Professor William Browne
School of Veterinary Science and Centre for Multilevel Modelling (& many co-workers!)

The STAT-JR package and it's potential use with social network models

University of BRISTOL

E·S·R·C ECONOMIC & SOCIAL RESEARCH COUNCIL

ESRC National Centre for Research Methods

Summary

- Background to STAT-JR package
- Some screen shots of the program features
- Multiple membership models for spatial models/ social networks
- STAT-JR named in memory of Jon Rasbash whose ideas started project.

University of BRISTOL

The E-STAT project and STAT-JR

STAT-JR developed jointly by LEMMA II and E-STAT ESRC nodes

Consists of a set of components many of which we have an alpha version for which contains:

Templates for model fitting, data manipulation, input and output controlled via a web browser interface.

Currently model fitting for 90% of the models that MLwiN can fit in MCMC plus some it can’t including greatly sped up REALCOM templates

Some interoperability with MLwiN, WinBUGS, R, Stata and SPSS (written by Camille Szmaragd)

University of BRISTOL

STAT-JR

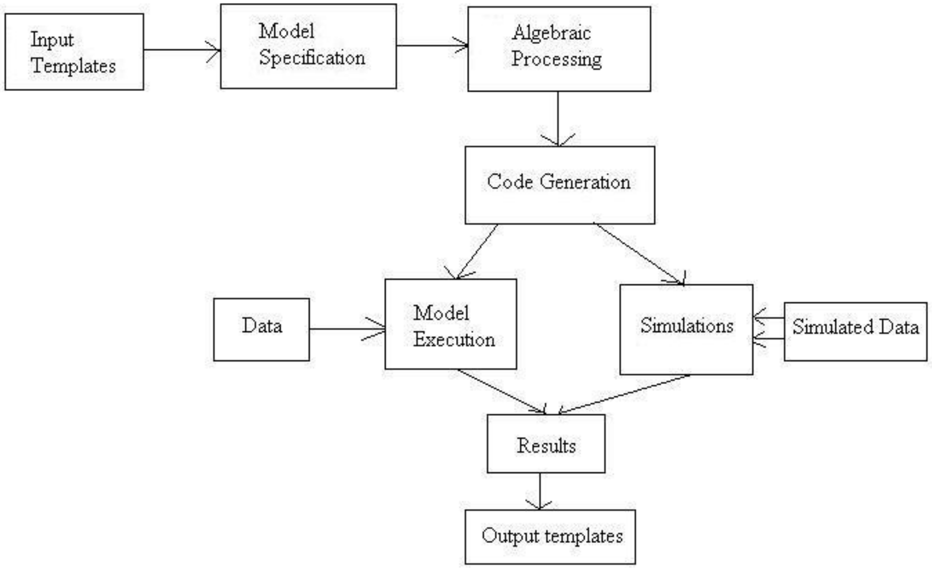
Jon identified 3 groups of users:

- Novice practitioners who want to use statistical software that is user friendly and maybe tailored to their discipline
- Advanced practitioners who are the experts in their fields and also want to develop tools for the novice practitioners
- Algorithm Developers who want their algorithms used by practitioners.
- See http://www.cmm.bristol.ac.uk/research/NCESS-EStat/news.shtml for details of Advanced User's guide for STAT-JR

University of BRISTOL

STAT-JR 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 currently a VB.net algebra processing system. A team of coders (currently me, Chris, Danius, Camille and Bruce) work together on the system.



Templates

- Consist of a set of code sections for advanced users to write.

For a model template it consists of at least:

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

Other optional functions required for more complex templates

University of BRISTOL

Regression 1 Example

```
from EStat.Templating import *
from mako.template import Template as
        MakoTemplate
import re

class Regression1(Template):
    'A model template for fitting 1 level Normal multiple
        regression model in E-STAT only. To be used in
        documentation.'

    tags = [ 'model' , '1-Level' ]

    invars = '''
y = DataVector('response: ')
tau = ParamScalar()
sigma = ParamScalar()
x = DataMatrix('explanatory variables: ')
beta = ParamVector()
beta.ncols = len(x)
'''
```

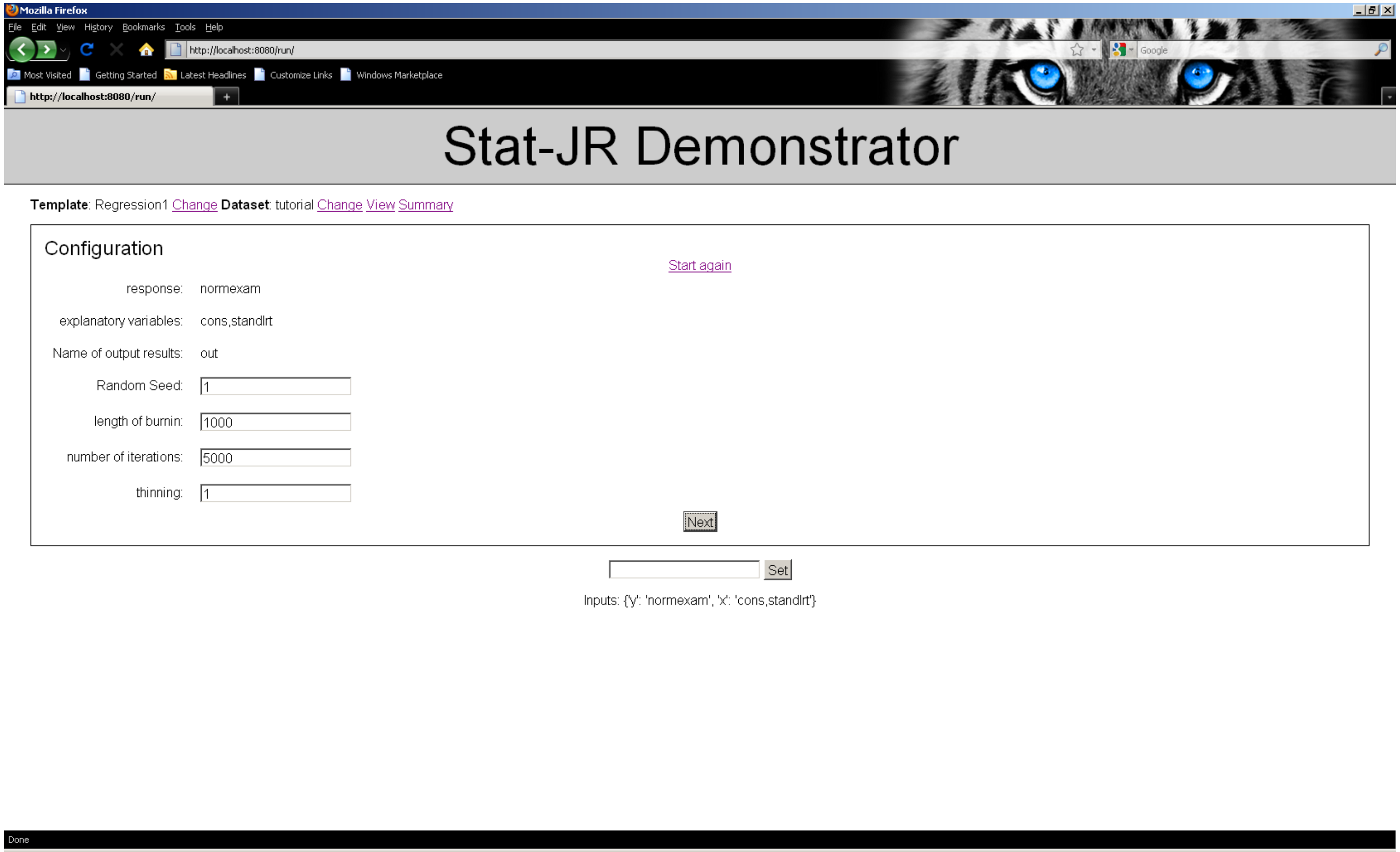
```
outbug = '''
model{
    for (i in 1:length(${y})) {
        ${y}[i] ~ dnorm(mu[i], tau)
        mu[i] <- ${mmult(x, 'beta', 'i')}
    }

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

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

Invars function

'''

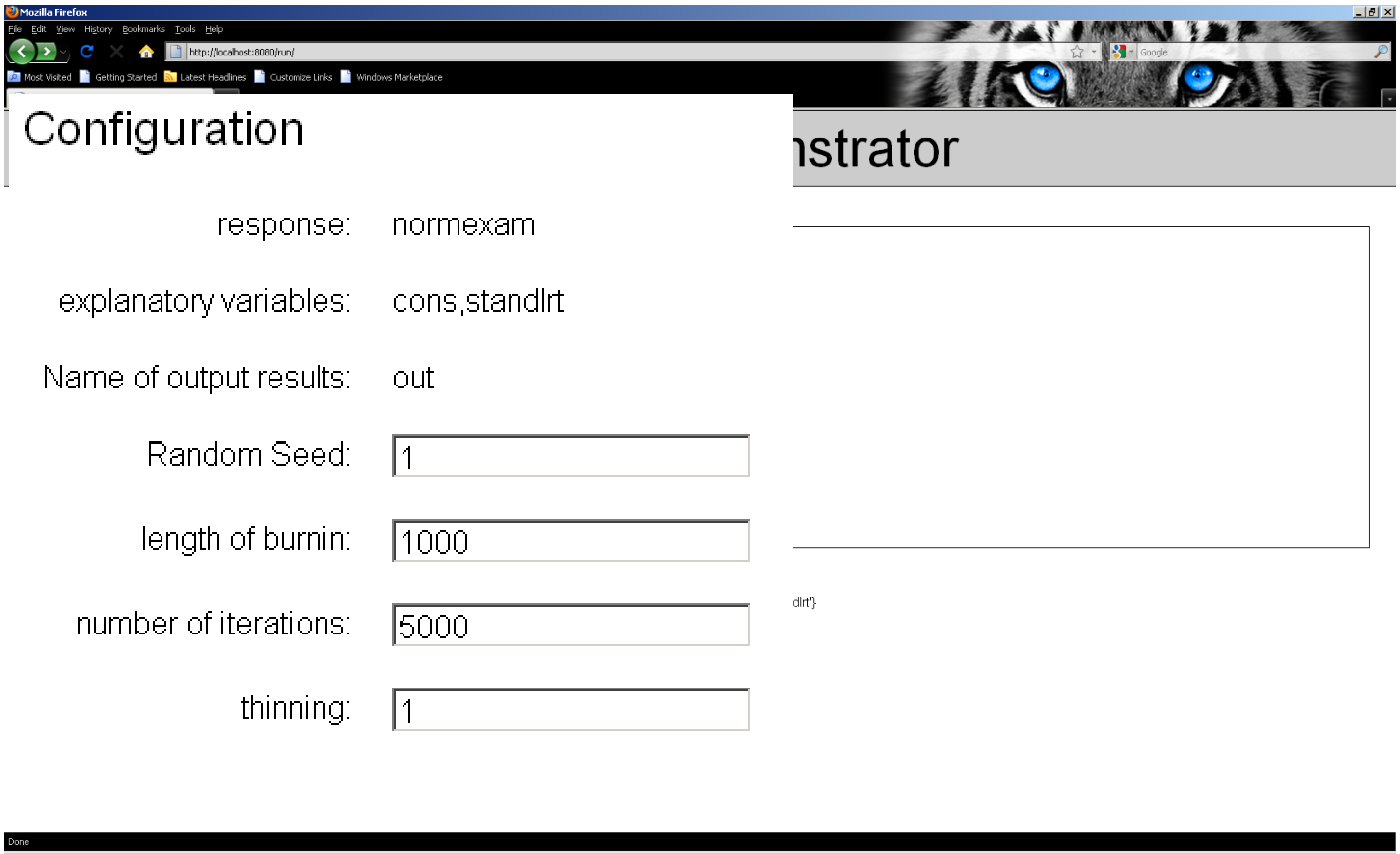
```
  invars = '''
y = DataVector('response: ')
tau = ParamScalar()
sigma = ParamScalar()
x = DataMatrix('explanatory variables: ')
beta = ParamVector()
beta.ncols = len(x)
'''
```

An example of STAT-JR – setting up a model



University of BRISTOL

An example of STAT-JR – setting up a model



University of BRISTOL

Equations for model and model code

Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://127.0.0.1:8080/run/

thinning: 1

Equation rendering

$$\begin{aligned}
\mathrm{normexam}_i &\sim \mathrm{N}(\mu_i, \sigma^2) \\
\mu_i &= \beta_0 \mathrm{cons}_i + \beta_1 \mathrm{standlrt}_i + u_{\mathrm{school}[i]} \\
u_{\mathrm{school}[i]} &\sim \mathrm{N}(0, \sigma_u^2) \\
\beta_0 &\propto 1 \\
\beta_1 &\propto 1 \\
\tau &\sim \Gamma(0.001, 0.001) \\
\sigma^2 &= 1/\tau \\
\tau_u &\sim \Gamma(0.001, 0.001) \\
\sigma_u^2 &= 1/\tau_u
\end{aligned}$$

Model

```
model {
    for (i in 1:length(normexam)) {
        normexam[i] ~ dnorm(mu[i], tau)
        mu[i] <- cons[i] * beta0 + standlrt[i] * beta1 + u[school[i]] * cons[i]
    }

    for (j in 1:length(u)) {
        u[j] ~ dnorm(0, tau_u)
    }

    # Priors
    beta0 ~ dflat()
    beta1 ~ dflat()

    tau ~ dgamma(0.001000, 0.001000)
    sigma <- 1 / sqrt(tau)

    tau_u ~ dgamma(0.001000, 0.001000)
    sigma_u <- 1 / sqrt(tau_u)
}
```

Run | Simulate | Selection | Simulate | Specify starting Values | Code | JS | Set

Inputs: {'y': 'normexam', 'L2ID': 'school', 'D': 'Normal', 'x': 'cons,standlrt'}

Done

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. This is a more complex template for 2 level models.

University of BRISTOL

Equations for model and model code

$$
\begin{aligned}
\text{normexam}_i &\sim \text{N}(\mu_i, \sigma^2) \\
\mu_i &= \beta_0 \text{cons}_i + \beta_1 \text{standlrt}_i + u_{\text{school}[i]} \\
u_{\text{school}[i]} &\sim \text{N}(0, \sigma_u^2) \\
\beta_0 &\propto 1 \\
\beta_1 &\propto 1 \\
\tau &\sim \Gamma(0.001, 0.001) \\
\sigma^2 &= 1/\tau \\
\tau_u &\sim \Gamma(0.001, 0.001) \\
\sigma_u^2 &= 1/\tau_u
\end{aligned}
$$

inputs: {y: normexam, L2ID: school, D: Normal, x: cons,standlrt}

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. This is a more complex template for 2 level models.

University of BRISTOL

Outbug function

```
outbug = '''
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)
    sigma <- 1 / sqrt(tau)
}
        '''
```

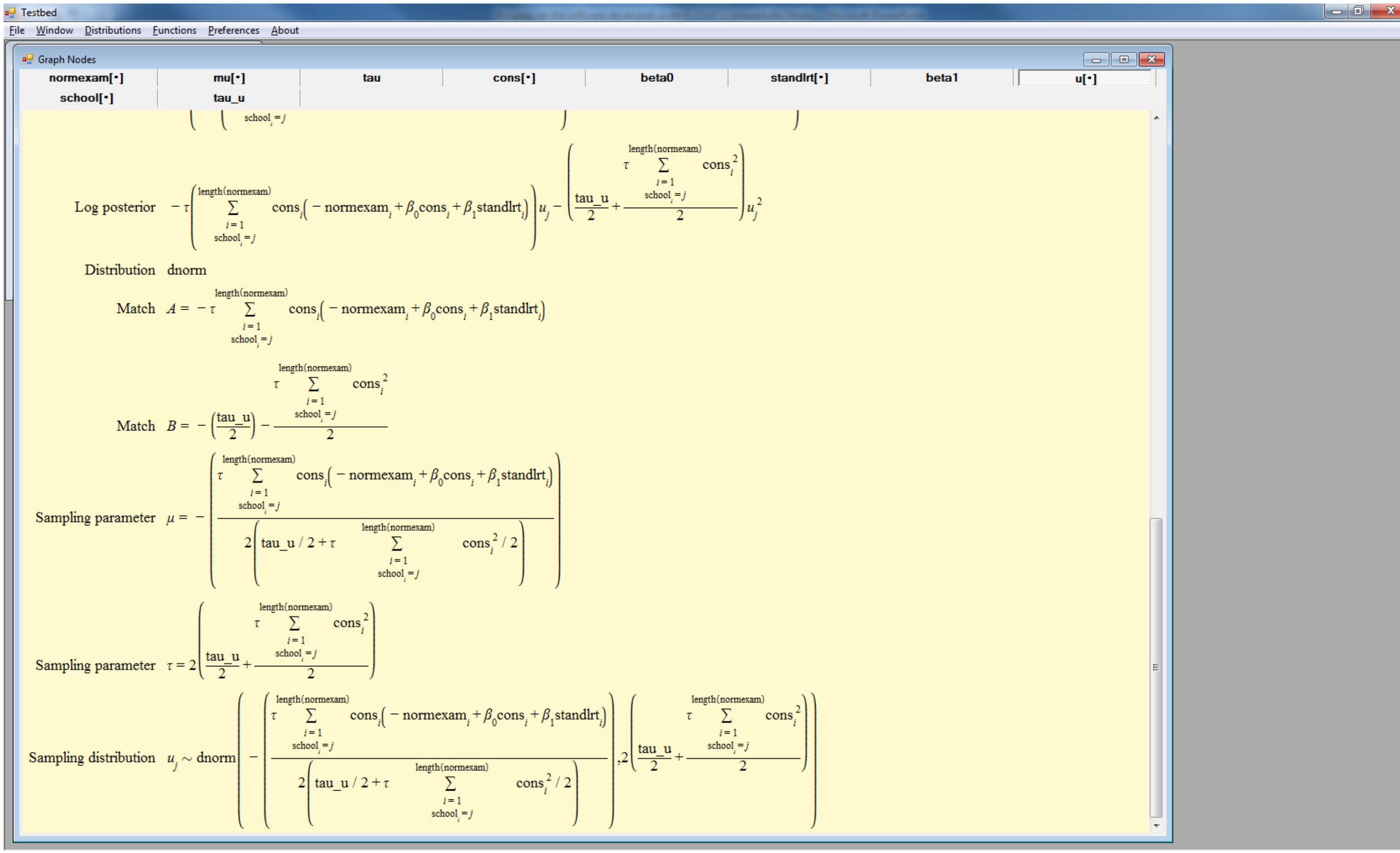
Model code in detail

```
model {
  for (i in 1:length(normexam)) {
    normexam[i] ~ dnorm(mu[i], tau)
    mu[i] <- cons[i] * beta0 + standlrt[i] * beta1 + u[school[i]] * cons[i]
   }
  for (j in 1:length(u)) {
    u[j] ~ dnorm(0, tau_u)
  }
# Priors
  beta0 ~ dflat()
  beta1 ~ dflat()
  tau ~ dgamma(0.001000, 0.001000)
  tau_u ~ dgamma(0.001000, 0.001000)
 }
```

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

University of BRISTOL

Bruce’s (Demo) algebra system step for parameter u



University of BRISTOL

Bruce's (Demo) algebra system step for parameter u

$$\text{Sampling parameter} \quad \mu = -\left(\frac{\tau \sum_{\substack{i=1 \\ \text{school}_i = j}}^{\text{length(normexam)}} \text{cons}_i \left(-\text{normexam}_i + \beta_0 \text{cons}_i + \beta_1 \text{standlrt}_i \right)}{2\left(\text{tau\_u} / 2 + \tau \sum_{\substack{i=1 \\ \text{school}_i = j}}^{\text{length(normexam)}} \text{cons}_i^2 / 2 \right)} \right)$$

$$\text{Sampling parameter} \quad \tau = 2\left(\frac{\text{tau\_u}}{2} + \frac{\tau \sum_{\substack{i=1 \\ \text{school}_i = j}}^{\text{length(normexam)}} \text{cons}_i^2}{2} \right)$$

University of BRISTOL

Output of generated C++ code

```
void iterate() {
// Update tau
    {
                    double sum0=0;
            for(int i=0; i<length(normexam); i++) {
                    double sel0 = double(normexam[int(i)]);
                    double sel1 = double(cons[int(i)]);
                    double sel2 = double(standlrt[int(i)]);
                    double sel5 = double(school[int(i)]);
                    double sel4 = double(u[int(sel5)]);
                    sum0+=pow((((sel0-(beta0*sel1))-(beta1*sel2))-(sel1*sel4)),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 sel0 = double(cons[int(i)]);
                    double sel1 = double(normexam[int(i)]);
                    double sel3 = double(standlrt[int(i)]);
                    sum0+=(sel0*((-sel1)+(beta0*sel0)+(beta1*sel3)));
            }
            for(int id_i=0; id_i<length(idmap_school[j]); id_i++) { int i = idmap_school[j][id_i];
                    double sel4 = double(cons[int(i)]);
                    sum1+=pow(sel4,2);
            }


        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 sel0 = double(standlrt[int(i)]);
```

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

University of BRISTOL

Output of generated C++ code

Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/run/ manuscript central jrss

```
// 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 sel0 = double(cons[int(i)]);
                        double sel1 = double(normexam[int(i)]);
                        double sel3 = double(standlrt[int(i)]);
                        sum0+=(sel0*((-sel1)+(beta0*sel0)+(beta1*sel3)));
                }
                for(int id_i=0; id_i<length(idmap_school[j]); id_i++) { int i = idmap_school[j][id_i];
                        double sel4 = double(cons[int(i)]);
                        sum1+=pow(sel4,2);
                }


            std::tr1::normal_distribution<double> normal((-((tau*sum0)/(2*((tau_u/2)+((tau*sum1)/2))))), 1/sqrt(

            u[j] = normal(eng);

    }}
```

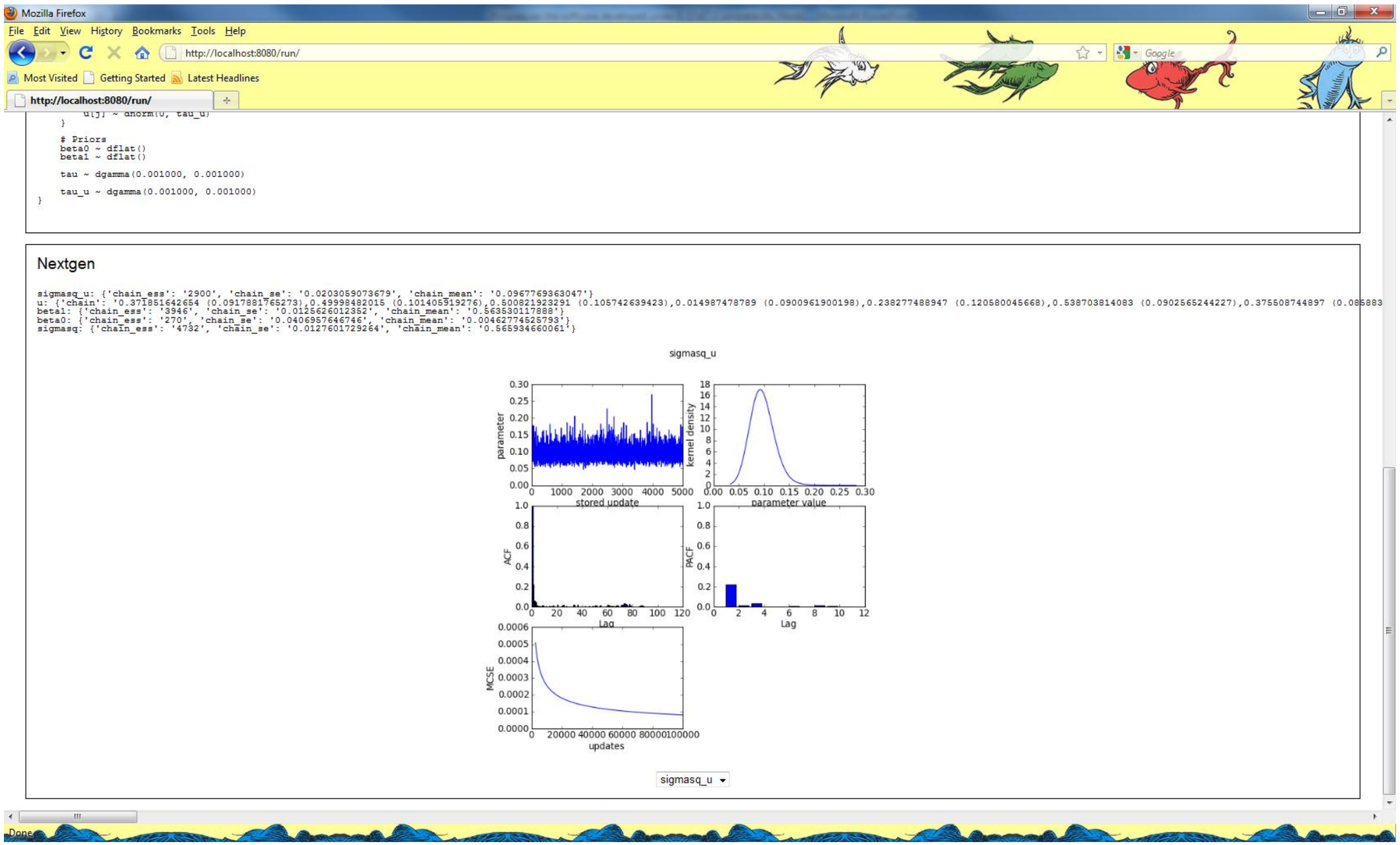
```
// Update beta1
    {
                        double sum0=0;
                double sum1=0;
                for(int i=0; i<length(normexam); i++) {
                        double sel0 = double(standlrt[int(i)]);
```

Done

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

University of BRISTOL

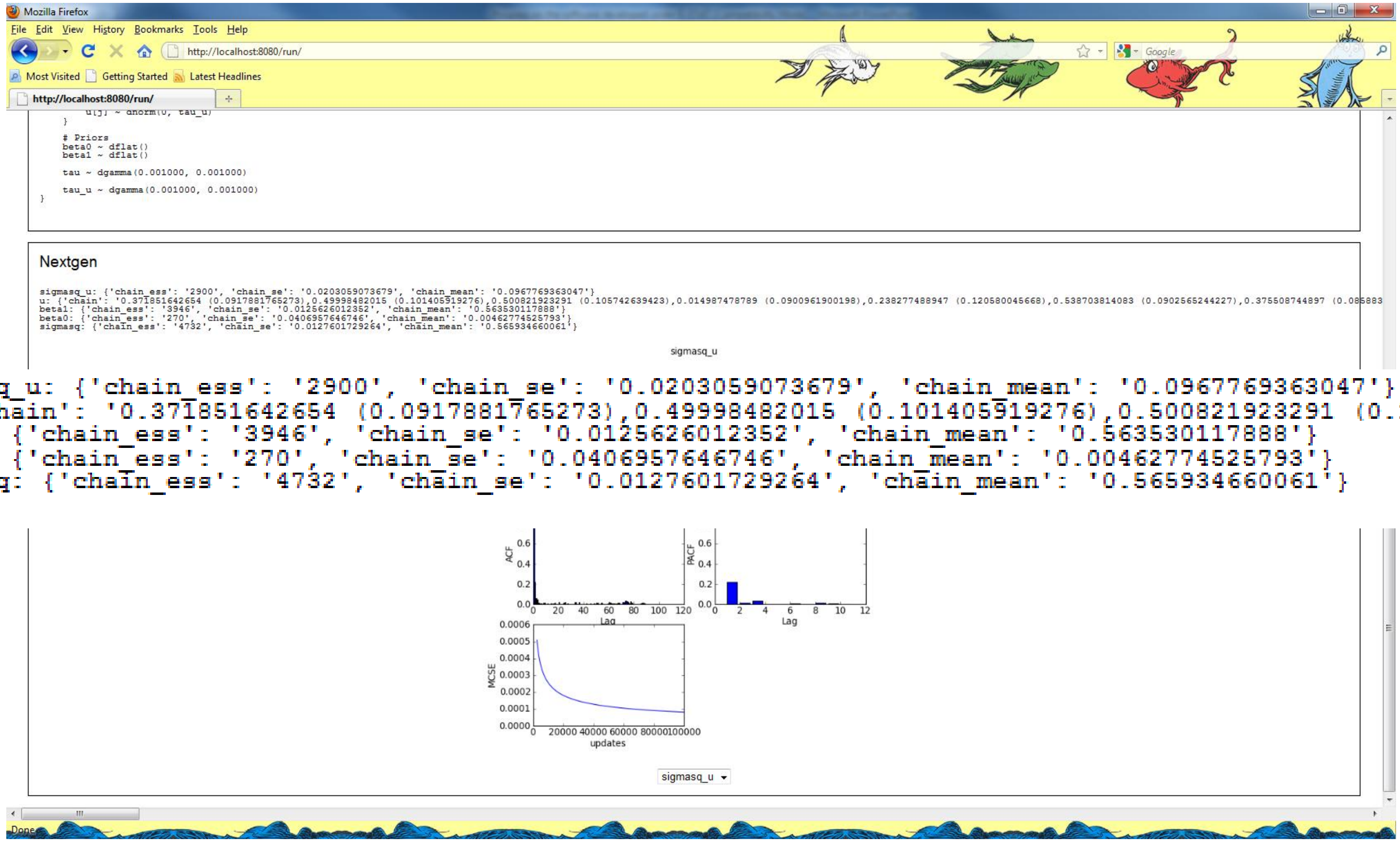
Output from the E-STAT engine



Here the six-way plot functionality is in part taken over to STAT-JR after the model has run. In fact graphs for all parameters are calculated and stored as picture files so can be easily viewed quickly.

University of BRISTOL

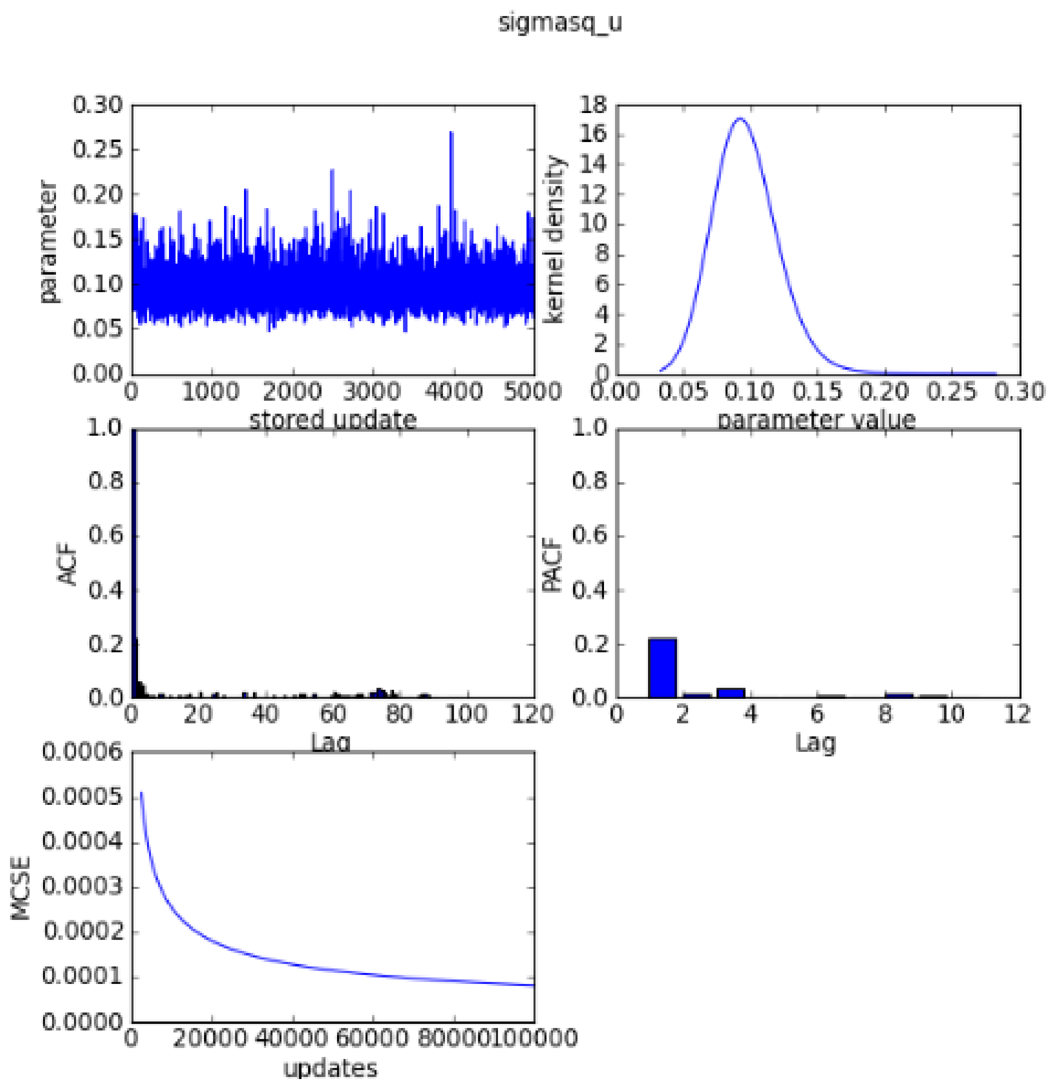
Output from the E-STAT engine



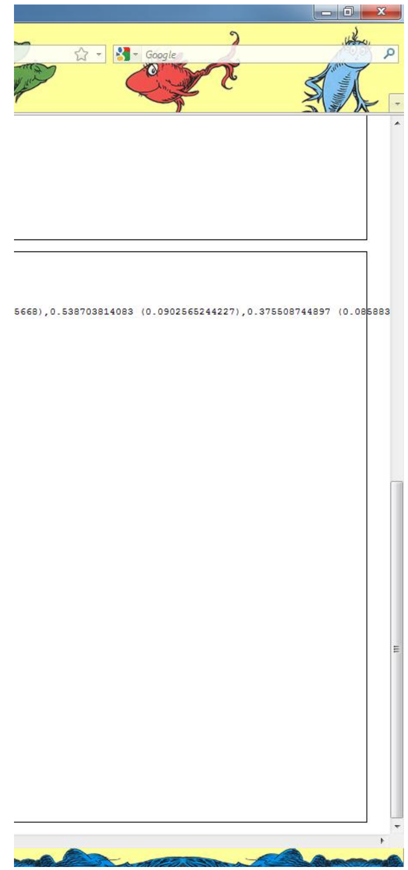

```
sigmasq_u: {'chain_ess': '2900', 'chain_se': '0.0203059073679', 'chain_mean': '0.0967769363047'}
u: {'chain': '0.371851642654 (0.0917881765273),0.49998482015 (0.101405919276),0.500821923291 (0.105742639423),
beta1: {'chain_ess': '3946', 'chain_se': '0.0125626012352', 'chain_mean': '0.563530117888'}
beta0: {'chain_ess': '270', 'chain_se': '0.0406957646746', 'chain_mean': '0.00462774525793'}
sigmasq: {'chain_ess': '4732', 'chain_se': '0.0127601729264', 'chain_mean': '0.565934660061'}
```

Here the six-way plot functionality is in part taken over to STAT-JR after the model has run. In fact graphs for all parameters are calculated and stored as picture files so can be easily viewed quickly.

Output from the E-STAT engine



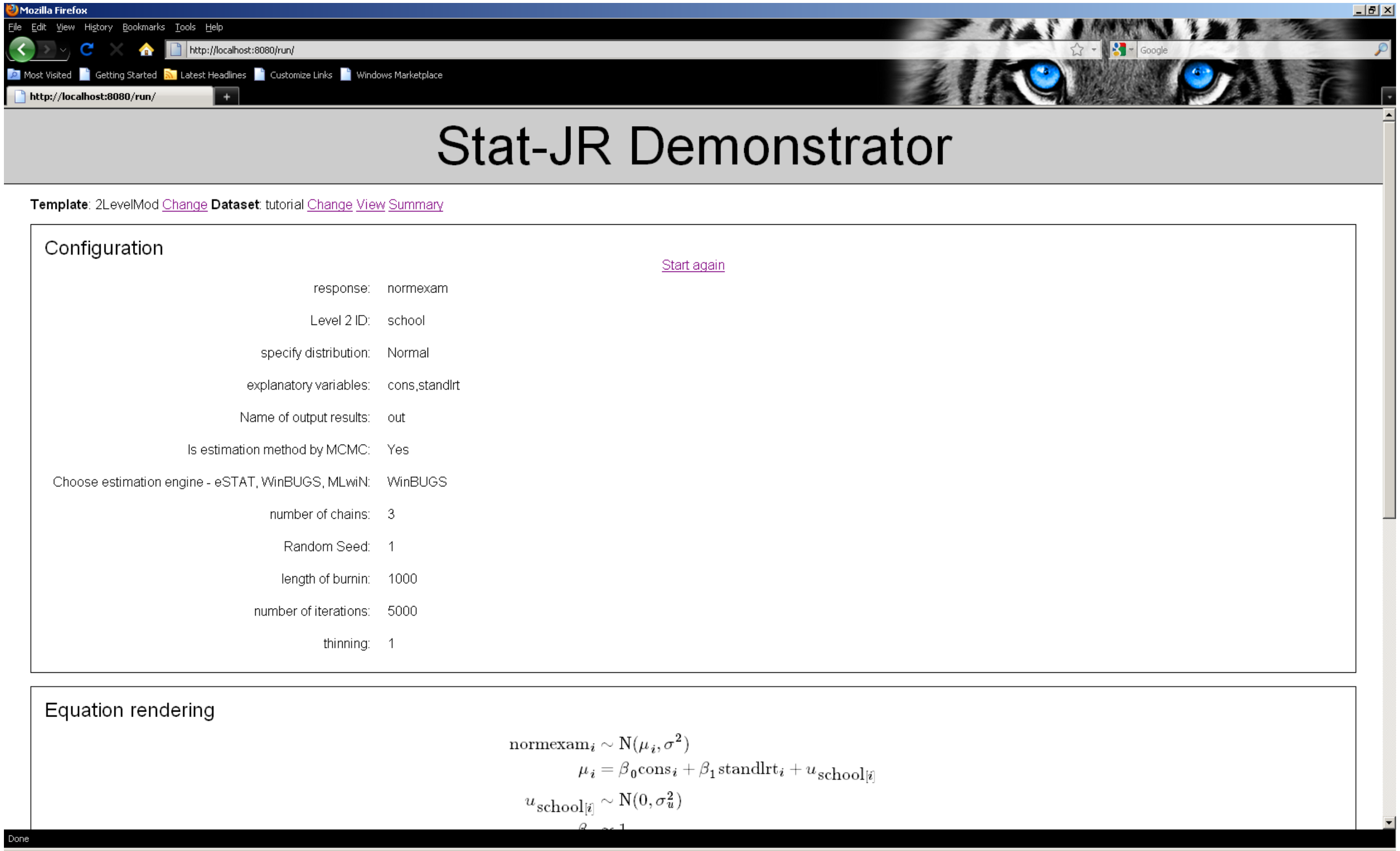
can be easily viewed quickly.



to

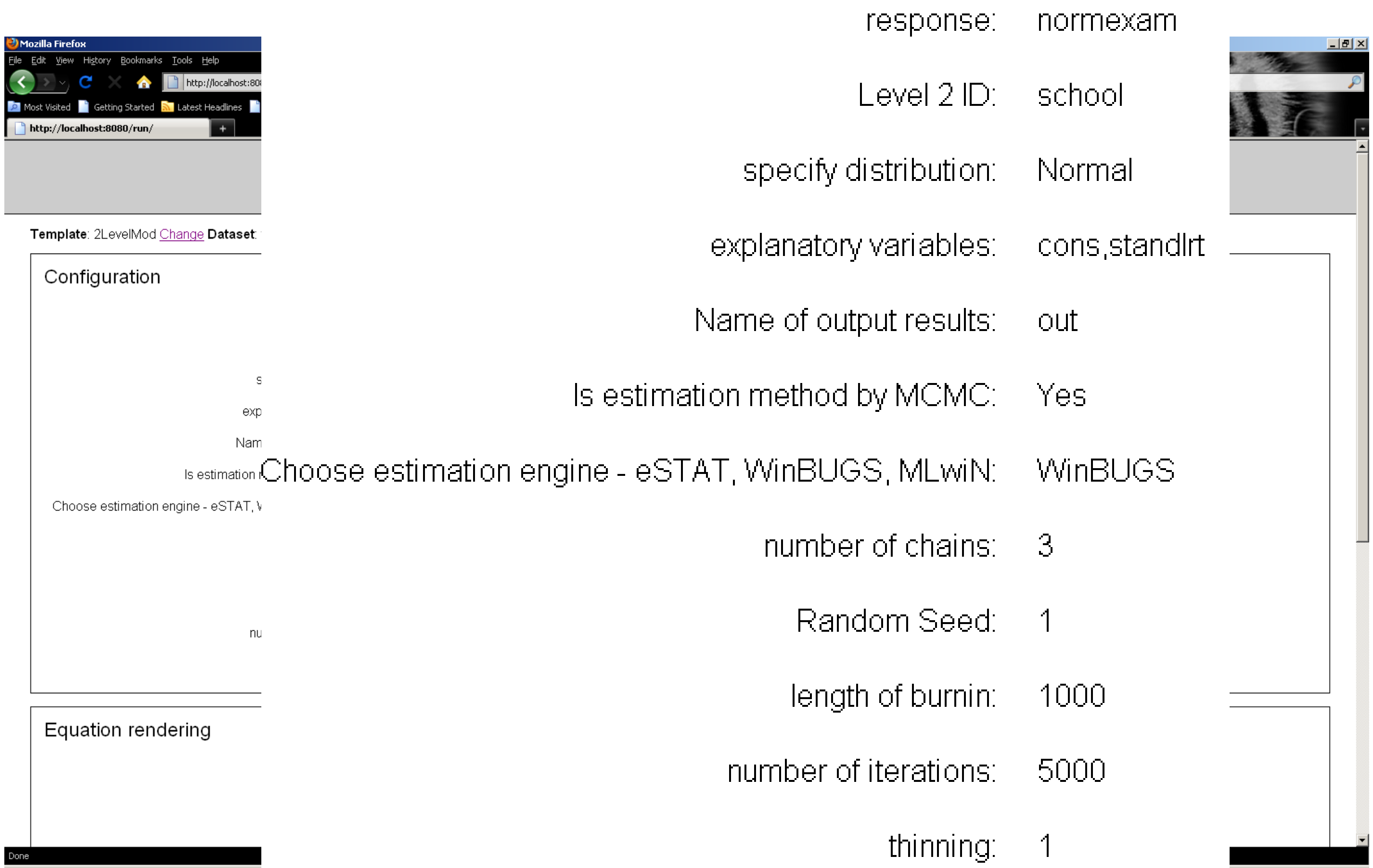
o

Interoperability with 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. The transfer of data between packages is however generic.

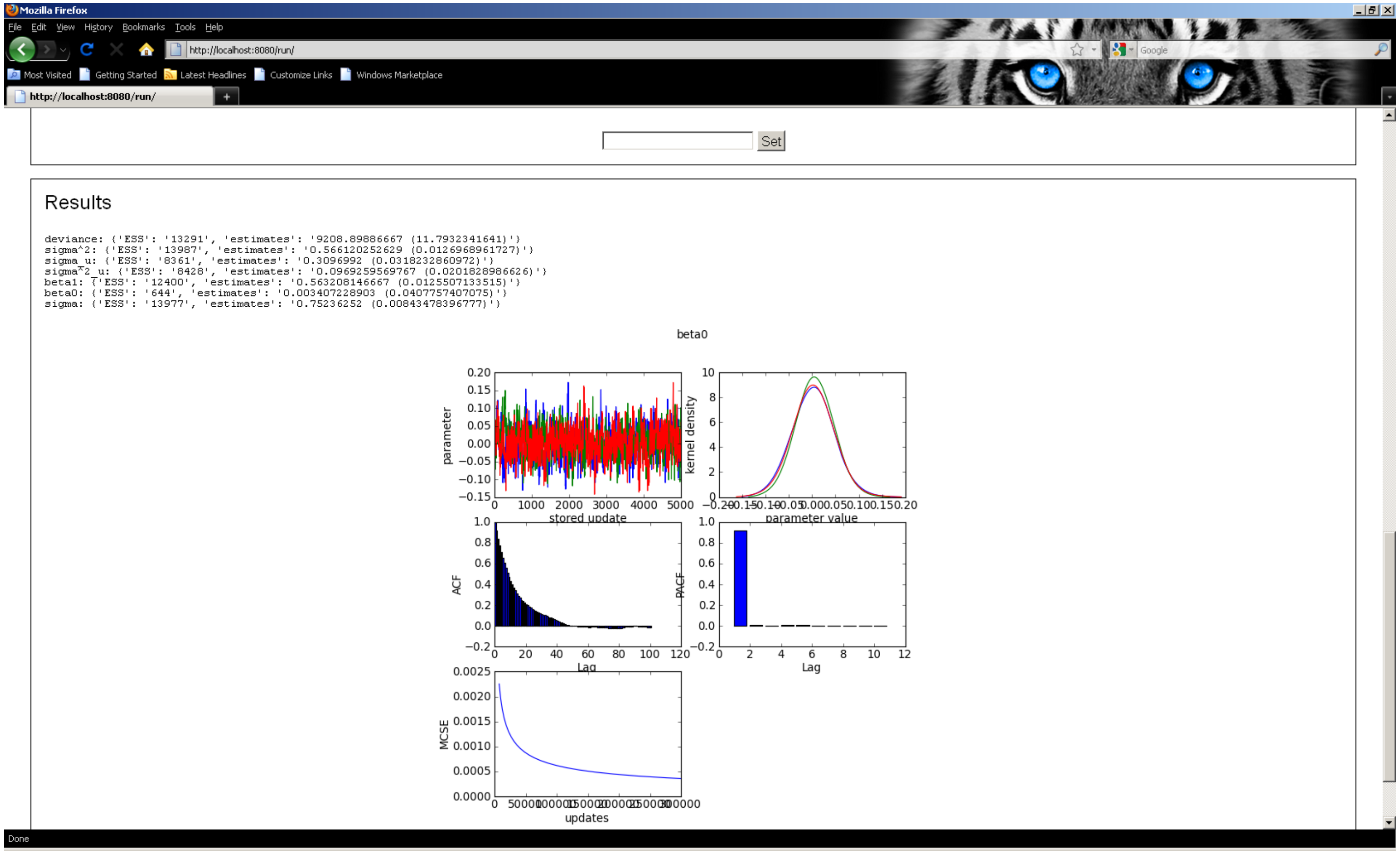
Interoperability with WinBUGS



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. The transfer of data between packages is however generic.

University of BRISTOL

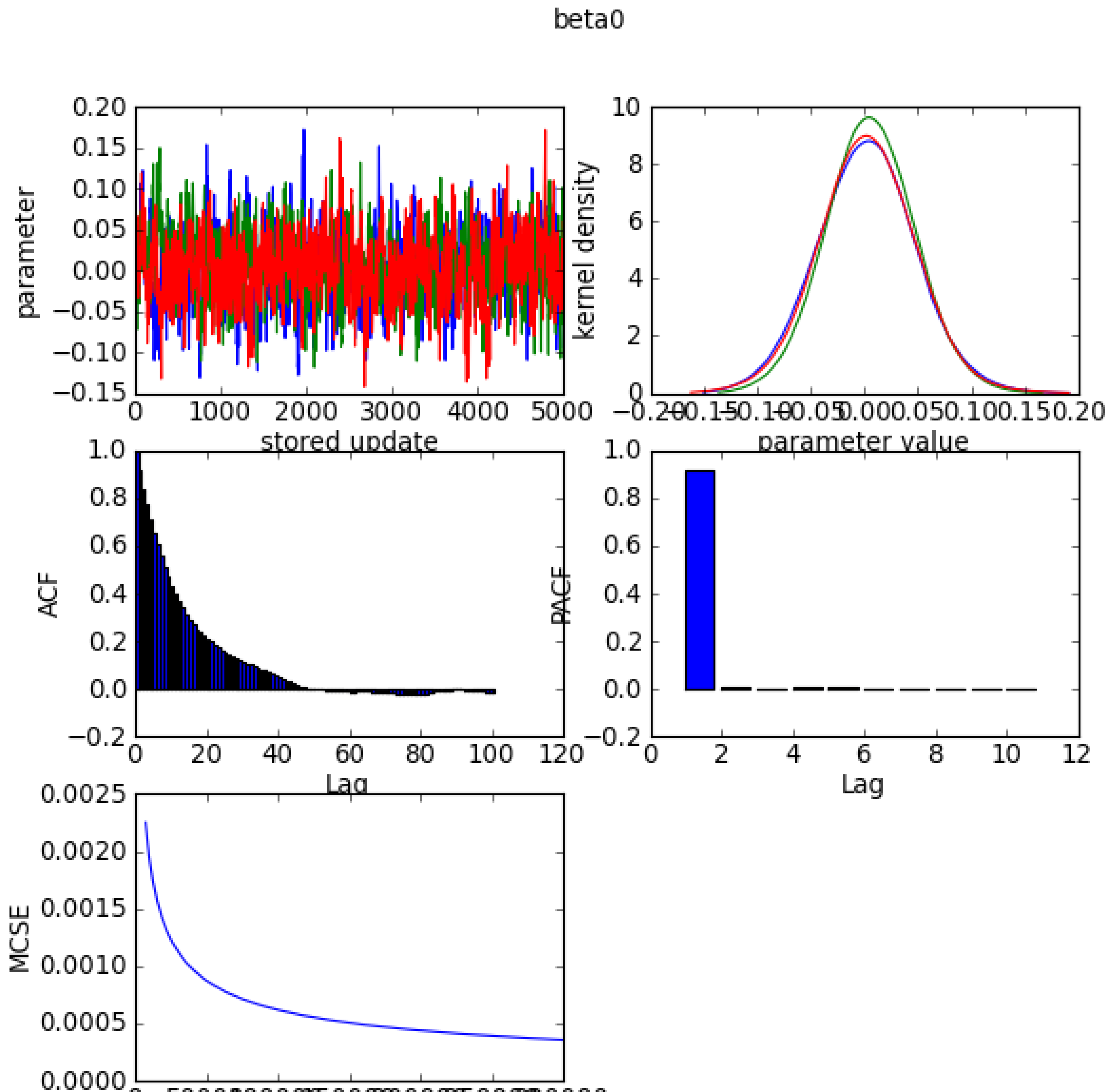
Output from WinBUGS with multiple chains



STAT-JR generates appropriate files and then fires up WinBUGS. Multiple Chains are superimposed in the sixway plot output.

University of BRISTOL

Output from WinBUGS with multiple chains



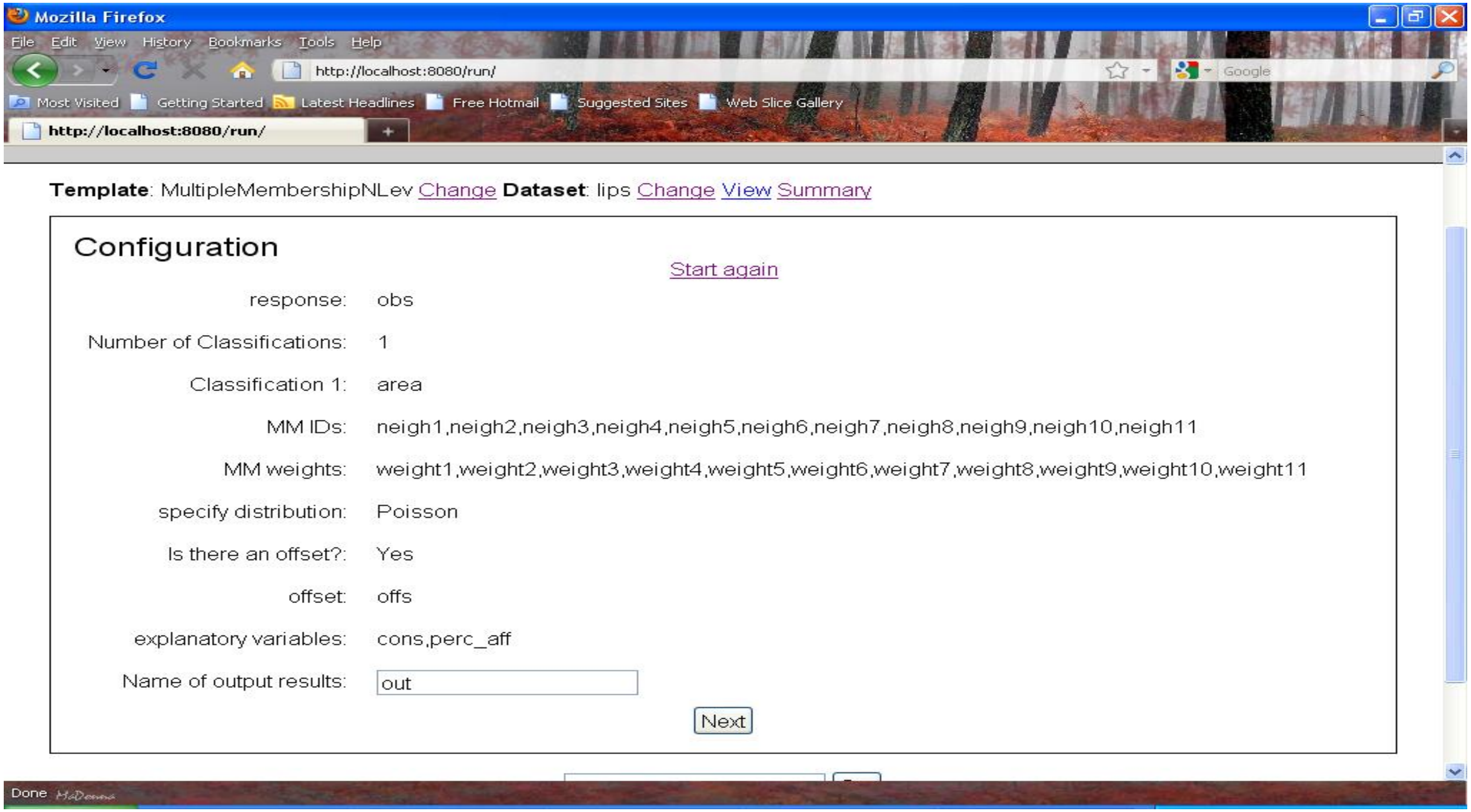
ot

output.

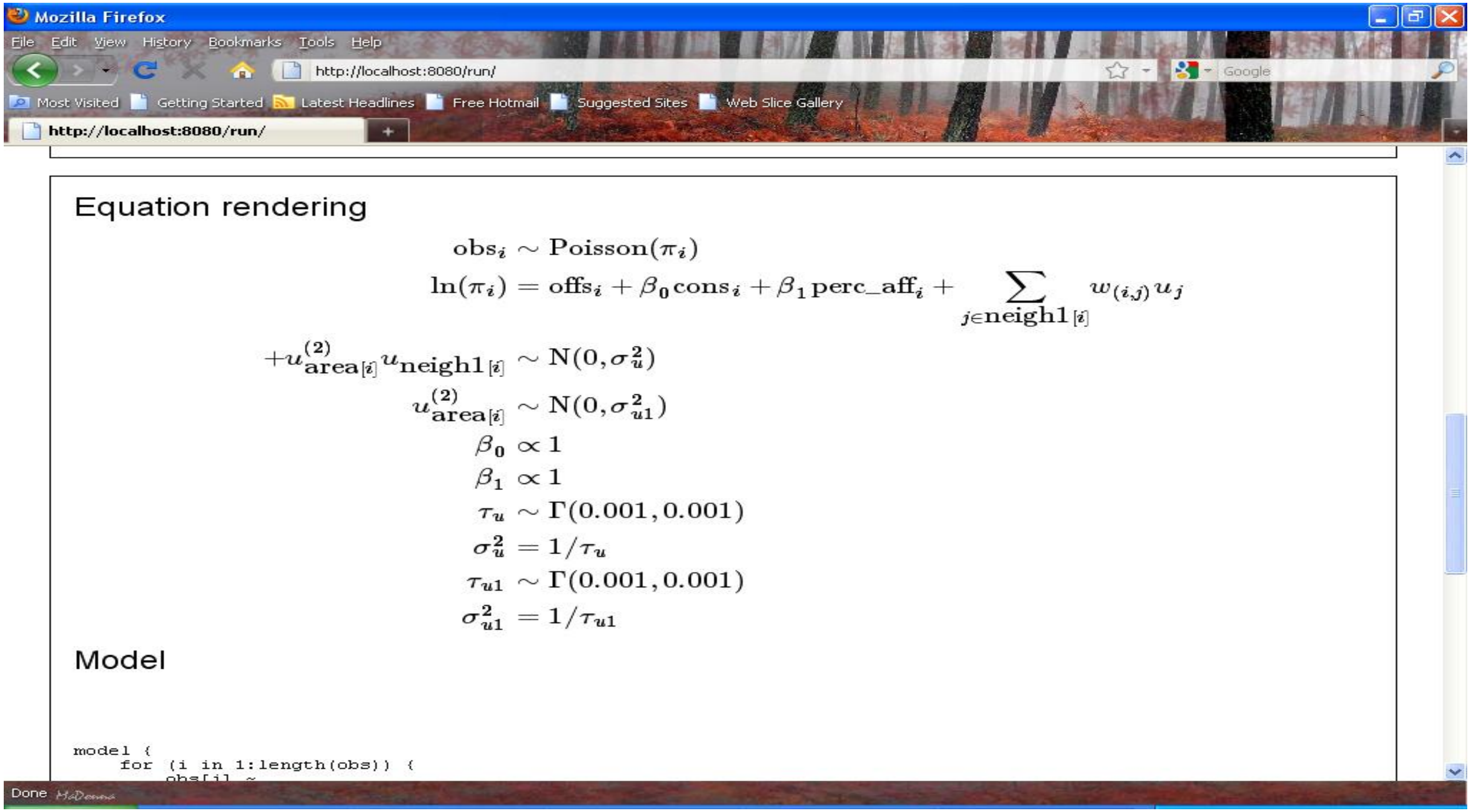
Multiple Membership Models

- Example is Scottish lip cancer data
- Response is Poisson (number of cases)
- Use as offset expected cases based on population size, makeup
- One predictor – percaff – percentage in agriculture, farming, fishing.
- Use the template MultipleMembershipNLev to allow both own random effect and neighbour random effects
- Template will allow fitting in STAT-JR engine, WinBUGS or MLwiN.

Inputs for model



LaTeX for Model



Model Code

Mozilla Firefox

http://localhost:8080/run/

```
model {
    for (i in 1:length(obs)) {
        obs[i] ~
        dpois(p[i])
        ln(p[i]) <-
        offs[i] +
        cons[i] * beta0 + perc_aff[i] * beta1
+ u1[area[i]] * cons[i]
+ weight1[i] * umm[neigh1[i]]
+ weight2[i] * umm[neigh2[i]]
+ weight3[i] * umm[neigh3[i]]
+ weight4[i] * umm[neigh4[i]]
+ weight5[i] * umm[neigh5[i]]
+ weight6[i] * umm[neigh6[i]]
+ weight7[i] * umm[neigh7[i]]
+ weight8[i] * umm[neigh8[i]]
+ weight9[i] * umm[neigh9[i]]
+ weight10[i] * umm[neigh10[i]]
+ weight11[i] * umm[neigh11[i]]
    }

    for (i1 in 1:length(u1)) {
        u1[i1] ~ dnorm(0, tau_u1)
    }

    for (j in 1:length(umm)) {
        umm[j] ~ dnorm(0, tau_u)
    }

    # Priors
    beta0 ~ dflat()
    beta1 ~ dflat()


    tau_u1 ~ dgamma(0.001000, 0.001000)
    sigma2_u1 <- 1 / tau_u1

    tau_u ~ dgamma(0.001000, 0.001000)
    sigma2_u <- 1/tau_u
}
```

Run | Simulate | Selection | Simulate | Specify starting Values | Code | JS | Save imputed every:

University of BRISTOL

The E-STAT project – still to come

We have lots of work to do:

- Parallel processing.
- E-books.
- Optimising code generation.
- Improving algebra system.
- Suites of templates for missing data and social network models.
- Interoperability with SAS and hooking up more templates for other packages.

University of BRISTOL