Introduction to the Stat-JR software package | William Browne

Hello, my name is Bill Browne, and I am a professor of statistics, and co-director of the centre for multilevel modelling, which is based at the University of Bristol. In the centre we do research into statistical methodology and statistical software, with applications in the social sciences. Today I'm going to talk to you about one of our newer statistical software packages, Stat-JR which was developed in grants funded by the ESRC and in part by the National Centre for Research methods (NCRM). This work is a team effort and many colleagues at the centre and at the University of Southampton have contributed to Stat-JR.

So, what is Stat-JR? Well firstly, it is a statistical software package that we have been developing for nearly a decade at the Centre for multilevel modelling. It is pronounced, as you have gathered, 'stature', and was named in tribute to our former colleague Jon Rasbash, who was the main programmer of our other main software package MLwiN. Jon came up with the initial ideas behind Stat-JR and it is meant to appeal to users of all abilities, from novices right through to experts, who might then develop their own functionality in the package. It is written in Python and has its own estimation engine which we call eStat. This uses Monte Carlo Markov chain methods. But the package also allows interoperability with other common statistical packages, and these features will be covered in this introductory lecture. For those of you who wish to also learn about Stat-JR’s statistical analysis assistants features or its ability to create SPSS based training materials, there will be other videos that cover these topics.

So, StatJR was designed as a set of components that interlink to create a complete software package. The system has its own algebraic processor which can be used to construct algorithms for model fitting. As you will see in this diagram behind me, there are templates, of which we will say more later, that take user input and use it to specify a statistical model. This model is then sent through an algebra system and programming code is produced. This code can be combined with data to execute a model and/or to perform simulations with simulated data. Results are produced which can then be presented in many different ways.

So, StatJR works via a series of templates. These can be user written, although we supply many with the software package, and are similar to the concept of packages in the software package, 'R,' say. Some templates are for fitting specific statistical model families, and these have a very standard form. They contain an inputs method. This is used for specifying the inputs that are required for that model, a model method that produces model code, and optionally a LaTeX method. This can be used to produce code that outputs the model in LaTeX format. For more complex templates there are other functions. Here behind me now is an example of a template code for a template that fits a regression model. This is shown in part to show that the Python code is really quite short, and you can clearly see the input, model, and LaTeX methods. The input method has two inputs corresponding to the response and the predictor in the regression template, and they are given as Y and X. These are required and in brackets you will see text that will be displayed when these inputs are asked for by the system. The input method also contains parameters that are needed for the model to run.

So, let's look at running this template in practice. Behind me now you will see a screen shot of Stat-JR. So Stat-JR runs in a web browser but it's hosted locally on your own machine. The font is small, so we can blow it up a bit. You will see here that the input text appears as in the template code for the first two inputs. There are then a series of other methodological inputs such as the number of iterations to run. These are needed for the MCMC estimation engine but they all have default set for them. Clicking on the next button at the bottom of the screen after filling in the inputs will mean the template creates several objects that can be viewed on the screen below. This can be seen below where the inputs are. The objects are accessed via the pulldown list to the left, and here we see the regression equation created in LaTeX format. Let's blow this one up a bit so we can see it clearly. Here we see a regression based on the user inputs. Note that we use the term 'cons' to represent a column of ones that represents an intercept parameter, and the first two lines represent a standard linear regression. As we are using MCMC estimation the remaining lines specify prior distributions for the model parameters. As noted we use a piece of software called MATHJAX, which allows the underlying LaTeX code to be copied. If we return to the pulldown list, we can choose another object. Here we see the model code that is used by the algebra system. We can also pop out objects by clicking on the pop out button and then they will appear in separate tabs in your web browser. So, several objects can be viewed at once. Let's look at this object in more detail. Here we see the model code that the template has constructed. This code is written in a language originally developed by the team of software developers who developed the WinBUGS package. It has slightly small differences like the length function that you see here, and here the code specifies a regression model along with some diffuse prior distributions. The code produced is then fed into StatJR's algebra system, which we see on the slide following. Here we see one of the steps that the algebra system performs. For this model we will be using the MCMC method called Gibbs sampling, and the Gibbs sampling algorithm involves finding the conditional posterior distribution for each parameter in turn and these form steps in an overall algorithm. Often this is done by forming the conditional posterior distribution and then matching it to a known distributional form. In the example behind me we are looking at the intercept parameter beta-zero, and we're matching it to a known distribution, the normal distribution. This means by showing all of the steps, expert users can check that the system is working correctly. If we move on here, on this screen you will see the series of steps one for each parameter. So, this gives us a summary of the overall algorithm. In practice the algebra system works independently of any data. It just takes the model code it has been given. So, as you see here where we show you the final step that is used for this parameter, the step is defined in terms of the names of the data only. When StatJR has access to the data then we can see in the second line that we see here that the code simplifies a lot, where we can substitute constants in, and the data in, for the various variables. StatJR will then construct C++ code and fit the model. If we look now we see another object which is the actual C++ code for fitting the model, and a software developer might take this away and modify it if for example they've created their own algorithm that they are interested in fitting. We can zoom in again because that was quite small font and look at the actual code as we've replicated here. And you'll see that the code looks fairly similar to the algebra steps. You can match line for line. Of course, many of our users, in particular than novice users, will not want to look at this code at all. But instead they can simply press the Run button and run the model. And if we do this StatJR will compile the C++ code, which all happens in the background, fit the model, and return the results as we will see in a minute.

Here we see for each parameter, the results that we get out of StatJR. We have posterior mean estimates because we're running MCMC with corresponding posterior standard deviations which correspond to the standard errors in a usual procedure, and effective sample sizes. These can be used to evaluate how well the MCMC algorithm has been performing. In fact, the effective sample sizes represent how many effective independent iterations of our MCMC algorithm have been run. And here, as we see, all of these effective sample sizes are very large, over 5,000, which is good. There is also the DIC, the deviance information criterion diagnostic, and this can be used with MCMC to compare different models.

Following on we can also look at MCMC diagnostic plots and these can be used to further investigate wherever the method has been run for long enough. In Stat-R the eStat engine that we have developed runs multiple chains in parallel. And so here we see in the top left, three chains and different colours. A red chain, a green chain, and a blue chain. As well, in the top right, we have multiple density plots. Again, one for each chain each colour. If we move on we have in the middle row of this diagram, time series Diagnostics and these are used to again assess how well the MCMC method is doing, while at the bottom we have two plots, a plot that gives the Monte Carlo standard error for different lengths of the chain. So if you discover your Diagnostics aren't very good, this will tell you how much further you might have to run your chain for. In the bottom right, we have a multiple chain convergence diagnostic. the Gelman-Rubin diagnostic.

A feature of StatJR is that it can run models using other software packages. For example, given that the templates already create model code that looks a bit like WinBUGS code, it is fairly straight- forward to allow StatJR to interoperate with WinBUGS. Here we see inputs to a second template which we've called Regression 2, and this is the same as the template Regression 1, but in addition we've added one extra input which asks what estimation engine you want to use. Here you will have a choice of many different software packages all of which can fit a regression model. We have selected WinBUGS and interoperability will work in the following way. Each template in the code has additional code to explain how to fit the model in the other packages. Although for WinBUGS, as the code is so similar to the model code we have already seen, and indeed for the other packages OpenBUGS and JAGS, the StatJR system will automatically fit these models as they share this same language. Here we see in amongst the objects created when WinBUGS has been selected we see a script file, and this file contains the commands that were required by the WinBUGS package to fit the model. There are other files that that StatJR will create for use with WinBUGS. These will identify the actual data in the correct format for WinBUGS, the initial values, and the model itself. Then when we click run WinBUGS will fire up in the background, fit the model, and return the estimates as we saw with the eStat engine. So, for a novice user you will see no difference. You will just run a different package in the background and get a different set of estimates. Interoperability is also available with packages like R. In R there are options, at least for regression models, to use both the GLM function and the MCMCglmm function. Here we have chosen the GLM function and you will see the R script below in the objects list. This is the series of commands that R requires to run and fit a regression model. StatJR will also create the data in the right format for R, and again we'll call R in the background and come back with the results. So, if we look at this second diagram, when we use interoperability with StatJR, StatJR requires that the package to be interoperated with is installed on your machine, it's not magic, and that the correct path to find it is added to the inputs in the settings file. It will then run the package in the background and return the outputs. Often these outputs are simply data files which StatJR will then post process. But as you can see in this example figures can be brought back from other packages. Here for example we see a plot from R which shows the residuals from the regression model against the fitted values.

Finally, StatJR is written in Python, and so can exploit Python's own functions including via Pythons Matplotlib package, some of the graphing functions that have been developed originally in the MATLAB package. Here we're going to use another template that comes with StatJR and we've called this template XYplot, and it is used to create scatter plots of two variables. So, in the example we see here I have chosen a test data set, and I've chosen two variables, normalized exam, and standardized LRT, and you will see the scatter plot of these two variables appears as an output in the browser window.

Hopefully in that whistle-stop tour of StatJR you have learnt a little bit of its capabilities. If you want to know more then look out for the other videos or visit the centre website at the web link provided on this final slide. Thank you very much for listening.