ABSTRACT

Continuous evaluation of dairy cattle with a random regression test-day model requires a fast solving method and algorithm. A new computing technique feasible in Jacobi and conjugate gradient based iterative methods using iteration on data is presented. In the new computing technique, the calculations in multiplication of a vector by a matrix were reordered to three steps instead of the commonly used two steps. The three-step method was implemented in a general mixed linear model program that used preconditioned conjugate gradient iteration. Performance of this program in comparison to other general solving programs was assessed via estimation of breeding values using univariate, multivariate, and random regression test-day models. Central processing unit time per iteration with the new three-step technique was, at best, one-third that needed with the old technique. Performance was best with the test-day model, which was the largest and most complex model used. The new program did well in comparison to other general software. Programs keeping the mixed model equations in random access memory required at least 20 and 435% more time to solve the univariate and multivariate animal models, respectively. Computations of the second best iteration on data took approximately three and five times longer for the animal and test-day models, respectively, than did the new program. Good performance was due to fast computing time per iteration and quick convergence to the final solutions. Use of preconditioned conjugate gradient based methods in solving large breeding value problems is supported by our findings.

(Key words: mixed linear models, preconditioned conjugate gradient, genetic evaluation)

Abbreviation key: CG = conjugate gradient, CPU = central processing unit, IMIT = in memory iteration, IOD = iteration on data, PCG = preconditioned conjugate gradient, SOJ = second-order Jacobi, RAM = random access memory, RR = random regression.

INTRODUCTION

National dairy cattle evaluations using animal model were previously unfeasible due to large random access memory (RAM) requirements. These evaluations became possible largely after the introduction of iteration on data (IOD) technique (16, 12). This technique does not require setting up the mixed model equations in the computer memory; only some vectors are needed. As computer memory has become less expensive, ever more complex models have been applied. Full advantage of the latest models, like test-day models, is realized in continuous evaluation (13). Consequently, both memory and time have become important.

Another trend in computation of genetic evaluation is the move toward general software that allows easy use of different statistical models. General programs may allow easier development of the evaluation environment (6). At least two general-purpose programs are in wide use at the moment, PEST (7) and DMU (9). A further development of the IOD solver in DMU software is the DMUIOD (11) program, which allows random regression (RR) test-day models. Despite the availability of general programs for genetic evaluation, most of the national dairy cattle evaluations use software that has been programmed for analysis of specific national data. A general program is often thought to use more memory or have slower convergence than that achieved with software tailored to a specific problem. These problems are diminishing with the introduction of ever-faster computers with increasing memory. For instance, breeding values in the Danish dairy cattle breeding evaluation are solved using the general program DMU (14).

Many national dairy cattle evaluations and PEST and DMU rely on Gauss-Seidel or Jacobi-based methods or their combinations when solving breeding value problems. For example, Finnish animal model evaluations use Gauss-Seidel on all effects, except on animal effects, which are iterated with successive overrelaxation (18). This approach of combining different iterative solvers is often due to computational limitations such as memory and computing time. In practice, convergence characteristics of successive overrelaxation and second-order Jacobi, a popular alternative, methods depend

Received January 5, 1999.
Accepted August 16, 1999.
1Corresponding author.
greatly on the relaxation factor. Some studies have indicated that conjugate gradient (CG)-based methods offer a competitive alternative (2). However, previous studies have used either the plain CG method (19) or the diagonal of the coefficient matrix as preconditioner (2, 3), or so-called Jacobi CG method. More realistic and better preconditioners have not been used. In addition, none of these studies has addressed problems in computations using IOD or efficiently applying CG-based methods for large data on dairy cattle.

Jacobi and CG-based methods have the same computationally intensive task, namely multiplication of a vector by the coefficient matrix of the mixed model equations. The purpose of this study was to introduce a computationally efficient method for calculating this coefficient matrix times a vector. Contributions that are efficiently applying CG-based methods for solving large mixed model equations can easily be extended. Computations for the mixed model equations can be written as

\[ y_i = x_i b + z_i u + e_i, \ i = 1, \ldots, N_q \]  

where \( N_q \) is the number of animals with observations, \( x_i \) and \( z_i \) are matrices having \( l_i \) rows with \( l_i \) equal to the number of traits observed on individual \( i \).

The usual assumption in Model 1 is that the residuals for different individuals are independently Gaussian distributed, and observations within individual are correlated with (co)variance structure \( R_i \), or the likelihood is \( y_i \sim N(x_i b + z_i u, R_i) \), \( i = 1, \ldots, N_q \). Prior density of breeding values \( u \) is \( N(0, G_0 \otimes A^{-1}) \) where \( G_0 \) is an \( L \times L \) additive genetic (co)variance matrix, and \( A \) is equal to the numerator relationship matrix. The mixed model equations can be written as

\[
\begin{bmatrix}
\sum_{i=1}^{N_q} x_i R_i^{-1} x_i & \sum_{i=1}^{N_q} x_i R_i^{-1} z_i \\
\sum_{i=1}^{N_q} z_i R_i^{-1} x_i & \sum_{i=1}^{N_q} z_i R_i^{-1} z_i + G_0^{-1} \otimes A^{-1}
\end{bmatrix}
\begin{bmatrix}
b \\
u
\end{bmatrix}
= 
\begin{bmatrix}
\sum_{i=1}^{N_q} x_i R_i^{-1} y_i \\
\sum_{i=1}^{N_q} z_i R_i^{-1} y_i
\end{bmatrix}
\]  

[2]

The summation presentation in the left-hand side of Equation [2] illustrates the IOD algorithm in calculating a coefficient matrix times a vector. Contributions to the product of the least squares part of coefficient matrix times the vector, because of the observations on individual \( i \), are accumulated as data in the form of \( x_i \) and \( z_i \), and read from the disk. Contributions that are due to the relationship matrix can be made in a separate step by reading individuals and their parents.

**Iterative Methods**

Denote the mixed model Equation [2] as \( Ca = r \). The second-order Jacobi (SOJ) method (12) is commonly used in solving large mixed model equations. Its blocked version may enhance convergence. A blocked SOJ method is

\[
a^{n+1} = a^n + \alpha(a^n - a^{n-1}) + B^{-1}(r - Ca^n)
\]

where \( B \) is a block diagonal matrix containing the non-zero elements from \( C \) corresponding to the blocks. For example, fixed effects with only limited number of unknowns may be given one block, and equations for different traits within animal may be blocked.
a^{(0)} \leftarrow \text{initial guess}; \quad r^{(0)} \leftarrow r - Ca^{(0)}
\quad d^{(0)} \leftarrow M^{-1}r^{(0)}; \quad f_0 \leftarrow r_0^{(0)}d^{(0)}
\quad \text{for } k = 1, 2, \ldots
\quad q^{(k)} \leftarrow Cd^{(k-1)}; \quad \alpha_k \leftarrow f_{k-1}/d^{(k-1)}q^{(k)}
\quad a^{(k)} \leftarrow a^{(k-1)} + \alpha_k d^{(k-1)}
\quad \text{if } k \text{ is divisible by } 100
\quad r_0^{(k)} \leftarrow r - Ca^{(k)}
\quad \text{else}
\quad r_0^{(k)} \leftarrow r_0^{(k-1)} - \alpha_k q^{(k)}
\quad s^{(k)} \leftarrow M^{-1}r_0^{(k)}
\quad f_k \leftarrow r_0^{(k)}s^{(k)}; \quad \beta_k \leftarrow f_k/f_{k-1}
\quad d^{(k)} \leftarrow s^{(k)} + \beta_k d^{(k-1)}
\quad \text{if not convergence continue iteration}
\end{algorithm}

Figure 1. Pseudo code for solving problem $Ca = r$ with the preconditioned conjugate gradient method.

Most of the memory requirements are due to vectors of size equal to number of unknowns when the IOD technique is used and large models are solved. Straightforward implementation of the SOJ iteration would require using three such vectors. A memory-efficient SOJ algorithm requiring two vectors of size equal to the number of unknowns has been presented (18). However, a different technique has been implemented in the IOD option of DMU package as well in the DMUIOD program. In these programs, one of the solution vectors is stored on disk and read at each iteration. The CG method has its origins in nonlinear optimization (5, 8, 17). In the PCG method, the linear system of equations is made simpler by solving an equivalent system of equations $M^{-1}Ca = M^{-1}r$ where $M$ is a symmetric, positive definite, preconditioner matrix that approximates $C$ (1, 17). Pseudo code for the PCG method is in Figure 1 (1, 17). The plain CG method is found by having the preconditioner matrix $M$ as identity matrix whereby the PCG algorithm that is given can be somewhat simplified. Note that it can be proved that the three-term presentation for CG method given by Berger et al. (2) is equal to the algorithm given in Figure 1, and the preconditioner matrix is equal to the identity matrix (8). Commonly, at least some kind of preconditioner is needed for the CG method to converge quickly. For example, the diagonal of the coefficient matrix as a preconditioner or Jacobi preconditioner (1) is easy to implement and gives a so-called Jacobi CG method (2, 8).

The presented PCG method requires storing four vectors of size equal to the number of unknowns. It is possible to implement PCG with three vectors by storing the solution vector to the disk, similar to the DMU and DMUIOD programs for the SOJ method. However, this method was not used in our implementation. The method was implemented with the preconditioner matrix $M$ equal to the block diagonal matrix $B$ in the SOJ method, as implemented in the DMU and DMUIOD packages.

Computational Remarks

The presented iterative methods required calculation of the product of the form $Cd$ where $C$ is the coefficient matrix in Equation [2], and $d$ is a vector. Vector $d$ has the current solutions in the SOJ method, but in PCG it has the search direction used to update the current solutions. Every iteration of the PCG method minimizes the distance between the current solutions and the true solutions in the search direction. Denote $w_i = [x_i, z_i]$ and $V = \begin{bmatrix} 0 & 0 \\ 0 & G_i \otimes A \end{bmatrix}$. So, we need to calculate
\[ N_q \sum_{i=1}^{N_q} w_i R_i^{-1} w_i d + V^{-1} d = \sum_{i=1}^{N_q} v_i + v_d. \]
Consider calculation of the summation term with IOD. In univariate models without regression effects, this procedure can be done by accumulating for each individual $i$, the product $v_i = T_i d$ where coefficients in matrix $T_i = w_i R_i^{-1} w_i$ can be deduced without performing any of the products, as $w_i$ contains zeros and ones only, and $R_i$ is a scalar (e.g., 16, 12). For a multivariate model, the rules for making matrix $T_i$ are essentially the same but with scalar contributions replaced by matrix $R_i$ (6). In DMUIOD the product is calculated in two steps:
\[ s_i \leftarrow R_i^{-1} w_i d; \quad v_i \leftarrow w_i s_i \]
where taking advantage of the block diagonal structure of $w_i$ will result in simple computations for the first step. We propose a three-step method for calculating the product $w_i R_i^{-1} w_i d$:
\[ s_i \leftarrow w_i d; \quad s'_i \leftarrow R_i^{-1} s_i; \quad v_i \leftarrow w_i s'_i \]
where vectors $s_i$ and $s'_i$ are of size equal to the number of traits observed on individual $i$, $l_i$. The order of calculations may have a large effect on the number of additions and multiplications performed.
Denote \( p_i \) as the number of effects over traits observed for individual \( i \). The number of multiplications and additions using the multivariate accumulation technique as in (6) is \( p_i^2 + p_i l_i \). The number of multiplications and additions performed for individual \( i \) using the DMUIOD technique [3] is \( 2p_i^2 + p_i \). In practice, the number of floating point operations is somewhat less in DMUIOD because the block-solving strategy simplifies the matrix multiplication. Using three-step method [4], the number of both operations is \( 2p_i + l_i^2 \). For example, when \( l_i = 1 (l_i = 3) \) and traits have five effects each, \( p_i = 5 \) \( (p_i = 15) \), the number of floating point operations would be 60 \( (720) \) with the first method, 85 \( (705) \) with the second, and 22 \( (78) \) with the last method. Hence, the number of computations is considerably less using the three-step approach. This operation is not all of the computations done within iteration, so actual differences are likely to be smaller.

In the Jacobi method, the first step in [4] constructs predicted observations \( \mathbf{s} \), using current solutions \( \mathbf{d} \), after which multiplication \( \mathbf{w} \mathbf{R}_i^{-1} \mathbf{s} \) is calculated. Indeed, by this method an adjusted right-hand side can be accumulated. To calculate \( \mathbf{r} - \mathbf{Cd} \), it is computationally best to accumulate product \( \mathbf{w} \mathbf{R}_i^{-1} \hat{\mathbf{e}} \), where \( \hat{\mathbf{e}} = \mathbf{y} - \mathbf{s} \). Thus, the technique can be named residual updating.

Product \( \mathbf{v}_d = \mathbf{V}^{-1} \mathbf{d} \) can be evaluated in many ways. We suggest a two-step approach:

\[
\mathbf{x} \leftarrow (\mathbf{I} \otimes \mathbf{A}^{-1}) \mathbf{d}; \quad \mathbf{v}_d \leftarrow (\mathbf{G}_0^{-1} \otimes \mathbf{I}) \mathbf{x}
\]

This procedure is numerically accurate because the first product is separate on traits, and the second product is over traits using genetic covariance structure and accumulated information. Implementation of these two steps was simple.

The blocking matrix \( \mathbf{B} \) and the preconditioner matrix \( \mathbf{M} \) in the iterative methods have a similar purpose of enhancing rate of convergence. In practice, there are many reasons why decomposition \( \mathbf{LDL}' \) is better to use than original matrix \( \mathbf{B} \) or \( \mathbf{M} \). First, products of form \( \mathbf{B}^{-1} \mathbf{r} \) and \( \mathbf{M}^{-1} \mathbf{r} \) can be evaluated without matrix inversion, making the calculations numerically more accurate. Second, number of multiplications is the same whether inverse or decomposition is used. Third, the decomposition allows using a richer family of matrices, namely approximate decompositions like incomplete Cholesky decomposition [e.g., (1, 4, 17)]. Use of an approximate decomposition as a preconditioner is quite common in PCG (1, 17).

Analysis of Data

Two sets of data were used; one having 305-d milk yield records, and the other having test-day records. The data for 305-d milk yield consisted of a sample of first, second, and third lactation records obtained from the Finnish milk-recording scheme. The test-day data consisted of first lactation test-day recordings from the Finnish milk-recording scheme. Three models were examined. Univariate and multivariate animal models on the data for 305-d milk yield and an RR model on test-day records.

In the univariate model, the first lactation 305-d milk yield was analyzed. In the multivariate model, the first three lactations of 305-d milk yield were considered as distinct traits. Both animal models had the same fixed and random effects. The fixed effects were calving age by days-open interaction, calving year by season interaction, and herd effect. The random effects were herd year of calving, animal, and residual. Variance components were derived from Pöö et al. (15). Herd-year variance was 79,834 in the single and multivariate models, and variance components for breeding value and residual were

\[
\mathbf{G}_0 = \begin{bmatrix}
265,841 & 239,257 & 225,965 \\
239,257 & 265,841 & 239,257 \\
225,965 & 239,257 & 265,841 \\
\end{bmatrix},
\]

and \( \mathbf{R}_0 = \begin{bmatrix}
431,102 & 215,551 & 172,441 \\
215,551 & 431,102 & 215,551 \\
172,441 & 215,551 & 431,102 \\
\end{bmatrix} \), respectively.

The RR model had fixed classification effects of herd, test-year by test-month interaction, calving age, and days carried calf. In addition, there were four fixed regression coefficients to describe the lactation curve, which were estimated within three calving season classes (October to February, March to June, and July to September). The random effects were test-month within herd, three regression coefficients for animal, and three regression coefficients for the nonheritable animal effect that described the covariance among different measurements within lactation. Variance for test month within herd effect was 2.015, and residual variance was 4.470. The three RR coefficients were Legendre polynomials (9) for days in milk up to a quadratic term for the genetic and nonheritable animal effects. Variance components for these effects were

\[
\mathbf{K}_r = \begin{bmatrix}
4.353 & -0.064 & -0.708 \\
-0.064 & 0.541 & 0.106 \\
-0.708 & 0.106 & 0.258 \\
\end{bmatrix},
\]

and \( \mathbf{K}_p = \begin{bmatrix}
8.080 & 0.162 & -0.454 \\
0.162 & 1.982 & 0.292 \\
-0.454 & 0.292 & 0.684 \\
\end{bmatrix} \).
for additive genetic ($K_a$) and nonhereditary animal ($K_v$) effects, each with three regression coefficients. These variance-covariance components were derived from multiple-trait REML variance components using the continuous covariance function approach described by Kirkpatrick et al (10). Number of unknowns in the mixed model equations was 450,746 for univariate, 1,294,564 for multivariate, and 7,280,153 for RR test-day models (Table 1). The 305-d data consisted of 7332 herds, with 292,454 and 7,280,153 for RR test-day models (Table 1). The 305-d data consisted of 7332 herds, with 292,454 first, 177,984 second, and 103,351 third lactation records. Pedigree information of these cows was traced as far as the current Finnish evaluation scheme permitted. This approach yielded an additive relationship matrix having 370,695 animals. The test-day data included approximately 6.73 million test-day records from 674,397 cows in 24,321 herds. The pedigree contained 1,099,730 animals and included all relations used in the Finnish dairy cattle evaluations.

**Solver Programs**

All computations were performed on a Digital Alpha Server 4000 having one gigabyte of RAM. The solver programs were PEST (version 2.8), DMU (version 4), and DMU-IOD (version 2.1). The PEST and DMU have options for building the mixed model equations in RAM. Runs using these options are called in memory iteration (IMIT) programs. The other runs used IOD and are referred as IOD programs. Only three of the programs allowed solving the RR test-day model data. The following runs were made: SOR(IMIT), successive overrelaxation using the PEST program with mixed model equations built in RAM; JCG(IMIT), Jacobi CG iteration using the DMU program with mixed model equations built in RAM; GSJ(IMIT), fixed calving age by days-open, and calving year by season interaction effects were iterated in the memory as in SOR(IMIT). The IOD was used for the other effects, Jacobi for animal effects, and Gauss-Seidel iteration for herd and herd-year effects with relaxation using the PEST program; GSSJ(IMIT), Gauss-Seidel iteration on herd effects and blocked SOJ iteration on the other effects using the DMU program; GSSJ(IMIT, RR) as GSSJ(IMIT), but this program allows RR test-day models as programmed in DMU-IOD; PCG(IMIT, RR), PCG iteration using IOD iteration. Multiplications in the IOD used the same two-step technique [3] used in the GSSJ iterations above; and PCG(IMIT, RR, U), PCG iteration with IOD using the new three-step computing technique [4].

Best relaxation factors, required in some of the iterative methods, were found by trial and error. The block diagonal and the preconditioner matrices were defined to be the same. Thus, a block Jacobi preconditioner was used (1). The calving age by days open and calving year by season interaction effects formed one block. The other block diagonal parts of the matrices were the diagonal elements for the univariate model and the 3 by 3 blocks for the multivariate model of the coefficient matrix. For the RR test-day model all fixed effects except for herd effect formed a single block. Diagonal elements were used for herd and test month within herd effect, and 3 by 3 blocks were used for additive genetic and nonhereditary animal effects.

The CG methods are guaranteed to converge to the correct solutions in finite number of iterations in absence of rounding errors if the coefficient matrix is symmetric and positive definite (8, 17). Mixed model equations can be made positive definite by making them full rank with restrictions. When no restrictions are imposed, there is no guarantee of convergence. The method may or may not converge, depending on the problem. In practice, we have observed that the PCG method converges fastest when there are no restrictions. The DMU imposed restrictions to the coefficient matrix when the IMIT option was used to make the coefficient matrix full rank. Likewise, DMU and DMU-IOD make these restrictions in the fixed effects part of the coefficient matrix to allow inversion of the fixed effects block when building the blocking matrix B.

**Comparison Criteria**

Comparison of the programs was based on central processing unit (CPU) time used to prepare data for the solver, CPU time of the solver, RAM and disk memory requirements, and number of iterations until convergence. The CPU time of the solver was from 2 runs. In the first run, 1000 iterations were made without considering convergence. In the second run, CPU time of the solver was until convergence. In the first run, work done before and after iteration, like reading initialization data or writing solutions, would be negligible, but in the second run these steps may affect results if convergence is very fast. Most of the programs had separate preparation and solver programs so that the CPU time was the total time to execute the solver program. However, SOR and JCG iterations were per-

**TABLE 1. Number of unknowns, animals, random effects, and fixed effects by model.**

<table>
<thead>
<tr>
<th>Model</th>
<th>Unknowns (no.)</th>
<th>Animals (no.)</th>
<th>Random effects per animal (no.)</th>
<th>Fixed effects for observation (no.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Univariate</td>
<td>450,746</td>
<td>370,695</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Multivariate</td>
<td>1,294,564</td>
<td>370,695</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>Test day</td>
<td>7,280,153</td>
<td>1,099,730</td>
<td>7</td>
<td>9</td>
</tr>
</tbody>
</table>
formed with a coefficient matrix in the memory, which
demanded creating this matrix in the solver program
before starting the iteration. To account for this, time
of the preparation phase was, in general, equal to the
time used by the program(s) before the start of iteration.

Convergence measure was distance to quasi-true
solutions of breeding values. Distance for trait \( j \) was calcu-
lated as

\[
C_n^j = \frac{\| \hat{u}_n^j - \hat{u}_j^j \|}{\| \hat{u}_j^j \|}
\]

where \( \hat{u}_j \) are quasi-true solutions for trait \( j \), and \( u_n^j \) is
breeding values at iteration \( n \) for trait \( j \). The distance
measure was \( \| x \| = \max_j |x_j| \). Convergence criterion at
round \( n \) was \( C_n = \max_j |C_n^j| \). Solutions were considered
converged to the true values when convergence criteria
\( C_n \) were less than \( 10^{-3} \). Hence, at least three significant
digits were correct for breeding values within all traits.

The new computing technique resulted in less com-
puting time per iteration than the old technique (Tables
2 and 3). The advantage was greatest with the RR test-
day model in which the computing time was reduced to
approximately one-third. This result can be expected
because this was the most complex model having the
most effects. In general, clear differences existed in
computing times for 1000 iterations by the different
methods (Table 2). The SOR(IMIT), JCG(IMIT), PCG
(IOD, RR), and PCG(IOD, RR, U) were the fastest, re-
quiring approximately 20 min to iterate the univariate
model. The IOD programs showed differences because
of implementation strategy; calculations of the slowest
program GSJ(IOD) took more than seven times longer
than with PCG(IOD, RR, U). In some cases, differences
between solver programs were more pronounced when
a multivariate model was solved. For example, SOR
(IMIT) and PCG(IOD, RR, U) required approximately
the same time in the computations for the univariate
model, but SOR(IMIT) needed approximately twice as
much time for the multivariate model. Some of the dif-
ferences could be attributed to reading data from disk.

Preprocessor programs used approximately the same
amount of time for the univariate model except the
preprocessor for JCG(IMIT) (Table 2). Preprocessing
was done by all programs within 10 min. The multivari-
ate model gave a larger system of equations. Conse-
quently, preprocessing times for the IOD programs
increased to greater than 40 min. The IOD programs
completed the preprocessing in 3 min. Preprocessing
time in SOR(IMIT) depended on the amount of memory
assigned in the PEST program. As more RAM was de-

defined as workspace, preprocessing time decreased with-
out effect on the general conclusion. The RR test-day
model was solved by IOD programs only. The prepro-
cessing times were less than those of IOD programs for
the multivariate model (Table 3), although the amount of processed data was much greater than for
simpler models. The results verified that the IOD pro-
grams have a fast preprocessing phase. In general, pre-
processing times of the different IOD programs were
close.

<table>
<thead>
<tr>
<th>Iterative method</th>
<th>Univariate model</th>
<th>Multivariate model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CPU(_{Pre})</td>
<td>CPU(_{1000})</td>
</tr>
<tr>
<td>SOR(IMIT)</td>
<td>2.3</td>
<td>19.9</td>
</tr>
<tr>
<td>JCG(IMIT)</td>
<td>7.1</td>
<td>18.1</td>
</tr>
<tr>
<td>GSJ(IOD)</td>
<td>1.5</td>
<td>142.9</td>
</tr>
<tr>
<td>GSJ(IOD, RR)</td>
<td>1.4</td>
<td>42.9</td>
</tr>
<tr>
<td>GSJ(IOD, RR, U)</td>
<td>2.1</td>
<td>34.2</td>
</tr>
<tr>
<td>PCG(IOD, RR)</td>
<td>1.3</td>
<td>22.4</td>
</tr>
<tr>
<td>PCG(IOD, RR, U)</td>
<td>1.3</td>
<td>19.6</td>
</tr>
</tbody>
</table>

1Some of the programs had the mixed model equations in the mem-
ory (IMIT), and others used iteration on data (IOD) algorithm when
performing the necessary computations. Random regression test-day
models (RR) were allowed by some of the programs. One of the pro-
grams used the new computing technique (U).

2SOR = Successive overrelaxation in PEST Version 2.8, JCG =
Jacobi conjugate gradient as in DMU Version 4, GSJ = Jacobi on
animal effects and Gauss-Siedel on others with relaxation as in PEST
Version 2.8, GSSJ = Gauss-Seidel on herd effect and second-order
Jacobi on others as in DMU Version 4 and DMUIOD Version 2.1,
and PCG = preconditioned conjugate gradient method.

3DMUIOD Version 2.1.
Unexpectedly, the IMIT programs did not perform very well in comparison to the IOD programs. The IOD programs could be a bit faster by reading all the data into memory. The IOD programs using PCG iteration were quickest. For the multivariate model, computing time was 68 and 89% of the best IMIT program using PCG(IOD,RR,U) and PCG(IOD,RR), respectively. These figures changed to 55 and 75% when the data were accessed from RAM.

The PCG(IOD,RR) was approximately 35, 27, and 15% faster than GSSJ(IOD,RR) in calculating 1000 iterations with univariate, multivariate, and test-day models, respectively. Good performance of GSSJ(IOD,RR) for the more complex models may be because multiplication of the coefficient matrix by the blocking matrix in $B^{-1}C$ is counted explicitly without numerically performing it; thus, the number of calculations in multivariate models in relative terms is reduced because $B^{-1}C$ has block identity matrices on the diagonal. Estimates of the number of operations using the simple formula presented as DMU technique [3] will overestimate number of floating point operations. However, some extra work is done in DMU program to achieve reduction in the number of floating point operations.

Total memory usage during the iteration was smaller in all IOD programs than in the IMIT programs as expected (Table 4). However, large differences existed between the IOD programs. The programs using Jacobi-based iteration had the smallest RAM requirements, except for GSJ(IOD), which included 10 Mbyte of pedigree data because the PEST program did not work with the option of having the pedigree on disk. Even omitting this usage, GSJ(IOD) used more RAM than the other IOD programs. Large memory requirements of PEST could be due to the single program approach, but some could be attributed to the method of making the mixed model equations as described below. All the other solvers split at least part of the preprocessor to another program. The PCG method required more RAM and less disk memory than the Jacobi-based method (Tables 3 and 4). Note that GSSJ(IOD) and GSSJ(IOD,RR) stored a vector of old solutions to the disk to save RAM. This procedure used disk storage of 7 Mbytes (31% of disk storage during iteration) for the univariate, 20 Mbytes (31%) for the multivariate, and 56 Mbytes (8%) for the test-day model. General conclusions on performance of the methods were the same when all data and the solutions vectors were in RAM. For example, PCG(IOD,RR,U) needed approximately 42% of the CPU time used by GSSJ(IOD,RR) when 1000 iterations were made, whether or not a disk was used in the univariate and multivariate models.

Differences existed in the size of the mixed model equations for the multivariate model reported by the programs. DMU correctly identified 1,294,564 unknowns of which 6 were set to zero in order to make a full rank coefficient matrix. DMUIOD and the implemented program with PCG iteration reported 1,296,243 equations of which 6 were set to zero in DMUIOD. Differences in number of unknowns are due to different ways of setting up the mixed model equations so that efficient code with small disk memory requirements can be made. The PEST reported 1,620,549 equations with the IMIT option and 1,620,567 equations with the IOD option. Differences in the number of equations contributed to some differences in RAM requirements but might have contributed to the differences in computing time per iteration as well. Only a small difference in computing time occurred between JCG(IMIT) and SOR(IMIT) when iterating the univariate model, but a much larger difference was found in solving the multivariate model. The number of equations was 80% and CPU time for 1000 iterations for JCG(IMIT) was 76% of that of SOR(IMIT). The number of nonzeros in the coefficient matrix was approximately equal.

Convergence to the quasi-true solutions was found most quickly with PCG iteration (Tables 3 and 4). Success of the PCG method was partly due to a reasonably good preconditioner. Straight use of textbook iterative methods seemed to give slow converging solvers as the following two examples illustrate. With only the diago-

### TABLE 3. Size of files in megabytes (disk), resident random access memory (RAM) requirements, central processing unit time in hours used for preprocessing (CPU_{pre}), iterating 1000 rounds (CPU_{1000}), and iterating until convergence (CPU_{conver}) with number of iterates until convergence ($N_{\text{conv}}$) for the random regression (RR) test-day model with different iterative methods to solve mixed model equations.

<table>
<thead>
<tr>
<th>Iterative method $^2$</th>
<th>Disk</th>
<th>RAM</th>
<th>CPU_{pre}</th>
<th>CPU_{1000}</th>
<th>CPU_{conver}</th>
<th>$N_{\text{conv}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSSJ(IOD,RR)</td>
<td>675</td>
<td>124</td>
<td>0.50</td>
<td>29.00</td>
<td>14.87</td>
<td>501</td>
</tr>
<tr>
<td>PCG(IOD,RR)</td>
<td>484</td>
<td>237</td>
<td>0.34</td>
<td>24.40</td>
<td>9.26</td>
<td>402</td>
</tr>
<tr>
<td>PCG(IOD,RR,U)</td>
<td>484</td>
<td>237</td>
<td>0.34</td>
<td>7.79</td>
<td>3.00</td>
<td>402</td>
</tr>
</tbody>
</table>

$^1$ The programs used iteration on data (IOD) algorithm when performing the necessary computations. One of the programs employed the new computing technique (U).

$^2$ GSSJ = Gauss-Seidel on herd effect and second-order Jacobi on others as in DMUIOD Version 2.1; PCG = preconditioned conjugate gradient method.
TABLE 4. Size of files in megabytes (Disk), resident random access memory (RAM) requirements, number of iterates until convergence ($N_{\text{Conv}}$), and central processing unit time of the solver in minutes until convergence ($\text{CPU}_{\text{Conv}}$) by model and iterative method.

<table>
<thead>
<tr>
<th>Iterative method</th>
<th>Univariate model</th>
<th></th>
<th></th>
<th></th>
<th>Multivariate model</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Disk</td>
<td>RAM</td>
<td>$N_{\text{Conv}}$</td>
<td>$\text{CPU}_{\text{Conv}}$</td>
<td>Disk</td>
<td>RAM</td>
<td>$N_{\text{Conv}}$</td>
<td>$\text{CPU}_{\text{Conv}}$</td>
</tr>
<tr>
<td>SOR(IMIT)</td>
<td>0</td>
<td>98</td>
<td>104</td>
<td>3.0</td>
<td>0</td>
<td>654</td>
<td>10753</td>
<td>992.3</td>
</tr>
<tr>
<td>JCG(IMIT)</td>
<td>0</td>
<td>79</td>
<td>101</td>
<td>2.1</td>
<td>0</td>
<td>404</td>
<td>453</td>
<td>37.3</td>
</tr>
<tr>
<td>GSJ(IOD)</td>
<td>38</td>
<td>37</td>
<td>3311</td>
<td>485.0</td>
<td>56</td>
<td>73</td>
<td>6202</td>
<td>2309.3</td>
</tr>
<tr>
<td>GSSJ(IOD)</td>
<td>28</td>
<td>14</td>
<td>159</td>
<td>6.6</td>
<td>72</td>
<td>28</td>
<td>257</td>
<td>27.8</td>
</tr>
<tr>
<td>GSSJ(IOD,RR)$^3$</td>
<td>22</td>
<td>14</td>
<td>159</td>
<td>5.3</td>
<td>63</td>
<td>28</td>
<td>257</td>
<td>25.2</td>
</tr>
<tr>
<td>PCG(IOD,RR)</td>
<td>17</td>
<td>20</td>
<td>75</td>
<td>2.0</td>
<td>42</td>
<td>47</td>
<td>159</td>
<td>11.1</td>
</tr>
<tr>
<td>PCG(IOD,RR,U)</td>
<td>17</td>
<td>20</td>
<td>75</td>
<td>1.8</td>
<td>42</td>
<td>47</td>
<td>159</td>
<td>8.6</td>
</tr>
</tbody>
</table>

$^1$Some of the programs had the mixed model equations in the memory (IMIT) and others used iteration on data algorithm when performing the necessary computations (IOD). Random regression (RR) test-day models were allowed by some of the programs. One of the programs employed the new computing technique (U).

$^2$SOR = Successive overrelaxation in PEST Version 2.8, JCG = Jacobi conjugate gradient as in DMU Version 4, GSJ = Jacobi on animal effects and Gauss-Seidel on others with relaxation as in PEST Version 2.8, GSSJ = Gauss-Seidel on herd effect and second order Jacobi on others as in DMU Version 4 and DMUIOD Version 2.1, and PCG = preconditioned conjugate gradient method.

$^3$DMUIOD Version 2.1.

In addition, all of them are general programs that allow solving many kinds of models. The PCG method implemented in the general software is a rewritten version of DMUIOD and, thus, has been designed especially to solve RR test-day problems but seems to work well with simpler models also. In practice, often even a slow program is sufficient. However, with large models time becomes a critical factor because computers cannot be used for other purposes than computing breeding values, which occurs with realistic RR test-day models. The simple RR test-day model was solved in 14.9 and 3.0 h by GSSJ(IOD,RR) and PCG(IOD,RR,U), respectively. Practical genetic evaluations use more complex models and more data requiring days of computing time. Consequently, a problem may be solved in 3 d with PCG (IOD,RR,U), whereas this procedure may take 2 wk with GSSJ(IOD,RR).

Iterative methods having relaxation factors have the agonizing phase of finding a reasonable relaxation factor. This phenomenon is especially so when dealing with large models that require days of computing because early iterates are seldom good predictors of success of the method during later stages of iteration. The preconditioned CG method does not necessarily have this initial phase and was found to converge faster. To find a suitable preconditioner may be considered a problem. However, because blocked versions of Jacobi and Gauss-Seidel based methods are in wide use, for example in DMU, DMUIOD, and in many national dairy cattle evaluations (4, 18), this simple blocking is at least a good start when looking for a preconditioner.

CONCLUSIONS

We have shown that the PCG method converges in less iterations to accurate solutions than do some of the

---

nal as a preconditioner, (JCG(IMIT)) was adequate in quickly solving the univariate problem, but in the multivariate problem, the blocked GSSJ methods were faster. Slow convergence of SOR(IMIT) in the multivariate model problem is an example of using successive overrelaxation as a black-box procedure. Iterations to achieve convergence with GSJ(IOD) was 30 times more than by SOR(IMIT) when the univariate model was solved. However, the multivariate model was solved with fewer iterations by using GSJ(IOD) than by SOR(IMIT). The most likely reason for the difference in convergence performance was that GSJ(IOD) uses blocking, unlike SOR(IMIT) (6).

Time to convergence can be deduced from the results already obtained (Tables 3 and 4). The PCG(IOD,RR) and PCG(IOD,RR,U) were clearly faster than the IMIT programs even in solving the univariate model. For the multivariate model, PCG(IOD,RR,U) used approximately 23% of the computing time of the best IMIT program JCG(IMIT). In general, the IOD programs did well because of their good convergence properties, except for GSJ(IOD). The PCG(IOD,RR,U) used one-third of the computing time of GSSJ(IOD,RR) to solve the univariate and multivariate models and was the second best IOD program [excluding PCG(IOD,RR)]. The difference was even larger with the test-day model: PCG(IOD,RR,U) was almost five times faster than GSSJ(IOD,RR).

Comparisons based on CPU time are often difficult to make, which is especially true when comparison is between different programs that were perhaps designed in different ways and had different objectives. However, most of the programs used here are used in practical genetic evaluations of at least moderate size. In addition, all of them are general programs that allow solving many kinds of models. The PCG method implemented in the general software is a rewritten version of DMUIOD and, thus, has been designed especially to solve RR test-day problems but seems to work well with simpler models also. In practice, often even a slow program is sufficient. However, with large models time becomes a critical factor because computers cannot be used for other purposes than computing breeding values, which occurs with realistic RR test-day models. The simple RR test-day model was solved in 14.9 and 3.0 h by GSSJ(IOD,RR) and PCG(IOD,RR,U), respectively. Practical genetic evaluations use more complex models and more data requiring days of computing time. Consequently, a problem may be solved in 3 d with PCG (IOD,RR,U), whereas this procedure may take 2 wk with GSSJ(IOD,RR).

Iterative methods having relaxation factors have the agonizing phase of finding a reasonable relaxation factor. This phenomenon is especially so when dealing with large models that require days of computing because early iterates are seldom good predictors of success of the method during later stages of iteration. The preconditioned CG method does not necessarily have this initial phase and was found to converge faster. To find a suitable preconditioner may be considered a problem. However, because blocked versions of Jacobi and Gauss-Seidel based methods are in wide use, for example in DMU, DMUIOD, and in many national dairy cattle evaluations (4, 18), this simple blocking is at least a good start when looking for a preconditioner.

CONCLUSIONS

We have shown that the PCG method converges in less iterations to accurate solutions than do some of the
other commonly used iterative methods in dairy cattle breeding. In addition, a computationally efficient algorithm was presented that was implemented in the PCG method for solving breeding values from large data sets using IOD. The CPU time per iteration using the new technique was lower than with the old technique; at best the computing time was reduced to one-third. Advantage depended on the model; more complicated models produced clearer differences. The program using PCG iteration was fastest among those compared. Its computing time was 85% (23%) of the best IMIT programs and 34% (34%) of the best IOD programs for the univariate (multivariate) model studied. For the RR test-day model, computing time of PCG iteration in comparison to a combination of Gauss-Seidel and SOJ iteration may take only days with PCG iteration. The quick computing time was due to fast computing time per iteration and the low number of iterations until convergence. Computationally fast implementation of the PCG method is a good choice for solving breeding values from large data sets.

ACKNOWLEDGMENTS

This work has been supported by the European Commission’s (Brussels, Belgium) Esprit programme on High-Performance Computing and Networking (Esprit project 23770).

REFERENCES