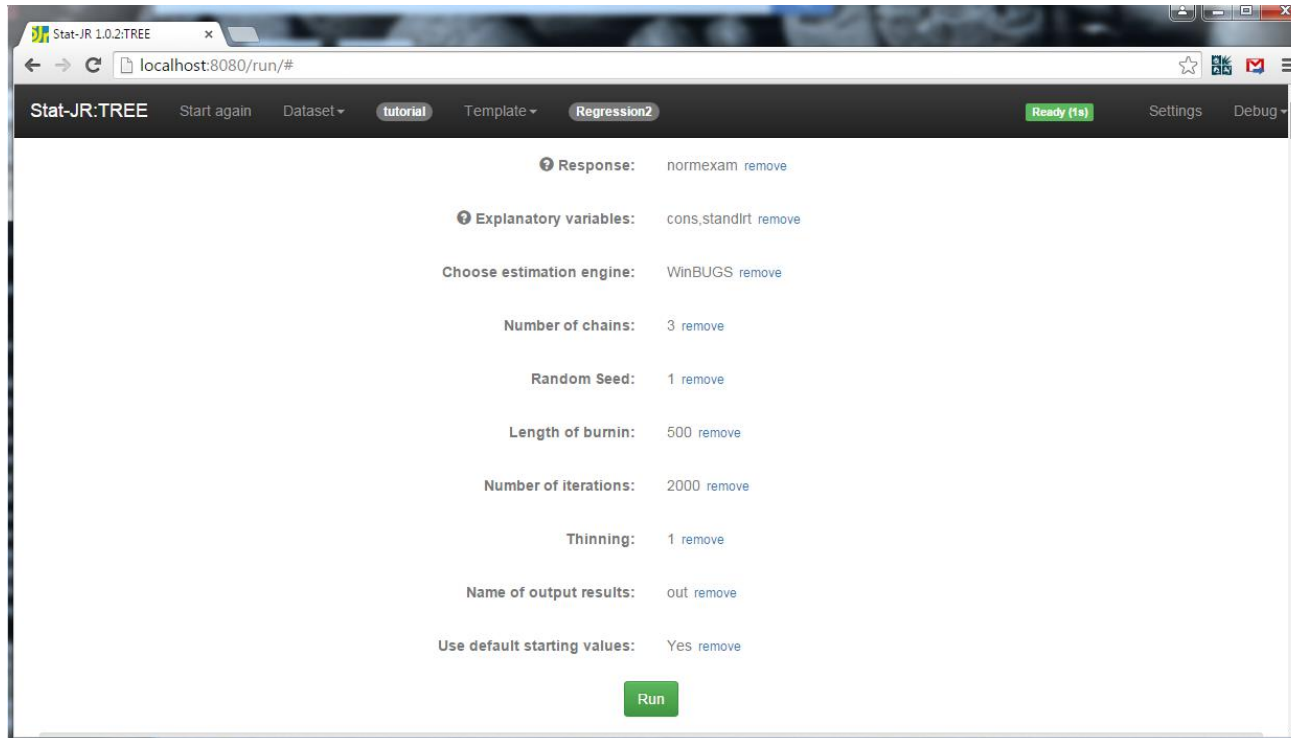
- Estimates and the DIC diagnostic can be viewed for the model fitted.

Output from the E-STAT engine



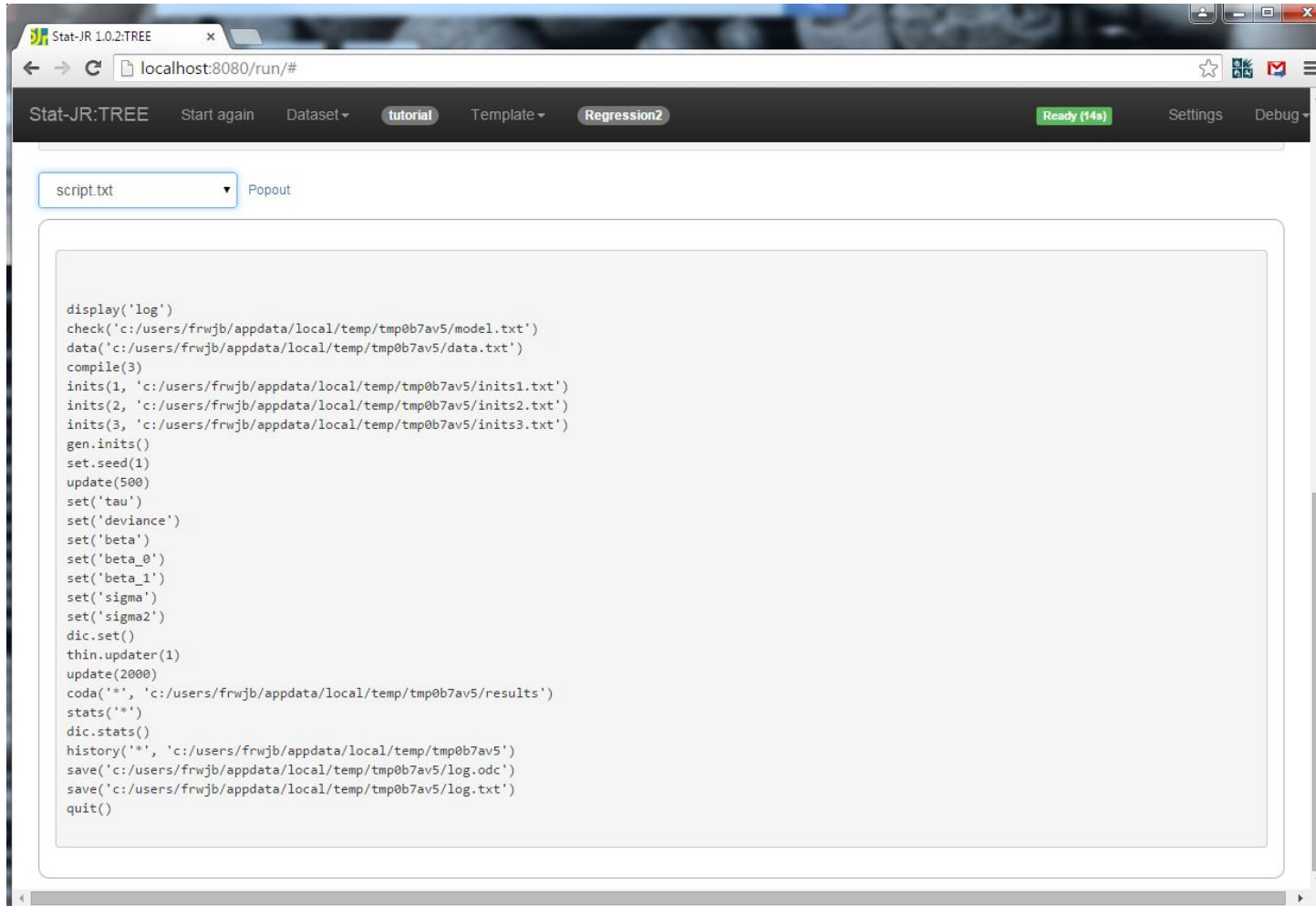
- E-STAT offers multiple chains so that we can use multiple chain diagnostics to aid convergence checking.
- Graphics are in svg format so scale nicely.

Interoperability with WinBUGS (Regression 2)



- This template offers the choice of many software packages for fitting a regression model.
- STAT-JR checks what is installed on the machine and only offers packages that are installed. Here we choose WinBUGS.
- Interoperability in the user interface is obtained via a few extra inputs. In fact in the template code user written functions are required for all packages apart from WinBUGS, OpenBUGS and JAGS. The transfer of data between packages is however generic.

Interoperability with WinBUGS (Regression 2)



The screenshot shows a web browser window with the address bar at localhost:8080/run/#. The page title is Stat-JR: TREE. The interface includes a navigation bar with 'Start again', 'Dataset', 'tutorial', 'Template', and 'Regression2'. A 'Ready (14s)' indicator is visible. A dropdown menu shows 'script.txt' and a 'Popout' button. The main content area displays a script for running WinBUGS, including commands for file paths, compilation, initialization, parameter setting, and saving results.

```
display('log')
check('c:/users/frwjb/appdata/local/temp/tmp0b7av5/model.txt')
data('c:/users/frwjb/appdata/local/temp/tmp0b7av5/data.txt')
compile(3)
inits(1, 'c:/users/frwjb/appdata/local/temp/tmp0b7av5/inits1.txt')
inits(2, 'c:/users/frwjb/appdata/local/temp/tmp0b7av5/inits2.txt')
inits(3, 'c:/users/frwjb/appdata/local/temp/tmp0b7av5/inits3.txt')
gen.inits()
set.seed(1)
update(500)
set('tau')
set('deviance')
set('beta')
set('beta_0')
set('beta_1')
set('sigma')
set('sigma2')
dic.set()
thin.updater(1)
update(2000)
coda('*', 'c:/users/frwjb/appdata/local/temp/tmp0b7av5/results')
stats('*')
dic.stats()
history('*', 'c:/users/frwjb/appdata/local/temp/tmp0b7av5')
save('c:/users/frwjb/appdata/local/temp/tmp0b7av5/log.odc')
save('c:/users/frwjb/appdata/local/temp/tmp0b7av5/log.txt')
quit()
```

- Here we can view the files required to run WinBUGS in the pane (script file shown but model, inits and data also available)
- The model can be run by press of a button.

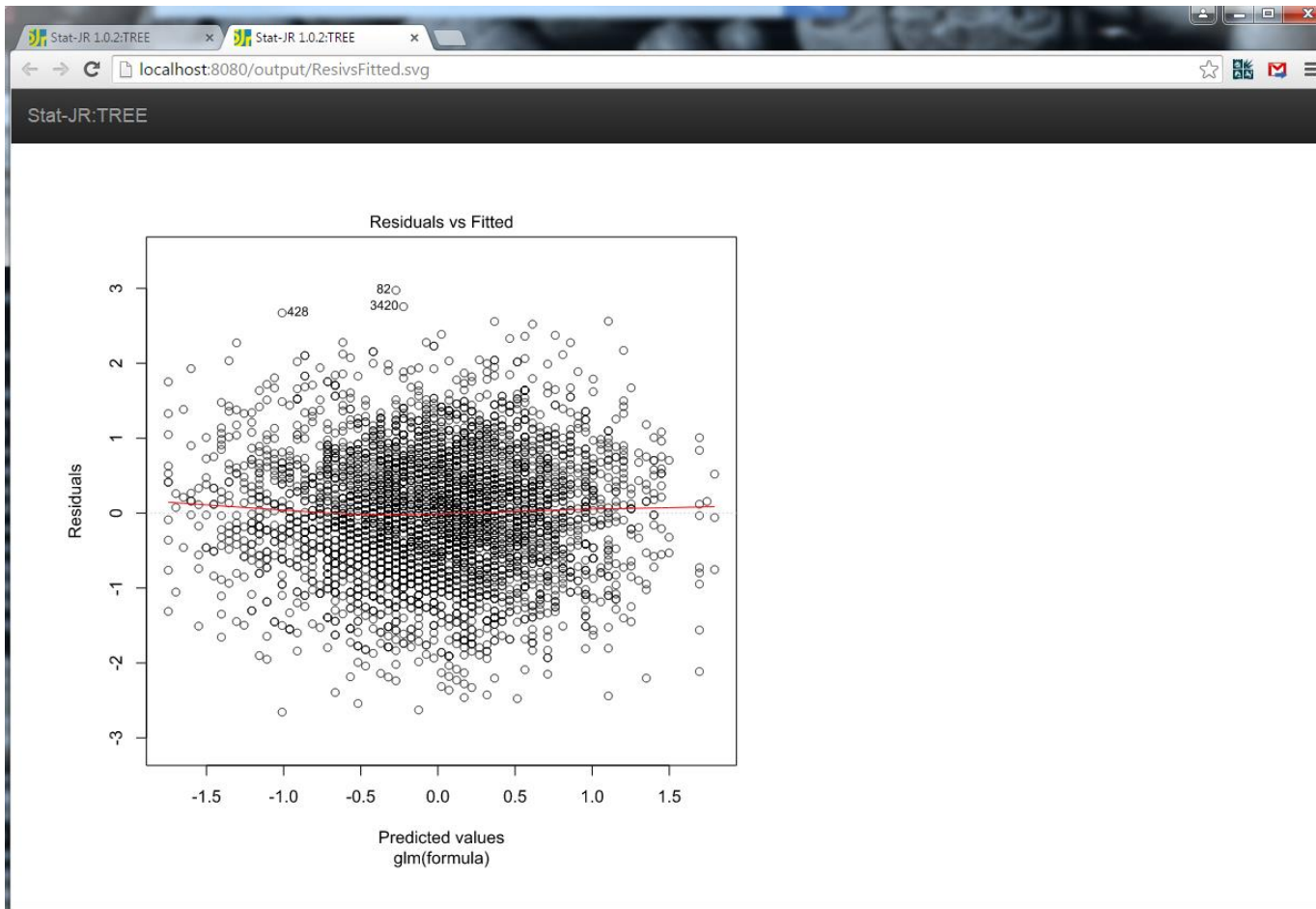
Interoperability with R

The screenshot shows the Stat-JR 1.0.2:TREE web interface. The browser address bar shows 'localhost:8080/run/#'. The interface has a dark header with 'Stat-JR: TREE', 'Start again', 'Dataset', 'tutorial', 'Template', 'Regression2', 'Ready (19)', 'Settings', and 'Debug'. Below the header, there are three configuration sections: 'Response: normexam remove', 'Explanatory variables: cons,standlrt remove', and 'Choose estimation engine: R_glm remove'. A green 'Run' button is centered below these sections. Below the 'Run' button is a 'Current input string' field containing '{'y': 'normexam', 'x': 'cons,standlrt', 'Engine': 'R_glm'}' and a 'Set' button. Below that is a 'Command' field containing 'RunStatJR(template='Regression2', dataset='tutorial', invars = {'y': 'normexam', 'x': 'cons,standlrt'}, estoptions = {'Engine': 'R_glm'})'. Below the command field is an 'Edit' button and a dropdown menu showing 'script.R' with a 'Popout' button. The main content area is a code editor showing an R script:

```
local({r <- getOption("repos"); r["CRAN"] <- "http://cran.r-project.org"; options(repos = r)})
#####
# Note that when Stat-JR interoperates with R, it sets the working
# directory to wherever the user's temporary files are stored, i.e.
# workdir = tempdir(). The data to be modelled, this script, and the
# files exported from R, are all saved there.
#####
# test to see if foreign package is already installed, if not, then install it
if (!require(foreign)) {
  install.packages("foreign")
  library(foreign)
}
```

- R can be chosen as another alternative. In fact here we have 2 choices – glm or MCMCglmm.
- You will see in the pane the script file ready for input to R. There will also be the data file that R requires.

Interoperability with R



- If written in to the code in the template – graphics from other software can be extracted.
- Here for example is a residual plot associated with the R fit of the model.

Other templates - XYplot



- There are also templates for plotting. For example here is a plot using the XYplot template.
- Shown is the plot whilst the Python command script is also available.
- For more details on StatJR go to <http://www.bristol.ac.uk/cmm/software/statjr/>

For more information visit
www.ncrm.ac.uk

