Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *http://www.jstatsoft.org/*

Fast Estimation of Multinomial Logit Models: R Package mnlogit

Asad Hasan
Scientific Computing
Sentrana Inc.

Wang Zhiyu
Dept of Mathematical Sciences
Carnegie Mellon University

Alireza S. Mahani
Scientific Computing
Sentrana Inc.

Abstract

We present R package **mnlogit** for estimating multinomial logistic regression models, particularly those involving a large number of categories and variables. Compared to existing software, **mnlogit** offers speedups of 10-50 times for modestly sized problems and more than 100 times for larger problems. Running in parallel mode on a multicore machine gives up to 4 times additional speedup on 8 processor cores. **mnlogit** achieves its computational efficiency by drastically speeding up computation of the loglikelihood function's Hessian matrix through exploiting structure in matrices that arise in intermediate calculations. This efficient exploitation of intermediate data structures allows **mnlogit** to utilize system memory much more efficiently, such that for most applications **mnlogit** requires less memory than comparable software by a factor that is proportional to the number of model categories.

Keywords: logistic regression, multinomial logit, discrete choice, large scale, parallel, econometrics.

1. Introduction

Multinomial logit regression models, the multiclass extension of binary logistic regression, have long been used in econometrics in the context of modeling discrete choice (McFadden 1974; Bhat 1995; Train 2003) and in machine learning as a linear classification technique (Hastie, Tibshirani, and Friedman 2009) for tasks such as text classification (Nigam, Lafferty, and McCallum 1999). Training these models presents the computational challenge of having to compute a large number of coefficients which increases linearly with the number of categories and the number of variables. Despite the potential for multinomial logit models to become computationally expensive to estimate, they have an intrinsic structure which can be exploited to dramatically speedup estimation. Our objective in this paper is twofold: first we describe

how to exploit this structure to optimize computational efficiency, and second, to present an implementation of our ideas in our R (R Core Team 2013) package **mnlogit** which is available from CRAN at: `http://cran.r-project.org/web/packages/mnlogit/index.html`.

An older method of dealing with the computational issues involved in estimating large scale multinomial logistic regressions has been to approximate it as a series of binary logistic regressions (Begg and Gray 1984). In fact the R package **mlogitBMA** (Sevcikova and Raftery 2013) implements this idea as the first step in applying Bayesian model averaging to multinomial logit data. Large scale logistic regressions can, in turn, be tackled by a number of advanced optimization algorithms (Komarek and Moore 2005; Lin, Weng, and Keerthi 2008). A number of recent R packages have focused on slightly different aspects of estimating regularized multinomial logistic regressions. For example: package **glmnet** (Friedman, Hastie, and Tibshirani 2010) is optimized for obtaining the entire $L1$-regularized paths and uses the coordinate descent algorithm with 'warm starts', package **maxent** (Jurka 2012) is intended for large text classification problems which typically have very sparse data and the package **pmlr** (Colby, Lee, Lewinger, and Bull 2010) which penalizes the likelihood function with the Jeffrey's prior to reduce first order bias and works well for small to medium sized datasets. There are also R packages which estimate plain (unregularized) multinomial regression models. Some examples are: the **VGAM** package (Yee 2010), the `multinom` function in package **nnet** (Venables and Ripley 2002) and package the **mlogit** (Croissant 2012).

Of all the R packages previously described, **mlogit** is the most versatile in the sense that it handles many data types and extensions of multinomial logit models (such as nested logit, heteroskedastic logit, etc.). These are especially important in econometric applications, which are motivated by the utility maximization principle (McFadden 1974), where one encounters data which depends upon both the observation instance and the choice class. Our package **mnlogit** provides the ability of handling these general data types while adding the advantage of very quick computations. This work is motivated by our own practical experience of the impossibility of being able to estimate large scale multinomial logit models using existing software.

In **mnlogit** we perform maximum likelihood estimation (MLE) using the Newton-Raphson (NR) method. We speed up the NR method by exploiting structure and sparsity in intermediate data matrices to achieve very fast computations of the Hessian of the loglikelihood function. This overcomes the NR method's well known weakness of incurring very high per-iteration cost, compared to algorithms from the quasi-Newton family (Nocedal 1992, 1990). Indeed classical NR estimations of multinomial logit models (usually of the Iteratively Reweighted Least Square family) have been slow for this very reason. On a single processor our methods have allowed us to achieve speedups of 10-50 times compared to **mlogit** on modest-sized problems while performing identical computations. In parallel mode[1], **mnlogit** affords the user an additional speedup of 2-4 times while using up to 8 processor cores.

We provide a simple formula-based interface for specifying a varied menu of models to **mnlogit**. Section 2 illustrates aspects of the formula interface, the expected data format and the precise interpretations of variables in **mnlogit**. To make the fullest use of **mnlogit** we suggest that the user understand the simple R example worked out over the course of this section. Section 3 and Appendix A contain the details of our estimation procedure, emphasizing the ideas that

[1]Requires **mnlogit** to be compiled with OpenMP support (usually present by default with most R installations, except on Mac OS X).

underlie the computational efficiency we achieve in **mnlogit**. In Section 4 we present the results of our numerical experiments in benchmarking and comparing **mnlogit**'s performance with other packages while Appendix C has a synopsis of our timing methods. Finally Section 5 concludes with a short discussion and a promising idea for future work.

2. On using mnlogit

The data for multinomial logit models may vary with both the choice makers ('individuals') and the choices themselves. Besides, the modeler may prefer model coefficients that may (or may not) depend on choices. In **mnlogit** we try to keep the user interface as minimal as possible without sacrificing flexibility. We follow the interface of the `mlogit` function in package **mlogit**. This section describes the **mnlogit** user interface, emphasizing data preparation requirements and model specification via an enhanced formula interface. To start, we load the package **mnlogit** in an R session:

```
R> library("mnlogit")
```

2.1. Data preparation

mnlogit accepts data in the 'long' format which requires that if there are K choices, then there be K rows of data for each individual (see also Section 1.1 of the **mlogit** vignette). Here is a snapshot from data in the 'long' format on choice of recreational fishing mode made by 1182 individuals:

```
R> data("Fish", package = 'mnlogit')
R> head(Fish, 8)
```

```
           mode   income     alt   price  catch chid
1.beach   FALSE 7083.332   beach 157.930 0.0678    1
1.boat    FALSE 7083.332    boat 157.930 0.2601    1
1.charter  TRUE 7083.332 charter 182.930 0.5391    1
1.pier    FALSE 7083.332    pier 157.930 0.0503    1
2.beach   FALSE 1250.000   beach  15.114 0.1049    2
2.boat    FALSE 1250.000    boat  10.534 0.1574    2
2.charter  TRUE 1250.000 charter  34.534 0.4671    2
2.pier    FALSE 1250.000    pier  15.114 0.0451    2
```

In the 'Fish' data, there are 4 choices ('beach', 'boat', 'charter', 'pier') available to each individual: labeled by the 'chid' (chooser ID). The 'price' and 'catch' column show, respectively, the cost of a fishing mode and (in unspecified units) the expected amount of fish caught. An important point here is that this data varies both with individuals and the fishing mode. The 'income' column reflects the income level of an individual and does not vary between choices. Notice that the snapshot shows this data for two individuals.

The actual choice made by an individual, the 'response' variable, is shown in the column 'mode'. **mnlogit** requires that the data contain a column with exactly two categories whose

levels can be coerced to integers by `as.numeric()`. The greater of these integers is automatically taken to mean `TRUE`.

The only other column strictly mandated by **mnlogit** is one listing the names of choices (like column 'alt' in Fish data). However if the data frame is an `mlogit.data` class object, then this column may be omitted. In such cases **mnlogit** can query the `index` attribute of an `mlogit.data` object to figure out the information contained in the 'alt' column.

2.2. Model parametrization

Multinomial logit models have a solid basis in the theory of discrete choice models. The central idea in these discrete models lies in the 'utility maximization principle' which states that individuals choose the alternative, from a finite, discrete set, which maximizes a scalar value called 'utility'. Discrete choice models presume that the utility is completely deterministic for the individual, however modelers can only model a part of the utility (the 'observed' part). Stochasticity entirely arises from the unobserved part of the utility. Different assumptions about the probability distribution of the unobserved utility give rise to various choice models like multinomial logit, nested logit, multinomial probit, GEV (Generalized Extreme Value), mixed logit etc. Multinomial logit models, in particular, assume that unobserved utility is i.i.d. and follows a Gumbel distribution (see Train (2003), particularly Chapters 3 and 5, for a full discussion).

We consider that the observed part of the utility for the i^{th} individual choosing the k^{th} alternative is given by:

$$U_{ik} = \xi_k + \vec{X}_i \cdot \vec{\beta}_k + \vec{Y}_{ik} \cdot \vec{\gamma}_k + \vec{Z}_{ik} \cdot \vec{\alpha}. \tag{1}$$

Here Latin letters (X, Y, Z) stand for design matrices while Greek letters (ξ, α, β, γ) stand for coefficients. The parameter ξ_k is called the intercept. For many practical applications, variables in multinomial logit models can be naturally grouped into two types:

- **Individual specific variables** $\vec{X}_i$ which do not vary between choices (e.g., income of individuals in the 'Fish' data of Section 2.1).
- **Alternative specific variables** $\vec{Y}_{ik}$ **and** $\vec{Z}_{ik}$ which vary with alternative and may also differ, for the same alternative, between individuals (e.g., the amount of fish caught in the 'Fish' data: column 'catch').

In **mnlogit** we model these two types of variables with three types of coefficients:

1. Individual specific variables with alternative specific coefficients $\vec{X}_i \cdot \vec{\beta}_k$
2. Alternative specific variables with alternative independent coefficients $\vec{Z}_{ik} \cdot \vec{\alpha}$.
3. Alternative specific variables with alternative specific coefficients $\vec{Y}_{ik} \cdot \vec{\gamma}_k$.

The vector notation denotes that more than one variable of each type may be used to build a model. For example in the fish data we may choose both the 'price' and 'catch' with either alternative independent coefficients (the $\vec{\alpha}$) or with alternative specific coefficients (the $\vec{\gamma_k}$).

Due to the principle of utility maximization, only differences between utilities are meaningful. This implies that the multinomial logit model can not determine absolute utility. We must

specify the utility for any individual with respect to an arbitrary base value which we choose to be 0. In choice model theory this is called 'normalizing' the model.

For convenience in notation, we fix the choice indexed by $k = 0$ as the base; thus normalized utility is given by:

$$V_{ik} = U_{ik} - U_{i0} = \xi_k - \xi_0 + \vec{X}_i \cdot (\vec{\beta}_k - \vec{\beta}_0) + \vec{Y}_{ik} \cdot \vec{\gamma}_k - \vec{Y}_{i0} \cdot \vec{\gamma}_0 + (\vec{Z}_{ik} - \vec{Z}_{i0}) \cdot \vec{\alpha}. \tag{2}$$

Notice that the above expression implies that $V_{i0} = 0 \ \forall i$. To simplify notation we re-write the normalized utility as:

$$V_{ik} = \xi_k + \vec{X}_i \cdot \vec{\beta}_k + \vec{Y}_{ik} \cdot \vec{\gamma}_k - \vec{Y}_{i0} \cdot \vec{\gamma}_0 + \vec{Z}_{ik} \cdot \vec{\alpha} \qquad k \in [1, K-1] \tag{3}$$

This equation retains the same meaning as Equation 2. Notice the restriction $k \neq 0$, since we need $V_{i0} = 0$. The most important difference is that $\vec{Z}_{ik}$ in Equation 3 stands for $\vec{Z}_{ik} - \vec{Z}_{i0}$ (in terms of the original design matrices as expressed in Equation 2). In Equation 3 we also write $\vec{\beta}_k$ and ξ_k for $\vec{\beta}_k - \vec{\beta}_0$ and $\xi_k - \xi_0$, respectively. Similarly we make the replacements $\vec{\beta}_k$ and ξ_k for $\vec{\beta}_k - \vec{\beta}_0$ and $\xi_k - \xi_0$, respectively in Equation 3.

The utility maximization principle, together with the assumption on the error distribution, implies that for multinomial logit models (Train 2003) the probability of individual i choosing alternative k, P_{ik}, is given by:

$$\mathsf{P}_{ik} = \mathsf{P}_{i0} e^{V_{ik}}. \tag{4}$$

Here V_{ik} is the normalized utility given in Equation 3 and $k = 0$ is the base alternative with respect to which we normalize utilities. The number of available alternatives is taken as K which is a positive integer greater than one. From the condition that every individual makes a choice, we have that $\sum_{k=0}^{k=K-1} \mathsf{P}_{ik} = 1$. This gives us the probability of individual i picking the base alternative:

$$\mathsf{P}_{i0} = \frac{1}{1 + \sum_{k=1}^{K-1} e^{V_{ik}}}. \tag{5}$$

Note that $K = 2$ is the familiar binary logistic regression model.

Equation 3 has implications about which model parameters may be identified. In particular for alternative-specific coefficients of individual-specific variables we may only estimate the difference $\vec{\beta}_k - \vec{\beta}_0$. Similarly for the intercept only the difference $\xi_k - \xi_0$, and not ξ_k and ξ_0 separately may be estimated. For a model with K alternatives we estimate $K - 1$ sets of parameters $\vec{\beta}_k - \vec{\beta}_0$ and $K - 1$ intercepts $\xi_k - \xi_0$.

2.3. Formula interface

To specify multinomial logit models in R we need an enhanced version of the standard formula interface - one which is able to handle multi-part formulas. In **mnlogit** we built the formula interface using tools from the R package **Formula** (Zeileis and Croissant 2010). Our formula interface closely confirms to that of the **mlogit** package. We illustrate it with examples motivated by the 'Fish' dataset (introduced in Section 2.1). Consider a multinomial logit model where 'price' has an alternative independent coefficient, 'income' data being individual-specific has an alternative-specific coefficient and the 'catch' also has an alternative-specific coefficient. That is, we want to fit a model that has the 3 types of coefficients described in Section 2.2. Such a model can be specified in **mnlogit** with a 3-part formula:

```
R> fm <- formula(mode ~ price | income | catch)
```

By default, the intercept is included, it can be omitted by inserting a '-1' or '0' anywhere in the formula. The following formulas specify the same model with omitted intercept:

```
R> fm <- formula(mode ~ price | income - 1 | catch)
R> fm <- formula(mode ~ price | income | catch - 1)
R> fm <- formula(mode ~ 0 + price | income | catch)
```

We can omit any group of variables from the model by placing a 1 as a placeholder:

```
R> fm <- formula(mode ~ 1 | income | catch)
R> fm <- formula(mode ~ price | 1 | catch)
R> fm <- formula(mode ~ price | income | 1)
R> fm <- formula(mode ~ price | 1 | 1)
R> fm <- formula(mode ~ 1 | 1 | price + catch)
```

When the meaning is unambiguous, an omitted group of variables need not have a placeholder. The following formulas represent the same model where 'price' and 'catch' are modeled with alternative independent coefficients and the intercept is included:

```
R> fm <- formula(mode ~ price + catch | 1 | 1)
R> fm <- formula(mode ~ price + catch | 1)
R> fm <- formula(mode ~ price + catch)
```

2.4. Using package mnlogit

The complete `mnlogit` function call looks like:

```
R> mnlogit(formula, data, choiceVar = NULL, maxiter = 50, ftol = 1e-6, gtol
+          = 1e-6, weights = NULL, ncores = 1, na.rm = TRUE, print.level = 0,
+          linDepTol = 1e-6, start = NULL, alt.subset = NULL, ...)
```

We have described the 'formula' and 'data' arguments in previous sections while others are explained in the man page, only the 'linDepTol' argument needs further elaboration. Data used to estimate the model must satisfy certain necessary conditions so that the Hessian matrix, computed during Newton-Raphson estimation, is full rank (more about this in Appendix B). In **mnlogit** we use the R built-in function `qr`, with its argument 'tol' set to 'linDepTol', to check for linear dependencies. If collinear columns are detected in the data, using the criteria discussed in Appendix B, then some are removed so that the remaining columns are linearly independent.

We now illustrate the practical usage of `mnlogit` and some of its methods by a simple example. Consider the model specified by the formula:

```
R> fm <- formula(mode ~ price | income | catch)
```

This model has:

- One alternative independent coefficient 'price' corresponding to $\vec{\alpha}$.
- Two alternative specific coefficients 'income' and intercept corresponding to $\vec{\beta_k}$. We treat the intercept as the coefficient corresponding to an alternative specific 'variable' consisting of a vector of ones.
- One alternative specific coefficient 'catch' corresponding to $\vec{\gamma_k}$.

In the 'Fish' data the number of alternatives $K = 4$, so the number of coefficients in the above model is:

- One coefficient for a variable that may vary with individuals and alternatives, corresponding to $\vec{\alpha}$.
- $2 \times (K - 1) = 6$, alternative specific coefficients for individual specific variables (note: that we have subtract 1 from the number of alternative because after normalization the base choice coefficient can not be identified), corresponding to $\vec{\beta_k}$.
- $1 \times K = 4$ alternative specific coefficients for variables which may vary with individuals and alternatives, corresponding to $\vec{\gamma_k}$.

Thus the total number of coefficients in this model is $1 + 6 + 4 = 11$.

We call the function `mnlogit` to fit the model using the 'Fish' dataset on two processor cores.

```
R> fit <- mnlogit(fm, Fish, ncores=2)
R> class(fit)

[1] "mnlogit"
```

For `mnlogit` class objects we provide a number of S3 methods. Besides the usual methods associated with R objects (`coef`, `print`, `summary` and `predict`), we also provide methods such as: `residuals`, `fitted`, `logLik`, `vcov`, `df.residual`, `update` and `terms`. In addition, the returned object (here 'fit') can be queried for details of the estimation process by:

```
R> print(fit, what = "eststat")

-------------------------------------------------------------
Maximum likelihood estimation using the Newton-Raphson method
-------------------------------------------------------------
  Number of iterations: 7
  Number of linesearch iterations: 10
At termination:
  Gradient norm = 2.09e-06
  Diff between last 2 loglik values = 0
  Stopping reason: Succesive loglik difference < ftol (1e-06).
Total estimation time (sec): 0.038
Time for Hessian calculations (sec): 0.005 using 2 processors.
```

The estimation process terminates when any one of the three conditions 'maxiter', 'ftol' or 'gtol' is met. In case one runs into numerical singularity problems during the Newton iterations, we recommend relaxing 'ftol' or 'gtol' to obtain a suitable estimate. The plain Newton method has a tendency to overshoot extrema. In **mnlogit** we have included a 'line search' (one dimensional minimization along the Newton direction) which avoids this problem and ensures convergence (Nocedal and Wright 2000).

As a convenience, the `print` method may be invoked to query an `mnlogit` object for the number and type of model coefficients.

```
R> print(fit, what = "modsize")

Number of observations in data = 1182
Number of alternatives = 4
Intercept turned: ON
Number of parameters in model = 11
  # individual specific variables = 2
  # choice specific coeff variables = 1
  # individual independent variables = 1
```

Finally there is provision for hypothesis testing. We provide the function `hmftest` to perform the Hausman-McFadden test for IIA (Independence of Irrelevant Alternatives), which is the central hypothesis underlying multinomial logit models (Train 2003, Chap. 3). Three functions to test for hypotheses, applicable to any model estimated by the maximum likelihood method, are also provided:

- Function `lrtest` to perform the likelihood ratio test.
- Function `waldtest` to perform the Wald test.
- Function `scoretest` to perform the Rao score test.

These intent of these tests is succinctly described in Section 6 'Tests' of the **mlogit** package vignette and we shall not repeat it here. We encourage the interested user to consult the help page for any of these functions in the usual way, for example the `lrtest` help may be accessed by:

```
R> library("mnlogit")
R> ?lrtest
```

Functions `hmftest` and `scoretest` are adapted from code in the **mlogit** package, while `lrtest` and `waldtest` are built using tools in the R package **lmtest** (Zeileis and Hothorn 2002).

3. Estimation algorithm

In **mnlogit** we employ maximum likelihood estimation (MLE) to compute model coefficients and use the Newton-Raphson method to solve the optimization problem. The Newton-Raphson method is well established for maximizing the logistic family loglikelihoods (Hastie

et al. 2009; Train 2003). However direct approaches of computing the Hessian of multinomial logit model loglikelihood function have deleterious effects on the computer time and memory required. We present an alternate approach which exploits structure of the intermediate data matrices that arise in Hessian calculation to achieve the same computation much faster while using drastically less memory. Our approach also allows us to optimally parallelize Hessian computation and maximize the use of BLAS (Basic Linear Algebra Subprograms) Level 3 functions, providing an additional factor of speedup.

3.1. Maximizing the likelihood

Before going into details we specify our notation. Throughout we assume that there are $K \geq 3$ alternatives. The letter i labels individuals (the 'choice-makers') while the letter k labels alternatives (the 'choices'). We also assume that we have data for N individuals available to fit the model (N is assumed to be much greater than the number of model parameters). We use symbols in **bold face** to denote matrices, for example $\mathbf{H}$ stands for the Hessian matrix. To simplify our calculations we organize model coefficients into a vector $\vec{\theta}$. If the intercept is to be estimated then it is simply considered as another individual specific variable with an alternative specific coefficient but with the special provision that the 'data' corresponding to this variable is unity for all alternatives. The likelihood function is defined by $L(\vec{\theta}) = \prod_i \mathsf{P}\left(y_i|\vec{\theta}\right)$, where each y_i labels the alternative chosen by individual i. Now we have:

$$\mathsf{P}\left(y_i|\vec{\theta}\right) = \prod_{k=0}^{K-1} \mathsf{P}\left(y_i = k\right)^{\delta_{y_i k}}.$$

Here $\delta_{y_i k}$ is the Kronecker delta which is unity if $y_1 = k$ and zero otherwise. The likelihood function is given by: $L(\vec{\theta}) = \Pi_{i=1}^{N} L(\vec{\theta}|y_i)$. It is more convenient to work with the loglikelihood function which is given by $l(\vec{\theta}) = \log L(\vec{\theta})$. A little manipulation gives:

$$l(\vec{\theta}) = \sum_{i=1}^{N} \left[-\log\left(1 + \sum_{k=1}^{K-1} \exp(V_{ik})\right) + \sum_{k=1}^{K-1} V_{ik}\delta_{y_i k} \right]. \tag{6}$$

In the above we make use of the identity $\sum_k \delta_{y_i k} = 1$ and the definition of P_{i0} in Equation 5. McFadden (1974) has shown that the loglikelihood function given above is globally concave. We solve the optimization problem by the Newton-Raphson (NR) method which requires finding a stationary point of the gradient of the loglikelihood. Note that MLE by the Newton-Raphson method is the same as the Fisher scoring algorithm (Hastie *et al.* 2009; Li 2013). For our loglikelihood function 6, this point (which we name $\hat{\theta}$) is unique (because of global concavity) and is given by the solution of the equations: $\frac{\partial l(\vec{\theta})}{\partial \vec{\theta}} = \vec{0}$. The NR method is iterative and starting at an initial guess θ^{old} obtains an improved estimate θ^{new} by the equation:

$$\vec{\theta}^{new} = \vec{\theta}^{old} - \mathbf{H}^{-1}\frac{\partial l}{\partial \vec{\theta}}. \tag{7}$$

Here the Hessian matrix, $\mathbf{H} = \frac{\partial^2 l}{\partial \vec{\theta} \partial \vec{\theta}'}$ and the gradient $\frac{\partial l}{\partial \vec{\theta}}$, are both evaluated at $\vec{\theta}^{old}$. The vector $\delta\vec{\theta} = -\mathbf{H}^{-1}\frac{\partial l}{\partial \vec{\theta}}$ is called the full Newton step. In each iteration we attempt to update $\vec{\theta}^{old}$ by this amount. However if the loglikelihood value at the resulting $\vec{\theta}^{new}$ is smaller,

then we instead try an update of $\delta\vec{\theta}/2$. This linesearch procedure is repeated with half the previous step until the new loglikelihood value is not lower than the value at $\vec{\theta}^{old}$. Using such a linesearch procedure guarantees convergence of the Newton-Raphson iterations (Nocedal and Wright 2000).

3.2. Gradient and Hessian calculation

Each Newton-Raphson iteration requires computation of the Hessian and gradient of the loglikelihood function. The expressions for the gradient and Hessian are well known (see for example Section 2.5 of Croissant (2012) and Chapter 3 of Train (2003)). In their usual form they are given by:

$$\begin{aligned}\frac{\partial l}{\partial \vec{\theta}} &= \sum_i \sum_k \left(\delta y_i k - \mathsf{P}_{ik}\right) \tilde{X}_{ik} \\ \mathbf{H} &= -\tilde{\mathbf{X}}^\top \tilde{\mathbf{W}} \tilde{\mathbf{X}}\end{aligned} \tag{8}$$

For a model where only individual specific variables are used (that is only the matrix $\mathbf{X}$ contributes to the utility in Equation 3), the matrices $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{W}}$ are given by (Li 2013; Bohning 1992):

$$\tilde{\mathbf{X}} = \begin{pmatrix} \mathbf{X} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{X} & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \mathbf{0} & & \ddots & \vdots \\ \mathbf{0} & \cdots & & \mathbf{0} & \mathbf{X} \end{pmatrix},$$

here $\mathbf{X}$ is a matrix of order $N \times p$ (p is the number of variables or predictors) and,

$$\tilde{\mathbf{W}} = \begin{pmatrix} \mathbf{W_{11}} & \mathbf{W_{12}} & \cdots & \mathbf{W_{1,K-1}} \\ \mathbf{W_{21}} & \mathbf{W_{22}} & \cdots & \mathbf{W_{2,K-1}} \\ \vdots & \vdots & \cdots & \vdots \\ \mathbf{W_{K-1,1}} & \cdots & \cdots & \mathbf{W_{K-1,K-1}} \end{pmatrix}.$$

Here the sub-matrices $\mathbf{W_{k,t}}$ are diagonal matrices of order $N \times N$, where $\text{diag}(\mathbf{W_{k,t}})_i = \mathsf{P}_{ik}(\delta_{kt} - \mathsf{P}_{it})$ and δ_{kt} is the Kronecker delta which equals 1 if $k = t$ and 0 otherwise. Using this notation the gradient can be written as (Li 2013):

$$\frac{\partial l}{\partial \vec{\theta}} = \tilde{\mathbf{X}}^\top \left(\vec{y} - \vec{\mathsf{P}}\right)$$

Where we take vectors $\vec{y}$ and $\vec{\mathsf{P}}$ as vectors of length $N \times (K-1)$, formed by vertically concatenating the N probabilities P_{ik} and responses $I(y_i = k)$, for each $k \in [1, K-1]$. The Newton-Raphson iterations of Equation 7 take the form: $\vec{\theta}^{\text{new}} = \vec{\theta}^{\text{old}} + \left(\tilde{\mathbf{X}}^\top \tilde{\mathbf{W}} \tilde{\mathbf{X}}\right)^{-1} \tilde{\mathbf{X}}(\vec{y} - \vec{\mathsf{P}})$. Although in this section we have shown expressions for models with only individual specific variables, a general formulation of $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{W}}$ including the two other types of variables appearing in Equation 3 exists (and is implemented in the R packages **mlogit** (Croissant 2012) and **VGAM** (Yee 2010)). This is presented in Appendix B but their specific form is tangential to the larger point we make (our ideas extend to the general case in a simple way).

An immediate advantage of using the above formulation, is that Newton-Raphson iterations can be carried out using the framework of IRLS (Iteratively Re-weighted Least Squares) (Hastie *et al.* 2009, Section 4.4.1). IRLS is essentially a sequence of weighted least squares regressions which offers superior numerical stability compared to explicitly forming $\mathbf{H}$ and directly solving Equation 7 (Trefethen and Bau 1997, Lecture 19). However this method, besides being easy to implement, is computationally very inefficient. The matrices $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{W}}$ are huge, of orders $(K-1)N \times (K-1)p$ and $N(K-1) \times N(K-1)$ respectively, but are otherwise quite sparse and possess a neat structure. We now describe our approach of exploiting this structured sparsity.

3.3. Exploiting structure - fast Hessian calculation

We focus our attention on computation of the Hessian since it is the most expensive step, as we later show from empirical measurements in Table 1 of Section 4. We start by ordering the vector $\vec{\theta}$, which is a concatenation of all model coefficients as specified in Equation 3, in the following manner:

$$\vec{\theta} = \left\{ \vec{\beta}_1, \vec{\beta}_2 \ldots \vec{\beta}_{K-1}, \vec{\gamma}_0, \vec{\gamma}_1, \ldots \vec{\gamma}_{K-1}, \vec{\alpha} \right\}. \tag{9}$$

Here, the subscripts index alternatives and the vector notation reminds us there may be multiple variables modeled by coefficients of type $\vec{\beta}$, $\vec{\gamma}$, $\vec{\alpha}$. In $\vec{\theta}$ we group together coefficients corresponding to an alternative. This choice is deliberate and leads to a particular structure of the Hessian matrix of the loglikelihood function with a number of desirable properties.

Differentiating the loglikelihood function with respect to the coefficient vector $\vec{\theta}$, we get:

$$\frac{\partial l}{\partial \vec{\theta}_m} = \begin{cases} \mathbf{M_m}^\top \left(\vec{y}_m - \vec{\mathsf{P}}_m \right) & \text{if } \vec{\theta}_m \text{ is one of } \left\{ \vec{\beta}_1, \ldots \vec{\beta}_{K-1}, \vec{\gamma}_0, \ldots \vec{\gamma}_{K-1} \right\} \\ \sum_{k=1} \mathbf{Z_k}^\top \left(\vec{y}_k - \vec{\mathsf{P}}_k \right) & \text{if } \vec{\theta}_m \text{ is } \vec{\alpha} \end{cases} \tag{10}$$

Here we have partitioned the gradient vector into chunks according to $\vec{\theta}_m$ which is a group of coefficients of a particular type (defined in Section 2.2), either alternative specific or independent. Subscript m (and subscript k) indicates a particular alternative, for example:

- if $\vec{\theta}_m = \vec{\beta}_1$ then $m = 1$
- if $\vec{\theta}_m = \vec{\beta}_{K-1}$ then $m = K-1$
- if $\vec{\theta}_m = \vec{\gamma}_1$ then $m = 1$.

The vector $\vec{y}_m$ is a vector of length N whose i^{th} entry is given by $\delta_{y_i m}$, it tells us whether the observed choice of individual i is alternative m, or not. Similarly $\vec{\mathsf{P}}_m$ is vector of length N whose i^{th} entry is given by P_{im}, which is the probability individual i choosing alternative m. The matrices $\mathbf{M_m}$ and $\mathbf{Z_k}$ contain data for choice m and k, respectively. Each of these matrices has N rows, one for each individual. Specifically:

$$\begin{aligned} \mathbf{M_m} &= \mathbf{X} \qquad \text{if } \vec{\theta}_m \in \left\{ \vec{\beta}_1, \ldots \vec{\beta}_{K-1} \right\} \\ \mathbf{M_m} &= \mathbf{Y_m} \qquad \text{if } \vec{\theta}_m \in \left\{ \vec{\gamma}_0, \ldots \vec{\gamma}_{K-1} \right\}. \end{aligned}$$

Similarly, the matrices $\mathbf{Z}_k$ are analogues of the $\mathbf{Y}_m$ and have N rows each (note that due to normalization $\mathbf{Z_0} = \mathbf{0}$).

To compute the Hessian we continue to take derivatives with respect to chunks of coefficients $\vec{\theta}_m$. In doing this we can write the Hessian in a very simple and compact block format as shown below.

$$\mathbf{H_{nm}} = \frac{\partial^2 l}{\partial \vec{\theta}_n \partial \vec{\theta}_m'} = \begin{cases} -\mathbf{M_n}^\top \mathbf{W_{nm}} \mathbf{M_m} & \text{if } \vec{\theta}_n, \vec{\theta_m} \in \left\{\vec{\beta}_1, \ldots \vec{\beta}_{K-1}, \vec{\gamma}_0, \ldots \vec{\gamma}_{K-1}\right\} \\ -\sum_{k=1} \mathbf{M_n}^\top \mathbf{W_{nk}} \mathbf{Z_k} & \text{if } \vec{\theta}_n \in \left\{\vec{\beta}_1, \ldots \vec{\gamma}_{K-1}\right\} \ \& \ \vec{\theta}_m \text{ is } \vec{\alpha} \\ -\sum_{k,t=1} \mathbf{Z_k}^\top \mathbf{W_{kt}} \mathbf{Z_t} & \text{if } \vec{\theta}_n \text{ is } \vec{\alpha} \ \& \ \vec{\theta}_m \text{ is } \vec{\alpha} \end{cases} \tag{11}$$

Here $\mathbf{H_{nm}}$ is a block of the Hessian and the matrices $\mathbf{W_{nm}}$ are diagonal matrices of dimension $N \times N$, whose i^{th} diagonal entry is given by: $\mathsf{P}_{in}(\delta_{nm} - \mathsf{P}_{im})$.The details of taking derivatives in this block-wise fashion are given in Appendix A.

The first thing to observe about Equation 11 is that it effectively utilizes sparsity in the matrices $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{W}}$ to obtain very compact expressions for $\mathbf{H}$. The block format of the Hessian matrix is particularly suited for extremely efficient numerical computations. Notice that each block can be computed independently of other blocks with two matrix multiplications. The first of these involves multiplying a diagonal matrix to a dense matrix, while the second requires multiplication of two dense matrices. We handle the first multiplication with a handwritten loop which exploits the sparsity of the diagonal matrix, while the second multiplication is handed off to a BLAS (Basic Linear Algebra Subprograms) call for optimal efficiency (Golub and Loan 2013). We implement Hessian computation in a set of optimized C++ routines. Computing the Hessian block-by-block is very efficient since we can use level 3 BLAS calls (specifically `DGEMM`) to handle the most intensive calculations. Another useful property of the Hessian blocks is that because matrices $\mathbf{W_{nm}}$ are diagonal (hence symmetric), we have the symmetry property $\mathbf{H_{nm}} = \mathbf{H_{mn}}^\top$, implying that we only need to compute roughly half of the blocks.

Memory use optimization: The block structure of the Hessian requires only matrices $\mathbf{X}$ and $\mathbf{Y_m}$ instead of explicitly requiring large sparse matrices like $\tilde{\mathbf{X}}$. This results in huge memory savings as we only need to store the K matrices $\mathbf{Y_0}, \ldots \mathbf{Y_{K-1}}$ and one $\mathbf{X}$. For most applications where the design matrices are not dominated by $\mathbf{Z_m}$, this reduces memory requirement by a factor of K or more.

Independence of Hessian blocks leads to a very fruitful strategy for parallelizing Hessian calculations: we simply divide the work of computing blocks in the upper triangular part of the Hessian among available threads. This strategy has the great advantage that threads do not require any synchronization or communication overhead. However the cost of computing all Hessian blocks is not the same: the blocks involving alternative independent coefficients (the $\vec{\alpha}$) take much longer to compute. In **mnlogit** implementation, computation of the blocks involving alternative independent coefficients is handled separately from other blocks and is optimized for serial computation.

Hessian calculation is, by far, the most time consuming step in solving the multinomial logit MLE problem via the Newton-Raphson method. The choice we make in representing the Hessian in the block format of Equation 11 has dramatic effects on the time (and memory) it takes for model estimation. In the next section we demonstrate the impact on computation times of this choice when contrasted with earlier approaches.

4. Benchmarking performance

For the purpose of performance profiling **mnlogit** code, we use simulated data generated using a custom R function `makeModel` sourced from `simChoiceModel.R` which is available in the package inside folder `mnlogit/vignettes/`. Using simulated data we can easily vary problem size to study performance of the code - which is our main intention here - and make comparisons to other packages. Our tests have been performed on a dual-socket, 64-bit Intel machine with 8 cores per socket which are clocked at 2.6 GHz[2]. R has been natively compiled on this machine using `gcc` with BLAS/LAPACK support from single-threaded Intel MKL v11.0.

The 3 types of model coefficients mentioned in Section 2.2 entail very different computational requirements. In particular it can be seen from Equations 10 and 11, that Hessian and gradient calculation is computationally very demanding for alternative independent coefficients. For clear-cut comparisons we speed test the code with 4 types of problems described below. In our results we shall use K to denote the number of alternatives and n_p to denote the total number of coefficients in the model.

1. **Problem 'X':** A model with only individual specific data with alternative specific coefficients.
2. **Problem 'Y':** A model with data varying both with individuals and alternatives and alternative specific model coefficients.
3. **Problem 'Z':** Same type of data as problem 'Y' but with alternative independent coefficients which are independent of alternatives.
4. **Problem 'YZ':** Same type of data as problem 'Y' but with a mixture of alternative specific and alternative independent coefficients.

Problem 'X' may be considered a special case of problem 'Y'. However we have considered it separately because it is typically used in machine learning domains as the simplest linear multiclass classifier (Hastie *et al.* 2009). We shall also demonstrate that **mnlogit** runs much faster for this class of problems and use it to compare with the R packages **VGAM** and **nnet** which run only this class of problems (see Appendix C). The 'YZ' class of problems serves to illustrate a common use case of multinomial logit models in econometrics where the data may vary with both individuals and alternatives while the coefficients are a mixture of alternative specific and independent types (usually only a small fraction of variables are modeled with alternative independent coefficients).

The workings of **mnlogit** can be logically broken up into 3 steps:

1. Pre-processing: Where the model formula is parsed and matrices are assembled from a user supplied `data.frame`. We also check the data for collinear columns (and remove them) to satisfy certain necessary conditions, specified in Appendix B, for the Hessian to be non-singular.
2. Newton-Raphson Optimization: Where we maximize the loglikelihood function to estimate model coefficients. This involves solving a linear system of equations and one needs to compute the loglikelihood function's gradient vector and Hessian matrix.

[2]The machine has 128 GB of RAM and 20 MB of shared L3 cache.

Problem	Pre-processing time(s)	NR time(s)	Hessian time(s)	Total time(s)	n_p
X	93.64	1125.5	1074.1	1226.7	4950
Y	137.0	1361.5	1122.4	1511.8	5000
Z	169.9	92.59	60.05	272.83	50
YZ	170.1	1247.4	1053.1	1417.5	4505

Table 1: Performance profile of **mnlogit** for different problems with 50 variables and $K = 100$ alternatives with data for $N = 100,000$ individuals. All times are in seconds. 'NR time' is the total time taken in Newton-Raphson estimation while 'Hessian time' (which is included in 'NR time') is the time spent in computing Hessian matrices. Column n_p has the number of model coefficients. Problem 'YZ' has 45 variables modeled with individual specific coefficients while the other 5 variables are modeled with alternative independent coefficients.

3. Post-processing: All work needed to take the estimated coefficients and returning an object of class `mnlogit`.

Table 1 shows the profile of **mnlogit** performance for the four representative problems discussed earlier. Profiling the code clearly shows the highest proportion of time is spent in Hessian calculation (except for problem 'Z', for which the overall time is relatively lower). This is not an unexpected observation, it underpins our focus on optimizing Hessian calculation. Notice the very high pre-processing time for problem 'Z' whereof a large portion is spent in ensuring that the data satisfies necessary conditions, mentioned in Appendix B, for the Hessian to be non-singular.

4.1. Comparing mnlogit performance

We now compare the performance of **mnlogit** in single-threaded mode with some other R packages. This section focuses on the comparison with the R package **mlogit** since it is the only one which covers the entire range of variable and data types as **mnlogit**. Appendix C contains a synopsis of our data generation and timing methods including a comparison of **mnlogit** with the R packages **VGAM** and **nnet**.

Table 2 shows the ratio between **mlogit** and **mnlogit** running times for the four categories of problems considered in Table 1. We see that for most problems, except those of type 'Z', **mnlogit** outperforms **mlogit** by a large factor. We have not run larger problems for this comparison because **mlogit** running times become too long, except for problem 'Z'. In this case with $K = 100$ and keeping other parameters the same as Table 2, **mnlogit** outperforms **mlogit** by factors of 1.35 and 1.26 while running the NR and BFGS, respectively.

Besides Newton-Raphson, which is the default, we have also run **mlogit** with the BFGS optimizer. Typically the BFGS method, part of the quasi-Newton class of methods, takes more iterations than the Newton method but with significantly lower cost per iteration since it never directly computes the Hessian matrix. Typically for large problems the cost of computing the Hessian becomes too high and the BFGS method becomes overall faster than the Newton method (Nocedal and Wright 2000). Our approach in **mnlogit** attacks this weakness of the Newton method by exploiting the structure and sparsity in matrices involved in the Hessian calculation to enable it to outperform BFGS.

Optimizer	Newton-Raphson			BFGS		
K	**10**	**20**	**30**	**10**	**20**	**30**
Problem X	18.9	37.3	48.4	14.7	29.2	35.4
Problem Y	13.8	20.6	33.3	14.9	18.0	23.9
Problem YZ	10.5	22.8	29.4	10.5	17.0	20.4
Problem Z	1.16	1.31	1.41	1.01	0.98	1.06

Table 2: Ratio between **mlogit** and **mnlogit** total running times on a single processor for problems of various sizes and types. Each problem has 50 variables with K alternatives and $N = 50 \times K \times 20$ observations to estimate the model. **mlogit** has been run separately with two optimizers: Newton-Raphson and BFGS. In all cases the iterations terminated when the difference between log likelihoods in successive iterations reduced below 10^{-6}. Note: These numbers can vary depending on the BLAS implementation linked to R and hardware specifications.

4.2. Parallel performance

We now turn to benchmarking **mnlogit's** parallel performance. Figure 1 shows the speedups we obtain in Hessian calculation for the same problems considered in Table 1. The value of n_p, the number of model parameters, is significant because it is the dimension of the Hessian matrix (the time taken to compute the Hessian scales like $O(n_p^2)$). We run the parallel code separately on 2, 4, 8, 16 processor cores, comparing in each case with the single core time. Figure 1 shows that it is quite profitable to parallelize problems 'X' and 'Y', but the gains for problem 'Z' are not high. This is because of a design choice we make: Hessian calculation for type 'Z' variables is optimized for serial processing. In practical modeling, number of model coefficients associated with 'Z' types variable is not high, especially when compared to those of types 'X' and 'Y' (because the number of the coefficients of these types is also proportional to the number of choices in the model, see Section 2.4). For problems of type 'YZ' (or other combinations which involve 'Z'), parallelization can bring significant gains if the number of model coefficients of type 'Z' is low, relative to other types.

It can also be seen from Figure 1 that parallel performance degrades quickly as the number of processor cores is increased beyond 4. This is a consequence of the fact that our OpenMP program runs on a machine with shared cache and main memory, so increasing the number of threads degrades performance by increasing the chance of cache misses and hence slowing memory lookups. This is an intrinsic limitation of our hardware for which there is a theoretically simple solution: run the program on a machine with a larger cache!

An important factor to consider in parallel speedups of the whole program is Amdahl's Law which limits the maximum speedup that may be achieved by any parallel program (see Chapter 6 of Chandra, Dagum, Kohr, Maydan, McDonald, and Menon (2001)). Assuming parallelization between n threads, Amdahl's law states that the ultimate speedup is given by: $S_n = \frac{1}{f_s + (1-f_s)/n}$, where f_s is the fraction of non-parallelized, serial code. Table 3 lists the observed speedups on 2, 4 and 8 processor cores together with f_s for problems of Table 1. We take the time not spent in computing the Hessian as the 'serial time' to compute f_s and neglect the serial time spent in setting up the parallel computation in Hessian calculation, which mainly involves spawning threads in OpenMP and allocating separate blocks of working memory for each thread. For type 'Z' problems, this is an underestimate because some

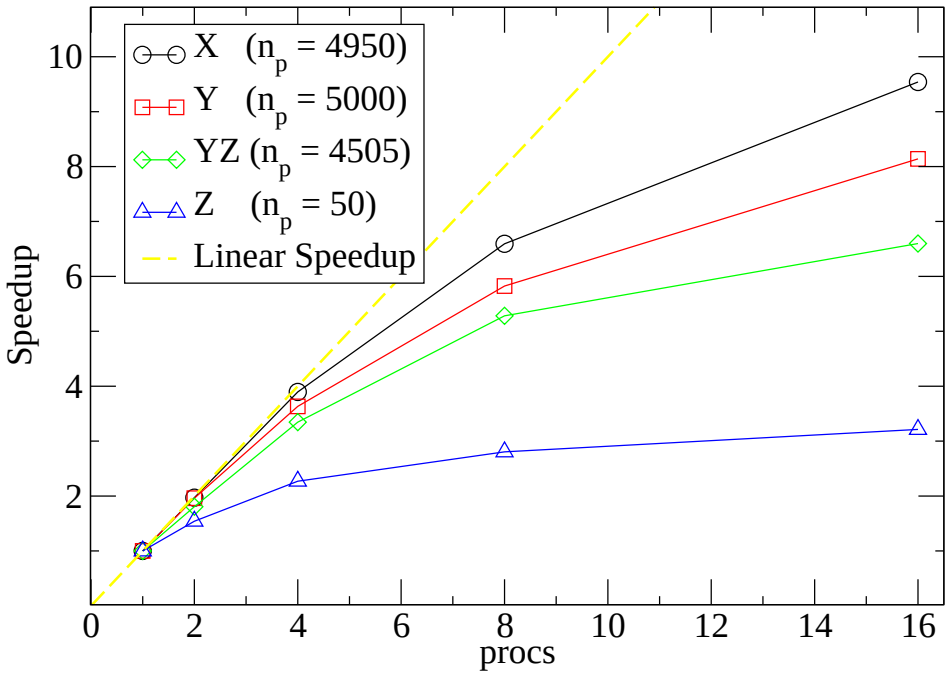

Figure 1: Parallel Hessian calculation speedup (ratio of parallel to single thread running time) for 2, 4, 8, 16, processor cores for problems of Table 1. The dashed 'Linear Speedup' guideline represents perfect parallelization.

Hessian calculation is also serial. Our tests have shown that compared to the Hessian calculation, the (serial) work required in setting up parallel computation is negligible, except for very small problems.

5. Discussion

Through **mnlogit** we seek to provide the community a package which combines quick calculation and the ability to handle data collinearity with a software interface which encompasses a wide range of multinomial logit models and data types used in econometrics and machine learning. Our main idea, exploiting matrix structure in large scale linear algebra calculations is not novel; however this work is the first, as far as we are aware, to apply it to the estimation of multinomial logit models problems in a working software package. The parallelization capability of **mnlogit**, which can easily add a 2x-4x factor of speedup on now ubiquitous mulitcore computers, is another angle which is underutilized in statistical software. Although **mnlogit** code is not parallelized to the maximum possible extent, parallelizing the most expensive parts of the calculation was an important design goal. We hope that practical users of the package benefit from this feature.

This work was initially motivated by the need to estimate large-scale multinomial logistic regression models. For very large-scale problems, Newton's method is usually outperformed by gradient based, quasi-Newton methods like the l-BFGS algorithm (Liu and Nocedal 1989). Hessian based methods based still hold promise for such problems. The class of inexact Newton (also called truncated Newton) methods are specifically designed for problems where

Problem	Serial fraction (f_s)	S_2	S_4	S_8
X	0.124	1.76(1.78)	2.87(2.92)	4.04(4.28)
Y	0.258	1.59(1.62)	2.26(2.27)	2.59(2.85)
Z	0.780	1.08(1.12)	1.14(1.20)	1.17(1.24)
YZ	0.257	1.44(1.59)	2.08(2.26)	2.36(2.86)

Table 3: Parallel speedup of **mnlogit** versus serial performance, (parentheses: predicted ultimate speedup from Amdahl's law) for problems of table 1. S_2, S_4 and S_{16} are observed speedups on 2, 4 and 16 processor cores respectively, while f_s is the estimated fraction of time spent in the serial portion of the code.

the Hessian is expensive to compute but taking a Hessian-vector product (for any given vector) is much cheaper (Nash 2000). Multinomial logit models have a Hessian with a structure which permits taking cheap, implicit products with vectors. Where applicable, inexact Newton methods have the promise of being better than l-BFGS methods (Nash and Nocedal 1991) besides having low memory requirements (since they never store the Hessian) and are thus very scalable. In the future we shall apply inexact Newton methods to estimating multinomial logit models to study their convergence properties and performance.

Acknowledgements

We would like to thank Florian Oswald of the Economics Department at University College London for contributing code for the `predict.mnlogit` method. We are also grateful to numerous users of **mnlogit** who gave us suggestions for improving the software and reported bugs.

A. Loglikelihood differentiation

In this Appendix we give the details of our computation of gradient and Hessian of the loglikelihood function in Equation 6. We make use of the notation of Section 3.3. Taking the derivative of the loglikelihood with respect to a chunk of coefficient $\vec{\theta}_m$ one gets:

$$\frac{\partial l}{\partial \vec{\theta}_m} = \sum_{i=1}^{N} \left[\frac{1}{\mathsf{P}_{i0}} \frac{\partial \mathsf{P}_{i0}}{\partial \vec{\theta}_m} + \sum_{k=1}^{K-1} I(y_i = k) \frac{\partial V_{ik}}{\partial \vec{\theta}_m} \right].$$

The second term in this equation is a constant term, since the utility V_{ik}, defined in Equation 3, is a linear function of the coefficients. Indeed we have:

$$\sum_{i=1}^{N} \sum_{k=1}^{K-1} I(y_i = k) \frac{\partial V_{ik}}{\partial \vec{\theta}_m} = \begin{cases} \mathbf{M_m}^\top \vec{y}_m & \text{if } \vec{\theta}_m \in \left\{ \vec{\beta}_1, \ldots \vec{\beta}_{K-1}, \vec{\gamma}_0, \ldots \vec{\gamma}_{K-1} \right\} \\ \sum_{k=1} \mathbf{Z_k}^\top \vec{y_k} & \text{if } \vec{\theta}_m \text{ is } \vec{\alpha} \end{cases} \tag{12}$$

The vectors $\vec{y}_m$ and the matrices $\mathbf{M_m}$ and $\mathbf{Z_k}$ are specified in Section 3.3. We take the derivative of the base case probability, which is specified in Equation 5, as follows:

$$\sum_{i=1}^{N} \frac{1}{\mathsf{P}_{i0}} \frac{\partial \mathsf{P}_{i0}}{\partial \vec{\theta}_m} = \begin{cases} -\mathbf{M_m}^\top \cdot \vec{\mathsf{P}}_m & \text{if } \vec{\theta}_m \in \left\{ \vec{\beta}_1, \ldots \vec{\beta}_{K-1}, \vec{\gamma}_0, \ldots \vec{\gamma}_{K-1} \right\} \\ -\sum_{k=1} \mathbf{Z_k}^\top \vec{\mathsf{P}}_k & \text{if } \vec{\theta}_m \text{ is } \vec{\alpha} \end{cases} \tag{13}$$

Here the probability vector $\vec{\mathsf{P}}_m$ is of length N with entries P_{im}. In the last line we have used the fact that, after normalization, $\mathbf{Z}_0$ is $\mathbf{0}$. Using Equations 12 and 13 we get the gradient in the form shown in Equation 10.

Upon differentiating the probability vector $\vec{\mathsf{P}}_k$ $(k \geq 1)$ in Equation 4 with respect to $\vec{\theta}_m$ we get:

$$\frac{\partial \vec{\mathsf{P}}_k}{\partial \vec{\theta}_m} = \begin{cases} \mathbf{W_{km}}\mathbf{M_m} & \text{if } \vec{\theta}_m \in \left\{\vec{\beta}_1, \ldots \vec{\beta}_{K-1}, \vec{\gamma}_0, \ldots \vec{\gamma}_{K-1}\right\} \\ \mathbf{D}(\vec{\mathsf{P}}_k)\left(\mathbf{Z_k} - \sum_{t=1} \mathbf{Z_t}\mathbf{D}(\vec{\mathsf{P}}_t)\right) & \text{if } \vec{\theta}_m \text{ is } \vec{\alpha} \end{cases} \tag{14}$$

where $\mathbf{D}(\vec{\mathsf{P}}_k)$ is an $N \times N$ diagonal matrix whose i^{th} diagonal entry is P_{ik} and, matrix $\mathbf{W_{km}}$ is also an an $N \times N$ diagonal matrix whose i^{th} diagonal entry is $\mathsf{P}_{ik}(\delta_{km} - \mathsf{P}_{im})$. In matrix form this is: $\mathbf{W_{km}} = \delta_{km}\mathbf{D}(\vec{\mathsf{P}}_k) - \mathbf{D}(\vec{\mathsf{P}}_k)\mathbf{D}(\vec{\mathsf{P}}_m)$ where δ_{km} is the Kronecker delta.

We write the Hessian of the loglikelihood in block form as:

$$\mathbf{H_{nm}} = \frac{\partial^2 l}{\partial \vec{\theta}_n \partial \vec{\theta}'_m} = \sum_{i=1}^{N} \left[\frac{1}{\mathsf{P}_{i0}} \frac{\partial^2 \mathsf{P}_{i0}}{\partial \vec{\theta}_n \partial \vec{\theta}'_m} - \frac{1}{\mathsf{P}_{i0}^2} \frac{\partial \mathsf{P}_{i0}}{\partial \vec{\theta}_n} \frac{\partial \mathsf{P}_{i0}}{\partial \vec{\theta}_m} \right].$$

However it can be derived in a simpler way by differentiating the gradient with respect to $\vec{\theta}_n$. Doing this and making use of Equation 14 gives us Equation 11. The first two cases of equation are fairly straightforward with the matrices $\mathbf{W}_{km}$ being the same as shown in Equation 14. The third case, when $(\vec{\theta}_n, \vec{\theta}_m$ are both $\vec{\alpha})$, is a bit messy and we describe it here.

$$\begin{aligned} \mathbf{H_{nm}} &= -\sum_{k=1}^{K-1} \left[\mathbf{Z_k}^\top \mathbf{D}(\vec{\mathsf{P}}_k) \left(\mathbf{Z_k} - \sum_{t=1}^{K-1} \mathbf{D}(\vec{\mathsf{P}}_t)\mathbf{Z_t} \right) \right] \\ &= -\sum_{k=1}^{K-1} \sum_{t=1}^{K-1} \mathbf{Z_k}^\top \left[\delta_{kt}\mathbf{D}(\vec{\mathsf{P}}_k) - \mathbf{D}(\vec{\mathsf{P}}_k)\mathbf{D}(\vec{\mathsf{P}}_t) \right] \mathbf{Z_t} \\ &= -\sum_{k=1} \sum_{t=1} \mathbf{Z_k}^\top \mathbf{W_{kt}} \mathbf{Z_t}. \end{aligned}$$

Here the last line follows from the definition of matrix $\mathbf{W_{kt}}$ as used in Equation 14.

B. Data requirements for Hessian non-singularity

We derive necessary conditions on the data for the Hessian to be non-singular. Using notation from Section 3.2, we start by building a 'design matrix' $\tilde{\mathbf{X}}$ by concatenating data matrices $\mathbf{X}$, $\mathbf{Y_k}$ and $\mathbf{Z_k}$ in the following format:

$$\tilde{\mathbf{X}} = \begin{pmatrix} \mathbf{X} & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & \mathbf{Z_1}/2 \\ 0 & \mathbf{X} & \cdots & 0 & 0 & 0 & \cdots & 0 & \mathbf{Z_2}/2 \\ \vdots & & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & \mathbf{X} & 0 & 0 & \cdots & 0 & \mathbf{Z_{K-1}}/2 \\ 0 & \cdots & \cdots & 0 & \mathbf{Y_0} & 0 & \cdots & 0 & 0 \\ 0 & \cdots & \cdots & 0 & 0 & \mathbf{Y_1} & \cdots & 0 & \mathbf{Z_1}/2 \\ 0 & \cdots & \cdots & 0 & 0 & 0 & \ddots & 0 & \vdots \\ 0 & \cdots & \cdots & 0 & 0 & 0 & \cdots & \mathbf{Y_{K-1}} & \mathbf{Z_{K-1}}/2 \end{pmatrix}. \tag{15}$$

In the above 0 stands for a matrix of zeros of appropriate dimension. Similarly we build two more matrices $\mathbf{Q}$ and $\mathbf{Q_0}$ as shown below:

$$\mathbf{Q} = \begin{pmatrix} \mathbf{W_{11}} & \mathbf{W_{12}} & \cdots & \mathbf{W_{1,K-1}} \\ \mathbf{W_{21}} & \mathbf{W_{22}} & \cdots & \mathbf{W_{2,K-1}} \\ \vdots & \vdots & \cdots & \vdots \\ \mathbf{W_{K-1,1}} & \cdots & \cdots & \mathbf{W_{K-1,K-1}} \end{pmatrix},$$

$$\mathbf{Q_0} = \begin{pmatrix} \mathbf{W_{10}} \\ \mathbf{W_{20}} \\ \vdots \\ \mathbf{W_{K-1,0}} \end{pmatrix}.$$

Using the two matrices above we define a 'weight' matrix $\tilde{\mathbf{W}}$:

$$\tilde{\mathbf{W}} = \begin{pmatrix} \mathbf{Q} & \mathbf{Q_0} & \mathbf{Q} \\ \mathbf{Q_0}^\top & \mathbf{W_{00}} & \mathbf{Q_0}^\top \\ \mathbf{Q} & \mathbf{Q_0} & \mathbf{Q} \end{pmatrix}, \tag{16}$$

The full Hessian matrix, containing all the blocks of Equation 11, is given by: $\mathbf{H} = \tilde{\mathbf{X}}^\top \tilde{\mathbf{W}} \tilde{\mathbf{X}}$. For the matrix $\mathbf{H}$ to be non-singular, we must have the matrix $\tilde{\mathbf{X}}$ be full-rank. This leads us to the following necessary conditions on the data for the Hessian to be non-singular:

1. All matrices in the set: $\{\mathbf{X}, \mathbf{Y_0}, \mathbf{Y_1} \ \ldots \ \mathbf{Y_{K-1}}\}$ must be of full rank.
2. At least one matrix from the set: $\{\mathbf{Z_1}, \mathbf{Z_2} \ \ldots \ \mathbf{Z_{K-1}}\}$ must be of full rank.

In **mnlogit** we directly test condition 1, while the second condition is tested by checking for collinearity among the columns of the matrix[3]:

$$\begin{pmatrix} \mathbf{Z_1} & \mathbf{Z_2} & \ldots & \mathbf{Z_{K-1}} \end{pmatrix}^\top.$$

Columns are arbitrarily dropped one-by-one from a collinear set until the remainder becomes linearly independent.

Another necessary condition: It can be shown with some linear algebra manipulations that if we have a model which has only data for alternative independent variables and includes the intercept, then the resulting Hessian will always be singular. **mnlogit** does not attempt to check the data for this condition which is independent of the 2 necessary conditions given above.

C. Timing tests

We give the details of our simulated data generation process and how we setup runs of the R packages **mlogit**, **VGAM** and **nnet** to compare running times against **mnlogit**. We start by loading **mlogit** into an R session:

[3]Since number of rows is less than the number of columns

```
R> library("mlogit")
```

Next we generate data in the 'long format' (described in Section 2) using the `makeModel` function sourced from the file `simChoiceModel.R` which is in the package folder `mnlogit/vignettes/`. The data we use for the timing tests shown here is individual specific (problem 'X' of Section 4) because this is the only one that packages **VGAM** and **nnet** can run. We generate data for a model with K choices as shown below.

```
R> source("simChoiceModel.R")
R> data <- makeModel('X', K=10)
```

Default arguments of `makeModel` set the number of variables and the number of observations, which are:

```
Number of choices in simulated data = K = 10.
Number of observations in simulated data = N = 10000.
Number of variables = p = 50.
Number of model parameters = (K - 1) * p = 450.
```

The next steps setup a `formula` object which specifies that individual specific data must be modeled with alternative specific coefficients and the intercept is excluded from the model.

```
R> vars <- paste("X", 1:50, sep="", collapse=" + ")
R> fm <- formula(paste("response ~ 1|", vars, " - 1 | 1"))
```

Using this formula and our previously generated `data.frame` we run **mnlogit** to measure its running time (in single threaded mode).

```
R> system.time(fit.mnlogit <- mnlogit(fm, data, "choices"))

   user  system elapsed
  2.012   0.064   2.076
```

Likewise we measure running times for **mlogit** running the same problem with the Newton-Raphson (the default) and the BFGS optimizers.

```
R> mdat <- mlogit.data(data[order(data$indivID), ], "response", shape="long",
+  alt.var="choices")
R> system.time(fit.mlogit <- mlogit(fm, mdat))   # Newton-Raphson

   user  system elapsed
 33.708   3.175  36.901

R> system.time(fit.mlogit <- mlogit(fm, mdat, method='bfgs'))

   user  system elapsed
 29.790   6.263  36.079
```

Here the first step is necessary to turn the `data.frame` object into an `mlogit.data` object required by **mlogit**. The default stopping conditions for **mnlogit** and **mlogit** are exactly the same. The timing results shown in Table 2 were obtained in a similar way but with different formulas for each type of problem. All our tests use the function `makeModel` to generate data.

For comparison with **nnet** we must make a few modifications: first we turn the data into a format required by **nnet** and then change the stopping conditions from their default to (roughly) match **mnlogit** and **mlogit**. We set the stopping tolerance so that 'reltol' controls convergence and roughly corresponds at termination to 'ftol' in these packages. Note that **nnet** runs the BFGS optimizer.

```
R> library("nnet")
R> ndat <- data[which(data$response > 0), ]
R> fm.nnet <- paste("choices ~", vars, "- 1")
R> system.time(fit.nnet <- multinom(fm.nnet, ndat, reltol=1e-12))

# weights:  510 (450 variable)
initial  value 23025.850930
iter  10 value 22825.619872
iter  20 value 22797.471897
iter  30 value 22795.961681
iter  40 value 22795.954086
iter  50 value 22795.953378
iter  60 value 22795.953341
final  value 22795.953340
converged
   user  system elapsed
  3.727   0.001   3.726
```

We remind the user that since **nnet** and **VGAM** only handle individual specific data, we can not test them on all the categories of problems listed in Table 2. To apply the same timing test to the `vglm` function from package **VGAM**, we first set the stopping condition to match the default condition for **mnlogit** and **mlogit** (`ftol = 1e-6`).

```
R> library("VGAM")
R> system.time(fit.vglm <- vglm(fm.nnet, data=ndat, multinomial(refLevel=1),
+  control=vglm.control(epsilon=1e-6)))

   user  system elapsed
 44.384   1.052  45.497
```

The precise times running times reported on compiling this Sweave document depend strongly on the machine, whether other programs are running simultaneously and the BLAS implementation linked to R. For reproducible results run on a 'quiet' machine (with no other programs running).

References

Begg CB, Gray R (1984). "Calculation of Polychotomous Logistic Regression Parameters Using Individualized Regressions." *Biometrika*, **71**, 11–18.

Bhat C (1995). "A heterocedastic extreme value model of intercity travel mode choice." *Transportation Research B*, **29**(6), 471–483.

Bohning D (1992). "Multinomial Logistic Regression Algorithm." *Annals of the Inst. of Statistical Math.*, **44**, 197–200.

Chandra R, Dagum L, Kohr D, Maydan D, McDonald J, Menon R (2001). *Parallel Programming in OpenMP*. Academic Press, New York.

Colby S, Lee S, Lewinger JP, Bull S (2010). ***pmlr**: Penalized Multinomial Logistic Regression*. R package version 1.0, URL http://CRAN.R-project.org/package=pmlr.

Croissant Y (2012). *Estimation of Multinomial Logit Model in R: The Package **mlogit***. R package version 0.2-3, URL http://CRAN.R-project.org/package=mlogit.

Friedman J, Hastie T, Tibshirani R (2010). "Regularization Paths for Generalized Linear Models via Coordinate Descent." *Journal of Statistical Software*, **33**(1), 1–22. URL http://www.jstatsoft.org/v33/i01/.

Golub GH, Loan CFV (2013). *Matrix Computations*. 4th. edition. The Johns Hopkins University Press.

Hastie T, Tibshirani R, Friedman J (2009). *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. 2nd. edition. Springer-Verlag.

Jurka TP (2012). "**maxent**: An R Package for Low-memory Multinomial Logistic Regression with Support for Semi-automated Text Classification." *The R Journal*, **4**, 56–59.

Komarek P, Moore AW (2005). "Making Logistic Regression a Core Data Mining Tool: A Practical Investigation of Accuracy, Speed, and Simplicity." *Technical report*, Carnegie Mellon University.

Li J (2013). "Logistic regression." Course Notes. URL http://sites.stat.psu.edu/~jiali/course/stat597e/notes2/logit.pdf.

Lin CJ, Weng RC, Keerthi SS (2008). "Trust Region Newton Method for Large-Scale Logistic Regression." *Journal of Machine Learning Research*, **9**, 627–650.

Liu DC, Nocedal J (1989). "On the Limited Memory BFGS Method for Large Scale Optimization." *Mathematical Programming*, **45**, 503–528.

McFadden D (1974). "The Measurement of Urban Travel Demand." *Journal of public economics*, **3**, 303–328.

Nash SG (2000). "A Survey of Truncated-Newton Methods." *Journal of Computational and Applied Mathematics*, **124**, 45–59.

Nash SG, Nocedal J (1991). "A Numerical Study of the Limited Memory BFSG Method and the Truncated-Newton Method for Large-Scale Optimization." *SIAM Journal of Optimization*, **1**, 358–372.

Nigam K, Lafferty J, McCallum A (1999). "Using Maximum Entropy for Text Classification." In *IJCAI-99 Workshop on Machine Learning for Information Filtering*.

Nocedal J (1990). "The Performance of Several Algorithms for Large Scale Unconstrained Optimization." In *Large Scale Numerical Optimization*.

Nocedal J (1992). "Theory of Algorithms for Unconstrained Optimization." *Acta Numerica*.

Nocedal J, Wright S (2000). *Numerical Optmization*. 2nd. edition. Springer-Verlag.

R Core Team (2013). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org.

Sevcikova H, Raftery A (2013). ***mlogitBMA**: Bayesian Model Averaging for Multinomial Logit Model*. R package version 0.1-6, URL http://CRAN.R-project.org/package=mlogitBMA.

Train KE (2003). *Discrete choice methods with simulation*. Cambridge University Press, Cambridge, UK.

Trefethen LN, Bau D (1997). *Numerical Linear Algebra*. Siam, Philadelphia.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S*. 4th. edition. Springer-Verlag, New York. ISBN 0-387-95457-0, URL http://www.stats.ox.ac.uk/pub/MASS4.

Yee TJ (2010). "The VGAM Package for Categorical Data Analysis." *Journal of Statistical Software*, **32**(10). URL http://www.jstatsoft.ord/v32/i10/.

Zeileis A, Croissant Y (2010). "Extended Model Formulas in R: Multiple Parts and Multiple Responses." *Journal of Statistical Software*, **34**(1), 1–13. URL http://www.jstatsoft.org/v34/i01/.

Zeileis A, Hothorn T (2002). "Diagnostic Checking in Regression Relationships." *R News*, **2**(3), 7–10. URL http://CRAN.R-project.org/doc/Rnews/.

Affiliation:

Asad Hasan
Scientific Computing Group
Sentrana Inc.
1725 I St NW
Washington, DC 20006
E-mail: asad.hasan@sentrana.com

Zhiyu Wang
Department of Mathematical Sciences
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213

Alireza S. Mahani
Scientific Computing Group
Sentrana Inc.
1725 I St NW
Washington, DC 20006
E-mail: alireza.mahani@sentrana.com

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume VV, Issue II *Submitted:* yyyy-mm-dd
MMMMMM YYYY *Accepted:* yyyy-mm-dd