Joice: A Mixed Logit Estimation Package for Julia

Connor Forsythe

November 2023

Abstract

The programming language Julia does not have a general discrete choice modelling software package that is widely available. Joice is a new package that fills this exact void. Freely available via GitHub, users of Julia can now estimate both simple logit models as well as mixed logit models with normally-distributed heterogeneous parameters. Performance as well as functionality for Joice currently lags behind other major logit estimation software. However, future development might shrink this performance gap.

Introduction

Choice Modelling Background

Choice modelling has been mentioned as an important tool in several contexts (Train, 2009, p.1). The logit model is an extremely popular discrete choice modelling tool (Train, 2009, p. 34), and the mixed logit model relaxes several assumptions present in the logit model (Train, 2009, p. 134). The estimation of mixed logit models can be achieved through the use of simulated maximum likelihood, which involves the sampling from parameter distributions in order to estimate a complex integral required for the likelihood calculation (Train, 2009, Ch. 6). Recent work has suggested that the number of points necessary to accurately and predictably estimate these models is much higher than than what has traditionally been used (Czajkowski and Budziński, 2019). Specifically, the authors of the aforementioned study find that the number of simulation draws necessary for certain levels of precision can extend anywhere from hundreds to tens of thousand depending on the characteristics of the data (Czajkowski and Budziński, 2019, Table 5). Czajkowski and Budziński (2019, p.83) point to the potential computational concerns of these additional integration points and notes that their provided MATLAB implementation performs better than several other implementations of which there are many.[1]

The current state of the art when it comes to the estimation of logit and mixed logit models has a wide breadth of options. Molloy et al. (2021, Table 1) provides a summary of popular logit estimation software offerings, which span across various languages and capabilities. Naturally, when researchers are deciding upon modelling tools, they must balance the utility of the additional cost of learning a new language or software package with the efficiency and capability of the software. Those who do not know a language with a discrete choice modelling software are particularly subject to the costs of learning a new language. One language that is not mentioned in Molloy et al. (2021) as having a discrete choice modelling software package is Julia.

Julia Background

Julia is a programming language that is relatively new as it has only been in development since 2009 (Bezanson et al., 2017, p.74). Julia is believed to be a solution to the "two language problem" (Bezanson et al., 2017, p.67-68). The two language problem is one where a developer initially develops a piece of software in an easy to interpret but slow language, and shifts to a much faster language for final implementation (Perkel, 2019, p.142). Bezanson et al. (2017, p. 68) write that the two language problem is addressed through several Julia features, such as multiple dispatch, just-in-time compilation and others. To my knowledge, there is not an openly-available Julia package that can be used for logit and mixed logit estimation.[2] Therefore, there

[1]The implementation used in Czajkowski and Budziński (2019) can be found at https://github.com/czaj/dce.

[2]There is a script that purports to do this that I haven't tested available at https://github.com/ighdez/MultiLogit. There are related Julia packages: the MixedModels package estimates generalized linear mixed models (Bates et al., 2021), the pack-

are no choice-modelling packages that are able to utilize the potential benefit of the speed enabled by the Julia programming language. In order to fill this void in the marketplace of Julia software, I will introduce a new package that is capable of estimating both simple and mixed logit models with an easy to use interface.

Joice Features, Implementation, and Usage

Joice Features and Implementation

Joice is package developed under Julia 1.6.3 that can be readily utilized in any Julia environment.[3] Joice is available for download at https://github.com/crforsythe/Joice-Releases. Joice has many features that one would expect in a logit choice-modelling software. Joice is able to estimate logit and mixed logit models (as described in Train (2009)) with arbitrarily-specified linear utility functions. Currently the heterogeneity estimated in mixed logit models is limited to normal distributions with a mean and standard deviation parameter.[4] Numerical integration of the log-likelihood function is performed using Sobol draws that have been scrambled, which was decided upon because of their superior performance characteristics (Czajkowski and Budziński, 2019). The Sobol draws are generated from the scipy Python package's "Sobol" function (Virtanen et al., 2020).[5] Further, Joice enables users to specify panel structure for proper characterization of dependence amongst observations.[6] Finally, Joice calculates both standard variance-covariance matrices as well as robust variance-covariance matrices.[7] Currently, all code is written using only a single processor in mind.[8]

Joice is reliant upon various packages that are already available to the Julia community, which are listed in the description of the package where their functionality is being leveraged. However, I would like to specifically call attention to the package enabling optimization functionality in Joice. Optimization is central to the estimation process and Joice utilizes JuMP as its optimization interface (Dunning et al., 2017). Actual optimization of the maximum likelihood utilizes JuMP by leveraging it as an interface to the nlopt solver (Johnson, n.d.).[9] Building Joice with JuMP functionality will allow for the easy investigation of altnerative solver performance for future development.[10]

Interfacing With Joice

`ModelInfo` Struct

Using Joice centers largely around a struct entitled `ModelInfo`. With an instantiation of a `ModelInfo` struct, the user can specify all of the necessary parameters in order to estimate a given model. In order to facilitate the easy and safe construction of `ModelInfo` structs, Joice gives users access to the function `ConstructModelInfo`. The set of arguments that an individual can pass to the `ConstructModelInfo` function to create a given instance of a `ModelInfo` struct are described in table 1.

age SossMLJ (available at https://github.com/cscherrer/SossMLJ.jl) is able to estimate multinomial logistic regression models using a Bayesian framework (example code found at https://cscherrer.github.io/SossMLJ.jl/stable/example-multinomial-logistic-regression/), and FRAC implements the estimation strategies found in Salanié and Wolak (2019) using code found at https://github.com/jamesbrandecon/FRAC.jl. I do not consider these packages to be choice modelling packages as they either are not focused on general discrete choice modelling or are not formatted as a package for easy distribution, extension, and interface.

[3] Package was constructed using the PkgTemplates package (available at https://github.com/invenia/PkgTemplates.jl).

[4] The Distributions Julia package was used to execute many statistical calculations in the estimation process (Lin et al., 2021).

[5] Utilizing the scipy Python package within Julia requires Conda package (available at https://github.com/JuliaPy/Conda.jl) and the PyCall package (Available at https://github.com/JuliaPy/PyCall.jl).

[6] See Hess and Rose (2009, Sec 2.2) for a discussion of data dependence in panel settings.

[7] Robust covariance formulation is based on the implementation on the logitr R package which I contributed to and is based on the implementation in Stata (Helveston, 2021; StataCorp, 2021). Hessians are numerically estimated using the FiniteDiff package available at https://github.com/JuliaDiff/FiniteDiff.jl.

[8] Although I have not personally written code to enable multiprocessing, CPU utilization loads at certain times exceed a single process load. This suggests that some multiprocessing capability is being used by a dependency.

[9] This is the same solver used by the logitr package (Helveston, 2021). A Julia interface for the nlopt solver can be found at https://github.com/JuliaOpt/NLopt.jl.

[10] JuMP-compatible solvers can be found at https://jump.dev/JuMP.jl/stable/installation/.

Table 1: Descriptions of arguments used in `ConstructModelInfo` function. Joice and other parts of this submission require and make generous use of the DataFrames package (available at https://github.com/JuliaData/DataFrames.jl).

Parameter	Type(s) Accepted	Optional	Description
Data	DataFrame	False	"Long" formatted data to be used in estimation.
ChoiceColumn	String, Symbol	False	Column within Data where choice indicators are recorded.
Parameters	String, Symbol, Vector of Strings or Symbols, Nothing	False	Column names within Data to be included in linear utility function
ChoiceSetIDColumn		False	Column within Data where IDs associated with each choice set are recorded.
RandomParameters	A dictionary with String of Symbol keys with String values, Nothing	True	Columns names within Data to be included in linear utility with respective random parameterization. Currently only supports normal distributions ("n").
PanelIDColumn	String, Symbol, Nothing	True	Column within Data where IDs associated with each panel observation are recorded.
Space	String	False	Currently not in use. Will be used in future development.
Name	String, Nothing	True	Model name
InterDrawCount	Int	True	Number of draws used to simulated log-liklihood in mixed logit models. Defaults to 2048.
WeightColumn	String, Symbol, Nothing	True	Currently not in use. Will be used in future development.

Estimating Models

After the construction of a `ModelInfo` struct. One can then call the function `EstimateModel`. The function `EstimateModel` will either take in both a `ModelInfo` struct and a vector of starting points or just a `ModelInfo` struct. When both potential arguments are passed, the package will begin the optimization routine from the given starting point. When the `EstimateModel` function takes in a constructed `ModelInfo` struct alone, Joice will decide upon a set of starting values for estimation. For simple logit models, a vector of zeroes will be used as the starting values. For mixed logit models, starting mean values will be coefficients of the same model that has been estimated assuming no heterogeneity and all standard deviation parameter starting values will be 0.3.[11] Using these or the user-specified starting values, an initial estimation step begins with relatively loose tolerances. After the initial estimation, estimates are refined using higher tolerances. From there, final gradient values, observation gradients[12], hessian, and covariance estimation occurs. The estimation of observation gradients, hessian, and covariance estimates can be skipped by passing a value of `false` for the optional `post` argument of `EstimateModel`.

Joice Estimation Outputs

If a solution is found, an `EstimateModel` call returns a dictionary with several relevant outputs from the estimation procedure. This includes the final log-likelihood (or simulated log-likelihood) from the estimation procedure with the key "LL", the coefficient vector ("coef") as a named tuple[13], non-robust and robust standard errors as well as covariance matrices ("se", "robustSE", "cov", "robustCov"), and finally, the

[11] This is just as is done in the Stata implementation (StataCorp, 2021), which can be seen in the provided Stata log files.

[12] The ProgressMeter Julia package (available at https://github.com/timholy/ProgressMeter.jl) is used to keep track of progress in calculation of observation gradients.

[13] A simple vector of coefficients will be returned if `post` is equal to `false`.

numerically estimated hessian ("hessian") along with observation-level and overall gradients are returned ("obsGrad", "grad"). If no solution is found, the `EstimateModel` call will return a `nothing` value.

Performance Relative to Other Software Packages

Data Generation

In order to test the performance of various software packages against Joice, I construct several simulated datasets.[14] These datasets are simulated based on a data generating process that yields several individual decision makers making repeated decisions across a set of varying alternatives. Across all datasets, there are ten choice sets and decisions per individual, and each choice set has three alternatives. There are then three datasets used for estimation that vary by the number of individuals - 100, 500, and 1,000 individuals are simulated.

The utility of each alternative is governed by a linear utility function. These linear utility functions factor in six attributes. This function can be seen in eq. (1) where ν represents latent utility, attribute k is denoted by x^k (each attribute is different and these superscripts do *not* represent exponentiation), β_k is the linear coefficient associated with attribute k. These variables are indexed by individual i, choice set t, or alternative j. All attribute levels are simulated from a standard uniform distribution.

$$\nu_{itj} = \beta_1 x^1_{itj} + \beta_2 x^2_{itj} + \beta_3 x^3_{itj} + \beta_{4i} x^4_{itj} + \beta_{5i} x^5_{itj} + \beta_{6i} x^6_{itj} + \epsilon_{itj} \tag{1}$$

Preferences for x^1, x^2, and x^3 are fixed across the population with values of -1, 0, and 1 respectively. Preferences for x^4, x^5, and x^6 are heterogeneous across individuals. These preferences are distributed normally with a given mean and standard deviation as shown in eq. (2). All these data can be simulated using the submitted notebook entitled "DataGeneration", which makes use of the pandas, scipy, and numpy Python packages (Harris et al., 2020; pandas development team, 2020; Virtanen et al., 2020; Wes McKinney, 2010).

$$\begin{aligned} \beta_4 &\sim \mathrm{N}\left(-1, 1\right) \\ \beta_5 &\sim \mathrm{N}\left(0, \frac{1}{4}\right) \\ \beta_6 &\sim \mathrm{N}\left(1, 1\right) \end{aligned} \tag{2}$$

Performance Results

In order to test the performance of Joice, I compare the time it takes to estimate a mixed logit model that mirrors the data generating specification (described in "Data Generation" section) across various logit estimation software package alternatives. The number of integration points is 2,048 across all software and runs.[15] Specifically, I test Joice against the R language packages Apollo (Hess and Palma, 2019, 2021) and mlogit (Croissant, 2020), as well as the estimation routines contained within the Stata language (StataCorp, 2021). Full details for the software configurations are in table 2.[16] Every run was performed within a Jupyter notebook[17] or Stata IDE on a machine with the characteristics shown in table 3. Finally, the packages IJulia (available at https://github.com/JuliaLang/IJulia.jl) as well as IRkernel (available at https://github.com/IRkernel/IRkernel) enabled the use of Julia and R within Jupyter notebooks.

[14] This simulation procedure is heavily influenced by various simulation examples as well as the notes provided in Alex Davis' Fall 2021 19-786 "Stochastic Discrete Choice Models: Estimation and Behavioral Theory" class.

[15] 2048 was decided upon because 1) the scipy Python package warns that balance properties of Sobol draws can only be guaranteed when the number of points is a power of 2 (Virtanen et al., 2020) (discussion can be found at https://scipy.github.io/devdocs/reference/generated/scipy.stats.qmc.Sobol.html) and 2) Czajkowski and Budziński (2019, Fig. 2) suggest that 2,000 integration points are required to ensure null hypothesis rejection (or failure to reject) is not a product of simulation error. However, the exact data specification being investigated in Czajkowski and Budziński (2019, Fig. 2) is not represented by the data that I generate. Therefore, this exact requirement might not hold in the situation investigated in this paper.

[16] Note, there is no version associated with cmxtmixlogit as it is packaged with the Stata language and doesn't receive its own version control scheme.

[17] Information on Jupyter notebooks can be found at https://jupyter.org/

Before discussion of the final results, it is important to note that there were many steps taken to minimize the difference between the estimation procedures across software packages to ensure comparability. First, all algorithms only make use of a single processing unit.[18] Further, I attempt to standardize the complete estimation process by utilizing the same starting point procedure across algorithms. I utilize the same baseline as Stata (StataCorp, 2021), which has been re-implemented in Joice. This involves multiple estimation steps for mlogit and Apollo where I first estimate the simple logit version of the model and pass those estimated coefficients as part of the starting coefficient vector when estimating the mixed logit version. When timing estimation with Apollo, the post-estimation time required for the simple logit estimation step is not included when calculating the overall time to estimate the full model. For mlogit, it is impossible to separate the time taken to complete post-estimation steps, so those are included in overall estimation times.[19] All packages utilize a BFGS-like algorithm to solve the optimization problem.[20] Finally, tolerances were set to make termination conditions as similar as possible. Available termination criteria were not entirely consistent across algorithms, so specifications may differ slightly across software. Exact criteria conditions can be viewed in the provided notebooks, Stata script, and Joice GitHub repo.

Table 2: Logit estimation software details

Package/Command	Package/Command Version	Language	Language Version
Joice	0.1.0	Julia	1.6.3
Apollo	0.2.5	R	4.1.1
mlogit	1.1-1	R	4.1.1
cmxtmixlogit	—	Stata	17

Table 3: Machine characteristics

Model	2017 MacBook Pro (15”)
Operating System	macOS 12.0.1
Processor	2.8 GHz Quad-Core Intel Core i7
RAM	16 GB 2133 MHz LPDDR3

I present two runs of each estimation for reasons of compilation in the Julia programming language. One of the primary features of the Julia language is that it is a compiled language (Bezanson et al., 2017, p.68). Therefore, there is an initial performance cost of compilation that shouldn’t be taken “seriously” and is not present in an additional run (The Julia Project, 2021, Sec. 34.2).[21] This has led to some benchmarks reporting both first and second runs (H20.ai, 2021; Queryverse, 2020).[22]

[18] I’ll again note that Joice seems to utilize more than one processing unit at certain points of estimation. I believe this to be due to some dependency. Any comparison between algorithms should take this into account, and Joice results should be considered optimistic.

[19] Timing of mlogit estimation uses the timeR R package (available at https://github.com/yusuzech/timeR).

[20] nlopt, the solver Joice utilizes, doesn’t have a traditional BFGS algorithm. Rather, a “Low-storage BFGS” is used, which is described in more detail at https://nlopt.readthedocs.io/en/latest/NLopt_Algorithms/. Apollo uses a simple BFGS (Hess and Palma, 2021, p.29). mlogit defaults to BFGS per Croissant (2020, p.39). However, my testing (and attached notebooks) show that mlogit forces the Newton-Raphson method to be used with simple logit models. Stata uses BFGS as noted in pg. 2 of the “rmaximize” documentation found at https://www.stata.com/manuals/rmaximize.pdf.

[21] This section can be found under the heading “Performance Tips” at https://docs.julialang.org/en/v1/manual/performance-tips/.

[22] The URLs associated with these benchmarks can be found at https://h2oai.github.io/db-benchmark/ and https://www.queryverse.org/benchmarks/ respectively.

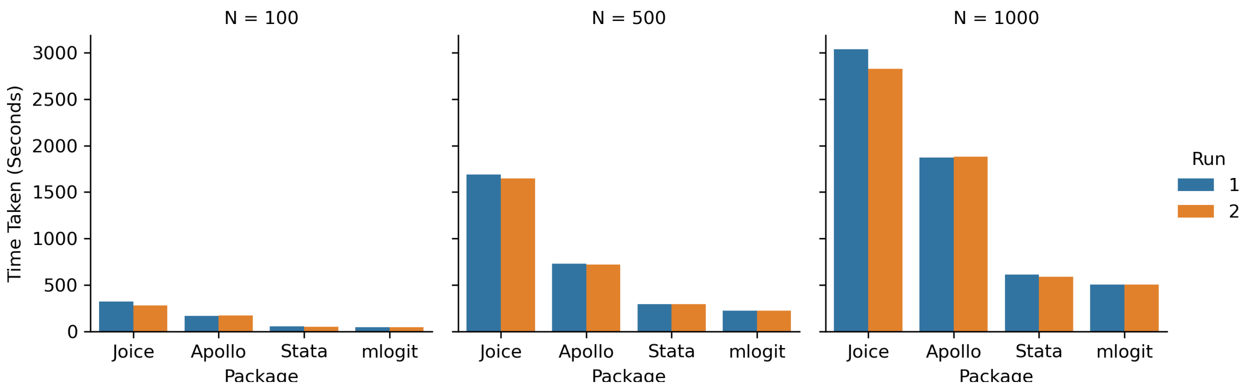

Figure 1: Timing results comparing across package as well as first and second estimation runs. Plotting done using the matplotlib and seaborn Python packages (Hunter, 2007; Waskom, 2021).

The timing results are presented in fig. 1. It is quite obvious that Joice's performance is lacking in terms of speed relative to other widely-available logit estimation packages. Consistently, the fastest Joice run is more than five times slower than the fastest run across all other alternatives in all situations. mlogit consistently outperforms the other software packages in terms of convergence speed. The exact reason for Joice's slow speed has yet to be addressed, but is likely a product of my lack of experience with the Julia language and ability to maximize efficiency. For instance, contributors to a recent blog post on https://discourse.julialang.org/ purports to have found and corrected Julia coding inefficiencies that lead to Julia performance improvements in an early version of Duarte et al. (2020).[23] It is almost undoubtedly the case that the Joice codebase contains many inefficiencies, which could be the source of the poor performance.

Although Joice took significantly longer to estimate, the estimation results were largely consistent across algorithms. I don't present final results here, but they can be viewed in the notebooks and log files provided with the submission. Final parameter vectors were different across packages, but remained fairly close to one another. There are many reasons as to why results are not exactly the same across packages. For one, differences in random sampling procedure. Both Joice and Apollo estimating procedures utilized Sobol draws with Owen scrambling, but mlogit uses Halton draws (Croissant, 2020, p. 30), and Stata defaults to Hammersley draws (StataCorp, 2021).[24] Additionally, any difference in optimization settings, implementation, or starting points would yield a different final solution. Still, very comparable simulated log-likelihood values were reached across all packages. However, I will note that, throughout my testing of the package, Joice occasionally had difficult reaching local maxima when the number of integration points was relatively small. This is likely due to the poor approximation of the integral and fairly tight tolerances being set.

Conclusions and Future Work

Conclusion

Joice is a new software package available for Julia users that can estimate both simple logit and mixed logit models for both cross-sectional and panel data. This is a new contribution to the Julia community, and due to it being open-source, will enable other Julia users to build upon the codebase and improve Joice. Currently Joice is slower than other packages that are commonly used, but it is still able to replicate results from other software packages. As evidenced by their documentation (Croissant, 2020; Hess and Palma, 2021; StataCorp, 2021),[25] other choice estimation software studied in this paper have many features that are not currently implemented in Joice. Still, Joice serves as a start to bringing choice-modelling software to the

[23]The blog post, is available at https://discourse.julialang.org/t/julia-slower-than-matlab-python-no/33128. At the onset of this project, this blog post also inspired me to investigate an alternative logit calculation algorithm than what is currently used in Joice. Those results can be found in the "AlternativeLogitAlgorithm" notebook. Within that notebook, the Optim Julia package is used (K Mogensen and N Riseth, 2018).

[24]You can find details on Stata's mixed logit estimation strategy at https://www.stata.com/manuals/cmcmxtmixlogit.pdf.

[25]Specific documentation for Stata's panel mixed logit command can be found at https://www.stata.com/manuals/cmcmxtmixlogit.pdf.

Julia community.

Future Work

There is plenty of work that is left to be done on this project. This package was built with my admittedly limited knowledge of and experience with the Julia programming language. Therefore, it is undoubtedly the case that there are areas for efficiency improvements that I was unable to address due to either time constraints or my own ignorance. Further, there are several functionality improvements (e.g. non-linear utility specification, alternative heterogeneity specifications, and others) that would greatly improve the package as it is. I hope to continue to develop Joice, and its open-source nature will allow others to contribute as well. Further, I'll note that there were several packages that Joice was not compared against. Future work should be done to compare those other packages (e.g. those listed in Molloy et al. (2021, Table 1) but not tested in this paper) against Joice's performance.

Additional Acknowledgements

First and foremost, this package has been developed with the help of many past discussions I have had with John P. Helveston as I contributed to his logit estimation package in R logitr (Helveston, 2021). Further, my experience using logitr as well as other choice modelling software has greatly influenced this package. These other packages include Apollo (Hess and Palma, 2019, 2021), Stata's choice modelling commands (StataCorp, 2021), the mlogit package (Croissant, 2020), as well as Train's estimation code.[26] Joice's interface as well as parts of its implementation have been influenced by the aforementioned packages. I cannot enumerate every way in which these packages have influenced various parts of Joice, but there are surely many similarities between Joice and those software packages listed.

Mixed logit simulated log-likelihood gradient calculations used in this package are based on formulations provided by Revelt and Train (1998), simulated log-likelihood values were also based on formulations in Revelt and Train (1998) as well as Alex Davis' Fall 2021 19-786 "Stochastic Discrete Choice Models: Estimation and Behavioral Theory" class. Simple logit log-likelihood calculation is based on a formulation given in Alex Davis' Fall 2021 19-786 "Stochastic Discrete Choice Models: Estimation and Behavioral Theory" class and gradient calculations are based on the formulation in Davis and De la Maza (2018, eq. 9.14).

The PyCharm IDE by JetBrains as well as the .ignore and Julia plugins for PyCharm were used in the development of this package.[27] This package also makes extensive use of coding practice recommendations from both the Julia and JuMP documentation. Specifically, the Julia "Performance Tips" and the JuMP non-linear "Tips and tricks".[28] Finally, the CSV Julia package was used to read CSV files for estimation purposes.[29]

[26]Train's estimation code is available at https://eml.berkeley.edu/ train/software.html

[27]Information for these software can be found at https://www.jetbrains.com/pycharm/, https://plugins.jetbrains.com/plugin/7495–ignore, and https://plugins.jetbrains.com/plugin/10413-julia respectively.

[28]Julia's Performance Tips can be found at https://docs.julialang.org/en/v1/manual/performance-tips/, and JuMP's non-linear tips and tricks are avaialable at https://jump.dev/JuMP.jl/stable/tutorials/nonlinear/tips_and_tricks/.

[29]The CSV Julia package can be found at https://github.com/JuliaData/CSV.jl.

References

Bates, Douglas, Phillip Alday, Dave Kleinschmidt, PhD José Bayoán Santiago Calderón, Likan Zhan, Andreas Noack, Alex Arslan, Milan Bouchet-Valat, Tony Kelman, Antoine Baldassari, Benedikt Ehinger, Daniel Karrasch, Elliot Saba, Jacob Quinn, Michael Hatherly, Morten Piibeleht, Patrick Kofod Mogensen, Simon Babayan, and Yakir Luc Gagnon, "JuliaStats/MixedModels.jl: v4.5.0," November 2021.

Bezanson, Jeff, Alan Edelman, Stefan Karpinski, and Viral B. Shah, "Julia: A fresh approach to numerical computing," *SIAM Review*, 2017, *59* (1), 65–98.

Croissant, Yves, "Estimation of Random Utility Models in R: The mlogit Package," *Journal of Statistical Software*, 2020, *95* (11), 1–41.

Czajkowski, Mikołaj and Wiktor Budziński, "Simulation error in maximum likelihood estimation of discrete choice models," *Journal of Choice Modelling*, 2019, *31* (April), 73–85.

Davis, Alex and Cristobal De la Maza, *19-786 Stochastic Discrete Choice Models* 2018.

Duarte, Victor, Diogo Duarte, Julia Fonseca, and Alexis Montecinos, "Benchmarking machine-learning software and hardware for quantitative economics," *Journal of Economic Dynamics and Control*, 2020, *111*, 103796.

Dunning, Iain, Joey Huchette, and Miles Lubin, "JuMP: A Modeling Language for Mathematical Optimization," *SIAM Review*, 2017, *59* (2), 295–320.

H20.ai, "Database-like ops benchmark," 2021.

Harris, Charles R., K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant, "Array programming with NumPy," *Nature*, September 2020, *585* (7825), 357–362.

Helveston, John Paul, *logitr: Random Utility Logit Models with Preference and Willingness to Pay Space Parameterizations* 2021. R package version 0.4.0.

Hess, Stephane and David Palma, "Apollo: a flexible, powerful and customisable freeware package for choice model estimation and application," *Journal of Choice Modelling*, 2019, *32.*

_ and _ , "Apollo version 0.2.5 user manual," 2021.

_ and John M. Rose, "Allowing for intra-respondent variations in coefficients estimated on repeated choice data," *Transportation Research Part B: Methodological*, 2009, *43* (6), 708–719.

Hunter, J. D., "Matplotlib: A 2D graphics environment," *Computing in Science & Engineering*, 2007, *9* (3), 90–95.

Johnson, Steven G., "The NLopt nonlinear-optimization package."

K Mogensen, Patrick and Asbjørn N Riseth, "Optim: A mathematical optimization package for Julia," *Journal of Open Source Software*, 2018, *3* (24), 615.

Lin, Dahua, Simon Byrne, John Myles White, Andreas Noack, Mathieu Besançon, David Widmann, Douglas Bates, John Pearson, John Zito, Alex Arslan, Kevin Squire, Moritz Schauer, David Anthoff, Theodore Papamarkou, Jan Drugowitsch, Benjamin Deonovic, Avik Sengupta, Giuseppe Ragusa, Glenn Moynihan, Brian J Smith, Martin O'Leary, Michael, Mike J Innes, Christoph Dann, Gustavo Lacerda, Iain Dunning, Jiahao Chen, Mohamed Tarek, and Tamas K. Papp, "JuliaStats/Distributions.jl: v0.25.23," October 2021.

Molloy, Joseph, Felix Becker, Basil Schmid, and Kay W. Axhausen, "mixl: An open-source R package for estimating complex choice models on large datasets," *Journal of Choice Modelling*, 2021, *39* (April 2020), 100284.

pandas development team, The, "pandas-dev/pandas: Pandas," February 2020.

Perkel, Jeffrey M., "Julia: come for the syntax, stay for the speed," *Nature*, 2019, *572* (7767), 141–142.

Queryverse, "Queryverse benchmarks," 2020.

Revelt, David and Kenneth Train, "Mixed logit with repeated choices: Households' choices of appliance efficiency level," *Review of Economics and Statistics*, 1998, *80* (4), 647–657.

Salanié, Bernard and Frank Wolak, "Fast, "Robust", and Approximately Correct: Estimating Mixed Demand Systems," Technical Report, National Bureau of Economic Research, Cambridge, MA apr 2019.

StataCorp, “Stata Statistical Sofware: Release 17,” 2021.

The Julia Project, “The Julia Language,” 2021.

Train, Kenneth, *Discrete Choice Methods with Simulation*, 2 ed., Cambridge University Press, 2009.

Virtanen, Pauli, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors, “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python,” *Nature Methods*, 2020, *17*, 261–272.

Waskom, Michael L., “seaborn: statistical data visualization,” *Journal of Open Source Software*, 2021, *6* (60), 3021.

Wes McKinney, “Data Structures for Statistical Computing in Python,” in Stéfan van der Walt and Jarrod Millman, eds., *Proceedings of the 9th Python in Science Conference*, 2010, pp. 56 – 61.