A Beginner’s Guide to
Stat-JR’s TREE Interface
version 1.0.4

Programming and Documentation by
William J. Browne*, Christopher M.J. Charlton*, Danius T.
Michaelides**, Richard M.A. Parker*, Bruce Cameron*,
Camille Szmaragd*, Huanjia Yang**, Zhengzheng Zhang*,
Harvey Goldstein*, Kelvyn Jones*, George Leckie* and Luc
Moreau**

*Centre for Multilevel Modelling,
University of Bristol, UK.

**Electronics and Computer Science,
University of Southampton, UK.

June 2016
1. About Stat-JR........................................................................................................4
   1.1 Stat-JR: software for scaling statistical heights..................................................4
   1.2 About the Beginner’s guide ..................................................................................6
2  Installing and Starting Stat-JR ...............................................................................6
   2.1 Installing Stat-JR..................................................................................................6
   2.2 The use of third party software and licenses.......................................................6
   2.3 Starting up TREE..............................................................................................6
   2.4 The structure and layout of the TREE interface................................................7
3  Application 1: Analysis of the tutorial dataset using the eStat engine..................19
   3.1 Summarising the dataset and graphs ..................................................................19
   3.2 Single-level Regression.......................................................................................24
   3.3 Multiple chains ..................................................................................................30
   3.4 Adding gender to the model..............................................................................31
   3.5 Including school effects ...................................................................................33
   3.6 Caterpillar plot ..................................................................................................35
4  Interoperability – a brief introduction ..................................................................37
   4.1 So why are we offering interoperability? .........................................................37
   4.2 Regression in eStat revisited ............................................................................38
   4.3 Interoperability with WinBUGS.........................................................................40
   4.4 Interoperability with OpenBUGS .....................................................................43
   4.5 Interoperability with JAGS ...............................................................................45
   4.6 Interoperability with MLwiN...............................................................................47
   4.7 Interoperability with R .....................................................................................50
   4.8 Interoperability with AML ...............................................................................54
5  Application 2: Analysis of the Bangladeshi Fertility Survey dataset ..................55
   5.1 The Bangladeshi Fertility Survey dataset ..........................................................55
   5.2 Modelling the data using logistic regression ....................................................56
   5.3 Multilevel modelling of the data......................................................................60
   5.4 Comparison between software packages .........................................................64
   5.5 Orthogonal parameterisation .........................................................................68
   5.6 Predictions from the model .............................................................................72
6  Miscellaneous other topics e.g. Data Input/Export ................................................75
   6.1 If your dataset is already in .dta format .............................................................75
   6.2 If your dataset is in .txt format .......................................................................75
6.3 Converting your dataset to .dta format.................................................................75
7 References ..................................................................................................................75
8 Appendix: List of Third Party Software that are used by Stat-JR..................................77
9 Index.............................................................................................................................79
Acknowledgements

The Stat-JR software is very much a team effort and is the result of work funded initially under three ESRC grants: the LEMMA 2 and LEMMA 3 programme nodes (Grant: RES-576-25-0003 & Grant:RES-576-25-0032) as part of the National Centre for Research Methods programme, and the e-STAT node (Grant: RES-149-25-1084) as part of the Digital Social Research programme. The work has continued with the ESRC grant ES/K007246/1.

We are therefore grateful to the ESRC for financial support to allow us to produce this software.

All nodes have many staff that, for brevity, we have not included in the list on the cover. We acknowledge therefore the contributions of:

Fiona Steele, Rebecca Pillinger, Paul Clarke, Mark Lyons-Amos, Liz Washbrook, Sophie Pollard, Robert French, Nikki Hicks, Mary Takahama and Hilary Browne from the LEMMA nodes at the Centre for Multilevel Modelling.

David De Roure, Tao Guan, Alex Fraser, Toni Price, Mac McDonald, Ian Plewis, Mark Tranmer, Pierre Walthery, Paul Lambert, Emma Housley, Kristina Lupton and Antonina Timofejeva from the e-STAT node.

A final acknowledgement to Jon Rasbash who was instrumental in the concept and initial work of this project. We miss you and hope that the finished product is worthy of your initials.

WJB June 2016.
1. About Stat-JR

1.1 Stat-JR: software for scaling statistical heights.

The use of statistical modelling by researchers in all disciplines is growing in prominence. There is an increase in the availability and complexity of data sources, and an increase in the sophistication of statistical methods that can be used. For the novice practitioner of statistical modelling it can seem like you are stuck at the bottom of a mountain, and current statistical software allows you to progress slowly up certain specific paths depending on the software used. Our aim in the Stat-JR package is to assist practitioners in making their initial steps up the mountain, but also to cater for more advanced practitioners who have already journeyed high up the path, but want to assist their novice colleagues in making their ascent as well.

One issue with complex statistical modelling is that using the latest techniques can involve having to learn new pieces of software. This is a little like taking a particular path up a mountain with one piece of software, spotting a nearby area of interest on the mountainside (e.g. a different type of statistical model), and then having to descend again and take another path, with another piece of software, all the way up again to eventually get there, when ideally you’d just jump across! In Stat-JR we aim to circumvent this problem via our interoperability features so that the same user interface can sit on top of several software packages thus removing the need to learn multiple packages. To aid understanding, the interface will allow the curious user to look at the syntax files for each package to learn directly how each package fits their specific problem.

To complete the picture, the final group of users to be targeted by Stat-JR are the statistical algorithm writers. These individuals are experts at creating new algorithms for fitting new models, or better algorithms for existing models, and can be viewed as sitting high on the peaks with limited links to the applied researchers who might benefit from their expertise. Stat-JR will build links by incorporating tools to allow this group to connect their algorithmic code to the interface through template-writing, and hence allow it to be exposed to practitioners. They can also share their code with other algorithm developers, and compare their algorithms with other algorithms for the same problem. A template is a pre-specified form that has to be completed for each task: some run models, others plot graphs, or provide summary statistics; we supply a number of commonly-used templates and advanced users can use their own – see the Advanced User’s Guide. It is the use of templates that allows a “building block” or modular approach to analysis and model specification.

At the outset it is worth stressing that there a number of other features of the software that should persuade you to adopt it, in addition to interoperability. The first is flexibility – it is possible to fit a very large and growing number of different types of model. Second, we have paid particular attention to speed of estimation and therefore in comparison tests, we have found that the package compares well with alternatives. Third it is possible to embed the software’s templates inside an e-book which is exceedingly helpful for training and learning, and also for replication. Fourth, it provides a very powerful, yet easy to use environment for accessing state-of-the-art Markov Chain Monte Carlo procedures for calculating model estimates and functions of model estimates, via its eStat engine. The eStat engine is a newly-developed estimation engine with the advantage of being transparent in that all the algebra, and even the program code, is available for inspection. While this is a beginner’s guide, we presume that you have a good understanding of statistical models which can be gained from, for example, the LEMMA online course.

At the outset it is worth stressing that there a number of other features of the software that should persuade you to adopt it, in addition to interoperability. The first is flexibility – it is possible to fit a very large and growing number of different types of model. Second, we have paid particular attention to speed of estimation and therefore in comparison tests, we have found that the package compares well with alternatives. Third it is possible to embed the software’s templates inside an e-book which is exceedingly helpful for training and learning, and also for replication. Fourth, it provides a very powerful, yet easy to use environment for accessing state-of-the-art Markov Chain Monte Carlo procedures for calculating model estimates and functions of model estimates, via its eStat engine. The eStat engine is a newly-developed estimation engine with the advantage of being transparent in that all the algebra, and even the program code, is available for inspection. While this is a beginner’s guide, we presume that you have a good understanding of statistical models which can be gained from, for example, the LEMMA online course.
It also pre-supposes familiarity with MCMC estimation and Bayesian modelling – the early chapters of Browne’s (2016) *MCMC Estimation in MLwiN* (which can be downloaded from http://www.bristol.ac.uk/cmm/software/mlwin/download/manuals.html) provide a practical introduction to this material.

Many of the ideas within the Stat-JR system were the brainchild of Jon Rasbash (hence the “JR” in Stat-JR). Sadly, Jon died suddenly just as we began developing the system, and so we dedicate this software to his memory. We hope that you enjoy using Stat-JR and are inspired to become part of the Stat-JR community: either through the creation of your own templates that can be shared with others, or simply by providing feedback on existing templates.

Happy Modelling,

The Stat-JR team.
1.2 About the Beginner’s guide

We have written several guides to go with the software: this Beginner’s Guide will cover how to start up and run the software, with a particular focus on the TREE (Template Reading and Execution Environment) interface. It will provide some simple examples and is designed for the researcher who wishes to be able to use the software package without worrying too much about how the mathematics behind the modelling works. As such, it does not go into detail on how users can contribute to extending the software themselves: that is covered in the second, Advanced User’s, guide, designed for those who want to understand in greater detail how the system works (a Quick-start guide is also available providing a very brief overview). There is also an E-book User’s guide which deals with the software’s DEEP (Documents with Embedded Execution and Provenance) E-book interface. Finally, a guide has been released alongside Stat-JR 1.0.4 to support the beta version of the workflow system, LEAF (Logging and Execution of Analysis Flows).

As well as these Guides, we also publish support, such as answers to frequently asked questions, on our website (http://www.bristol.ac.uk/cmm/software/statjr), where you can also find our forum in which users can discuss the software.

In this Beginner’s Guide we look at an example application taken from education research, fitting a Normal response model for a continuous outcome. Here our aim is more to illustrate how to use the software than primarily how to do the best analysis of the dataset in question, and we will demonstrate the interoperability features with some of the other software packages that link to Stat-JR as well. We will then look at a second example from demography that illustrates binomial response models for a discrete outcome.

2 Installing and Starting Stat-JR

2.1 Installing Stat-JR

Stat-JR has a dedicated website (http://www.bristol.ac.uk/cmm/software/statjr) from which you can request a copy of the software, and which contains instructions for installation.

2.2 The use of third party software and licenses

Stat-JR is written primarily in the Python (see https://www.python.org/) package but also makes use of many other third party software packages. We are grateful to the developers of these programs for allowing us to use their products within our package. When you have installed Stat-JR you will find a directory entitled licences in which you can find subdirectories for each package detailing the licensing agreement for each. The list of software packages that we are using can be found in the Appendix to this document.

2.3 Starting up TREE

Stat-JR’s interface is viewed and operated via a web browser, but it is started by running an executable file.

To start Stat-JR select the Stat-JR TREE link from the Centre for Multilevel Modelling suite on the start up menu. This action opens a command prompt window in the background to which commands are printed out. This window is useful for viewing what the system is doing: for example, on the machine on which we have run TREE, you can see commands like the following:
The last line quoted here (although more lines will appear beneath it on start-up) indicates that Stat-JR is locating the default web browser on your machine; once it has done so it will open that web browser and display TREE’s welcome page. The lines such as “WARNING:root:Failed to load package GenStat model (GenStat not found)” are not necessarily problematic but are warning you that the Genstat statistical package – one of the third-party statistical packages with which Stat-JR can interoperate – has not be found (where Stat-JR expects to find it if it is installed) on your particular machine.

Stat-JR works best with either Chrome or Firefox, so if the default browser on your machine is Internet Explorer it is best to open a different browser and copy the html path to it; this will be something like localhost:52228 (although the number will likely differ each time you run Stat-JR). You can change your default browser via Settings in the Chrome menu, or via Options > General in the Firefox menu (both menus are found in top-right of their respective browser windows).

2.4 The structure and layout of the TREE interface
Stat-JR can be thought of as a system that manages the use of a set of templates written either by the developers, and supplied with the software, or by users themselves. Each template will perform a specific function: for example, fitting a specific family of models, summarising a dataset, plotting a graph, and so on. The Stat-JR system therefore allows the user to select and use specific templates with their datasets, and to capture and display the outputs that result.

When operating Stat-JR through TREE, you generally proceed through the following five stages:
Stage 1. Firstly, choose the dataset you want to analyse / plot / summarise / etc., and the template you want to use to do so. Each template contains commands to perform certain functions: some run models, others plot graphs, or provide summary statistics, and so on...

Stage 2. You will be asked for further template-specific input: e.g. which variables from your dataset you would like to include in your model / which variables you would like to plot / summarise / etc.

Stage 3. Once you’ve answered all the input queries, Stat-JR generates all the commands, scripts, macros, equations, and instructions necessary to perform, or describe, the function you’ve requested. You can view these within TREE, and can download them too...

Stage 4. Stat-JR then runs these commands / scripts / macros, employing externally-authored software (e.g. R, MLwiN, WinBUGS, SPSS, Stata, etc.), or in-house software (such as the eStat engine), as appropriate.

Stage 5. Finally, the results are returned; depending on the template these can include model estimates, graphs, summary tables, and so on. Again, these can be viewed within TREE, and are also downloadable. The output may also include datasets (e.g. MCMC chains), which you can then feed back into the system by matching them up with a template back in Stage 1.
Returning to our start-up of the software, when the line http://0.0.0.0:50215/ appears, and after refreshing the web browser, the browser window should appear as follows:

This is the start screen for the TREE interface to Stat-JR, and contains information on funders, authors, and a link to the Stat-JR website which contains further guidance, such as answers to frequently asked questions, and a user forum.

Pressing Begin returns the following screen:

At the top you’ll see a black title bar. From left to right, this contains:

- a link (Stat-JR:TREE) back to the welcome page;
- an option (Start again) to clear all inputs the user has chosen for the current template;
- a Dataset menu allowing the user to Choose, Drop (from temporary memory cache), View the dataset (as well as summary statistics, the option to add / delete variables, edit data values, edit descriptive labels, duplicate the current dataset, etc.), return a List of datasets, and Upload / Download (see Section 6) datasets. For example, selecting Dataset > Choose returns a scrollable list of all the datasets that the system is aware of: i.e. those which are in the user’s data folder (by default under Users\user_name\statjr\datasets) and those globally available (in the datasets subdirectory of this installation of Stat-JR). This pane can
be used to change the selected dataset via the **Use** button; for inputting your own data set you can use the **Upload** button;

- the name of the currently-selected dataset (in the grey box) – if you hover your cursor over this name, it returns a textual description of the dataset;

- a **Template** menu allowing the user to **Choose, List** (described below), **Upload** individual templates not already uploaded in the current session or **Set Inputs** for the current template (as an input string, rather than pointing-and-clicking through the inputs; this option also allows you to retrieve and re-use input values from previous template executions). If you select **Template > Choose**, a box appears which contains a scrollable list of all the templates that the system is aware of: i.e. those which are in the user’s template folder (by default under `Users\user_name\statjr\templates`) and those globally available (in the `templates` subdirectory of this installation of Stat-JR). This can be used to change the selected template via the **Use** button. Each template has defined ‘tags’ which are terms to describe what it does: these appear as blue phrases in the ‘cloud’ above the list of templates, whereas the estimation engines supported by each template appear in the cloud in red. When you select a template, its name and description appear to the right of the list. Clicking on the symbol that looks like a baggage label returns the tags for that template, whereas clicking on the ‘cog’ symbol returns a list of engines that particular template supports;

- the name of the currently-selected template (in the grey box) – again, if you hover your cursor over this name, it also returns a description of the template;

- a progress gauge indicating whether Stat-JR is “Idle” (before it has run anything), “Ready” (once it has run something, and is ready for further user input), “Initialising”, “Working” or whether it has encountered an “Error”;

- a link to a page containing options to change a variety of **Settings**. This allows the user to specify the path to their datasets, templates, workflows, and the eBook information store (where eBooks loaded into Stat-JR’s DEEP interface are saved, although it is unlikely users would wish to edit this folder directly: amendments to it can be made via the DEEP interface itself). These paths refer to **user-specific folders** (by default under `Users\user_name\statjr\`) whilst those datasets, templates and workflows which are **globally available** (to all users) are saved in their respective subdirectories in this installation of Stat-JR. The **Settings** window also displays a number of settings that the program uses with each possible software package: some of these are relatively straightforward, such as where the executables for each package are found, and some are relatively advanced, such as for the eStat engine, optimisation, starting values and standalone code options;

- a **Debug** button; this produces a drop-down list from which one can choose to **Reload templates**, **Reload datasets** or **Reload packages**, allowing users upload changes to files they make outside the **TREE** interface, without having to start-up Stat-JR again. For example, a user could paste a new dataset into the (global or user-specific) datasets directory, or modify a template in the (global or user-specific) templates directory, and reload them so that they appear in their lists in the browser window. In addition, if the user changes the path to a
third-party software package (via Settings), then Reload packages will implement this change in the current session.

We will now look at The View dataset window:

Select Dataset > Choose from the menu in the black title bar.

Scroll down the dataset list, towards the bottom, and click on rats; its name and description will appear to the right of the list.

Click on the Use button, and the name of the current dataset (in the grey box in the black title bar at the top) should have changed accordingly.

Select Dataset > View; this will open a new tab in your browser: if you click on this you will be able to see the dataset we have just selected, as follows:

The rats dataset is a small, longitudinal animal growth dataset which contains the weights of 30 laboratory rats on 5 weekly occasions from 8 days of age (see Gelfand et al (1990) for more details). The five measurements are labelled y8, y15, y22, y29 and y36, respectively, and the dataset also contains a constant column – a vector of ones, named cons, and a rat identifier column, rat. Initially, we are going to perform a regression analysis of the initial weight (y8) on the final weight (y36),
including an intercept (\( \text{cons} \)). The tabs above the dataset allow the user to quickly add a new variable or delete an existing variable from the dataset. We can also view a summary of the dataset:

To view a summary of the dataset, click on the Summary tab above the data and the screen will look as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Count</th>
<th>Missing</th>
<th>Min</th>
<th>Max</th>
<th>Mean</th>
<th>Std</th>
<th>Description</th>
<th>Value Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>y8</td>
<td>30</td>
<td>0</td>
<td>132</td>
<td>177</td>
<td>152.16666667</td>
<td>10.975983884</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y15</td>
<td>30</td>
<td>0</td>
<td>180</td>
<td>254</td>
<td>201.76666667</td>
<td>12.459374427</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y22</td>
<td>30</td>
<td>0</td>
<td>219</td>
<td>295</td>
<td>245.03333333</td>
<td>15.117687796</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y29</td>
<td>30</td>
<td>0</td>
<td>258</td>
<td>300</td>
<td>289.5</td>
<td>18.825639275</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y36</td>
<td>30</td>
<td>0</td>
<td>291</td>
<td>376</td>
<td>324.8</td>
<td>19.118234015</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cons</td>
<td>30</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
<td>0.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rwi</td>
<td>30</td>
<td>0</td>
<td>1</td>
<td>30</td>
<td>15.5</td>
<td>8.6554414684</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Here we get a very short summary of the dataset, giving, for each variable, the minimum value, maximum value, mean and standard deviation. If the dataset has had descriptions added or has categorical variables then they will appear in the last two columns. More extensive summaries are available by using specific templates to summarise datasets, as we will see later.

Let’s now look at the Template menu:

Back on the main page, if you click on Template > List the following screen will appear in a new tab:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Contains</th>
<th>Tags</th>
<th>Engines</th>
</tr>
</thead>
<tbody>
<tr>
<td>1levelBlock</td>
<td>Fits 1-level Normal models, with an option of a multivariate step for the fixed effects.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>[\text{MCMC}, \text{MLwiN}, \text{R} &amp; \text{JAGS}, \text{Bugs}]</td>
<td>[\text{Model}, \text{1-level}, \text{Normal}]</td>
<td>[\text{MCMC}, \text{MLwiN}, \text{R} &amp; \text{JAGS}, \text{Bugs}]</td>
</tr>
<tr>
<td>1levelBlockc</td>
<td>Fits 1-level Normal models, with a multivariate step for the fixed effects.</td>
<td>[\text{inputs}, \text{_module_}, \text{_code_}, \text{_customcode_}, \text{_data_}, \text{_version_}, \text{_tags_}, \text{_engines_}, \text{_description_}, \text{_customcode_}, \text{_data_}, \text{_version_}, \text{_tags_}, \text{_engines_}, \text{_description_}]</td>
<td>[\text{Model}, \text{1-level}, \text{Normal}]</td>
<td>[\text{MCMC}, \text{MLwiN}, \text{R} &amp; \text{JAGS}, \text{Bugs}]</td>
</tr>
<tr>
<td>1levelBlockcomp</td>
<td>Fits 1-level Normal models, with a multivariate step for the fixed effects.</td>
<td>[\text{inputs}, \text{_module_}, \text{_code_}, \text{_customcode_}, \text{_data_}, \text{_version_}, \text{_tags_}, \text{_engines_}, \text{_description_}, \text{_customcode_}, \text{_data_}, \text{_version_}, \text{_tags_}, \text{_engines_}, \text{_description_}]</td>
<td>[\text{Model}, \text{1-level}, \text{Normal}]</td>
<td>[\text{MCMC}, \text{MLwiN}, \text{R} &amp; \text{JAGS}, \text{Bugs}]</td>
</tr>
</tbody>
</table>

This rather busy screen (we don’t reproduce it all here, due to its length) contains, in the two columns on the left, a tabular list of all the templates that are available with a short description of what each template does. The next column is of more interest to advanced users, and contains a list of functions in the template code, whilst the final two columns contain the tags that identify the template type, and the engines that are supported by the template.
We will next demonstrate running a template, using the default `Regression1` template that fits a 1-level Normal response regression model: this is appropriate as the response, the weights of the rats, is a continuous measure.

Return to the main menu screen, which should look as follows:

In the middle of the screen you can see the inputs required for this template (these are template-specific, and so will likely change when you use a different template). Since some inputs are conditional (i.e. are only required when earlier inputs take specific values), the opportunity to specify inputs proceeds through sequential steps. Here we see the two initially-required inputs for the `Regression1` template are the `Response` variable and `Explanatory variables`. Since this template only allows for one response variable to be specified, a pull-down list is displayed, but since it allows for several explanatory variables to be specified, a multiple selection list is displayed for that input value. In the case of the latter, variables are selected by clicking on their name in the left-hand list; to de-select them, click on their name in the right-hand list.

The `Start again` link (in the top black bar) will clear any inputs the user has already selected and return you to the first template input screen (i.e. the current screen, in this case), whilst the `Next` button will allow the user to move on and specify further inputs once those on the current screen have all been chosen.

Use the input controls and the `Next` button(s) to fill in the screen as follows:
So here we have entered **Response**: y36; **Explanatory variables**: cons,y8; **Number of chains**: 1; **Random Seed**: 1; **Length of burnin**: 1000; **Number of iterations**: 5000; **Thinning**: 1; **Use default algorithm settings**: Yes; **Generate prediction dataset**: No; **Use default starting values**: Yes; **Name of output results**: out.

Note that an option to **remove** appears next to each input previously submitted; this will remove the current input, but keep the other inputs you have specified (as far as it can; if they are conditional on the input you have removed, then they will be, out of necessity, removed too).

So, here we are performing a regression of the initial weight (y8) on the final weight (y36), including an intercept (cons). The other inputs refer to the Monte Carlo Markov chain (MCMC) estimation procedures in Stat-JR. MCMC estimation methods are simulation-based, and so require certain parameters to be set. The methods involve taking a series of random (dependent) draws from the posterior distribution of the model parameters in order to summarise each parameter. The inputs required here are as follows:

- **Number of chains**: this is the number of starting points from which we will take random draws;
- **Random Seed**: the value from which random numbers are initially drawn. This allows repeatability, as a run using the same starting values and random seed will give the same answers. When multiple chains are used this seed is generally multiplied by the chain number to give a unique seed for each chain;
- **Length of burnin**: the initial length of the chain (i.e. the number of iterations at the start) which are excluded from the parameter summaries (the rationale for this is explained a little further in the example, below, with the tutorial dataset);
- **Number of iterations**: the length of chain following the burnin, from which the parameter summaries are drawn;
- **Thinning**: this determines how often the values are stored: i.e. store every nth iteration.

By answering Yes to the question **Use default algorithm settings**, we have used defaults for other settings for which we will therefore not be prompted to complete. By answering No to **generate prediction dataset** we have chosen not to generate a dataset of predictions from our model. By answering Yes to **Use default starting values** we have chosen not to start the chain at values of our choosing, instead accepting Stat-JR’s defaults. We will discuss MCMC estimation in slightly more detail in the applications in the next section. The final input we’re asked for is the **Name of output results**: this is the name (here we’ve chosen *out*) given to any dataset of parameter chains that results from running the template.

You will notice, towards the bottom of the window, a box with a rather long text string labelled **Current input string** above it and another labelled as **Command** below it. The input string allows the user to specify all the inputs directly, via the **Set Inputs** option in the **Template** pull down list, without having to point-and-click through the list as we have done. If you click on **Template > Set Inputs** you will see this input string reproduced in the **Input string** box; clicking on the **Use** button populates the inputs with these values, which obviously will have no effect here, but it would if you first changed a value, or indeed used the inputs from a previously-run template execution, as selected from the **History** box above. The input string needs to be in a certain format, as illustrated below, and note that the inputs will often not have exactly the same name as appears in the prompts you answered earlier (e.g. **Use default starting values**: Yes corresponds to ‘defaultsv’: ‘Yes’). In this guide we will reproduce the input string for each example (and also specify the dataset and template, in case it is unclear from the screenshot, etc.) so you can simply copy and paste it in if you need to (although note that the appropriate template will need to be pre-selected – it won’t change the template for you). So the input string for the inputs we have just specified is as follows:

**Dataset**: rats; **Template**: Regression1; **Input string**: `{"burnin": '1000', 'defaultsv': 'Yes', 'outdata': 'out', 'thinning': '1', 'nchains': '1', 'defaultalg': 'Yes', 'iterations': '5000', 'y': 'y36', 'x': 'cons,y8', 'seed': '1', 'makepred': 'No'}`
Returning to the main window, the second text string (labelled **Command**) can be used by the command-driven version of Stat-JR to perform the same operations.

Clicking on the **Next** button will now pre-process the template inputs; this will result in the following new pane at the bottom of the window:

The object currently specified in the pull-down list (**equation.txt** is selected by default here) appears in the pane below it. These objects are any outputs constructed by Stat-JR before and during the execution of the template, so here we see a nice mathematical description of the model. If we now select the object **model.txt** from the list we see a description of the regression model that we wish to fit in the language that is used by the eStat engine:

At this point we haven’t actually run the template, and so the objects that can be selected from the pull-down list are those present pre-model run, and include computer code to actually fit the model.

Click the **Run** button to run the template.

Once the progress gauge, towards the right of the black title bar, has changed from “Working” (blue) to “Ready” (green), select **ModelResults** from the pull-down list.
The screen will then look as follows:

Here we see parameter estimates, along with standard deviations (SDs) as a measure of precision for each parameter. We will explain these further in the next section. At the top of the screen shot above (which is in fact the middle of the full window, vertically-speaking) we now have a few additional buttons. The **Extra Iterations** box, along with the **More** button, will allow us to run for longer (i.e. for a number of iterations additional to those we have already run for). The **Download** button will produce a zipped file that contains a folder with files for many of the objects contained in the two pull-down lists whilst the **Add to ebook** button can be used if one wants to construct an ebook to be used with the **DEEP** eBook interface into Stat-JR. Finally, the **Make workflow** button relates to functionality supporting the beta version of the workflow interface, **LEAF (Logging and Execution of Analysis Flows)**, released with Stat-JR v.1.0.4.

You’ll recall that we earlier named the output results **out**, so if we choose this from the pull-down list just above the output pane, we’ll be able to view it, as follows:

<table>
<thead>
<tr>
<th>parameter</th>
<th>mean</th>
<th>sd</th>
<th>ESS</th>
<th>variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>tau</td>
<td>0.00418080767558</td>
<td>0.0011904973038</td>
<td>4327</td>
<td></td>
</tr>
<tr>
<td>beta_0</td>
<td>161.880410595</td>
<td>34.51866849</td>
<td>23</td>
<td>cons</td>
</tr>
<tr>
<td>beta_1</td>
<td>1.02425050556</td>
<td>0.25651981782</td>
<td>23</td>
<td>yB</td>
</tr>
<tr>
<td>sigma2</td>
<td>257.210134104</td>
<td>73.2900156192</td>
<td>4429</td>
<td></td>
</tr>
<tr>
<td>sigma</td>
<td>15.887481078</td>
<td>2.1389042306</td>
<td>4397</td>
<td></td>
</tr>
<tr>
<td>deviance</td>
<td>260.68090715</td>
<td>2.2300617033</td>
<td>403</td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dbar</td>
<td>250.08939715</td>
</tr>
<tr>
<td>D(thetabar)</td>
<td>240.013138421</td>
</tr>
<tr>
<td>pD</td>
<td>2.0708120348</td>
</tr>
<tr>
<td>DIC</td>
<td>263.06674108</td>
</tr>
</tbody>
</table>
Here we see columns containing the chains of values for each parameter in the model. As well as being able to view this file here, it is also a dataset (stored in temporary memory) and so will appear in the dataset list (at least for the duration of our current session using the software) accessible via the Dataset menu in the top title bar (emboldened to indicate that it has been created in this run of the software). This means that we can string template executions together, as we can select out as a dataset and perform operations on it using another template.

This ends our whistle-stop tour of many of the windows in Stat-JR. We will next look at a practical application.
3 Application 1: Analysis of the tutorial dataset using the eStat engine

3.1 Summarising the dataset and graphs
In this section we will look at performing some analysis of an example dataset from education. The dataset in question is known as the tutorial dataset, and is used as an example in the MLwiN software manuals (see, for example, Browne 2012). In fact, much of the material here owes a lot to Browne (2012), which employs similar analysis but using MLwiN.

Let us start by looking at the tutorial dataset.

Select tutorial via Dataset > Choose (see the title bar), then click Use.

If you then select Dataset > View, and click on the Summary tab the following should appear in a new tab in the browser window containing summary information, as follows:

The tutorial dataset contains data on exam scores of 4059 secondary school children from 65 schools at age 16. These exam scores have been normalised to have a mean of zero and a standard deviation of one and are named normexam. There are several predictor variables, including a (standardised) reading test (standlrt) taken at age 11, the pupils gender (girl), and the school’s gender (schgend) which takes values 1 for mixed, 2 for boys and 3 for girls. Each variable is described in the Description column and if you hover over any of value label names that appear in the Value Labels column, the category labels will be displayed.

We can explore the dataset in more detail, prior to fitting any models, by using the many data manipulation templates available in Stat-JR. We will first look at some plots of the data:

Select Template > Choose and then select Histogram from the template list that appears and click Use.
Fill in the inputs as shown below and click **Next** and then **Run** and select **histogram.svg** from the output list.

**Dataset:** tutorial; **Template:** Histogram; **Input string:** {'vals': 'normexam', 'bins': '20'}

Here you will see, in the output pane, a histogram plot that shows that the response variable we will model, normexam, appears Normally-distributed.

Select **Template > Choose** and this time select **XYPlot** from the template list, then click **Use**.

Fill in the inputs as shown below and click **Next** and then **Run** and select **graphxy.svg** from the list.
Here we see that there appears to be a positive relationship between normexam and standlrt, with pupils that have higher intake scores performing better, on average, at age 16.

We can display the graph in a separate tab in the browser window by clicking on the Popout button next to the pull down list:
For the sake of brevity, for the remainder of this documentation we will assume you now know how to change template/dataset, and also how to display output in separate tabs, so we’ll refrain from repeating this information in detail again.

Next, we might like to examine how correlated the two variables, \textit{normexam} and \textit{standlrt}, actually are:

Select \textbf{Average and Correlation} as the template, and complete the inputs as follows before clicking on \textbf{Next} and \textbf{Run} and selecting table from the outputs:
Here we see that the correlation is 0.592, so fairly strong and positive. We might also like to look at how exam score varies by gender:

Select Tabulate as the template, and complete the inputs as follows, before clicking on Next and Run and selecting table from the output list:

**Dataset:** tutorial; **Template:** Tabulate; **Input string:** {'subset': 'No', 'varcol': 'normexam', 'rows': 'cons', 'cols': 'girl', 'op': 'means'}

We have to enter variables for column values and row values, and so here we have specified column values as girl (taking value 1 for girls and 0 for boys) and row values as cons (which is a constant), and then we get 2 columns in the output labelled 0 and 1 for boys and girls, respectively. Looking at the means, it appears that girls do slightly better than boys, and looking at the standard deviations
(sds) they are slightly less variable than boys in their scores. Let us now consider performing some statistical modelling on the dataset.

### 3.2 Single-level Regression

As in the last chapter, with the `rats` dataset, we will start by fitting a simple linear regression model to the `tutorial` dataset. Here we will regress `normexam` on `standlrt` by using a modelling template.

**Select Regression1 as the template and fill it in as follows:**

**Dataset:** `tutorial`; **Template:** `Regression1`; **Input string:**

```python
{'burnin': '1000', 'defaultsv': 'Yes', 'outdata': 'out', 'thinning': '1', 'nchains': '1', 'defaultalg': 'Yes', 'iterations': '5000', 'y': 'normexam', 'x': 'cons,standlrt', 'seed': '1', 'makepred': 'No'}
```

Here we are fitting a linear regression, and so have `standlrt` as an explanatory variable, but also `cons` (which is a column of ones) as we would like to include an intercept as well. For now we have set-up the MCMC estimation options as we did for the `rats` dataset, and we will overwrite the output file `out`.

Clicking on the **Next** button will populate a pull-down list of objects created by Stat-JR at the bottom of the screen and by default we see the object `equation.tex`:
In the pane we find a mathematical representation of the chosen model. Note that the file is a LaTeX file that is being rendered in the browser by a piece of software called MathJaX (v2.3, 2013), so if you are a LaTeX-user you can copy this file straight into a document. If we instead choose model.txt from the list we see the following:

Here we see the text file that represents the model we wish to fit in the language that the algebra system used by the built-in eStat engine requires. The Regression1 template only uses the eStat MCMC-based estimation engine, so as you can see in the mathematical formulae in equation.tex we are fitting a Bayesian version of a linear regression, and the last four lines of the output are prior distributions for the unknown parameters, $\beta_0$, $\beta_1$ and the precision $\tau$ (where $\tau=1/\sigma^2$).

Whilst we will keep our description of Bayesian statistics and MCMC estimation to a minimum, and recommend Chapter 1 of Browne (2012) for more details, in brief we are interested in the joint posterior distribution of all unknown parameters given the data (and the prior distributions specified). In practice, in complex models, this distribution has many dimensions (in our simple regression we have 3 dimensions) and is hard to evaluate analytically. Instead, MCMC algorithms work by simulating random draws from a series of conditional posterior distributions (which can be evaluated). It can then be shown (by some mathematics) that after a period of time (required for the simulations to move from their possibly arbitrary starting point) that the draws will be a dependent
sample from the joint posterior distribution of interest. It is common, therefore, to throw away the first \( n \) draws which are deemed a burn-in period.

For the simple linear regression, it is a mathematical exercise to show that the conditional posterior distributions have standard forms and are Normal (for the fixed effect) and Gamma (for the precision = 1/variance). The eStat engine has a built in algebra system which takes the text file (model.txt) in the left-hand pane and returns the conditional posterior distributions; you can view these as follows:

Select `algorithm.txt` from the list and click on the Popout button and the algebra steps will appear in a new tab as follows:

The eStat engine then takes these posterior distributions and wraps them up into computer code (C++) which it will compile and run for the model. By default this will be several pieces of code that are linked together by Stat-JR, although the Settings screen (accessible via a link towards the top of the main menu screen, as we saw earlier) has an option to output completely standalone code that can be taken away and run separately from the Stat-JR system; this is, however, a topic for more advanced users.

Returning to the tab, in the browser window, containing the model template, click on the Run button and wait for the model to run.

Then select ModelResults from the pull down list and pop it out into a separate tab.
Here the model results can be split into two parts:

The first part of the results (under the heading ‘Parameters’) contains the actual parameter estimates. Here, for each parameter, we get 3 numbers: a posterior mean estimate (mean), a posterior standard deviation (sd), and an effective sample size (ESS).

Here we see that beta_0 has a mean estimate of approximately 0, which we would expect as both the response and predictor have been normalised, or standardised. The slope beta_1 has mean 0.595 with standard deviation 0.013, and is highly significant, as its mean estimate is many times its standard deviation (a Bayesian equivalent of a standard error). The value 0.595 represents the average increase in the normexam score for a 1-point (1 sd, due to standardising) increase in standlrt. The residual variance, sigma2, has value 0.649 showing that, as the initial response variance was 1.0, standlrt has explained 35.1% of the variability.

The ESS is a diagnostic which reflects the simulation-based (stochastic) nature of the MCMC estimation procedure: we have based our results on the 5,000 iterations post burn-in, but we know that the method produces dependent samples, and so the ESS gives an equivalent number of independent samples for the parameters involved; in effect a measure of the information content of the chain. In this case, all parameters have ESS of > 4000, and so the chains are almost independent.

The second part (under the heading ‘Model’) refers to the model fit for this particular model and the DIC diagnostic (Spiegelhalter et al. 2002). The DIC diagnostic is an information criterion which is a measure of how good a specific model is, consisting of a combination of how well the model fits the data (here defined by the model deviance) and how complex the model is (here defined by pD: the effective number of parameters). Basically the better fitting the model is, the better the model is, but it has to be penalised by how complex it is. The DIC statistic is defined as the deviance of the mean + 2pD. In this example the deviance at the mean (D(thetabar)) is 9760.5 and pD is ~3 (reflecting the three parameters of the model that are being estimated) and so we have a DIC value of 9766.4. This number is not particularly interesting in isolation but it is when we compare values for several models.
We can also get more information from the diagnostic plots that are available in the list of objects.

Return to the model run tab in the browser window, and select beta_1.svg from the pull-down list above the output pane and pop it out into a separate tab.

This “sixway” plot gives several graphs that are constructed from the chain of 5,000 values produced for beta_1. The top-left graph shows the values plotted against iteration number, and is useful to confirm that the chain is ‘mixing well’, meaning that it visits most of the posterior distribution in few iterations. The top-right graph contains a kernel density plot which is like a smoothed histogram and represents the posterior distribution for this parameter. Here the shape is symmetric and looks like a Normal distribution which we expect given theory for fixed effects in a normal model.

The two graphs in the middle row are time series plots known as the autocorrelation (ACF) and partial autocorrelation (PACF) functions. The ACF indicates the level of correlation within the chain; this is calculated by moving the chain by a number of iterations (called the lag) and looking at the correlation between this shifted chain and the original. In this case, the autocorrelation is very small for all lags. The PACF picks up the degree of auto-regression in the chain. By definition a Markov chain should act like an autoregressive process of order 1, as the Markov definition is that the future state of the chain is independent of all the past states of the chain given the current value. If, for example, in reality the chain had additional dependence on the past 2 values, then we would see a significant PACF at lag 2. In this case all PACF values are negligible. All of this suggests that we have good mixing and it would be appropriate to proceed to the interpretation of the parameters.
The bottom-left plot is the estimated Monte Carlo standard error (MCSE) plot for the posterior estimate of the mean. As MCMC is a simulation-based approach this induces (Monte Carlo) uncertainty due to the random numbers it uses. This uncertainty reduces with more iterations, and is measured by the MCSE, and so this graph details how long the chain needs to be run to achieve a specific MCSE. The sixth (bottom-right) plot is a multiple chains diagnostic and doesn’t make much sense when we have run only one chain, and we will therefore consider running multiple chains in the next section.

We can also get some other diagnostics and summary statistics for the model as follows:

Click on the **Template** pull-down list at the top of the screen and select **Choose** and **SummaryStats** as the template.

Next click on the **Dataset** pull down list and select **Choose** and **out** as the dataset.

Run the **SummaryStats** template and select the inputs as follows before clicking on **Run**:

**Dataset: out; Template: SummaryStats; Input string: `{cols: 'beta_0,beta_1,sigma2'}`**

Press **Run**, and then select **table** from the drop-down list of outputs, and display it in a separate tab:

Here we see a more extensive summary of the three parameters of interest. This summary table includes various quantiles of the distribution which are calculated by sorting the chain and picking the values that lie x% into the sorted chain (where x is 2.5, 5, 50 etc.). These allow for accurate
interval estimates that do not rely on a Normal distribution assumption. The inter-quartile range (IQR) is similarly calculated by picking the values that lie 25% and 75% through the sorted list and calculating the distance between them.

The final statistic is an MCMC diagnostic designed to suggest a length of chain to be run. The Brooks-Draper diagnostic is based on measuring the mean estimate to a particular accuracy (number of significant figures set to 2 by default). For example, it states that to quote $\sigma^2$ as 0.65 with some desired accuracy only requires 32 iterations. The anomaly here is $\beta_0$, however, since the true value is 0 we have difficulty quoting such a value to 2 significant figures!

3.3 Multiple chains

MCMC methods are more complicated to deal with than classical methods as we have to specify many estimation parameters, including how long to run the MCMC chains for. The idea of running chains for a longer period is to counteract the fact that the chains are serially-correlated, and therefore are not independent samples from the distribution. Another issue that might cause problems is that the posterior distribution of interest may have several possible maxima (i.e. may be multimodal). This is not usually an issue in the models we cover in this book, but it is still a good idea to start off the estimation procedure from several places, or with several runs with different random number seeds, to confirm we get the same answers.

From the top bar change Template and Dataset using the respective pull down lists and Choose so you have Regression1 as the template and tutorial as the dataset.

This time fill in the screen as follows:

**Dataset: tutorial; Template: Regression1; Input string:**

```json
{'burnin': '500', 'defaultsv': 'Yes', 'outdata': 'out3', 'thinning': '1', 'nchains': '3', 'defaultalg': 'Yes', 'iterations': '2000', 'y': 'normexam', 'x': 'cons,standlrt', 'seed': '1', 'makepred': 'No'}
```

Click on the **Next** and **Run** buttons.
When the model has run select `beta_1.svg` from the outputs list and pop it out to view it in a new tab.

Here we see the three chains superimposed on each other in the top-left pane – note the chain looks primarily red simply because this chain (chain 3) has been plotted on top of the other two, and due to good mixing obscures them. Each chain has its own kernel plot in the top-right pane and this also suggests that, by the similarity of shape and position, the chains are mixing well.

We have previously described what all the graphs here mean in Section 3.2, apart from the Brooks-Gelman-Rubin diagnostic plot (BGRD; Brooks and Gelman, 1998) in the bottom-right corner. This plot looks at mixing across the chains: the green and blue lines measure variability between and within the chains, and the red is their ratio. For good convergence this red line should be close to 1.0, and here the values get close to 1.0 fairly quickly. We can have a lot of faith in the estimates of our model.

### 3.4 Adding gender to the model

We have so far been more focused on understanding the MCMC methods but now we will return to modelling. We next wish to look at whether gender has an additional effect on `normexam` on top of that we have observed for intake score (`standlrt`).

To do this, click on the remove link next to explanatory variables in the browser window, and fill-in the template as follows:
**Dataset: tutorial; Template: Regression1; Input string:**

{\'burnin\': '500', \'defaultsv\': 'Yes', \'outdata\': 'outgend', \'thinning\': '1', \'nchains\': '3', \'defaultalg\': 'Yes', \'iterations\': '2000', \'y\': 'normexam', \'x\': \'cons,standlrt,girl\', \'seed\': '1', \'makepred\': 'No'}

Click on **Next** and then **Run** to run the model.

After the model finishes running select **ModelResults** from the drop-down list of outputs, and display in a new tab.

This new model has one additional fixed effect parameter (beta_2) associated with gender, and we see it has a positive effect (0.170) which appears highly-significant (at least twice its standard deviation, which is 0.025). Note that in our earlier tabulation we saw that the difference in gender means was 0.093-(-0.140) = 0.233 and so the effect here is somewhat smaller, probably due to correlation between gender and intake score.
Looking at the DIC diagnostic to assess whether this model is better we see this has dropped from 9766.4 to 9724.9, which is a big drop, and so the model with gender is indeed much better.

Finally we see that the ESS for two of the parameters is lower (beta_0 and beta_2), at around 1600, so the model doesn’t mix quite as well; however, these ESS are still large enough not to require further iterations. Here is the graph for beta_2.svg, displayed in a new tab:

![Graph showing beta_2 distribution](beta_2.svg)

We see reasonable mixing, and can clearly see the significance of the effect as well (as the kernel density plot in the top-right corner indicates that 0 is nowhere near the posterior distribution). From a modelling perspective we have thus far ignored the fact that our data is a two-stage sample and that we should account for the clustering of the pupils within secondary schools. To do this we need to fit a 2-level model, and use a different template.

### 3.5 Including school effects

Stat-JR contains many different model-fitting templates some of which can fit whole families of models and some of which can fit just one or two specific models. We have thus far looked at the rather restrictive Regression1 template that only fits single level normal response models. To include school effects we will now look at the 2LevelMod template, which not only includes a set of random effects but also supports different response types and estimation engines, features that we will look at later.

On the Template pull-down list at the top of the screen select Choose and select 2LevelMod as the template and stick with tutorial for the dataset.

Set-up the inputs as shown below:

**Dataset:** tutorial; **Template:** 2LevelMod; **Input string:** `{'Engine': 'eStat', 'L2ID': 'school', 'burnin': '500', 'D': 'Normal', 'outdata': 'out2level', 'storeresid': 'Yes', 'thinning': '1', 'nchains': '3', 'defaultalg': 'Yes',`
Press **Next** and then **Run** to fit the model. Note that running may take a while as we are storing all 65 school effects and so for each one the software needs to construct diagnostic plots.

When the model finishes select **ModelResults**, from the output list, and show the results in a separate tab.
Here if you scroll down we see that the DIC value for the two-level model is 9245, compared with 9725 for the simpler model, showing that it is important to account for the two levels in the data. If you scroll down to the beta fixed effect parameters, as shown in the table below, you will find that their mean estimates have changed little.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Single level Mean(sd)</th>
<th>Single level ESS</th>
<th>2level Mean(sd)</th>
<th>2level ESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta_0</td>
<td>-0.103 (0.0196)</td>
<td>1615</td>
<td>-0.091 (0.0429)</td>
<td>319</td>
</tr>
<tr>
<td>beta_1</td>
<td>0.590 (0.0126)</td>
<td>5488</td>
<td>0.560 (0.0126)</td>
<td>4951</td>
</tr>
<tr>
<td>beta_2</td>
<td>0.170 (0.0254)</td>
<td>1623</td>
<td>0.170 (0.0330)</td>
<td>775</td>
</tr>
</tbody>
</table>

The standard deviations for beta_0 and beta_2 have increased due to taking account of the clustering, and the ESS values have reduced due to correlation in estimating the fixed effects and level 2 residuals.

### 3.6 Caterpillar plot

The random effects in the 2-level model are also interesting to look at, and one graph that is often used is a caterpillar plot. This can be produced in Stat-JR using a template specifically designed for producing this plot. This template requires the user to select all the ‘u’s to be displayed in the plot, which can be time-consuming if there are many of them:

From the top bar we need to select **Choose for Template and Dataset.**
Choose **CaterpillarPlot95** as the template and **out2level** as the dataset.

You need now to select all the ‘u’s from \(u_0\) to \(u_{64}\) which is best done by clicking on \(u_0\) and holding down the mouse and scrolling down to multiselect all the ‘u’s together.

Once all are selected press the **Next** and **Run** buttons.

Select **caterpillar.svg** in the pull down list and view in a new tab as follows:

**Dataset:** out2level; **Template:** CaterpillarPlot95; **Input string:** {'residuals':'\(u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8, u_9, u_{10}, u_{11}, u_{12}, u_{13}, u_{14}, u_{15}, u_{16}, u_{17}, u_{18}, u_{19}, u_{20}, u_{21}, u_{22}, u_{23}, u_{24}, u_{25}, u_{26}, u_{27}, u_{28}, u_{29}, u_{30}, u_{31}, u_{32}, u_{33}, u_{34}, u_{35}, u_{36}, u_{37}, u_{38}, u_{39}, u_{40}, u_{41}, u_{42}, u_{43}, u_{44}, u_{45}, u_{46}, u_{47}, u_{48}, u_{49}, u_{50}, u_{51}, u_{52}, u_{53}, u_{54}, u_{55}, u_{56}, u_{57}, u_{58}, u_{59}, u_{60}, u_{61}, u_{62}, u_{63}, u_{64}\)'}

![Graph showing schools in order of ascending mean with 95% confidence interval](caterpillar.svg)

This graph shows the schools in order of ascending mean whilst the bars give a 95% confidence interval around each mean. The school in the middle with the wide confidence interval (i.e. very large bars) has only 2 pupils and so there is much greater uncertainty in the estimate.

In this chapter we have explored fitting three models to the tutorial dataset. This has illustrated how the Stat-JR system works, how to interpret the output from MCMC and eStat, and how to compare models via the DIC diagnostic. There are better models that can be fitted to the dataset: for example, we could look at treating the effect of intake score (standlrt) as random, and fit a random slopes model using the template **2LevelRS**; in the future we may add material on this subject to this manual, but for now we leave this as an exercise for the reader. Next we turn to the interoperability features of Stat-JR.
4 Interoperability – a brief introduction

In this section we look at interoperability with other software packages.

In order for Stat-JR to interoperate with a third-party package, the user needs to check that the Stat-JR template he/she wishes to use with it supports interoperability with that third-party software package (this can be checked via Template > Choose in the black bar at the top of the TREE interface, and then either using the red cloud terms or by clicking on the ‘cog’ symbol next to the name of the currently-selected template), that that third-party package is installed, and that Stat-JR knows where to find it (see the paths specified in Settings via the black bar at the top of the TREE interface; note that if you change a path, then make sure you press the Set button at the bottom of the Settings screen and then select Debug > Reload packages (via the black bar at the top) to implement this change in the current session).

So, whilst all the templates used in this section support interoperability with the packages we explore, in order to run this section successfully, the user will need to have these packages installed and to have told Stat-JR where to find them.

Stat-JR can interoperate with a variety of third-party statistical packages (see http://www.bristol.ac.uk/cmm/software/statjr/downloads/additionalsoft.html for more details), including the following:

- aML
- GenStat
- gretl
- JAGS
- MATLAB
- Minitab
- MIXREGLS
- MLwiN
- Octave
- OpenBUGS
- PSPP
- R
- SABRE
- SAS
- SPSS
- Stata
- SuperMix
- WinBUGS

...as well as, as we’ve seen, being able to use its own in-house model estimation engine (eStat), and a variety of Python (which is the main language in which Stat-JR is written) functions.

In this section we demonstrate interoperation by selecting a few of these third-party packages.

4.1 So why are we offering interoperability?
There are many motivations that could be given for the benefits of having an interoperability interface. First and foremost it opens up functionality in other software packages through a common interface.

One important feature that the template, Regression1AML (which we cover at the end of this chapter), shows is that not all model templates need to use the built-in eStat engine. It would be perfectly reasonable for a user to construct a template that fitted a specific family of models in the WinBUGS software and then novice users would have access to a user-friendly interface to such models without having to understand the subtleties of writing WinBUGS code; it can thus play an important role introducing packages, such as WinBUGS, to new users. This follows earlier work: for example the MLwiN-WinBUGS interface that we developed 10 years ago.

It also offers an easy way of comparing different software packages for a multitude of examples, and we will return to this in Section 5.4. Finally it can be thought of as a teaching tool, so that a user familiar with one package can use Stat-JR and directly compare the script files, etc., required for the package with which they are familiar to those required for an alternative package.

**4.2  Regression in eStat revisited**

In Section 3 we looked at fitting a few models to the tutorial dataset using the built-in eStat engine: a newly-developed estimation engine with the advantage of being transparent in that all the algebra, and even the program code, is available for inspection. It is an MCMC-based estimation method, but is also rather quick. In this chapter we will stick with one simple example, the initial linear regression model that we fitted to the ‘tutorial’ dataset that we considered in Section 3. We will need to use a new template, Regression2, as the Regression1 template only supports the eStat engine.

We will begin by setting-up the model and running it in eStat:

```
From the top bar select Regression2 as the template, and tutorial as the dataset using the Choose options on the pull-down lists for templates and datasets and set-up the inputs as follows:
```
Dataset: tutorial; Template: Regression2; Input string: {'Engine': 'eStat', 'burnin': '500', 'defaultsv': 'Yes', 'outdata': 'outestat', 'thinning': '1', 'nchains': '3', 'defaultalg': 'Yes', 'iterations': '2000', 'y': 'normexam', 'x': 'cons,standlrt', 'seed': '1', 'makepred': 'No'}

Click on Next and Run to fit the model.
Select ModelResults from the pull down list, and show this output in a new tab which should look as follows:

```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>mean</th>
<th>sd</th>
<th>ESS</th>
<th>variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>tau</td>
<td>1.541609595074</td>
<td>0.0340056114631</td>
<td>5799</td>
<td></td>
</tr>
<tr>
<td>beta_0</td>
<td>-0.00127935184971</td>
<td>0.0125770014327</td>
<td>5960</td>
<td>cons</td>
</tr>
<tr>
<td>beta_1</td>
<td>0.564956154334</td>
<td>0.012745358164</td>
<td>6129</td>
<td>standlrt</td>
</tr>
<tr>
<td>sigma2</td>
<td>0.648087956705</td>
<td>0.0143068971085</td>
<td>5784</td>
<td></td>
</tr>
<tr>
<td>sigma</td>
<td>0.80564947358</td>
<td>0.0089775878981</td>
<td>5789</td>
<td></td>
</tr>
<tr>
<td>deviance</td>
<td>9763.4884832</td>
<td>2.43302399601</td>
<td>6061</td>
<td></td>
</tr>
</tbody>
</table>

Model:

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dbar</td>
<td>9763.4884832</td>
</tr>
<tr>
<td>D(theta_bar)</td>
<td>9760.5097897</td>
</tr>
<tr>
<td>pD</td>
<td>2.9766934714</td>
</tr>
<tr>
<td>DIC</td>
<td>9766.46716766</td>
</tr>
</tbody>
</table>
```
These results are identical to those we obtained using Regression1 earlier, although we only looked at the plot for \( \beta_1 \) in Section 3.3. We will use this as a benchmark, contrasting these results with those we obtain from the other packages, although it is worth noting that all packages will have good mixing and converge quickly for this simple linear regression model. You might like to explore differences between engines / packages for other models yourself after reading this chapter.

4.3 Interoperability with WinBUGS

WinBUGS (Lunn et al., 2000) is an MCMC-based package developed (as BUGS – Bayesian inference Using Gibbs Sampling) originally in the early 1990s by a team of researchers at the MRC Biostatistics Unit in Cambridge. It is a very flexible package and can fit, in a Bayesian framework, most statistical models, provided you can describe them in its model specification language. In Stat-JR we have borrowed much of this language for our own algebra system, and so many templates feature interoperability with WinBUGS.

To fit the current model using WinBUGS we can click on remove next to the Choose estimation engine input and set up the template inputs as follows:

**Dataset: tutorial; Template: Regression2; Input string:**

```json
{ 'Engine': 'WinBUGS', 'burnin': '500', 'defaultsv': 'Yes', 'outdata': 'outwinbugs', 'thinning': '1', 'nchains': '3', 'iterations': '2000', 'y': 'normexam', 'x': 'cons,standlrt', 'seed': '1' }
```

When we press **Next** the Stat-JR software will construct all the files required to run WinBUGS so for example we can choose *model.txt* from the list:
Here we see the model defined in the WinBUGS model specification language in the output pane. This file is almost identical to that used by eStat aside from the expression `length(normexam)` being replaced here by its value 4059.

Selecting `script.txt` from the list and popping out to a new tab gives the following:

```
# This file contains the model specification based on the eStat system or newly defined

model{
  for (i in 1:4059) {
    normexam[i] ~ dnorm(mu[i], tau)
    mu[i] <- cons[i] * beta_0 + stand[i] * beta_1
  }
  # Priors
  beta_0 ~ dflat()
  beta_1 ~ dflat()
  tau ~ dgamma(0.0001, 0.0001)
  sigma2 ~ 1/ln(ita)
  sigma ~ 1/ln(ita)
}
```

Here we see a list of the commands to be run in the WinBUGS language to fit the model. Note that this is done using a temporary directory and so this pathname appears in many commands.

Return to the tab containing the main page and click on the Run button.

The WinBUGS package then pops up in its own window, runs the above script, and returns control to Stat-JR when it has finished estimating the model.
If we look at the *ModelResults* output from the list and pop it out to its own tab we will see the following:

### Results

**Parameters:**

<table>
<thead>
<tr>
<th>parameter</th>
<th>mean</th>
<th>sd</th>
<th>ESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta_0</td>
<td>-0.0010441340655</td>
<td>0.0126334002778</td>
<td>5728</td>
</tr>
<tr>
<td>beta_1</td>
<td>0.5947198</td>
<td>0.0127951159423</td>
<td>6685</td>
</tr>
<tr>
<td>deviance</td>
<td>6763.501</td>
<td>2.46491892508</td>
<td>6148</td>
</tr>
<tr>
<td>sigma</td>
<td>0.0005993303333</td>
<td>0.0099070237399</td>
<td>5750</td>
</tr>
<tr>
<td>sigma2</td>
<td>0.0490518333333</td>
<td>0.0143578592423</td>
<td>5761</td>
</tr>
<tr>
<td>tau</td>
<td>1.5414645</td>
<td>0.034078493477</td>
<td>5748</td>
</tr>
</tbody>
</table>

**Model:**

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dbar_normexam</td>
<td>9703.5</td>
</tr>
<tr>
<td>Dhat_normexam</td>
<td>9700.51</td>
</tr>
<tr>
<td>pD_normexam</td>
<td>2.986</td>
</tr>
<tr>
<td>DIC_normexam</td>
<td>9766.48</td>
</tr>
<tr>
<td>Dbar_total</td>
<td>9703.5</td>
</tr>
<tr>
<td>Dhat_total</td>
<td>9760.51</td>
</tr>
<tr>
<td>pD_total</td>
<td>2.986</td>
</tr>
<tr>
<td>DIC_total</td>
<td>9786.48</td>
</tr>
</tbody>
</table>

These estimates, as one might expect, are very close to those from eStat, and again all ESS values are around 5,000-6,000. We can also look at the log file from WinBUGS:

Return to the template tab and choose *log.txt* in the outputs list.

Scroll the *log.txt* file down to the bottom, and the screen should look as follows:
Here we see that the estimates and the DIC diagnostic are embedded in the log file, and take a similar value to eStat. WinBUGS required initial value files for each run (and these are stored in three text files beginning with *init* and the chain number), together with a data file as well as the model and script files already shown. All of these are available to view and to use again, thus Stat-JR is useful for learning how these other packages, such as WinBUGS, work.

### 4.4 Interoperability with OpenBUGS

Our next package to consider is OpenBUGS (Lunn et al., 2009). OpenBUGS was developed by members of the same team who developed WinBUGS, but differs in that it is open source so other coders may get access to the source code, and in theory develop new features in the software.

To run OpenBUGS via Stat-JR click on the word *remove* next to the *Choose Estimation engine* input, set up the template as follows, and then click on *Next*:
Dataset: tutorial; Template: Regression2; Input string: {'Engine': 'OpenBUGS', 'burnin': '500', 'defaultsv': 'Yes', 'outdata': 'outopenbugs', 'thinning': '1', 'nchains': '3', 'iterations': '2000', 'y': 'normexam', 'x': 'cons,standlrt', 'seed': '1'}

This will have set-up the files required for OpenBUGS; these are similar, but not identical, to WinBUGS: the script file, in particular, is somewhat different and is split into three parts called initscript.txt, runscript.txt (shown below) and resultsscript.txt (you can access these from the objects list):

OpenBUGS allows us to change the working directory, and so there is no need for other commands to include the temporary directory path. Unlike WinBUGS, OpenBUGS will run in the background, and so will not appear when we click run.

Clicking on Run and selecting ModelResults in its own tab gives the following:
Again, these results are very similar in terms of parameter estimates and ESS values to the other software packages.

### 4.5 Interoperability with JAGS

The third standalone MCMC estimation engine available, via Stat-JR, is JAGS (“Just Another Gibbs Sampler”), developed by Martyn Plummer (Plummer, 2003). JAGS also uses WinBUGS model language, but has a few differences in terms of script files and data files.

To run JAGS via Stat-JR click on the **remove** text next to **Choose estimation engine** and set-up the template as follows, before clicking on **Next**:
**Dataset:** tutorial; **Template:** Regression2; **Input string:**

```
{'Engine': 'JAGS', 'burnin': '500', 'defaultsv': 'Yes', 'outdata': 'outjags', 'thinning': '1', 'nchains': '3', 'iterations': '2000', 'y': 'normexam', 'x': 'cons,standlrt', 'seed': '1'}
```

This will set-up the files required for JAGS; for example, here you can see the script file (script.txt) which show some differences to those for WinBUGS (as to the initial value file formats):

```
load dic: model is 'mod.txt'
data in 'data.txt'
compile, makedir1();
parameters in 'init1.txt', chain1();
parameters in 'init2.txt', chain2();
parameters in 'init3.txt', chain3();
Initialize
update 5e
monitor tau, thin1();
monitor deviance, thin1();
monitor beta & thin1();
monitor beta1, thin1();
monitor sigma, thin1();
monitor sigma2, thin1();
monitor end, thin1();
monitor p0
update 2000
code *, state('results')
parameters to 'chainstate1.txt', chain1();
parameters to 'chainstate2.txt', chain1();
parameters to 'chainstate3.txt', chain1();
samplers to 'samplers.txt'
exit
```

Like OpenBUGS, JAGS will run in the background (i.e. it will not open as a window on your screen).

**Clicking on Run**, and placing **ModelResults** in a new tab, gives the following:
As you can see, we have similar estimates and effective sample sizes to the other estimation methods we've used. Whilst JAGS can be faster than WinBUGS and OpenBUGS, it fits a slightly smaller number of models.

### 4.6 Interoperability with MLwiN
MLwiN (Rasbash et al. 2009) is a software package specifically written to fit multilevel statistical models. It features two estimation engines (for MCMC and likelihood-based (IGLS) methods, respectively) with a menu-driven, point-and-click user interface. It also has an underlying macro language, however, and this is what we use to interoperate with Stat-JR. We will first consider the MCMC engine. As it is limited in the scope of models it fits, this means it is generally quicker than the other MCMC packages. MLwiN is a single chain program, but can be made into a multiple chain engine with Stat-JR, since the latter can start-up three separate instances of MLwiN. At present these are given different random number seeds, but the same starting values, however we will try and change this in future.

To run MCMC in MLwiN, via Stat-JR, click on the remove text by Choose estimation engine input and set-up the template as follows before clicking on **Next**:
**Dataset:** tutorial;  **Template:** Regression2;  **Input string:** \{'Engine': 'MLwiN_MCMC', 'burnin': '500', 'outdata': 'outmlwin', 'thinning': '1', 'nchains': '3', 'defaultalg': 'Yes', 'iterations': '2000', 'y': 'normexam', 'x': 'cons,standlrt', 'seed': '1'}

You can see, in the pull-down list, the dataset (in .dta format) that is used by MLwiN. There are also several MLwiN script files for the multiple chains and the several stages of model fitting.

**Clicking on the Run button** will set off three instances of MLwiN (in the background) and Stat-JR will then collate the results together. Choosing **ModelResults**, and displaying them in a new tab, gives the following:

Once again here we have similar estimates, although the naming convention is slightly different for MLwiN. To show that we have multiple chains we can examine the chains for the slope (**beta3**), as shown below:
Stat-JR also offers the option of using the likelihood-based IGLS estimation engine in MLwiN.

To do this in MLwiN, via Stat-JR, click once again on the remove text next to the Choose estimation engine input and set-up the template as follows, before clicking on Next:

Dataset: tutorial; Template: Regression2; Input string: {'y': 'normexam', 'x': 'cons,standlrt', 'Engine': 'MLwiN_IGLS', 'defaultalg': 'Yes'}

Again the dataset will appears in the output pane, and this time pressing Run will give the following in the ModelResults output:
Here we get the Deviance (-2*Loglikelihood) value, together with parameter estimates with standard errors. The likelihood-based methods are far faster to run than the MCMC-based methods.

4.7 Interoperability with R

R (R Core Team, 2016) is another more general purpose package that can be used to fit many statistical models. R has many parallels with Stat-JR in that users can supply functions (like Stat-JR templates) which are then added to the library of R packages. Here we will demonstrate fitting a model using the R package MCMCglmm (Hadfield, 2010), which is MCMC-based, and also using the glm function (from R’s stats “base package”), which is a standard regression modelling function\textsuperscript{1}. We will firstly demonstrate fitting a model using R’s MCMCglmm package.

To run MCMC in R, via Stat-JR, click on the remove text by the Choose estimation engine input and set-up the template as follows, and click on Next:

\textsuperscript{1} Interoperability is also offered via R’s nimble package (de Valpine et al, 2016), although note that this in turn has a dependency on Rtools (https://cran.r-project.org/bin/windows/Rtools/) since it compiles code dynamically.
Dataset: tutorial; Template: Regression2; Input string: {'Engine': 'R_MCMCglmm', 'burnin': '1000', 'outdata': 'outR', 'thinning': '1', 'iterations': '5000', 'y': 'normexam', 'x': 'cons,standlrt', 'seed': '1'}

After pressing **Next**, if we look at the script file, *script.R*, which we can select from the outputs list, we see the following:

```r
local({
  CRAN <- paste0("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.
# writeDir = 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(forlign)) {
  install.packages("forlign")
  library(forlign)
}
# use foreign package to read .dta file (stata format) into R data frame ('mydata')
mydata <- read.dta('datafile.dta')
# print summary of the data
summary(mydata)
# test to see if MCMCglmm package is already installed, if not, then install it
if (!require(MCMCglmm)) {
  install.packages("MCMCglmm")
  library(MCMCglmm)
}
# specify starting seed for random number generator
set.seed(1)
#
# Below we specify the model formula, formatted as y = x1 + x2 + ...
# Since Stat-JR assumes users have included the Intercept in their list
# of explanatory variables, -1 removes the Intercept which the glmer
# function otherwise adds by default.
Formula <- normexam ~ cons + standlrt - 1
#
# Here we define the prior. B refers to the fixed effects, a list
# consisting of the (co)variance matrix, V, and expected value, mu,
# As such the expected value for each fixed effect has a mean of zero,
# and the diagonal variance matrix has large variances (1e6). R (the
# n-structure: expected (co)variances of the residuals) is an inverse
# wishart with expected variance (V) of 1, and degree of belief
# parameter (nu) of 0.002 (equivalent to inverse Gamma(0.001, 0.001)).
```
MCMCglmm can fit all forms of generalised linear mixed models, of which a linear regression is a rather trivial case. You will see that the script file contains some setup code which will actually download and install the MCMCglmm library the first time you execute the script (so ensure your machine is connected to the internet) before calling the MCMCglmm command and then producing summaries.

Clicking on Run in the main window will create several outputs.

The ModelResults are similar to other software but we can also look at diagnostics plots that are specific to R by selecting DiagPlots1.png:

Here R gives trace plots and kernel density plots for both the intercept and the slope parameter.

Turning next to the glm function we can click on the remove text by Choose estimation engine and set-up the template as follows, before clicking on Next:

Dataset: tutorial; Template: Regression2; Input string: {'y': 'normexam', 'x': 'cons,standlrt', 'Engine': 'R_glm'}

Clicking on Run will return results in ModelResults as usual. There are additional graphical plots that come back from R; for example, below is a plot of residuals of the model fit against fitted values (ResivsFitted.svg).
Before finishing with R, we will also demonstrate a non-model Stat-JR template which interoperates with R called PlotsViaR; this gives the Stat-JR user access to R’s lattice (Sarkar, 2008) graphics package through the Stat-JR interface.

Click on Choose from the Template pull-down list at the top of the screen to get a list of all the templates. Note that the search cloud is useful with interoperability as it can be used to show which templates offer interoperability with a particular package (the engines are in red).

Click on Plots and also R_script in the blue tag cloud. You’ll see that the list of templates, underneath, is accordingly reduced to just those that draw plots using R.

Select PlotsViaR from the list, and click Use.

Set up the template inputs as shown below:
**Dataset:** tutorial; **Template:** PlotsViaR; **Input string:** {'var1': 'normexam', 'Gp': 'Yes', 'group': 'girl', 'trellis1': 'schgend', 'howmany': 'One', 'plottypeGUI': 'Density Plot', 'striptitle': 'Yes'}

These options will display kernel plots for the exam scores of pupils grouped by gender, with separate (panelled or trellised) plots for each school gender type. We can now press Run and show the plot (Plot1.svg) in a separate tab:

---

**4.8 Interoperability with AML**

We will next look at another software package that can fit many statistical models via likelihood-based estimation. AML (Lillard & Panis, 2003) is very useful for fitting multi-process models, but as with other software packages can fit a simple regression as a special case. In our development work on Stat-JR we have written special templates for interoperability with AML as opposed to incorporating interoperability in the standard templates. We therefore need to do the following:
Click on the **Choose** option from the Template pull-down list.

Select **Regression1AML** from the template list and click on **Use**, and stick with the **tutorial** dataset.

Note that if you have earlier clicked on Plots and R_script in the cloud of terms you will need to either unselect them or click on [reset] to see the required template.

Fill in the inputs as follows, and press **Next**:

**Dataset:** tutorial; **Template:** Regression1AML; **Input string:** {'y': 'normexam', 'x': 'cons,standlrt'}

![Image of model inputs](image-url)

Now click on **Run** to run the model in AML and select **ModelResults** from the list:

![Image of Run button](image-url)

Here we see the model results are similar to other packages. AML has three input datasets: amlfit.raw, amlfit.aml and amlfit.r2a. There are also three additional output files from AML: amlfit.out, amlfit.tab and amlfit.sum. For more information on how AML works we recommend looking at the reference manual for the software.

5 Application 2: Analysis of the Bangladeshi Fertility Survey dataset

5.1 The Bangladeshi Fertility Survey dataset
The Bangladeshi dataset (bang1) is an example dataset from the 1988 Bangladeshi Fertility Survey. It contains records from 1934 women based in 60 districts in Bangladesh, and we are planning to investigate variables that predict whether the women were using contraception or not at the time of the survey. Let us first look at the data and the variables we will consider.

Select Choose and pick bang1 from the Dataset list and click on Use. Click on View from the Dataset list to view the data as follows:

Here we see records for the first 27 women in district 1 displayed. The response variable use takes value 1 if the woman was using contraceptives during the time of the survey, and 0 if she was not. There are then several predictor variables, both woman-level and district-level. Here we will focus on just two: the number of living children (lc), which is a categorical variable with four categories (no kids, one kid, two kids, three+kids), and the respondents’ age, which is measured to the nearest year and has been centred around its grand mean. We will now consider modelling the dataset.

5.2 Modelling the data using logistic regression

We will firstly consider a simple linear regression model relating contraception use to the age of the woman.

Choose the template 1LevelMod from the Template list and click on Use. Then setup the model with inputs as below.
Dataset: bang1; Template: 1LevelMod; Input string: {'Engine': 'eStat', 'burnin': '500', 'D': 'Binomial', 'outdata': 'out', 'n': 'cons', 'nchains': '3', 'thinning': '1', 'link': 'logit', 'defaultalg': 'Yes', 'iterations': '2000', 'y': 'use', 'x': 'cons, age', 'makepred': 'No', 'seed': '1', 'defaultsv': 'Yes'}

Clicking on Next and choosing equation.tex in the pull down list and we see the following:

Here we the logistic regression model, in LaTeX, in the output pane. If we select model.txt we can then see the model code that the algebra system will interpret:
Now choosing `algorithm.tex` from the output pane, and placing it in its own tab in the browser window, gives the following:

Whilst it is a little difficult to see in this screenshot, you will see better on your own screen that the eStat engine uses a different MCMC method, random walk Metropolis, for the steps for the fixed effects (beta0 and beta1) when fitting logistic regression models. We will come back to this modelling decision in Section 5.4 when we compare different software packages.

Returning to the main pane and clicking on Run will now run the model. Once it has finished, if we select ModelResults from the list, and look at it in a new tab, we get the following:
Age doesn’t appear to have a significant effect (its estimate (0.0064) is similar in magnitude to its standard error (0.0051)). To see this more clearly we can look at the graph beta_1.svg in its own browser tab.

Here, whilst the values on the x-axis overlap and therefore aren’t particularly clear, we can see that all three chains show strong support for the value 0.00 in the kernel density plot (i.e. it’s comfortably within the distribution). It might be the case, however, that contraceptive use has a non-linear relationship with age (possibly quadratic) and this could also be confounded by how far through their own family-formation process the woman is, which we will model via the variable lc. We might also be interested in accounting for any clustering effects of having women nested within districts.

In order to fit a quadratic function to age we will need to construct the variable age² which we can easily do via Dataset > View and using the variable creation tool.

Return to the main screen and select View from the Dataset pull-down list at the top of the page. Click on the Add Variable tab and type the following (New Variable name: age2; Expression: age*age):
Here we are going to overwrite the existing dataset (at least in temporary memory) with a version to which we have appended an additional column. Clicking on Create and looking at the data by (clicking on the Data tab) below gives the following:

Here you see age2 ($age^2$) appearing in the column on the far right. Whilst we could explore adding further explanatory variables to this 1-level model, we are going to move straight into fitting a 2-level model to account for districts in which we will also investigate the effect of a quadratic function of age.

### 5.3 Multilevel modelling of the data

We will now require a template that will fit a 2-level logistic regression model to our dataset. In the earlier sections we looked at the template **2LevelMod** and we will once again use it here and also illustrate how to fit categorical predictor variables.

On the main tab, click on **Choose** in the **Template** pull-down list and select **2LevelMod** and click on **Use** button to run this template.

Fill in the template inputs as follows:
Here we need to specify several extra inputs, including an input for the level 2 identifiers and also to let the software know which predictor variables are categorical (by ticking the box indicating that the variable lc is categorical). Continue with the inputs as follows:

**Dataset: bang1; Template: 2LevelMod; Input string:**

```plaintext
{'D': 'Binomial', 'storeresid': 'No', 'nchains': '3', 'link': 'logit', 'defaultalg': 'Yes', 'iterations': '2500', 'outdata': 'out', 'seed': '1', 'defaultsv': 'Yes', 'Engine': 'eStat', 'L2ID': 'district', 'burnin': '2500', 'n': 'cons', 'thinning': '1', 'y': 'use', 'x': 'cons,age,age2,lc:cat', 'makepred': 'No'}
```
Clicking on **Next** will run the algebra system and set up code to fit the model. If we select *model.txt* in the output list we will see the following:

```
model {
  for (i in 1:length(use)) {
    logit(p[i]) <- c(1) * beta_0 + age[i] * beta_2 + age[i] * beta_3 + lc_1[i] * beta_4 + lc_2[i] * beta_5 + u[dist[i]]
  }
  for (i in 1:length(a)) {
    a[i] = norm(mu, tau_u)
  }
  # Priors
  beta_0 ~ dflat()
  beta_2 = dflat()
  beta_3 = dflat()
  beta_4 = dflat()
  beta_5 = dflat()

  tau_u = dgamma(0.000001, 0.000000)
  sigma2_u <- 1 / tau_u
}
```

Here we see the more complicated model code for this 2-level model in the output pane. Note that the *lc* predictor is treated as categorical and thus appears as three dummy variables (*lc_1, lc_2, lc_3*)
If we select \( \text{tau}_u.xml \) in the output list we will see the following:

Here we see the algorithm step for the parameter \( \text{tau}_u \). Although most parameters in this model are updated by Random Walk Metropolis sampling, this parameter is updated by Gibbs Sampling as its conditional posterior distribution has a standard form.

If we now click on Run then after 52s (on a machine with Intel Core i7-3770S; this includes time for compiling and adapting) the model will have run and if we select \( \text{sigma2}_u.svg \) we will see the following:

Here we can see that convergence and mixing, for this parameter at least, are reasonable. In fact, if we look at the diagnostic plots for the other parameters, we see similar convergence there as well.

Next we can look at ModelResults in its own tab to see the parameter estimates:
Here we see that $\beta_2$ (the coefficient estimate for $age_2$) is significant and negative (and larger than $\beta_1$ (age)) suggesting a quadratic fit to the age predictor. As the data is centred around its mean, this implies that contraceptive use is reduced the further from the mean age the woman is. We will look at this in more detail at the end of the chapter.

The parameters $\beta_3$ to $\beta_5$ are all significant, and positive (and of similar magnitude), which suggests that women with children are more likely to use contraceptives than those without (since the reference category here is nokids). The parameter $\sigma^2_u$ is fairly large, suggesting there are differences between districts in terms of contraceptive use.

What is slightly disappointing here are the ESS values for all the fixed parameters. We have run each chain, after burnin, for 2,500 iterations resulting in a total of 7,500 actual iterations (i.e. from 3 chains) but the effective sample sizes are of the order of 100-350. As this indicates, the default algorithm in eStat – random walk Metropolis – is not very efficient for this example. We will look at two possible solutions in the next two sections.

### 5.4 Comparison between software packages

Not all software packages fit the same MCMC algorithm for this model. So here we will show how to fit the same model in another package, OpenBUGS (Lunn et al, 2009), which uses a different method: namely multivariate updating for the fixed effects in a GLMM, as developed by Gamerman (1997). This method results in slower estimation, but, as we will see, far better ESS. We will then look at a table comparing all the possible MCMC algorithms in the different packages for this model, which you can verify for yourselves.

To fit the model in OpenBUGS click on the remove text next to Choose estimation engine and set-up the model as follows:
Dataset: bang1; Template: 2LevelMod; Input string: 

```{"Engine": 'OpenBUGS', 'L2ID': 'district', 'burnin': '2500', 'D': 'Binomial', 'outdata': 'outopenbugs', 'storeresid': 'No', 'n': 'cons', 'nchains': '3', 'thinning': '1', 'link': 'logit', 'iterations': '2500', 'y': 'use', 'x': 'cons,age,age2,lc:cat', 'seed': '1', 'defaultsv': 'Yes'}
```

Clicking on **Next** and **Run** will (after 2 min 18s on my machine) give the following, having selected **ModelResults** from the drop-down box above the output pane, and opening it in a new tab:
Here we see far better effective sample size values, with runs of 7,500 iterations translating into ESS values of between 2,500 and 5,500 for the beta parameters.

We can repeat this analysis using WinBUGS, JAGS and MLwiN with the same run lengths. Note for JAGS you will need to edit the initial value files or it will not run. To do this view each in the output window and click on the Edit button. If you change the value for beta_2 (the fixed effect associated with age2) from 0.1 to 0.0 in all three initial values files and click Save each time then JAGS should run. It should also be noted here that results may vary a little if you have different versions of the third party software packages or have changed options in them.

We could also fit the model using the MCMCglmm package in R, although here we would need to run a single chain and logistic regression models for binary data are the one GLMM where the answers can be a little different as it assumes over-dispersion which is inappropriate in this case.
The table overleaf details the results of fitting many of these options (unless otherwise stated, each with **Dataset**: bang1; **Template**: 2LevelMod):

**Input string – eStat:** `{D: 'Binomial', 'storeresid': 'No', 'nchains': '3', 'link': 'logit', 'defaultalg': 'Yes', 'iterations': '2500', 'outdata': 'out', 'seed': '1', 'defaultsv': 'Yes', 'Engine': 'eStat', 'L2ID': 'district', 'burnin': '2500', 'n': 'cons', 'thinning': '1', 'y': 'use', 'x': 'cons,age,age2,lc:cat', 'makepred': 'No'}

**Input string – WinBUGS:** `{Engine': 'WinBUGS', 'L2ID': 'district', 'burnin': '2500', 'D': 'Binomial', 'outdata': 'outwinbugs', 'storeresid': 'No', 'n': 'cons', 'nchains': '3', 'thinning': '1', 'link': 'logit', 'iterations': '2500', 'y': 'use', 'x': 'cons,age,age2,lc:cat', 'seed': '1', 'defaultsv': 'Yes'}

**Input string – OpenBUGS:** `{Engine': 'OpenBUGS', 'L2ID': 'district', 'burnin': '2500', 'D': 'Binomial', 'outdata': 'outopenbugs', 'storeresid': 'No', 'n': 'cons', 'nchains': '3', 'thinning': '1', 'link': 'logit', 'iterations': '2500', 'y': 'use', 'x': 'cons,age,age2,lc:cat', 'seed': '1', 'defaultsv': 'Yes'}

**Input string – JAGS (remember to change the initial values files before running—see above):**
{"Engine': 'JAGS', 'L2ID': 'district', 'burnin': '2500', 'D': 'Binomial', 'outdata': 'outjags', 'storeresid': 'No', 'n': 'cons', 'nchains': '3', 'thinning': '1', 'link': 'logit', 'iterations': '2500', 'y': 'use', 'x': 'cons,age,age2,lc:cat', 'seed': '1', 'defaultsv': 'Yes'}

**Input string – MLwiN:** `{Engine': 'MLwiN_MCMC', 'L2ID': 'district', 'burnin': '2500', 'D': 'Binomial', 'outdata': 'outmlwin', 'storeresid': 'No', 'n': 'cons', 'nchains': '3', 'thinning': '1', 'link': 'logit', 'defaultalg': 'Yes', 'iterations': '2500', 'y': 'use', 'x': 'cons,age,age2,lc:cat', 'seed': '1'}

**Input string – eStat – orthogonal parameterisation (see Section 5.5); Template:**
**NLevelOrthogParamRS:** `{Engine': 'eStat', 'x1': 'cons', 'burnin': '2500', 'D': 'Binomial', 'outdata': 'outorthog', 'storeresid': 'No', 'thinning': '1', 'link': 'logit', 'defaultalg': 'Yes', 'iterations': '2500', 'y': 'use', 'x': 'cons,age,age2,lc:cat', 'seed': '1', 'defaultsv': 'Yes', 'makepred': 'Yes', 'defaultsv': 'Yes'}

---

2 This particular comparison used WinBUGS 1.4.3, OpenBUGS 3.2.3, JAGS 4.2.0 (64-bit), MLwiN 2.36, all run on Windows 64-bit machine with Intel Core i7-3770S; eStat times are of the form: *including compiling time* (excluding compiling time).
In summary we see that MLwiN is by far the fastest of the packages, with eStat quicker than the other three as well. Both MLwiN and eStat use the simple random walk Metropolis algorithm, which is not the best method for this model and gives fairly poor ESS. Interestingly, both WinBUGS and OpenBUGS use the Gamerman method, but in this case OpenBUGS performs better in terms of time taken and ESS. This is somewhat puzzling as when each is run with a single chain, their performance is almost identical. Finally, JAGS is slower than the two BUGS packages with ESS generally poorer too; however, there have been many comparisons between JAGS and BUGS for different models, and which method is better varies from model to model, so we need to take care when making comparisons based on just one example. The final column shows another eStat method which we will discuss next.

### 5.5 Orthogonal parameterisation

The reason eStat (and MLwiN) perform badly in terms of ESS in this instance is that they are performing single-site updating, and the parameters are correlated. So here we will consider a reparameterisation method that aims to fit parameters that are less correlated, and then translates

<table>
<thead>
<tr>
<th>Parameter</th>
<th>eStat</th>
<th>WinBUGS</th>
<th>OpenBUGS</th>
<th>JAGS</th>
<th>MLwiN</th>
<th>eStat orthogonal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beta0</td>
<td>-0.761(0.184)</td>
<td>-0.789(0.170)</td>
<td>-0.790(0.173)</td>
<td>-0.792(0.175)</td>
<td>-0.815(0.170)</td>
<td>-0.784(0.180)</td>
</tr>
<tr>
<td>ESS</td>
<td>92</td>
<td>396</td>
<td>2595</td>
<td>250</td>
<td>93</td>
<td>979</td>
</tr>
<tr>
<td>Beta1</td>
<td>0.0076(0.0097)</td>
<td>0.0068(0.0090)</td>
<td>0.0066(0.0091)</td>
<td>0.0063(0.0092)</td>
<td>0.0050(0.0089)</td>
<td>0.0068(0.0096)</td>
</tr>
<tr>
<td>ESS</td>
<td>183</td>
<td>951</td>
<td>5031</td>
<td>638</td>
<td>247</td>
<td>1792</td>
</tr>
<tr>
<td>Beta2</td>
<td>-0.0049(0.00074)</td>
<td>-0.0048(0.00072)</td>
<td>-0.0048(0.00073)</td>
<td>-0.0048(0.00072)</td>
<td>-0.0047(0.00071)</td>
<td>-0.0048(0.00073)</td>
</tr>
<tr>
<td>ESS</td>
<td>319</td>
<td>1286</td>
<td>5057</td>
<td>911</td>
<td>315</td>
<td>1799</td>
</tr>
<tr>
<td>Beta3</td>
<td>0.761(0.164)</td>
<td>0.779(0.160)</td>
<td>0.782(0.162)</td>
<td>0.784(0.163)</td>
<td>0.799(0.162)</td>
<td>0.779(0.165)</td>
</tr>
<tr>
<td>ESS</td>
<td>227</td>
<td>1117</td>
<td>5291</td>
<td>635</td>
<td>268</td>
<td>1686</td>
</tr>
<tr>
<td>Beta4</td>
<td>0.809(0.192)</td>
<td>0.822(0.181)</td>
<td>0.826(0.186)</td>
<td>0.827(0.185)</td>
<td>0.856(0.183)</td>
<td>0.823(0.190)</td>
</tr>
<tr>
<td>ESS</td>
<td>169</td>
<td>780</td>
<td>5181</td>
<td>484</td>
<td>196</td>
<td>1726</td>
</tr>
<tr>
<td>Beta5</td>
<td>0.805(0.192)</td>
<td>0.824(0.180)</td>
<td>0.828(0.184)</td>
<td>0.832(0.181)</td>
<td>0.863(0.177)</td>
<td>0.823(0.191)</td>
</tr>
<tr>
<td>ESS</td>
<td>114</td>
<td>547</td>
<td>4443</td>
<td>353</td>
<td>131</td>
<td>1666</td>
</tr>
<tr>
<td>Sigma2u</td>
<td>0.320(0.099)</td>
<td>0.318(0.100)</td>
<td>0.317(0.100)</td>
<td>0.317(0.103)</td>
<td>0.328(0.103)</td>
<td>0.322(0.101)</td>
</tr>
<tr>
<td>ESS</td>
<td>807</td>
<td>1764</td>
<td>1753</td>
<td>1282</td>
<td>733</td>
<td>756</td>
</tr>
<tr>
<td>Pd</td>
<td>42.96</td>
<td>42.44</td>
<td>42.66</td>
<td>42.16</td>
<td>43.08</td>
<td>43.21</td>
</tr>
<tr>
<td>DIC</td>
<td>2394.03</td>
<td>2393.36</td>
<td>2394.0</td>
<td>2393.40</td>
<td>2393.65</td>
<td>2394.51</td>
</tr>
<tr>
<td>Time (s)</td>
<td>40 (24)</td>
<td>192</td>
<td>137</td>
<td>217</td>
<td>8</td>
<td>36 (22)</td>
</tr>
</tbody>
</table>
them back to the original parameters. For this we construct a set of orthogonal vectors from the original predictor variables (see Browne et al. (2009) for details).

We will therefore now look at the **NLevelOrthogParamRS** template in order to use orthogonalisation on our model. This template actually fits a larger family of models: those with any number of higher levels/classifications (hence “NLevel”), allowing for the possibility of random slopes at each of these levels (hence “RS”), and so our 2-level random intercept model is perhaps the simplest case that the template fits.

Click on the **Template** pull-down list and click **Choose** then select **NLevelOrthogParamRS** from the template list.

Click on **Use** and fill in the template inputs as follows:

**Dataset**: bang1; **Template**: NLevelOrthogParamRS; **Input string**: 
{"Engine": 'eStat', 'x1': 'cons', 'burnin': '2500', 'D': 'Binomial', 'outdata': 'outorthog', 'storeresid': 'No', 'thinning': '1', 'n': 'cons', 'nchains': '3', 'orthtype': 'Orthogonal', 'link': 'logit', 'defaultalg': 'Yes', 'iterations': '2500', 'y': 'use', 'x': 'cons,age,age2,lc:cat', 'C1': 'district', 'NumLevs': '1', 'seed': '1', 'useorthog': 'Yes', 'makepred': 'Yes', 'defaults': 'Yes'}
Clicking on **Next** and selecting *equation.tex* in the pull down list (we’ve opened it in a new tab) will show the following:
Here we see that the model code is actually fitting a different set of predictors, each with the prefix ‘orth’ and a corresponding set of coefficients. There is then a set of deterministic statements that translate these coefficient values to the coefficient values for the original predictors (again, see Browne et al. (2009) for details).

Clicking on the Run button will run the model (which took 36s on this particular machine, including compiling), after which selecting ModelResults from the pull down list, and popping out into a new tab, gives the following:
The estimates, their ESS, and the time taken to run the model are all added to the end of the software comparison table we looked at above. It indicates that, compared to the other method we employed to fit the model in eStat, there is no obvious overhead incurred when performing the orthogonalising algorithm, and it is much faster than OpenBUGS, and the ESS are now much better (if still not as good as OpenBUGS). We therefore have two ways of fitting the model that are reasonably comparable in terms of ESS/s, with little to choose between them. This orthogonalising approach is also available in MLwiN: this will be faster again, and should have similar ESS to the method in eStat, and therefore may be the best overall in terms of ESS/s, but we leave this for the reader to investigate.

5.6 Predictions from the model
When we ran this model we discussed some interpretation of the fit, but it would be nice to plot some predictions from the model as well. In this latest version of Stat-JR we have added the option to store predictions when fitting the model. So hopefully in the last model fit you will have ticked “Yes” to the generate prediction dataset question. This will generate a new dataset named prediction_datafile which contains the original data and several prediction columns formed from the model fit.
To use this dataset we need to select **Choose** on the dataset list and select *prediction_datafile* from the list and click **Use**.

In fact the dataset has a full prediction column called *pred_full* but this also contains the district random effects. We would here like to simply predict from the fixed part of the model so we can construct the variable *pred_fixed* as follows:

Click on **View** from the **Dataset** menu, then choose **Add variable**, and input the new variable *pred_fixed* as indicated below.

Click on **Create** to create the variable.

This has created a variable on the fixed predictor scale but as we are fitting a logistic regression we need to take an anti-logistic transform to convert these predictions to probabilities. This can be done by creating another column in the dataset as shown below:

In order to plot separate fitted curves for the various numbers of living children we can use the template **XYGroupPlot** as shown below:
**Dataset:** prediction_datafile; **Template:** XYGroupPlot; **Input string:** `{'group': 'lc', 'xaxis': 'age', 'yaxis': 'fitprob'}`

Clicking on **Run** and popping out `graphxygroup.svg` gives the following:
Here we see the four curves (although three of them are very close together) which clearly showing that the women with children have higher probabilities of using contraceptives, and that the peak for each group is around the average age of the sample, as discussed earlier.

Hopefully this section has shown firstly that Stat-JR can fit models other than Normal response models; in fact there are a vast number of model templates which fit lots of other model classes. Secondly, we hope we’ve shown its utility in terms of comparing model-fitting across different software packages for different models, accessing each from a common hub.

6 Miscellaneous other topics e.g. Data Input/Export
Stat-JR works with datasets saved in Stata format, i.e. with a .dta extension. It looks for these in the...\datasets folder of the Stat-JR install, and also in a folder saved, by default, under your user name, e.g. C:\Users\YourName\statjr\datasets (you can change the path via Settings in the black bar at the top of the browser window in the TREE interface; if you do this then make sure you press the Set button, and then Debug > Reload datasets in the black bar at the top).

6.1 If your dataset is already in .dta format
If your dataset is already in .dta format (see below), then you can upload it, in TREE, via (i) Dataset > Upload (menu options in the black bar at the top of the browser window), which will upload it into the temporary memory cache, or by (ii) saving your dataset in one of the datasets folders (as discussed above), and then selecting Debug > Reload datasets (again, accessible via the black bar at the top of the browser window).

In the case of option (i), the dataset will be available for use in the current session, but you then need to download it (as a .dta file) via Dataset > Download (e.g. saving it into the StatJR\datasets folder) for use in the future sessions too. In the case of option (ii), the dataset will be available in future sessions since it has been saved in one of the folders in which Stat-JR searches for datasets on start-up.

6.2 If your dataset is in .txt format
If, instead, you have your dataset saved as a .txt file, you can use Stat-JR’s LoadTextFile template to save it into the temporary memory cache (the template LoadTextFileMoreOptions allows the user to specify more particulars, and can also handle string variables).

This dataset will be available for use in the current session, but you then need to download it (as a .dta file) via Dataset > Download (e.g. saving it into the StatJR\datasets folder) for use in the future sessions too.

6.3 Converting your dataset to .dta format
Via the procedure described in Section 6.2 (and downloading), Stat-JR will save your .txt dataset as a .dta file, but you can also create .dta files via Stata, MLwiN and R (e.g. the foreign package in R).

7 References


Appendix: List of Third Party Software that are used by Stat-JR

Stat-JR makes use of several third party software products that are included within the distributed code or (in the case of MinGW) need to be downloaded separately. These software products each have a license file that can be viewed from the links in the table below and/or in the licences subdirectory of the installed code.

<table>
<thead>
<tr>
<th>Package</th>
<th>Link</th>
<th>Licence terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>beautifulsoup</td>
<td><a href="http://bazaar.launchpad.net/~leonardr/beautifulsoup4/view/head:/COPYING.txt">http://bazaar.launchpad.net/~leonardr/beautifulsoup4/view/head:/COPYING.txt</a></td>
<td>MIT</td>
</tr>
<tr>
<td>BLAS</td>
<td><a href="http://www.netlib.org/blas/faq.html#2">http://www.netlib.org/blas/faq.html#2</a></td>
<td>Own licence (Netlib)</td>
</tr>
<tr>
<td>Blockly</td>
<td><a href="https://github.com/google/blockly/blob/master/LICENSE">https://github.com/google/blockly/blob/master/LICENSE</a></td>
<td>Apache (v2)</td>
</tr>
<tr>
<td>Bootstrap</td>
<td><a href="https://github.com/twitter/bootstrap/blob/master/LICENSE">https://github.com/twitter/bootstrap/blob/master/LICENSE</a></td>
<td>MIT</td>
</tr>
<tr>
<td>bottle</td>
<td><a href="https://github.com/bottlepy/bottle/blob/master/LICENSE">https://github.com/bottlepy/bottle/blob/master/LICENSE</a></td>
<td>MIT</td>
</tr>
<tr>
<td>bottle-websocket</td>
<td><a href="https://github.com/zeekay/bottle-websocket/blob/master/LICENSE">https://github.com/zeekay/bottle-websocket/blob/master/LICENSE</a></td>
<td>MIT</td>
</tr>
<tr>
<td>cssselect</td>
<td><a href="http://www.opensource.org/licenses/bsd-license.php">http://www.opensource.org/licenses/bsd-license.php</a></td>
<td>BSD</td>
</tr>
<tr>
<td>cx_freeze</td>
<td><a href="http://cx-freeze.readthedocs.org/en/latest/license.html">http://cx-freeze.readthedocs.org/en/latest/license.html</a></td>
<td>PSF</td>
</tr>
<tr>
<td>cycler</td>
<td><a href="https://opensource.org/licenses/bsd-3-Clause">https://opensource.org/licenses/bsd-3-Clause</a></td>
<td>BSD</td>
</tr>
<tr>
<td>dateutil</td>
<td><a href="http://opensource.org/licenses/bsd-2-Clause">http://opensource.org/licenses/bsd-2-Clause</a></td>
<td>Simplified BSD</td>
</tr>
<tr>
<td>decorator</td>
<td><a href="https://micheles.googlecode.com/hg/decorator/documentation.html#licence">https://micheles.googlecode.com/hg/decorator/documentation.html#licence</a></td>
<td>BSD</td>
</tr>
<tr>
<td>gevent</td>
<td><a href="https://github.com/gevent/gevent/blob/master/LICENSE">https://github.com/gevent/gevent/blob/master/LICENSE</a></td>
<td>MIT</td>
</tr>
<tr>
<td>gevent-websocket</td>
<td><a href="https://bitbucket.org/noppo/gevent-websocket/src/0df192940acd288e8a8f6d2dd30329a3381c90f1/LICENSE?fileviewer=file-view-default">https://bitbucket.org/noppo/gevent-websocket/src/0df192940acd288e8a8f6d2dd30329a3381c90f1/LICENSE?fileviewer=file-view-default</a></td>
<td>Apache(v2)</td>
</tr>
<tr>
<td>html5lib</td>
<td><a href="https://github.com/html5lib/html5lib-python/blob/master/LICENSE">https://github.com/html5lib/html5lib-python/blob/master/LICENSE</a></td>
<td>MIT</td>
</tr>
<tr>
<td>isodate</td>
<td><a href="http://www.opensource.org/licenses/bsd-license.php">http://www.opensource.org/licenses/bsd-license.php</a></td>
<td>BSD</td>
</tr>
<tr>
<td>jqgrid</td>
<td><a href="http://www.trirand.com/blog/?page_id=87">http://www.trirand.com/blog/?page_id=87</a></td>
<td>Dual MIT/GPL(v2)</td>
</tr>
<tr>
<td>jquery</td>
<td><a href="http://jquery.org/license">http://jquery.org/license</a></td>
<td>MIT</td>
</tr>
<tr>
<td>jquery-cookie</td>
<td><a href="https://github.com/carhartl/jquery-cookie/blob/master/jquery.cookie.js">https://github.com/carhartl/jquery-cookie/blob/master/jquery.cookie.js</a></td>
<td>MIT</td>
</tr>
<tr>
<td>jQuery File Upload</td>
<td><a href="http://opensource.org/licenses/MIT">http://opensource.org/licenses/MIT</a></td>
<td>MIT</td>
</tr>
<tr>
<td>jQuery text align</td>
<td><a href="http://www.opensource.org/licenses/bsd-license.php">http://www.opensource.org/licenses/bsd-license.php</a></td>
<td>BSD</td>
</tr>
<tr>
<td>jquery-treeview</td>
<td><a href="https://github.com/jzaefferer/jquery-treeview">https://github.com/jzaefferer/jquery-treeview</a></td>
<td>Dual MIT/GPL</td>
</tr>
<tr>
<td>jquery-ui</td>
<td><a href="http://jquery.org/license">http://jquery.org/license</a></td>
<td>MIT</td>
</tr>
<tr>
<td>Library</td>
<td>URL</td>
<td>License</td>
</tr>
<tr>
<td>------------</td>
<td>----------------------------------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>jQuery-xpath</td>
<td><a href="http://opensource.org/licenses/MIT">http://opensource.org/licenses/MIT</a></td>
<td>MIT</td>
</tr>
<tr>
<td>keepalive</td>
<td><a href="https://github.com/wikier/keepalive/blob/master/LICENSE">https://github.com/wikier/keepalive/blob/master/LICENSE</a></td>
<td>LGPL</td>
</tr>
<tr>
<td>LAPACK</td>
<td><a href="http://www.netlib.org/lapack/LICENSE.txt">http://www.netlib.org/lapack/LICENSE.txt</a></td>
<td>Modified BSD</td>
</tr>
<tr>
<td>lxml</td>
<td><a href="http://lxml.de/index.html#license">http://lxml.de/index.html#license</a></td>
<td>BSD</td>
</tr>
<tr>
<td>mako</td>
<td><a href="http://www.opensource.org/licenses/mit-license.php">http://www.opensource.org/licenses/mit-license.php</a></td>
<td>MIT</td>
</tr>
<tr>
<td>markupsafe</td>
<td><a href="http://www.opensource.org/licenses/bsd-license.php">http://www.opensource.org/licenses/bsd-license.php</a></td>
<td>BSD</td>
</tr>
<tr>
<td>MathJax</td>
<td><a href="http://cdn.mathjax.org/mathjax/2.0-latest/LICENSE">http://cdn.mathjax.org/mathjax/2.0-latest/LICENSE</a></td>
<td>Apache</td>
</tr>
<tr>
<td>matplotlib</td>
<td><a href="http://matplotlib.sourceforge.net/users/license.html">http://matplotlib.sourceforge.net/users/license.html</a></td>
<td>Modified BSD</td>
</tr>
<tr>
<td>MinGW</td>
<td><a href="http://www.mingw.org/license">http://www.mingw.org/license</a></td>
<td>Not distributed with software directly</td>
</tr>
<tr>
<td>networkx</td>
<td><a href="http://networkx.github.io/documentation/development/reference/legal.html">http://networkx.github.io/documentation/development/reference/legal.html</a></td>
<td>BSD</td>
</tr>
<tr>
<td>numexpr</td>
<td><a href="http://www.opensource.org/licenses/mit-license.php">http://www.opensource.org/licenses/mit-license.php</a></td>
<td>MIT</td>
</tr>
<tr>
<td>numpy</td>
<td><a href="http://numpy.scipy.org/license.html">http://numpy.scipy.org/license.html</a></td>
<td>BSD</td>
</tr>
<tr>
<td>pandas</td>
<td><a href="http://pandas.pydata.org/pandas-docs/stable/overview.html#license">http://pandas.pydata.org/pandas-docs/stable/overview.html#license</a></td>
<td>Modified BSD</td>
</tr>
<tr>
<td>patsy</td>
<td><a href="https://github.com/pydata/patsy/blob/master/LICENSE.txt">https://github.com/pydata/patsy/blob/master/LICENSE.txt</a></td>
<td>BSD</td>
</tr>
<tr>
<td>ply</td>
<td><a href="http://www.dabeaz.com/plg/README.txt">http://www.dabeaz.com/plg/README.txt</a></td>
<td>BSD</td>
</tr>
<tr>
<td>prov</td>
<td><a href="https://github.com/trungdong/prov/blob/master/LICENSE">https://github.com/trungdong/prov/blob/master/LICENSE</a></td>
<td>MIT</td>
</tr>
<tr>
<td>provpy</td>
<td><a href="http://opensource.org/licenses/bsd-2-Clause">http://opensource.org/licenses/bsd-2-Clause</a></td>
<td>BSD</td>
</tr>
<tr>
<td>pyparsing</td>
<td><a href="http://www.opensource.org/licenses/mit-license.php">http://www.opensource.org/licenses/mit-license.php</a></td>
<td>MIT</td>
</tr>
<tr>
<td>pyquery</td>
<td><a href="http://www.opensource.org/licenses/bsd-license.php">http://www.opensource.org/licenses/bsd-license.php</a></td>
<td>BSD</td>
</tr>
<tr>
<td>Python</td>
<td><a href="http://docs.python.org/license.html">http://docs.python.org/license.html</a></td>
<td>PSF</td>
</tr>
<tr>
<td>pytz</td>
<td><a href="https://pypi.python.org/pypi/pytz/">https://pypi.python.org/pypi/pytz/</a></td>
<td>MIT</td>
</tr>
<tr>
<td>rdflib</td>
<td><a href="http://www.opensource.org/licenses/bsd-license.php">http://www.opensource.org/licenses/bsd-license.php</a></td>
<td>BSD</td>
</tr>
<tr>
<td>reset-fonts-grids</td>
<td><a href="http://yuilibrary.com/license/">http://yuilibrary.com/license/</a></td>
<td>BSD</td>
</tr>
<tr>
<td>scipy</td>
<td><a href="http://www.scipy.org/License_Compatibility">http://www.scipy.org/License_Compatibility</a></td>
<td>BSD</td>
</tr>
<tr>
<td>setuptools</td>
<td><a href="http://docs.python.org/license.html/">http://docs.python.org/license.html/</a></td>
<td>PSF</td>
</tr>
<tr>
<td>six</td>
<td><a href="https://bitbucket.org/gutworth/six/src/e3da7fd96039a6ed89493f89d121c4f3797e6713/LICENSE?at=default">https://bitbucket.org/gutworth/six/src/e3da7fd96039a6ed89493f89d121c4f3797e6713/LICENSE?at=default</a></td>
<td>MIT</td>
</tr>
<tr>
<td>sparqlwrapper</td>
<td><a href="http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231">http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231</a></td>
<td>W3C</td>
</tr>
<tr>
<td>statsmodels</td>
<td><a href="https://github.com/statsmodels/statsmodels/blob/master/LICENSE.txt">https://github.com/statsmodels/statsmodels/blob/master/LICENSE.txt</a></td>
<td>Modified BSD</td>
</tr>
<tr>
<td>tinymce</td>
<td><a href="https://github.com/tinymce/tinymce/blob/master/LICENSE.TXT">https://github.com/tinymce/tinymce/blob/master/LICENSE.TXT</a></td>
<td>LGPL</td>
</tr>
<tr>
<td>weave</td>
<td><a href="http://projects.scipy.org/scipy/browser/trunk/Lib/weave/LICENSE.txt?rev=1511">http://projects.scipy.org/scipy/browser/trunk/Lib/weave/LICENSE.txt?rev=1511</a></td>
<td>BSD</td>
</tr>
</tbody>
</table>
## Index

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Location/path/reloading</th>
<th>Range of software possible</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location/path/reloading</td>
<td>10–11, 75</td>
<td></td>
</tr>
<tr>
<td>Selecting, viewing &amp; editing</td>
<td>9–10, 11–12, 18, 19, 35, 55–56, 59–60, 73</td>
<td></td>
</tr>
<tr>
<td>eStat engine</td>
<td>4, 10, 16, 19–36, 38–40, 56–64, 68–72</td>
<td></td>
</tr>
<tr>
<td>Standalone C++</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>Installing Stat-JR</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Interoperability</td>
<td>4, 6, 7, 37–55</td>
<td></td>
</tr>
<tr>
<td>AML</td>
<td>38, 54–55</td>
<td></td>
</tr>
<tr>
<td>Comparison of engines</td>
<td>67–68</td>
<td></td>
</tr>
<tr>
<td>JAGS</td>
<td>45–47</td>
<td></td>
</tr>
<tr>
<td>Location/path/reloading</td>
<td>10–11, 37</td>
<td></td>
</tr>
<tr>
<td>MLwiN</td>
<td>47–50, 72</td>
<td></td>
</tr>
<tr>
<td>OpenBUGS</td>
<td>43–45, 72</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>50–54</td>
<td></td>
</tr>
<tr>
<td>WinBUGS</td>
<td>38, 40–43</td>
<td></td>
</tr>
<tr>
<td>Starting the TREE interface</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Stat-JR Forum</td>
<td>6, 9</td>
<td></td>
</tr>
<tr>
<td>Templates</td>
<td>4, 5, 7, 12</td>
<td></td>
</tr>
<tr>
<td>Editing outputs (e.g. scripts / macros)</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>Identifying interoperability</td>
<td>10, 12</td>
<td></td>
</tr>
<tr>
<td>Input string</td>
<td>10, 15</td>
<td></td>
</tr>
<tr>
<td>Location/path/reloading</td>
<td>10–11</td>
<td></td>
</tr>
<tr>
<td>Outputs</td>
<td>16–18</td>
<td></td>
</tr>
<tr>
<td>Popout outputs (in separate tab)</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>Selecting</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Specifying inputs</td>
<td>13–16</td>
<td></td>
</tr>
<tr>
<td>Tags</td>
<td>10, 12</td>
<td></td>
</tr>
<tr>
<td>Workflows</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Location/path/reloading</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>