Twin Research and Human Genetics
Volume 20 ▌ Number 4 ▌ pp. 290–297 ▌ © The Author(s) 2017 ▌ doi:10.1017/thg.2017.28

CSOLNP: Numerical Optimization Engine for Solving Non-linearly Constrained Problems

Mahsa Zahery,[1] Hermine H. Maes,[2] and Michael C. Neale[2]
[1]*Department of Computer Science, Virginia Commonwealth University, Richmond, Virginia, USA*
[2]*Department of Psychiatry, Virginia Commonwealth University, Richmond, Virginia, USA*

We introduce the optimizer CSOLNP, which is a C++ implementation of the R package RSOLNP (Ghalanos & Theussl, 2012, Rsolnp: General non-linear optimization using augmented Lagrange multiplier method. R package version, 1) alongside some improvements. CSOLNP solves non-linearly constrained optimization problems using a Sequential Quadratic Programming (SQP) algorithm. CSOLNP, NPSOL (a very popular implementation of SQP method in FORTRAN (Gill et al., 1986, User's guide for NPSOL (version 4.0): A Fortran package for nonlinear programming (No. SOL-86-2). Stanford, CA: Stanford University Systems Optimization Laboratory), and SLSQP (another SQP implementation available as part of the NLOPT collection (Johnson, 2014, The NLopt nonlinear-optimization package. Retrieved from http://ab-initio.mit.edu/nlopt)) are three optimizers available in OpenMx package. These optimizers are compared in terms of runtimes, final objective values, and memory consumption. A Monte Carlo analysis of the performance of the optimizers was performed on ordinal and continuous models with five variables and one or two factors. While the relative difference between the objective values is less than 0.5%, CSOLNP is in general faster than NPSOL and SLSQP for ordinal analysis. As for continuous data, none of the optimizers performs consistently faster than the others. In terms of memory usage, we used Valgrind's heap profiler tool, called Massif, on one-factor threshold models. CSOLNP and NPSOL consume the same amount of memory, while SLSQP uses 71 MB more memory than the other two optimizers.

■ **Keywords:** OpenMx, non-linear programming, sequential quadratic programming, RSOLNP, NPSOL, SLSQP

In all structural equation modeling (SEM) techniques, alternative models are designed to test a hypothesis of interest. These models are fitted to data to find the best set of parameters that minimizes the difference between models and data. To minimize the misfit between the models and data, SEM needs optimization, which unfortunately is not an exact science. Optimization problems have different 'landscapes' to search, and the best search algorithm depends somewhat on the features of the landscape. OpenMx currently offers three optimizers, which differ in their abilities to find the solution. A goal of this article is to help the applied researcher select the right optimizer for the problem at hand. We focus on full information maximum likelihood (ML) of ordinal data because this is a complex problem that is subject to local minima.

CSOLNP, short for **C++**-based optimizer for **Sol**ving **N**onlinear **P**rograms, is one of the optimizers available in the OpenMx package (Boker et al., 2011; Neale et al., 2016). OpenMx is an open-source software package licensed under Apache 2.0, which is widely used for SEM and other statistical modeling. It runs inside R, and provides model specification in both path style and matrix style, as well as optimizers for handling non-linear equality and inequality constraints. NPSOL (short for Nonlinear Programming at Systems Optimization Laboratory at Stanford; Gill et al., 1986) was the only available optimizer in Mx or OpenMx since the early 1990s. Recently, SLSQP (short for Sequential Least-Squares Quadratic Programming) from NLOPT collection (Johnson, 2014) has been added to OpenMx. Here, we compare CSOLNP to NPSOL and SLSQP within the OpenMx package. Similar to NPSOL and SLSQP, CSOLNP solves non-linear problems by applying the SQP method to a linearly constrained Augmented Lagrangian objective function. While optimizers use similar algorithms,

RECEIVED 7 December 2016; ACCEPTED 13 April 2017. First published online 24 May 2017.

ADDRESS FOR CORRESPONDENCE: Mahsa Zahery, Department of Computer Science, Virginia Commonwealth University, Richmond, VA 23284, USA. E-mail: zaherym@vcu.edu

the implementations are different. Details of the CSOLNP optimizer will be explained in the methods section. The rest of this section provides a general description of the CSOLNP optimizer.

CSOLNP solves non-linear problems of the form:

$$\begin{gathered}\text{argmin } f(x)\\ \text{subject to :}\\ g(x) = 0\\ l_h \le h(x) \le u_h\\ l_x \le x \le u_x\end{gathered}$$

where

$f(x) : R^n \to R$ is the objective function (n is the number of free variables),
$g(x) : R^n \to R^{m_e}$ are the equality constraint functions (m_e is the number of equality constraints),
$h(x) : R^n \to R^{m_i}$ are the inequality constraint functions (m_i is the number of inequality constraints),
l_h,u_h are the lower and upper bounds for inequalities, and l_x,u_x are the bounds for free variables.

CSOLNP is an iterative algorithm that solves a QP sub-problem at each major iteration. QP is a special case of non-linear programming optimization where the objective function is quadratic and the constraints are linear. Each major iteration starts by solving a linearly constrained problem with an augmented Lagrangian objective function of the following form:

$$\begin{gathered}\text{argmin } f(x) - y^k g(x) + \left(\tfrac{\rho}{2}\right)\left\|g(x)\right\|^2\\ \text{subject to :}\\ J^k(x - x^k) = -g(x^k)\\ l_x \le x \le u_x\end{gathered}$$

The inequality constraints are converted to equality constraints by adding slack variables. The superscript k denotes the iteration number, and $J^k$is the Jacobian matrix:

$$J^k = \frac{\partial g}{\partial x}|_{x^k}$$

The original objective function is converted to an augmented Lagrangian function, which incorporates a penalty term (ρ), as well as a Lagrange multiplier term (y). The penalty term is used to penalize the objective function if the current point estimation is violating the constraints, while the Lagrange multiplier is used to reduce the computational cost imposed by updating the penalty term at each iteration.

The augmented Lagrangian objective function plays the role of a merit function measuring the quality of each iteration for finding a better point estimate.

The augmented Lagrangian objective function is a very common choice for a merit function (Biegler et al., 2003). This method does not have the drawback of penalty methods in terms of having to face an ill-conditioned unconstrained problem with huge gradients. For penalty methods, the penalty parameter is increased at each iteration to ensure that the unsatisfied constraints are penalized more severely, which eventually helps the optimizer stay close to a feasible region. Hence, optimization is achieved when the penalty parameter is increased to infinity while the term $\|g(x)\|^2$is close to zero, suggesting that no constraints are violated. But increasing the penalty parameter to infinity can result in increasing the condition number of the problem to infinity as the algorithm proceeds. Condition number is the sensitivity of the output of a system with respect to small errors in the input. Hence, a large condition number implies that small changes in the input data can make drastic changes in the solution of a system. A system with a large condition number is called ill-conditioned and its solution is not reliable. The augmented Lagrangian method uses a Lagrange multiplier term that avoids ill-conditioning by stopping the penalty parameter from approaching infinity (Wright & Nocedal, 1999).

After converting the problem to an augmented Lagrangian function with linearized constraints, CSOLNP continues with a feasibility check of the current point. The region that is bounded by the constraints of the problem is called the feasible region. Any point in this region is called feasible. If the current point is feasible, CSOLNP continues with finding an optimal solution in the feasible region. Otherwise, a phase 1 Linear Programming (LP) procedure is applied to find a feasible point.

A two-phase LP technique approaches the optimal solution of a system in two phases, feasibility seeking and optimality seeking. In phase 1, an auxiliary problem is constructed by introducing artificial variables. Artificial variables do not have any physical meaning. They are only introduced to the problem to find a feasible solution.

In phase 1, one artificial variable is added for each $\ge$ and $=$ constraints. The original problem is then replaced with the sum of these artificial variables. Since artificial variables should not become part of an optimal solution to the original problem, they have to be zero at the feasible solution, and subsequently the sum of them should be equal to zero. Hence, the goal is to minimize the new objective function subject to the constraints of the original problem. If the minimum objective value is zero, then the original problem has a feasible solution. This feasible solution is used as the starting point for phase 2, where the original objective function is minimized for finding the optimal solution.

After finding a feasible point, a QP sub-problem is solved to find the search direction. The idea behind using a QP sub-problem is that it can reflect the non-linearity of the original problem in its quadratic objective function. Also, the linear constraints make the original problem easy to solve. An obvious choice of a QP sub-problem would have the following format:

$$\begin{gathered}\text{argmin } \nabla f(x^k)^T(x - x^k) + \tfrac{1}{2}(x - x^k)^T H(x - x^k)\\ \text{subject to :}\\ \nabla g(x^k)(x - x^k) + g(x^k) = 0\\ \nabla h(x^k)(x - x^k) + h(x^k) = 0\end{gathered}$$

https://doi.org/10.1017/thg.2017.28 Published online by Cambridge University Press

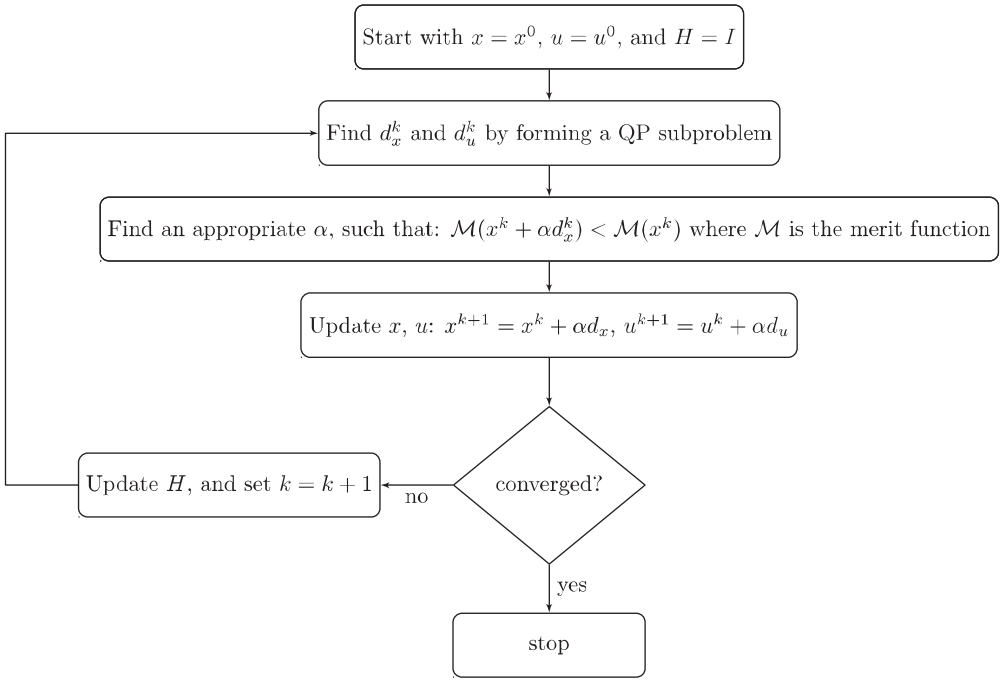

FIGURE 1

A flowchart of CSOLNP's algorithm. Starting with initial set of free variables x^0, and Lagrange multipliers u^0, the search directions d_x^k and d_u^k are found by solving a QP sub-problem. Finding an appropriate step length α, CSOLNP updates the free variables and Lagrange multipliers. If the difference between the current objective value and the previous objective value is less than the optimality tolerance, the point estimates are considered to be converged, and CSOLNP reports the final set of free variables as the optimum. Otherwise, the Hessian matrix *H* is updated, and CSOLNP continues with the next iteration.

where

∇ denotes the first derivative,
T denotes transpose, and
H is an approximation to the hessian of the Lagrangian of the objective function $\mathcal{L}(x^k, u^k)$.

Here, the objective function is obtained by the quadratic approximation of the original objective function at the current estimate x^k, and the constraints are the linear approximations of the actual constraints at the same point estimate. After the direction $d_x^k = x - x^k$ is found, CSOLNP proceeds by finding a step length (α) that satisfies all the constraints and provides sufficient decrease in the augmented Lagrangian merit function. The next iteration starts from the new point estimate $x^{k+1} = x^k + \alpha d_x^k$.

The solution of each QP sub-problem is a search direction toward a better point estimate. Eventually, after each iteration of SQP algorithm, a better approximation, x^{k+1}, is constructed. The sequence of these approximations are hoped to converge to a solution for the original constrained non-linear problem. A flowchart of CSOLNP algorithm is provided in Figure 1.

In the methods section, CSOLNP's algorithm will be detailed. What occurs at each iteration of both the SQP algorithm and the QP sub-problem will be explained. Next, the choice of step length α and the convergence criteria will be discussed. Finally, a new feature added to the optimizer for handling inequality constraints when not satisfied at the starting point is discussed.

Methods

CSOLNP Algorithm

The procedure through which CSOLNP finds the solution of a non-linear problem is as follows:

Initialization. Given the objective function and the constraints, CSOLNP starts with setting the following control parameters. All of these parameters except the penalty parameter can be user defined. The default values for these parameters are provided in parentheses:

1. maximum number of major iterations (iterations of the SQP algorithm = 400);
2. maximum number of minor iterations (iterations of the QP subproblem = 800);
3. penalty parameter ($\rho = 1$);
4. perturbation parameter δ in finite differences method for finding the numerical gradient (= 1e-7);
5. tolerance on feasibility and optimality (= 1e-9).

The objective function and the constraints are evaluated. The Lagrange multipliers are initialized to a vector of zeros with length equal to the total number of constraints; in case there are no constraints, it is set to zero.

https://doi.org/10.1017/thg.2017.28 Published online by Cambridge University Press

An augmented parameter vector containing the inequality evaluations at the starting point as well as free variables' starting values is created. The corresponding Hessian matrix is initialized with the identity matrix for the first iteration.

Iterations of the SQP algorithm (major iterations). The first major iteration of the SQP algorithm starts by scaling the objective value, the constraints and the free variables.

The gradients and the Jacobian are calculated using the forward difference method. The default value for the perturbation parameter (δ) is 1e-7.

The candidate point is checked for feasibility. If it is not feasible, a phase 1 LP procedure is performed to start the QP algorithm with a feasible point. CSOLNP implements phase 1 with a combination of Affine Scaling Method and Gradient Projection Method in the sense that the feasibility direction is found using the Affine Scaling Method, while the step to move along this direction is found using the Gradient Projection Method.

The Affine Scaling Method is a simplified variant of Karmarkar's algorithm (Karmarkar, 1984). The basic idea behind this method is to start with a point lying in the interior (inside the boundaries) of the feasible region, and move in the direction of negative gradient descent to reduce (for minimization) the objective value at the fastest possible rate. Moving toward the direction of negative gradient descent, we might fall out of the feasible region. To have more space for reducing the objective value before hitting the boundaries of the feasible region, the Affine Scaling Method changes the coordinates of the feasible point to be placed at equal distance from the boundaries. In other words, it transforms the feasible region to place the current point at its center. So, for a standard LP problem of the form:

$$\begin{gathered}\min\ c^T x \\ \text{subject to :} \\ Ax = b \\ x \geq 0\end{gathered}$$

where A is of size m by n; x and c are vectors of n elements and b is a vector of m elements, the Affine Scaling Method aims at moving in the direction of negative gradient of the objective function which is $-c$. Moving in this direction, the objective value is reduced, but we may violate $Ax = b$. To avoid this, $-c$ is projected into the null space of matrix A which is the set of all feasible direction vectors. This projection is

$$P = I - A^T(AA^T)^{-1}A$$

Hence, the projected gradient is Pc and the feasible direction would be $-Pc$. The last part of the Affine Scaling Method is to have the projected gradient near the center of the feasible region. This way, there is more room for further iterations of the algorithm. For this purpose, the current point x is rescaled to the point $X = D^{-1}x$, where D is a diagonal matrix of the elements of vector x. This changes the LP problem to minimizing $c^T DX$, subject to $ADX = b$ and $x \geq 0$. Subsequently, the projection becomes $P = I - \tilde{A}^T(\tilde{A}\tilde{A}^T)^{-1}\tilde{A}$, where $\tilde{A} = AD$. So, the projected gradient is $P\tilde{c}$ with $\tilde{c}$ being Dc. The new point is then $x^{k+1} = x^k - \alpha P\tilde{c}$, where α is the step length which is obtained by the Gradient Projection Method as the following:

$$\alpha = \begin{cases} \frac{x_i - u_i}{v_i}, & \text{if } v_i < 0 \\ \frac{x_i - l_i}{v_i}, & \text{if } v_i > 0 \end{cases}$$

where v is the search direction obtained by the Affine Scaling Method, and l_i, u_i are respectively the lower and upper bounds for x_i.

The objective value and the constraints are re-evaluated in case the starting point has been replaced with a feasible one. The merit function is also evaluated at the candidate point.

Iterations of the QP algorithm (minor iterations). The gradient of the merit function is calculated (using the forward difference method). If this is the first iteration of the QP algorithm, the algorithm considers the identity matrix as the Hessian approximation. Otherwise, a quasi-Newton approximation to the Hessian of the Lagrangian is calculated. In general, when the Hessian of the problem is dense, the quasi-Newton approximation can be a better choice, as it saves the computation time per iteration. CSOLNP uses the Broyden–Fletcher–Goldfarb–Shanno (BFGS) quasi-Newton approximation to the Hessian at each iteration of the QP algorithm:

$$H^{k+1} = H^k + \frac{1}{y^T d}yy^T - \frac{Hdd^T H}{d^T Hd}$$

Where H is the Hessian matrix, d is $x^{k+1} - x^k$, and y is the change in the gradient: $g^{k+1} - g^k$.

Finding the search direction. The QP algorithm finds the search direction using the Newton method. The search direction is obtained by Cholesky factorization of the Hessian matrix: $H^k d^k = -g^k$.

The Newton algorithm is considered to converge when all the constraints (formulae 1 and 2) and free variables' bounds (formula 3) are satisfied.

Finding steplength α. After finding the search direction, a new temporary point is approximated: $x^{k+1} = x^k + d^k$. This point is not yet considered a new estimate for the next iteration. It is necessary to figure out the length of the step to move along the direction from the current point toward this temporary point.

The step size is found using a binary search method: the interval between the current point and the temporary point is searched for a step size that results in the lowest merit function. The search continues until this interval is less than some tolerance.

https://doi.org/10.1017/thg.2017.28 Published online by Cambridge University Press

Finding the next point estimate. Having the search direction and the step size, CSOLNP finds the next point estimate: $x^{k+1} = x^k + \alpha d^k$, and a new iteration of the QP algorithm starts.

Convergence of the QP algorithm and restoring the results. The QP algorithm stops if the difference between the objective value at the current iteration of the QP algorithm and the previous iteration is less than the optimality tolerance.

The vector of free variables, the Hessian matrix, as well as the objective value and the Lagrange multipliers (in case there are constraints) are updated, and a new iteration of the SQP algorithm (major iteration) is started.

Convergence of the SQP algorithm. The problem is converged when the difference between the current objective value and the previous objective value is less than the optimality tolerance. Additionally, the constraints are satisfied to within the feasibility tolerance.

Improvements of CSOLNP Over RSOLNP

Although RSOLNP can find the solution when the inequality constraints are not satisfied at the starting point, there are cases where it fails to find the correct optimum. We have overcome this difficulty in CSOLNP by adding a new feature. For cases where the inequality constraints are not satisfied initially, CSOLNP replaces the objective function with the sum of violated inequalities, and optimizes the parameters with respect to this new objective function. The optimum will then be used as the starting point for the original problem (original objective function). This feature has enhanced the performance of CSOLNP over RSOLNP, in the sense that models failing with RSOLNP now run successfully with CSOLNP (the results are validated by the other two optimizers).

Application

We have compared the performances of CSOLNP, NPSOL, and SLSQP on factor models with ordinal and continuous variables. A brief description of factor models and how ordinal variables are measured with such models is presented in the remainder of this section.

Factor analysis is a statistical method assuming a set of unobserved, underlying factors (latent variables) are responsible for variation among a set of observed variables. The regression of an observed variable on a factor is interpreted as factor loading. Considering p observed variables, m factors, and n subjects, the factor model can be written as

$$Y_{ij} = b_i X_j + E_{ij}$$

where $i = 1, \ldots, p$ variables and $j = 1, \ldots, n$ subjects. Y_{ij}s represent observed variables for each subject. $X_j$are factor scores, which are values of each factor for each of the subjects j in the sample. b_is are factor loadings. $E_{ij}$is unique for each observed variable, and explains the variability beyond that explained by common factors. Factor loadings are estimated as

$$\Sigma_{YY} = BPB' + E$$

where $\sum_{yy}$is a $p \times p$ covariance matrix of observed variables, B is a $p \times m$ matrix of factor loadings, P is an $m \times m$ covariance matrix of the common factors, and E is a $p \times p$ matrix of specific variances.

In confirmatory factor models, a hypothesized factor model is tested to find whether the sample data supports the model. There are several estimation procedures to find the model parameters. One is ML, which evaluates the goodness of fit of the hypothesized model to the sample data by estimating the matrix of factor loadings. Model estimation is considered successful if the original covariance matrix can be reproduced from the estimated factor loadings. If it cannot be reproduced, then the hypothesized model may have not been correctly specified.

Maximum likelihood estimation (MLE) assumes multivariate normality (mvn) of residuals of a model with continuous measures. Behavioral data are often binary (yes/no responses) or ordinal (none/some/a lot), which are inherently less accurate than continuous measures. A common approach with binary/ordinal data is to assume that there is a latent, normally distributed continuous variable underlying each binary/ordinal symptom or item. For example, in substance use behavior, sensitivity to the rewarding experience of drug use may form part of the underlying propensity to use substances frequently.

Such liability is typically thought of as being due to the additive effects of a large number of factors each of small effect, which the central limit theorem predicts will generate a normal distribution of liability. Thresholds on this liability distribution delimit binary or ordinal response categories. For binary data, yes responses are observed above a threshold, while no responses are observed below that. For ordinal data, the number of thresholds is one fewer than the number of categories in the data. For example, subjects with scores below the first threshold have observed value of none. Subjects with scores in between the first and second thresholds have observed value of some, and those with scores above the second threshold have observed value of a lot.

We have compared the performances of CSOLNP, NPSOL, and SLSQP on a variety of threshold models as well as continuous data. MLE is used as fit function for threshold models to estimate factor loadings and thresholds. The likelihood function is the joint probability of the latent continuous variables underlying the set of ordinal variables and is defined as multivariate integration of the distribution over the intervals defined by the thresholds. For mvn integration, we use Genz's SADMVN routine (Genz, 1992). The precision with which SADMVN computes the mvn integration is varied between 1e-3 and 1e-7 in our simulations to compare

https://doi.org/10.1017/thg.2017.28 Published online by Cambridge University Press

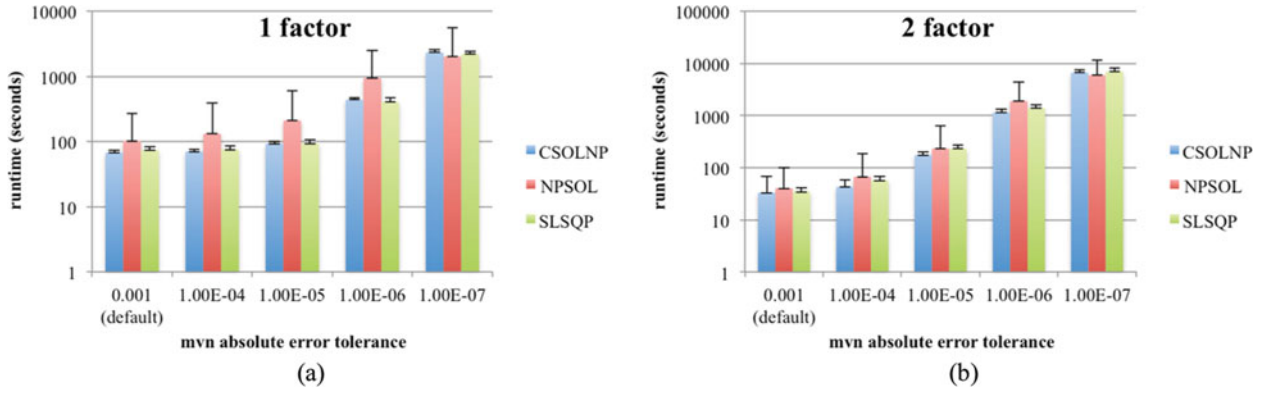

FIGURE 2

(Colour online) Runtimes of CSOLNP, NPSOL, and SLSQP in logarithmic scale for threshold models with (a) five variables and one factor, and (b) five variables and two factors for a sample of size of 1,000. Runtimes are averaged over 250 different runs.

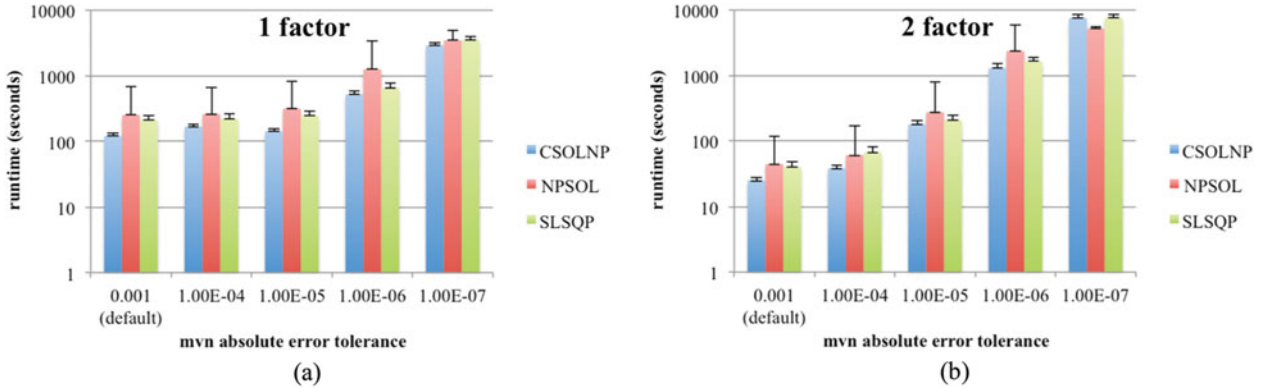

FIGURE 3

(Colour online) Runtimes of CSOLNP, NPSOL, and SLSQP in logarithmic scale for threshold models with (a) five variables and one factor, and (b) five variables and two factors for a sample of size of 10,000. Runtimes are averaged over 250 different runs.

the performances of CSOLNP, NPSOL, and SLSQP. Other varying elements in our simulations are the number of latent variables (factors) and the sample sizes. The models are run with 1 and 2 factors on datasets of size 1,000, 10,000, and 20,000 samples. The results section illustrates the performances of the three optimizers on different threshold and continuous models averaged over 250 simulations.

Results and Discussion

We compared the performances of CSOLNP, NPSOL, and SLSQP on threshold models, as well as continuous models with five variables and one to two factors.

Having five variables and one or two factors, the parameters for equation $\Sigma_{YY} = BPB' + E$ are defined as the following: matrix B is a 5×2 matrix of factor loadings. The starting values are set to 0.2 for all factor loadings. Matrix P is an identity matrix in our models. Matrix E is a 5×5 matrix of residual variances obtained by the equation $E = 1 - B^*B$. Finally, the model's expected covariance is calculated. At each iteration of the optimization algorithm, the difference between the model's expected covariance and the observed covariance is minimized to find the model that best fits the data.

Comparing Runtimes of Optimizers

For threshold models, the performances were compared with respect to runtime of the optimizers when mvn integration absolute error tolerance is reduced from 1e-3 to 1e-7. Figures 2–4 illustrate the runtimes in logarithmic scale for threshold models with one and two factors on samples of 1,000, 10,000, and 20,000 sizes, respectively. The results are averaged over 250 simulations. The error bars show the standard deviation of each optimizer's runtime obtained from 250 different runs. We also compared the final objective values at which the three optimizers stop. The relative difference between the final objective values are less than 0.5%, and hence not shown.

In all the threshold models shown in this section, the optimizers take significantly more time as the requested numerical integration precision for absolute error tolerance is reduced from 1e-3 to 1e-7. This is expected as smaller

https://doi.org/10.1017/thg.2017.28 Published online by Cambridge University Press

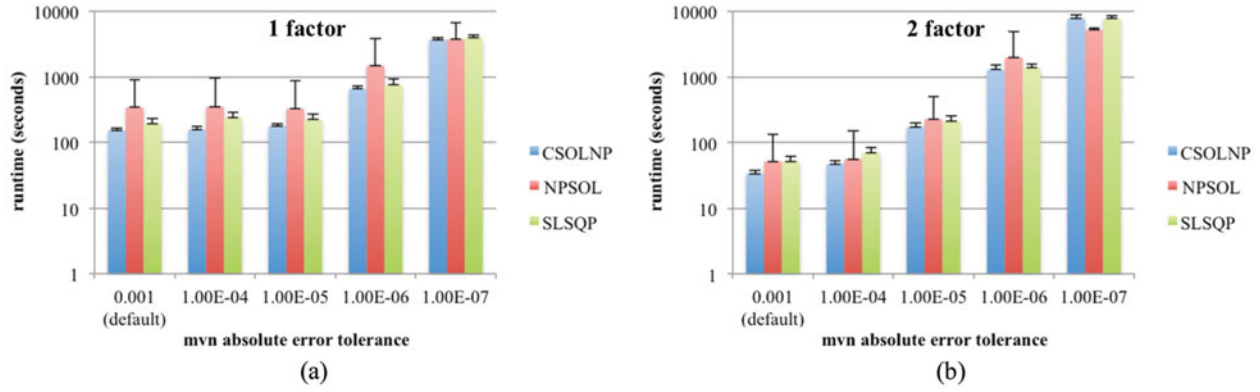

FIGURE 4

(Colour online) Runtimes of CSOLNP, NPSOL, and SLSQP in logarithmic scale for threshold models with (a) five variables and one factor, and (b) five variables and two factors for a sample of size of 20,000. Runtimes are averaged over 250 different runs.

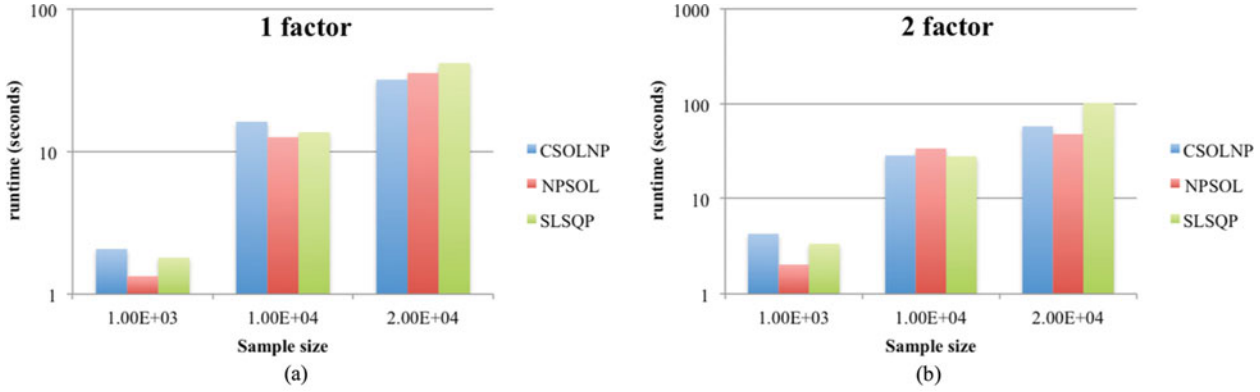

FIGURE 5

(Colour online) Runtimes of CSOLNP, NPSOL, and SLSQP in logarithmic scale for continuous models with (a) five variables and one factor, and (b) five variables and two factors for samples of sizes 1,000, 10,000, and 20,000. Runtimes are averaged over 250 different runs.

absolute error tolerance dictates the accuracy of numerical integration. Since there is no closed form solution for the integration of the mvn distribution, it is carried out numerically to a particular degree of numerical precision. However, the more precise the integral calculation, the longer it takes. NPSOL is in general the slowest optimizer, except when the absolute error tolerance is 1e-7. For other values of absolute error tolerance, CSOLNP is faster than the other optimizers.

Figure 5 illustrates runtime of the three optimizers in logarithmic scale for continuous models with one and two factors on samples of 1,000, 10,000, and 20,000 sizes, respectively. The results are averaged over 250 simulations. Since mvn integration is not required to calculate the likelihood for continuous data, the mvn parameter value is irrelevant. The final objective values are almost identical for the three optimizers and hence not shown.

The runtimes differ little between optimizers for continuous data analysis. None of the optimizers performs consistently faster than the others for such models.

Comparing Memory Usage of Optimizers

Valgrind is a memory management tool suite used for debugging and profiling. We used Massif, a heap profiler tool available in Valgrind, to compare memory consumption of the optimizers. We ran the one-factor, five-variate model with default value for mvn absolute error tolerance (1e-3) under Massif for all three optimizers. CSOLNP and NPSOL reach to peak memory of 417.8 MB, while SLSQP's peak is at 489.8 MB. SLSQP consumes about 71 MB memory more than CSOLNP and NPSOL.

CSOLNP appears to be a better choice of optimizer for threshold model analysis, due to its faster performance, reliability, and efficiency. It is the fastest optimizer except for the case where absolute error tolerance is 1e-7. Given the default value for absolute error tolerance in OpenMx package is 1e-3, CSOLNP is the fastest optimizer in all the scenarios. Furthermore, we consider CSOLNP more consistent than the other two optimizers, as the standard deviation of runtime over 250 different runs is much smaller with CSOLNP than the other two. NPSOL is very

https://doi.org/10.1017/thg.2017.28 Published online by Cambridge University Press

inconsistent and has the largest standard deviations in most of the cases. In terms of memory management, CSOLNP uses less amount of memory in comparison with SLSQP or the same amount when compared with NPSOL. All the above points make CSOLNP the preferred OpenMx optimizer for threshold model analysis.

The simple factor model is expected to have performance similar to any analysis of multivariate data, such as the classical twin study. Thus, we can recommend CSOLNP as the preferred choice for ordinal data from twins, although it should be noted that multidimensional integration becomes intractable computationally beyond a total of about 20 variables, that is, 10 variables per twin.

References

Biegler, L. T., Ghattas, O., Heinkenschloss, M., & Bloemen Waanders, B. (2003). Large-scale PDE-constrained optimization: An introduction. *Large-Scale PDE-Constrained Optimization*, *30*, 3–13.

Boker, S., Neale, M., Maes, H., Wilde, M., Spiegel, M., Brick, T., … Mehta, P. (2011). OpenMx: An open source extended structural equation modeling framework. *Psychometrika*, *76*, 306–317.

Genz, A. (1992). Numerical computation of multivariate normal probabilities. *Journal of Computational and Graphical Statistics*, *1*, 141–149.

Ghalanos, A., & Theussl, S. (2012). Rsolnp: General nonlinear optimization using augmented Lagrange multiplier method. R package version, 1. Retrieved from http://CRAN.R-project.org/package=Rsolnp

Gill, P. E., Murray, W., Saunders, M. A., & Wright, M. H. (1986). *User's guide for NPSOL (version 4.0): A Fortran package for nonlinear programming (No. SOL-86-2)*. Stanford, CA: Stanford University Systems Optimization Laboratory.

Johnson, S. G. (2014). The NLopt nonlinear-optimization package. Retrieved from http://ab-initio.mit.edu/nlopt.

Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing, pp. 302–311. New York, NY: ACM Special Interest Group on Algorithms and Computation Theory.

Neale, M. C., Hunter, M. D., Pritikin, J. N., Zahery, M., Brick, T. R., Kirkpatrick, R. M., …Boker, S. M. (2016). OpenMx 2.0: Extended structural equation and statistical modeling. *Psychometrika*, *81*, 535–549.

Wright, S., & Nocedal, J. (1999). Numerical optimization. *Springer Science*, *35*, 67–68.

https://doi.org/10.1017/thg.2017.28 Published online by Cambridge University Press