SabreR, HPC and GRID Computing

Version 2 (Draft)

Rob Crouchley
r.crouchley@lancaster.ac.uk
Centre for e-Science
Lancaster University

March 19, 2009

Contents

Preface

The main aim of this supplement is to describe the use of sabreR on a HPC or Grid. Sabre is available in three forms: (1) stand-alone, (2) the R plugin (as discussed here), (3) the Stata plugin. The stand-alone version and the R plugin versions can be deployed in parallel on high performance computers (HPCs) or computational grids running Linux. Three other booklets deal with the statistical modelling in sabreR, these are:

- Multivariate Generalised Linear Mixed Models via SabreR (Sabre in R)
- Exercises for sabreR (Sabre in R)
- Solutions Manual for sabreR (Sabre in R) Exercises

These booklets are available from `http://sabre.lancs.ac.uk/sabreRuse_intro.html`.

If you have any suggestions as to how this booklet could be improved, for instance by the addition of other material, could you please let us know via the Sabre mailing list, `sabre@lancaster.ac.uk`.

Acknowledgements

Many thanks to Dan Grose for writing the R side of the sabreR library, this includes writing the code to submit sabreR jobs to the grid from the desktop. Dan also wrote much of the introductory material on R in Appendix B. Dave Stott and John Pritchard did all of the recent development work on Sabre. Dave wrote the standard Gaussian and adaptive Gaussian quadrature algorithms. John wrote the algorithm for manipulating the large sparse matrices used by the fixed effect estimator. John also wrote the procedures that enable Sabre to go parallel on multiple processors.

This work was supported by the ESRC research grants RES-149-28-1003: The Colaboratory for Quantitative e-Social Science (E-Social Science Centre Lancaster Node) and RES-149-25-0010: An OGSA Component-based Approach to Middleware for Statistical Modelling. Rob Crouchley was the principal applicant on both grants. We accept no liability for anything that might happen as a consequence of your use of sabreR, though we are happy to accept recognition of its successful use.

Chapter 1

Motivation and Example

1.1 Motivation

All of the examples we have used in the booklet "Multivariate Generalised Linear Mixed Models via SabreR (Sabre in R)" were reasonably quick to estimate. Various simplifications have been used. These simplifications included: using a subset of the possible explanatory covariates, using a subset of the original cases; using fewer quadrature points than were actually needed, and ignoring the endogeneity of some of the covariates. These simplifications helped us to give quick results in demonstrations during workshops. However, empirical research does not always lead to models that can be estimated so conveniently.

Causal modelling of social processes generally requires the use of multivariate models like MGLMMs, these were introduced in Chapter 8 of "Multivariate Generalised Linear Mixed Models via SabreR (Sabre in R)". The example of this Booklet is a trivariate example of: wages, training and promotion for individuals from the British Household Panel Survey (BHPS). Joint modelling of simultaneous responses in these examples allows us to disentangle the direct effects of the different responses on each other from any correlation that occurs in the random effects of the responses. Without a multivariate multilevel GLM for complex social process like these we risk inferential errors. But estimating these MGLMMs can be computationally demanding.

In this Booklet we demonstrate how easy it is use the large scale computational resources of the grid and obtain results in a reasonable amount of time without the need to make inappropriate simplifications. We first compare the performance of several commercial software systems (e.g. Stata, SAS) for estimating these models on a small data set. We use the description "small" for panel data with a few thousand (5k) individuals, we use the description "large" for something like the year 7 cohort of all pupils at school in the UK (1.5 Million). While "tiny" would be a data set with a few 10s of individuals.

1.1.1 Why Quadrature

We consider quadrature based approaches (standard Gaussian quadrature (GQ), adaptive Gaussian quadrature (AQ)) to be currently the best methods for MGLMMs. The alternatives have weaknesses, for instance:

- Penalised Quasi Likelihood (PQL): Parameter estimates tend to be biased for binary dependent variables with small cluster sizes and high intraclass correlations (e.g. Rodriguez and Goldman, 1995, 2001). Also PQL does not involve a likelihood which prohibits the use of likelihood based inference.

- Laplace Approximation: The 6th order expansion (Raudenbush et al., 2000) worked as well as 7-point AQ in simulations of a two-level binary dependent variable model. The precision of GQ and AQ can be increased by simply using more quadrature points. We cant increasing the degree of the Laplace Expansion beyond the number of terms allowed for.

- Computer intensive alternatives to quadrature based approaches include simulation based approaches such as Markov Chain Monte Carlo (MCMC) (e.g. Gelman et al., 2003) and maximum simulated likelihood (MSL) (Hajivassiliou and Ruud, 1994). The hierarchical structure of multilevel models lends itself naturally to MCMC using for instance Gibbs sampling. If vague priors are specified, the method essentially yields maximum likelihood estimates. Unfortunately, a problem with MCMC is how to ensure that a truly stationary distribution has been obtained for MGLMMs, especially when we have a lot of structural and incidental parameters.

1.1.2 Software for estimating MGLMMs

There is a range of software tools that have quadrature based or similar approaches for estimate MGLMMs. We have listed some of the main ones and summarised their features. Why have we excluded GEE?

Packages for R at http://cran.r-project.org/ for GLMMs and MGLMMs

1. lmer: http://cran.r-project.org/web/packages/lme4/index.html. However, Lmer uses the Laplace Approximation to the integrals and fits the model using penalized iteratively reweighted least squares.

2. npmlreg: http://cran.r-project.org/web/packages/npmlreg/index.html. Npmlreg contains both standard Gaussian Quadrature and Non Parametric Maximum Likelihood as alternative methods for handling for the random effects, it uses the EM algorithm to fit the model.

Commercial Software for MGLMMs

1. Stata: http://www.stata.com/. The xt set of procedures include standard Gaussian quadrature and adaptive Gaussian Quadrature for the random effects. The xt procedures use Newton Raphson to fit the model. Stata also have Stata MP for running on multiple processors.
2. SAS: http://www.sas.com/. SAS has the procedure PROC NLMIXED which can use adaptive quadrature for the random effects, the default procedure for fitting the model is quasi Newton. SAS also have the procedure PROC MPCONNECT for using multiple processors. There is also SAS Grid computing.
3. Limdep: http://www.limdep.com/. Limdep has a range of special procedures for estimating GLMMS, these include standard Gaussian quadrature, the model is fitted using a quasi Newton algorithm

Other Systems

1. MLwiN: http://www.cmm.bristol.ac.uk/. MLwiN uses both penalised quasi likelihood (PQL) and the Laplace approximation for GLMMs, the model is fitted using Iterative reweighted least squares. MlwiN also provide the Bayesian procedure MCMC for fitting models.
2. Gllamm (Stata prog): http://www.gllamm.org/. Gllamm is a Stata program, it has both standard Gaussian quadrature and adaptive Gaussian quadrature, the model is fitted using Stata ML procedure, which uses Newton Raphson.

1.1.3 The Relative Performance of Different Software Packages for Estimating Multilevel Random Effect Models

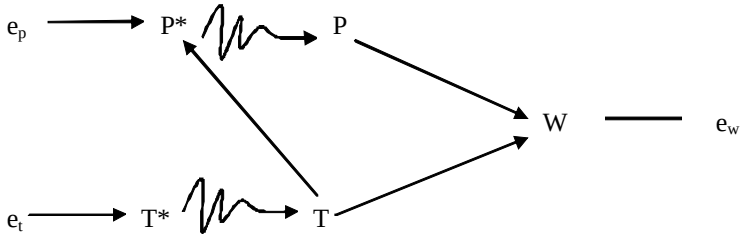
We next compare the performance of Stata, gllamm (Stata) and SAS with Sabre for estimating a range of computationally models on the same small data set. In the 1st set of comparisons we use a single processor. We then show the extra speed up that can be obtained by using the grid.

1.1.4 Example: Wages, Promotion and Training

In this example we use a sample of males from the BHPS who were employed and earning a wage at some point over the period 1991-2003. This gives a total of 5130 individuals with a sequence of responses that occurred somewhere in the 13 year interval. At the 1st sample point of the survey (1991) there were was 2316 individuals of whom 945 of these males had some form of training in the previous 12 months, 106 had been promoted in the previous 12 months.

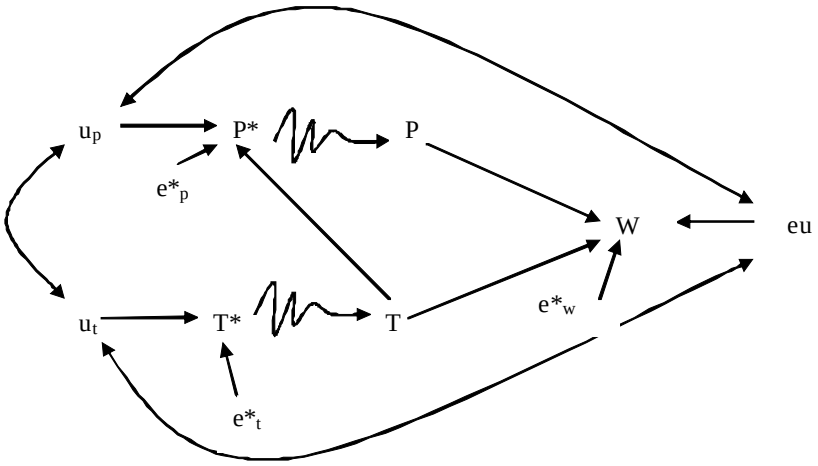
The mean of the log of their weekly wage was 5.65 (Sterling) with a standard deviation of 5.87.

We use a trivariate model to disentangle the observed and unobserved dependencies between: Promotion (P=1,0) in the last 12 months (latent variable P*), on the job training (T=1,0) in the last 12 months (latent variable T*) and current wages (W). The observed P is derived from the latent variables P*, such that P=1 if P*>0, and P=0 otherwise, similarly for T. In the diagram below we have assumed independence between the stochastic disturbances (e_p,e_t and e_w) of the model.



Trivariate Model Assuming Independence Between the Disturbance Terms

In this figure we have put a direct effect between T and P*, though we could just have easily had it between P on T*. This model can be quickly estimated using standard glm software. In the next Figure we have added the time constant random effects (u_p, u_t and u_w) to the model. The other variables can vary with wave of the panel, but for simplicity we have left off this detail. The figure also includes curved lines to represent the correlation between these random effects.



Correlated Random Effects Model

The models also contain a range of explanatory covariates, for instance there are 74 (including T and P) in the wage equation. This is typical of the complexity that can occur in evidence based research. We have estimated the univariate panel models (assuming independence), the bivariate model (T and P) and the trivariate model on a single core to give you some idea of the increased demands that occur as we add complexity.

Researchers in the exploratory phase of their work typically need quick but reliable estimates of the many different versions of the model they are trying to fit, os it is important to have some idea of the performance of the alternative statistical tools that are available for estimating these models.

1.1.5 Comparison

In this section we compare Sabre, gllamm (Stata), SAS and Sabre for estimating multivariate multilevel random effect models on the Lancaster node of the NW-GRID. The NW-GRID is affiliated to the UK's National Grid Service (NGS). In all comparisons we use the default or recommended starting values of the different procedures. The HPC execution nodes are 48 Sun Fire X4100 servers with two dual-core 2.4GHz Opteron CPUs, for a total of 4 CPUs per node. The standard memory per node is 8G, with a few nodes offering 16G. All nodes also offer dedicated inter-processor communication in the form of SCore over gigabit Ethernet, to support message passing (parallel) applications. In these comparisons, whenever possible, adaptive Gaussian quadrature was used, the number in brackets is the number of quadrature points used.

Example	data	Obs	Cases	Vars	Size (tab)	Method	Stata	gllamm	SAS	npmlreg	lmer	Sabre 1
univariate	Wages (W)	31022	5285	74	17.1MB	AQ (12)	15"	22h23'	3h26'	1h28'	2'03"	1'05"
univariate	Train (T)	31022	5285	71	17.1MB	AQ (16)	11'51"	25h32'	7+days	44'39"	5'51"	50"
univariate	Prom (P)	31022	5285	72	17.1MB	AQ (16)	15'08"	25h32'	7+days	58'37"	4'37"	52"
bivariate	T & P	62044	5285	143	34.2MB	AQ (16x16)	na	150+days	30+days	na	nd	1h42'
trivariate	W & T & P	93066	5285	217	51.3MB	AQ (12x16x16)	na	15+yrs	1+yrs	na	nd	115h45'

Key

Sabre 1 is Sabre running on 1 core

na: neither Stata 9 nor npmlreg can estimate multivariate random effects models using quadrature

time+: indicates an estimated lower CPU limit

Obs is the number of Observations in the data set

Vars is the number of explanatory variables in the model

MB is the size of the raw data set (R tab form) in Megabytes

Neither standard Gaussian quadrature nor adaptive Gaussian quadrature have been implemented for `lmer`, the procedures for REML and ML give the same answer as the Laplace approximation. The times quoted for `lmer` are for the Laplace approximation. For `npmlreg,` the times are for standard Gaussian quadrature as adaptive Gaussian quadrature times are not available. Sabre used Portand Group PGF90 7.1-6 Compiler with –FAST (Level 2 optimization). In the Table the times are system times (these were very close to real times in all figures), and there was very little variation between runs. The time for Stata for the univariate Wage (W) equation is for the analytic model using `xtreg`, i.e. this does not use quadrature and similarly for SAS which used PROC MIXED.

As expected for the univariate linear model of Wages (W) with 74 covariates, the analytic model of Stata is the fastest. This was followed in order by Sabre 1, lmer, npmlreg, SAS and gllamm. For the univariate binary response mode for Training (T), with 71 covariates, Sabre 1 is much faster than the others, it is followed by lmer, Stata, nplmreg, gllamm and SAS. The same is true for the univariate binary response model for Promotion (P). Neither Stata nor npmlreg have implemented quadrature based procedures for multivariate response models. Sabre appears to out form both gllamm and SAS, though both these times were estimated. It is not just the actual times that are important, as these will get shorter on faster processors, its worth comparing the relative times.

For the univariate models and if we ignore the Stata timings for the linear model (analytic integral), then this Table shows that serial Sabre (Sabre 1) is over 10 times faster than Stata and well over 1000 times faster than gllamm. Serial Sabre is also about 50 times faster than npmlreg and 2-5 times faster than lmer. Sabre also seems to be well over 100 times faster than SAS PROC NLMIXED.

A similar set of comparisons for the bivariate model of Chapter 8 (wages and union) of "Multivariate Generalised Linear Mixed Models via SabreR (Sabre in R)" on a slightly smaller number of number of cases but with only 8 covariates in each linear predictor is presented below.

Example	data	Obs	Cases	Vars	Size (tab)	Method	Stata	gllamm	SAS	npmlreg	lmer	Sabre 1
univariate	Wages (W)	18995	4132	8	1.61MB	AQ(16)	0.5″	19'37″	1'06"	12'20″	1'06"	6″
univariate	Union (U)	18995	4132	8	1.61MB	AQ(36)	51″	41'53″	23'25"	24'56″	1'25"	11″
bivariate	W & U	37990	4132	16	3.2MB	AQ(16x36)	na	167h42'	21h10"	na	nd	9'58″

Timings for a smaller example

All the software tools are faster but the story stays the same. These results are only suggestive, there may well be other substantive social science research settings with similar sized data sets for which the picture is very different. There could also be non default settings that radically change the performance of an algorithm for this class of model.

However, what is scientifically important, is that our substantive results change as the modelling becomes more comprehensive. The trivariate analysis not only provided a estimate of the correlations in the unobserved random effects of the different responses (W,T,P) but also the inference on a range of explanatory variables changed. Most importantly the coefficient on Promotion in the wage equation is much smaller in the trivariate model, when compared to that obtained from the homogeneous model and that from the independent random effect model for wages.

		Models		
		Homog	Indep	Dep
Covariate	Promo	0.09499	0.06103	0.05288
Coeff		0.00824	0.00599	0.00611
in Wage	Train	-0.00683	-0.00865	-0.00864
Equation		0.00526	0.00396	0.00405
Likelihood		-38471.93	-29448.19	-29419.52

Changing Inference on direct effects in the Wage equation

For further comparisons on other tiny and small data sets and with other software systems see `http://sabre.lancs.ac.uk/comparison3.html`. At this point it seems that the bigger the data set, or the more complex the model, the better the relative performance of Sabre. In all of our comparisons to date, the numerical properties of Sabre's estimates compare favourably with those of the alternatives and it has the best (like for like) overall computational speed. The speed produced by Sabre made it possible to explore many more comprehensive model specifications of MGLMMs in a reasonable time period. In the next

section we show the extra speed up that can be obtained by using Sabre on the Lancaster node of the NW-Grid.

1.2 Submitting a sabreR grid job

This section introduces the basic commands provided by `sabreR` for grid computing, it shows how a `sabreR` grid session object, necessary for connecting to a `sabreR` server and onto the grid (in our example the Lancaster node part of the North West Grid , see http://www.nw-grid.ac.uk/?q=index). The North West Grid is affiliated to the UK National Grid Service, see http://www.grid-support.ac.uk/). We use `sabreR` to estimate the bivariate response model (`ln_wage` and `tunion)` on the `nls.tab` data set of Chapter 8 of "Multivariate Generalised Linear Mixed Models via SabreR (Sabre in R)". This model can take a couple of minutes to estimate on a laptop. We repeat the detail below to remind you what was involved. We then show the extra steps that are needed to do the same computation on 4 processors of the Lancaster node of the NW-GRID and then obtain the results. The process we follow is quite general and with little change can be used for any remote system and any number of processors.

1.2.1 Data description for `nls.tab`

Number of observations: 18995
Number of level-2 cases: 4132

1.2.2 Variables

`ln_wage`: ln(wage/GNP deflator) in a particular year
`black`: 1 if woman is black, 0 otherwise
`msp`: 1 if woman is married and spouse is present, 0 otherwise
`grade`: years of schooling completed (0-18)
`not_smsa`: 1 if woman was living outside a standard metropolitan statistical area (smsa), 0 otherwise
`south`: 1 if woman was living in the South, 0 otherwise
`union`: 1 if woman was a member of a trade union, 0 otherwise
`tenure`: job tenure in years (0-26)
`age`: respondent's age
`age2` : age* age

The data displayed below (`nls.tab`), is used for to estimate the joint model for `lnwage` and `union`.

idcode	year	birth_yr	age	race	msp	nev_mar	grade	collgrad	not_smsa	c_city	south	union	ttl_exp	tenure	ln_wage	black	age2	ttl_exp2	tenure2
1	72	51	20	2	1	0	12	0	0	1	0	1	2.26	0.92	1.59	1	400	5.09	0.84
1	77	51	25	2	0	0	12	0	0	1	0	0	3.78	1.50	1.78	1	625	14.26	2.25
1	80	51	28	2	0	0	12	0	0	1	0	1	5.29	1.83	2.55	1	784	28.04	3.36
1	83	51	31	2	0	0	12	0	0	1	0	1	5.29	0.67	2.42	1	961	28.04	0.44
1	85	51	33	2	0	0	12	0	0	1	0	1	7.16	1.92	2.61	1	1089	51.27	3.67
1	87	51	35	2	0	0	12	0	0	0	0	1	8.99	3.92	2.54	1	1225	80.77	15.34
1	88	51	37	2	0	0	12	0	0	0	0	1	10.33	5.33	2.46	1	1369	106.78	28.44
2	71	51	19	2	1	0	12	0	0	1	0	0	0.71	0.25	1.36	1	361	0.51	0.06
2	77	51	25	2	1	0	12	0	0	1	0	1	3.21	2.67	1.73	1	625	10.31	7.11
2	78	51	26	2	1	0	12	0	0	1	0	1	4.21	3.67	1.69	1	676	17.74	13.44
2	80	51	28	2	1	0	12	0	0	1	0	1	6.10	5.58	1.73	1	784	37.16	31.17
2	82	51	30	2	1	0	12	0	0	1	0	1	7.67	7.67	1.81	1	900	58.78	58.78
2	83	51	31	2	1	0	12	0	0	1	0	1	8.58	8.58	1.86	1	961	73.67	73.67
2	85	51	33	2	0	0	12	0	0	1	0	1	10.18	1.83	1.79	1	1089	103.62	3.36
2	87	51	35	2	0	0	12	0	0	1	0	1	12.18	3.75	1.85	1	1225	148.34	14.06
2	88	51	37	2	0	0	12	0	0	1	0	1	13.62	5.25	1.86	1	1369	185.55	27.56
3	71	45	25	2	0	1	12	0	0	1	0	0	3.44	1.42	1.55	1	625	11.85	2.01
3	72	45	26	2	0	1	12	0	0	1	0	0	4.44	2.42	1.61	1	676	19.73	5.84
3	73	45	27	2	0	1	12	0	0	1	0	0	5.38	3.33	1.60	1	729	28.99	11.11
3	77	45	31	2	0	1	12	0	0	1	0	0	6.94	2.42	1.62	1	961	48.20	5.84

First few lines of `nls.tab`

We take `ln_wage` (linear model) and `union` (probit link) to be the response variables and model them with a random intercept and a range of explanatory variables.

Besides allowing for the overdispersion in `ln_wage` and `union`, and correlation between them, the `ln_wage` equation contains `union` as an explanatory variable. We start by estimating the joint model on the sequences of `ln_wage` and `union` from `nls.tab`. We use standard Gaussian quadrature (the default) with the default number of quadrature points (12) for both responses. We now present the script needed to estimate the bivariate model locally (as in Chapter 8 of "Multivariate Generalised Linear Mixed Models via SabreR (Sabre in R)") and the script needed to undertake the same task on the grid (from a networked Windows PC). In the grid job script we use a proxy certificate (grid session object) that was created earlier to submit a job to 4 processors of the Lancaster node of the NW-Grid, we then retrieved the results. You should notice that there are several small differences between these two examples. In the second example we pass over the need to create the grid session object, as this had been done earlier. We will return to this in the next section.

1.2.3 Sabre commands: local job

```
# open up the log file
sink(file="/Rlib/SabreRCourse/examples/ch13/l9grid1.log")

#load the sabreR library
library(sabreR)

# set up the data
nls <- read.table(file="/Rlib/SabreRCourse/data/nls.tab")
# but because the variable union is reserved we need to reset it
attr(nls,"names")[13] <- "tunion"
attach(nls)
```

```
# look at the 1st 10 lines and colums of the data
nls[1:10,1:10]

# estimate the bivariate model
sabre.model.3 <- sabre(ln_wage~black+msp+grade+not_smsa+south+tunion+
                        tenure+1,
                        tunion~age+age2+black+msp+grade+not_smsa+
                        south+1,
                        case=idcode,first.family="gaussian",
                        second.link="probit")
                       sabre.model.3
Look at the results
sabre.model.3

detach(nls)
rm(nls,sabre.model.3)
sink()
```

1.2.4 Sabre commands: grid job

```
# open up a new log file
sink(file="/Rlib/SabreRCourse/examples/ch13/l9grid2.log")

#load the sabreR library
library(sabreR)

# set up the data
nls <- read.table(file="/Rlib/SabreRCourse/data/nls.tab")
# but because the variable union is reserved we need to reset it
attr(nls,"names")[13] <- "tunion"
attach(nls)

# look at the 1st 10 lines and colums of the data
nls[1:10,1:10]

# create and save your credentials (grid.demo.session.R)
# here is one we did earlier
# grid.demo.session<-sabre.session.dlg()
# save(file="grid.demo.session.R",grid.demo.session)

#clear objects
#rm(list=ls())

#load the saved session object (proxy certificate)
load(file="grid.demo.session.R")

#look at its propertoes
grid.demo.session

# run the model (parallel on 4 processors)
```

```
sabre.modelg.3 <- sabre(ln.wage~black+msp+grade+not.smsa+south+tunion+
                        tenure+1,
                        tunion~age+age2+black+msp+grade+not.smsa+
                        south+1,
                        case=idcode,first.family="gaussian",
                        second.link="probit",
                        session=grid.demo.session,
                        description="bivar model")

# recover the results
sabre.results(grid.demo.session,sabre.modelg.3)

#clear the workspace
detach(nls)
rm(nls,sabre.modelg.3,grid.demo.session)

sink()
```

1.2.5 Sabre log file

```
   Parameter               Estimate         Std. Err.
    ------------------------------------------------------
    (intercept).1           0.75162          0.26753E-01
    black.1                -0.69805E-01      0.12511E-01
    msp.1                  -0.14237E-02      0.59871E-02
    grade.1                 0.73275E-01      0.19736E-02
    not.smsa.1             -0.14524          0.88679E-02
    south.1                -0.74533E-01      0.89063E-02
    tunion.1                0.96328E-01      0.70837E-02
    tenure.1                0.28328E-01      0.65261E-03
    (intercept).2           -2.5481          0.38382
    black.2                 0.84621          0.69172E-01
    msp.2                  -0.64955E-01      0.41090E-01
    grade.2                 0.64562E-01      0.12164E-01
    not.smsa.2             -0.10254          0.58471E-01
    south.2                -0.73260          0.56972E-01
    age.2                   0.20406E-01      0.23558E-01
    age2.2                 -0.18467E-03      0.37617E-03
    sigma1                  0.26170          0.15009E-02
    scale1                  0.27466          0.36213E-02
    scale2                   1.4765          0.37284E-01
    corr                    0.11927          0.24144E-01


    Correlated bivariate model

    Standard linear/probit
    Gaussian random effects

    Number of observations             =   37990
    Number of cases                    =    4132
```

```
X-var df           =    16
Sigma df           =     1
Scale df           =     3

Log likelihood =     -12529.120     on   37970 residual degrees of freedom
```

1.2.6 Differences between the 2 sabreR scripts

Both scripts are very similar. The main additions are in the second example where we used the command

```
load(file="grid.demo.session.R")
```

This command loads the `grid.demo.session` object which contains the proxy certificate + server + port number details (see below) and had been created and saved earlier. This object (`grid.demo.session`) was then used as part of the `sabre.modelg.3` object to submit the data and the R commands (sabreR model) to the remote Lancaster Node of the NW-grid. The `sabre.modelg.3` object also included the script

```
description="bivar model"
```

to provide a short description of the job. This short description can help distinguish this job from the many others that may be submitted. The command

```
sabre.results(grid.demo.session,sabre.modelg.3)
```

retrieved the results from the remote system.

1.3 Creating a proxy certificate and grid session object

In the R example of the "Sabre commands: grid job" sub section we used a grid session object (proxy certificate + server address + port number) that had been created earlier. To create a grid session object use the command

```
grid.demo.session<-sabre.session.dlg()
```

this will open a dialogue box. To complete this dialogue box you will need several important pieces of information, these are:

- The host address of a server that provides the sabreR services, in the above example this was stats-grid.hpc.lancs.ac.uk.

- The particular service (port) to use, this determines the number of processors, in this example we used port `50204`, this port provides a queue for 4 processors on the Lancaster Node of the NW-Grid. Other ports give access to different numbers of processors, these are listed in the Table below:

Port	processors
50201	1
50202	2
50204	4
50208	8
50216	16
50232	32
50248	48

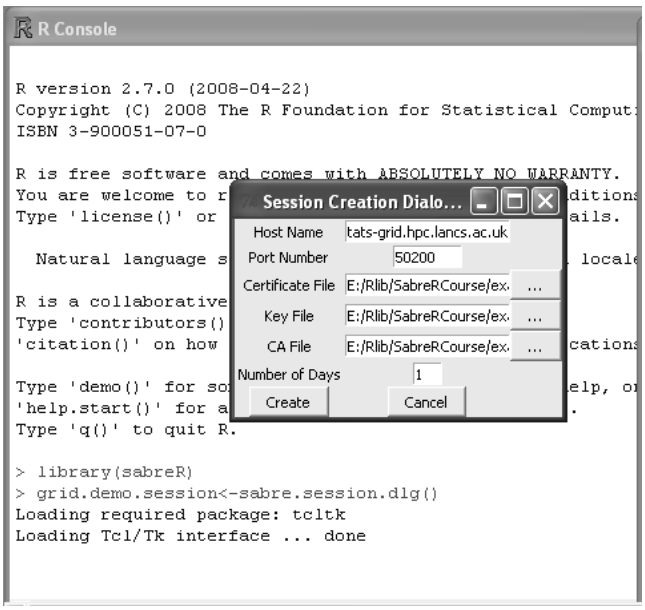
- The certificate and key files to be used for creating the proxy credentials necessary to identify the user and to encrypt the communication channel. These are obtained from your `pk12` file using OpenSSL.

- The Certificate Authority files used to authenticate the user (to the server) and the server (to the user), these can be obtained from http://www.grid-support.ac.uk/content/view/182/184/ .

- The number of days for which the credentials produced from this information are to be valid for

- The password used to create your private key with OpenSSL.

To make providing this information as easy as possible, user input is made via a graphical dialogue. The Figure below shows this dialogue being used to enter the information.



After pressing `Create`, you will need to enter your OpenSSL password in the passord dialogue box, which looks like this:

Click on `Okay`, after you have entered your OpenSSL password. On successful completion, R will return control to the screen. To save the proxy certificate as `grid.demo.session.R` in the current directory for later reuse, type the following command

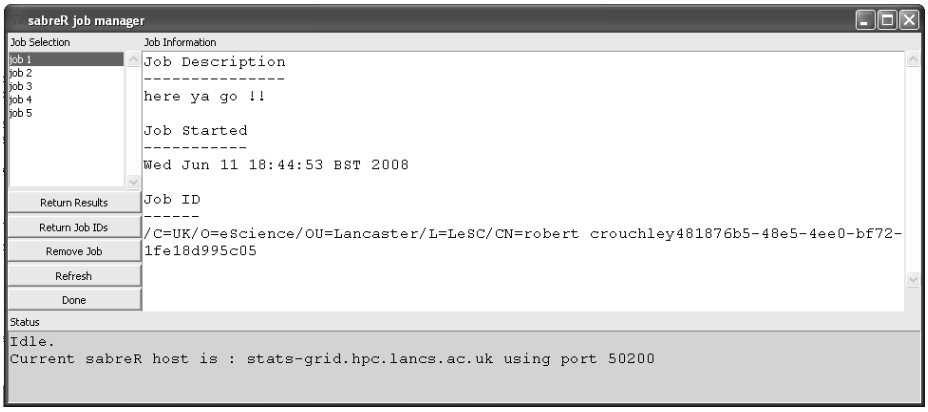
```
save(file="grid.demo.session.R",grid.demo.session)
```

1.4 Managing Jobs and Obtaining Old Results

To obtain results, you do not need to use the same grid session object that was used to submit the job, the proxy certificate may have expired, but to get your results you will need to use a grid session object that uses the same server (in our case `stats-grid.hpc.lancs.ac.uk`) and the same port you used (in our case `50204`). Different ports with have differerent lists of jobs. You can find out what results are available by using the command

```
my.old.job<-sabre.jobs.dlg(grid.demo.session)
```

This will open the job manager window, as follows. The job manager window contains the information on all the jobs that you have submitted.



Sabre Job Manager on the UK grid

The job manager can be used to check which jobs are complete, it can be run from a different system in a different location. The job manager `Job Selection` sub window above shows a set of jobs, (job1,...,job5), these jobs are sorted by job ID. Often `job 1` is the most recent. The status of the highlighted job ID is displayed in the `Job Information` window. It is best to double click on the job you are interested in, as this software can be a bit slow to respond. It is at this point that the Job Description submitted with the job can be used to help you understand what the task was about. The date the Job Started and the Job ID given by the system are also displayed.

The results for each job can be obtained when required by using the `Return Results` button, this returns the results from that job to the R screen. Each session object has its own set of results.

Results are only removed from the server when the user clicks on the `Remove Job` button for the highlighted job, or after a pre-defined period of time specified (usually 1 month) by the sabreR service provider.

The `Return Job IDs` button returns all of the job IDs and closes the dialogue.

The `Refresh` button refreshes the job information.

The `Done` button closes the dialogue.

Appendix A

Parallel Sabre and Grid Computing in the UK

A.1 Parallel Sabre

Parallel Sabre may be run on any multiprocessor system on which the library MPI has been installed. It provides huge improvements in speed, in particular on very large problems where speed is most crucial.

A.1.1 MPI

MPI or Message Passing Interface is a library of routines which may be called from a program written in C or Fortran. Its programmer interface is a de-facto standard and it is available for a wide range of computer hardware. Some MPI implementations are available free of charge. It allows the programmer to run multiple copies of a program and have them communicate with each other. This is of benefit when the program has a sequence of tasks to perform, each of which does not depend on the others. These tasks can, instead of being performed in sequence, be performed simultaneously or "in parallel" with each copy of the program running on a separate processor.

Both commercial and free implementations of MPI have been written. A comprehensive MPI resource is available from Argonne National Laboratory at `http://www-unix.mcs.anl.gov/mpi/`

A.1.2 Simple example of the use of MPI

The example below, written in Fortran 90, shows the source code of a program written to sum the first n squares in parallel by using MPI. Obviously it is a simple job to do in a normal program on one processor but, just for the purpose of demonstrating MPI, we shall do it on n processors where n is the number of squares to be summed. The task, although trivial, is suitable for performing in parallel since the calculation of any one square is independent of the calculation of the others so all can be done at the same time. When using MPI you create multiple copies of the same program to run on the number of processors you choose. It is obviously no use if each copy of the program does exactly the same thing so there has to be a means for each copy to identify itself so that it can perform a task which is different from that performed by the other copies. MPI provides this means by first numbering the n processors used from 0 to n-1 and then providing the following 2 routines:

`MPI_COMM_SIZE` to determine how many processors are in use. This allows the program to be written in such a way that it can run on any number of processors from 1 to the maximum number available on the system being used.

`MPI_COMM_RANK` to allow a copy of the program to discover which processor it is running on.

In the example, each copy of the program, having first called `MPI_INIT` to initialize MPI internally, calls `MPI_COMM_SIZE` to find out how many processors are in use. The response is returned to the program copy in the variable num_processors. Each then calls `MPI_COMM_RANK` and the number of the processor the program copy is running on is returned to it in the variable this_processor. It is this variable which allows each copy to identify itself and perform the appropriate task. Since each copy of the program is to calculate a square between 1 and n squared where n is the number of processors, the appropriate task for each program copy is to take its processor number, add one to it (since processor numbers start at zero) and square it.

Having done this, each program copy will have one of the squares stored in its copy of the variable nsquared. How then to sum them? To do this, one program copy takes charge by receiving from each of the others their value of nsquared. In the example the program copy which does this is identified by the parameter `MASTER` and this is assigned to be the copy running on processor 0. It could have been any other copy but 0 is a good choice since however many processors are in use there will always be at least a processor number 0.

MPI provides the basic communication routines `MPI_SEND` and `MPI_RECV.` The program copies apart from the master copy, which are typically called the slaves, each send their copy of nsquared to the master copy by calling `MPI_SEND.` The master copy in turn receives the value sent from each of the others by calling `MPI_RECV` n-1 times. It then adds these values to its own square then calculates the sum and prints the answer.

(Note that the calculation of a sum need not be done on just one processor. In a simple example it makes sense for just one to do it but if there were very many terms in the sum then each of the program copies could perform a partial sum and return that to the master copy which would then sum the partial sums to get a total sum. In general the decision about what to do in parallel and what to leave to just the master is determined by comparing the time saving in having operations performed in parallel against the time taken to perform the inter-process communication. The communication time or latency is very hardware dependent so a certain amount of experimentation is required before deciding if running in parallel will be beneficial.)

A.1.3 Example code

```
!    A program to calculate the squares of the first n integers in parallel
!
      implicit none
!    Include header file provided with MPI installation for declarations of
!    MPI names
      include 'mpif.h'
!    Let processor 0 be the master
      integer, parameter :: MASTER = 0
      integer :: num_processors, ierror, this_processor, n, nsquared, sum, &
                 slave, slave_value, status( MPI_STATUS_SIZE )
!
!    Initialise MPI
!
      call mpi_init(ierror)
!
!    Find out how many processors there are
      call mpi_comm_size( MPI_COMM_WORLD, num_processors, ierror )
!    Find out which processor this copy of the program is running on
      call mpi_comm_rank( MPI_COMM_WORLD, this_processor, ierror )
!
!    Each processor has to square its processor number + 1 since processor
!    numbers start at zero
!
      n = this_processor + 1
      nsquared = n**2
!
!    Slaves send their value to MASTER
!
      if( this_processor /= MASTER ) then
         call mpi_send( nsquared, 1, MPI_INTEGER, MASTER, 0, MPI_COMM_WORLD, &
                        ierror )
      else
!
!    Master initialises the sum with its value and then receives values from
!    each of the slaves
!
         sum = nsquared
```

```
         do slave = 1, num_processors-1
            call mpi_recv( slave_value, 1, MPI_INTEGER, &
             slave, MPI_ANY_TAG, MPI_COMM_WORLD, status, ierror )
            sum = sum + slave_value
         end do
      end if
!
!    Finished with MPI
      call mpi_finalize(ierror)
!
!    Master prints the answer
!
      if( this_processor == MASTER ) then
         print *, 'The sum of the first ', num_processors, ' squares is ', sum
      end if
!
      stop
      end
```

A.1.4 How is it done in sabreR

In the random effects models the log likelihood function, the Hessian of second derivatives and the score vector are calculated by summing over each of the individuals or cases in the data set. Each term in the sum is independent of the others so the calculations for each individual are suitable for running in parallel. Each individual's calculations are allocated to the next available processor until each processor has been utilized and then Sabre continues to cycle around each of the processors until every individual's contributions to the sum have been calculated. Each processor therefore calculates a partial sum of the log likelihood, score vector and Hessian and at the end of the iterations these are summed by the "master" processor.

In the fixed effects model the algorithm is very different involving the solution of a set of linear equations for the parameter estimates. Most of the time taken in this model is in the calculation of the standard errors which requires the inversion of the $[\mathbf{X}'\mathbf{X}]$ matrix whose rank is equal to the number of individuals. Each column of the inverse can be calculated independently of the others so Sabre cycles through the processors as many times as are required until each column of the inverse has been calculated.

A.2 Why R

Tools for computationally demanding statistical research are becoming available as part of commercial systems, e.g. SAS grid computing and Stata MP. However, these systems can be of limited use on a public grid, e.g. Stata MP can't have

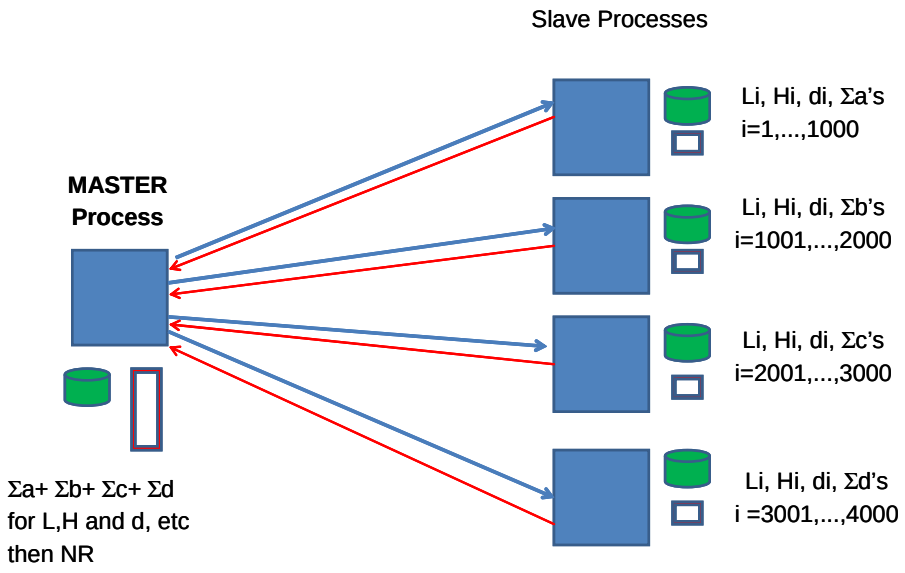


Figure A.1: Use of mpi in sabreR

multiple data sets in memory and neither system provides access to their source code. Furthermore, there are no plans to install them on the UK National Grid Service (NGS) because of cost/licensing issues.

We use R as the framework to enable statistical modelling on the grid because:

1. R is an effective, efficient and easy to use tool for Statistical Modelling
2. Many existing tried and tested statistical methods already available for R and can easily be modified to exploit the benefits of grid computing
3. Work flows to support the modelling process are simple to create
4. R is easy to install on most popular operating systems (Windows, Unix, OSX) and can be used directly from a USB memory stick
5. R includes a programming environment, which when used in conjunction with our multiR package, automatically provides a data centric scripting tool for grid computing
6. There are no licensing issue

A.2.1 Enabling Technology

The tools we have developed as part of the Lancaster Centre for e-Science are: (1) multiR (coarse grained parallel job submission) which can be used in the

model exploration/checking stages.(2) sabreR which includes (fine grained parallel Sabre job submission) which can be used to estimate computationally demanding event history models. All multiR and sabreRgrid require is:

1. An internet connection
2. The installation of our multiR and sabreRGrid packages for R
3. A certificate to identify the client to the host – typically a grid certificate

Importantly there is no need for users to install or have familiarity of Globus, VDT, gsissh, gsiscp, grid-ftp, grid-proxy tools or any other GRID related software. To the user there is very little difference between using the Sabre library from within R on the desktop, and using Sabre for statistical modelling on the grid from within R. MultiR and sabreR have a similar enabling technology, for further details about the architecture behind this enabling technology see Grose, D., (2008) High Throughput Distributed Computing using R: the multiR Package, at http://www.barnholme.plus.com/multiR/multiR-paper.pdf.

A.3 Using the National Grid Service

To use resources on the National Grid Service (UK grid), you first need to be able to prove your identity, i.e. authenticate yourself. This is done by means of a certificate issued by a Certification Authority (CA). Your certificate proves you are who you claim to be, but doesn't in itself entitle you to use any given resource. Currently the best place to find out more about the UK grid is from the "Documentation" pages of the National Grid Service, at http://www.grid-support.ac.uk/content/view/206/115/. These pages contain an introduction to the grid and grid computing, how to get a certificate, authentication on the NGS etc. However, the pages titiled "How to connect to the NGS" do not specifically mention connecting from a windows PC using sabreR.

Before you submit a sabreR job to the NGS you will need to export your joint certificate and key file (say user.p12) from the PC and browser you used to obtained it. Once you have exported this file you will need to convert this file into separate certificate and key files . This can be done using the openssl command, openssl can be activated by clicking on its icon in the sabreR directory. You should use the command, where [name] is your filename

```
openssl pkcs12 -in [name]-cert.p12 -clcerts -nokeys -out [name]-cert.pem
```

to generate the certificate file and

```
openssl pkcs12 -in [name]-cert.p12 -nocerts -out [name]-key.pem
```

to generate the key file.

You will also need to obtain the uk e-science Certification Authority (CA) certificate file, from http://www.grid-support.ac.uk/content/view/182/184/. Use a text editor and add the concatanate of all these components into it, I called my file CAcert.txt.

Finally you will need to register to use the resouces you want to use, in our case this is the NW-GRID, you can register on nw-grid, https://man4.nw-grid.ac.uk:8443/user_registration/, it may want to exchange certificates with your browser, you can ignore this. Sabre is a specific resource on the NW-GRID, so under project you can click on `sabre/sabreR (PI: Rob Crouchley)`

When you submit a job, it takes a certificate with it to the computing resources where it will run. To reduce the effect of any security breach, it does not take your full certificate, but a proxy certificate which entitles it to (some of) your privileges, but has a restricted lifetime. SabreR uses proxy certificates. Before running any jobs you will need to generate this proxy certificate using the command

```
grid.demo.session<-sabre.session.dlg()
```

this will ask you to set the number of days that you want this particular proxy certificate (in this case grid.demo.session) to last.

A.4 Grid Certificates

Authentication to and from the NGS (and most other) grid services is mediated by a modified version of standard X509 certificates. The grid middleware which deals with this is the Globus Security Infrastructure (GSI) component of the Globus Toolkit. To this end, users must begin by obtaining a valid user certificate.

Users can obtain a user certificate from a valid Certification Authority (in the case of the NGS, this is UKeSCA, the UK e-Science Certification Authority), who will digitally sign the user certificate with their own root certificate. Each user certificate has a unique identifier called a Distinguished Name (DN). E.g.:

```
/C=UK/O=eScience/OU=Lancaster/L=LeSC/CN=rob crouchley
```

A user's DN entry must be added to a server's gridmapfile before they can access that server's grid services. Obtaining a UKeSCA signed user certificate doesn't automatically give you access to any UK e-Science resources - it's up to individual sites or grids to add your DN entry before you can access their services. More information on exactly how that's done, and how that information is co-ordinated across the NGS can be found from the www.ngs.ac.uk site.

The purpose of a user certificate is to allow for a single sign-on across all authorised grid facilites. It is possible for a user to launch a job request to one grid server which might then require grid services from other authorised servers. This requires that the job automatically authenticate itself to those other servers without the need for further user intervention. This is acheived by grid services passing around a special version of the user's credentials.

As passing around the original user certificate details is potentially insecure, the Globus Security Infrastructure implements a scheme where users create a limited lifespan self-signed proxy certificate from their original user certificate, with its own public and private key. This proxy certificate is then used for authentication. In the unlikely event that the proxy certificate is intercepted, the proxy's limited lifespan (by default 12 hours) ensures that it cannot be feasibly decrypted and used for unauthorised purposes within its own lifespan.