



Working Paper

mixl : An open-source R package for estimating complex choice models on large datasets

Author(s):

Molloy, Joseph; Schmid, Basil; Becker, Felix

Publication Date:

2019

Permanent Link:

<https://doi.org/10.3929/ethz-b-000334289> →

Rights / License:

[In Copyright - Non-Commercial Use Permitted](#) →

This page was generated automatically upon download from the [ETH Zurich Research Collection](#). For more information please consult the [Terms of use](#).

mixl : An open-source R package for estimating complex choice models on large datasets

Joseph Molloy, Basil Schmid, Felix Becker

IVT ETH Zurich, Stefano-Frascini-Platz 5, 8093 Zurich, Switzerland

Abstract

This paper introduces *mixl*, a new R package for the estimation of advanced choice models. The estimation of such models typically relies on simulation methods with a large number of random draws to obtain stable results. *mixl* uses inherent properties of the loglikelihood problem structure to greatly reduce both the memory usage and runtime of the estimation procedure for specific types of mixed multinomial logit models. Functions for prediction and posterior analysis are included. Parallel computing is also supported, with near linear speedups observed on up to 24 cores. *mixl* is directly accessible from R, and easy to use. This paper presents the architecture and performance of the package, details its use, and presents some results using real world data and models.

Keywords: Multinomial logit, mixed logit, choice modelling, R, hybrid choice, estimation

1. Introduction

Choice modelling is an important tool in many fields, including transportation, marketing, behavioural psychology, and economics. For example in a transportation context, individuals must make many repeated decisions, whether to travel or not, which mode of transport to take, and which route or transit line to travel on. All of these decisions involve a choice of one alternative from multiple options: a discrete choice. Since the 1970's multinomial logit models have applied random utility theory to model how decision makers compare and evaluate alternatives (McFadden, 1974). However, for a closed-form solution some mathematical constraints of the decision need to be accepted. However, these restrictions, although computationally convenient,

limit the realism of the models. As such, increasingly more complex models and model families have been proposed, including Mixed MNL (McFadden and Train, 2000), Generalized Extreme Value (McFadden, 1980) and Hybrid Choice Models (HCM) (Walker and Ben-Akiva, 2001; Ben-Akiva *et al.*, 2002). These models relax the behavioural restrictions on the classic MNL model, but in the process, require simulation methods to estimate.

In particular, mixed MNL and hybrid choice models are increasingly being used to explain human behaviour (e.g. Schmid and Axhausen, 2018). Additionally, the size of the datasets that researchers are working with are also becoming larger. Hence, it is now common that complex model formulations can take many hours or even days or weeks to estimate when using simulation methods. There are widely used software packages for estimating such models, such as *biogeme* (Bierlaire, 2016) and *ALOGIT*. Ideally, the model estimation process integrates seamlessly with the workflow of the modeller, which is commonly the R language and Rstudio. Publicly available R software packages for choice model estimation are limited both in functionality and by their reliance on the R language, making them both inefficient and unable to handle large problems when using simulation. As such, there is a need for an open-source R software that can estimate complex choice models on large datasets.

This paper presents a new R package, *mixl*, for the estimation of multinomial logit models (MNL), mixed MNL models (MMNL) and hybrid choice models (HCM), arguably currently the three most common model structures. In Section 2.1, the necessary background on log-likelihood computation and estimation, including the available software for doing so, is covered. Section 3 describes the software architecture and design decisions behind the *mixl* package. Section 5 explains how to use the package, and the final section presents an application of the software.

2. Background

2.1. (Simulated) Maximum loglikelihood estimation

In this section the formulation of the log-likelihood calculation are presented. Let us assume that each decision maker n is trying to maximize their utility in the form $U_{nj} = V_{nj} + \epsilon_{nj}$ where V_{nj} is the observed utility and ϵ_{nj} represents the unobserved factors (Ben-Akiva and Lerman, 1987). This leads to the following succinct closed formed expression for the choice probability (Train, 2009):

$$P_{nit} = \frac{\exp(x_{nit}\beta)}{\sum_j \exp(x_{njt}\beta)} \quad (1)$$

where n is the decision maker, i the chosen alternative in choice scenario t , and T_n is the number of choice tasks for individual n . β represents the model parameters to be estimated, and X_{ijt} the vector of observed variables. In the mixed case with panel data, the formulation is extended with the distributions $f(\beta)$:

$$L_n = \int \prod_t^{T_n} P_{nit} f(\beta|\theta) d\beta \quad (2)$$

where θ are the parameters that describe the density of β . These mixed models can be formulated in several ways, with two main derivations using random coefficients and/or error components. They vary over decision makers with the density $f(\beta)$, which is, as mentioned, a function of the parameters θ . As such, in a mixed model, the parameters β vary over the population.

The calculation of the likelihood for a simple MNL model is straightforward. The likelihood is simply the product of the chosen probabilities for each individual. For panel data, the logs of the probabilities are summed up over all observations for each individual. To calculate the probabilities in a mixed model, R values β^r are drawn from $f(\beta|\theta)$ and used to calculate the likelihood. In equation 1, β is hence replaced by β_r to calculate P_{nit}^r . The simulated probability \hat{P}_n for individual n is given by the average over all R draws, and the simulated log-likelihood (SLL) for the sample follows:

$$\hat{P}_n = \frac{1}{R} \sum_r^R \prod_t^{T_n} P_{nit}^r \quad (3)$$

$$\hat{LL} = \sum_n \ln(\hat{P}_n) \quad (4)$$

2.2. Maximum likelihood estimation

The process of estimating the optimal value of the parameters β' is called maximum likelihood estimation (MLE), or maximum simulated likelihood estimation (MSLE). In this process, an optimization routine tries to find the set of parameters that give the maximum log-likelihood by repeatedly

calculating the log-likelihood with different β' until the process converges. To decide which betas to try next, the routine needs the derivatives of $f(\beta')$. The most common approach, when an analytical gradient is not available is to calculate a numerical gradient:

$$f'(\beta') = \frac{f(\beta' - \Delta) + f(\beta' + \Delta)}{2\Delta} \quad (5)$$

To calculate the gradient at each optimization step, the objective function, namely the log-likelihood, needs to be calculated $2k + 1$ times, where k is the number of free parameters to be estimated. However, it is worth noting that the only parameter to $f(\beta')$ that ever changes is β' itself.

2.3. Overview of available choice modelling software

There is a wide variety of software available for choice model estimation, including but not limited to *biogeme* (Bierlaire, 2016), the *mlogit* package for R (Hasan *et al.*, 2014), *ALOGIT* (ALOGIT, 2016), and *NLOGIT* (Greene, 2002). Each of these packages handle various model types and mixed models. However, three of the four do not fit in with the work-flow of many researchers and modellers, namely being able to work completely in R, and the second two packages are proprietary, each with their own unique syntax. Table 2.3 summarizes the attributes of the packages.

Arguably the leading open source software for discrete choice modeling is *biogeme*. *pythonBiogeme* can estimate an extremely wide range of parametric models. It has been under development for many years and is robust and stable software. The user can specify arbitrary utility functions and the likelihood formulation. Additionally, it takes advantage of both compilation to C++ and automatic derivation to achieve excellent performance. However, its incredible flexibility comes with a challenging learning curve. Additionally, as stand-alone software, it is not yet interfactable from R. One disadvantage of *biogeme* is the need to prepare the data in a specific format, which is prone to errors.

In the R universe, the *mlogit* package provides the most accessible tools for working with MNL models. It also supports mixed logit, but struggles with larger problem sizes. The *mnlogit* package provides significant speed improvements over *mlogit* by optimizing the calculation of the Hessian matrix and using the Newton-Raphson method for MSLE. However, Nocedal and Wright (2000) observed that quasi-Newton methods such as BFGS perform better on larger MSLE problems. Additionally, the *mnlogit* package does

not support mixed models, and the utility function is only allowed to have linear-in-parameter specifications.

Furthermore, both these R packages are restricted by their reliance on the R *formula* package and syntax for specifying the utility function. For more complicated models with many alternatives or latent variables, this syntax is at best cumbersome, and sometimes insufficient. Also, *mnlogit* also does not support random coefficients. Both R packages do not support hybrid choice or other more advanced model formulations.

The two most well known proprietary packages, *ALOGIT* and *NLOGIT*, are well established, but are neither open-source, nor freely available to researchers.

	Open-source	R	Mixed models	HCM	Large problems
biogeme	yes		yes	yes	yes
R mlogit	yes	yes	yes		
R mnlogit	yes	yes			yes
ALOGIT			yes		yes
NLOGIT			yes		yes

Table 1: Comparison of main software packages for multinomial logit modelling

2.4. Limitations and potentials of the R language

The open-source statistical software *R* is an incredibly powerful and popular platform for data processing, analysis and visualization. However, it is well known for its liberal use of memory, and sometimes unwieldy syntax. A particular performance bottleneck in R is iteration. *for* loops have a significant overhead in the R language. A such, for many common operations, functions available in R and its packages are written in the programming language C++ for better performance, or ‘vectorized’ to work on vectors or matrices to avoid R-based iteration.

The *Rcpp* package (Eddelbuettel and Francois, 2011) is most commonly used to improve the performance of R scripts by rewriting critical functions in C++. Functions written in Rcpp accept and return R datatypes such as Vectors and Matrices. Code written in C++ and called from R is often many times faster than the equivalent R code. However, C++ code must be compiled before execution, either when the package is created, or inside

the script itself, in which case the user requires a software compiler on their machine.

3. Software Architecture

In section 2.2 it was noted that while the utility function must be calculated many times during the MSLE process, all the data used by the function except the parameters to be estimated do not change. Furthermore, each observation can be seen as independent from a calculation perspective. For every observation, the utility of each alternative is calculated. From there the log of the probability of the chosen alternative is simple to calculate, which are then summed over each individual for repeated observations. Since this operation is associative, it also doesn't matter in which order the observations are processed.

This fact is exploited to drastically reduce the memory that the package uses during estimation. Instead of storing the log-probability results for every single observation, a matrix P of size $n \times R$ where n is the number of individuals and R is the number of draws used. For datasets with a large panel structure, this saves a significant amount of memory (a factor equal to the average number of observations per individual).

However, this requires using iteration constructs rather than vectorized linear algebra operations, as the datasets are of different sizes. As mentioned in Section 2.4, this results in a debilitating performance bottleneck in the R language. The solution is to code the log-likelihood function in C++. However, it then follows that $f(\beta')$ must also be written and compiled in C++.

In an effort to avoid this, the package includes a pre-compiler that takes a model specification written in plain text, and converts it to a C++ utility function callable from R and optimisation routines. Section 4.1 details how specifications need to be written. The pre-compiler validates the specification against the dataset to check that all variables are present, and automatically identifies model properties such as mixed effects and hybrid choice components. *mixl* detect errors in the model specification and reports them to the user.

3.1. Parallelization

Since P_{ni} is calculated for each observation separately, the calculation of the log-likelihood is an *embarrassingly parallel problem* which can be handled

efficiently using data parallelism. While there are packages to perform data parallelism in R, for example *parallel* and *foreach*, they include significant communication overheads as new processes are spawned. Since in *mixl* the log-likelihood function is implemented in C++, we can take advantage of the openMP (Chapman and Massaioli, 2005) framework to efficiently parallelize the for-loop over observations. Since all data except the intermediate utilities of each alternative are shared between all cores, no copying of the data across cores is needed to run the log-likelihood function in parallel. Compiling with the openMP framework even provides a performance boost on a single core, due to certain optimizations the framework enables in the compiler.

4. Using the *mixl* package

4.1. Syntax

To aid the specification of models and improve error reporting to the modeller, a small amount of syntax is required:

- Variables from the dataset must be prefixed with a \$
- Coefficients with a @
- Every statement must end with a ;
- Intermediate variables that are calculated dont get prefixed by anything
- The utility functions are prefixed by U_x, where x is the choice id. U_1 or U_2 are, for example, valid.
- Draws are prefixed by *draw*. Passing nDraws parameter into the max-Likelihood function, a set of draws will be generated automatically. Currently, this defaults to a halton sequence. Alternatively, any set of draws (Sobol, MLHS (Hess *et al.*, 2006), etc) can be passed in as an argument, as long as the matrix is large enough to accommodate the number of individuals and random parameters.
- Standard mathematical functions such as addition, multiplication, exponentiation, and equality comparisons are allowed.

The syntax is best explained with a small example based on the classic Train dataset available in the *mlogit* package:

```

ASC_A_RND = @ASC_A + draw_1 * @SIGMA_A1 + draw_2;
ASC_B_RND = @ASC_B + draw_3 * @SIGMA_B;

U_pt = ASC_A_RND + @B_price * $price_A + @B_timeA * $time_A / 60
      + @B_change * $change_A;

U_car = ASC_B_RND + @B_price * $price_B + @B_timeB * $time_B / 60;

```

In this example all the words prefixed with @ (ASC_A, SIGMA_A1, ...) are parameters to be estimated. Those with \$ are variables of the observations available in the data. In this example there are three random parameters required, indicated by the draw_ variables. Two intermediate variables are also calculated, ASC_A_RND and ASC_B_RND, which are then used in the two utility functions. The _RND suffix indicates that these are random coefficients, for which posteriors can be calculated. These specifics and more details on the syntax are covered in the user-guide supplied with the package. The availabilities must be supplied as a separate matrix, with one row for each observation, and one column for each alternative used in the model specification. The package also provides functions for prediction and the calculation of posteriors for random variables.

Unlike in *biogeme*, the likelihood function cannot be modified by the user. However, for a large range of problems, it is sufficient to encode the behaviour in the utility functions, and use a standard log-likelihood function for panel data, as described in Section 2.1.

4.2. Estimation

For estimation, *mixl* wraps the optimisation routine from the *maxLik* package (Henningsen and Toomet, 2011). As Train (2009) suggests, the BFGS (Witzgall and Fletcher, 1989) optimization procedure is used as default. The interface is designed so that all possible parameters to *maxlik* can be passed through, including the choice of modeller, hessian function, and a limit on the number of iterations. The fixing of parameter values is also supported.

4.3. Estimation of hybrid choice models

The package also supports hybrid choice models. If models include variable definitions with the prefix P_indic_, the model is assumed to have hybrid

components, and the `P_indic_` variables will be considered as probability indicators for each observation. The pre-compiler detects these automatically, and generates the code to include the product of the probability indicators in the log-likelihood as such:

```
p_choice = log(chosen_utility / sum(utilities));
p_indic_total = P_indic_1 * P_indic_2 * .... P_indic_k;
p_choice = p_choice + (1/count) * log(p_indic_total);
```

The count variable, used to normalize the choice indicator, represents number of choice observation per individual and must also be included in the data. On convergence, the model estimation will return both the choice log-likelihood and the model log-likelihood. One extra column is required in the dataset to enable hybrid choice, namely a ‘count’ column with the total number of observations for the individual making the choice.

4.4. Post processing

The *mixl* package provides some key post processing functions for working with an estimated model. The estimation results include all the expected components, such as the (robust) covariance matrix, table of coefficients, standard errors, hessian matrix. The predict function generated for each model allows different scenarios to be tested. For models with mixed distributions, posteriors can also easily be calculated. Random variables in the model specification are automatically detected (those with an equation including a draw), and the posterior function returns a labeled matrix of the posteriors for each individual and random variable. This can all be done without leaving the R environment, as the results are always returned as either matrices or dataframes.

5. Example

In this section we present the iterative development of a model, starting with a basic MNL model, followed by a mixed MNL model. This provides a good example of how the successively more advanced models can be iteratively developed using the *mixl* package.

```

1 library(mixl)
2 data('Train', package='mlogit')
3 head(Train, 3)
4 Train$ID <- Train$id
5 Train$CHOICE <- as.numeric(Train$choice)
6
7 mnl_test <- '
8   U_PT = @B_price * $price_A / 1000 + @B_time * $time_A / 60 + @B_change * $change_A;
9   U_Car = ASC_B + @B_price * $price_B / 1000 + @B_timeB * $time_B / 60;
10  '
11
12 model_spec <- specify_model(mnl_test, Train)
13
14 est <- setNames(c(0.1,0.1,0.1,0.1,0.1,0.1),
15               c('B_price', 'B_time', 'B_timeB', 'B_change', 'ASC_B', 'SIGMA_B'))
16
17 availabilities <- mixl::generate_default_availabilities(
18               Train, model_spec$num_utility_functions
19               )
20
21 model <- estimate(model_spec, est, Train, availabilities)
22 summary(model)

```

Using the mode choice dataset from the *mlogit* package, it is straightforward to set up the data for our model in lines 2-5. Only the ID and CHOICE variables need to be converted to continuous values starting from 1. We then specify a straight forward model as a string of text in R as follows (7-11). The model has two alternative modes, public transport (PT) and car. PT has one extra attribute, namely the number of transfers (change_A). Variables in the dataset are prefixed with \$ and β to be estimated with @. We then call *specify_model* to convert this model specification to a log-likelihood function. The dataset is passed in so that the variables in the model can be verified.

The starting values are specified on line 15 and the availabilities on line 18. Here we use a function to indicate that both alternatives are always available. The *estimate* function is then called on the model specification, and the results presented to the user.

The output from a estimated model is presented in the console as follows:

Model diagnosis: successful convergence

Number of decision makers: 235

Number of observations: 2929

LL(null): -2030.228

LL(init): -2030.228

LL(final): -1842.251

Rho2: 0.093

Estimated parameters: 5

Estimates:

	est	se	trat_0	trat_1	robse	robtrat_0	robtrat_1	...
B_price	-1.0396	0.0599	-17.36	-34.05	0.1055	-9.86	-19.34	...
B_time	-0.8071	0.1415	-5.70	-12.77	0.1694	-4.76	-10.67	...
B_timeB	-0.9534	0.1508	-6.32	-12.95	0.1656	-5.76	-11.80	...
B_change	-0.1406	0.0576	-2.44	-19.82	0.0620	-2.27	-18.38	...
ASC_B	0.1979	0.1917	1.03	-4.18	0.1839	1.08	-4.36	...

To add mixing, some random components are added to the model specification, with the random draws prefixed with *draw_*. The specification adds mixing on the alternative-specific constant for car and on journey time, using two random parameters.

```
mnl_test <- `
  ASC_B.RND = @ASC_B + draw_1 * @SIGMA_B;
  TIME_A.RND = @B_timeA + draw_2 * @SIG_time;
  TIME_B.RND = @B_timeB + draw_2 * @SIG_time;

  U_PT = @B_price * $price_A / 1000 + TIME_A.RND * $time_A / 60 + @B_change * $change_A;
  U_Car = ASC_B.RND + ( @B_price * $price_B / 1000 + TIME_B.RND * $time_B / 60);
`
```

The same code as before is used to estimate the model with the addition of the required number of halton draws (in this example 20) which has to be defined in the estimate function. By including just two random parameters, a scale and a sigma, the log-likelihood improves by almost 20 units. Table 2 shows the results of the MMNL model, outputted using a function provided in *mixl* using the *texreg* package.

6. Performance

Since the random draws are generated per individual and not per choice task (of which an individual may make multiple), there are two possible

	Mixed MNL
B_price	-1.10*** (0.11)
B_timeA	-0.85*** (0.21)
B_timeB	-1.05*** (0.21)
B_change	-0.17** (0.06)
ASC_B	0.28 (0.20)
SIGMA_B	-0.07 (0.07)
SIG_time	1.89*** (0.35)
# estimated parameters	7
Number of respondents	235
Number of choice observations	2929
LL(null)	-2030.23
LL(final)	-1824.96
McFadden R2	0.101
AIC	3663.92
AICc	3664.41
BIC	3705.79

*** $p < 0.01$, ** $p < 0.05$, * $p < 0.1$

Table 2: Latex model output from a mixed MNL model esitmatd with *mixl*

approaches to combining the fixed and random variables. To take advantage of R vectorization, the draws must be replicated for each choice task of an individual. However as mentioned, this approach breaks down for large panel datasets, especially when the number of draws increases. The approach used in the *mixl* package avoids this, by accessing the required draws using the ID of the individual. Hence, as seen in Table 3 the memory usage increases to store the larger draw matrix and the output probability matrix. For smaller problems the memory is bound by other requirements of the program such as the compiler. Essentially, the program is not limited by the number of individuals or repeated choices in the dataset, number of random dimensions or the number of draws used.

Draws	10	100	1000	10,000
Memory (MB)	530	534	538	2097

Table 3: Memory usage for the Train example data with different numbers of draws

With the widespread availability of multi-core machines and computing clusters, multi-core scalability is an important consideration. *mixl* achieves consistent speedups even on large numbers of cores. Figure 1 and Table 4 show the performance of the isolated log-likelihood function for different numbers of draws over an increasing number of cores. An average of 50 repetitions for each configuration is taken. For nearly all draw configurations, large speedups are observed. For the 10 draw configuration, communication costs dominate and a maximum speedup of 4.78x is observed. As the number of draws used is increased, so do the benefits of using parallel computing. On 24 processing cores for 10,000 draws, a speedup of 19.3x is observed. It is worth noting that only the utility calculation has been parallelized, and still more potential remains in other parts of the log-likelihood function to improve this figure slightly. The results in Table 4 show a super-linear speedup in the 4 core case. This can be attributed to cache-effects on the processor.

Figure 5 illustrates the runtime for the estimation of a large MMNL model on a large panel dataset with 491 individuals and 17120 observations. The model contained 27 free parameters, 15 utility functions and 12 random parameters. With the previous R-only code (CMC, 2017), estimation with 1000 draws took 3 days and 16 hours and required 27 gigabytes of memory. Estimation with 10,000 draws would have taken at least 30 days. Compared to

Draws	Processors				
	2	4	8	16	24
10	1.56	2.81	2.73	3.51	4.78
100	1.92	3.83	5.09	6.96	14.76
500	1.94	4.04	5.91	9.09	17.40
1000	1.95	4.04	6.16	9.60	18.49
5000	1.96	4.11	6.28	10.22	19.58
10,000	1.97	4.21	6.31	10.25	19.35

Table 4: Speedup over multiple cores for the log-likelihood calculation

the results in Table 4, the speedup over multiple cores is lower, due to outer optimization loop, which is not coded in C++. Again, larger problem sizes extract better relative performance from more cores. The multicore feature of *mixl* is automatically available on Linux machines. for computers running windows and OSX, a short one time setup process is required.

Draws	Processors					
	1	2	4	8	16	24
10	00:09:14	00:04:01 (2.3)	00:03:03 (3.02)	00:02:01 (4.57)	00:01:36 (5.76)	00:01:11 (7.76)
100	01:40:52	00:51:09 (1.97)	00:27:16 (3.7)	00:18:33 (5.44)	00:14:23 (7.01)	00:08:10 (12.33)
1000	17:00:51	07:55:16 (2.15)	04:39:28 (3.65)	02:53:57 (5.87)	01:59:29 (8.54)	01:13:23 (13.91)
10,000	135:09:02	81:52:51 (1.65)	51:18:38 (2.63)	25:45:06 (5.25)	16:00:17 (8.44)	10:10:01 (13.29)

Table 5: *mixl* performance on a large dataset, speedup in brackets

7. Conclusion

This paper presents the *mixl* R package for estimating multinomial logit models. Mixed models and hybrid choice models are supported through a flexible and intuitive syntax. The package has been designed to have an intuitive model specification syntax, and is engineered with both large datasets

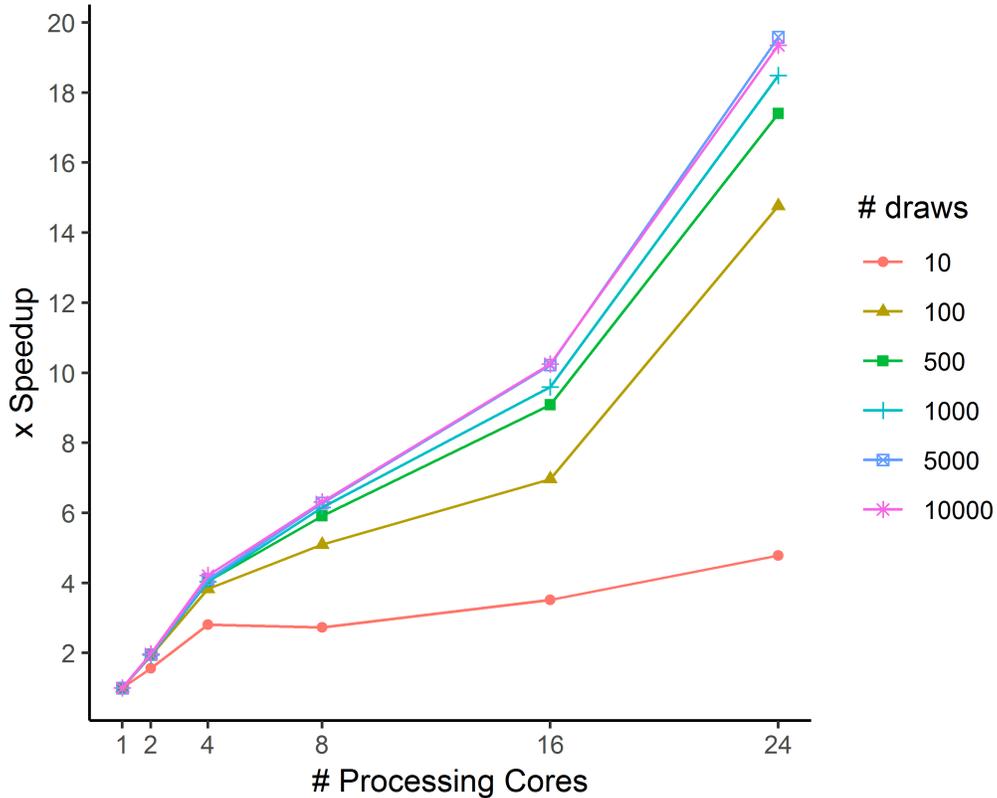


Figure 1: Performance of the log-likelihood function over multiple cores

and complex mixed models in mind. There is no practical limit to either the size of the dataset or the number random draws that can be used, as both estimation and memory usage scale linearly with the number of draws. For large problems, parallel computing is an attractive way to gain significant speed increases, and *mixl* achieves speedups of 14x using 24 cores. The paper presents performance indicators on a complex mixed MNL model estimated on a large dataset with over 17,000 observations. The package has also already been used in modelling projects (among them, Schmid *et al.*, 2017) with hundreds of thousands of observations and 10,000 random draws, indicating its robustness and scalability. Future work will aim to integrate the work with other estimation packages and support more model types such as the probit kernel.

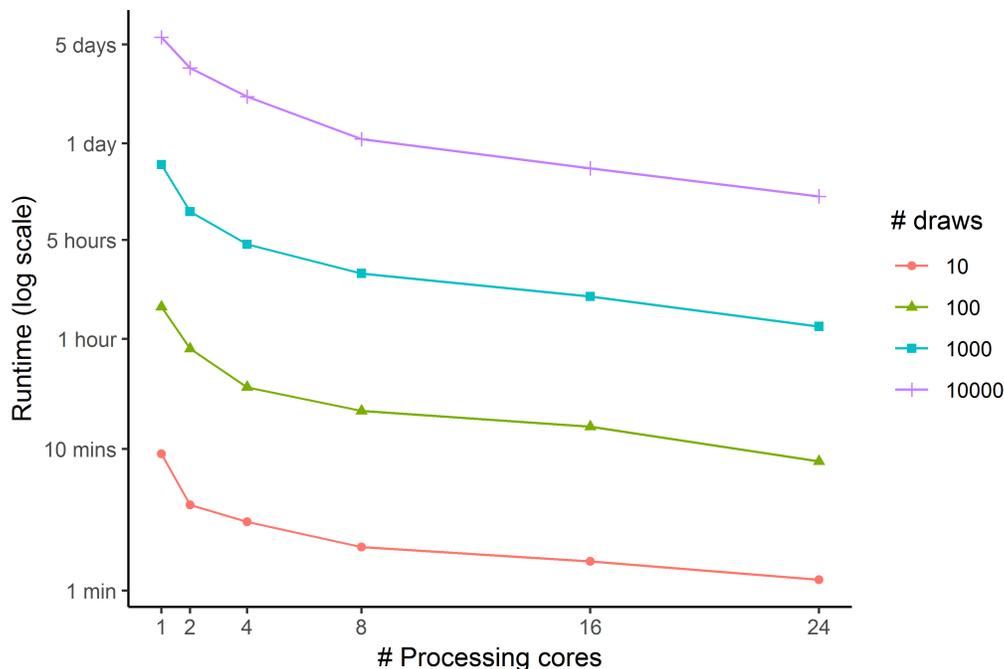


Figure 2: Estimation runtime of a large model with different numbers of draws

8. Acknowledgements

The Authors would like to thank Kay W. Axhausen for his input and comments. Additional thanks goes to Stephane Hess, who's original R script for discrete choice modeling was the inspiration for this R package. The work was undertaken with the financial support of SNF and the German Railways.

Downloading the package

The estimation software is provided as an R package on the code sharing website, Github. To install the code, first make sure the devtools package is installed. Then run the following command:

```
devtools::install_github('joemolloy/fast-mixed-mnl')
```

References

- ALOGIT (2016) ALOGIT Software & Analysis Ltd.
- Ben-Akiva, M., D. McFadden, K. E. Train, J. Walker, C. Bhat, M. Bierlaire, D. Bolduc, A. Boersch-Supan, D. Brownstone, D. S. Bunch, A. Daly, A. de Palma, D. Gopinath, A. Karlstrom and M. a. Munizaga (2002) Hybrid Choice Models : Progress and Challenges, *Marketing Letters*, **13** (3) 163–175.
- Ben-Akiva, M. E. and S. R. Lerman (1987) *Discrete Choice Analysis: Theory and Application to Predict Travel Demand*.
- Bierlaire, M. (2016) PythonBiogeme : a short introduction, TRANSP-OR 160706, Series on Biogeme, *Technical Report*, Transport and Mobility Laboratory, EPFL, Switzerland.
- Chapman, B. M. and F. Massaioli (2005) OpenMP, *Parallel Computing*, **31** (10) 957–1174.
- CMC (2017) CMC Choice modelling code for R, *Technical Report*, Choice Modelling Centre, University of Leeds.
- Eddelbuettel, D. and R. Francois (2011) Rcpp: Seamless R and C ++ integration, *Journal Of Statistical Software*, **40** (8) 1–18.
- Greene, W. H. (2002) NLOGIT reference guide : version 3.0.
- Hasan, A., W. Zhiyu and A. S. Mahani (2014) Fast Estimation of Multinomial Logit Models: R Package mnlogit, *Journal of statistical software*, **75** (3).
- Henningsen, A. and O. Toomet (2011) MaxLik: A package for maximum likelihood estimation in R, *Computational Statistics*, **26** (3) 443–458.
- Hess, S., K. E. Train and J. W. Polak (2006) On the use of a Modified Latin Hypercube Sampling (MLHS) method in the estimation of a Mixed Logit Model for vehicle choice, *Transportation Research Part B: Methodological*.
- McFadden, D. (1974) Conditional logit analysis of qualitative choice behavior, in P. Zarembka (ed.) *Frontiers in Econometrics*, 105–142, Academic Press.

- McFadden, D. (1980) Econometric Models of Probabilistic Choice among Products, *Journal of Business*, **53** (3) 13–29.
- McFadden, D. and K. Train (2000) Mixed MNL models for discrete response, *Journal of Applied Econometrics*, **15** (5) 447–470.
- Nocedal, J. and S. J. Wright (2000) *Numerical optimization 2nd edition*, Springer.
- Schmid, B., F. Aschauer, S. Jokubauskaite, S. Peer, R. Hssinger, R. Gerike, S. R. Jara-Diaz and K. W. Axhausen (2017) A pooled RP/SP mode, route and destination choice model to capture the heterogeneity of mode and user-type effects, paper presented at the *5th International Choice Modeling Conference (ICMC)*, Capetown.
- Schmid, B. and K. W. Axhausen (2018) In-store or online shopping of search and experience goods: A hybrid choice approach, *Journal of Choice Modelling (In Press)*.
- Train, K. E. (2009) *Discrete choice methods with simulation*, Cambridge university press.
- Walker, J. and M. Ben-Akiva (2001) Extensions of the Random Utility Model, *Technical Report*, MIT.
- Witzgall, C. and R. Fletcher (1989) Practical Methods of Optimization., *Mathematics of Computation*.