Facilitating the Calculation of the Efficient Score Using Symbolic Computing

Alexander Sibley* Zhiguo Li*† Yu Jiang† Cliburn Chan† Andrew Allen†
Kouros Owzar*†

Abstract

The score function (Rao, 1948) continues to serve a fundamental role in statistical inference. In the context of analyzing data from high-throughput genomic assays, inference on the basis of the score, as opposed to the asymptotically equivalent Wald or likelihood ratio tests, usually enjoys greater stability, considerably higher computational efficiency, and lends itself more readily to the use of resampling methods. While the score function often depends on a set of unknown nuisance parameters, which have to be replaced by estimators, the efficient score accounts for the variability induced by estimating these parameters. We illustrate using symbolic computing with computer algebra systems to facilitate the derivation of the efficient score. We demonstrate this process within the context of a standard example, and observe that this approach removes the burden of calculation and is less prone to error than manual derivations. In addition, the resulting symbolic expressions can be readily ported to compiled languages for the purpose of developing fast numerical algorithms for high throughput genomic analysis. We conclude by considering extensions of this approach.

Key Words: efficient score, symbolic computing, computer algebra

1. Introduction

The three primary methods of inference for data from high-throughput genomic assays are the Wald, likelihood ratio, and score tests. Conducting inference on the basis of the score test is asymptotically equivalent to the other two tests, and offers certain advantages. The score statistic is more stable, and while the likelihood ratio test requires an estimation of variability, the score test can be employed where the variability is difficult to estimate. Inference on the basis of the score also offers higher computational efficiency, as nuisance parameters are optimized only once, and the parameters of interest never have to be optimized. The score test also lends itself to the implementation of resampling methods. Though all three methods suffer from the variability induced by the estimation of the nuisance parameters, this can be overcome for the score test by the use of the efficient score. The cost of accounting for this variability is some additional algebraic derivation required prior to implementation. Computer algebra systems can relieve us of this burden, while offering benefits of their own.

1.1 The Efficient Score

We begin by establishing notation. We let $\vec{\theta}$ be the vector of our parameters, which we parse into $\vec{\beta}$, our parameters of interest, and $\vec{\eta}$, the nuisance parameters. We denote $\mathcal{L}(\vec{\theta}|\vec{z})$ to be our log-likelihood function of $\vec{\theta}$, given our observed variables, $\vec{z}$. We define $U_x(\vec{\theta}|\vec{z}) = \frac{\delta}{\delta x}\mathcal{L}(\vec{\theta}|\vec{z})$, e.g. if we knew that the nuisance parameters were orthogonal to $\vec{\beta}$, we could use the naive score, $U_\beta(\vec{\theta}|\vec{z}_i) = \frac{\delta}{\delta\beta}\mathcal{L}(\vec{\theta}|\vec{z}_i)$, to conduct our analysis.

That not being the case, we calculate the efficient score (Tsiatis, 2006) as:

$$U_{\text{eff}}(\theta|\vec{Z}_i) = U_\beta(\theta|\vec{Z}_i) - C_{\beta,\eta}(\theta|\vec{Z}_i)\{V_{\eta,\eta}(\theta|\vec{Z}_i)\}^{-1}U_\eta(\theta|\vec{Z}_i) \quad (1)$$

*Duke Cancer Institute, Duke University Medical Center
†Biostatistics and Bioinformatics, Duke University School of Medicine

where

$$C_{\beta,\eta}(\theta|\vec{Z}_i) = \mathbb{E}[U_\beta(\theta|\vec{Z}_i)U_\eta^T(\theta|\vec{Z}_i)], \text{ and} \tag{2}$$

$$V_{\eta,\eta}(\theta|\vec{Z}_i) = \mathbb{E}[U_\eta(\theta|\vec{Z}_i)U_\eta^T(\theta|\vec{Z}_i)] \, . \tag{3}$$

Deriving the efficient score therefore requires taking multiple derivatives and expected values, as well as carrying out matrix multiplication and inversion, magnifying the complexity of the initial likelihood function with each step. Such complications afford many opportunities for errors if these derivations are carried out manually. Symbolic computing offers a more reliable alternative.

1.2 Symbolic Computation

Computer algebra systems allow software to do the algebraic derivation and manipulation of equations formerly confined to white boards or pencil and paper. These systems are able to carry out, to varying degrees, assorted operations used in algebra, linear algebra, and statistical derivations. The transition from analog to digital offers the same benefits to algebra as it did for arithmetic. Namely, the steps can be performed faster, repeatably, and more reliably by a computer than by a person.

Symbolic computing therefore offers a convenient means of addressing the main drawback of using the efficient score, i.e. the sometimes arduous derivation of the score function. Once the likelihood function has been transcribed to the symbolic language, the rest of the derivation is implemented programmatically. This makes symbolic computation less vulnerable to error than manual derivations. Additionally, since the derivation procedure does not change from implementation to implementation, much of the symbolic computing code generated for one application can be reused for another. Furthermore, since the equations exist digitally, various options exist for transferring the formulae to either functional analytical code, or to a form of markup language for presentation purposes. This makes symbolic processing useful for rapid prototyping, since it is easy to convert algebraic equations and formulae into code for simulation and testing. Further, whether in prototyping or development, the effects of any changes to the initial specifications can easily be carried forward through the rest of the implementation.

Multiple symbolic processors are currently available; some proprietary, others offered via general public license. For the purposes of this report, we chose to use SymPy (SymPy Development Team, 2014), an open-souce module which allows symbolic processing in Python. SymPy was chosen because the use of Python allows for direct integration with LaTeX via PythonTeX (Poore, 2013). For example, the block below shows actual Python code which is executed when the report is compiled.

```
from sympy import *
from sympy.printing import *
from sympy.stats import *
```

Here we load the SymPy module and the tools necessary for our derivation, below.

2. An Example

We use a standard example in biostatistics to demonstrate facilitating the calculation of the efficient score using symbolic computing: testing the association of genotype with a continuous phenotype in high throughput genomic assays in the context of family-based trio data (Abecasis et al., 2000). Here, Y_i represents the phenotype of interest of the ith

patient. We let G_i represent the genotype of the ith patient at the locus being interrogated, where $G \in \{0, 1, 2\}$ represents the number of mutant alleles, and $G_{M,i}$ and $G_{F,i}$ are the corresponding parental genotypes. We then separate the genotype into between- and within-family components (Fulker et al., 1999) by defining $G_{bi} = \frac{G_{M,i}+G_{F,i}}{2}$, and $G_{wi} = G_i - G_{bi}$. Under such a separation, the within-family component, G_{wi}, is robust against confounding due to population stratification. Finally, we define $\vec{X}_i$, a vector of p cofactors (possibly including an intercept) also included in the model.

We suppose that, for all patients, the conditional distribution of $\vec{Z}$ given $G_b = g_b, G_w = g_w$ and $\vec{X} = \vec{x}$ is normal, with mean $\vec{\alpha}^T\vec{x}_i + \beta_b g_{b,i} + \beta_w g_{w,i}$ and variance σ^2, i.e.

$$Y_i|G_{bi} = g_b, G_{wi} = g_w, \vec{X}_i = \vec{x} \sim N(a_0 + a_1x_1 + a_2x_2 + b_bg_b + b_wg_w,\ \sigma^2)$$

So, under the efficient score notation, $\vec{Z}_i = (Y_i, \vec{X}_i, G_{bi}, G_{wi})^T$ and $\vec{\theta} = (a_0, a_1, a_2, b_b, b_w\sigma)^T$. Note that, for the purposes of this example, we are restricting the model to two cofactors and an intercept, though a more complex model could easily be implemented.

Under this model, the null hypothesis is that there is no genetic effect on the phenotype, i.e. $b_b = b_w = 0$, so our parameters of interest are $\vec{\beta} = (b_b, b_w)^T$, but finding the likelihood of $\vec{\beta}$ requires estimating the nuisance parameters, $\vec{\eta} = (a_0, a_1, a_2, \sigma)^T$, which may induce variability, thus necessitating the use of the efficient score.

2.1 Initial Specifications

Having imported the SymPy module above, in order to use symbolic processing, we first have to declare which expressions will represent algebraic variables.

```
a0,a1,a2,b_b,b_w = symbols("a0 a1 a2 b_b b_w", real=True)
y, x1,x2,g_b,g_w = symbols("y  x1 x2 g_b g_w", real=True)
sigma = symbols("sigma", positive=True)
```

We can then define our two sets of parameters.

```
beta = [b_b, b_w]
eta  = [a0, a1, a2] + [sigma]
```

Next, we use SymPy's built-in probability distributions to initialize the conditional distribution of $Y|G_b, G_w, \vec{X}$.

```
distY = Normal("distY",
          a0 + a1*x1 + a2*x2 + b_b*g_b + b_w*g_w, sigma)
```

We can confirm that the density has been properly defined by outputting it:
`density(distY)(y)` $= \frac{\sqrt{2}}{2\sqrt{\pi}\sigma} e^{-\frac{1}{2\sigma^2}(-a_0-a_1x_1-a_2x_2-b_bg_b-b_wg_w+y)^2}$
Note that the equation above was constructed by SymPy and automatically rendered for this report by PythonTEX. No LATEX coding was required to include it here.

The basis of the efficient score is the log-likelihood function for Y_i, which we get from the above formula.

```
likelihood = simplify(density(distY)(y))
logLikelihood = simplify(log(likelihood))
```

Which gives us the contribution to the log-likelihood for one observation, $\vec{z}_i = (y_i, g_{bi}, g_{wi}, \vec{x}_i)$, while the total log-likelihood is given by the sum over all patients.

2.2 Derivatives

To get the efficient score, we will take the derivative of the log-likelihood, `logLikelihood` $= -\log(\sigma) - \frac{1}{2}\log(\pi) - \frac{1}{2}\log(2) - \frac{1}{2\sigma^2}(a_0 + a_1x_1 + a_2x_2 + b_bg_b + b_wg_w - y)^2$ with respect to each and every one of the parameters. Depending on the form of the initial likelihood function and the number of parameters, this could entail extensive amounts of algebra. Instead, our symbolic processor does the work for us. We define a function to take the derivative of our equation with respect to a list of parameters,

```
def U(params):
    return [simplify(diff(logLikelihood, var))
            for var in params]
```

and apply our function to get the partial derivatives, $U_\beta(\vec{\theta}|\vec{Z}_i)$,

```
Ubeta = U(beta)
```

`Matrix(Ubeta)` $= \begin{pmatrix} -\frac{g_b}{\sigma^2}(a_0+a_1x_1+a_2x_2+b_bg_b+b_wg_w-y) \\ -\frac{g_w}{\sigma^2}(a_0+a_1x_1+a_2x_2+b_bg_b+b_wg_w-y) \end{pmatrix}$

and $U_\eta(\vec{\theta}|\vec{Z}_i)$.

```
Ueta = U(eta)
```

`Matrix(Ueta)` $= \begin{pmatrix} \frac{1}{\sigma^2}(-a_0-a_1x_1-a_2x_2-b_bg_b-b_wg_w+y) \\ -\frac{x_1}{\sigma^2}(a_0+a_1x_1+a_2x_2+b_bg_b+b_wg_w-y) \\ -\frac{x_2}{\sigma^2}(a_0+a_1x_1+a_2x_2+b_bg_b+b_wg_w-y) \\ \frac{1}{\sigma^3}\left(-\sigma^2+(a_0+a_1x_1+a_2x_2+b_bg_b+b_wg_w-y)^2\right) \end{pmatrix}$

Next we note that the efficient score requires $\mathbb{E}[\frac{\partial\mathcal{L}}{\partial x}\frac{\partial\mathcal{L}}{\partial y}]$. Recall that, for a function f with parameters x and y, under certain regularity conditions, $\mathbb{E}[\frac{\partial f}{\partial x}\frac{\partial f}{\partial y}] = -\mathbb{E}[\frac{\partial^2 f}{\partial x \partial y}]$, so we proceed to calculate the second partial derivatives.

We define a function to take partial derivatives of a vector with respect to a list of parameters,

```
def secPar(score, pars):
    return transpose(Matrix([[simplify(diff(x, p))
                    for x in list(score)] for p in pars]))
```

and apply it to our first partial derivatives.

```
cmat = secPar(Ubeta, eta)
vmat = secPar(Ueta, eta)
```

This gives us

`cmat` $= \begin{pmatrix} -\frac{g_b}{\sigma^2} & -\frac{g_bx_1}{\sigma^2} & -\frac{g_bx_2}{\sigma^2} & \frac{2g_b}{\sigma^3}(a_0+a_1x_1+a_2x_2+b_bg_b+b_wg_w-y) \\ -\frac{g_w}{\sigma^2} & -\frac{g_wx_1}{\sigma^2} & -\frac{g_wx_2}{\sigma^2} & \frac{2g_w}{\sigma^3}(a_0+a_1x_1+a_2x_2+b_bg_b+b_wg_w-y) \end{pmatrix}$

and

`vmat` $=$

$$\begin{pmatrix} -\frac{1}{\sigma^2} & -\frac{x_1}{\sigma^2} & -\frac{x_2}{\sigma^2} & \frac{1}{\sigma^3}(2a_0+2a_1x_1+2a_2x_2+2b_bg_b+2b_wg_w-2y) \\ -\frac{x_1}{\sigma^2} & -\frac{x_1^2}{\sigma^2} & -\frac{x_1x_2}{\sigma^2} & \frac{2x_1}{\sigma^3}(a_0+a_1x_1+a_2x_2+b_bg_b+b_wg_w-y) \\ -\frac{x_2}{\sigma^2} & -\frac{x_1x_2}{\sigma^2} & -\frac{x_2^2}{\sigma^2} & \frac{2x_2}{\sigma^3}(a_0+a_1x_1+a_2x_2+b_bg_b+b_wg_w-y) \\ \frac{1}{\sigma^3}(2a_0+2a_1x_1+2a_2x_2+2b_bg_b+2b_wg_w-2y) & \frac{2x_1}{\sigma^3}(a_0+a_1x_1+a_2x_2+b_bg_b+b_wg_w-y) & \frac{2x_2}{\sigma^3}(a_0+a_1x_1+a_2x_2+b_bg_b+b_wg_w-y) & \frac{1}{\sigma^4}(\sigma^2-3(a_0+a_1x_1+a_2x_2+b_bg_b+b_wg_w-y)^2) \end{pmatrix}$$

which are at the cores of the covariance (2) and variance (3) matrix definitions.

2.3 Expected Values

In general, we would take the expectation over the joint distribution of Y_i, $\vec{X}_i$, G_{bi}, and G_{wi}. It is at this point that we need to remain aware of the capabilities and limitations of SymPy, or any symbolic processor. SymPy does not currently have the functionality to take expectations over a joint distribution. However, by virtue of our model definition, we can avoid this by taking a double-expectation instead, e.g. $\mathbb{E}[\mathbb{E}[Y|G_b, G_w, \vec{X}]]$. We also must define our own function to take the expectation over the cells of a matrix.

```
def EV(A):
  return transpose(Matrix(map(E,A)))
```

We can then use the available expectation functionality to take the conditional expectation of $Y|G_b, G_w, \vec{X}$ using the distribution we specified above.

```
Cmat = simplify(-EV(cmat.subs(y, distY))).reshape(2,4)
```

$$\texttt{Cmat} = \begin{pmatrix} \frac{g_b}{\sigma^2} & \frac{g_b x_1}{\sigma^2} & \frac{g_b x_2}{\sigma^2} & 0 \\ \frac{g_w}{\sigma^2} & \frac{g_w x_1}{\sigma^2} & \frac{g_w x_2}{\sigma^2} & 0 \end{pmatrix}$$

For the second expectation, we first declare new symbolic variables to stand in the for the expected values of the remaining random variables and products.

```
ExGb, ExGbX1, ExGbX2 = symbols(
            "ExGb ExGbX1 ExGbX2", real=True)
```

We then take the expectation of the remaining terms by careful substitution, maintaining non-separable terms. Note that it can be shown that $G_w \perp \vec{X}$ and that $\mathbb{E}[G_w] = 0$.

```
Cmat = Cmat.subs([(g_b*x1, ExGbX1), (g_b*x2, ExGbX2)])
Cmat = Cmat.subs([(g_b, ExGb), (g_w, 0)])
```

This gives us our final covariance matrix:

$$\texttt{Cmat} = \begin{pmatrix} \frac{ExGb}{\sigma^2} & \frac{ExGbX_1}{\sigma^2} & \frac{ExGbX_2}{\sigma^2} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Notice that the cells corresponding to b_w (the second row) are all zero, indicating the score for b_w is trivially efficient. Notice also that the cells corresponding to the nuisance parameter σ (the last column) are also zero, from which we conclude that this parameter is in fact orthogonal to our parameters of interest.

We repeat the same process for the variance matrix (3), first taking the conditional expectation using our function and the built-in capabilities.

```
Vmat = simplify(-EV(vmat.subs(y, distY))).reshape(4,4)
```

$$\texttt{Vmat} = \begin{pmatrix} \frac{1}{\sigma^2} & \frac{x_1}{\sigma^2} & \frac{x_2}{\sigma^2} & 0 \\ \frac{x_1}{\sigma^2} & \frac{x_1^2}{\sigma^2} & \frac{x_1 x_2}{\sigma^2} & 0 \\ \frac{x_2}{\sigma^2} & \frac{x_1 x_2}{\sigma^2} & \frac{x_2^2}{\sigma^2} & 0 \\ 0 & 0 & 0 & \frac{2}{\sigma^2} \end{pmatrix}$$

Next we make a further simplifying assumption that the X_{ij} are normalized, so that $\mathbb{E}[X_{ij}] = 0$ and $\mathbb{E}[X_{ij}^2] = 1$. We declare a new symbolic variable for the remaining product, and make our substitutions to complete the second expectation.

```
ExX1X2 = symbols("ExX1X2", real=True)
Vmat = Vmat.subs([(x1**2,1),(x2**2,1),(x1*x2,ExX1X2)])
Vmat = Vmat.subs([(x1, 0),(x2, 0)])
```

$$\texttt{Vmat} = \begin{pmatrix} \frac{1}{\sigma^2} & 0 & 0 & 0 \\ 0 & \frac{1}{\sigma^2} & \frac{ExX1X2}{\sigma^2} & 0 \\ 0 & \frac{ExX1X2}{\sigma^2} & \frac{1}{\sigma^2} & 0 \\ 0 & 0 & 0 & \frac{2}{\sigma^2} \end{pmatrix}$$

Notice the block-diagonal structure.

We now have all the components we need to compose the efficient score (1). Once more, symbolic computing makes this process simple.

```
Vinv = simplify(Vmat.inv())
Ueff = simplify(Matrix(Ubeta)-(Cmat*Vinv*Matrix(Ueta)))
```

$$\texttt{Ueff} = \begin{pmatrix} \frac{1}{\sigma^2(ExX1X2^2-1)}\left(-x_1(ExGbX_1-ExGbX_2ExX1X2)+x_2(ExGbX_1ExX1X2-ExGbX_2)+(ExGb-g_b)\left(ExX1X2^2-1\right)\right)(a_0+a_1x_1+a_2x_2+b_bg_b+b_wg_w-y) \\ -\frac{g_w}{\sigma^2}(a_0+a_1x_1+a_2x_2+b_bg_b+b_wg_w-y) \end{pmatrix}$$

In a real world application, the remaining nuisance parameters would be replaced with MLEs, the expectations replaced with empirical estimators, and, under the null hypothesis, $\vec{\beta} = 0$, yielding

```
Ustar = Ueff[0,0].subs([(b_b, 0),(b_w, 0)])
```

$$\texttt{Ustar} = \frac{1}{\sigma^2(ExX1X2^2-1)}\left(-x_1\left(ExGbX_1-ExGbX_2ExX1X2\right)+x_2\left(ExGbX_1ExX1X2-ExGbX_2\right)+\left(ExGb-g_b\right)\left(ExX1X2^2-1\right)\right)\left(a_0+a_1x_1+a_2x_2-y\right)$$

We could then sum the U^* over all observations to arrive at the efficient score for our hypothesis. To reiterate, no manual derivations were required to execute this example, and no LaTeX coding had to be done to include these equations in this report.

3. Discussion

With the help of symbolic processing, we were able to derive the efficient score for our model. The effort saved through the use of symbolic computing is significant. By manipulating our equations symbolically, our task was reduced to the application of the theory. In fact, much of the process (all of section 2.2) is agnostic to the model being used, so that, excepting the specification of the likelihood function and expected values, the code is reusable across applications. Add to this the potential time which might have been spent finding and correcting arithmetic or algebraic errors in a manual derivation, and the benefits are even more compelling. This was a simple example with a straightforward likelihood function, but benefits scale dramatically as the complexity of the likelihood function increases.

Though we were spared the encumbrance of the algebraic calculations, we still had to remain vigilant that the symbolic processing was being performed accurately. As we saw, SymPy does not offer all of the functionality we need "right out of the box". To ensure that errors are not allowed to blindly propagate, it was necessary to confirm that each manipulation of the equations gave reasonable results. However, in practice, if an error had been discovered after the fact, or a correction needed to be made, the alterations to the consequent calculations could be carried out instantly by simply re-executing the subsequent portion of the script.

For this example, we used SymPy, though other symbolic processors are available. SymPy, despite its limitations, was chosen for the readability of its code, its ease of integration with LaTeX via PythonTeX, and because it has the added benefit of being free and open

source. Another computer algebra system currently available is Mathematica (Wolfram Research, Inc., 2012). While not freely available like SymPy, it is far more highly developed; for example it has the capability to compute expectations over joint distributions. NCAlgebra (Helton et al., 1996), an independently developed package for Mathematica, adds the capability to perform non-commutative (i.e. matrix) algebra. In our example, this would have given us the option to keep the equations in vector format. However, Mathematica's lack of LaTeX integration would have made reporting more difficult. Maple (Maplesoft, a division of Waterloo Maple Inc., 2014), another proprietary software package, shares the depth and robust capabilities of Mathematica, and has the option of exporting code to LaTeX. It also has the distinct ability of being able to generate optimized code based on user-specified equations. In case a resulting score function is not as straightforward as our example, Maple can produce optimized functions for several compiled languages, including Python and C, which could then be used in simulation or analysis.

The portability of code also makes symbolic processors useful for rapid prototyping. Equations can quickly be generated by the computer algebra system, and then used in simulations to explore the operating characteristics of a statistic, or to check the implications of something like our normalizing assumption for the covariates.

4. Conclusion

Inference on the basis of the score has many advantages. When conducting inference on high-throughput genomic assays, the score has greater computational efficiency and more stability than the Wald and likelihood ratio tests, while being asymptotically equivalent. It offers a convenient implementation of resampling methods. The use of the efficient score accounts for the variability induced by the estimation of unknown nuisance parameters, at the cost of a sometimes complicated derivation. Symbolic computing simplifies and streamlines this derivation, removing the burden of calculation and reducing the potential for arithmetic error. Much of the code for the derivation is reusable and it is entirely reproducible. Depending on the choice of software, this approach can even offer automatic conversion into optimized code, to easily apply the inferential method, or direct integration into LaTeX for reporting and presentation. Symbolic computing removes the complications of an attractive but potentially complex analysis, inference on the basis of the score, and allows code to flow directly from derivation, to analysis, to reporting.

Acknowledgments

Partial support for this research was provided by a grant from the National Cancer Institute (CA142538).

References

Abecasis, G. R., Cardon, L. R., and Cookson, W. O. (2000). A general test of association for quantitative traits in nuclear families. *The American Journal of Human Genetics*, 66:279–292.

Fulker, D. W., Cherny, S. S., Sham, P. C., and Hewitt, J. K. (1999). Combined linkage and association sib-pair analysis for quantitative traits. *The American Journal of Human Genetics*, 64:259–267.

Helton, J. W., Miller, R. L., and Stankus, M. (1996). Ncalgebra: A mathematica package for doing noncommuting algebra. http://math.ucsd.edu/ ncalg.

Maplesoft, a division of Waterloo Maple Inc. (2014). *Maple v18.0*. Springer Science+Business Media, LLC, Waterloo, Ontario.

Poore, G. M. (2013). Reproducible documents with pythontex. *Proceedings of the 12th Python in Science Conference*, pages 78–84.

Rao, C. R. (1948). Tests of significance in multivariate analysis. *Biometrika*, 35:58–79.

SymPy Development Team (2014). Sympy: Python library for symbolic mathematics. http://www.sympy.org.

Tsiatis, A. A. (2006). *Semiparametric Theory and Missing Data*. Springer Science+Business Media, LLC, New York, NY.

Wolfram Research, Inc. (2012). *Mathematica v9.0*. Wolfram Research, Inc., Champaign, IL.