Published in final edited form as:
R J. 2013 June ; 5(1): 181–187.

Fast Pure R Implementation of GEE: Application of the Matrix Package

Lee S. McDaniel,
University of Wisconsin-Madison 1300 University Ave. Madison, WI 53706 USA
mcdaniel@stat.wisc.edu

Nicholas C. Henderson, and
University of Wisconsin-Madison 1300 University Ave. Madison, WI 53706 USA
nhenders@stat.wisc.edu

Paul J. Rathouz
University of Wisconsin-Madison 1300 University Ave. Madison, WI 53706 USA
rathouz@biostat.wisc.edu

Abstract

Generalized estimating equation solvers in R only allow for a few pre-determined options for the link and variance functions. We provide a package, **geeM**, which is implemented entirely in R and allows for user specified link and variance functions. The sparse matrix representations provided in the **Matrix** package enable a fast implementation. To gain speed, we make use of analytic inverses of the working correlation when possible and a trick to find quick numeric inverses when an analytic inverse is not available. Through three examples, we demonstrate the speed of **geeM**, which is not much worse than C implementations like **geepack** and **gee** on small data sets and faster on large data sets.

Introduction

An extension of generalized linear models and quasilikelihood (McCullagh and Nelder, 1986), generalized estimating equations (GEEs; Liang and Zeger, 1986) are a useful method for analyzing clustered (or longitudinal) data which may be correlated within cluster but are independent between clusters.

There are two packages on CRAN which will solve GEEs and produce standard errors in R: **geepack** (Højsgaard et al., 2006) and **gee** (Carey. et al., 2012). **gee** provides the basic functionality necessary for fitting GEEs, while **geepack** also allows for regression models for both the scale and correlation parameters, as described in Yan and Fine (2004). Both of these packages rely heavily on a computational engine implemented in C and called by the R function wrapper. They use the family object as input and hard code each of the link and variance function options within the C code. Whereas this configuration yields computational speed, it has two important drawbacks: it does not allow the user to create and specify his own link and variance functions, and it is not easy for a programmer with only modest programming skills to alter the code for methodological developments related to or extending GEE.

To see why the user may need to define link and variance functions, consider the methodology presented in Schildcrout and Rathouz (2010). The authors look at binary response data in an outcome dependent sampling situation. Because the response is binary, the authors are able to use the offset option in standard GEE software to correct the bias from the sampling scheme. However, if the response takes some other form (e.g. counts or a continuous response), then we would need to define link and variance functions to correct this bias. These link and variance functions involve integrals that need to be computed whenever the functions are evaluated at different points.

Our new package, called **geeM** (the M is to emphasize the use of the **Matrix** package; Bates and Maechler, 2013) is coded completely in R, which grants the user the flexibility to define link and variance functions as part of the model specification (McDaniel and Henderson, 2013). It also allows for easy modification of the code by method developers. The main function in the package, geem, takes a list that includes the link function, variance function, inverse link function, and the derivative of the inverse link function, as an argument. These functions must all be vectorized.

In spite of this, we are still able to solve the estimating equations quickly for many problems. To accomplish this, the package relies heavily on the recently developed **Matrix** package for speed. We especially exploit the sparse matrix representation objects contained therein, which allow for efficient operations on matrices in which most elements are zero. Besides introducing our package, a key aim of this article is to demonstrate in the context of a well-known statistical algorithm the power, speed and flexibility of the **Matrix** package.

We first give the details of the implementation, and then describe some limitations of the package. Finally, we show through examples and simulations the relative speed of the new R package. Along the way, we point out innovative uses of **Matrix** package functions.

GEE solutions

To set up the problem, consider Y_i, a multivariate n_i-vector of responses $Y_i = (Y_{i1}, \dots, Y_{it}, \dots, Y_{in_i})$ with mean vector $\mu_i = (\mu_{i1}, \dots, \mu_{it}, \dots, \mu_{in_i})$. Denote $E(Y_{it}) = \mu_{it}$. Let $X_i = (x_{i1}, \dots, x_{it}, \dots, x_{in_i})'$ be a $n_i \times p$ design matrix of covariate vectors corresponding to each observation for the ith cluster.

In the typical specification, the expectations are related to the mean by the monotone link function,

$$g(\mu_{it}) = \eta_{it} = x_{it}^{\mathsf{T}} \beta,$$

and the variances are a function of the mean

$$var(\mathrm{Y}_{it}) = \phi a_{it} = \phi a(\mu_{it}),$$

where φ is a dispersion parameter. Let A_i be a diagonal matrix with kth diagonal element equal to $a(\mu_{ik})$. Next we define

$$V_i = \phi A_i^{\frac{1}{2}} R_i(\alpha) A_i^{\frac{1}{2}}, \quad (1)$$

where $R_i(\alpha)$ is the working correlation matrix for the ith cluster, parameterized by α, which may be a vector.

Using these conventions, Liang and Zeger (1986) describe a modified Fisher scoring algorithm to estimate regression coefficients β. The generalized estimating equations are given by

$$\sum_{i=1}^{K} D_i^\top V_i^{-1} S_i = 0, \quad (2)$$

where $D_i = A_i \Delta_i X_i$ and Δ_i is a diagonal matrix with kth entry equal to $\frac{dg^{-1}(\eta_{ik})}{d\eta_{ik}}$. Finally, $S_i = Y_i - \mu_i$.

Instead of using a summation, we can instead put all of this in matrix form. Let $\sum_{i=1}^{K} n_1$, X be a $N \times p$ matrix with the X_i matrices stacked on top of one another, Δ and A be $N \times N$ diagonal matrices with the appropriate entries from Δ_i and A_i on the diagonal, and S be a N-vector μ generated by stacking up the S_i's. Finally, $R(\alpha)$ is a $N \times N$ block diagonal matrix with the individual $R_i(\alpha)$ matrices as blocks. With these definitions, our GEE representation is

$$X^\top \Delta A \left(A^{\frac{1}{2}} R(\alpha) A^{\frac{1}{2}}\right)^{-1} S = 0. \quad (3)$$

Implementation details

The **geeM** package is coded entirely in R. This allows for easier modification by methodologists considering variants and extensions of GEE, as well as increased flexibility for some specific analyses.

Uses of the Matrix package

Creating large matrices to solve the estimating equations leads to large sparse matrices. The **Matrix** package allows us to easily exploit this sparsity with its sparse matrix representations, which store only the non-zero elements of a matrix. Without the memory savings from these representations, even moderately sized problems would lead to huge memory demands.

Further, we benefit from an impressive gain in the speed of operating on these large sparse matrices. Multiplication is the largest burden computationally, and the speed gains in very large, very sparse matrices are immense. This is accomplished by a separately defined %*% method for objects in the “sparseMatrix” class. When multiplication is of the form $B^\mathrm{T}C$ or $B^\mathrm{T}B$ we can take advantage of the crossprod function.

In the modified Fisher scoring algorithm, we have many opportunities to take advantage of the “sparseMatrix” class. Two diagonal matrices are used, *A* and Δ, which we create using the Diagonal function. We also have the working correlation matrix, $R(\alpha)$, which is a sparse symmetric block diagonal matrix because observations within clusters may be correlated but observations between clusters are independent.

Additionally, the Fisher scoring algorithm requires that we compute the inverse of the working correlation matrix, $R(\alpha)$. Because we never need $R(\alpha)$ directly, we only compute $R(\alpha)^{-1}$ and build it directly for any given α. For that we use the sparseMatrix function, to which we supply three vectors of the same length: an integer vector i of row positions of non-zero entries, an integer vector j of column positions of non-zero entries, and a vector × of entries.

Analytic inverses

For three common correlation structures, independence, exchangeable, and AR-1, analytic inverses are available. In these cases we use these analytic inverses to create $R(\alpha)^{-1}$.

In the AR-1 case, we are able to quickly construct the inverse correlation matrix as a linear combination of three basic matrices, with coefficients depending on the value of α (Sutradhar and Kumar, 2003). The inverse can be constructed as

$$R(\alpha)^{-1}=\frac{1}{1-\alpha^2}\left\{L-\alpha M+\left(1+\alpha^2\right)N\right\},$$

where *L*, *M*, and *N* are all block diagonals, the blocks of which take the forms

$$L_i=\begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix},$$

$$M_i=\begin{pmatrix} 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 1 & 0 & 1 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & & & \ddots & & & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 1 & 0 \\ 0 & 0 & 0 & \cdots & 1 & 0 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 1 & 0 \end{pmatrix},$$

and

$$N_i = \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 0 \end{pmatrix}.$$

These basic matrices are constructed only once at the beginning of the code and saved for later use.

The exchangeable correlation matrix is a little more complicated because the entries in the inverse depend on the cluster size (Sutradhar and Kumar, 2003). The analytic inverse can be computed as

$$R_1(\alpha)^{-1} = (a-b)\, I_{n_i} + bJ_{n_i},$$

where

$$\begin{aligned} a &= \tfrac{1+(n_i-2)\alpha}{(1-\alpha)\{1+(n_i-1)\alpha\}}, \\ b &= -\tfrac{\alpha}{(1-\alpha)\{1+(n_i-1)\alpha\}}, \end{aligned}$$

I_{ni} is the $n_i \times n_i$ identity matrix, and J_{ni} is an $n_i \times n_i$ matrix of ones. In the implementation, we compute a vector with every entry in the correlation matrix (a vector of length $\sum_{i=1}^{K} n_i^2$) and create the new matrix with the sparseMatrix function.

Numeric inverses

For all other correlation structures, we use the solve command in R. To describe the method, let us suppose that we have k different cluster sizes, $n_1, \ldots, n_k$, in descending order.

We first construct the working correlation matrix for all clusters of size n_1, then take the upper left corner of this matrix to construct correlation matrices of appropriate size for clusters of size $n_2, \ldots, n_k$. We then calculate the inverses of each of these k matrices. Finally, we construct the block diagonal working correlation matrix for the whole sample using the inverses of the appropriate size.

This technique exploits the fact that the inverse of a block diagonal matrix is the inverse of the blocks, allowing us to reduce the number of invocations of the solve function to k, the number of unique cluster sizes.

Using this technique, we have support for M-dependence (M should be specified by the user), unstructured correlation, fixed correlation, and user-defined correlation structures. Under M-dependence, two observations, x_{it} and x_{it}' have correlation $\alpha_{|t-t'|}$if $|t-t'| \leq M$ and 0 otherwise. Under unstructured correlation, the correlation, the correlation between x_{it} and x_{it}', is asummed to be α_{tt}'. The fixed correlation option requires that the user provides a correlation matrix giving the correlations between all observations in the largest cluster. For a user-defined correlation structure, the function accepts a square matrix of the size of the largest cluster which defines the entries in the correlation matrix assumed to be the same.

Note that in all correlation structures, if any variables are missing, then those rows are dropped. So, if rows have missing data, or a given subject has fewer than the maximum number of observations, then the leading principal submatrices of the appropriate dimensions are used. This may not preserve the desired correlation structure. The desired structure can be recovered by including rows with weights of zero for those time points with missing data.

Estimation of correlation parameters

For all supported correlation structures, we use simple moment estimators. These are the exact same estimators used by SAS proc genmod (SAS Institute Inc., 2010) and are estimated solely from the Pearson residuals.

Missing values

The weights argument can be used to handle missing values in the response by creating a new row for the missing observation and assigning a weight of 0. This will maintain the desired correlation structure and is equivalent to how SAS proc genmod handles missing data. Observations with a weight of 0 do not contribute to estimation of the coefficients or the variance and only observations with a positive weight are used to calculate the correlation parameters. Any observation with an NA in the response or predictors is automatically assigned a weight of 0.

Limitations

The **geeM** package calculates the inverse correlation matrix directly, possibly resulting in a less stable algorithm than **gee** and **geepack** which solve systems of equations instead of inverting directly. Therefore it is possible that the **geeM** package may have problems with numerical stability on some problems, especially for correlation structures for which inverses are calculated numerically. We have not seen any such issues in testing at this point.

Another issue has to do with how the calculations are completed. In testing the largest dataset (presented later as the birth outcomes data) we found that **geeM** was more likely to have memory problems than **gee** or **geepack**. This is probably because **geeM** creates large sparse matrices for computing its estimates as opposed to the technique in Liang and Zeger (1986), which involves summing over many more small dense matrices.

Speed comparisons

All speed comparisons were run on a computer with an Intel Core 2 Duo 3.0 GHz processor and 4 GB of RAM. Windows 7 was the operating system and R version 2.15.1 was used.

For these comparisons, we fit the same model to the same data set multiple times in order to gain some stability in the estimates of run time. The packages **geepack** and **gee** test convergence differently than **geeM**, so the tolerance used for **geeM** was chosen to be no less strict than the other two. Because **geeM** assesses convergence relative to the magnitude of parameters, this requires some knowledge of the values of the fitted parameters.

Ohio data set

geepack contains a data set on the wheezing status of children in Ohio. The data are 537 clusters each with 4 observations. The response (resp) is a binary variable with 1 corresponding to wheezing and 0 not. We fit the model resp ~ age + smoke + age:smoke, where age is the age of the child and smoke is an indicator of maternal smoking at the first year of the study. In this case we use a logit link and the $\mu(1 - \mu)$ variance function.

Table 1 shows the speed results under four different correlation structures. The average speed (Avg.) is calculated based on fitting the same data 5 times for each method and the relative speed (Rel.) is the average speed of each method divided by the average speed of the fastest method.

This is a relatively small data set, so all solvers perform very quickly in this case. The longest average time for any of them is 0.20 seconds, so speed does not appear to be too much of an issue here.

Simulated count data

For a slightly larger data set, we simulated count data from an overdispersed Poisson distribution. There are 10,000 clusters with 5 observations in each cluster. The formula we use is count ~ time. Here we use the log-linear link and the identity variance function.

Table 2 shows the speed results under four different working correlation structures. The average speed (Avg.) is calculated based on fitting the same data 5 times for each method and the relative speed (Rel.) is the average speed of each method divided by the average speed of the fastest method.

Birth outcomes data

Finally, we move to a large data set. These data contains the information on birth outcomes from Abrevaya (2006). We use gestational age as the continuous response, and mother's age as the only covariate. We use the Gaussian family for the link and variance functions.

The data contain 141,929 clusters of size 2 or 3 each. This results in a total of 296,218 observations. It is worth noting that for a data set this size, all other programs except R needed to be shut down. If too many other programs are open, **geeM** may take a much longer time or even run out of memory.

Table 3 shows the speed results under four different working correlation structures. The average speed (Avg.) is calculated based on fitting the data 5 times for each method and the relative speed (Rel.) is the average speed of each method divided by the average speed of the fastest method.

Illustrative use

Before concluding, we provide examples of how to use the geem function. Here we fit the ohio data with different link and variance functions. In this case, we use a skewed logit link function, as described in Prentice (1976), which allows us to remove the symmetry constraint imposed by the logit and probit link functions. According to Prentice, this type of skewed logit link may provide a better fit to dose response data. The inverse link function takes the form

$$p=\left(\frac{e^{x^{\top}\beta}}{1+e^{x^{\top}\beta}}\right)^{k},$$

where k is known. When $k = 1$ we have the logistic model, when $k < 1$ the distribution is negatively skewed, and when $k > 1$ the distribution is positively skewed. In the example, we let $k = 2$ and choose $\mu(1 - \mu)$ as the variance function. When defining these four functions, it is essential that they all be vectorized, that is, they accept a vector argument and return a vector of the same length.

```
k <- 2 linkfun <- function(p) { log((p^(1/k))/(1 - p^(1/k))) }

variance <- function(p) { p * (1-p) }

linkinv <- function(eta) { (exp(eta)/(1 + exp(eta)))^k

}

mu.eta <- function(eta) { k * (exp(eta))^(k-1) / (1 + exp(eta))^(k+1) }

FunList <- list(linkfun, variance, linkinv, mu.eta) geem(resp ~ age + smoke + age:smoke,
id=id, data = ohio, family = FunList, corstr = “unstructured”)
```

If we just want to use the binomial family to fit the ohio data, then the call is:

```
geem(resp ~ age + smoke + age:smoke, id=id, data = ohio, family = binomial, corstr =
“unstructured”)
```

Conclusion

This paper describes a pure R implementation of a generalized estimating equation solver relying heavily on the **Matrix** package. Coding in R allows for the use of user-defined link and variance functions, giving a useful kind of flexibility.

The **Matrix** package is what enables a fast implementation. Sparse matrix representations give ways to efficiently build the large, sparse matrices needed to solve the GEEs without using too much memory. Operations on these sparse matrices are also much faster.

Much of the work in fitting GEEs comes from matrix inversion or solving a system of equations. For three correlation structures, we are able to analytically invert the largest matrix. For the rest of the correlation structures, we rely on inversion of a relatively small number of blocks to find the inverse.

Unfortunately, this technique of gaining speed in matrix inversion can cause problems. Because we invert matrices instead of solving systems, it is possible that the function may be less numerically stable.

Finally, we show that, in some cases, the **geeM** package is faster than **gee** or **geepack**. In other cases it is not much slower, with **geeM** tending to run faster on larger data sets.

Acknowledgements

McDaniel's and Henderson's efforts were funded by NIGMS grant 5T32GM074904. Rathouz's effort was funded by NIH grant R01 HL094786.

Bibliography

Abrevaya J. Estimating the effect of smoking on birth outcomes using a matched panel data approach. Journal of Applied Econometrics. 2006; 21(4):489–519. [p185].

Bates, D.; Maechler, M. Matrix: Sparse and Dense Matrix Classes and Methods. 2013. URL http://CRAN.R-project.org/package=Matrix. R package version 1.0-12. [p181]

Carey, VJ.; Lumley, T.; Ripley, B. gee: Generalized Estimation Equation solver. 2012. URL http://CRAN.R-project.org/package=gee. R package version 4.13-18. [p181]

Højsgaard S, Halekoh U, Yan J. The R package geepack for generalized estimating equations. Journal of Statistical Software. 2006; 15(2):1–11. [p181].

Liang K, Zeger S. Longitudinal data analysis using generalized linear models. Biometrika. 1986; 73(1):13–22. [p181, 182, 184].

McCullagh, P.; Nelder, J. Generalized Linear Models. second edition. Chapman and Hall; 1986. [p181]

McDaniel, L.; Henderson, N. geeM: Fit Generalized Estimating Equations. 2013. URL http://CRAN.R-project.org/package=geeM. R package version 0.06. [p181]

Prentice R. A generalization of the probit and logit methods for dose response curves. Biometrics. 1976; 32(4):761–768. [p185]. [PubMed: 1009225]

SAS Institute Inc.. SAS/STAT Software, Version 9.22. Cary, NC: 2010. URL http://support.sas.com/documentation/cdl/en/statug/63347/HTML/default/viewer.htm#statug_genmod_sect[zero.noslash]49.htm. [p184]

Schildcrout J, Rathouz P. Longitudinal studies of binary response data following case-control and stratified case-control sampling: design and analysis. Biometrics. 2010; 66(2):365–373. [p181]. [PubMed: 19673861]

Sutradhar B, Kumar P. The inversion of correlation matrix for MA(1) process. Applied Mathematics Letters. 2003; 16(3):317–321. [p183].

Yan J, Fine JP. Estimating equations for association structures. Statistics in Medicine. 2004; 23(6):859–880. [p181]. [PubMed: 15027075]

Table 1

Speed comparisons for ohio data set.

	Independence		AR-1		Exchangeable		Unstructured	
	Avg. (s)	**Rel.**	**Avg. (s)**	**Rel.**	**Avg. (s)**	**Rel.**	**Avg. (s)**	**Rel.**
geeM	0.11	1.83	0.11	1.00	0.11	1.33	0.16	1.11
gee	0.07	1.17	0.12	1.07	0.08	1.00	0.15	1.00
gee pack	0.06	1.00	0.18	1.53	0.12	1.48	0.20	1.34

Table 2

Speed comparisons for the simulated count data set.

	Independence		AR-1		Exchangeable		Unstructured	
	Avg. (s)	**Rel.**	**Avg. (s)**	**Rel.**	**Avg. (s)**	**Rel.**	**Avg. (s)**	**Rel.**
geeM	1.36	1.00	1.56	1.00	1.58	1.00	2.30	1.00
gee	2.16	1.59	4.06	2.60	2.45	1.55	3.26	1.42
geepack	1.55	1.14	3.27	2.10	2.95	1.86	5.64	2.46

Table 3

Speed comparisons for the birth outcomes data.

	Independence		AR-1		Exchangeable		Unstructured	
	Avg. (s)	**Rel.**	**Avg. (s)**	**Rel.**	**Avg. (s)**	**Rel.**	**Avg. (s)**	**Rel.**
geeM	5.28	1.00	9.87	1.00	7.33	1.00	8.18	1.00
gee	10.02	1.90	29.39	2.98	20.09	2.74	21.49	2.63
geepack	9.66	1.83	23.67	2.40	22.59	3.08	25.67	3.14