Parallel Computation of the Multivariate t-Distribution

E. de Doncker, L. Cucos, R. Zanny
Department of Computer Science
Western Michigan University
Kalamazoo, MI 49008, U.S.A.
{elise,lcucos,rrzanny}@cs.wmich.edu

A. Genz
Department of Mathematics
Washington State University
genz@math.wsu.edu

Keywords: Quasi-Monte Carlo, asynchronous, distributed integration, lattice rules, student t-distribution.

Abstract

We present a distributed algorithm for the computation of the multivariate t-distribution, based on quasi-Monte Carlo (QMC) techniques, specifically, lattice (Korobov) rules. Multiple comparison problems, as arise in statistical computations, are discussed as a sample application. Experimental results of our implementation show good speedup on a network of workstations. As the t-distribution integrals require transformation in order to apply the underlying QMC method, we propose a design for an interface with a transformation function library, which will also serve for different applications. We are merging the QMC techniques with our existing parallel adaptive integration package, PARINT.

1. INTRODUCTION

Many statistical applications make use of the multivariate t-distribution, $T(\mathbf{a}, \mathbf{b}, \Sigma, \nu) =$

$$\frac{\Gamma(\frac{\nu+s}{2})}{\Gamma(\frac{\nu}{2})\sqrt{|\Sigma|(\nu\pi)^s}} \int_{\prod_{i=1}^{s}[a_i,b_i]} (1 + \frac{1}{\nu}\mathbf{x}^\tau \Sigma^{-1}\mathbf{x})^{-\frac{\nu+s}{2}}\, d\mathbf{x}, \tag{1}$$

such as the computation of simultaneous confidence intervals (Genz and Bretz, 1999). Note that equation (1) represents the standardized $s-$variate central t-distribution function where Σ is a positive definite correlation matrix.

Our parallel algorithm for Student's t-distribution incorporates a transformation given in (Genz and Bretz, 1999). In (de Doncker et al., 2000) we gave an asynchronous algorithm for the computation of high-dimensional integrals, based on randomized lattice and Richtmyer equidistributed rules, and suited to run in a heterogeneous environment. In the present paper we will outline the algorithm with an improved load balancing strategy and present applications together with performance results.

2. NOTATIONS AND METHODS

After the Cholesky decomposition of $\Sigma = CC^\tau$, the transformation sequence of (Genz and Bretz, 1999) yields an integral of dimension $d = s - 1$, via $\mathbf{x} = C\mathbf{y}$ (eliminating the dependence on the correlation matrix from the integrand, while putting this dependence into the integration limits), a transformation which factors and separates the variables in the integrand function, and finally a transformation in each variable $u_i = t^{-1}_{\nu+i-1}(z_i)$, $1 \le i \le s$, where t_ν is the univariate t-distribution with ν degrees of freedom.

The algorithm then calculates a sequence of randomized lattice rules to approximate the integral over the $d-$dimensional unit hypercube. Let $K_N(\boldsymbol{\beta}) = \frac{1}{N}\sum_{i=1}^{N} f(\{\frac{i}{N}\mathbf{v} + \boldsymbol{\beta}\})$, where $\mathbf{v}$ is the (Korobov) lattice generator vector, randomized by a uniformly distributed random vector $\boldsymbol{\beta}$ (Cranley and Patterson, 1976). Then, using a sample set of random $\boldsymbol{\beta}_j$'s, of size q,

$$\bar{K}_N = \frac{1}{q}\sum_{j=1}^{q} K_N(\boldsymbol{\beta}_j) \tag{2}$$

allows for a standard error estimation by $E_N = \frac{1}{q(q-1)} \sum_{j=1}^{q} (K_N(\boldsymbol{\beta}_j) - \bar{K}_N)^2$.

The algorithm calculates the $\bar{K}_N$ values of (2) for successively larger numbers $N \leq N_{\max}$ of integration points until either an answer is found to the user-specified accuracy, the function count limit is reached, or the $N_{\max}$-point rule is completed. As $\bar{K}_{N_1}, \bar{K}_{N_2}, \ldots, \bar{K}_{N_{\max}}$ are obtained, the overall result $\bar{K}$ is calculated as a weighted sum where $\bar{K}_{N_i}$ is weighted with the inverse of its corresponding squared standard error E_{N_i}. The overall error estimate is denoted by $\bar{E}$.

In Section 4 we represent the randomized lattice rules of (2) as $\kappa_{ij} = K_{N_i}(\boldsymbol{\beta}_j)$.

3. MVT PROBLEMS FOR MULTIPLE COMPARISONS

An important application area where multivariate Student's t (MVT) probabilities are needed is that of multiple comparison problems which arise in statistical computations. Consider a general linear model with fixed effects: $Y = X\boldsymbol{\beta} + \boldsymbol{\epsilon}$. Y is an $N \times 1$ data vector, X is an $N \times p$ design matrix, $\boldsymbol{\beta}$ is an unknown $p \times 1$ parameter vector, and $\boldsymbol{\epsilon}$ is an $N \times 1$ error vector, with i.i.d. normally distributed components that have unknown variance σ^2. The setup for multiple comparison problems (Hochberg and Tamhane, 1987; Hsu, 1992; Hsu, 1996; Somerville, 1997; Somerville, 1998; Stoline, 1981) provides an $m \times p$ *comparison (or contrast) matrix* C. The covariance matrix for the multiple comparison problem is then $\boldsymbol{\Sigma} = C(X^t X)^{-1} C^t$, an $m \times m$ positive semi-definite matrix. The resulting basic numerical problem is the determination of confidence intervals (CI's) for

$$x_i = \sum_{j=1}^{p} c_{i,j} \beta_j, \quad i = 1, \ldots, m.$$

The distribution for the m-component $\mathbf{x}$ is MVT, with covariance matrix $\boldsymbol{\Sigma}$ and degrees of freedom $\nu = N - rank(X)$. The final input for the CI determination problem is a confidence level α. The actual numerical problem consists of finding the *critical value* t_α where $P(t_\alpha) = 1 - \alpha$, with

- $P(t) = \mathbf{T}(-\boldsymbol{\infty}, \mathbf{t}, \Sigma, \nu)$ for one-sided CI's, **or**
- $P(t) = \mathbf{T}(-\mathbf{t}, \mathbf{t}, \Sigma, \nu)$ for two-sided CI's.

Worker:

Compute initial κ_{ij}
do
    Send κ_{ij} to Controller
    Receive taskmsg $(i, j, \text{Termination})$
    Compute κ_{ij}
until Termination

Figure 1: Asynchronous algorithm: Worker.

Here $\mathbf{t} = (t, ..., t)^\tau$ and $\boldsymbol{\infty} = (\infty, ..., \infty)^\tau$. The complete numerical problem is therefore a combined problem of using an appropriate numerical optimization method to determine t_α with an efficient numerical integration method for evaluating $P(t)$, and several $P(t)$ values are usually needed before an accurate t_α can be found.

An example from (Westfall et al., 1999) uses data for housing prices. If all-pairs comparison CI's are needed, then $\nu = 57$ and $m = 10$. Another example uses data for starch thickness taken from (Hsu and Nelson, 1998). If all-pairs comparison CI's are needed, then $\nu = 86$ and the $m = 21$.

4. ASYNCHRONOUS DISTRIBUTED ALGORITHM

The algorithm uses p processes, one of which assumes a controlling role and either p or $p-1$ processes are workers (the working role of the controller is optional). The κ_{ij} values are calculated by the worker processes and applied (asynchronously) to update the overall $\bar{K}_{N_i}$ results by the controller.

Particularly on a heterogeneous network of workstations, in which the various processors may be operating at different speeds and which may have different communication latencies between processors, the asynchronous algorithm must perform dynamic load balancing to keep the processors busy. A simplified representation of the algorithm is given here in Figures 2 and 1 for the controller and worker processes, respectively. Initially, all workers independently and *statically* assign themselves an initial task for a low-level rule and report their answer to the controller via an update message. After that, the controller *dynami-*

Controller:

Termination $\leftarrow$ 0
Receive κ_{ij} from worker w_0
Update $\bar{K}_{N_i} \leftarrow \kappa_{ij}$; $\bar{K} \leftarrow \kappa_{ij}$
Select a new (i, j)
Send taskmsg $(i, j, \text{Termination})$ to worker w_0
do
    Receive κ_{ij} from worker w
    Update $\bar{K}_{N_i}$, $\bar{E}_{N_i}$, $\bar{K}$, $\bar{E}$
    if ($\bar{E} \leq \max(\varepsilon_a, \varepsilon_r|\bar{K}|)$)
        Termination $\leftarrow$ 1
    else
        Select a new (i, j)
    Send taskmsg $(i, j, \text{Termination})$ to worker w
until Termination

Figure 2: Asynchronous algorithm: Controller (not acting as worker).

cally assigns tasks to the workers via work assignment messages (*taskmsg* in Figure 2) as the updates are received. In this presentation, the controller process is not acting as an integration worker, and a single κ_{ij} corresponds to a single unit of requested work.

The controller in our implementation of (de Doncker et al., 2000) only sent work requests corresponding to a single κ_{ij}. Extending this to allow variable sized work requests allows for the controller to better balance the load among the workers. Furthermore it speeds the calculation of the κ_{ij} for small values of i. These rules consist of small numbers of points and are calculated quickly; if each work request consists of only a single unit of work, the communication latency of the message exchange between the controller and worker dominates the time spent doing calculations.

Another problem is addressed when calculating the higher lattice rules, i.e., those with many points (e.g., $\approx 600,000$ points for the $i = 25^{\text{th}}$ rule). These rules take a relatively long time to calculate. Previously the controller, acting as a worker and evaluating a rule, didn't listen for incoming messages until the rule was completed. An improvement is to allow the controller to be able to respond to workers' update messages while it is working. Similarly, the workers can listen for stop messages during their rule computation to reduce braking loss.

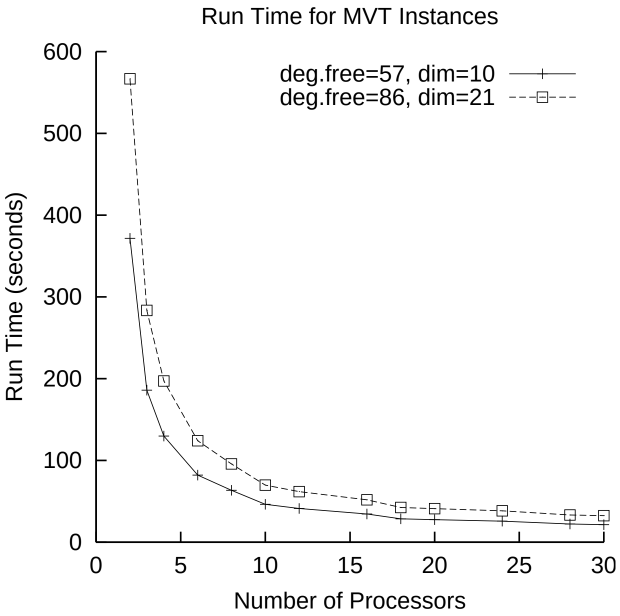

Figure 3: Timing Graph for Multiple Comparisons Problems

5. EXPERIMENTAL RESULTS

In this section we present the results of parallel runs on *Athena*, our Beowulf cluster, consisting of 32 (800 MHz) Athlon processors connected by a fast ethernet switch.

Figure 3 shows the times resulting for the integrals of the multiple comparisons problems listed in Section 3. The algorithm, as described in the previous section, shows a high degree of parallelism.

Yet it remains necessary to address and guard against possible causes of parallel performance loss, including starvation and braking loss. Figure 4 gives speedup results for the calculation of $P(t)$ of Section 3 for the starch thickness sample problem (all comparisons), and shows the effect of starvation loss (where some processors are without work), for different values of q, i.e., the maximum number of randomized lattice rule values for any rule. These graphs represent the speedup for runs that ran through the entire set of $N_{\max} = 25$ lattice rules.

The speedup is close to ideal up to q processors for each graph. Since, in these runs, the entire set of rules was used, as the last row in the κ table was calculated, only q workers are able to perform useful work.

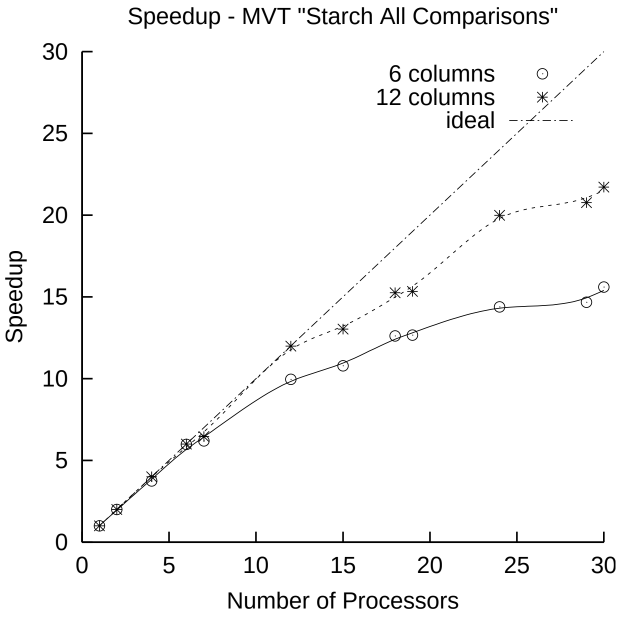

Figure 4: Speedup Graph for Student-t Problem

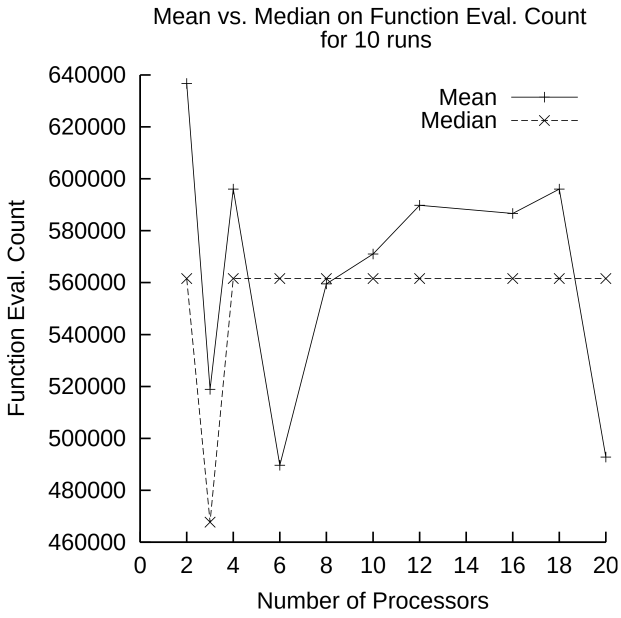

Figure 5: Mean vs. Median

Therefore, as q gets larger, more processors are able to work during the computation of the last κ row, and the speedup increases. The effect is particularly prevalent in view of the large number of function evaluations involved in this part of the computation. Note that the controller was set to handle only the updates, i.e., not to perform any rule evaluations, and the speedup is graphed relative to the number of workers.

To reduce possible braking loss (where some processors continue to work after termination is signaled), each worker periodically checks for stop messages from the controller in the course of its rule calculations.

With respect to the testing, multiple runs are needed for the same problem, in order to decrease spurious effects of system status and the randomized nature of the algorithm. The latter combined with the potential large increase in function evaluations when a new κ row is started may lead to a large variance in the running time of repeated runs with the same parameters. It is customary to average the results of repeated runs. As shown in Figure 5, showing the mean as well as the median of the number of function evaluations over a set of 10 runs of a sample problem, the general behavior may be better reflected by the median rather than the mean.

6. HANDLING OF TRANSFORMATION FUNCTIONS IN PARINT

Initial versions of the software presented in this paper were used to calculate multivariate normal probabilities (de Doncker et al., 1999). A related example yielding high-dimensional integrals is derived from computational finance.

These problems required a specific integral transformation into a form suitable for the application of QMC methods. The current application to MVT problems requires a different transformation, yet the underlying QMC method is the same.

We propose a unifying design to allow incorporating various integral transformations in a general way, so that the available set of integration methods can be applied to the transformed function. Several other uses of transformations can thus be addressed, such as for integration over product regions. For example, consider an integral over the $d-$dimensional unit simplex,

$$I_{\triangle} f = \int_0^1 \mathrm{d}x_1 \int_0^{1-x_1} \mathrm{d}x_2 \cdots \int_0^{1-\sum_{i=1}^{d-1} x_i} f(\mathbf{x}),$$

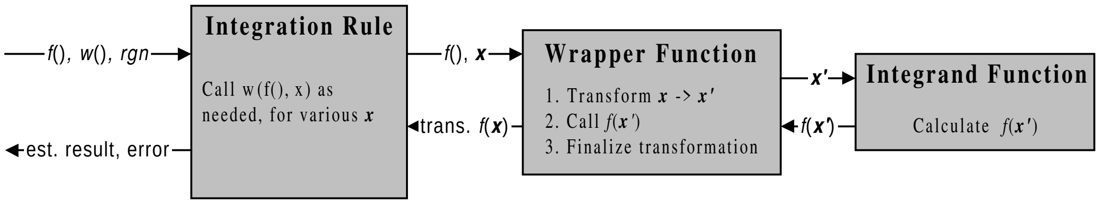

Figure 6: Diagram of Design for Transformation Functions

which can be transformed to the unit hypercube via $x_j = (1 - \sum_{i=1}^{j-1} x_i)v_j$ for $1 \le j \le d$, resulting in the integral

$$I_{\triangle} f = \int_0^1 \mathrm{d}v_1 \int_0^1 \mathrm{d}v_2 \cdots \int_0^1 \mathrm{d}v_d \prod_{j=1}^{d} (1 - \sum_{i=1}^{j-1} x_i) f(\mathbf{x}'),$$

where the transformed coordinate $\mathbf{x}' = (v_1, (1 - x_1)v_2, \ldots, (1 - \sum_{i=1}^{j-1} x_i)v_j, \ldots, (1 - \sum_{i=1}^{d-1} x_i)v_d)$.

In our code the lattice rule is implemented as a function which takes the integrand function $f()$ as one of its arguments. The integrand function is called as needed to calculate the result of the integration rule. Our design for handling transformation functions allows for a separate callback function $w()$, termed a *wrapper* function, to also be passed into the integration rule function. When the integration rule needs to calculate an $f(\mathbf{x})$, it calls $w()$, passing to it the integrand function $f()$ and the coordinate $\mathbf{x}$. The wrapper function transforms $\mathbf{x}$ into $\mathbf{x}'$ and calls $f()$ at the transformed point.

The result of the transformation is then completed through the calculations necessary for evaluating and multiplying with the Jacobian. General product regions can be treated similarly, and some transformations can be implemented for infinite regions. Note that the Jacobian may in general introduce nonregular behavior in the transformed integrand. However, as in (Robinson and de Doncker, 1981; Roose and de Doncker, 1981), they may be used for alleviating boundary singularities.

The interface design is presented in Figure 6. Note that some transformations may require initial calculations which are done once and for all (such as the initial Cholesky decomposition for MVT); for simplicity this is not reflected in Figure 6, but can be included in an initialization function, with the results passed to the wrapper function.

The generality of the design allows implementing a library of transformation functions which may be called upon as needed, not only via QMC methods, but also the from adaptive methods (based on a set of cubature rules for the unit hypercube) currently in ParInt. Accordingly, we are merging our parallel QMC code with our existing ParInt package. Users are able to select their choice technique by specifying the integration method.

7. CONCLUSIONS

Experimental results for the t-distribution application show the efficiency of our distributed algorithm. We implemented alternative versions of the strategy such as with/ without the controller as worker, and assigning multiple randomized rules per message exchange. Other modifications include an improved control for the beginning calculations and at termination.

We proposed a general interface for the incorporation of integral transformation methods into ParInt, so that transformations can be applied for different types of problems, such as the MVT, MVN, integration over various regions, or dealing with boundary singularities. Careful implementation of a transformation wrapper allows using the same back-end (consisting of the rules over the unit hypercube) and front-end (user code interface). This results in a versatile distributed integration package to integrate functions of relatively low dimension using adaptive methods, or to select QMC methods to solve problems of moderate to high dimension.

References

Cranley, R. and Patterson, T. N. L. (1976). Randomization of number theoretic methods for multiple integration. *SIAM J. Numer. Anal.*, 13:904–914.

de Doncker, E., Genz, A., and Ciobanu, M. (1999). Parallel computation of multivariate normal probabilities. *Computing Science and Statistics*, 30.

de Doncker, E., Zanny, R., Ciobanu, M., and Guan, Y. (2000). Distributed Quasi Monte-Carlo methods in a heterogeneous environment. In *Proceedings of the IPDPS Heterogeneous Computing Workshop*, pages 200–206.

Genz, A. and Bretz, F. (1999). Numerical computation of multivariate t-probabilities with application to power calculation of multiple contrasts. *J. Stat. Comp. Simul.*, 63:361–378. http://www.sci.wsu.edu/math/faculty/genz/homepage.

Hochberg, Y. and Tamhane, A. (1987). *Multiple Comparison Procedures*. John Wiley and Sons, New York.

Hsu, J. C. (1992). Simultaneous inference in the general linear model. *J. Comput. Graph. Stat.*, 1:151–168.

Hsu, J. C. (1996). *Multiple Comparisons*. Chapman and Hall, London.

Hsu, J. C. and Nelson, B. (1998). Multiple comparisons in the general linear model. *J. Comput. Graph. Stat.*, 7:23–41.

Robinson, I. and de Doncker, E. (1981). Algorithm 45 - automatic computation of improper integrals over a bounded or unbounded planar region. *Computing*, 27:253–284.

Roose, D. and de Doncker, E. (1981). Algorithm 022 - automatic integration over a sphere. *Journal of Computational and Applied Mathematics*, 7:203–224.

Somerville, P. N. (1997). Multiple testing and simultaneous confidence intervals: Calculation of constants. *Comp. Stat. & Data Analysis*, 25:217–223.

Somerville, P. N. (1998). Numerical computation of multivariate normal and multivariate-t probabilities over convex regions. *J. Comput. Graph. Stat.*, 7:529–545.

Stoline, M. (1981). The status of multiple comparisons: Simultaneous estimation of all pairwise comparisons in one-way anova design. *The American Statistician*, 35:134–141.

Westfall, P., Tobias, R., D., R., and Wolfinger, R. (1999). *Multiple Comparisons and Multiple Tests using the SAS System*. SAS Institute Inc, Cary, NC.