A Beginner’s Guide to Stat-JR
(Beta release)

Programming and Documentation by

William J. Browne
Bruce Cameron
Christopher M.J. Charlton
Danius T. Michaelides*
Richard M.A. Parker
Camille Szmaragd
Huanjia Yang*
and
Zhengzheng Zhang
Centre for Multilevel Modelling,
University of Bristol.
*Electronics and Computer Science,
University of Southampton.

December 2012
1. About Stat-JR (e-STAT) .................................................................................................................. 2
   1.1 Stat-JR: software for scaling statistical heights ........................................................................ 2
   1.2 About the Beginner’s guide ........................................................................................................ 3
2 Installing and Starting Stat-JR ........................................................................................................... 3
   2.1 Installing Stat-JR .......................................................................................................................... 3
   2.2 The use of third party software and licenses ............................................................................. 3
   2.3 Starting up webtest ...................................................................................................................... 4
3 Quick-start guide .................................................................................................................................. 6
   3.1 Quick view .................................................................................................................................... 6
   3.2 Stage 1: Selecting a template & database .................................................................................... 7
   3.3 Stage 2: Providing template-specific input .................................................................................. 8
   3.4 Stage 3: Outputting the files to run the desired execution ............................................................. 10
   3.5 Stage 4: Running the execution ................................................................................................... 11
   3.6 Stage 5: The results ...................................................................................................................... 11
4 A detailed guide with worked examples ............................................................................................ 13
   4.1 The structure and layout of the webtest interface ......................................................................... 13
   4.2 Application 1: Analysis of the tutorial dataset using the E-STAT engine .................................. 23
      4.2.1 Summarising the dataset and graphs ..................................................................................... 23
      4.2.2 Single-level Regression ........................................................................................................ 28
      4.2.3 Multiple chains .................................................................................................................... 34
      4.2.4 Adding gender to the model ................................................................................................ 35
      4.2.5 Including school effects ..................................................................................................... 38
      4.2.6 Caterpillar plot ..................................................................................................................... 40
   4.3 Interoperability – a brief introduction ......................................................................................... 41
      4.3.1 So why are we offering Interoperability? .............................................................................. 41
      4.3.2 Regression in e-STAT revisited ............................................................................................ 42
      4.3.3 Interoperability with WinBUGS ............................................................................................ 43
      4.3.4 Interoperability with OpenBUGS .......................................................................................... 46
      4.3.5 Interoperability with JAGS .................................................................................................. 48
      4.3.6 Interoperability with MLwiN ................................................................................................ 49
      4.3.7 Interoperability with R .......................................................................................................... 52
      4.3.8 Interoperability with AML ................................................................................................... 56
   4.4 Application 2: Analysis of the Bangladeshi Fertility Survey dataset ........................................ 57
      4.4.1 The Bangladeshi Fertility Survey dataset .............................................................................. 57
4.4.2 Modelling the data using logistic regression .................................................58
4.4.3 Multilevel modelling of the data .................................................................62
4.4.4 Comparison between software packages ......................................................65
4.4.5 Orthogonal parameterisation ......................................................................69
4.4.6 Predictions from the model .........................................................................72
4.5 Miscellaneous other topics e.g. Data Input/Export ..........................................75
5 References ........................................................................................................76
6 Appendix: List of Third Party Software that are used by Stat-JR ..........................77
Acknowledgements

The Stat-JR software is very much a team effort and is the result of work funded 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. 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, Harvey Goldstein, George Leckie, Rebecca Pillinger, Kelvyn Jones, 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 for their input into the software.

David De Roure, Luc Moreau, 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 for their input into the software.

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 December 2012.
1. About Stat-JR (e-STAT)

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 this has been followed by a corresponding 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 to use 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 is 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 algorithm 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.

Many of the ideas within the Stat-JR system were the brainchild of Jon Rasbash (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 three initial guides to go with the software: this short Beginner’s Guide will cover how to start up and run the software. It will provide some simple (social science) 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. It introduces the software’s webtest interface, but 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. There is also a third, E-book User’s, guide which deals with the software’s alternative E-book interface.

In this Beginner’s Guide we first describe how to install Stat-JR, and then provide a ‘Quick-start’ guide as a quick visual overview, with brief notes, of the basics of how to work with Stat-JR via webtest. There then follows more detailed sections which provide further explanation, together with point-and-click examples for you to work through.

We look at an example application taken from education research, fitting a Normal response model. 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 the other software packages that link to Stat-JR as well. We will then look briefly at a second example from demography that illustrates binomial response models.

2 Installing and Starting Stat-JR

2.1 Installing Stat-JR

Stat-JR has a dedicated website for requests for a copy of the software and which contains instructions for installation. This is currently located at the following address:

www.bristol.ac.uk/cmm/software/statjr/downloads/

2.2 The use of third party software and licenses

Stat-JR is written primarily in the Python 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 webtest

Stat-JR runs in a web browser; whilst it will work in most web browsers we currently suggest not using Internet Explorer, although it is hoped support for more browsers will be added in future. To start Stat-JR, open Windows Explorer / My Computer and go to the directory that contains the Stat-JR software, typically C:\Stat-JR. Here double-click on the file webtest.cmd; this should bring up a (Firefox) web browser.

When you double-click on webtest.cmd, this action starts 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 I have run webtest.cmd I can see the following commands:

```
C:\StatJR\SET PATH=C:\StatJR\MinGW\bin;C:\Program Files (x86)\ImageMagick-6.7.7-Q16;C:\Program Files (x86)\MiKTeX 2.9\miktex\bin;C:\Program Files\Common Files\Microsoft Shared\Windows Live\C:\Program Files (x86)\Common Files\Microsoft Shared\Windows Live;C:\windows\system32;C:\windows\system32\Wbem;C:\windows\System32\WindowsPowerShell\v1.0\;C:\Program Files (x86)\Windows Live\Shared;C:\Program Files\TortoiseSVN\bin
C:\StatJR\SET PATH=C:\StatJR\JAGS-3.1.0;\bin;C:\StatJR\MinGW\bin;C:\Program Files (x86)\ImageMagick-6.7.7-Q16;C:\Program Files (x86)\MiKTeX2.9\miktex\bin;C:\Program Files\Common Files\Microsoft Shared\Windows Live;C:\Program Files (x86)\Common Files\Microsoft Shared\Windows Live;C:\windows\system32;C:\windows\System32\Wbem;C:\windows\System32\WindowsPowerShell\v1.0\;C:\Program Files (x86)\Windows Live\Shared;C:\Program Files\TortoiseSVN\bin
C:\StatJR\cd src\apps\webtest
C:\StatJR\src\apps\webtest>start http://localhost:8081/
C:\StatJR\src\apps\webtest>..\..\..\App\python.exe webtest.py 8081
Failed to load package GenStatModel (GenStat not defined)
Failed to load package MatlabScript (Matlab not defined)
Failed to load package MinitabModel (Minitab not defined)
Failed to load package MinitabScript (Minitab not defined)
Failed to load package OctaveScript (Octave not defined)
Failed to load package Sabre (Sabre not defined)
Failed to load package SASModel (SAS not defined)
Failed to load package SASScript (SAS not defined)
Failed to load package SPSSModel (SPSS not defined)
Failed to load package SPSScript (SPSS not defined)
Trying to locate and open Firefox
could not locate runnable browser
Trying to locate and open Chrome
could not locate runnable browser
Trying to locate and open default web browser
http://0.0.0.0:8081/
```

The most important command when starting up is the line:

```
http://0.0.0.0:8081/
```
This only appears when the program has performed all its initial set-up routines. This may take a while, particularly the first time you use the program. You should then be able to view the main page of webtest in your browser (note: not all browser packages are supported: see note above); if you can’t, then try refreshing the browser window, or typing localhost:8081 into the address bar.
3 Quick-start guide

This section provides a ‘quick-start’ guide to using Stat-JR, via the webtest application; for more detail on how to operate webtest, together with worked point-and-click examples, see later sections. We’re assuming you’ve installed Stat-JR, and can see the opening page of the webtest application in your browser (see Section 2).

3.1 Quick overview

When operating webtest, you generally proceed through the following five stages:

Stage 1. On the main page you can specify 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. Once you’ve made your choices, 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 webtest, 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 eSTAT), 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 webtest, 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.

Stage 1:
- Stat-JR writes commands, etc., to perform requested function on dataset (displayed in browser window / available for download)

Stage 2:
- Stat-JR prompts user for input needed by template to perform function

Stage 3:
- Function performed

Stage 4:
- Results of function produced (displayed in browser window / available for download)

Stage 5:
- (If applicable) results outputted as dataset...

Equations
Scripts
Macros
(If applicable) results outputted as dataset...

Results
Tables
Charts

myModel<- glm(normexam~
Summary(myModel)
plot(myModel,1)

normexam<- ~ N(\mu,\sigma^2)
\mu_i = \beta_0 + \beta_1 x_i
\tau = \sigma (0.001, 0.001)
\sigma^2 = 1/\tau

myModel<- glm(normexam~
Summary(myModel)
plot(myModel,1)
Below we briefly highlight the main features, with screenshots, of each of these five stages.

**Stage 1: Selecting a template & database**

- On the opening, main page, you can specify the template you want to use by selecting it (clicking on its name in the scrollable list) and then pressing **Use**; you do the same for the dataset too. Once you’ve done so, you will see their names, and descriptions, appear at the top of the window.

- Note you can use the blue ‘cloud’ tags, as well as the template descriptions, to help you find the template you need (you can combine search terms by clicking on more than one blue tag).

- Once you’ve made your choices, press **Run** (at the top of the window).
Stage 2: Providing template-specific input

- Once you’ve pressed Run, you’ll then be asked, on the pages which follow, to supply the input required by Stat-JR to allow the template to perform the appropriate executions; these inputs vary between templates, and also within templates too, depending on your earlier choices as you progress through the screens.

- After you have made your choices on each screen, press Next.

- For multi-choice lists you can de-select variables by simply clicking on their name in the right-hand list.

- If at any point you want to go back and re-specify all your inputs, press Start again; pressing either of the Change hyperlinks, at the top of the screen, will take you back to the main page (i.e. Stage 1); you can also use the back-page button in your browser.

- When asked for the Name of the output results, this will be the name given to any outputted dataset which results (see Stage 5).
As you progress through the screens, you can see your choices reflected in the RunStatJR command, and the input string, at the bottom (see later sections).

If, at any point, you want to re-specify all your inputs, then press Start again.

Again, once you’re happy with your choices, press Next...

This is the name given to any outputted dataset (e.g. MCMC chains produced by the model run).

There may be a short delay after pressing Next following the final input, as Stat-JR then generates the code to perform your desired functions...

(We’ve skipped a screen, where we were asked about this input...)
3.3 Stage 3: Outputting the files to run the desired execution

- Once you’ve pressed Next after the final input, two panes appear at the bottom of the following screen; you can select what appears in these panes via the drop-down list at the top of each.

- Note that Stat-JR hasn’t done what you want it to do yet: it’s just producing preliminary files telling you what it’s going to do, and how it’s going to do it.

- The left-hand pane generally contains ‘inputs’, the right ‘outputs’.

- You can right-click on a pane, and choose e.g. This pane > Open frame in new tab, to view the content of a pane in its own tab.

- Pressing Run performs the executions described in the two panes.
3.4 Stage 4: Running the execution

- Once you’re pressed **Run**, the executions specified by you are performed.

- Depending on your choices, this may take anything from a second or two (e.g. to produce a simple plot, fit a model using a non-iterative method of estimation, produce summary data, etc.), to many minutes (e.g. to run MCMC chains for a large number of iterations).

- If appropriate (e.g. if the template supports inter-operability, and if you have chosen to employ it when prompted), externally-authored software packages (e.g. R, MLwiN, WinBUGS, SPSS, etc.) are opened, run, then closed, and the results are returned to Stat-JR.

- Whilst the execution runs, you may see a lot of activity in the black command window, which can help you keep a track of progress.

3.5 Stage 5: The results

- Once the executions have run, the results will appear in the right-hand pane. As before, you can choose what to view in that pane via the selection box just above it.

- Depending on the template, a range of buttons / boxes appear above the two panes allowing you to e.g. **Download** the results, **Change estimation settings** (in this example, this returns you to the screen in which you were prompted to specify the random seed, burnin, etc.), and run chains for extra iterations (via **More**, and the box to the left of it).

- If applicable, an outputted dataset now appears in the list of datasets back on the main page.
The results (e.g. plots, model estimates, etc.) are returned in the right-hand pane.

You can Download results, run for More iterations, or Change estimation settings here...

Here you can add, to an eBook: the inputs you have just entered, the details of the template and dataset you have just chosen, and the outputs you would like to be displayed...

The results (e.g. plots, model estimates, etc.) are returned in the right-hand pane.
This ends the quick start guide. In the next chapter we describe the operation of webtest in more detail, and work through examples.

4 A detailed guide with worked examples

4.1 The structure and layout of the `webtest` 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, or plotting a graph. 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.
Returning to our start-up of the software, when the line http://0.0.0.0:8081/ appears, and after refreshing the web browser, the browser window should appear as follows:

This is the main menu screen for the webtest interface to Stat-JR, and consists of the following:

- The black **title bar** running along the top contains a link to the **Settings** page; this takes the user to a page containing options to change a variety of settings (discussed further below). To the right of the title bar you will see the **Debug** button; this produces a drop-down list from which one can choose to reload the templates, datasets or packages. This allows users to upload changes to files they make outside the webtest interface, without having to start-up webtest again; e.g. a user could paste a new dataset into the datasets directory, or modify a template in the templates directory, and reload them so that they appear in their respective pull-down lists in the browser window.

- The **Configuration** pane, below the black title bar, contains information about the current set-up: i.e. the template and dataset currently selected, and their description (if available). There is a link to **View dataset** (or at least the first $n$ rows of it, as specified on the **Settings** page – see below) and a **View summary** link to see summary statistics of the dataset.
Finally, the Run button is the access point to further screens which ask the user for template- and dataset-specific inputs prior to performing the execution desired.

- The Change template pane contains a scrollable list of all the templates that the system is aware of: i.e. those which appear in the templates subdirectory of this installation of Stat-JR. This pane can be used to change the selected template via the Use button. As we anticipate there being many templates, each template has defined ‘tags’ which are terms to describe what it does. When you select a template, its tags, together with a description, appear to the right of the list. In addition, all tags, from all templates, also appear in blue above the pull-down list and can be selected and used as search criteria to produce a smaller list of specific templates (you can combine search terms by clicking on more than one tag; pressing ‘reset’ clears your choices). Beneath the scrollable list there is a link to the List of templates (described below); there is also an Upload templates button at the bottom of this pane allowing you to upload individual templates not already uploaded in the current session.

- The Change dataset pane, like the templates pane, contains a scrollable list, this time of all the datasets that the system is aware of: i.e. those which appear in the datasets subdirectory of this installation of Stat-JR. This pane can be similarly used to change the selected dataset via the Use button. There is a List of datasets link which opens a page listing the names, descriptions, and list of variables for each dataset (note, this may take some time to open the first time it is accessed in a given session). It also contains buttons at the bottom of the pane that can upload and download datasets which we will discuss in Section 4.5.

We will now demonstrate the other screens that are accessed via the main menu screen, starting with the Settings window, accessible via the black title bar. This displays a number of settings that the program uses: some of these are relatively basic, such as display preferences, and some are relatively advanced, such as optimisation, starting values and standalone code options. The Display preferences option allows you to change the number of rows which appear in the View dataset window, which we will now look at:

If you are not already on it, press the Stat-JR Demonstrator button in the black title bar at the top to access the main menu page.

Scroll down the dataset list, towards the bottom of the page, and click on rats.

Click on the Use button and the current dataset in the Configuration pane should change.

Press the View dataset link in the Configuration pane. 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 $y_8$, $y_{15}$, $y_{22}$, $y_{29}$ and $y_{36}$, respectively, and the dataset also contains a constant column, cons, and a rat identifier column, rat.

To view a summary of the dataset, click on either the first tab in the browser window containing the main menu screen.

Click on the View summary button in the Configuration pane.

Click on the second tab and the screen will look as follows:

Here we get a very short summary of the dataset, giving, for each variable, the minimum value, maximum value, mean and standard deviation. More extensive summaries are available by using specific templates to summarise datasets, as we will see later.
Let’s now look at the Template Information window:

Back on the main menu page, click on the List of all Templates link below the scrollable list of templates in the Change template pane and the following screen will appear:

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 other three columns are of more interest to advanced users, but contain a list of functions in the template code, tags that identify the template type, and the engines that are supported by the template.

Clicking on the back arrow in the browser returns us to the main menu screen.

We will next try out running a template to show the structure of the screens involved when doing so. We will use the default Regression1 template that fits a 1-level Normal response regression model.

Back on the main menu screen, click on the Run button to the right of the Configuration pane and the template-running screen will appear as follows:
This is the standard layout that you will see when you run any template. At the top of the screen are the names (and descriptions, if available) of the current template and dataset, together with the Change options which will return you to the main menu to allow you to change template and/or dataset (pressing the Stat-JR Demonstrator link will achieve the same).

In the middle of the screen is the Configuration pane; this displays the inputs required for this template, and is thus template-specific. 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 1 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. 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 (which also appears in the following screens) will clear any inputs the user has already selected and return him or her 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 are performing a regression of the initial weight \((y_8)\) on the final weight \((y_{36})\), including an intercept \(\text{cons}\).

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:

- **the number of chains**: this is the number of starting points from which we will take random draws;
- **length of the 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;
- **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;
- **thinning**: this determines how often the values are stored: i.e. store every \(n\)th 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 ‘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 that results from running the template.
You will notice, towards the bottom of the window, a box for an **Input string** to be entered, next to a **Set** button, both of which are above two rather long text strings. The input string allows the user to specify all the inputs directly, without having to point-and-click through the list as we have done. These have to be formatted in a certain way, however, as illustrated by the second (**Input String**) text string which Stat-JR has written for us as a result of our inputs. This (i.e. the string between, and including, the curly brackets: in this example, from `{"Engine":"eSTAT", to \"adaptiter\": 5000}` can be copied and pasted into the box, and the **Set** button pressed (following any edits you would like to make to the input values), in order to specify inputs directly. The first text string (labelled **Command**) can be used by the cmdtest version of Stat-JR to perform the same operations, although we will not discuss this further here.

Clicking on the **Next** button will now pre-process the template inputs, and will result in the following new panes at the bottom of the window:
The object currently specified in the left pull-down list (model.txt is selected by default here) appears in the left-hand pane below it. These objects are inputs to the estimation engine and/or third party software packages that are used by the system. For example, the file model.txt contains a description of the regression model that we wish to fit.

Similarly, the right-hand pull-down list contains a list of objects that can appear in the right-hand pane. These objects are outputs from the system and/or third party software packages that are used by the system. At this point we haven’t actually run the template, and so the outputs are pre-model run, and include computer code to actually fit the model.

Click the Run button to run the template.
After the model has run select **ModelResults** from the right-hand list pull-down list.

The screen will then look as follows:

![Image of Stat-JR Demonstrator interface]

Here we see parameter estimates, along with standard deviations (SDs), for each parameter. We will explain these further in the next section. In the middle of the screen (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), whilst the **Change Estimation settings** button will allow us to input different estimation settings while maintaining the same model settings. Finally, 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.
You’ll recall that we earlier named the output results ‘out’, so if we choose this from the right-hand pull-down list, we’ll be able to view it, as follows:

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 on the main menu (at least for the duration of our current session using the software). This means that we can string templates 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.

4.2 Application 1: Analysis of the tutorial dataset using the eSTAT engine

4.2.1 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.

Begin at the main menu (if you’re following-on from the last section, you can click on either of the Change buttons at the top of the screen to return you to the main menu, or the Stat-JR Demonstrator link in the black title bar).

Select tutorial from the dataset pull-down list, then click Use.

Click on the View summary button in the Configuration pane, then (if it doesn’t display automatically) click on the new tab which appears in the browser window to view the summary information:
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 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.

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:

From the main menu (first tab in the browser window) select Histogram from the template list, click Use, and then click Run.

Fill in the inputs as shown below and click Next and then Run.
Here you will see, in the right-hand pane, a histogram plot that shows that the response variable we will model, normexam, appears Normally-distributed.

Click on Change to return to the main menu, and this time select XYPlot from the template list, then click Use, and then Run.

Fill in the inputs as shown below and click Next and then Run.
Here we see that there appears to be a positive relationship between *normexam* and *standlrf*, 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 right-clicking on it with the mouse, and selecting **This Frame -> Display this frame in New Tab**. You can then select the new tab created to view it, as shown below:
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, normexam and standlrt, actually are:

Select AverageandCorrelation as the template, and complete the inputs as follows before clicking on Next and Run:

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:

27
Select \textit{Tabulate} as the template, and complete the inputs as follows, before clicking on \textbf{Next} and \textbf{Run}:

We have to enter a row value, and so here we have simply specified it as \texttt{cons} (a column of ones), which returns columns labelled 0.0 and 1.0 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.

\textbf{4.2.2 Single-level Regression}

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

Select \texttt{Regression1} as the template and fill it in as follows:
Here we are fitting a linear regression, and so have \textit{standlrt} as an explanatory variable, but also \textit{cons} (which is a column of 1s) 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 \textit{rats} dataset, and we will overwrite the output file \textit{out}.

Clicking on the \textbf{Next} button, and choosing \textit{equation.tex} in the right-hand list, gives the following:

So here in the left-hand pane we see the text file that represents the model we wish to fit, whilst in the right-hand pane we find a mathematical representation of the same model. Note that the file on the right is a LaTeX file that is being rendered in the browser by a piece of software called MathJaX (\textit{v1.1}, 2011), so if you are a LaTeX-user you can copy this file straight into a document.
The Regression1 template is using the built-in eSTAT MCMC-based estimation engine, so as you can see in the mathematical formulae 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 $\tau$ ($\tau=1/\sigma^2$).

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.tex in the right-hand pane, and request it to be displayed in a new tab to see the following:

The eSTAT software 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 right-hand pane and display in a separate tab.

### Results

#### Model:

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dbar</td>
<td>9763.477545</td>
</tr>
<tr>
<td>D(\theta_bar)</td>
<td>9760.509852</td>
</tr>
<tr>
<td>pD</td>
<td>2.967693</td>
</tr>
<tr>
<td>DIC</td>
<td>9766.445238</td>
</tr>
</tbody>
</table>

#### Parameters:

<table>
<thead>
<tr>
<th>parameter</th>
<th>mean</th>
<th>sd</th>
<th>ESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>tau</td>
<td>1.541481</td>
<td>0.034110</td>
<td>5188</td>
</tr>
<tr>
<td>beta_0</td>
<td>-0.001242</td>
<td>0.012550</td>
<td>4651</td>
</tr>
<tr>
<td>beta_1</td>
<td>0.595024</td>
<td>0.012665</td>
<td>5113</td>
</tr>
<tr>
<td>sigma2</td>
<td>0.649045</td>
<td>0.014389</td>
<td>5191</td>
</tr>
<tr>
<td>sigma</td>
<td>0.805584</td>
<td>0.008925</td>
<td>5191</td>
</tr>
<tr>
<td>deviance</td>
<td>9763.477545</td>
<td>2.432184</td>
<td>5250</td>
</tr>
</tbody>
</table>

Here the model results can be split into two parts: the top 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(\theta_bar)) is 9760.5 and pD is ~3 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.

The second 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 mean estimate 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. 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 this case, all parameters have ESS of > 4000, and so the chains are almost independent. We can get more information from the diagnostic plots that are available in the right-hand pane of the results window.

Return to the model run tab in the browser window, and select beta_1.png from the right-hand pull-down list and view in 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.

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 either of the Change buttons (or the Stat-JR Demonstrator link), at the top of the window, to change template / dataset, and select SummaryStats as the template, and out as the dataset. Run the SummaryStats template and select the inputs as follows before clicking on Run:

Now select table in the right-hand list 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 31 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!

### 4.2.3 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.

Click on Change (or the Stat-JR Demonstrator link) to change template / dataset and select Regression1 as the template and tutorial as the dataset.

This time fill in the screen as follows:
Click on the **Next** and **Run** buttons.

When the model has run select `beta_1.png` from the right-hand list and 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 three, 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 4.2.2, 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.

### 4.2.4 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 **Start again** on the main tab in the browser window, and fill-in the template as follows:
Click on **Next** and then **Run** to run the model.

After the model finishes running select **ModelResults** from the right-hand pane, and display in a new tab.
Results

Model:

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dbar</td>
<td>9720.963136</td>
</tr>
<tr>
<td>D(theta_bar)</td>
<td>9717.008266</td>
</tr>
<tr>
<td>pD</td>
<td>3.954870</td>
</tr>
<tr>
<td>DIC</td>
<td>9724.918007</td>
</tr>
</tbody>
</table>

Parameters:

<table>
<thead>
<tr>
<th>parameter</th>
<th>mean</th>
<th>sd</th>
<th>ESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>tau</td>
<td>1.558339</td>
<td>0.034350</td>
<td>5953</td>
</tr>
<tr>
<td>beta_0</td>
<td>-0.102739</td>
<td>0.019700</td>
<td>1559</td>
</tr>
<tr>
<td>beta_1</td>
<td>0.590609</td>
<td>0.012637</td>
<td>6097</td>
</tr>
<tr>
<td>beta_2</td>
<td>0.169718</td>
<td>0.025447</td>
<td>1578</td>
</tr>
<tr>
<td>sigma2</td>
<td>0.642021</td>
<td>0.014167</td>
<td>5960</td>
</tr>
<tr>
<td>sigma</td>
<td>0.801213</td>
<td>0.008837</td>
<td>5958</td>
</tr>
<tr>
<td>deviance</td>
<td>9720.963136</td>
<td>2.794708</td>
<td>4030</td>
</tr>
</tbody>
</table>

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 sd, 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 1500, so the model doesn’t mix as well; however, these ESS are still large enough not to required further iterations. Here is the graph for beta_2.png, displayed in a new tab:
We see reasonable mixing, and can clearly 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.

### 4.2.5 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.

Select **Change** at the top of the screen to return to the main menu.

Select **2LevelMod** as the template and stick with ‘tutorial’ for the dataset.

Set-up the inputs as shown below:
Press **Next** and then **Run** to fit the model. Note that running will 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 right-hand pane, and show the results in a separate tab.
Here we see that the DIC value for the two-level model is 9246, 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 (0.0197)</td>
<td>1559</td>
<td>-0.098 (0.0456)</td>
<td>312</td>
</tr>
<tr>
<td>beta_1</td>
<td>0.591 (0.0126)</td>
<td>6097</td>
<td>0.560 (0.0125)</td>
<td>4732</td>
</tr>
<tr>
<td>beta_2</td>
<td>0.170 (0.0254)</td>
<td>1578</td>
<td>0.171 (0.0337)</td>
<td>713</td>
</tr>
</tbody>
</table>

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

### 4.2.6 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 input of all the ‘u’s to be displayed in the plot, which can be time-consuming, so we describe a short-cut using the input box in the instructions below:

1. Click on Change at the top of the screen to get to the main menu.
2. Choose CaterpillarPlot95 as the template and out2level as the dataset.
3. Click on the Run button.
4. Type the following into the input box at the bottom:
   ```python
   {'residuals': ',','.join(['u_'+str(i) for i in range(0,65 )])}
   ```
5. Press the Set button next to the input box, choose script.py from the left pull-down list on the screen, then press the Run button.
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.

4.3 Interoperability – a brief introduction

In this section we look at interoperability with other software packages. In order to run this section you will need to have installed the other packages and told Stat-JR where they are. For more details look at http://www.bristol.ac.uk/cmm/software/statjr/downloads/additionalsoft.html

4.3.1 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 WinBUGS 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 4.4.4. 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.

### 4.3.2 Regression in eSTAT revisited

In Section 4.2 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 4.2. 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 main menu, select Regression2 as the template, and tutorial as the dataset.

Click on Run to run the template, and set-up the inputs as follows:

![Image of eSTAT configuration with response set to normal, explanatory variables set to cons, cblind, and csmoke, estimation engine set to eSTAT, number of chains set to 3, random seed set to 1, length of burnin set to 500, number of iterations set to 2000, thinning set to 1, use default algorithm settings set to Yes, use default starting values set to Yes, name of output results set to outputstat.](image)

Click on Next and Run to fit the model.

Select ModelResults from the right-hand list, and show this output in a new tab which should look as follows:
These results are identical to those we obtained using Regression1 earlier, although we only looked at the plot for $\beta_1$ in Section 4.2.3. 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.

### 4.3.3 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 the **Change estimation settings** button and set up the template inputs as follows:
When we press **Next** the Stat-JR software will construct all the files required to run WinBUGS:

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

define: stdnorm = normdist(0,1)
mux[i] = cons * (bet1 + stdnorm[s] * bet2)

# fitting
bet1 = fitline(bet1, bet2)
bet1 = fitline(bet1, bet2)
sigma = fitline(sigma, sigma)
```

Here we see the model defined in the WinBUGS model specification language in the left-hand 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 left-hand list and displaying in a new tab gives the following:
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 template pane 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 right-pane in its own tab we will see the following:

**Results**

**Model:**

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dbar_normexam</td>
<td>9763.500000</td>
</tr>
<tr>
<td>Dhat_normexam</td>
<td>9760.510000</td>
</tr>
<tr>
<td>pD_normexam</td>
<td>2.986000</td>
</tr>
<tr>
<td>DIC_normexam</td>
<td>9766.480000</td>
</tr>
<tr>
<td>Dbar_total</td>
<td>9763.500000</td>
</tr>
<tr>
<td>Dhat_total</td>
<td>9760.510000</td>
</tr>
<tr>
<td>pD_total</td>
<td>2.986000</td>
</tr>
<tr>
<td>DIC_total</td>
<td>9766.480000</td>
</tr>
</tbody>
</table>

**Parameters:**

<table>
<thead>
<tr>
<th>parameter</th>
<th>mean</th>
<th>sd</th>
<th>ESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>tau</td>
<td>1.541465</td>
<td>0.034078</td>
<td>5735</td>
</tr>
<tr>
<td>deviance</td>
<td>9763.501000</td>
<td>2.464819</td>
<td>6143</td>
</tr>
<tr>
<td>beta1</td>
<td>0.594717</td>
<td>0.012705</td>
<td>6647</td>
</tr>
<tr>
<td>beta0</td>
<td>-0.0001044</td>
<td>0.012633</td>
<td>5728</td>
</tr>
<tr>
<td>sigma</td>
<td>0.889588</td>
<td>0.008908</td>
<td>5747</td>
</tr>
<tr>
<td>sigma2</td>
<td>0.649052</td>
<td>0.014358</td>
<td>5750</td>
</tr>
</tbody>
</table>

These estimates, as one might expect, are very close to those from eSTAT, and again all ESS values are greater than 5,000. We can also look at the log file from WinBUGS:
Return to the template tab and choose log.txt in the right-hand list and inits2.txt in the left-hand 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 (the left pane shows the value for run 2), 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.

4.3.4 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 Change Estimation Settings button, set up the template as follows, and then click on Next:
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 (you can access this from the left-hand list):

```r
modelDisplay('log')
modelSet('c:/u/richard/appdata/local/tmp/tom3\')
modelCheck('model.txt')
modelData('data.txt')
modelCompile(3)
modelInit('initial.txt', 1)
modelInit('initial2.txt', 2)
modelInit('initial3.txt', 3)
modelSetDir()
modelSetB(0)
modelUpdate(500, 1)
sampleSet('tau')
sampleSet('variance')
sampleSet('beta1')
sampleSet('beta2')
sampleSet('beta3')
sampleSet('shapes')
sampleSet('shapes2')
discard()
modelUpdate(2000, 1)
sampleCoda('**', 'results')
sampleCoda('**')
discard()
sampleCoda('**')
modelDisplay('log.cdf')
modelDisplay('log.txt')
modelExternalize('modelstate.bug')
modelQuit('yes')
```

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:
Results

Model:

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dbar_normex</td>
<td>9764 0.000000</td>
</tr>
<tr>
<td>Dhat_normex</td>
<td>9761 0.000000</td>
</tr>
<tr>
<td>pD_normex</td>
<td>3.071000</td>
</tr>
<tr>
<td>DIC_normex</td>
<td>9767 0.000000</td>
</tr>
<tr>
<td>Dbar_total</td>
<td>9764 0.000000</td>
</tr>
<tr>
<td>Dhat_total</td>
<td>9761 0.000000</td>
</tr>
<tr>
<td>pD_total</td>
<td>3.071000</td>
</tr>
<tr>
<td>DIC_total</td>
<td>9767 0.000000</td>
</tr>
</tbody>
</table>

Parameters:

<table>
<thead>
<tr>
<th>parameter</th>
<th>mean</th>
<th>sd</th>
<th>ESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>tau</td>
<td>1.542130</td>
<td>0.035107</td>
<td>5045</td>
</tr>
<tr>
<td>deviance</td>
<td>9763.582000</td>
<td>2.464131</td>
<td>5787</td>
</tr>
<tr>
<td>beta</td>
<td>0.595048</td>
<td>0.012867</td>
<td>5853</td>
</tr>
<tr>
<td>beta0</td>
<td>-0.001295</td>
<td>0.012631</td>
<td>6020</td>
</tr>
<tr>
<td>sigma</td>
<td>0.805423</td>
<td>0.001678</td>
<td>5941</td>
</tr>
<tr>
<td>sigma2</td>
<td>0.648788</td>
<td>0.014768</td>
<td>5947</td>
</tr>
</tbody>
</table>

Again, these results are very similar in terms of parameter estimates and ESS values to the other software packages.

4.3.5 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 Change Estimation Settings button and set-up the template as follows, before clicking on Next:

![Image of JAGS configuration panel]
This will set-up the files required for JAGS; for example, here you can see the script file (script.txt) and initial value files (accessible via the left-hand list; here we show inits2.txt) which show some differences to those for WinBUGS:

```
load dir
data in 'model.dat'
data in 'data.txt'
compile, nchains(1)
parameters in 'inits.txt', chain(1)
parameters in 'inits2.txt', chain(2)
parameters in 'inits3.txt', chain(3)
initialise
update 500
monitor seq, thin(1)
monitor deviance, thin(1)
monitor beta0, thin(1)
monitor beta1, thin(1)
monitor sigma, thin(1)
monitor sigma2, thin(1)
'step' <- 0.038e00327258
'beta0' <- 0.606649000004
'beta1' <- -9.9971250365
'.RNG.name' "base::Mersenne-Twister"
'.RNG.seed' <- 2
update 2000
code (*, stem('results'))
parameters in 'chainstats.txt', chain(1)
parameters in 'chainstats2.txt', chain(2)
parameters in 'chainstats3.txt', chain(3)
samples to 'samples.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:

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dbar</td>
<td>9763.541142</td>
</tr>
<tr>
<td>pD</td>
<td>3.018875</td>
</tr>
<tr>
<td>DIC</td>
<td>9766.560017</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>mean</th>
<th>sd</th>
<th>ESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>tau</td>
<td>1.540140</td>
<td>0.034196</td>
<td>6314</td>
</tr>
<tr>
<td>pD</td>
<td>3.018875</td>
<td>1.786616</td>
<td>5559</td>
</tr>
<tr>
<td>deviance</td>
<td>9763.541142</td>
<td>2.448828</td>
<td>6227</td>
</tr>
<tr>
<td>beta1</td>
<td>0.595978</td>
<td>0.012762</td>
<td>5736</td>
</tr>
<tr>
<td>beta0</td>
<td>-0.001162</td>
<td>0.013803</td>
<td>5879</td>
</tr>
<tr>
<td>sigma</td>
<td>0.865933</td>
<td>0.008849</td>
<td>6293</td>
</tr>
<tr>
<td>sigma2</td>
<td>0.644612</td>
<td>0.014428</td>
<td>6287</td>
</tr>
</tbody>
</table>

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.

### 4.3.6 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 **Change Estimation Settings** button and set-up the template as follows before clicking on **Next**:

![Image of MLwiN configuration settings]

---

**Note:**

- $\text{normexm} \sim \mathcal{N}(\mu, \sigma^2)$
- $\mu = \beta_0 \text{const} + \beta_1 \text{standrt}$
- $\beta_0 \sim 1$
- $\beta_1 \sim 1$
- $\tau \sim \Gamma(0.001, 0.001)$
- $\sigma^2 = 1/\tau$
You can see, in the left-hand pane, the script file (for chain1) that is to be used in MLwiN. Also available in that pane is the dataset (.dta format) that is used by MLwiN.

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:

### Results

#### Model:

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_{bar}$</td>
<td>9763.521484</td>
</tr>
<tr>
<td>$D(\text{thetabars})$</td>
<td>9760.513021</td>
</tr>
<tr>
<td>pD</td>
<td>3.008191</td>
</tr>
<tr>
<td>DIC</td>
<td>9766.520622</td>
</tr>
</tbody>
</table>

#### Parameters:

<table>
<thead>
<tr>
<th>parameter</th>
<th>mean</th>
<th>sd</th>
<th>ESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>deviance</td>
<td>9763.521355</td>
<td>2.452417</td>
<td>5564</td>
</tr>
<tr>
<td>beta2_cons</td>
<td>-0.001064</td>
<td>0.012511</td>
<td>5802</td>
</tr>
<tr>
<td>beta3_standlrt</td>
<td>0.595001</td>
<td>0.012756</td>
<td>5937</td>
</tr>
<tr>
<td>sigma1_I_var(levres)</td>
<td>0.649179</td>
<td>0.014621</td>
<td>6236</td>
</tr>
</tbody>
</table>

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 ($\text{beta3\_standlrt}$), 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 on the **Change Estimation Settings** button and set-up the template as follows, before clicking on **Next**:

![Stat-JR Demonstrator](image)

Again a macro file appears in the left-hand pane, and this time **pressing Run** will give the following:

**Results**

**Model:**

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>converged</td>
<td>1.0</td>
</tr>
<tr>
<td>iterations</td>
<td>2.0</td>
</tr>
</tbody>
</table>

$2^\text{LogLikelihood} = 9760.51$

**Parameters:**

<table>
<thead>
<tr>
<th>parameter</th>
<th>mean</th>
<th>se</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta2_cons</td>
<td>-0.00119112</td>
<td>0.0126392</td>
</tr>
<tr>
<td>beta3_standlrt</td>
<td>0.595507</td>
<td>0.012727</td>
</tr>
<tr>
<td>sigma1_var(levres)</td>
<td>0.648419</td>
<td>0.0143933</td>
</tr>
</tbody>
</table>

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.

### 4.3.7 Interoperability with R

R (R Development Core Team, 2011) 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. We have thus far implemented interoperability features with R for several of these R functions; for example, for the template ‘Regression2’, we have implemented two R engines: *MCMCglmm*, which is MCMC-based, and *MASS*, which is a standard regression modelling function. We will firstly demonstrate *MCMCglmm*.

To run MCMC in R, via Stat-JR, click on the **Change Estimation Settings** button and set-up the template as follows, and click on **Next**:
After pressing **Next**, if we look at the script file, *script.R*, which we can select from the left-hand pane, we see the following:

```r
library(foreign)
mydata=read.dta("datafile.dta")
PACKages=as.character(data.frame(installed.packages())$Package)
sf (if (test):  
    install.packages("MCMCglmm",repos="http://cran.r-project.org")  
}
library(MCMCglmm)

myprior<-list (B=list(V=diag(2),mu=rep(0,2)),S=list(V=1,mu=0.0002))
myModel<-MCMCglmm(nomexam~cons+standint-1,family="gaussian",prior=myprior,burnin=500,nitt=5000,thin=1,data=mydata)
summary(myModel)
myModelOut<-coda(myModel$Sd1,myModel$VCV)
myModelOut<-summary(myModelOut)
summary(myModelOut)

png("DiagPlots1.png",width=333,height=500)
plot(myModelOut[,1:3(0:1)],ylab="units")  
dev.off()

colnames (myModelOut)[which(colnames(myModelOut) == "units")]<-"sigma2"
write.dta(data.frame(myModelOut),file="chains.dta")
```

*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 MASS package we can click on Change Estimation Settings and set-up the template as follows, before clicking on Next:

Clicking on Run will this time run the MASS package and give 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.
Before finishing with R we will also demonstrate a non-model template developed with R called *PlotsViaR* that gives the Stat-JR user access to R’s lattice graphics package through the Stat-JR interface.

Click on **Change** to return to the main menu. Note that the blue tag cloud is useful with interoperability as it can be used to show which templates offer interoperability with a particular package.

Click on **Plots** and also **R** 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 **Set** and **Run**.

Set up the template inputs as shown below:

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Type of plot:</th>
<th>densityplot</th>
</tr>
</thead>
<tbody>
<tr>
<td>X values:</td>
<td>nomexam</td>
<td></td>
</tr>
<tr>
<td>Do you want a (within-plot) grouping variable:</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Grouping variable:</td>
<td>girl</td>
<td></td>
</tr>
<tr>
<td>How many panelling variables do you want:</td>
<td>one</td>
<td></td>
</tr>
<tr>
<td>Which panelling variable would you like to use:</td>
<td>schgend</td>
<td></td>
</tr>
<tr>
<td>Do you want the variable name included in panel bar, or just the level:</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

[Diagram of a scatter plot with residuals vs fitted values]

Before finishing with R we will also demonstrate a non-model template developed with R called *PlotsViaR* that gives the Stat-JR user access to R’s lattice graphics package through the Stat-JR interface.

Click on **Change** to return to the main menu. Note that the blue tag cloud is useful with interoperability as it can be used to show which templates offer interoperability with a particular package.

Click on **Plots** and also **R** 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 **Set** and **Run**.

Set up the template inputs as shown below:

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Type of plot:</th>
<th>densityplot</th>
</tr>
</thead>
<tbody>
<tr>
<td>X values:</td>
<td>nomexam</td>
<td></td>
</tr>
<tr>
<td>Do you want a (within-plot) grouping variable:</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Grouping variable:</td>
<td>girl</td>
<td></td>
</tr>
<tr>
<td>How many panelling variables do you want:</td>
<td>one</td>
<td></td>
</tr>
<tr>
<td>Which panelling variable would you like to use:</td>
<td>schgend</td>
<td></td>
</tr>
<tr>
<td>Do you want the variable name included in panel bar, or just the level:</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

[Diagram of a scatter plot with residuals vs fitted values]
These options will display kernel plots for the exam scores of pupils grouped by gender, with separate (panelled) plots for each school gender type. We can now press Run and show the plot (Plot1.png) in a separate tab:

Here (by coincidence) we have blue for boys and pink for girls!

4.3.8 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. We have so far written special templates for AML, so we need to do the following:

Click on Change to return to the main menu.

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

Click on Run to run the template and fill in the inputs as follows, and press Next:
Now click on Run to run the model in AML:

Here we see the model results to the right, and the AML input dataset (amlfit.raw) to the left. AML requires two other data files as input: you can see these via the left-hand list (amlfit.aml and amlfit.r2a). To the right are 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.

We also have interoperability support for a variety of other packages, including GenStat, MATLAB, Minitab, Octave, Sabre, SAS, SPSS, Stan (via RStan) and Stata. These packages are either not installed on the machine I am currently using, or are not supported by the Regression2 template I have been demonstrating, and so Stat-JR realises this and does not offer them to me.

4.4 Application 2: Analysis of the Bangladeshi Fertility Survey dataset

4.4.1 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 bang1 from the dataset list and click on Use. Click on View to view the data as follows:
Here we see records for the first 20 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.

4.4.2 Modelling the data using logistic regression

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

Return to the main pane and Select 1LevelMod from the template list and click on Use. Click on the Run button and then set-up the model with inputs as below.
Clicking on Next and choosing equation.tex in the right-hand pane we see the following:

Here we have WinBUGS-like code for a simple linear regression in the left-hand pane and the model, in LaTeX, in the right-hand pane.

Now choosing algorithm.tex from the right-hand pane, and placing it in its own tab in the browser window, gives the following:
Here we see 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 4.4.4 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 right-hand pane, and look at it in a new tab, we get the following:

**Results**

**Model:**

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dbar</td>
<td>3277.490161</td>
</tr>
<tr>
<td>D(thetabar)</td>
<td>3275.513867</td>
</tr>
<tr>
<td>pD</td>
<td>1.976294</td>
</tr>
<tr>
<td>DIC</td>
<td>3279.466456</td>
</tr>
</tbody>
</table>

**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>-7.810815</td>
<td>0.035584</td>
<td>1484</td>
</tr>
<tr>
<td>beta_1</td>
<td>0.003696</td>
<td>0.004009</td>
<td>1406</td>
</tr>
<tr>
<td>deviance</td>
<td>3277.490161</td>
<td>1.965078</td>
<td>1367</td>
</tr>
</tbody>
</table>

Perhaps disappointingly, age doesn’t appear to have a significant effect (its estimate (0.0037) is similar in magnitude to its standard error (0.004)). To see this more clearly we can look at the graph beta_1.png 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 age\(^2\) which we can easily do via the Calculate template.

Click on either of the Change buttons at the top to get back to the main screen.
Choose Calculate from the template list and click on Use and then Run.
Fill in the template as follows:
Here we are going to overwrite the existing dataset (at least in temporary memory) with a version in which we have appended an additional column to it. Clicking on Run and scrolling the right-hand pane to the extreme right 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.

### 4.4.3 Multilevel modelling of the data

We now need to find a template that will fit a 2-level logistic regression model to our dataset. In the earlier sections we looked at the template 2LevelMod but here we have the additional complexity of a categorical predictor and so will introduce a further template, 2LevelCat, which can deal with this.

Click on either of the **Change** buttons at the top to return to the main screen. Select 2LevelCat from the template list and click on **Use** and **Run** 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.

Clicking on **Next** will run the algebra system and set up code to fit the model. If we select `tau_u.xml` in the right-hand list we will see the following:
Here we see the more complicated model code for this 2-level model in the left-hand pane. In the right-hand pane we see the algorithm step for the parameter $\tau_u$. 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 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 to see the parameter estimates:
Results

Model:

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dbar</td>
<td>2351.233841</td>
</tr>
<tr>
<td>D(theta bar)</td>
<td>2308.839901</td>
</tr>
<tr>
<td>pD</td>
<td>42.393940</td>
</tr>
<tr>
<td>DIC</td>
<td>2393.627780</td>
</tr>
</tbody>
</table>

Parameters:

<table>
<thead>
<tr>
<th>parameter</th>
<th>mean</th>
<th>sd</th>
<th>ESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>sigma2_u</td>
<td>0.316172</td>
<td>0.105622</td>
<td>707</td>
</tr>
<tr>
<td>beta_0</td>
<td>-0.791214</td>
<td>0.183200</td>
<td>108</td>
</tr>
<tr>
<td>beta_1</td>
<td>0.005838</td>
<td>0.009445</td>
<td>221</td>
</tr>
<tr>
<td>beta_2</td>
<td>-0.004809</td>
<td>0.000748</td>
<td>310</td>
</tr>
<tr>
<td>beta_3</td>
<td>0.773955</td>
<td>0.161948</td>
<td>302</td>
</tr>
<tr>
<td>beta_4</td>
<td>0.829740</td>
<td>0.193973</td>
<td>203</td>
</tr>
<tr>
<td>beta_5</td>
<td>0.837048</td>
<td>0.192435</td>
<td>144</td>
</tr>
<tr>
<td>tau_u</td>
<td>3.518587</td>
<td>1.191705</td>
<td>706</td>
</tr>
<tr>
<td>deviance</td>
<td>2351.233841</td>
<td>11.484409</td>
<td>1257</td>
</tr>
</tbody>
</table>

Here we see that $\beta_2$ is significant and negative (and larger than $\beta_1$) 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 $\beta_3$-$\beta_5$ are all significant, and positive (and of similar magnitude), which suggests that women with children are more likely to use contraceptives than those without. 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.

4.4.4 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, 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 **Change Estimation Settings** and set-up the model as follows:

Clicking on **Next** and **Run** will (after 4 min 20s on my machine) give the following, having selected **ModelResults** in the right-hand pane, and opening it in a new tab:
Results

Model:

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dbar_use</td>
<td>2351.000000</td>
</tr>
<tr>
<td>Dhat_use</td>
<td>2309.000000</td>
</tr>
<tr>
<td>pD_use</td>
<td>42.670000</td>
</tr>
<tr>
<td>DIC_use</td>
<td>2394.000000</td>
</tr>
<tr>
<td>Dbar_total</td>
<td>2351.000000</td>
</tr>
<tr>
<td>Dhat_total</td>
<td>2309.000000</td>
</tr>
<tr>
<td>pD_total</td>
<td>42.670000</td>
</tr>
<tr>
<td>DIC_total</td>
<td>2394.000000</td>
</tr>
</tbody>
</table>

Parameters:

<table>
<thead>
<tr>
<th>parameter</th>
<th>mean</th>
<th>sd</th>
<th>ESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>sigma2_u</td>
<td>0.317735</td>
<td>0.101043</td>
<td>1771</td>
</tr>
<tr>
<td>deviance</td>
<td>2255.185333</td>
<td>11.518498</td>
<td>4456</td>
</tr>
<tr>
<td>beta5</td>
<td>0.827237</td>
<td>0.183636</td>
<td>4446</td>
</tr>
<tr>
<td>beta4</td>
<td>0.825589</td>
<td>0.186231</td>
<td>5205</td>
</tr>
<tr>
<td>beta3</td>
<td>0.781904</td>
<td>0.162415</td>
<td>5291</td>
</tr>
<tr>
<td>beta2</td>
<td>-0.004813</td>
<td>0.000728</td>
<td>3037</td>
</tr>
<tr>
<td>beta1</td>
<td>0.006576</td>
<td>0.009101</td>
<td>5055</td>
</tr>
<tr>
<td>beta0</td>
<td>-0.790131</td>
<td>0.172829</td>
<td>2607</td>
</tr>
<tr>
<td>tau_u</td>
<td>3.475831</td>
<td>1.128493</td>
<td>1616</td>
</tr>
</tbody>
</table>

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. We could also fit the model using the mcmclmm 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 doesn’t make sense in this case.

The table overleaf details the results:
<table>
<thead>
<tr>
<th>Parameter</th>
<th>e-STAT</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>-0.791(0.183)</td>
<td>-0.789(0.170)</td>
<td>-0.790(0.173)</td>
<td>-0.792(0.180)</td>
<td>-0.833(0.175)</td>
<td>-0.784(0.172)</td>
</tr>
<tr>
<td>Beta0 ESS</td>
<td>108</td>
<td>396</td>
<td>2607</td>
<td>248</td>
<td>94</td>
<td>1032</td>
</tr>
<tr>
<td>Beta1</td>
<td>0.0058(0.0094)</td>
<td>0.0068(0.0090)</td>
<td>0.0066(0.0091)</td>
<td>0.0062(0.0092)</td>
<td>0.0051(0.0091)</td>
<td>0.0064(0.0092)</td>
</tr>
<tr>
<td>Beta1 ESS</td>
<td>221</td>
<td>953</td>
<td>5055</td>
<td>582</td>
<td>244</td>
<td>1616</td>
</tr>
<tr>
<td>Beta2</td>
<td>-0.0048(0.00075)</td>
<td>-0.0048(0.00072)</td>
<td>-0.0048(0.00073)</td>
<td>-0.0048(0.00074)</td>
<td>-0.0047(0.00073)</td>
<td>-0.0048(0.00073)</td>
</tr>
<tr>
<td>Beta2 ESS</td>
<td>310</td>
<td>1288</td>
<td>5037</td>
<td>779</td>
<td>332</td>
<td>1726</td>
</tr>
<tr>
<td>Beta3</td>
<td>0.774(0.162)</td>
<td>0.779(0.160)</td>
<td>0.782(0.162)</td>
<td>0.784(0.167)</td>
<td>0.797(0.163)</td>
<td>0.777(0.163)</td>
</tr>
<tr>
<td>Beta3 ESS</td>
<td>302</td>
<td>1117</td>
<td>5291</td>
<td>588</td>
<td>250</td>
<td>1656</td>
</tr>
<tr>
<td>Beta4</td>
<td>0.830(0.194)</td>
<td>0.822(0.181)</td>
<td>0.826(0.186)</td>
<td>0.833(0.190)</td>
<td>0.855(0.187)</td>
<td>0.828(0.184)</td>
</tr>
<tr>
<td>Beta4 ESS</td>
<td>203</td>
<td>780</td>
<td>5205</td>
<td>456</td>
<td>192</td>
<td>1727</td>
</tr>
<tr>
<td>Beta5</td>
<td>0.837(0.192)</td>
<td>0.824(0.180)</td>
<td>0.827(0.184)</td>
<td>0.834(0.187)</td>
<td>0.861(0.183)</td>
<td>0.829(0.188)</td>
</tr>
<tr>
<td>Beta5 ESS</td>
<td>144</td>
<td>547</td>
<td>4446</td>
<td>327</td>
<td>128</td>
<td>1532</td>
</tr>
<tr>
<td>Sigma2u</td>
<td>0.316(0.106)</td>
<td>0.318(0.100)</td>
<td>0.317(0.101)</td>
<td>0.317(0.102)</td>
<td>0.317(0.099)</td>
<td>0.319(0.101)</td>
</tr>
<tr>
<td>Sigma2u ESS</td>
<td>707</td>
<td>1764</td>
<td>1771</td>
<td>1225</td>
<td>770</td>
<td>782</td>
</tr>
<tr>
<td>Pd</td>
<td>42.39</td>
<td>42.44</td>
<td>42.67</td>
<td>42.15</td>
<td>42.49</td>
<td>42.53</td>
</tr>
<tr>
<td>DIC</td>
<td>2393.63</td>
<td>2393.35</td>
<td>2394.0</td>
<td>2393.43</td>
<td>2393.65</td>
<td>2393.60</td>
</tr>
<tr>
<td>Time (s)</td>
<td>129</td>
<td>350</td>
<td>260</td>
<td>421</td>
<td>35</td>
<td>150</td>
</tr>
</tbody>
</table>

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, for this example, JAGS is the slowest and its performance is disappointingly in the middle with regard ESS; however, there have been many comparisons between JAGS and WinBUGS for different models, and which method is better varies from model to
model, so we shouldn’t dismiss it based on just this one example. The final column shows another eSTAT method which we will discuss next.

### 4.4.5 Orthogonal parameterisation.

The reason eSTAT (and MLwiN) perform badly 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 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 use a template that doesn’t offer the option of categorical predictors, and so we firstly need to manually construct indicator variables for the various numbers of (living) children using the MakeCats template.

Click on Change and then on the main menu select MakeCats from the template list. Click on Use and then Run and fill in the template inputs as follows:

Here we are intending to add to the existing dataset and so our output dataset is given the same name: bang1. Clicking on Next and Run, and then scrolling to the right in the right-hand pane, gives the following:
Here we see that three indicator predictor variables, \( lc_1 \) (1 child), \( lc_2 \) (2 children) and \( lc_3 \) (3 or more children) have been constructed. When we earlier used the 2LevelCat template and specified, when prompted, that \( lc \) was a categorical variable, essentially the same operations to construct these variables were done in the background.

We will 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 Change, and then on the main menu select NLevelOrthogParamRS from the template list. Click on Use, and then Run, and fill in the template inputs as follows:
Clicking on Next and selecting `equation.tex` in the right-hand pane (we’ve opened it in a new tab) will show the following:

\[
\begin{align*}
\text{use}, & \sim \text{Binomial}(\text{cons}, n_i) \\
\logit(n_i) & = \text{constant} + \beta_0 \text{orthage}, + \beta_1 \text{orthage}_2, + \beta_2 \text{orthage}_3, + \beta_3 \text{orthage}_4, + \beta_4 \text{orthage}_5, + \beta_5 \text{orthage}_6, + \epsilon_i \\
\beta_\text{orthage} & = N(0, \sigma^2_\beta) \\
\epsilon_i & \sim \text{N}(0, \sigma^2_\epsilon) \\
\sigma^2_\beta & = 1/\nu_\beta \\
\sigma^2_\epsilon & = 1 \\
\end{align*}
\]
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 2 minutes 30s on my machine), after which selecting ModelResults from the right-hand pane, and opening it in a new tab, gives the following:

Results

Model:

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_{bar}$</td>
<td>2351.069682</td>
</tr>
<tr>
<td>$D(\theta_{bar})$</td>
<td>2308.536053</td>
</tr>
<tr>
<td>$pD$</td>
<td>42.533628</td>
</tr>
<tr>
<td>DIC</td>
<td>2393.60310</td>
</tr>
</tbody>
</table>

Parameters:

<table>
<thead>
<tr>
<th>parameter</th>
<th>mean</th>
<th>sd</th>
<th>ESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma^2_{u_0}$</td>
<td>0.318822</td>
<td>0.101438</td>
<td>782</td>
</tr>
<tr>
<td>deviance</td>
<td>2351.069682</td>
<td>11.205091</td>
<td>1249</td>
</tr>
<tr>
<td>$\beta_{ort_0}$</td>
<td>-0.582025</td>
<td>0.093676</td>
<td>286</td>
</tr>
<tr>
<td>$\beta_{ort_1}$</td>
<td>0.009076</td>
<td>0.005818</td>
<td>1754</td>
</tr>
<tr>
<td>$\beta_{ort_2}$</td>
<td>-0.006331</td>
<td>0.000678</td>
<td>1643</td>
</tr>
<tr>
<td>$\beta_{ort_3}$</td>
<td>0.320895</td>
<td>0.130034</td>
<td>1699</td>
</tr>
<tr>
<td>$\beta_{ort_4}$</td>
<td>0.308305</td>
<td>0.141074</td>
<td>1752</td>
</tr>
<tr>
<td>$\beta_{ort_5}$</td>
<td>0.828640</td>
<td>0.187847</td>
<td>1532</td>
</tr>
<tr>
<td>$\beta_0$</td>
<td>-0.784175</td>
<td>0.172235</td>
<td>1032</td>
</tr>
<tr>
<td>$\beta_1$</td>
<td>0.006151</td>
<td>0.009176</td>
<td>1616</td>
</tr>
<tr>
<td>$\beta_2$</td>
<td>-0.004801</td>
<td>0.000729</td>
<td>1726</td>
</tr>
<tr>
<td>$\beta_3$</td>
<td>0.776893</td>
<td>0.162634</td>
<td>1656</td>
</tr>
<tr>
<td>$\beta_4$</td>
<td>0.827901</td>
<td>0.184443</td>
<td>1727</td>
</tr>
<tr>
<td>$\beta_5$</td>
<td>0.828650</td>
<td>0.187837</td>
<td>1532</td>
</tr>
<tr>
<td>$\tau_{u_0}$</td>
<td>3.460536</td>
<td>1.131051</td>
<td>782</td>
</tr>
</tbody>
</table>

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 more of an overhead incurred when performing the orthogonalising algorithm, however it is still 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 OpenBUGS still having a slight edge. 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.

4.4.6 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. We will do this manually via the *Calculate* template.

**Press Change** to get to the main menu, and select *Calculate* from the template list. Click on **Use** and **Run** to run the template, and then set-up the screen as follows (here we are using estimates from the fit in OpenBUGS):

Clicking **Next**, then **Run**, will give predictions on the logit scale for women with no children. To translate these into probabilities we need to perform the antilogit transformation, so **click Start again** and set-up as follows:

Here we have had to express the anti-logit transformation in its functional form and we will store the predicted probabilities for the childless women in *prob0*. To do this **click on Next**, then **Run**. We will next repeat the process for the women with 1 child, so **click Start again** and set-up the screen as follows:
Here again we need to convert to probabilities, so **click Next, then Run**, and once running has completed **click Start Again**, and then finally set-up as follows:

**Click Next, then Run, to produce our requested column of probabilities.**

We can now plot these probabilities on one graph against *age*; to do this we will use the **XYplot** template.

**Click on Change and select XYPlot as the template from the list.**

**Click on Use and then Run and fill in the inputs as follows:**
To view the graph we can **click on Run** and then display `graphxy.png` in its own tab as follows:

![Graph](image)

Here we see the two graphs 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 chapter 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.

### 4.5 Miscellaneous other topics e.g. Data Input/Export

It is very easy to import data into Stat-JR. Data has to be in the .dta dataset format that is used by Stata (and supported by MLwiN). Any .dta files that are in the “datasets” subdirectory of Stat-JR will automatically be loaded in the start-up of the software. As discussed earlier, if you wish to add a dataset to the directory after start-up you can press **Debug > Reload datasets** from the main menu page to reload the whole ‘datasets’ folder again, or you can upload an individual file by finding it via **Browse...** (beneath the scrollable list of datasets on the main menu page) and then click on the **Upload** button. Similarly, any datasets constructed while running the software, and stored in temporary memory, can be downloaded for storage elsewhere by selecting them as the current dataset and using the **Download** button.
References


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

<table>
<thead>
<tr>
<th>Package</th>
<th>Link</th>
<th>Licence terms</th>
</tr>
</thead>
<tbody>
<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>Bootstrap</td>
<td><a href="https://github.com/twitter/bootstrap/blob/master/LICENSE">https://github.com/twitter/bootstrap/blob/master/LICENSE</a></td>
<td>Apache</td>
</tr>
<tr>
<td>Jquery</td>
<td><a href="http://jquery.org/license">http://jquery.org/license</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>Dual MIT/GPL</td>
</tr>
<tr>
<td>Jquery-ba-haschange</td>
<td><a href="http://benalman.com/about/license/">http://benalman.com/about/license/</a></td>
<td>Dual MIT/GPL</td>
</tr>
<tr>
<td>Jquery-treeview</td>
<td><a href="https://github.com/jzaeferr/jquery-treeview">https://github.com/jzaeferr/jquery-treeview</a></td>
<td>Dual MIT/GPL</td>
</tr>
<tr>
<td>Py2exe</td>
<td><a href="http://www.py2exe.org/LICENSE.txt">http://www.py2exe.org/LICENSE.txt</a></td>
<td>MIT</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>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>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>Reset-fonts-grids</td>
<td><a href="http://yuilibrary.com/license/">http://yuilibrary.com/license/</a></td>
<td>BSD</td>
</tr>
<tr>
<td>Jquery-progressbar</td>
<td><a href="http://t.wits.sg/jquery-progress-bar/">http://t.wits.sg/jquery-progress-bar/</a></td>
<td>BSD</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>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>Rdfextras</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>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>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>Lxml</td>
<td><a href="http://lxml.de/index.html#li">http://lxml.de/index.html#li</a></td>
<td>BSD</td>
</tr>
<tr>
<td>Package</td>
<td>License URL</td>
<td>License Type</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>numpy</td>
<td><a href="http://numpy.scipy.org/license.html#license">http://numpy.scipy.org/license.html#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>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>
<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>jquery-tools</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MinGW</td>
<td><a href="http://www.mingw.org/license">http://www.mingw.org/license</a></td>
<td>Not distributed</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>BLAS</td>
<td><a href="http://www.netlib.org/blas/faq.html#2">http://www.netlib.org/blas/faq.html#2</a></td>
<td>Not distributed</td>
</tr>
<tr>
<td>web.py</td>
<td><a href="https://github.com/webpy/webpy/blob/master/LICENSE.txt">https://github.com/webpy/webpy/blob/master/LICENSE.txt</a></td>
<td>PSF</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>pywin32</td>
<td><a href="http://docs.python.org/license.html">http://docs.python.org/license.html</a></td>
<td>PSF</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>gNumerator</td>
<td><a href="http://www.gnu/licenses/lgpl-2.1.html">http://www.gnu/licenses/lgpl-2.1.html</a></td>
<td>LGPL 2.1</td>
</tr>
</tbody>
</table>